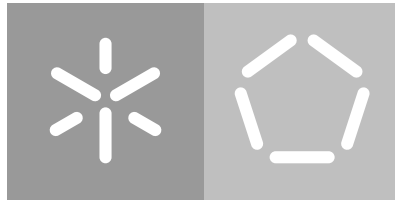


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Válter Ferreira Picas Carvalho

Hypatiamat - I Want To Solve Questions About...

November 2022



Universidade do Minho

Escola de Engenharia

Departamento de Informática

Válter Ferreira Picas Carvalho

Hypatiamat - I Want To Solve Questions About...

Master's Dissertation

Master's Degree in Informatics Engineering

Dissertation supervised by

José Carlos Leite Ramalho

Ricardo Manuel Neves Pinto

November 2022

AUTHOR COPYRIGHTS AND TERMS OF USAGE BY THIRD PARTIES

This is an academic work that can be used by third-parties as long as the internationally accepted rules and good practises are respected on what the copyright and related rights are concerned.

Therefore, the present work may be used on the terms foreseen by the licence indicated below. In case the user needs permission to use this work in any unforeseen conditions by the indicated license, he should contact the author, through RepositoriUM at University of Minho.

License provided to the users of this work



Attribution-NonCommercial

CC BY-NC

<https://creativecommons.org/licenses/by-nc/4.0/>

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ACKNOWLEDGEMENTS

First and foremost, I want to thank the supervisors, Professor José Carlos Leite Ramalho and Doctor Ricardo Manuel Neves Pinto, for their unwavering support and availability throughout the entire project. They provided me an efficient asynchronous means of communication, their years of experience and technical expertise which led this dissertation to its completion and consequently marks my presence in the academic world, albeit small, as a scientist.

These past five years have been an incredible journey and I never thought I could learn so much in such a short time frame, which is something that I never took the time to really think about until recently. To put things into perspective, on a technical level I went from writing my first *"Hello World"* to deploying several full-stack web applications and writing technical papers and, on a personal level, I am now a more efficient communicator, I learned the importance of teamwork and how much it matters in building trust with your team, I am now a more disciplined individual who thrives when given the opportunity to grow and learn and I was taught that perseverance is the key to success. It is hard to imagine what the future holds because if you told me, five years ago, that I would get this far, I would have called you a lunatic on the spot. I hope that five years from now I will be having the same thoughts and saying "I cannot believe I have gotten this far", for that is who I strive to be, someone that at any given moment is a downgrade compared to its future counterpart.

Along the way, I have met so many incredibly talented people and, as such, they have motivated me in their own way to become a better and more complete software developer and person. To the percentage of them that I can safely call friends, I am grateful for allowing me to create so many wonderful memories and experience so many joyous moments. It is so rare having such a large group of like-minded people that shared notes taken in classes, did group studies at the library together and never allowed anyone to fall behind. Nobody did this for a reward, to be noticed or to be praised, everyone just wanted to keep facing new obstacles, together.

Lastly, to my family that provided me with something I can call a home, raised me, fed me and kept me safe, **thank you**, from the bottom of my heart. That is all I can say as I cannot express with words the gratitude I feel and the warmth I receive from them daily without ever expecting anything in return. I am humbled by the fact that I would not be here had the circumstances been any different.

ABSTRACT

Hypatiamat is a Portuguese project comprised of several applications that aim to develop the Math skills of students from the 1st through 9th grades (Basic Education). The ingraining of mental calculation strategies, numbering systems, and logical operations lead to a better success rate in this subject in later years.

One of the project's components is the online platform (<https://www.hypatiamat.com>), which aims to foster autonomous learning through more interactive practices due to the current ease of technological access in this age group, by trying to appropriate teaching to everyday life. Several tools are made available, such as videos, tutorials, explanations, questions, etc. on various Math topics that students can easily access at any time.

Teachers that aim to enhance their students' learning process using this digital approach can exercise it in multiple applications provided by the platform, where the interactions are carried out and controlled through these means.

The monolithic architecture (written in PHP) has received contributions from multiple developers over the years in order to address the scalability issues introduced with this platform's growing popularity, which thus far demanded manual efforts for maintenance and content insertion. As such, there has been an incremental process of modernization, turning the various constituent applications into distinct microservices.

"I Want to Solve Questions About..." is one of these applications where students are provided with a large selection of questions in the form of mini-games (multiple choice, true or false, ...), regarding the themes mentioned above.

The first objective of the dissertation is to develop a back-office that allows the teachers in charge of the project to manage existing questions as well as add new ones for the students, since the current process requires updating the database manually.

The second one is the modernization of the application's interface at the technological level, by making use of adequate frameworks and programming languages and at the user level, by making an effort to maintain the intuitive workflow that led to its popularity but with a modernized design, in order to be consistent with other online tools.

Keywords: API, REST, Node.js, Strapi, Vue.js, Vuetify, JavaScript, Back-End, Front-End, Full-Stack, MySQL, NoSQL, Web Application, Swagger, Documentation.

RESUMO

O *Hypatiamat* é um projeto português constituído por várias aplicações que visa desenvolver as aptidões, na disciplina de Matemática, de alunos do 1º ao 9º ano de escolaridade (Educação Básica). O enraizamento de estratégias de cálculo mental, sistemas de numeração e operações lógicas originam uma melhor taxa de sucesso nesta disciplina em anos posteriores.

Uma das componentes deste projeto é a plataforma online (<https://www.hypatiamat.com>), cujo propósito é fomentar a aprendizagem autónoma através de práticas mais interativas, devido à facilidade de acesso tecnológico atual desta faixa etária, tentando apropriar o ensino ao quotidiano. São disponibilizadas várias ferramentas, tais como vídeos, tutoriais, explicações, questões, etc sobre os vários temas da Matemática (Ensino Básico) que os alunos podem facilmente aceder a qualquer momento.

Professores que pretendam enriquecer a aprendizagem dos seus alunos com esta metodologia digital podem exercê-lo nas várias aplicações que a plataforma disponibiliza, onde a interação é realizada e controlada através destes meios.

A arquitetura monolítica (escrita em PHP) tem recebido contribuições de vários desenvolvedores ao longo dos anos de modo a colmatar os problemas de escalabilidade introduzidos com a popularidade crescente desta plataforma, que até agora exigia esforço manual para manutenção e inserção de conteúdo. Assim, tem existido um processo incremental de modernização, tornando as várias aplicações constituintes em microsserviços distintos.

A "*Quero resolver questões de...*" é uma destas aplicações, onde são disponibilizadas aos alunos várias questões, sob a forma de mini-jogos (escolha múltipla, verdadeiro ou falso, ...), relativas aos temas mencionados anteriormente.

O primeiro objetivo da dissertação é o desenvolvimento de um *backoffice* que permita aos professores responsáveis gerirem as questões existentes assim como adicionarem novas para os alunos, visto que o processo atual obriga a atualização manual na base de dados.

O segundo é a modernização da interface da aplicação ao nível: tecnológico, utilizando *frameworks* e *linguagens de programação* adequadas ao problema; do utilizador, de modo a manter o fluxo intuitivo que gerou a sua popularidade mas tendo em conta um *design* mais atualizado para manter a consistência com outras ferramentas *online*.

Palavras-Chave: API, REST, Node.js, Strapi, Vue.js, Vuetify, JavaScript, *Back-End*, *Front-End*, *Full-Stack*, MySQL, NoSQL, Aplicação Web, Swagger, Documentação.

CONTENTS

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Objectives	3
1.4	Methodology	4
1.5	Document Structure	4
2	State of The Art	6
2.1	Alternative Initiatives	6
2.2	Study of the Platform	7
2.3	Study of the Technologies	8
2.3.1	<i>Client-Sided vs Server-Sided Approaches</i>	9
2.3.2	NoSQL vs SQL	10
2.3.3	MEAN vs LAMP	12
3	Question Submission Back-Office	14
3.1	Requirements Specification	14
3.1.1	Functional Requirements	15
3.1.2	Non-Functional Requirements	17
3.2	Initial Assessment	18
3.2.1	Teachers	19
3.2.2	Questions	21
3.3	Technology Selection	24
3.4	Logical Data Model	26
3.5	Back-End Development	30
3.5.1	Initial Setup	31
3.5.2	Creation of the API Endpoints	37
3.5.3	Authentication and JWTs	45
3.5.4	Protecting the API	48
3.6	Front-End Development	50
3.6.1	Initial Setup	51
3.6.2	Creation of the Interfaces	51
3.6.3	Saving the User State	78
4	I Want To Solve Questions About...	81
4.1	Requirements Specification	81

4.1.1	Functional Requirements	82
4.1.2	Non-Functional Requirements	85
4.2	Initial Assessment	86
4.2.1	Favourites	86
4.2.2	Likes	87
4.3	Technology Selection	88
4.4	Logical Data Model	89
4.5	Back-End Development	91
4.5.1	Initial Setup	91
4.5.2	Creation of the API Endpoints	94
4.5.3	Authentication	96
4.5.4	Protecting the API	97
4.6	Front-End Development	99
4.6.1	Initial Setup	99
4.6.2	Creation of the Interfaces	100
4.6.3	Saving the Blocked User State	114
5	Deployment	116
5.1	Back-End Servers	116
5.2	Front-End Servers	118
5.3	Image Server Redirects	119
6	Documentation	122
7	Conclusion	127
7.1	Outcome	127
7.2	Future Work	128
7.3	Closing Thoughts	129
A	Mock-Ups - Question Submission Back-Office	133
B	Mock-Ups - "I Want To Solve Questions About..."	139

LIST OF FIGURES

Figure 1	Homepage of the Hypatiamat website	2
Figure 2	Web page for the "I Want To Solve Questions About..." application	3
Figure 3	<i>Hypatiamat's LAMP</i> architecture	8
Figure 4	Client-Server Model	9
Figure 5	Different types of NoSQL Databases	11
Figure 6	<i>MEAN</i> Technological Stack	13
Figure 7	Interface for the insertion of a new <i>TPC</i>	17
Figure 8	Teachers and schools tables in " <i>hypati67_aplicacoes</i> "	20
Figure 9	Questions and themes tables in " <i>hypati67_testeconhecimentos</i> "	22
Figure 10	Technological stack for the back-office	25
Figure 11	Logical data model for " <i>hypati67_questoes</i> " (Simplified)	27
Figure 12	All the tables in " <i>hypati67_questoes</i> "	29
Figure 13	Strapi's Interface	30
Figure 14	Strapi School Collection	33
Figure 15	Strapi State Collection	33
Figure 16	Strapi Notification Collection	34
Figure 17	Strapi Question Backup Collection	34
Figure 18	Strapi Question Collection	35
Figure 19	Strapi Hidden Question Collection	35
Figure 20	Strapi Teacher Collection	35
Figure 21	Strapi Quarantine Collection	36
Figure 22	Strapi Deleted Submission Collection	36
Figure 23	Strapi Theme Collection	36
Figure 24	Strapi School Collection Folder	38
Figure 25	Strapi Default Roles	48
Figure 26	Strapi 'Public' role and its linked handlers (Example)	49
Figure 27	Strapi 'Authenticated' role and its linked handlers (Example)	50
Figure 28	Authentication Interface	52
Figure 29	Dashboard Interface	53
Figure 30	Form Submission Interface (1st step)	54
Figure 31	Form Submission Interface (1st step) - Text Multiple Choice	55
Figure 32	Form Submission Interface (1st step) - Image Multiple Choice	55
Figure 33	Form Submission Interface (1st step) - Open Ended, 'auxiliar' 1	56

Figure 34	Form Submission Interface (1st step) - Open Ended, 'auxiliar' 1000	56
Figure 35	Form Submission Interface (1st step) - Open Ended, 'auxiliar' 0	57
Figure 36	Form Submission Interface (1st step) - Open Ended, 'auxiliar' 22	57
Figure 37	Form Submission Interface (1st step) - Open Ended, 'auxiliar' 2	58
Figure 38	Form Submission Interface (1st step) - Open Ended, 'auxiliar' 10	58
Figure 39	Form Submission Interface (1st step) - True or False	59
Figure 40	Form Submission Interface (1st step) - Vertical Symmetry	59
Figure 41	Form Submission Interface (1st step) - Horizontal Symmetry	60
Figure 42	Form Submission Interface (2nd step) - Image Query	60
Figure 43	Form Submission Interface (2nd step) - Image Query Templates	61
Figure 44	Form Submission Interface (3rd step) - ToastUI Image Editor	62
Figure 45	Form Submission Interface (4th step) - Resolution Query	62
Figure 46	Form Submission Interface (4th step) - Resolution Query Templates	63
Figure 47	Form Submission Interface (5th step) - Toast UI Image Editor	63
Figure 48	Form Submission Interface (6th step) - Question Preview	64
Figure 49	Form Submission Interface (7th step) - Confirmation	64
Figure 50	Form Submission Interface (Edit) - 1st Step	65
Figure 51	Form Submission Interface (Edit) - 2nd and 4th Steps	65
Figure 52	Approved Questions Interface	66
Figure 53	Approved Questions (Preview) Interface - Text Multiple Choice	67
Figure 54	Approved Questions (Preview) Interface - Image Multiple Choice	67
Figure 55	Approved Questions (Preview) Interface - Open Ended, 'auxiliar' 1	68
Figure 56	Approved Questions (Preview) Interface - Open Ended, 'auxiliar' 1000	68
Figure 57	Approved Questions (Preview) Interface - Open Ended, 'auxiliar' 0	69
Figure 58	Approved Questions (Preview) Interface - Open Ended, 'auxiliar' 22	69
Figure 59	Approved Questions (Preview) Interface - Open Ended, 'auxiliar' 2	70
Figure 60	Approved Questions (Preview) Interface - Open Ended, 'auxiliar' 10	70
Figure 61	Approved Questions (Preview) - True or False	71
Figure 62	Approved Questions (Preview) - Vertical Symmetry	71
Figure 63	Approved Questions (Preview) - Horizontal Symmetry	72
Figure 64	Active Submissions List Interface	72
Figure 65	Hidden Questions List Interface	73

Figure 66	Submission History Interface	74
Figure 67	Order Themes Interface	75
Figure 68	Change Theme Images Interface	76
Figure 69	Current Monthly Questions Interface	77
Figure 70	Current Monthly Questions (Selection) Interface	77
Figure 71	Monthly Questions History Interface	78
Figure 72	Current "I Want To Solve Questions About..." landing page	82
Figure 73	Current "I Want To Solve Questions About..." question solving page	83
Figure 74	Current "I Want To Solve Questions About..." favourites page	84
Figure 75	Current "I Want To Solve Questions About..." search bar	84
Figure 76	Technological stack for "I Want To Solve Questions About..."	88
Figure 77	Logical data model for "hypati67_qrq" (Simplified)	89
Figure 78	Strapi Student Collection	92
Figure 79	Strapi Favourite Collection	92
Figure 80	Strapi Like Collection	93
Figure 81	Strapi Theme Order Collection	93
Figure 82	Strapi Theme Image Collection	93
Figure 83	Strapi Monthly Question Collection	94
Figure 84	Strapi Monthly Question Resolution Collection	94
Figure 85	Strapi 'Public' Role - Questions API	98
Figure 86	Strapi 'Authenticated' Role - Monthly Questions API	98
Figure 87	First iteration of the digital keyboard (João Vieira)	99
Figure 88	Second iteration of the digital keyboard	99
Figure 89	Authentication Interface	100
Figure 90	Theme Selector Interface	101
Figure 91	Theme Selector Interface (Tablet)	102
Figure 92	Current Question Solving Interface	103
Figure 93	Question Solving Interface	103
Figure 94	Current Question Solving Interface (Difficulty Selector)	104
Figure 95	Question Solving Interface (Difficulty Selector)	104
Figure 96	Question Solving Interface (Search)	105
Figure 97	Question Solving Interface (Favourites)	105
Figure 98	Question Solving Interface (Open-Ended) - Incorrect	106
Figure 99	Question Solving Interface (Open-Ended) - Correct	107
Figure 100	Question Solving Interface (Multiple Choice) - Text	107
Figure 101	Question Solving Interface (Multiple Choice) - Images	108
Figure 102	Question Solving Interface (True or False)	108

Figure 103	Question Solving Interface (Symmetry) - Vertical, Incorrect	109
Figure 104	Question Solving Interface (Symmetry) - Vertical, Correct	109
Figure 105	Question Solving Interface (Symmetry) - Horizontal, Incorrect	110
Figure 106	Question Solving Interface (Symmetry) - Horizontal, Correct	110
Figure 107	Question Solving Interface - Next and Previous Buttons	111
Figure 108	Favourites Interface	111
Figure 109	Monthly Questions Interface	112
Figure 110	Monthly Questions Interface	113
Figure 111	Leaderboards Interface	114
Figure 112	Timer Interface	115
Figure 113	Image Server Redirects Diagram	121
Figure 114	Documentation Folder Example	123
Figure 115	Documentation Final HTML File	124
Figure 116	Documentation for the back-office	125
Figure 117	Documentation for the "I Want To Solve Questions About" application	125
Figure 118	Question Submission Back-Office - Dashboard Mockup	133
Figure 119	Question Submission Back-Office - New Submission Form	134
Figure 120	Question Submission Back-Office - Approved Questions	134
Figure 121	Question Submission Back-Office - Active Submissions List	135
Figure 122	Question Submission Back-Office - Hidden Questions	135
Figure 123	Question Submission Back-Office - Submission History	136
Figure 124	Question Submission Back-Office - Order Themes	136
Figure 125	Question Submission Back-Office - Change Theme Images	137
Figure 126	Question Submission Back-Office - Current Monthly Questions	137
Figure 127	Question Submission Back-Office - Monthly Questions History	138
Figure 128	"I Want To Solve Questions About..." - Theme Selector	139
Figure 129	"I Want To Solve Questions About..." - Question Solving Screen	140
Figure 130	"I Want To Solve Questions About..." - Favourites	140
Figure 131	"I Want To Solve Questions About..." - Monthly Questions	141
Figure 132	"I Want To Solve Questions About..." - Monthly Questions History	141
Figure 133	"I Want To Solve Questions About..." - Leaderboards	142

ACRONYMS

A

ACID Atomicity, Consistency, Isolation, Durability.

API Application Programming Interface.

AWS Amazon Web Services.

C

CAP Consistency, Availability, Partition Tolerance.

CLI Command-Line Interface.

CMS Content Management System.

CRUD Create, Read, Update, Delete.

CSS Cascading Style Sheets.

G

GCP Google Cloud Platform.

H

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

J

JSON Javascript Object Notation.

JWT JSON Web Token.

L

LAMP Linux, Apache, MySQL, PHP.

M

MEAN MongoDB, Express.js, AngularJS, Node.js.

MERN MongoDB, Express.js, React, Node.js.

MEVN MongoDB, Express.js, Vue.js, Node.js.

N

NOSQL Not Only SQL.

NPM Node Package Manager.

P

PHP Hypertext Preprocessor.

R

REST Representational State Transfer.

S

SPA Single Page Application.

SQL Structured Query Language.

T

TPC Homework (*Trabalho Para Casa*).

U

URL Uniform Resource Locator.

X

XML Extensible Markup Language.

INTRODUCTION

Before diving into the development phase of the project, it is necessary to point out some necessary concepts about *Hypatiamat* itself.

The *Hypatiamat* project was created as a response to the growing concern of the educational community over academic underperformance, especially in Math, of students from the 1st through 9th grades (Basic Education)([Hypatiamat](#)).

The main goal of this chapter is to show why this is a relevant project for the problems it attempts to solve and how it interlaces with *Hypatiamat*'s objectives in the Portuguese community.

1.1 CONTEXT

Math in schools is typically taught incrementally, i.e, it is assumed that the student, if they achieved a passing grade, has the capacity to learn new topics that require others that are covered in previous years. This implies that if they do not have a solid grasp on these concepts, they will struggle later on due to the various Math terms and the extensive range of knowledge about strategies, such as mental calculus, numbering systems, logical operations, among others. These terms and strategies are directly associated with the development of logical reasoning, therefore it is fundamental for students to achieve overall academic success on their journey and, consequently, succeed in learning Math effectively.

Nowadays, with the large-scale adoption of Internet services, teachers and schools have ways of fitting new interactive teaching methods into the daily lives of students, who usually find them very easy to use. These strategies that diverge from the traditional classroom or even gamifying the multiple branches of Mathematics lead to greater student engagement and foster the pursuit of knowledge ([Pinto, 2014](#)).

One of this project's facets is the online platform (<https://www.hypatiamat.com>) and it has a large national adherence, where it counts with the contribution of teachers and students from various municipalities ([Redação, 2019](#)) and other educational communities ([Martins, 2014](#)), both monitored by the *Hypatiamat* team using the multiple back-offices available.



Figure 1: Homepage of the Hypatiamat website

It provides a database with over 3,000 questions that grows monthly, which can be true or false, open ended responses, multiple choice, and others; assessment sheets; educational material in the form of videos, explanations and tutorials, and other resources intended for classroom usage.

1.2 MOTIVATION

Initially, the team responsible for developing the digital platform did not expect this boom in popularity, so they built it using a monolithic stack of technologies that are currently in disuse due to their scalability problems (Karanjit, 2016), which will be explored in the next chapter, as well as not providing the necessary functionalities for the project's current use cases.

The platform has been undergoing a modernization process in terms of its global architecture with the collaboration of several developers, who have been making the continuous effort of deploying the various constituent applications using microservices with the appropriate technologies and functionalities for each case.

"I Want To Solve Questions About..." has not yet been migrated using these techniques. It is responsible for presenting a set of questions in the form of mini-games - multiple choice, true or false, among others - which the students or teachers can answer. They are stored in a database (currently it has more than 3000 distinct entries, as mentioned above), so it is up to the team in charge to edit, insert, add and remove questions **manually**, since there is no *backoffice* for this purpose. Naturally, there will always be problems that arise from directly

manipulating persisted data such as errors, inconsistencies, unintended changes or the loss of information, in the worst case scenario.



Figure 2: Web page for the “I Want To Solve Questions About...” application

The interface itself is outdated considering that it was developed more than 15 years ago. The way in which the software was written makes it too rigid and difficult to change, so it naturally is comprised of patchworks that accumulate and quickly become unsustainable. There are also consistency issues due to the huge disparity between it and the current revamped services.

1.3 OBJECTIVES

The main objective is to update “*I Want To Solve Questions About...*” in order to fit the current standards of the *Hypatiamat* project, which implies undergoing the mutation process that the other applications and microservices went through.

Understanding the constituent problems of the application depicted previously and their resolution form the two main steps required in order to accomplish this goal.

Firstly, a back-office that allows *CRUD* operations to be performed on data such as the questions needs to exist. It would minimize human error rates due to the automatic checks done before permanent changes are made to the database; it would remove man-hours dedicated exclusively to dealing with technical computer jargon by people with expertise in other areas and it would speed up and facilitate the process of maintaining the application in the future. Its interface should be similar to the other services already in operation, so that the consistency is kept between all of them.

Regarding "*I Want To Solve Questions About...*" itself, it needs to be modernized using more appropriate technologies with greater longevity and ease of maintenance, to ensure that the current situation is not repeated in the future. In addition to the technological improvement, it also needs a visual update (design) so that it is consistent with the other applications of *Hypatiamat* and other online tools. However, it is crucial to maintain the intuitive workflow that has made it popular in the first place, as the primary audience is mainly children, with possibly limited technical skills. Thus, a responsive, efficient, intuitive and visually appealing interface is desired, above all.

1.4 METHODOLOGY

The work methodology regarding the creation of both applications ("*I Want To Solve Questions About...*" and the question submission back-office) will be comprised of:

- Study of the monolithic web application *Hypatiamat*;
- Study of the recently developed microservices and applications, based on their relevance to the topics that are covered in this paper;
- Requirements gathering and analysis;
- Study of the suitability of technologies in relation to the requirements;
- Usability analysis (if applicable);
- Software development taking into account the requirements and the elected technologies;

In order to ensure that the resulting software implementation is coherent and consistent with the requirements, periodic meetings will be held, depending on availability between the supervisors and the student, *online* or in person as needed.

1.5 DOCUMENT STRUCTURE

This document is divided into multiple chapters named after an overarching subject. Each one of them aggregates a number of sections and corresponding subsections, where the information that relates to them is displayed.

This initial chapter contains a brief descriptive introduction of the *Hypatiamat* project, namely the reasons behind its existence and the goals it intends to achieve. Simultaneously, it contains the motivation for the dissertation topic, which is directly related to the platform's ecosystem of already existing applications, as well as its utility when materialized. Finally, the proposed methodology for its development is outlined.

Chapter 2 is exclusively dedicated to technological analysis and initial insights on the platform in order to ensure that its technical integration is adequate in function of the current market and tools available.

Chapter 3 goes over all development phases of the question submission back-office project, starting with requirements gathering, and then starts mentioning the technological decisions, initial approach, front-end mock-up prototyping and explaining the process of creating a new *API* server from the ground up.

Similarly, chapter 4 follows a similar structure but it refers to the development of the new "I Want To Solve Questions About...", which adds an extra chapter dedicated to usability analysis.

All of these applications need to be fully deployed in their production servers, so chapter 5 will follow the steps taken in order to achieve exactly that.

Since the documentation of the *API* server endpoints is a good practice in software development, chapter 6 will explain the steps behind its creation and making it accessible to any software developer that may need it.

Lastly, chapter 7 includes the final notes and future work that may be done in order to improve their functionality and efficiency.

STATE OF THE ART

There is one more additional step before starting the development itself, the foundation and basis in which any carried assumption will be based on.

Understanding not only the technologies but also their context in the market is fundamental in order to complete a functional end product, and it is exactly what will be detailed in this chapter.

2.1 ALTERNATIVE INITIATIVES

There are currently several similar alternatives to *Hypatiamat* in the market that seek to combat the academic underachievement that students suffer in several courses, not only Math.

The non-profit organization *Khan Academy* aims to provide free and high-quality education for anyone that requires it. The digital platform (<https://khanacademy.org/>) offers several educational resources on a variety of subjects such as Math, Physics, Medicine and Economy to the users.

Its worldwide success demonstrates the potential for educational projects of this kind and the adoption of online didactic models. Like *Hypatiamat*, it offers mini-games linked to each subject by the form of multiple choice questions and the like, however, it lacks the personalized aspect that the former provides, that is, the possibility of educators being able to work directly with their students using the tools available on the *website*.

Skillshare (<https://www.skillshare.com>) offers thousands of free lessons created by the community for other users. These can also be monetized in several tiers, which ultimately gives the creators the freedom to modify the accessibility of their content as they please.

This platform creates a self-feeding cycle to its advantage where the lessons they offer directly depend on the number of users that are using it. The more people use the *Skillshare*, the more content is provided to the community and thus more users are attracted to it, which is where the cycle appears. In contrast to *Khan Academy*, this platform is typically more business-oriented, which when it comes to having a more personal bond between student and teacher, it is at a greater disadvantage than the former.

Wordwall (<https://wordwall.net/pt-pt/community/>) is an online application that allows the users to create their own didactic resources on several topics.

This initiative had the support of multiple schools and colleges such as Marista (Marca, 2021) because of how easy it is to create personalized content using the templates it provides, such as questionnaires, crossword puzzles, diagrams, etc.

Despite its vast library of content, it lacks an efficient search engine, so a large majority of these mini-games end up not being found by other users simply because they are not easily accessible.

There are numerous web applications that aim to provide entertainment via more complex games, but despite the development of motor and intellectual skills, they do not provide educational resources. Namely, *Coquinhos* (<https://www.coquinhos.com/>), *Poki* (<https://poki.pt/>) and *Brincar* (<https://www.brincar.pt/>) have several video games available but they do not provide the means by which the students can learn the underlying topics for each one.

Evidently, there are considerably more alternatives, but it is important to refer to the previous ones as a point of comparison with *Hypatiamat*, in order to analyze their virtues and market relevance.

2.2 STUDY OF THE PLATFORM

In the previous chapter, it was stated that "*I Want to Solve Questions About...*" is still hosted on the monolithic web application, which requires a study on its underlying architecture before writing the software itself.

There are 4 key pillars to this platform, described by the *LAMP* stack, which was commonly used in the industry back when the project was initially developed. This software bundle has many alternatives in its constituent layers, however the ones that make *Hypatiamat* as a whole are:

- **Linux:** it is the stack's base layer, the remaining ones were picked based on their compatibility with this operating system;
- **Apache Web Server:** used as a web server that is responsible for processing and responding to *HTTP* requests that target the application;
- **MySQL:** relational database management system that persists and allows access to data using tables derived from the application;
- **PHP:** scripting layer built on top of the stack that abstracts the access to the previous layer and renders the *HTML* web pages with its data.

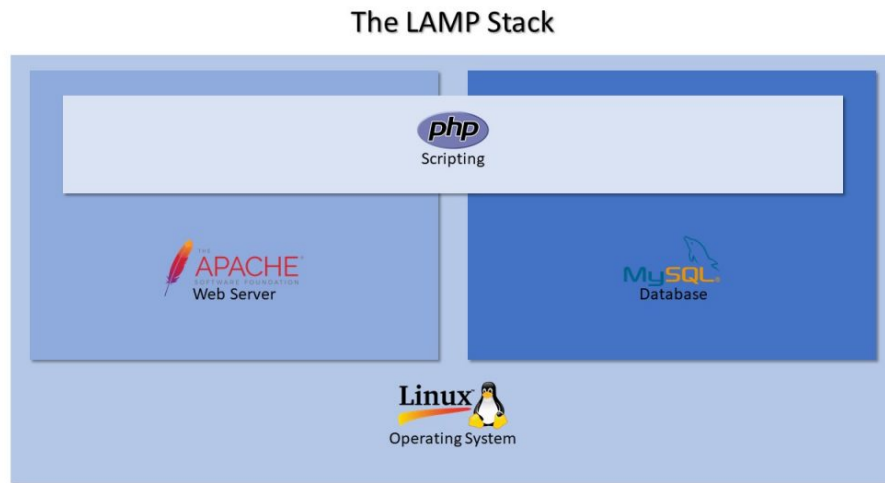


Figure 3: *Hypatiamat's LAMP* architecture

This architecture is typically referred to as monolithic because all these layers coexist in the same system. It typically suffers from poor overall performance since all the machine's resources are shared by all the layers, which causes a bottleneck and consequently impacts its availability and scalability.

Therefore, rendering pages using *PHP* is done on the server. Assuming it has 100 daily requests to fetch the home-page, it will mindlessly execute the same rendering routine for those 100 requests, which contributes to the waste of resources and processing power that could be better used elsewhere (this is discussed in more detail in section 2.3.1). The performance is effectively lower than other modern alternatives, such as *MEAN*, which are the foundation of the various microservices that currently form *Hypatiamat's* backbone.

2.3 STUDY OF THE TECHNOLOGIES

Nowadays, web applications provide a wide range of options regarding the technologies that can be used and the difficulty lies in picking the ones that fit some scenario the best, since they all specialize in different aspects such as the programming language, scope, interoperability, etc.

The gradual process of technological development is especially noticeable in this area, in which the main problems are usually consequences of the poor **scalability**, both horizontal and vertical that they initially had.

The creation and investment in cloud services (Rahman, 2021) such as *GCP*, Azure and *GCP* initially emerged as a response to the growing demand from users, since the servers they had did not have the capacity to handle all the requests successfully due to hardware and software limitations. Consequently, latency problems, low availability and errors in

handling the requests themselves were frequent and, in fact, were generating net capital losses for the companies.

One of the main causes for the aforementioned problems was the large-scale adoption of tools that tended to use server-sided approaches for all routines related to dispatching requests, which entails the entire process from validating them to rendering the *HTML* page that would be sent to the client, which will be explored in more detail in section 2.3.1.

Traditional relational databases are hard to scale horizontally, as they satisfy the *ACID* principles. However, they were the primary choice as the persistence layer for most applications that existed in the start of the 21st century because there simply did not exist any strong alternatives effectively mitigated the issues that these systems brought (covered in section 2.3.2), which led to monolithic architectures being seen as the norm back then.

Thus, the industry trend has moved from using *LAMP* systems, or variations thereof, to *MEAN* systems (2.3.3), as well as its alternatives, which implement the many different tools that arise from technological developments across all elements of the depicted stack.

2.3.1 Client-Sided vs Server-Sided Approaches

The overwhelming majority of web applications work via the client-server model where the communication method is based on requests and responses to a central server by a client, using *HTTP*.

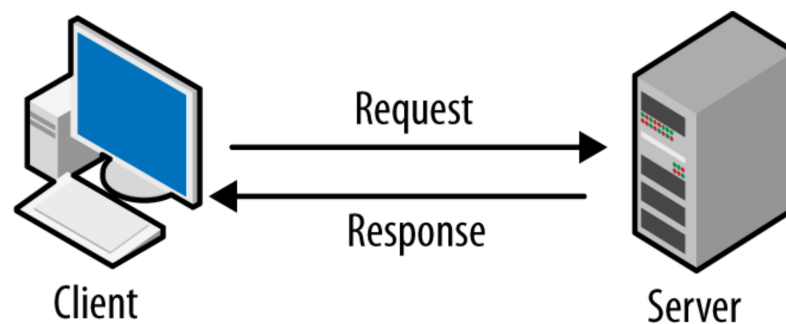


Figure 4: Client-Server Model

In this view, all processes that take place on the client side are called *client-sided*. Alternatively, *server-sided* are the ones that execute solely on the server side (Cloudflare).

In the past, the entire request handling process was server-sided, which included rendering the *HTML* pages, accessing the persistence layer, validating requests, formatting several types of data, among others. The client-sided process was much simpler, as they would only receive text in the form of *HTML* and it would be the user's responsibility to display it and apply the respective styling correctly (after the introduction of *CSS* to the Internet), which was typically performed by the average web browser.

This strategy faces the problem that the vast majority of these renderings have the exact same result, effectively wasting computational resources that could be used for other tasks. It is not enough to generate the *HTML*, which takes a certain amount of time, it is also necessary to access the data layer in order to retrieve the information required for it, which also uses a time frame for its execution. In pages with dynamic content, one would observe poorer performance due to this increase in latency. As seen earlier (2.2), *PHP* is one of the technologies that employ this method.

Recently, technologies have begun to emerge that try to mitigate this problem, that is, instead of assigning the role of scripting and rendering the pages to the server, these are carried out on the client side instead. In this paradigm, they are typically referred to as *API* servers, which only provide the data needed to build the page, which frees the computational burden on the server side of rendering *HTML* that can be used to handle more clients and in a more efficient manner, achieving less latency and a higher availability rates. For example, Node.js, Java Spring, Ruby On Rails, Flask are typically used in this fashion.

To complement this new server-sided approach, on the client side, frameworks like Vue, React and Angular currently dominate the market, as well as other key responsive engines for recent web applications (Kaluža and Vukelic, 2018), as they only consume the data provided by *API* servers and render *HTML* responsively, which allows for a better user experience and more creative freedom for developers.

2.3.2 NoSQL vs SQL

The data definition language is used in relational database management systems such as *MySQL* and *Oracle Database*, which typically consist of strict data models based on their table and column structure.

One of their main features is the adherence to the *ACID* principles in their handling of transactions:

- **Atomicity** - A transaction is only completed if all the steps that comprise it are fully executed;
- **Consistency** - A transaction must not cause inconsistencies in the transition between states defined before and after its execution, that is, all the associated invariants must be kept;
- **Isolation** - A transaction, although there may be several others running concurrently, should not interfere with the result of any of them, that is, the final state should be identical to the outcome of running them sequentially;
- **Durability** - A completed transaction should persist in the database even after a system failure.

Relational based approaches are typically recognized as not being able to scale horizontally, as the rigidity of these principles constrain it in distributed environments.

As a result of the growing need for scalable alternatives, databases have emerged which, by relaxing the *ACID* principles, achieve better performance in exchange for **consistency** since they discard the need for data models, i.e., they allow more freedom when deciding the format of the information that needs to be persisted, which contrasts the table model and its fixed format in approaches.

The flexibility of these formats is mainly seen in the distinct number of options (Wang and Yang, 2017) these systems can adopt, namely:

- **Key-Value Store** - information is represented through key-value pairs in a dictionary, where the key is unique linked to a value with a completely arbitrary representation;
- **Document Store** - data is encapsulated in files with known formats and encodings such as *JSON* and is organized using collections or analogous approaches;
- **Column Store** - tables are used, just as in , but instead of rows they are organized by columns, which is useful for data that requires aggregation operations frequently;
- **Graph Store** - the data is stored via a graph formed using the different relations between the nodes, which is useful for information with highly connected content and properties within them.

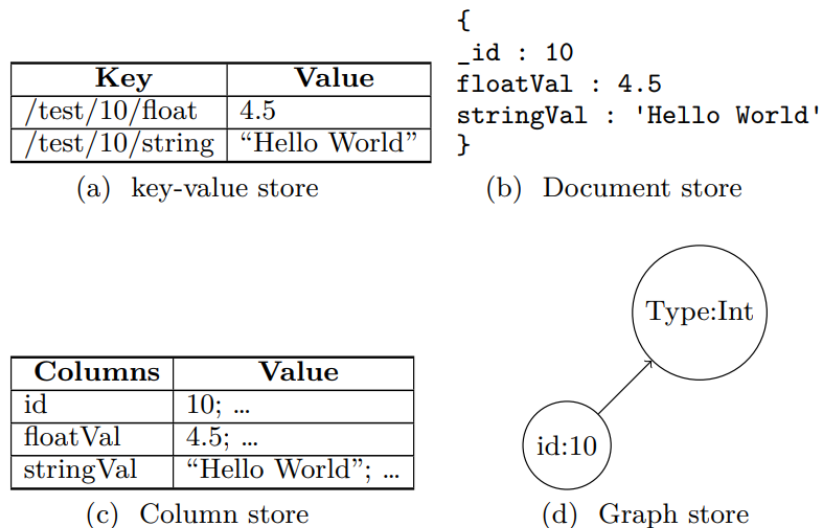


Figure 5: Different types of NoSQL Databases

The *CAP* theorem emerges in this paradigm, in which it suggests that a distributed database system will simultaneously provide only 2 of the following 3 assurances: consistency, availability, and fault-tolerant partitions.

The fact that no distributed system is protected against failures makes fault-tolerant partitions a fundamental requirement, which means it is typically picked as one of the two assurances possible, so the problem relies in picking between the remaining two.

If consistency is chosen over availability, the queries on the partition may return an error or stop executing entirely due to a timeout since the information may not be accessible because of its distribution across multiple devices over the network, be it because of faulty systems or latency issues.

Alternatively, availability may be chosen over consistency ensuring that the partition always answers the query with the most recent information at that time, despite the possibility that it may not be up to date compared to the others in the system.

On the other hand, relational databases tend to follow a more conservative model in which they privilege consistency and availability, due to the restrictions associated with the *ACID* principles and there being no need for replicas to be considered in the system on account of poor horizontal scalability.

2.3.3 *MEAN vs LAMP*

LAMP architectures as seen in 2.2 are known for their low scalability because of shared resources in the same system.

However, the use of easy-to-learn languages in their technology stack makes it a good alternative for small applications that need to be deployed as quickly as possible, as well as those that demand strict data models to be used in the relational approach.

The main disadvantages of adopting this solution, excluding the lack of horizontal scalability as seen previously, are its restrictive choices on operating systems (Linux only) and the use of very dissimilar languages (*PHP*, Python, ...) which, despite their simplicity, can cause confusion in the development process due to the different ways they handle the same data.

After a few years of technological development, the *MEAN* stack emerges, which aims to address the issues on the many layers of the *LAMP* architecture separately with the goal of improving all of them with newer and more efficient alternatives.

- **MongoDB** - traditional relational databases are discarded in favor of using systems based off of which, in this particular case, is a *document store* (2.3.2) approach that uses *JSON* as its document format;
- **Express.js** - this framework acts on top of Node.js and automatically generates boilerplate code common to every web application, such as general-purpose configuration files and middleware creation;

- **Angular** - reactive client-sided framework (2.3.1) that allows client-side rendering of pages and many other useful features operated with TypeScript, a superset of JavaScript;
- **Node.js** - responsible for creating the server that responds to *HTTP* requests and handles client requests, written in Javascript.

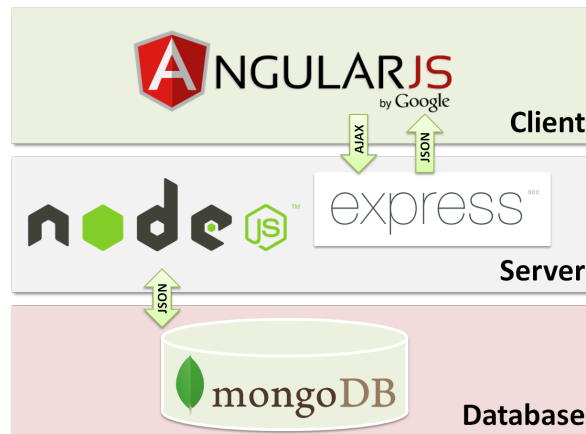


Figure 6: *MEAN* Technological Stack

There are other alternatives to this stack where the main difference is the Angular layer, since there are several frameworks on the market with similar function and performance. The most well-known and used are the *MERN* stack that uses React and the *MEVN* stack that is centered around Vue.js (Karanjit, 2016). These are all alternatives that seek efficiency, responsiveness and load de-copulation from the main server, which is the current foreseeable trend for web development.

No matter the stack variation, there is always the guarantee that all of the layers work in any operating system, one of *LAMP*'s shortcomings. However, the main benefit is related to the improved performance brought upon the application server due to the several stack layers' native support for *JSON*. The conversion of these objects into persistent data in the database or communication between the layers other does not require any intermediate step (marshalling and unmarshalling) because it is a common type to all of them.

This stack and its variants has taken the world by storm and it has been adopted on a large scale, including by multinational companies such as Google and IBM, due to its flexibility and its extensive applicability in progressive web applications. Most companies realized that a better user experience coupled with less overall costs on server maintenance increases overall profit when compared to *LAMP* alternatives.

QUESTION SUBMISSION BACK-OFFICE

The back-office's main goal is to simplify the teacher's current method of adding questions to the database.

Instead of immediately diving into developing the code for the project itself, it is important to dedicate some time to understanding what requisites that need to be implemented and their consequence in the final product.

Only after this initial step should any logical model or code be started, it is unwise to start developing something that can be rolled back simply by changing something minor, which impacts productivity in the long term and consequently stalls the business.

3.1 REQUIREMENTS SPECIFICATION

Before attempting to implement anything using software, it is of utmost importance to take some time in the development cycle to devise a strategy taking into account all the functionalities that the project should provide.

Ignoring this stage may result in many mistakes further into the development cycle, which include missing or not fully implementing a feature, rewriting some code segments multiple times because the edge cases were not fully fledged out from the beginning, and so on (West, 2011).

Assuming this stage is done correctly, all that remains is to effectively implement all of the features, there is no wasted time in-between cycles rewriting or rethinking functionalities and technological decisions.

It is important to mention that this back-office will work in parallel to *Hypatiamat*, as stated previously. Thus, decoupling them will allow for greater technological and creative flexibility, so all the requirements will be outlined in this light.

3.1.1 Functional Requirements

The back-office's main purpose is to allow the insertion of new questions for mini-games or, alternatively, to edit or remove those that have already been inserted by the *Hypatiamat* team, which is inherently implied that it must provide ways to execute *CRUD* operations on the questions table.

However, this possibility is only open to the teachers who own *Hypatiamat*, i.e, the users with the highest permission level in all the applications associated with the project. Authenticating the user ensures that they belong to this very specific class of individuals and that they are not someone who could potentially damage the integrity of the questions, which is something that must be avoided.

Deleting a question can be done without any intermediate step. According to the owners, the need to delete a question occurs very rarely and only when it is deemed truly unsalvageable. Since they all know each other personally, there is a relation of trust between them and they all agree on not being ill-intentioned when making such drastic changes.

However, regarding the insertion of a new question, it should not be added directly into the questions table. Whenever a teacher intends to create a new question, it must go to an intermediate **quarantine** state. The difference between adding and rejecting is that the submission may have **errors** and, as such, they should only be accepted if at least two people thoroughly verify it and consent to it. Decisions on a submitted question can never be made by the same element that submitted it, i.e, for a question to be inserted in the database it must be put up for approval by a teacher and another must approve it. In this quarantine phase, the submission can be:

1. **Pending** - this is the default state, no tables are changed while the submission awaits a decision;
2. **Rejected** - the teacher does not agree that the submission is relevant or pertinent to the mini-games and other applications, leaving the questions table unchanged;
3. **Accepted** - the teacher agrees that the submitted question should be added to the platform, resulting in its addition to the questions table;
4. **Edited** - the teacher detects some kind of error, incoherence or inconsistency on the submission and intends to fix it, resulting in the original submission being discarded in favor of a new pending one where the ownership is transferred to them and with the implemented fixes, leaving the questions table unchanged;

Any question in the database prior to the creation of this back-office is considered the result of an accepted submission.

Whenever a teacher wants to edit an accepted question, a new pending submission should be created with all the changes they want to make and it should be removed from the table entirely, since it is not their intention that the students have access to something deemed wrong.

The back-office should also provide a collection of filters to the questions table so that the teachers have an easier time finding specific ones. There are currently over 3000 questions in the database, which means it should be possible to find all the questions that belong to a certain theme instead of scrolling through all of them to find those select few, for example.

It is not enough to be able to filter the questions, it should also be possible to **preview** them, to make it easier to visualize the way they will appear to the students so that adjustments can be made accordingly.

One of the shortcomings of "I Want to Solve Questions About..." is that it does not allow the owners to **hide** certain questions without removing them entirely from the database. Since one of the goals of this project is to completely rework it, this back-office should be capable of handling this new feature without changing the existing tables.

The second one is that it does not allow the owners to change the order in which the themes of the questions appear in the theme selection screen, which is something they would like implemented in this new version.

The third one is that it does not provide a way to change the images associated with the themes in the same screen as before, due to them being hard-coded in the original application. This would be useful in the case they happen to dislike any of them, since they could just easily swap it out without changing any of the legacy code.

The fourth and final one is that it lacks a deeper "gamification" factor and a reason to incentivize the users to return, i.e, a way to have a more engaging experience with the other users, not just being able to solve questions. In order to improve this, the teachers decided that the new revised version should also have a monthly questions selection of questions divided by 4 categories:

1. **Level 1:** aimed exclusively at students in the 1st and 2nd grades;
2. **Level 2:** aimed exclusively at students in the 3rd and 4th grades;
3. **Level 3:** aimed exclusively at students in the 5th and 6th grades;
4. **Level 4:** aimed exclusively at students in the 7th, 8th and 9th grades;

This selection should only last the month that they were published at, resetting on the first day of each one. The students should be able to submit their answers to a question if they belong to the grades in that specific category, within this time frame and it should be final, i.e, the student can not change their answer after submitting it.

It is important to note that every single question that is selected for any edition should be **hidden** until the reset day arrives, since the students could just easily search the solution prior to the submission.

Keeping the history of previous editions of the monthly questions would be a favourable addition as it would allow the owners to keep track of which questions were already selected in order to keep a large variety of questions every month.

3.1.2 Non-Functional Requirements

A basic requirement of the back-office is that it has to be a cross-platform web application, it should work both on an average computer and on a mobile device, granting the user the possibility to access it from anywhere. However, the priority is a functional version for desktop computers and laptops, having it be mobile-friendly is a plus.

Evidently, the fact that it will be developed as a web application means that it should be taken into account it may be accessed by the many different browsers in the market. Making it compatible with them, ensuring that it at least works on the most popular ones (*Chrome, Safari, Edge and Firefox*) (Statista, 2021), is fundamental since the teachers do not use the same browser every time nor are they expected to do so in the future, due to their individual preferences not being final.

Initially, it was mentioned that interface has to be consistent with the multiple micro-services of *Hypatiamat*. In particular, the basis of comparison will be with the *TPC* management system designed by João Vieira (Vieira, 2021), since it was conducted in exactly the same way, by following the guidelines from previous projects.

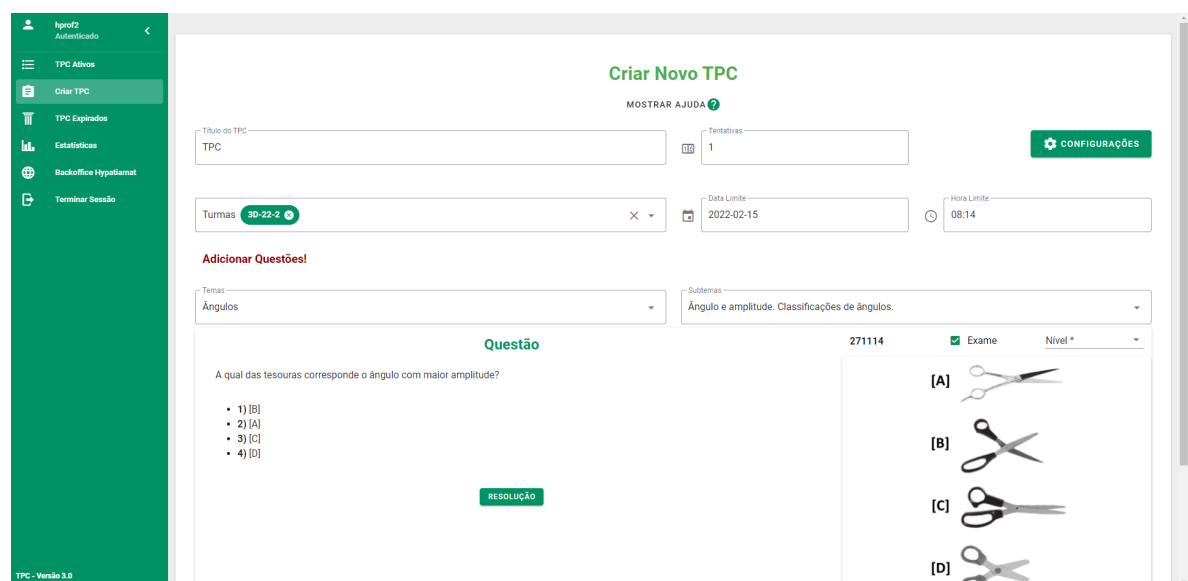


Figure 7: Interface for the insertion of a new *TPC*

Regarding the interface for inserting a new *TPC*, available when a user successfully authenticates, there are a few caveats regarding its design. Namely, there is a retractable navigation bar on the left that fills a percentage of the screen vertically and it contains all the relevant navigation links for the application. The remaining space is filled with the specific content that needs to be displayed in each page.

In addition to this, the color palette is also important. The content section (right) uses shades of green to highlight titles, buttons or any element that needs to draw more attention; white for empty sections; black for general-purpose text; shades of grey for text that requires a more subtle presence such as help messages, as well as borders for input sections; red to indicate some kind of error. The navigation bar (left) only uses green as a background color and white text (preceded by explanatory icons) to contrast it. The font is identical between these two components, where only the sizes vary depending on the information that needs to be rendered.

The back-office must keep these design choices in order to achieve the desired consistency between all the applications that comprise *Hypatiamat*. It is not enough to follow these guidelines, it should also be intuitive to use, with a simplified user flow, since the platform will be used by teachers of varying technical expertise. In order to achieve this, the use of tools that allow the creation of responsive and reactive web applications, such as the ones used by João in the *TPC* manager, are critical to its success.

Any question, whether they are quarantined or not, should also not break any application previously developed, as stated previously. There should be mechanisms in place that thoroughly error-check them and ensure that no corruption of data occurs.

3.2 INITIAL ASSESSMENT

Hypatiamat already has a database server in production with multiple databases to ensure that the different types of data are compartmentalized and organized. It is currently deployed using MySQL and they all have the same prefix ("*hypati67_*") followed by their identifier that represents what type of tables are contained within it. For example, "*hypati67_aplicacoes*" has all the tables related to the types of users that are registered in the platform, which can be a teacher or a student, their school and class (if applicable), and so on.

As seen in 3.1.1, the back-office should only allow access for its data-manipulating functionalities to the owner teachers of *Hypatiamat* itself who stand at the top of the hierarchy, as they are the only ones who ultimately give the final decision on what gets added or not into every application the website offers. What this entails is an authentication system that filters out anyone that is not part of this very small group of people and, in order to do that, it needs access to the database responsible for storing all of the information necessary for this process, which is the focus of section 3.2.1.

The main features of this application revolve around manipulating the database that stores all the question data (3.1.1). For this to become possible, the back-office will also need access to the database that currently holds all the data on the ever-increasing amount of distinct questions and their associated types, explored further in 3.2.2.

Naturally, both of these very distinct databases **must not** be corrupted in any way and since they are the foundation for the wide variety of applications *Hypatiamat* offers. Adding to this, it is of utmost importance that the **structure** of the tables are not changed since they can have negative impacts on the already established services due to incompatibilities between data and can cause total shutdown for maintenance, in the worse case scenario.

Assuming these databases are permanent, since as seen above it is unlikely there will ever be any change regarding which one the tables are saved in, the back-office only needs access to these 2 **existing** databases.

Evidently, since this a new application it will most likely need new tables (especially because of the "quarantine" feature) which begs the question: where do they go? There are several possibilities, however, it is important to note that this application should not interfere in any way with the current compartmentalized structure of the MySQL server and the corruption issues mentioned above, which means the best option in these circumstances is to create a new database that contains only the tables related to the question submission back-office, leaving the others the way they are.

All that remains is to settle on a name for this new database, which must follow the aforementioned rule in order to maintain consistency with the others. After the approval of the owners, it will be named "*hypati67_questoes*" and will be initially empty, as expected, and permission to execute any operation on its content by the back-office.

It is important to note that, henceforth, any database schema or image that may appear as a representation of any database or table in the MySQL server will be a **simplified** version with only the necessary information, i.e, without all its relations, foreign keys and indexes as they are too many and would cause unnecessary confusion and visual clutter.

3.2.1 Teachers

In the previous section, it was given as an example that the database "*hypati67_aplicacoes*" hosts all the information on teachers, students, schools, classes, etc and that one of the requirements for the back-office is blocking out the access of anyone that is not an owner.

Thus, the first step in designing a solution for the authentication is studying and understanding the teachers table specifically. Simultaneously, it is important to note that the schools table is also important since it contains specific information of the school where they teach at, which will be relevant further into the paper.

The logical model below represents the schools and teachers tables, where the "escola" column in "professores" (teachers) is a reference to the "cod" column in "escolas" (schools).



Figure 8: Teachers and schools tables in "hypti67_aplicoes"

On the left side, the table "professores" contains the following columns:

- **id** - auto-incremented primary key;
- **codigo** - unique identifying code of the teacher;
- **nome** - name of the teacher;
- **email** - email of their *Hypatiamat* account;
- **password** - hashed password of their *Hypatiamat* account, calculated with BCrypt;
- **confirmacao** - whether or not their account has been confirmed;
- **premium** - integer in the interval 0-5 that indicates the permissions they have in the platform, where 5 is only given to the owners of the platform;
- **validade** - date when their permissions expire;
- **socionum** - *Hypatiamat* partner number (if applicable),
- **projeto** - which project is associated with them;
- **school** - unique identifying code of the school where they teach at;

On the right side, the table "escolas" contains the following columns:

- **id** - auto-incremented primary key;
- **nome** - name of the school;

- **localidade** - town where the school is located;
- **distrito** - district where the school is located;
- **pais** - country where the school is located;
- **cod** - unique identifying code of the school;

After analysing this logical model, the immediate conclusion is that "email", "password" and "premium" columns are relevant to the authentication, since they provide a way to identify the user that attempts to log into the platform. However, there is one more condition that needs to be validated, which is whether the date when their permissions get revoked has been reached or not, accomplished via the "validade" column.

3.2.2 Questions

The questions are the main focus of the back-office, in fact, its purpose is to offer several functionalities that interact with them in some way. Therefore, analyzing the type of questions and its quirks present in the MySQL server is fundamental to achieving a successful end product.

The database responsible for hosting them is "hypati67_testeconhecimentos" and it also contains other important information, most of which is not relevant for the completion of this project. The only exception is the themes table, which contains the names of the themes and sub-themes of the questions, granting less redundant and repeated entries.

The logical model below represents these two tables, where "Base_dados" is the aforementioned questions database and "subtemas" contains the themes and sub-themes associated with them. The columns "tema" and "subtema" in the former refer to "codtemaN" and "codsubtema" in the latter, respectively.

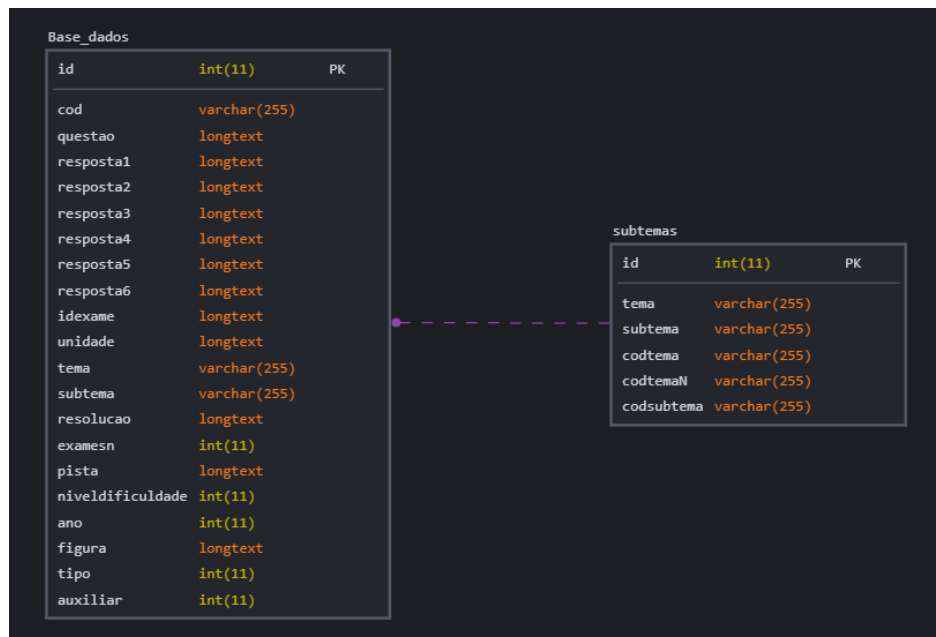


Figure 9: Questions and themes tables in "hypati67_testeconhecimentos"

The questions table is comprised of the following columns:

- **id** - auto-incremented primary key;
- **codigo** - unique identifying code for the question;
- **questao** - question statement that may contain the following sequence of special characters:
 - **
**: line break;
 - **x<sub>y<sub>**: powers (x^y);
 - **<raiz>x<raiz>**: square roots (\sqrt{x});
 - **%2B**: addition symbol (+);
 - **+xpto e -xpto**: $+\infty$ e $-\infty$, respectively.
- **resposta1 ... resposta6** - the information these columns store may also contain the same sequence of characters as before and their content depends on the "tipo" and "auxiliar" fields, which are explained below;
- **examesn** - binary value that indicates whether the question came from an exam/test or not;
- **idexame** - the name of the exam/test the question came from, if applicable;

- **unidade** - if "tipo" is 1 and the statement requires a measurement unit, this column contains it;
- **tema** - identifying code of the question theme;
- **subtema** - identifying code of the question sub-theme;
- **resolucao** - contains the [URL](#) to the image of the question's solution in case it exists;
- **pista** - redundant legacy column;
- **niveldificuldade** - integer in the interval 1 to 4 that indicates the difficulty level of the question;
- **ano** - integer in the interval 1 to 9 that indicates which grade this question is aimed at;
- **figura** - contains the [URL](#) to the image that is used in junction with the statement for better clarity or auxiliary information in case it exists;
- **tipo** - integer in the interval 0 to 7 that indicates the type of the question itself, which in turn indicates the contents of "resposta1" through "resposta6":
 - 0 - multiple choice with 4 available **text** answers, each one of these will occupy the slots "resposta1" through "resposta4";
 - 1 - open-ended question with several text options, where the contents of "resposta1" through "resposta6" are determined by the "auxiliar" column as seen below;
 - 2 - multiple choice with 4 available **image** answers, each one of these will occupy the slots "resposta1" through "resposta4" using their [URLs](#);
 - 3 - true or false with only 2 possible answers which only use the "resposta1" and "resposta2" columns;
 - 4 - multiple choice with 3 available **text** answers, similar as type 0 but requires one less "resposta" column ("resposta1" to "resposta3");
 - 5 - multiple choice with 5 available **text** answers, similar as type 0 but requires one more "resposta" column ("resposta1" to "resposta5");
 - 6 - multiple choice with 6 available **text** answers which require the slots "resposta1" through "resposta6";
 - 7 - grid symmetry, where "resposta1" is used to store its layout and axis;
- **auxiliar** - integer that defines variants within the open-ended type of questions and its default value is 0 to the others:

- **0** - all the answers in any of the columns from "resposta1" up to "resposta6" are accepted;
- **1** - only "resposta1" exists and is the only acceptable answer;
- **2** - only "resposta1" and "resposta2" are filled and the correct answer is the irreducible fraction resulting from (*resposta1/resposta2*);
- **22** - similar to **2** but it accepts any equivalent fraction to (*resposta1/resposta2*), it does not have to be in its irreducible form;
- **10** - indicates that a protractor is required in order to calculate some angle, where the correct answer is in the interval between "resposta1" and "resposta2";
- **1000** - only "resposta1" and "resposta2" are filled and both have the correct solution.

The themes table is simple when compared to the previous one:

- **tema**: name of the theme;
- **subtema**: name of the sub-theme;
- **codtema**: redundant legacy column;
- **codtemaN**: unique identifying code for the theme;
- **codsubtema**: unique identifying code for the sub-theme;

It is important to note that **any** question that may be added in the future to the database **must** maintain this format and follow all of the rules above, since all the other micro-services that use these it would break in some way were there to exist any deviations, something that is extremely against the main goals of the project. This also includes using valid codes for the themes and sub-themes.

3.3 TECHNOLOGY SELECTION

After many iterations, the following image shows the technological stack that will be the foundation for the back-office's development:

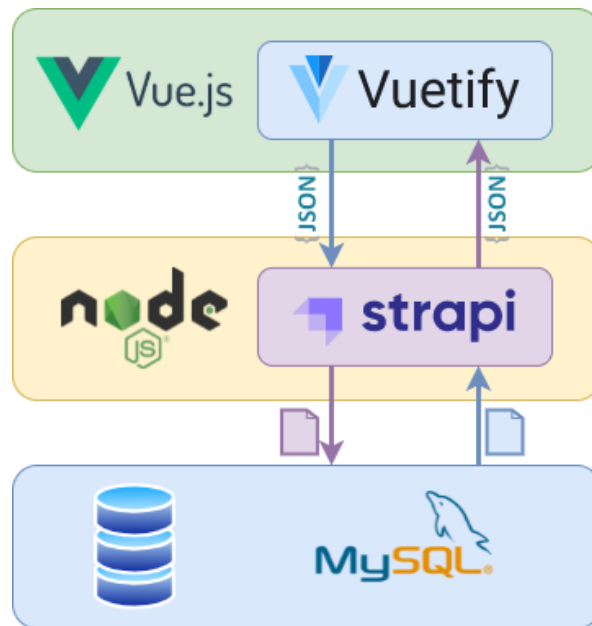


Figure 10: Technological stack for the back-office

As stated previously in chapter 2.3.3, *MEAN* architectures are currently dominating the market when it comes to developing full-stack web applications, mainly because they offer a greater scaling capability when compared to the *LAMP* alternatives. Thus, it makes sense that the back-office should follow a similar or equal ideology since it is also going to be a web application.

The selected stack in figure 10 depicts a slight variation on *MEVN*, namely on the MongoDB, Express.js and Vue layers. In this paradigm, Strapi will act as a RESTful *API* server and whenever the client (Vue.js) needs access to the persisted data, it will send a request which will be handled by the server and it will send back a response with the resulting *JSON* object. The browser running Vue.js will then render the *HTML* pages using this data.

Starting with the persistence layer, MySQL is used in every single application that the project offers and it contains all the necessary data for their functionality. Assuming a new MongoDB server instance is created, it would have to keep an updated read-only version of all the necessary databases (3.2) and extra local collections to ensure that it works seamlessly in tandem with MySQL, which requires a lot of processing power and maintenance. It would be the optimal solution if the back-office were to be a totally independent application but this is not the case, so, a compromise has to be made and that is to use the already existing MySQL instance, despite the loss of performance when handling persisted information (transforming result-sets into *JSON* instead of using only *JSON* like in MongoDB), it pales in comparison to the performance hit the MongoDB alternative would cause in the overall system.

Express.js is a very powerful framework that automatically creates templates with boilerplate JavaScript (Node.js) code and modularizes the *REST* routes in folders so that they are easily created and accessible. Strapi goes a step further and **automatically** generates *CRUD* routes for every table that it is given access to and it also offers complete compatibility. The administrator has access to a web application where they can visually set up the tables and their relationships, create permission levels and associate any route with them and use a complex querying system on any table to avoid using `directly`, just to name a few features. Evidently, this tool is extremely powerful and it still uses the “blazing-fast” Node.js engine that Express.js is built on, while adding a few improvements and features that greatly improve the developer’s efficiency.

Similarly, Vuetify is a framework that adds new user interface functionalities to Vue.js, by providing a wide variety of easy to use components that positively impact the user experience on the end product. It is extremely popular since it is an easy way for a developer to create complex-looking applications without having a deep understanding of *CSS*, since it is abstracted by the use of Vuetify-specific classes. It makes sense to use it in this particular back-office because it was also used in the other projects, which makes it possible to reuse their components without “reinventing the wheel” and adding new content pages quicker.

In short, the final structure of the back-office follows a relatively close variant to *MEVN* with a front-end built using Vue.js and Vuetify, a back-end built with Strapi and Node.js and MySQL as the persistence layer, by tweaking or improving the layers in order to comply with the implicit rules imposed by the other micro-services.

3.4 LOGICAL DATA MODEL

The first step in implementing the functional requisites is designing the tables and relationships between them which will fill the “hypati67_questoes” database. This can be done using a logical data model, as seen below:

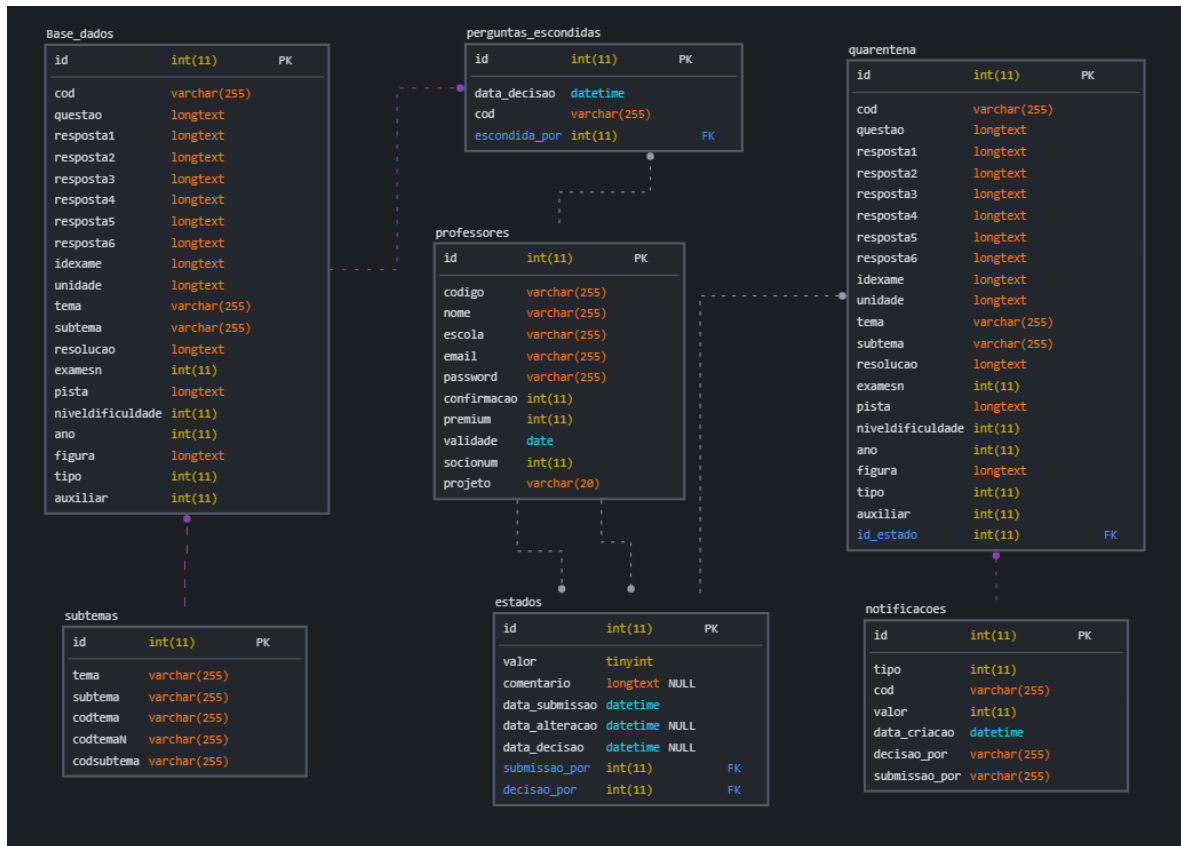


Figure 11: Logical data model for "hypati67_questoes" (Simplified)

The tables "Base_dados" and "subtemas" belong to "hypati67_testeconhecimentos" and "professores" is part of "hypati67_aplicacoes", as stated in chapter 3.2, meaning that these will not be a part of the new database, they are just added to the simplified model in order to show the relationships between the new tables and the already existing ones.

In order to allow the creation of question submissions, the table "quarentena" is the first one that needs to exist. It is a direct copy of the questions table since a submission must contain the information necessary in order to create a new entry on the former table. However, it has an added column named "id_estado" that acts as foreign key to the table where all the data related to the **state** of the submission is stored, as seen in chapter 3.1.1, and other meta-data:

- **id** - auto-incremented primary key;
- **valor** - integer in the range from 0 to 2 that refers to the current state of the submission, where:
 - 0 - Pending;
 - 1 - Accepted;

- 2 - Rejected;
- **comentario** - if the submission was rejected, the teacher may leave a comment explaining why;
- **data_submissao** - date when the submission was first created;
- **data_alteracao** - date when the submission was edited;
- **data_decisao** - date when the submission was given a final decision by a teacher;
- **submissao_por** - foreign key to the teacher who created the submission;
- **decisao_por** - foreign key to the teacher who gave the final decision on the submission.

Since the questions table should not be altered in any way, in order to allow the teachers to hide questions there needs to be a new table which stores that information, which in this case is called "perguntas_escondidas" and contains the following fields:

- **id** - auto-incremented primary key;
- **cod** - identifying code for the question that is hidden;
- **data_decisao** - date when the question was hidden;
- **escondida_por** - foreign key to the teacher who hid the question.

The notifications table was not created in function of a required feature, its purpose is to notify the teachers whenever the state of their submission changes or whenever a new submission was created by another user. Despite being an extra feature, it creates a more fluid experience and a better user experience since they do not have to manually check whenever either of these events happen. It contains the following columns:

- **id** - auto-incremented primary key;
- **tipo** - defines the notification type which can be either:
 - 0 - Result of the decision on a submission;
 - 1 - New submission;
- **cod** - identifying code for the submitted question;
- **valor** - if the "tipo" column is set to 0, this field refers to the result of the decision on the submission:
 - 1 - Accepted;
 - 2 - Rejected;

- **data_decisao** - date when the notification was created;
- **decisao_por** - if the "tipo" column is set to 0, this column refers to who gave the final decision on the submission;
- **submissao_por** - defines the teacher who caused the notification to be created.

Some of the requisites imply the **deletion** of information, which is inherently a permanent operation that causes the loss of data for all the *Hypatiamat* applications. Since memory is extremely cheap nowadays, a better solution would be to flag the information as **unavailable** using an extra column or migrate them to tables with the same structure that keep all the deleted entries. By taking into account that one of the requirements is to not alter any of the existing tables, the first option is immediately discarded in favor of the second one.

These "delete" operations may occur in the questions and submissions tables, granting the need for two additional tables that act as backups in the case that the information needs to be restored, namely "perguntas_backup" and "submissoes Eliminadas", respectively. Whenever a question or a submission is deleted that specific entry is removed from their respective table, however, a copy of it is saved as a backup. This basic feature allows the teachers to make mistakes without being permanently punished for it, as they now have the ability to restore them in case they need to.

There are also a few tables created by Strapi which should not be altered or manually updated, since they work as the core of the instance itself and they keep all the information required for it to work correctly. The image below lists every table in the "hypati67_questoes" database and those that belong to Strapi are the ones not mentioned above.

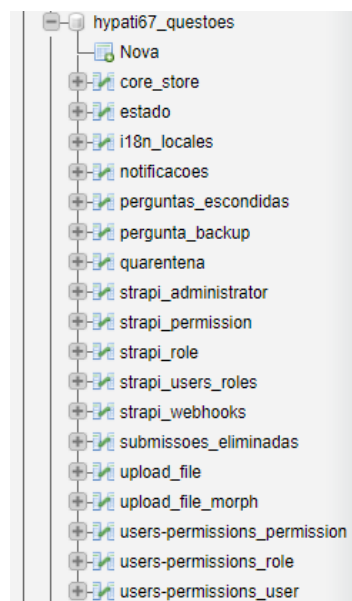


Figure 12: All the tables in "hypati67_questoes"

Regarding the other requisites, such as associating images to themes, sorting them by some order and the monthly questions, they will be done in a separate database, which will be explained in chapter 4.4. These new tables have to be detached from "hypati67_questoes" since they are not related to the back-office itself, like the "hypati67_aplicacoes" database with the teachers table for example. This way, the different types of information are kept separately, which is the implicit rule of the *Hypatiamat* MySQL server.

3.5 BACK-END DEVELOPMENT

After completing the logical data model, the next step is to configure Strapi correctly, as it is the technology of choice for this particular use case.

Strapi is a self-hostable, customizable and open-source headless *CMS* for creating and managing RESTful or GraphQL servers that runs on top of a Node.js engine. It features an administrator panel where it allows the management of every feature without explicitly writing JavaScript code, making it an attractive option for any developer if they want to quickly develop and deploy an *API* server.

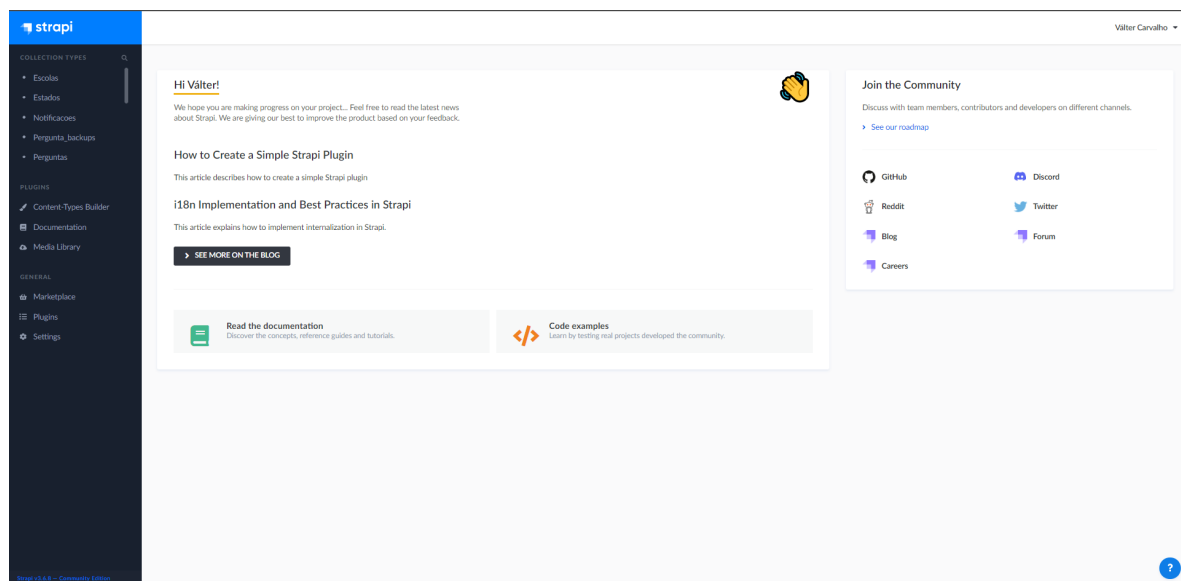


Figure 13: Strapi's Interface

The following sections will go in detail into how a barebones Strapi instance was configured in order to accommodate all the required features, where most of the process is done through the interface displayed above.

3.5.1 Initial Setup

Connecting a new Strapi server to an existing database is done through a single JavaScript file which is located in `/config/database.js`, assuming that the current working directory is the root of the installed Strapi instance. Note that the following connectors do not match with the actual *Hypatiamat* credentials, as they are kept secret in order to respect their privacy policy, instead they only refer to the *localhost* build of the *API* server.

```

1 module.exports = ({ env }) => ({
2   defaultConnection: 'default',
3   connections: {
4     default: {
5       connector: 'bookshelf',
6       settings: {
7         client: 'mysql',
8         host: env('DATABASE_HOST', '127.0.0.1'),
9         port: env.int('DATABASE_PORT', 3306),
10        database: env('DATABASE_NAME', 'hypati67_questoes'),
11        username: env('DATABASE_USERNAME', 'root'),
12        password: env('DATABASE_PASSWORD', ''),
13        ssl: env.bool('DATABASE_SSL', false),
14      },
15      options: {}
16    },
17    testeconhecimentos: {
18      connector: 'bookshelf',
19      settings: {
20        client: 'mysql',
21        host: env('DATABASE_HOST', '127.0.0.1'),
22        port: env.int('DATABASE_PORT', 3306),
23        database: env('DATABASE_NAME', 'hypati67_testeconhecimentos'),
24        username: env('DATABASE_USERNAME', 'root'),
25        password: env('DATABASE_PASSWORD', ''),
26        ssl: env.bool('DATABASE_SSL', false),
27      },
28      options: {}
29    },
30    aplicacoes: {
31      connector: "bookshelf",
32      settings: {
33        client: "mysql",
34        host: env("DATABASE_HOST", "127.0.0.1"),
35        port: env.int("DATABASE_PORT", 3306),
36        database: env("DATABASE_NAME", "hypati67_aplicacoes"),
37        username: env("DATABASE_USERNAME", "root"),
38        password: env("DATABASE_PASSWORD", ""),
39        ssl: env.bool("DATABASE_SSL", false),

```

```

    },
    options: {},
  },
  quereresolverquestoes: {
    connector: "bookshelf",
    settings: {
      client: "mysql",
      host: env("DATABASE_HOST", "127.0.0.1"),
      port: env.int("DATABASE_PORT", 3306),
      database: env("DATABASE_NAME", "hypati67_qrq"),
      username: env("DATABASE_USERNAME", "root"),
      password: env("DATABASE_PASSWORD", ""),
      ssl: env.bool("DATABASE_SSL", false),
    },
    options: {},
  }
},
});

```

Since *Hypatiamat* has multiple databases, it required multiple connection strings, as seen above. Since Strapi aims to provide a better user experience, it removes the need to create the connection strings by hand since the administrator may opt to use a different type of database in the future and that would force them to redo them entirely from the start. Instead, they only have to specify the correct configuration parameters and Strapi will automatically create the connection strings for them.

One of many features Strapi offers is allowing the modelling and creation of “collections”, the term used by Strapi which equates to tables in , for example. They are comprised of multiple “attributes” each having a type (date, string, integer...), which in are the columns. Each collection also allows the creation of multiple relationships with others, which can either be **one-to-one**, **one-to-many**, **many-to-many**, **one-way**, **many-way** or **polymorphic** (Onyenma, 2022).

Creating the tables in Strapi is straight-forward once you have a logical data model to work with. The first step is to create a new “collection type”, which defines its name. The second step is to navigate into the created collection and add all the “attributes” with the correct data types. The final step is linking the collection to the correct table in the database, using the previously configured connection strings and adding only the table name. When this is done, Strapi will automatically add the table to the database or, in case it already exists, update it. Naturally, any operation of data manipulation or retrieval in these collections is done as if they were executed on the tables themselves.

On a side note, there is no need to specify the existence of an "id" field, Strapi automatically generates an auto-incremented one and hides it from the collection schema.

These are the collections the back-end is currently using (translated to English) and the tables they link to:

- **School** - "escolas" in "hyati67_aplicacoes"

Ab	nome	Text
Ab	cod	Text
Ab	localidade	Text
Ab	distrito	Text
Ab	pais	Text

Figure 14: Strapi School Collection

- **State** - "estado" in "hyati67_questoes"

123	valor	Number
Ab	decisao_por	Text
Ab	comentario	Text
Ab	submissao_por	Text
📅	data_submissao	Datetime
📅	data_decisao	Datetime
📅	data_alteracao	Datetime

Figure 15: Strapi State Collection

- **Notification** - "notificacoes" in "hyati67_questoes"

Ab	cod	Text
123	tipo	Number
Ab	submissao_por	Text
Ab	decisao_por	Text
123	valor	Number
📅	data_criacao	Datetime

Figure 16: Strapi Notification Collection

- **Question Backup** - "perguntas_backup" in "hypati67_questoes"

Ab	cod	Text	123	examesn	Number
Ab	questao	Text	Ab	pista	Text
Ab	resposta1	Text	123	niveldificuldade	Number
Ab	resposta2	Text	123	ano	Number
Ab	resposta3	Text	Ab	figura	Text
Ab	resposta4	Text	123	tipo	Number
Ab	resposta5	Text	123	auxiliar	Number
Ab	resposta6	Text	Ab	tema	Text
Ab	idexame	Text	Ab	subtema	Text
Ab	unidade	Text	Ab	eliminador_por	Text
Ab	resolucao	Text	📅	dataeliminacao	Datetime

Figure 17: Strapi Question Backup Collection

- **Questions** - "Base_dados" in "hypati67_testeconhecimentos"





















 cod	UID	 resolucão	Text
 questao	Text	 examesn	Number
 resposta1	Text	 pista	Text
 resposta2	Text	 nivelidificuldade	Number
 resposta3	Text	 ano	Number
 resposta4	Text	 figura	Text
 resposta5	Text	 tipo	Number
 resposta6	Text	 auxiliar	Number
 idexame	Text	 tema	Text
 unidade	Text	 subtema	Text

Figure 18: Strapi Question Collection

- **Hidden Question** - "perguntas_escondidas" in "hyati67_questoes"


 cod	Text
 escondida_por	Text
 data_decisao	Datetime

Figure 19: Strapi Hidden Question Collection

- **Teacher** - "professores" in "hyati67_aplicacoes"









 codigo	UID
 nome	Text
 escola	Text
 email	Text
 password	Text
 confirmacao	Number
 validade	Date
 premium	Number

Figure 20: Strapi Teacher Collection

- **Quarantine** - "quarentena" in "hyati67_questoes"

cod	UID	examesn	Number
questao	Text	pista	Text
resposta1	Text	niveldificuldade	Number
resposta2	Text	ano	Number
resposta3	Text	figura	Text
resposta4	Text	tipo	Number
resposta5	Text	auxiliar	Number
resposta6	Text	tema	Text
idexame	Text	subtema	Text
unidade	Text	id_estado	Number
resolucao	Text		

Figure 21: Strapi Quarantine Collection

- Deleted Submission - "submissoes_eliminadas" in "hypati67_questoes"

cod	UID	examesn	Number
questao	Text	pista	Text
resposta1	Text	niveldificuldade	Number
resposta2	Text	ano	Number
resposta3	Text	figura	Text
resposta4	Text	tipo	Number
resposta5	Text	auxiliar	Number
resposta6	Text	tema	Text
idexame	Text	subtema	Text
unidade	Text	submissao_por	Text
resolucao	Text		

Figure 22: Strapi Deleted Submission Collection

- Theme - "subtemas" in "hypati67_testeconhecimentos"

tema	Text
subtema	Text
codtema	Text
codsubtema	Text

Figure 23: Strapi Theme Collection

There are three more that are explained in section 4.4 in more detail, including pictures:

- **Theme Order** - "tema_ordem" in "hypati67_qrq"
- **Theme Image** - "tema_imagens" in "hypati67_qrq"
- **Monthly Question** - "questoes_mensais" in "hypati67_qrq"

In this back-office, using relationships between collections is not particularly useful because all of them are done explicitly through queries using the *knex* *NPM* module. Configuring collection relationships are, in fact, useful when creating new applications that are not very complex in nature, which is not the case. For instance, Strapi does allow joining multiple tables but it does not allow nested selects in its query language, which is a major disadvantage when attempting to build *API* paths that require this operation. So, instead of using collection relationships in some cases and explicit queries in others, it is better to just stick with one for consistency and ease of maintenance which, in this case, is the latter.

3.5.2 Creation of the API Endpoints

One of the most time-saving features of Strapi is the automatic creation of commonly used *CRUD* routes. Whenever a new collection is inserted, the following paths are automatically added to the server:

- **GET** /{{collection}} - returns a list with every entry in the collection;
- **GET** /{{collection}}/count - returns the total number of entries in the collection;
- **GET** /{{collection}}/:id - returns the entry in the collection with the specified *id*, if it exists;
- **POST** /{{collection}} - creates a new entry in the collection;
- **PUT** /{{collection}}/:id - updates an entry in the collection with the specified *id*, if it exists;
- **DELETE** /{{collection}}/:id - removes an entry from the collection with the specified *id*, if it exists.

Strapi saves these routes in a folder, '/api/', which is automatically setup for each one of the defined collections. For example, the **School** collection is located in '/api/escola' and has the following structure:

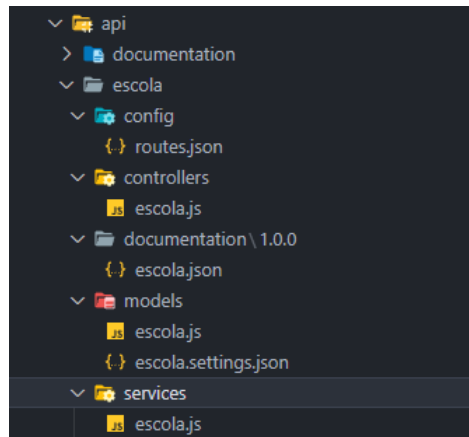


Figure 24: Strapi School Collection Folder

The `./config` sub-folder only has one *JSON* file that defines the routes that exist for that specific collection.

```

1 {
2   "routes": [
3     {
4       "method": "GET",
5       "path": "/escolas",
6       "handler": "escola.find",
7       "config": {
8         "policies": []
9       }
10    },
11    {
12      "method": "GET",
13      "path": "/escolas/count",
14      "handler": "escola.count",
15      "config": {
16        "policies": []
17      }
18    },
19    {
20      "method": "GET",
21      "path": "/escolas/:cod",
22      "handler": "escola.findOne",
23      "config": {
24        "policies": []
25      }
26    }
27    (...)
28  ]
29 }

```

This *JSON* file only has one object called "routes" that defines an array of objects and each one of those elements are the defined *API* paths. The "method" key refers to the *HTTP* method; the "path", as the name implies, is the *API* endpoint; the "handler" is the identifier for the asynchronous function that is executed when that endpoint is accessed and the "config" can have some complex security configurations that are not relevant for this application.

The '/controllers' folder also only has one JavaScript file, named after the collection, where all the aforementioned functions are defined. Peeking inside it, this is the entire file:

```

1 "use strict";
2
3 /**
4  * Read the documentation (https://strapi.io/documentation/developer-docs/latest/concepts/controllers.html#core-controllers)
5  * to customize this controller
6  */
7
8 const { sanitizeEntity } = require("strapi-utils");
9
10 module.exports = {
11   async findOne(ctx) {
12     const { cod } = ctx.params;
13
14     const entity = await strapi.services.escola.findOne({ cod });
15     return sanitizeEntity(entity, { model: strapi.models.escola });
16   }
17 };

```

So, how exactly does Strapi connect the "handler" keys to these functions? Analysing those keys reveals that they follow a pattern, a string followed by a '.' and another string where the first one defines the collection and the second one is the function in its correspondent '/controllers' file. The *escola.findOne* handler refers to the *async findOne(ctx)* function, which is visible above, for example. However, there are some functions that are not defined in this file, such as the one that "escola.find" refers to. This is another of Strapi's features, the automatically created routes have **default** functions that they use instead of copying and pasting them inside every single file inside the project. As seen above, the "findOne" function is supposedly one of these default function but it is explicitly defined in the JavaScript file, which overrides the original one for that particular collection.

The `./documentation` folder will be analyzed in chapter 6 but it refers to the documentation for that particular collection.

Inside the `./models` folder, there are two files: a JavaScript file and a *JSON* file. The first one is not used for this particular project, but essentially it allows defining additional behavior to the created collections such as relationships. The second one is the collection configuration, exactly as seen in figure 14, that contains the database connector, table name and attributes.

```

1 {
2   "kind": "collectionType",
3   "connection": "aplicacoes",
4   "collectionName": "Escolas",
5   "info": {
6     "name": "Escola",
7     "description": ""
8   },
9   "options": {
10    "increments": true,
11    "timestamps": false,
12    "populateCreatorFields": false,
13    "draftAndPublish": false
14  },
15  "attributes": {
16    "nome": {
17      "type": "string"
18    },
19    "cod": {
20      "type": "string"
21    },
22    "localidade": {
23      "type": "string"
24    },
25    "distrito": {
26      "type": "string"
27    },
28    "pais": {
29      "type": "string"
30    }
31  }
32 }

```

Lastly, the `./services` folder also contains a single file and its intended purpose is to save some auxiliary functions or variables, as a way to make the code more organized and

maintainable. This collection in particular does not contain anything inside this file, but the teachers one does, for example:

```

1  "use strict";
2
3  /**
4   * Read the documentation (https://strapi.io/documentation/v3.x/concepts/services
5   *   .html#core-services)
6   * to customize this service
7   */
8
9  const md5 = require("md5");
10
11 module.exports = {
12   validatePassword(password, hash) {
13     let code = md5(password);
14     return code === hash;
15   },
16   fetchAuthenticatedUser(id) {
17     return strapi.query("professor").findOne({ id });
18   },
19 };

```

In order to create custom endpoints, the following steps must be followed:

1. Define the function in the '/controllers' folder;
2. Create the necessary auxiliary functions in '/services' (Optional);
3. Add a new element to the "routes" list in '/config' with the correct handler link, *HTTP* method and unique path.

After repeating these steps for all the necessary *API* routes in every collection, the result is a very complete set of distinct paths that work in tandem with the front-end. The following routes are only the custom ones, since the default paths are common for every single one of them and they do the same thing.

Documentation

- **GET** /documentacao - contains the Swagger documentation for the back-end;
- **GET** /documentation - exactly the same as the previous one.

School

- **GET** /escolas/:cod - overrides the original '/escolas/:id' since it is easier to access the code that identifies the school instead of the database auto-incremented identifier.

State

This one does not contain any custom routes, this table is only useful when joined with the submissions table, therefore the basic *CRUD* routes are plenty for what is required of it.

Notification

- **GET** /notificacoes/utilizador - returns a list with all the notifications that can be seen by the user that sent the request;
- **DELETE** /notificacoes/utilizador - deletes all the notifications from the user that sent the request.

Question Backup

- **GET** /pergunta_backup - overrides the default route with a new one that lists every entry but removes questions of invalid types and with the correct text formats;
- **GET** /pergunta_backup/:cod - overrides the original '/pergunta_backup/:id' since it is easier to access the question given its code instead of the auto-incremented identifier and applies the text transformations.

Question

- **GET** /perguntas - overrides the default route with a new one that lists every entry but removes questions of invalid types and with the correct text formats;
- **GET** /perguntas/diretas - does the same as the previous one but it does not apply the text transformations;
- **GET** /perguntas/visiveis - returns a list with all the questions that are currently marked as 'visible';
- **GET** /perguntas/invisiveis - returns a list with all the questions that are currently marked as 'hidden';
- **GET** /perguntas/exames - returns a list of all the distinct tests or exams in the questions table;

- **GET** /perguntas/:cod - overrides the default '/perguntas/:id' since it is easier to access the question given its code instead of the auto-incremented identifier and applies the text transformations.
- **POST** /perguntas/converter - converts a question object to a new one with the correct text transformations;
- **PUT** /perguntas/editar/cod - whenever a question needs to be edited, this endpoint deletes the question with the given code from the questions table, creates a new submission with it on the quarantine table, a new pending state and a new notification;
- **DELETE** /perguntas/:cod - deletes the question with the given code from the questions, quarantine, state and notifications tables.

Hidden Question

- **GET** /perguntas_escondidas/:cod - overrides the default route with a new one since it is easier to access the question given its code instead of the auto-incremented identifier;
- **POST** /perguntas_escondidas/expor/:cod - marks the question with the given code as 'visible';
- **POST** /perguntas_escondidas/:cod - marks the question with the given code as 'invisible'.

Teacher

- **GET** /professores/:cod - overrides the default route with a new one since it is easier to access the teacher given their code instead of the auto-incremented identifier;

Quarantine

- **GET** /quarentena - overrides the default route with a new one that returns a list with all the submitted questions joined with their current state and correct text transformations;
- **GET** /quarentena/submissoes - returns a list with all the current pending submissions;
- **GET** /quarentena/estatisticas - returns an object with statistical data about the submissions for the dashboard;
- **GET** /quarentena/:cod - overrides the default '/quarentena/:id' since it is easier to access the submission given its code instead of the auto-incremented identifier, applies the text transformations and joins it with its current state;

- **POST** /quarentena - overrides the default route and creates a new submission, pending state and notification. The code for the question is automatically calculated:
 1. For the submitted question's theme and sub-theme, calculate the maximum of the inserted codes from all the questions;
 2. If that maximum exists, increment it by 1 and check if it exists in the database (which usually does not, but there is a chance it might);
 3. Otherwise, assume that the maximum is given by the number resulting from appending the theme and sub-theme together as a string and adding 6 zeroes after it. For example, for a theme with code '125' and sub-theme with code '1251', the resulting code would be '1251251000000';
 4. Check if the resulting code exists in the database;
 5. If it exists, increment it by 2^i where i is the current iteration counter and go back to 4. This avoids collisions when there are many sequential codes in a short interval;
 6. If it does not exist, the code is valid and can be used.
- **PUT** /quarentena/aceitar/:cod - whenever a submission is accepted, this endpoint sets its state to 'Accepted', adds the question to the questions table and creates a new notification with the result;
- **PUT** /quarentena/recusar/:cod - whenever a submission is rejected, this endpoint sets its state to 'Rejected' and creates a new notification with the result;
- **PUT** /quarentena/editar/:cod - updates a 'Pending' or 'Rejected' submission with the new question data, sets its state to 'Pending', creates a new code with the rules above if the themes or sub-themes have changed, deletes all the notifications related to the previous submission and creates a new one;
- **DELETE** /quarentena/:cod - deletes the submission with the given code from the state, quarantine and notifications tables and adds it to the submission backups table.

Monthly Question

- **GET** /questoes_mensais/mes - returns the monthly questions for the current month, divided by categories;
- **GET** /questoes_mensais/todas - returns the history of monthly questions, divided by year, months and categories;

- **POST** /questoes_mensais - adds a set of questions divided by categories to the monthly questions for the current month, adds them to the hidden questions table and removes all the other ones from the previous edition from it.

Deleted Submission

- **GET** /submissoes_eliminadas - overrides the default route with a new one that lists every entry with the correct text formats;
- **GET** /submissoes_eliminadas/:cod - overrides the original 'submissoes_eliminadas/:id' since it is easier to access the submission given its question's code instead of the auto-incremented identifier and applies the text transformations.

Theme Image

- **PUT** /tema_imagens/tema/:cod - creates a new entry with the code of the theme and the URL of the picture for it;

Theme Order

- **POST** /tema_ordem - overrides the default route with a new one that removes all the previous rows in the table and adds new ones ordered in the request body;

Theme

- **GET** /temas - returns the list of all the themes and sub-themes grouped by the name of the theme itself;
- **GET** /temas/agrupar - does the same as the previous one but uses the code of the theme instead;
- **GET** /temas/agruparOrdem - does the same as the previous one but groups them according to the order saved in the "tema_ordem" table;
- **GET** /temas/agruparImagens - returns a list with the themes group with their respective image saved in the "tema_imagens" table.

3.5.3 Authentication and JWTs

There are many ways that allow a program or application to identify which user sent every request they receive. When talking about a **stateless** web application, the most popular way is via authentication tokens, which are unique to every user and each request that requires

authentication comes signed with it, whether it is in the headers, body, query string or the path itself. This strategy is typically referred to as **token-based authentication** (Swoop, 2020).

A **token** consists of three basic elements:

1. **header** - defines the type and signing algorithm used;
2. **payload** - contains the claims, i.e, information about the user and other metadata;
3. **signature** - verifies the authenticity of the message and can identify the sender.

A *JWT* is an open-standard (RFC 7519) token-based authentication strategy by the means of *JSON* objects. This particular implementation, when signed, has the format "xxxxx.yyyyyy.zzzzzz", where *x*, *y* and *z* refer to the Base64URL encoded strings from the enumerated elements above as *JSON* objects, respectively.

The signature, however, has a particular way of doing this conversion, which allows the application to easily discard requests if the message was altered and, if it was signed with a private key, can also verify the user's identity. For example, if the chosen algorithm is HMACSHA256, the Base64URL string would be given by the following formula:

```
HMACSHA256(
2  base64UrlEncode(header) + "." +
   base64UrlEncode(payload) ,
4  secret)
```

Simply calculating it again when the request is received can prove whether or not the message was altered or if it came from another user. This **secret** should be an arbitrarily long word kept private by the developer(s) in order to ensure that the application is secured, since it is the only field that is impossible to replicate by an intruder.

Another way this particular strategy uses on detecting invalid requests is checking the **expiry dates** on the tokens themselves.

Thankfully, Strapi has built-in compatibility with *JWTs*, where some configurations can be changed at will. For example, swapping the token secret is as straightforward as changing a single word in the file located at '/extensions/users-permissions/config/jwt.js':

```
module.exports = {
2   jwtSecret: process.env.JWT_SECRET || '${SECRET}',
   jwt: {
4     expiresIn: "8h",
   }
6 };
```

Naturally, the secret is redacted in order to comply with *Hypatiamat*'s privacy policy. The 'expiresIn' field defines that, from the moment the token is signed, it is valid for the following 8 hours, which is typically the duration of a workday and the teachers will never use the platform longer than that. The reason why this lifespan can not be longer is because of security concerns, it is less likely that an intruder would gain access to the user's token in 8 hours than 1 month, for example. Even if they get access to an older token, it is not valid anymore so it poses no security threat. The downside is it forces the user to authenticate every day, which can be mitigated easily by having automating sign-ins with the locally stored identifier and password or other similar strategies.

In order to submit an authentication request, a user has to use one of Strapi's default routes that are unrelated to the collections, namely '/auth/local', which refers to the local-authentication *JWT* method described in this chapter. The process is relatively simple for the user, they only have to submit a *POST HTTP* request to that route with the identifier and password in its body where the identifier in this particular case is the e-mail or their unique code (see figure 8). For example:

```
{
  2   "identifier": 'hprof1', // or hypatiamate@gmail.com
     "password": "password"
  4 }
```

Once the request is received in the server, assuming that everything goes right and the request is valid and the user is authenticated correctly, it returns them a *JSON* object with two main entries. The first one is the token so they can use it in future requests and the second one is the information of that particular user.

```
{
  2   "token": {
     "jwt": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1IjoiYmVudGlnbyI6Imhwc29mMSIsImVtYWlsIjoiaHlwYXRpYW1hdGVAZ2Z1haWwY29tIiwiaXNjb2xhIjoiaHlwYXRpYTAxIiwidHlwZSI6NTAsInRwY1R5cGU0Ijwcm9mZXNzb3IiLCJhZ3JlcGFtZW50byI6IkvzY29sYSBkbyBIeXBhdGlnbWF0LCBCcmFnYSJ9LCJpYXQ0IjoiE2NjI1NzQ0MzcsImV4cCI6MTY2MjYwNjgzN30.sPYfN9zStUhev6k7FuzW0LhoG6q8n_7XncAjl16n_1c",
     "expiresIn": 28800
  4   },
  6   "user": {
     "id": 241,
     "codigo": "hprof1",
     "nome": "Professor Hypatia - Testes",
     "escola": "hypatia01",
  8   "email": "hypatiamate@gmail.com",
  10 }
```

```

12     "confirmacao": 1,
13     "premium": 5,
14     "validade": "2025-08-31",
15     "socionum": 0,
16     "projeto": "1",
17     "type": "professor"
18   }
19 }

```

After the authentication process is completed by the teacher, they are now free to use the endpoints as they wish, as they now have complete control on the databases and their contents.

3.5.4 Protecting the API

By having the *JWT* approach for authenticating users, it is a relatively simple process of blocking routes for anyone that is not a teacher, has an invalid token or does not have one at all.

In a typical Node.js application, authentication is done using middleware that intercept the endpoints themselves and block the user from accessing them if they do not meet the established criteria. In order to avoid inserting the middleware on every created route, which causes code repetition, Strapi has a built-in functionality that allows linking **roles** to every endpoint by using their handler function (see chapter 3.5.2).

This feature is called "Roles & Permissions" which comes installed by default on every new Strapi instance. It is very simple to complete the linking once the routes themselves are created, since it is done by an intuitive and efficient interface. A **role** is just an internal identifier for Strapi to categorize the multiple users that may use its *API* services and by default only two exist, namely 'Authenticated' and 'Public'.

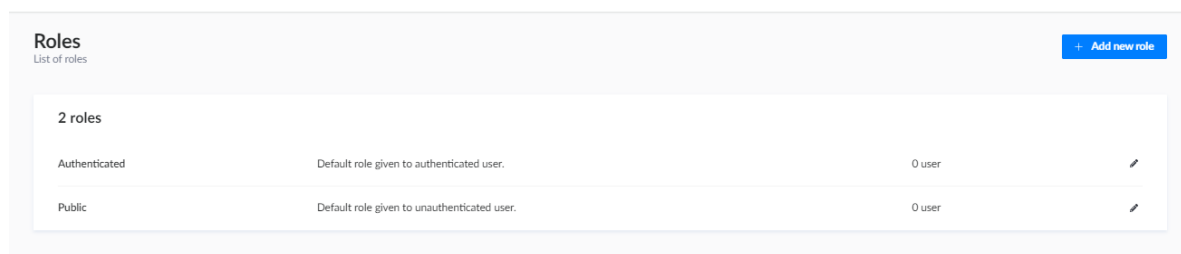


Figure 25: Strapi Default Roles

Evidently, these are just the ones that are shipped with Strapi, an arbitrary number of other roles can be created. However, in this particular application, it is only necessary to know whether the user is authenticated or not, there is no need to differentiate between authenticated users since they are all teachers that belong to the same group.

Considering that the 'Public' role will only be used by anyone that has no token, they must not have access to any of the routes since an ill-intentioned intruder may cause irreversible damage to the database server. The only exception is the documentation endpoint, which must be available for any future developer that may use the *API* server.

Permissions
Only actions bound by a route are listed below.

APPLICATION
Define all allowed actions for the application plugin.

DOCUMENTATION Select all

index

ESCOLA Select all

count create

delete find

findone update

ESTADO Select all

count create

delete find update

NOTIFICACOES Select all

Figure 26: Strapi 'Public' role and its linked handlers (Example)

In contrast, the 'Authenticated' role must have every box ticked so that the teachers can effectively use the platform. Were these boxes unticked, everytime they accessed the endpoint(s) that execute the asynchronous handler, they will be given a **403 - Forbidden HTTP** status response.

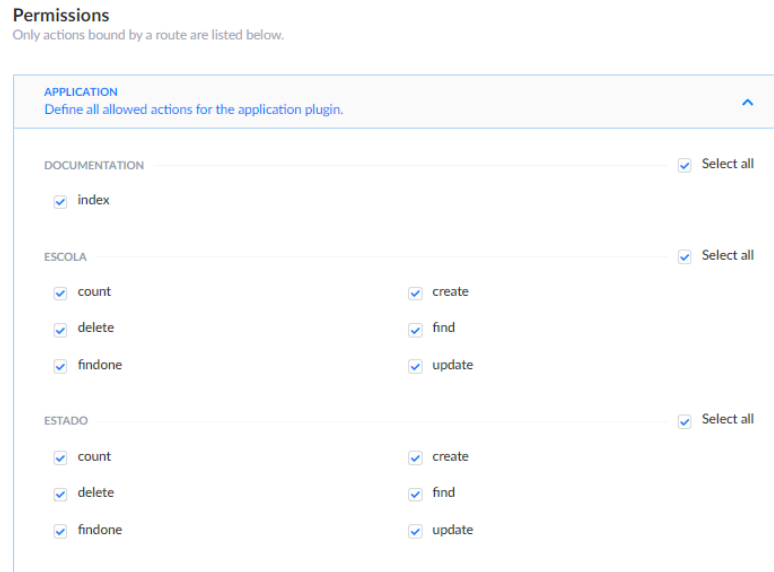


Figure 27: Strapi 'Authenticated' role and its linked handlers (Example)

The teachers will not directly use the *API* server, as this is something typically not human-readable. It is the front-end's responsibility to consume these endpoints and allow the user to have a more dynamic and human-friendly approach to them.

3.6 FRONT-END DEVELOPMENT

The selected technology is Vue.js, as seen in chapter 3.3, due to its numerous advantages both for the developer, the end-user and the *Hypatiamat* servers.

Vue is a complex framework aimed at building responsive and reactive user interfaces, via a *SPA* approach. In order to mimic the behaviour of entering different pages inside the application, it uses a built-in router that injects the *HTML* content for that specific page, which is defined inside components. These are files that contain three sections:

- **template** - an *HTML*-based syntax that defines the page's layout with multiple quality of life features such as two-way data-binding, conditional rendering, reactivity and many other recent technologies;
- **script** - will be executed as an ES6 module in run-time and contains all behaviour regarding the page's reactivity, computing and importing other components (children);
- **style** - contains the *CSS* styles for that page (or its children).

Because these components are written in separate files, it is possible to re-use them in other different applications as simply as pasting them inside the project's folder without

having to rewrite their content, if they do not include any component dependency, which will be explored in subsection 3.6.1.

Henceforth, during this chapter, it is implicit that whenever any view is mentioned and it contains data of any kind, it should be assumed it came from the back-end *API* server mentioned in the previous section, as the 'script' module inside each component can have *HTTP* requests executed via JavaScript and their response data can be saved using local variables or browser storage.

3.6.1 Initial Setup

When compared to the back-end server, the front-end was had a quick and straightforward process to get its development started due to it being a completely isolated entity that only interacts with the *API* endpoints using *HTTP* requests.

As seen in section 3.1.2, the interfaces need to be consistent with João Vieira's *TPC* management back-office. They were also built using Vue.js and with his and *Hypatiamat*'s permission, it is possible to re-use the most important ones for this project aswell, namely the authentication and main layout components.

This modularity brings an effective way of building cross-platform components, which will be great for any future application that needs to be developed following these guidelines, as they do not need to be rebuilt and mimicked from the ground up and avoids the risk of not exactly matching the original in both behaviour and appearance.

The next step is to construct the views themselves in order to complete the requirements set at the beginning of the project, which does not require any initial setup.

3.6.2 Creation of the Interfaces

The interfaces are effectively where everything is linked together, the business logic, persistence and view layers work together to allow the back-office to have its requirements fulfilled.

There are 10 different pages in total, each using a different Vue component, and they suffered an incremental process where features and design were tweaked in order to achieve a good work environment for the teachers, where the priority was keeping them simple and intuitive.

A logical first step is mocking up the interfaces in order to provide *Hypatiamat* with a concise idea of what the end application would look like, they are not meant to be the final version but rather a guide to what should be expected of it. These prototypes also ensure that no time is wasted on developing an interface in real-time since they may not work as envisioned.

If the idea and design is approved, it will then pass to its development phase in code. In the following sub-sections, the final interfaces will be compared with their mock-ups in order to show the process behind creating a finished functional product.

On a side note, all these interfaces were prototyped using Justinmind, a popular tool that contains many readily available components such as buttons, forms, text inputs and geometric shapes for use in designing interfaces for any screen size or pixel density.

Authentication

This is one of the few interfaces that did not need to be prototyped first before developing, because the Vue component itself is being reused from João Vieira's project, who conducted their research on the matter and concluded this would be the final design to be used in the application.

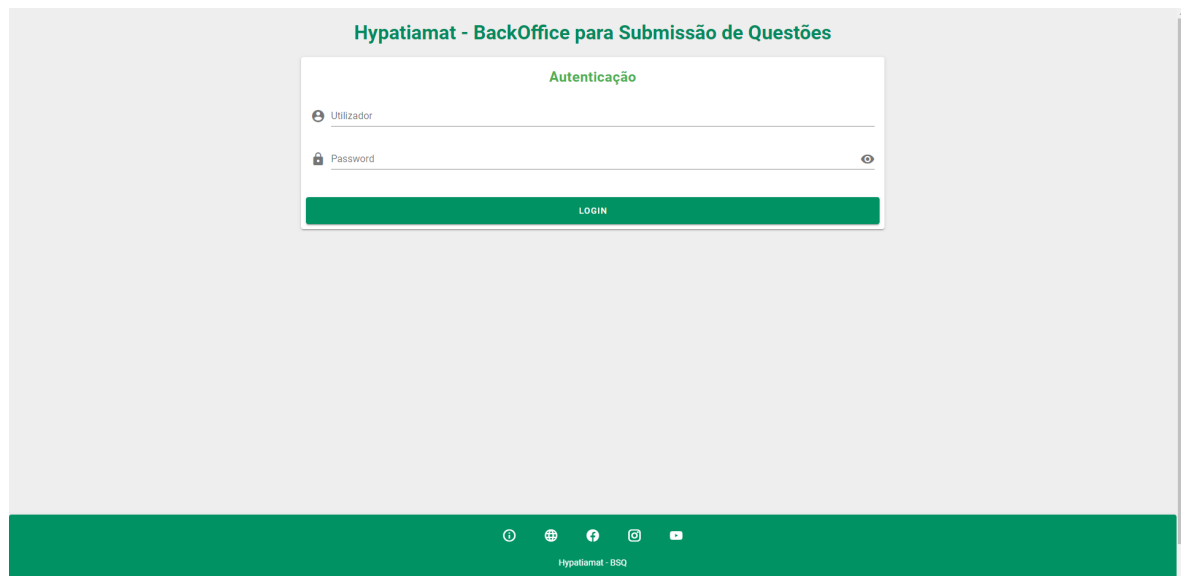


Figure 28: Authentication Interface

This is a very simple interface and it has two main components: a body and a footer, it does not have the navigation bar on the left side yet because the authentication process has not been completed yet.

The body has a simple authentication form where the user inputs their username or e-mail and password and, if these values are validated by the back-end, they are successfully logged in, granting them access to the back-office.

The footer contains the links to *Hypatiamat's* social media platforms and a small information modal that contains some credits to the author and supervisors of the project.

Dashboard

The dashboard, on the other hand, suffered some changes when compared to its mock-up (figure 118).

The first major change was the inclusion of two graphs using the “highcharts” *NPM* module:

1. **Submission Distribution** - uses a pie chart to represent the total number of submissions and the users that created them;
2. **Current Submission State** - semi-circle pie chart that represents the percentage of the user’s submissions with a given state (pending, accepted or rejected).

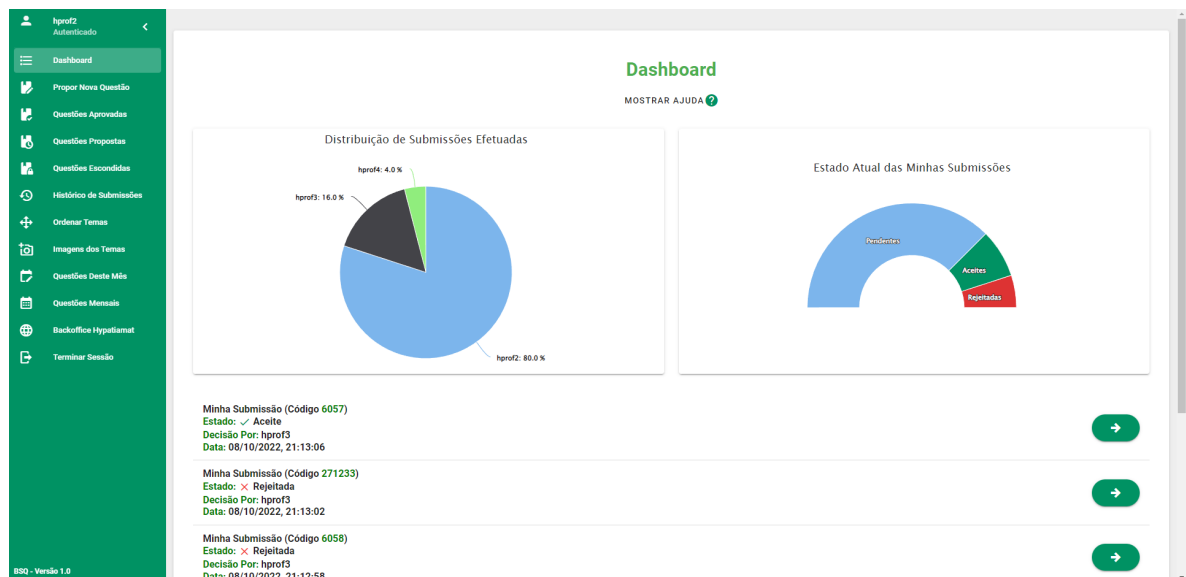


Figure 29: Dashboard Interface

The second was on the notifications themselves, which were simplified to align the text to the left side and including a redirection button on the other side, which makes it more visually appealing than the prototype.

Some additional information such as the submission’s state was also added and the title was reworked to better represent the two different notification types.

Submission Form

This is the most complex interface in the entire application because it contains many details and it suffered the most changes when compared to its prototype (figure 119).

It was clear from the start that there needed to exist a form of some kind in order to allow the teacher to fill the fields in their entirety with the correct information, however, there exist

some steps that have an extra complexity layer in order to fulfill *Hypatiamat's* needs and improvement requests.

Instead of having a single-page form, it has been divided in segments using a stepper component (Vuetify).

In the first step, the form has been improved visually by using titles in order to remove the cluttered and clumped together fields that the prototype has.

Figure 30: Form Submission Interface (1st step)

The input fields equate to every column on the database, except the questions and answers themselves. Due to their large variety of different types, there have to be some auxiliary interface elements in order to complete an effective algorithm that takes these elements and figures out the correct types and underlying columns.

These auxiliary elements are the type and sub-type selectors in the section for filling the question information, which change the answers input fields accordingly.

When the question is a multiple choice, the following changes will occur in the interface according to the selected sub-type:

- Text - since the number of answers can range from a minimum of 3 to a maximum of 6, 3 input text fields are created by default. The last input will always contain two buttons, one that increments the number of fields up to 6 and one that removes a field, down to 3. It is very intuitive to use and allows the teachers to rapidly create or remove answers without having to leave the one they are currently working on.

The interface is titled "Questão" in green. It features two dropdown menus: "Tipo" set to "Escolha Múltipla" and "Subtipo" set to "Respostas de Texto". Below these is a text input field labeled "Questão". The "Respostas" section contains three text input fields: "Resposta", "Alternativa Incorreta 1", and "Alternativa Incorreta 2". To the right of the last field are two circular buttons, one green with a plus sign and one grey with a minus sign. A "CONTINUAR" button is at the bottom.

Figure 31: Form Submission Interface (1st step) - Text Multiple Choice

- Image - the range displayed above does not apply here, only 4 answers are allowed and they must all be images. Therefore, each one of the answers must be a file input instead of text, allowing the users to submit images from their own computer and previewing them using the eye icon on the right side of each one of these fields.

The interface is titled "Questão" in green. It features two dropdown menus: "Tipo" set to "Escolha Múltipla" and "Subtipo" set to "Respostas de Imagens". Below these is a text input field labeled "Questão". The "Respostas" section contains four file input fields: "Resposta", "Alternativa Incorreta 1", "Alternativa Incorreta 2", and "Alternativa Incorreta 3". Each field has a camera icon on the left and an eye icon on the right. A "CONTINUAR" button is at the bottom.

Figure 32: Form Submission Interface (1st step) - Image Multiple Choice

Regarding the open-ended questions, they have 6 different types that dictate their 'auxiliar' column (more information in chapter 3.2.2) and thus need to be explicitly defined:

- There only exists one answer and that is the correct one - this would give a value of 1 to the 'auxiliar' column. Since there is only one accepted answer, there only needs to exist a single text input field for it and the optional unit of measure.

The screenshot shows a form titled "Questão" with two dropdown menus: "Tipo" set to "Resposta Aberta" and "Subtipo" set to "1 Resposta Correta". Below these is a text input field labeled "Questão". Under the "Respostas" section, there is a single text input field labeled "Resposta" and a smaller input field labeled "Unidade". A "CONTINUAR" button is at the bottom.

Figure 33: Form Submission Interface (1st step) - Open Ended, 'auxiliar' 1

- There only exist two answers and both are correct - same as before but creates only two input fields, giving the 'auxiliar' column the value of 1000.

The screenshot shows a form titled "Questão" with "Tipo" set to "Resposta Aberta" and "Subtipo" set to "2 Respostas Corretas". Below is the "Questão" text input field. Under "Respostas", there are two text input fields: "Resposta" and "Alternativa", plus a "Unidade" field. A "CONTINUAR" button is at the bottom.

Figure 34: Form Submission Interface (1st step) - Open Ended, 'auxiliar' 1000

- Multiple answers and all are correct - this would give the 'auxiliar' field a value of 0. It works similarly to the text multiple choice answers, reusing the increment/decrement buttons but this time for a minimum of 1 answer, instead of 3.

Questão

Tipo: Resposta Aberta

Subtipo: Várias Respostas Corretas

Questão

Respostas

Resposta

Alternativa 1

Unidade

CONTINUAR

Figure 35: Form Submission Interface (1st step) - Open Ended, 'auxiliar' 0

- Equivalent fractions - 'auxiliar' is 22 in this case. There are two input fields, one for the numerator and one for the denominator, along with the optional unit of measure.

Questão

Tipo: Resposta Aberta

Subtipo: Fração Redutível

Questão

Respostas

Numerador

Denominador

Unidade

CONTINUAR

Figure 36: Form Submission Interface (1st step) - Open Ended, 'auxiliar' 22

- Irreducible fraction - the 'auxiliar' column is 2 and it has the same fields as the previous one.

Questão

Tipo: Resposta Aberta

Subtipo: Fração Irredutível

Questão

Respostas

Numerador Irredutível

Denominador Irredutível

Unidade

CONTINUAR

Figure 37: Form Submission Interface (1st step) - Open Ended, 'auxiliar' 2

- Protractor required - the 'auxiliar' column is 10 and it indicates the need of a protractor, requiring two text input fields that contain the lower and upper ranges of the accepted values.

Questão

Tipo: Resposta Aberta

Subtipo: Utilizar Transferidor

Questão

Respostas

Limite Inferior (°)

Limite Superior (°)

CONTINUAR

Figure 38: Form Submission Interface (1st step) - Open Ended, 'auxiliar' 10

The true and false questions are very simple, as they only need two input fields for the true and false or its equivalent expressions.

Questão

Tipo: Verdadeiro ou Falso Subtipo: Subtipo

Questão

Respostas

Resposta Correta

Resposta Incorreta

CONTINUAR

Figure 39: Form Submission Interface (1st step) - True or False

Lastly, the symmetry questions had a special treatment due to their unique answer format. These types of questions essentially have a symmetry axis on a grid which can be either vertical or horizontal, and they can start on either side of that axis.

Therefore, it is possible to conclude that it is both easier and more efficient to manipulate the grid itself to form the symmetry, instead of a text input section with a formatted string that would be converted into a grid preview.

As stated before, there are two types of possible symmetry:

- Vertical

Questão

Tipo: Simetria Subtipo: Eixo Vertical - Esquerda

Questão

Respostas

CONTINUAR

Figure 40: Form Submission Interface (1st step) - Vertical Symmetry

- Horizontal

Questão

Tipo: Subtipo:

Questão

Respostas

CONTINUAR

Figure 41: Form Submission Interface (1st step) - Horizontal Symmetry

The orange grid elements are the ones the teachers interact with, the green ones indicate the correct answer for the selected symmetry and they are updated in real time. This ensures that the teachers can both input the symmetry and visualize what its corresponding answer would be, which is very efficient and less annoying to work with.

After every field is completed correctly, it is possible to advance to step number 2. As soon as it is entered, the user is prompted with the question of whether or not they want to add a reference image to the question.

Escrever Questão — 2 Adicionar Imagem

Adicionar imagem?

CONTINUAR VOLTAR

Figure 42: Form Submission Interface (2nd step) - Image Query

If they check the box, a new menu will appear below it that will allow them to pick between a number of predefined templates, which will be useful later in the next step.

Propor Nova Questão
MOSTRAR AJUDA ?

Escrever Questão — 2 Adicionar Imagem — 3 Editar Imagem — 4 Adicionar Resolução — 5 Editar Resolução — 6 Pré-Visualizar — 7 Confirmar

Adicionar imagem?

Selecione um dos seguintes *templates*.

Caso pretenda utilizar uma imagem do seu computador, selecione a opção "Em Branco" e, de seguida, selecione **Carregar**.

Em Branco

Pensar

Gráfico

Grelha 1

Grelha 2

CONTINUAR VOLTAR

Figure 43: Form Submission Interface (2nd step) - Image Query Templates

If the box was not selected, the next step will be skipped to number 4. However, if they do check it and they pick a template, they will be redirected to step number 3. In this step, the user has an image editor opened with that selected image by default.

It is called ToastUI Image Editor, an open source editor that contains a lot of features such as the insertion of geometric shapes, text, filters and drawing tools.

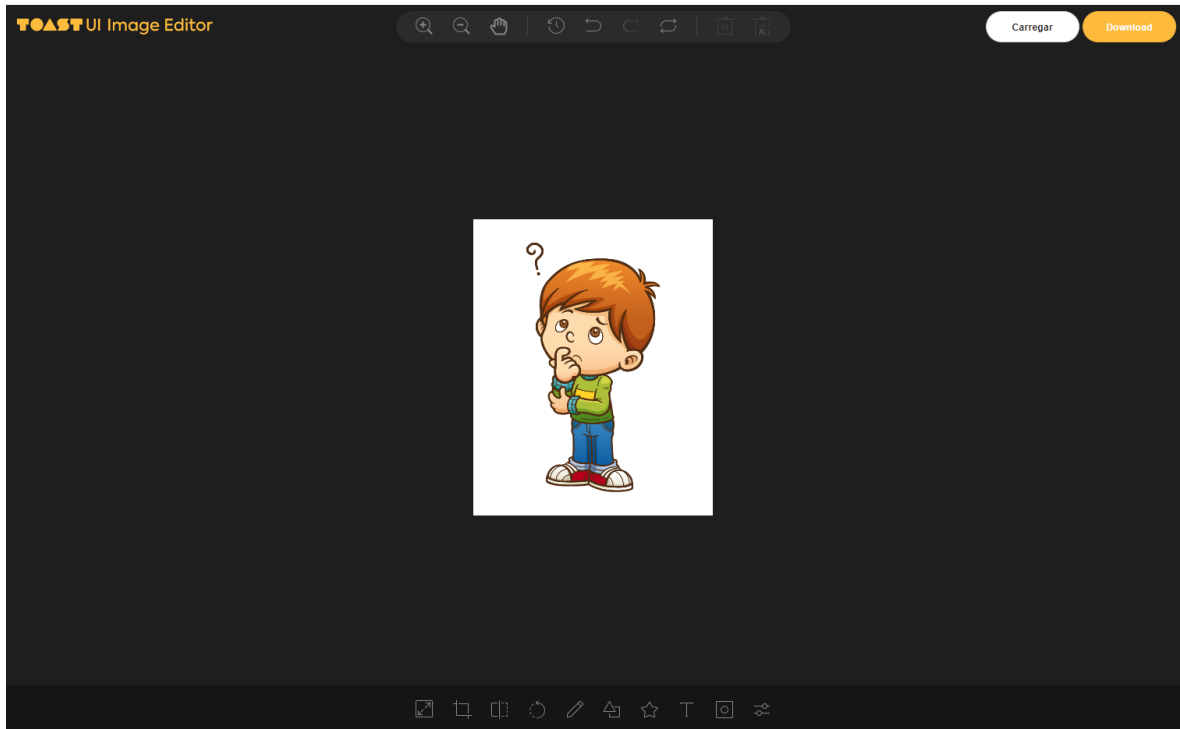


Figure 44: Form Submission Interface (3rd step) - ToastUI Image Editor

This is not intended to replace Adobe Photoshop or any other software alternative, as they are much more powerful than this one, but rather a convenient way for the teachers to edit images quickly without having to do it outside the application, since a lot of the times they need to include an image in the question, they only need these basic features in addition to the templates.

When they continue to step 4, they will be greeted by a similar screen to step 2, except this time it refers to the resolution image.

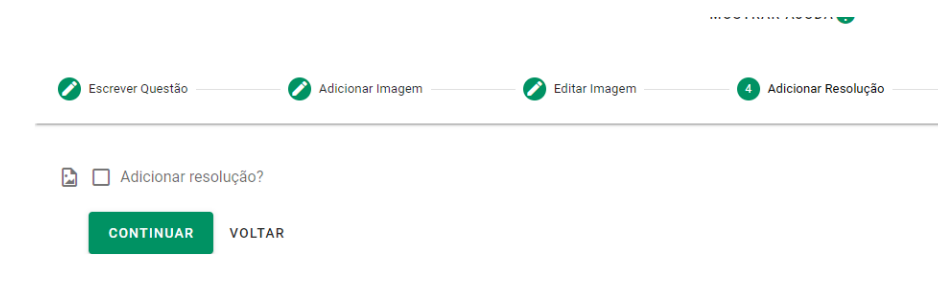


Figure 45: Form Submission Interface (4th step) - Resolution Query

The templates are also different and include the ones that appeared more frequently in other questions.

Figure 46: Form Submission Interface (4th step) - Resolution Query Templates

Similarly to before, if the user checks the box, they will be redirected to step number 5 with the same image editor, this time with the resolution template opened. If they do not check it, they will instead skip straight ahead to step number 6.

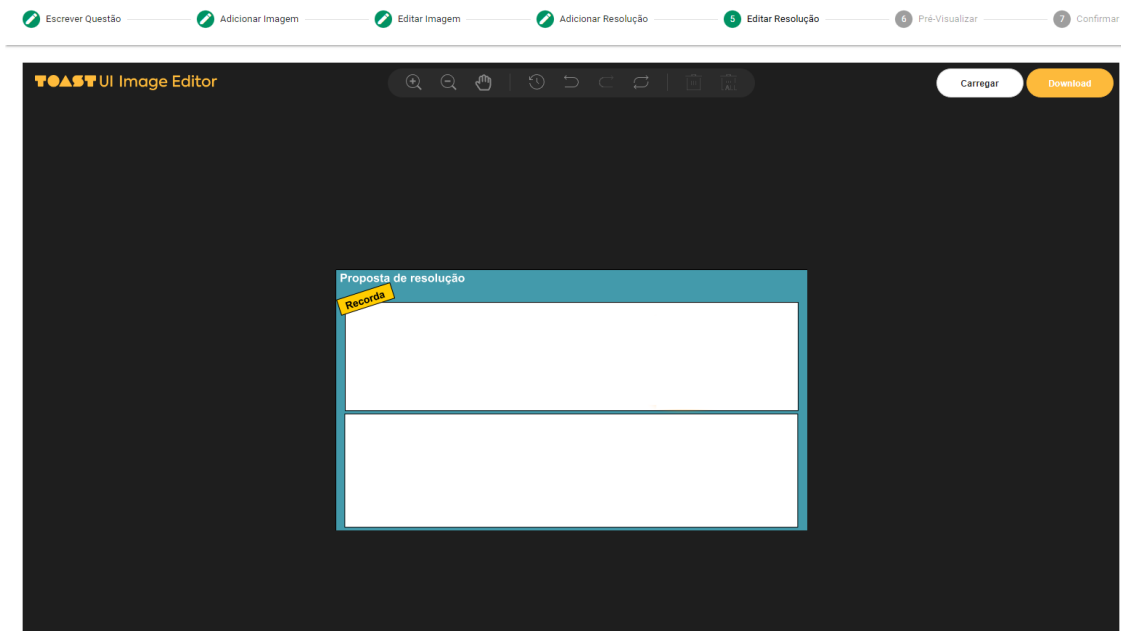


Figure 47: Form Submission Interface (5th step) - Toast UI Image Editor

The 6th step is simply a way for the user to preview how the question looks at the end of the process, so that they can visually verify whether or not they need to change something

in previous steps. If they do find a mistake or need to include some more information, they can always backtrack and change those fields.

Figure 48: Form Submission Interface (6th step) - Question Preview

The preview's appearance is intended to be as close as possible to the other application's question visualization interface (this will be detailed further in the next interface), which is why the resolution image is hidden by default, i.e, to simulate its behaviour in the final product. The teacher can preview it by clicking the 'Resolução' button.

If everything is correct, they can then continue to the 7th step, which is also the last one in the entire process. This is simply a formal check to ensure that the user wants to submit the question for peer evaluation, otherwise the submission is not registered.

Figure 49: Form Submission Interface (7th step) - Confirmation

If they confirm that they want to submit the question, they get an alert that a new submission was created and they can now freely leave the page.

Naturally, this entire process is very similar in **editing** submissions or questions.

The first difference is in the first step, where the fields come automatically filled with the question's information, because it would make no sense to force the user to input the fields manually when they are already persisted in the database.

The screenshot shows a form titled "Form Submission Interface (Edit) - 1st Step". It is divided into three main sections:

- Meta-Dados:** Contains fields for "Tema" (Proporcionalidade Inversa), "Subtema" (Grandezas Inversamente proporcionais), "Nível" (1), and "Ano" (9). There is also a checkbox for "É questão de exame?" and a dropdown menu for "Exame".
- Questão:** Contains a "Tipo" dropdown (Escolha Múltipla) and a "Subtipo" dropdown (Respostas de Imagens). Below this is a text area for the question: "Qual a equação que traduz a seguinte relação (expressando y em função de x com k constante): y é inversamente proporcional a x ."
- Respostas:** Contains a list of five answer options, each with a unique ID and a delete icon:
 - Resposta: inv_RP_1_0_0ea1e9a6f2
 - Alternativa incorreta 1: inv_RP_2_0_992a99781b
 - Alternativa incorreta 2: inv_RP_4_0_4ff9a9ad6b
 - Alternativa incorreta 3: inv_RP_5_0_5bdf7f2ae0

A green "CONTINUAR" button is located at the bottom left of the form.

Figure 50: Form Submission Interface (Edit) - 1st Step

The next difference is on the steps 2 and 4. Instead of giving the user only one option, which is whether to use a template or not, they now have three different options:

The screenshot shows the "Form Submission Interface (Edit) - 2nd and 4th Steps". At the top, there is a progress bar with four steps: "Escrever Questão", "Adicionar Imagem", "Editar Imagem", and "Adicionar Resolução" (which is currently selected and highlighted in blue). Below the progress bar, there is a heading "Editar Submissão" and a link "MOSTRAR AJUDA?".

The main content area contains the following text and options:

Esta questão tem, atualmente, uma resolução associada.

Caso não pretender efetuar nenhuma alteração, não selecione nenhuma das opções e clique em "Continuar".

Caso contrário, selecione uma das opções abaixo:

- Editar Resolução Atual
- Remover Resolução Atual
- Adicionar Nova Resolução

At the bottom, there are two buttons: a green "CONTINUAR" button and a "VOLTAR" button.

Figure 51: Form Submission Interface (Edit) - 2nd and 4th Steps

This selection menu replaces the first one if the question/submission does not contain the reference or resolution image, otherwise it just follows the same flow as before.

The first option allows the user to edit the current saved image, which will open the ToastUI editor in step 3 or 5 with it; the second one, if checked, completely removes the image from the question or submission being edited and the third one, if selected, opens the template menu for that particular step and opens ToastUI with it in the following step.

If no option is selected, the image is kept as it is with no changes.

Approved Questions

This one suffered very little changes when compared to its mock-up (figure 120).

The main difference between the two is the way the questions list is presented, where in the final version it ended up becoming a table for a number of reasons. First, it allows the user to display the items in the table in ascending or descending order of the columns. Second, it is more organized and more intuitive to understand what each field represents. Third, it allows a larger amount of information to be displayed in a more compact manner.

The screenshot shows the 'Questões Aprovadas' interface. On the left is a green sidebar menu with options like 'Dashboard', 'Propor Nova Questão', 'Questões Aprovadas', 'Questões Propostas', 'Questões Escondidas', 'Histórico de Submissões', 'Ordenar Temas', 'Imagens dos Temas', 'Questões Deste Mês', 'Questões Mensais', 'Backoffice Hypatiaam', and 'Terminar Sessão'. The main content area has a title 'Questões Aprovadas' and a 'MOSTRAR AJUDA' button. Below the title are several filter fields: 'Código', 'Tipo', 'Subtipo', 'Nível', 'Ano', 'Exame', 'Tema', 'Subtema', and 'Questão'. A table below the filters displays a list of approved questions with columns for 'Código', 'Tipo', 'Questão', 'Exame', 'Tema', 'Subtema', 'Nível', and 'Ano'. The table contains five rows of data.

Código	Tipo	Questão	Exame	Tema	Subtema	Nível	Ano
21101	Escolha Múltipla	Qual é o perímetro da figura? Escolha a opção correta.	Mathematics - 2009 School Certificate Test - Austrália(Board of Studies NSW)	Perímetros	Polígonos	2	5
21102	Escolha Múltipla	Foram cortados quartos de círculos com o mesmo raio. Qual das figuras tem o maior perímetro?	Mathematics - 2010 School Certificate Test - Austrália(Board of Studies NSW)	Perímetros	Círculos e polígonos inscritos em circunferências	3	5
21103	Escolha Múltipla	O jardim da Ema tem a forma de dois semicírculos (ver figura). Assinala, entre as opções seguintes, aquela que melhor aproxima o perímetro do jardim, em metros.	Mathematics - 2009 School Certificate Test - Austrália(Board of Studies NSW)	Perímetros	Círculos e polígonos inscritos em circunferências	3	8
21104	Resposta Aberta	O perímetro de um triângulo equilátero é igual a 90 cm. Qual é o comprimento de cada um dos lados?	Mathematics - 2010 School Certificate Test - Austrália(Board of Studies NSW)	Perímetros	Polígonos	1	5
21105	Resposta Aberta	O perímetro de um terreno é igual a 89 m (ver figura). Qual é o comprimento da parte curva da fronteira do terreno?	Mathematics - 2010 School Certificate Test - Austrália(Board of Studies NSW)	Perímetros	Outras figuras	2	5

Figure 52: Approved Questions Interface

Instead of hiding the filters, this version leaves them fully visible for the sake of convenience since it would be counterproductive to hide them from the teachers when it is expected that they will use them frequently.

The filters are also executed in real time and simultaneously, allowing the teachers to get specific questions that fit their criteria quickly and efficiently.

Each of the table's items can be expanded, showing a preview for the selected questions. Its structure is transversal to every one of them, containing only four main elements:

- Question statement;
- Answer(s);
- Reference Image (optional);
- Resolution Image (optional).

The only difference that may appear is in the answers component, since they depend on the question's type, just like the previous interface.

The multiple choice questions have two types, and they change the preview accordingly:

- Text - the answers ranging from 3 to 6 are displayed in sequence. The first one is considered the correct one, which the preview represents with a green check mark after the answer itself.

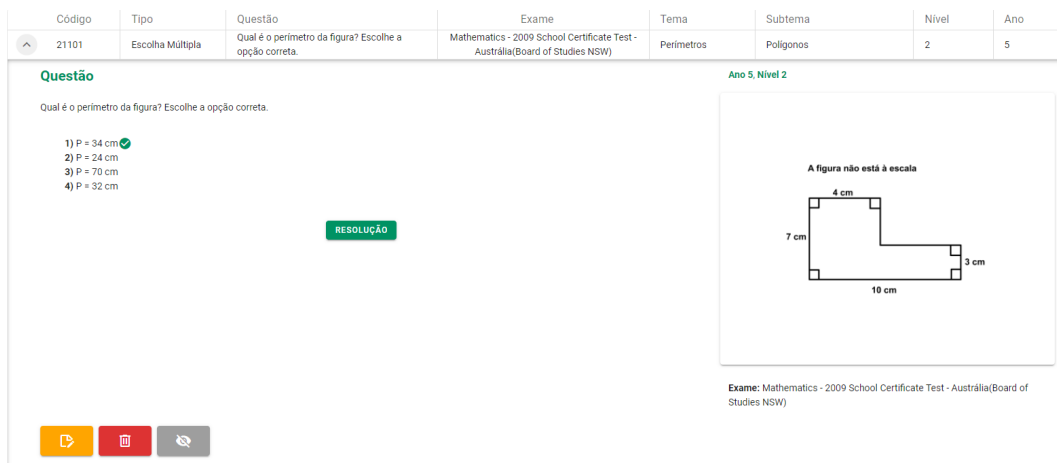


Figure 53: Approved Questions (Preview) Interface - Text Multiple Choice

- Image - similar to the text answers, except images are displayed instead.

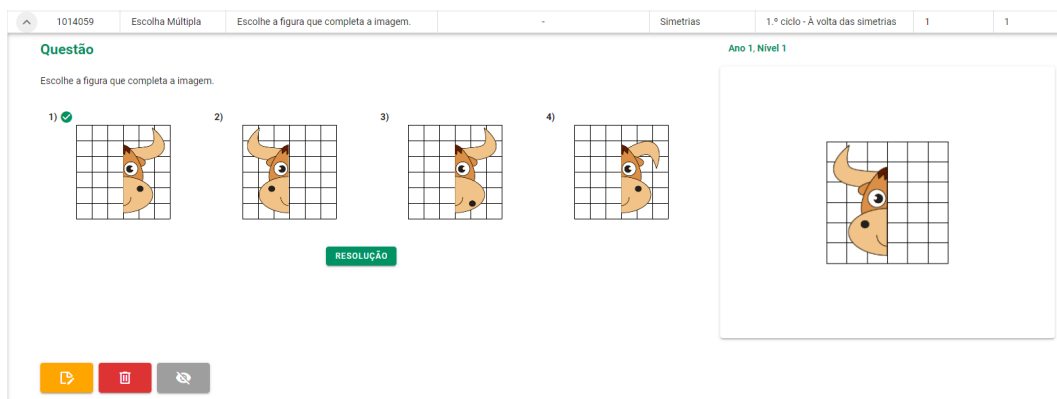


Figure 54: Approved Questions (Preview) Interface - Image Multiple Choice

As for the open-ended questions, they change according to the value of the 'auxiliar' column in the database:

- 1 - There is only one answer that is considered correct for that question and it is displayed below the input.

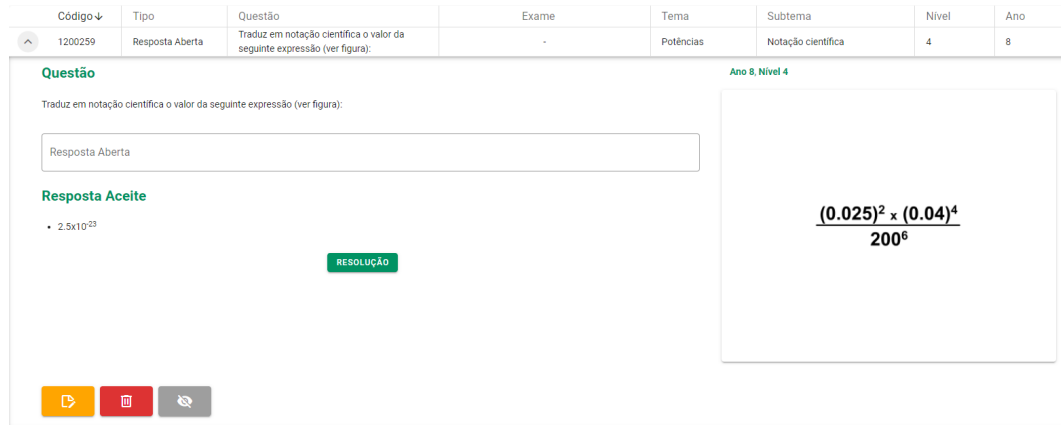


Figure 55: Approved Questions (Preview) Interface - Open Ended, 'auxiliar' 1

- 1000 - there are only two answers that will be considered correct for that question, displayed sequentially below the input field.

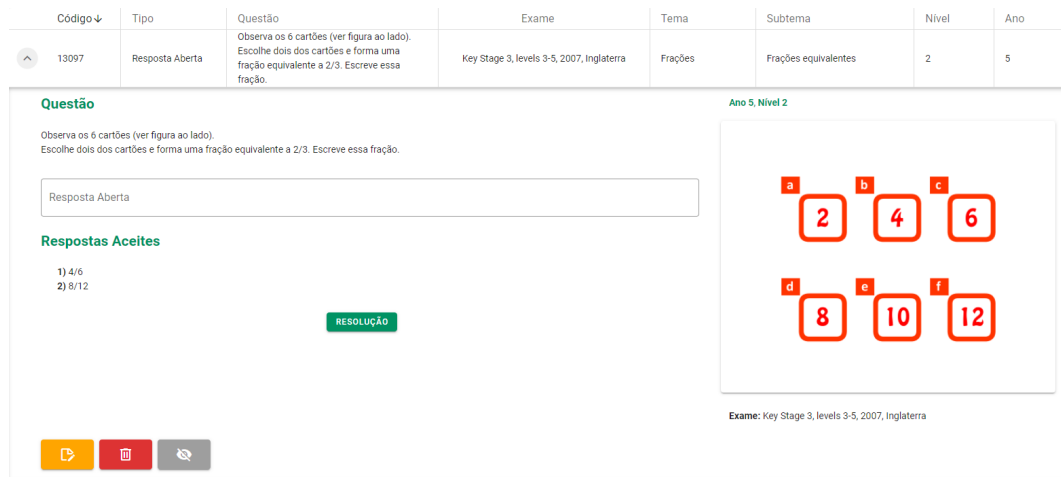


Figure 56: Approved Questions (Preview) Interface - Open Ended, 'auxiliar' 1000

- 0 - similar to the previous one, except there can be up to 6 answers.

Código	Tipo	Questão	Exame	Tema	Subtema	Nível	Ano
510087	Resposta Aberta	O José adicionou uma centena a 642. Assinala o número que o José obteve.	Teste Intermédio de Matemática, 2.º ano, 2013, Portugal	Adição e subtração - Naturais e zero	Adição e subtração - Naturais e zero	1	1

Questão Ano 1, Nível 1

O José adicionou uma centena a 642. Assinala o número que o José obteve.

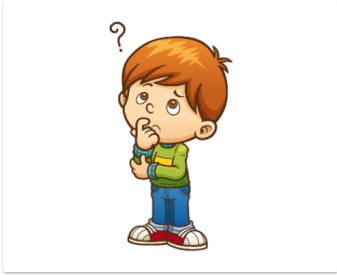
Resposta Aberta

Respostas Aceites

1) 742
2) 643
3) 652
4) 753

RESOLUÇÃO

▶
✖
↺



Exame: Teste Intermédio de Matemática, 2.º ano, 2013, Portugal

Figure 57: Approved Questions (Preview) Interface - Open Ended, 'auxiliar' o

- 22 - the answer can be any fraction equivalent to the one displayed below the input field.

Código	Tipo	Questão	Exame	Tema	Subtema	Nível	Ano
13035	Resposta Aberta	Escreve uma fração que represente a parte (do todo) pintada na figura.	-	Frações	Representação por frações	1	5

Questão Ano 5, Nível 1

Escreve uma fração que represente a parte (do todo) pintada na figura.

Resposta Aberta

Respostas Aceites

• Qualquer fração equivalente a: $\frac{1}{4}$

RESOLUÇÃO

▶
✖
↺

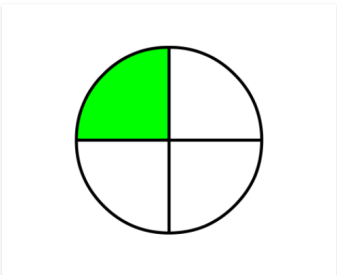


Figure 58: Approved Questions (Preview) Interface - Open Ended, 'auxiliar' 22

- 2 - the answer can only be the irreducible fraction in the answer displayed below the input field.

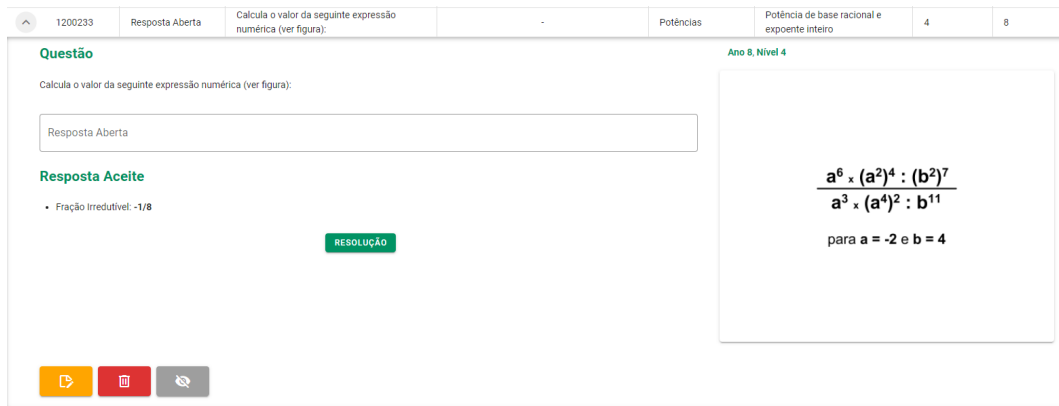


Figure 59: Approved Questions (Preview) Interface - Open Ended, 'auxiliar' 2

- 10 - the answer must be calculated via a protractor and it must be between the lower and upper bounds defined below the input field.

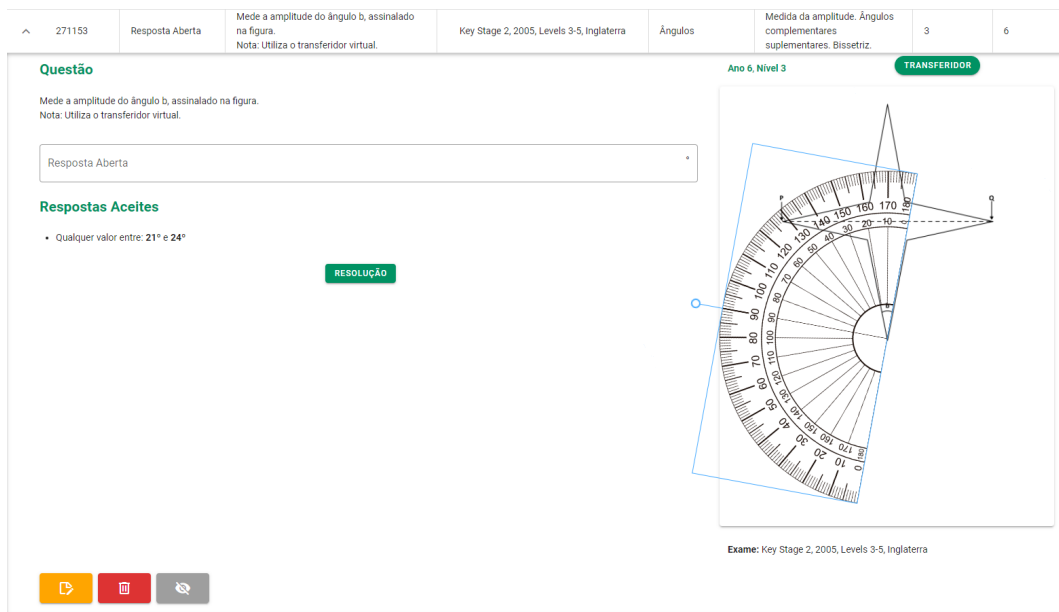


Figure 60: Approved Questions (Preview) Interface - Open Ended, 'auxiliar' 10

The true or false questions are very similar to the text multiple choice ones, the difference is that they only have 2 possible answers instead of a minimum of 3. They also display the check mark after the first answer to indicate that it is the correct one.

5100061	Verdadeiro ou Falso	X e Y são duas grandezas cujas medidas (em relação a unidades previamente escolhidas), respetivamente x e y, tomam os valores correspondentes, registados na tabela ao lado. Y é diretamente proporcional a X?	-	Proporcionalidade Direta	Grandeza diretamente proporcional a outra	1	0
---------	---------------------	---	---	--------------------------	---	---	---

Questão

X e Y são duas grandezas cujas medidas (em relação a unidades previamente escolhidas), respetivamente x e y, tomam os valores correspondentes, registados na tabela ao lado. Y é **diretamente proporcional** a X?

1) Não
2) Sim

RESOLUÇÃO

Ano 0, Nivel 1

x	2	3	4	6
y	6	9	12	15

Figure 61: Approved Questions (Preview) - True or False

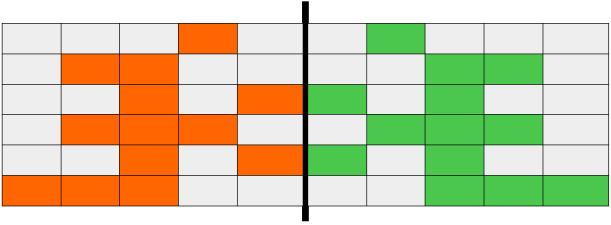
As it was shown in the previous interface section, the symmetry questions can either be:

- Vertical

1014041	Simetria	A figura é simétrica em relação ao eixo representado. Completa-a.	-	Simetrias	1.º ciclo - À volta das simetrias	1	2
---------	----------	---	---	-----------	-----------------------------------	---	---

Questão

A figura é simétrica em relação ao eixo representado. Completa-a.



RESOLUÇÃO

Ano 2, Nivel 1




Figure 62: Approved Questions (Preview) - Vertical Symmetry

- Horizontal

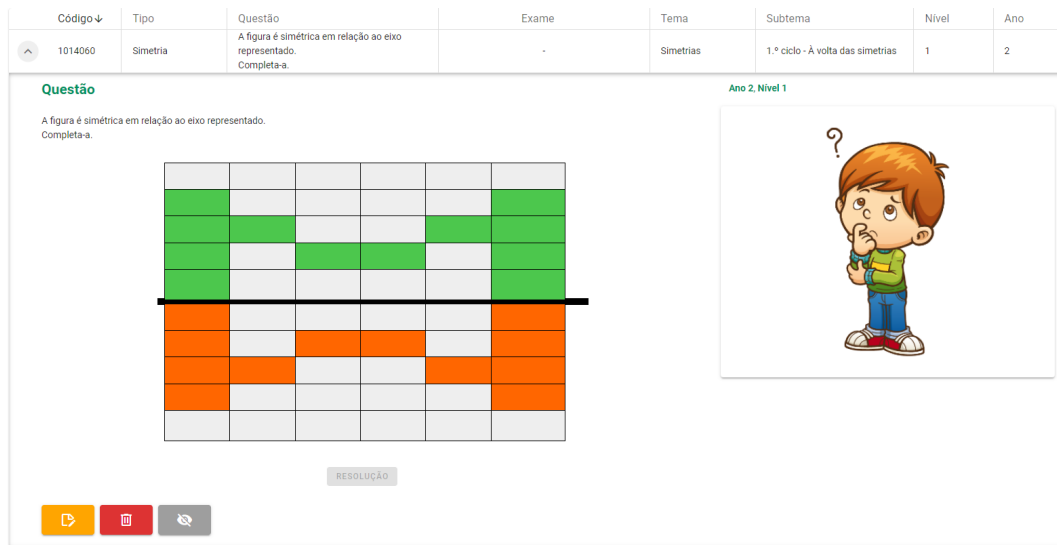


Figure 63: Approved Questions (Preview) - Horizontal Symmetry

The grid color scheme is also kept, where orange indicates the symmetry question statement and green refers to the expected solution, i.e, the correct answer.

After the preview, the operations buttons were also slightly redesigned, so they appear cleaner and tidier. Their function is to edit, remove or hide the question, respectively.

Active Submissions List

As it was the case with the approved questions list, this interface also changed from what was prototyped (figure 121) to a table-based approach for the same reasons as before.



Figure 64: Active Submissions List Interface

There is no need for a filter because not many submissions will be active at any given time and they are ordered by default from the oldest to the most recent, ensuring that older ones are not left forgotten.

One small addition was the inclusion of a check box that displays the user's submissions, which are hidden by default. The reason for this is because they can not be accepted by the same person that submitted them, they would just clutter the list with submissions that would obfuscate the ones that are from someone and are currently pending approval.

The same previews as before are reused here to give visual information on how the question looks like to the user, instead of using text-intensive approaches.

The buttons were reworked once again to fit the design of the page and to be more visually appealing, and their functions are to approve, edit or reject the submission, respectively.

Hidden Questions

Similarly to the last two interface sections, this one also swapped to a table-based view instead of what was planned at the beginning (figure 122).

Código	Tipo	Questão	Exame	Tema	Subtema	Nivel	Ano
21101	Escolha Múltipla	Qual é o perímetro da figura? Escolhe a opção correta.	Mathematics - 2009 School Certificate Test - Austrália(Board of Studies NSW)	Perímetros	Polígonos	2	5

Questão

Qual é o perímetro da figura? Escolhe a opção correta.

1) P = 34 cm
 2) P = 24 cm
 3) P = 70 cm
 4) P = 32 cm

RESOLUÇÃO

A figura não está à escala

4 cm
 7 cm
 10 cm
 3 cm

Exame: Mathematics - 2009 School Certificate Test - Austrália(Board of Studies NSW)

Figure 65: Hidden Questions List Interface

This one only has one button, that follows the same design as the other ones for the sake of consistency, and its function is to mark the question as visible.

Submission History

The table-based approach is also used in this interface, instead of the prototype's initial idea (figure 123).

The screenshot shows a web application interface for managing question submissions. On the left is a green sidebar with navigation options like 'Dashboard', 'Propor Nova Questão', and 'Histórico de Submissões'. The main content area is divided into three tabs: 'PENDENTES (0)', 'ACEITES (11)', and 'REJEITADAS (0)'. The 'ACEITES' tab is active, displaying a table of accepted submissions. Below the table, a detailed view of a submission is shown, including the question text, a list of multiple-choice options (with '1) Acutângulo' selected), a 'RESOLUÇÃO' button, and a diagram of an equilateral triangle with 60-degree angles. At the bottom, a 'Submissão Aceite' section shows the decision by 'donpepito' and buttons for 'Deletar' and 'Reenviar'.

PENDENTES (0)		ACEITES (11)			REJEITADAS (0)	
Código	Tipo	Questão	Tema	Subtema	Data de Submissão	Última Alteração
20201000044	Escolha Múltipla	Apenas num dos triângulos desenhados, as amplitudes dos ângulos são as indicadas. Assinala a opção correta.	Triângulos	Triângulos - Definições e classificações	26/09/2022, 14:45:51	29/09/2022, 19:10:41
20201000043	Escolha Múltipla	Quanto aos ângulos, o triângulo representado na figura é:	Triângulos	Triângulos - Definições e classificações	26/09/2022, 14:27:37	29/09/2022, 19:07:27

Questão Ano 5, Nível 3

Quanto aos ângulos, o triângulo representado na figura é:

- 1) Acutângulo
- 2) Retângulo
- 3) Obtusângulo
- 4) Equilátero
- 5) Escaleno
- 6) Isósceles não equilátero

RESOLUÇÃO

Submissão Aceite

• Decisão por: donpepito

Deletar Reenviar

Figure 66: Submission History Interface

This page's main quirk is the division of the submissions in three different tabs: pending, accepted and rejected, respectively, that contain them listed using the aforementioned table.

The main advantage of this design is that it is more efficient, if the questions were grouped in a single table it would be necessary to use filters, which would also make the page more cluttered and cumbersome to navigate.

There is also additional information after the preview for each submission, where it appears who accepted or rejected the question and the reason why, which the user can take as feedback and re-submit the question for approval if they deem necessary, otherwise they can just delete them.

Order Themes

This was a very straight implementation from its prototype (figure 124), since the concept itself is quite simple and there is not much room for improvement.

It was only slightly redesigned to be consistent with the other interface's color scheme, mainly in the color of the theme list containers.

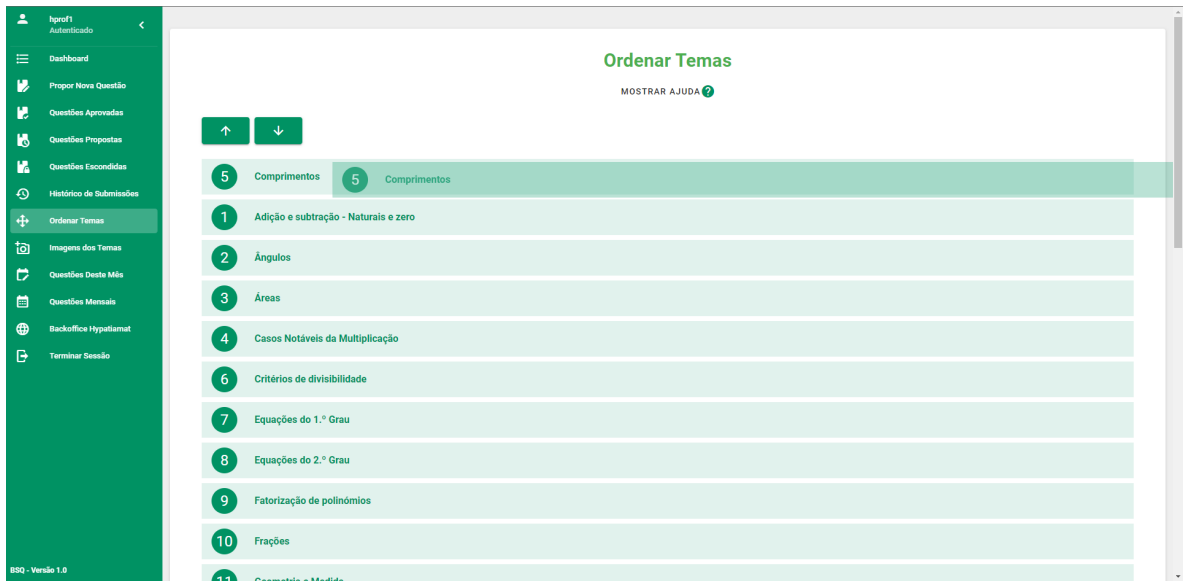


Figure 67: Order Themes Interface

The main difference is that instead of inserting up and down buttons to every single theme (like the mock-up), they can simply be dragged onto the correct place instead, saving the user multiple clicks and making it more intuitive to use.

Since the themes could be re-arranged multiple times, it was a good idea to offer the teachers some default ordering for them to use in case they need to, which is represented by the up and down arrows at the top of the page, representing ascending and descending alphabetical order, respectively.

Change Theme Images

The main issue with the mock-up (figure 125) is that it does not behave well in a tablet or phone due to it being stretched horizontally and it may overflow on a smaller screen.

Therefore, simply swapping the layout so that it goes from horizontal to vertical solves that issue, while keeping the simple design of the original prototype.

There are two sections, one for the current image associated with the theme and other dedicated to giving the user a place to input the new image and preview it, on the right side.



Figure 68: Change Theme Images Interface

On a side note, these previews appear with the same design as they would in the second application, to be explored in the next chapter, with green rounded borders so that the teachers can visualize how it will look without having to save the changes and navigating to it to view them in action.

Current Monthly Questions

This interface is a slight extension of the approved questions list, in fact, it keeps the same filters and question list in a table, which slightly differs from its prototype (figure 126) in that regard.

This interface features three main sections:

- Category selector - there are currently 4 categories (they act as tabs) that represent students in a given grade (chapter 3.1.1);
- Selected questions - questions codes (represented with a chip element) that are selected for the selected category;
- Question selector - question table and filters that allow the user to pick which questions they want to add to the selected category;

The design was slightly changed to include the iconic green and white color scheme that is consistent with the other pages.

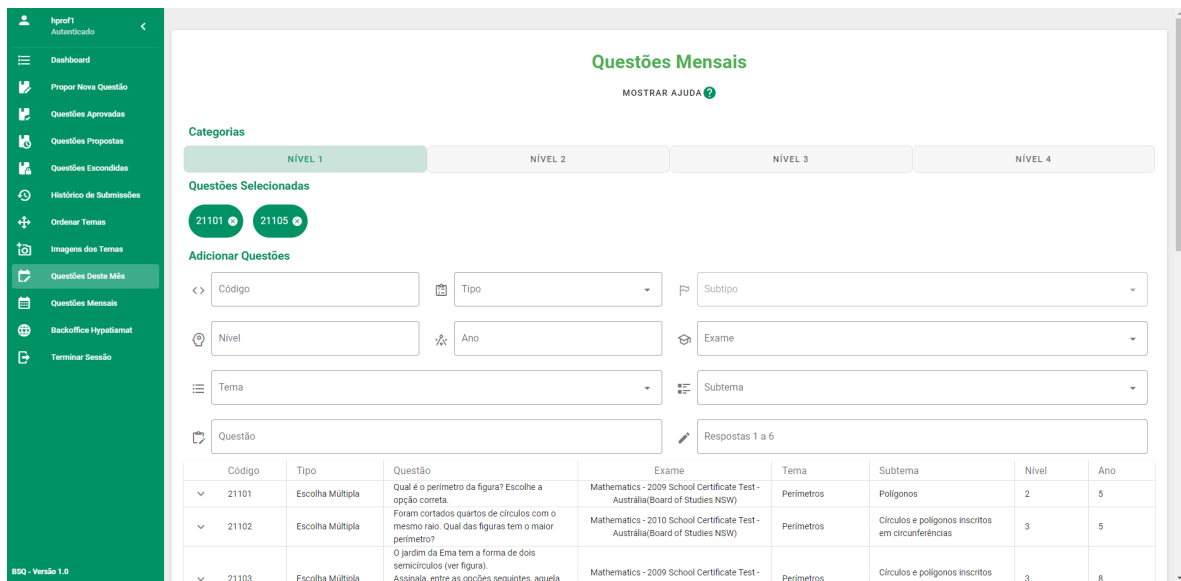


Figure 69: Current Monthly Questions Interface

The preview interface of these questions is exactly the same as the ones used in all the other ones above, the only difference is the button on the bottom of it.

This is the button used to add the question to the current selected category, which can be undone by clicking the 'X' on the chip element itself in the selected questions section.

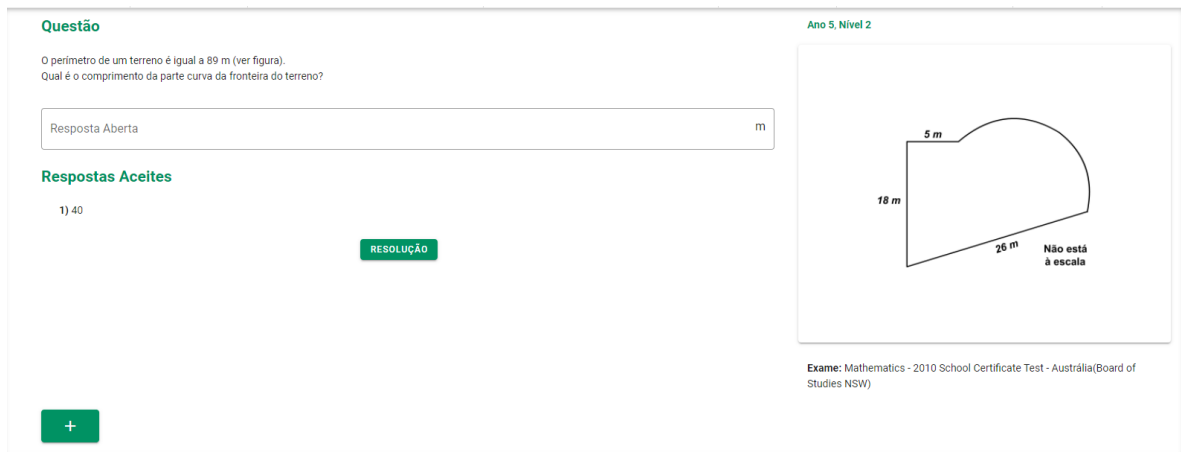


Figure 70: Current Monthly Questions (Selection) Interface

It is important to keep in mind that these monthly questions are only saved for the **current** month, whenever a new month starts they are reset and get added to the history.

Monthly Questions History

Lastly, this interface went through some design changes when compared to its prototype (figure 127) to make it easier to use and be more efficient, in terms of user effort.

The year and month selectors proved to be quite inadequate for this particular page because there could be gaps in them, i.e, there could be some where no questions were added to the monthly rotation. By having the incremental/decremental number selector, it would not exclude these scenarios and it would force the user to scroll through them even though they contain no important information, which is a terrible design choice.

Fortunately, the fix is rather simple and instead of having a plus or minus button, it has next and previous buttons that can only contain valid values, i.e, years and months that had monthly questions. Shifting them to the middle makes them the focus of attention, which is a small improvement but it makes the entire process more intuitive for the user.

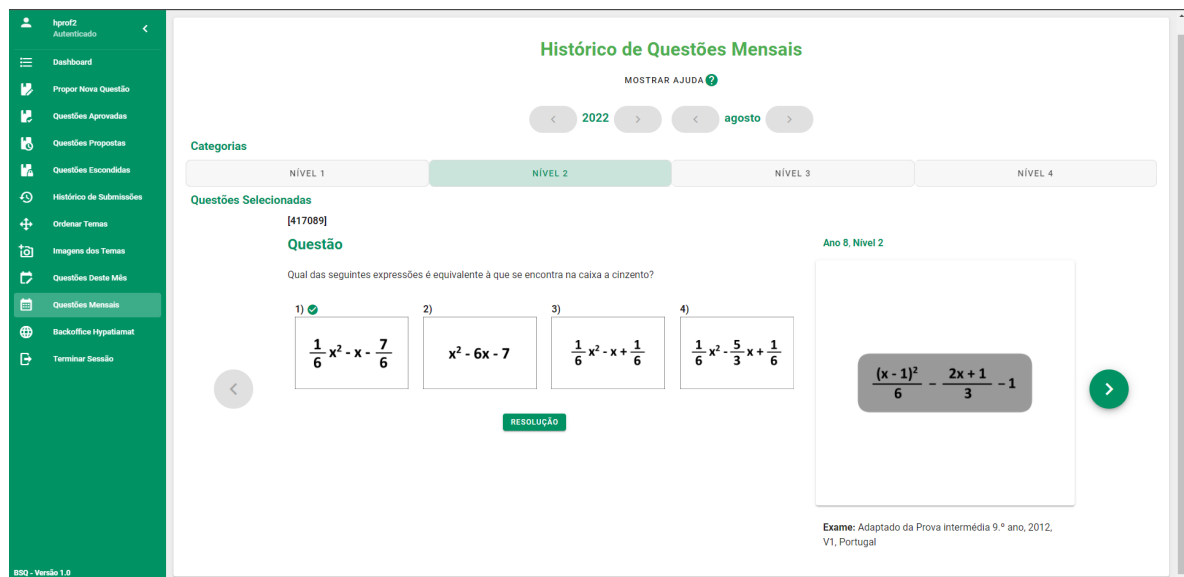


Figure 71: Monthly Questions History Interface

After the user has successfully selected the correct year and month they wish to browse, they need to pick the category they wish to visualize, which are the same as the ones in the monthly questions page.

Afterwards, the same question preview interface as the one seen in the other pages appears for each question in that category. The user can then scroll through that list using the right and left chevron buttons that wrap the component.

3.6.3 Saving the User State

The user has to complete an authentication step before they are able to access the back-office itself. If their *JWT* token is not saved, they will not be able to complete the requests to the back-end's endpoints because it will assume they are not logged in, thusly blocking their access.

It becomes clear that anytime the user successfully logs into the application, its corresponding data should be stored for posterior use.

Evidently this token needs to be stored in a non-volatile manner, i.e, it should not be saved in local variables that get reset whenever the page reloads but rather in the browser's storage itself. If it does not follow this rule, the user would have to authenticate every time they enter or refresh the page, which would be terrible for productivity and it would be a nuisance after some time.

Every reputable browser offers a simple local storage *API*, typically by the property 'Window.localStorage' (MDN, 2022). This simplicity is due to just having a key-value pair and methods to update them, and they are restricted to very few data types such as numbers and strings.

Furthermore, it is clear that this state should be **global** and every single Vue component and their children should have access to it, since they can include their own *HTTP* requests. By default, this is not something supported (or recommended) since every component has its own variables and if the root component were to pass this global state to its children, it would massively hamper the performance since it would have to update every single one that references it.

This is solved using Vuex, a state management library that acts as a centralized store for all the components in a Vue application. This is still done using volatile memory, so it is important to use it in tandem with the 'Window.localStorage' property, ensuring that whenever the state changes, it gets saved in the browser's storage. Likewise, whenever the window is reloaded, the Vuex state manager must import the saved data so that the components can use it freely.

Each Vuex module may contain the following sections:

- **state** - variables and other local data;
- **getters** - computed properties for the state variables;
- **mutations** - methods that modify the state (synchronous);
- **actions** - methods that commit mutations and may contain asynchronous operations.

There is only one Vuex module for this particular application, namely one dedicated to the authentication process and it contains two main focus points: logins and logouts.

Whenever a user successfully logs in, an **action** is dispatched where the *JWT* and other data is stored in the Vuex instance and local storage. Since the authentication response contains the time when the token will expire, it is possible to include a timer (asynchronous, hence the need for an action) that will run until that date is reached, in this case it will last a total of 8 hours. Since this is done using JavaScript, this timer will only be active while the window is open, which means it needs to be refreshed whenever the page is reloaded.

On the other hand, a logout is relatively simple. It either happens when a user clicks the logout button or when the timer runs out, in both cases the state is completely wiped from both the Vuex instance and local storage, forcing the user to authenticate again in order to use the back-office.

The mutations themselves are quite trivial, they are only used to change the state from one value to another, similar to a setter in the object oriented programming paradigm.

The getters allow the creation of computed properties based on the current state values such as getting a boolean value for whether the user is logged in or not, which is useful when building the views themselves and for actions that require access to current state values.

This results in a very simple way of knowing whether the user is authenticated or not and allowing them to stay logged in while the token does not expire, ensuring that this information is kept whenever the page is reloaded or opened for the first time.

Now that every component of the back-office is complete, with security and all its other features in mind, it is now ready for deployment (chapter 5), which was a challenge in of itself. The next chapter will be dedicated to its counterpart, the "I Want To Solve Questions About..." application that will also use this authentication method.

I WANT TO SOLVE QUESTIONS ABOUT...

This chapter denotes the development of the new *"I Want To Solve Questions About..."* application, which is similar to the back-office in terms of its underlying architecture.

As it was the case previously, there needs to be a first step of understanding the problems and thinking of ways to fix them before attempting to develop the code and logical models themselves.

Creating a coherent final product that aligns with the teachers' vision is fundamental not only for the products success but also for the platform's relevance in the market.

4.1 REQUIREMENTS SPECIFICATION

According to the teachers, the previous application does not fit their needs, both visually and technologically, hence the need for the creation of a new one that attempts to mitigate these issues.

As it was the case with the previous chapter, this section is extremely important not only to understand what the teachers require from the new application, but also what exactly is wrong with the current one, so that a consensus can be reached on what needs to be changed and why.

Therefore, it is important that before any code is developed, all the features are outlined and fully understood before committing to the task, where its complexity lies in how the interfaces can have improved usability and appeal.

These improvements should also give the students a more complete and organized application to work with, which should further improve the way the consume knowledge in it.

4.1.1 Functional Requirements

The goal of this application is to replace the current version in production, currently lacking in a modern user interface and technologies, causing it to not work properly in mobile devices such as tablets and phones.

As such, the main improvements that need to be done are on a visual level, specifically the uncluttering of the information displayed to the user at once, and making sure that the consistency with the other *Hypatiamat's* applications design choices.

The following figures represent its current state and can be accessed at the following [URL](https://www.hypatiamat.com/questoesde/resolverquestoesde.html): <https://www.hypatiamat.com/questoesde/resolverquestoesde.html>.



Figure 72: Current "I Want To Solve Questions About..." landing page

The landing page of the current "I Want To Solve Questions About" is relatively simple to understand, as it is just a theme selector (the big squares on the image) with three pages, currently. On the left and right sides, the user can go to the next or previous pages by clicking the yellow arrow buttons.

Note that the theme selection concept itself, despite its outdated look, is not exactly a bad design choice for this application, it works perfectly well on a mobile environment due to its oversized buttons and in the computer as well, since it can fit the screen with more items according to the user's resolution or screen size. This particular application does not take that into account due to technological limitations, it simply scales down the canvas size in order to fit the screen, rendering it completely unusable in smaller screens.

The order of the themes is also hard-coded, something that the teachers deeply regret doing as they can not easily change their order. In the new version, it is expected that the themes appear in the set order saved in the back-office.

Another thing that looks somewhat out of place in this page are the buttons in the bottom left corner, which represent the user's favourite questions, a search bar and instructions, and on the top right corner, which is a simple login button. Instead of having them clutter this page, because they look like afterthought additions instead of design choices, it is better to give them their own dedicated areas on the new interface.



Figure 73: Current "I Want To Solve Questions About..." question solving page

Whenever a theme is picked, this page will be shown to the user with the questions for that particular theme. On the left of the canvas there is the sub-theme selector and, once they click one of them, the question solving screen will be appear in the center of the screen.

On the top of the screen, the question's code appears on top of the question statement and the page's numeration on the right. There is also the difficulty selector right at the middle, represented by the buttons from 1 to 5 and a T that represents any difficulty.

On the reference picture at the right side, there is a small star on its top left corner, which allows the user to mark this question as a favourite. However, it only allows the user to have **one** list, which needs to improve in the new application and scale to **any** number of lists, similar to how YouTube implements it. This is a direct requirement from the teachers, since they could create these lists with some questions for their next classes, which is currently unfeasible with the available version. This would also benefit the students, as they could save their favourite questions and browse through them quicker due to them being better organized.

There are three other buttons on top of the image that enable a protractor, show the favourites list and open a search bar, respectively. The first one should not be enabled by default, as only one specific type of question makes use of it, so it just becomes useless and wasted space in all the remaining types.

The second one, opens the favourites list, which was explained above.

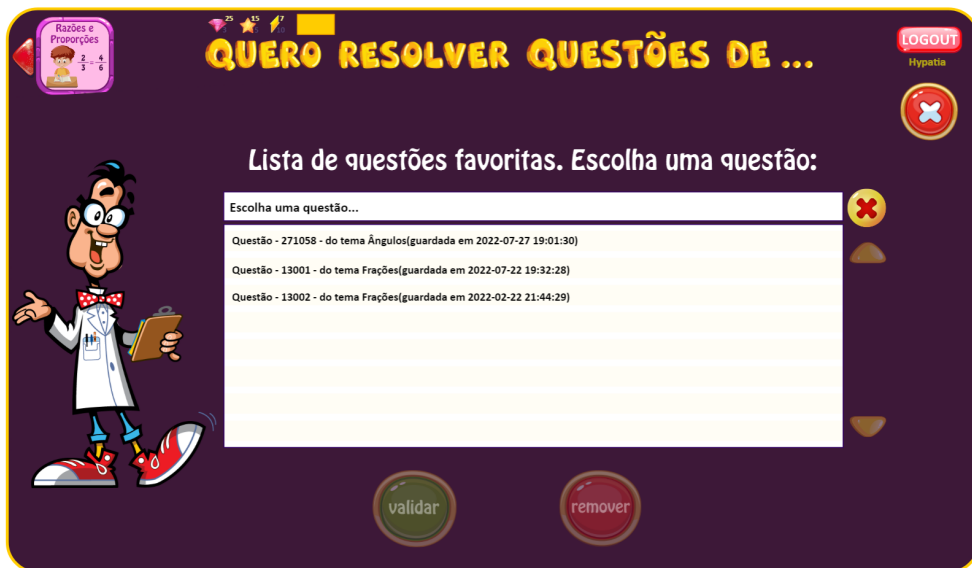


Figure 74: Current "I Want To Solve Questions About..." favourites page

The way it displays the information does not fit its purpose. Every item on the drop down list contains only the code, theme and date when it the question was added to the list, which does not suffice the teachers' needs. They have no way of quickly navigating to the question other than memorizing those codes and typing them in the search bar, which is the third button. This problem also applies to the students, it is not expected of these young students to memorize the codes either. This is another improvement that must be done in the new application, reworking the favourites to grant all these features.

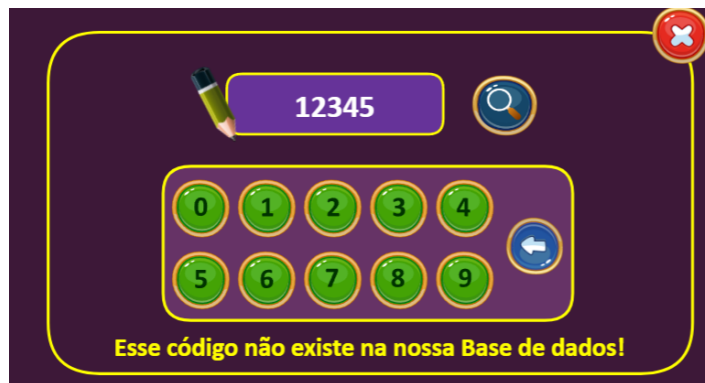


Figure 75: Current "I Want To Solve Questions About..." search bar

The search bar, simply put, is inefficient and inadequate. The only way for the user to search a specific question is by knowing its specific code and the user should never have to memorize anything that can possibly be over 10 digits long, considering that there are over 3000 questions in the database currently. Therefore, the teachers asked for a major improvement on this functionality and allow the user to search questions based on any of their meta-data, not only its code.

On the bottom right side, the previous and next buttons indicate that the user can scroll through all the questions in that particular sub-theme, whenever they want to, an unlimited number of times. This is not something that will be kept in the new application, the idea is that unauthenticated users can only view up to **5** questions **maximum** of all themes and sub-themes combined **per day**, whereas authenticated ones have unlimited access, in order to incentivize them to create an account on the website and make use of all of its features, which is proven to boost overall engagement and assiduity in the platform.

The page also features a 'Like' button, which works as intended and the teachers and students are happy with its functionality, where the users can only use it if they are logged in, which avoids them maliciously trying to break the website and, in case they do, they can be tracked more easily.

The page overall is efficient and intuitive, it only needs some slight touch ups to ensure that it is on par with the other *Hypatiamat* applications.

Along with these improvements to the existing features, the new application should also implement new ones to further deepen and boost its gamification factor. These should also be locked behind authenticated accounts, so that "free" users consider creating an account even more.

The first gamification feature is the monthly questions, which was explained in detail in section 3.1.1.

The existence of a leaderboard that keeps track of how many correct answers the students got each month is fundamental for this feature's success, since it incentivizes them to solve the questions correctly by making them compete against each other. It should be updated whenever the monthly questions reset, which is on the first day of each month, and should be organized by school year.

After this reset, these questions should be added to an history (also explained in 3.1.1), so that the users can see if they got the questions wrong or right and their respective resolution.

4.1.2 *Non-Functional Requirements*

The first part of these requirements are exactly the same as the ones in 3.1.1, i.e, making sure that the interface is consistent with all the other applications in the platform.

There is one major difference between the two, which is not only being compatible with the popular browser choices, but also with multiple mobile devices. In this application, it is no longer considered a bonus to support these devices, but a requirement, since many of the students have tablets at their disposal.

The currently solution is scaling the canvas itself, suitable for when the user is using a computer. However, if they swap to a smartphone or tablet, the buttons become too small to use, which severely hinders the application's usability.

On the new application, the interface itself should be responsive and change according to the device the user is on, which is possible using the current available technologies.

4.2 INITIAL ASSESSMENT

As seen in section 3.2, the *Hypatiamat* database server already contains a large number of tables, some of which were relevant for the back-office.

In this application, both databases will be also be necessary, i.e, both the teachers ("*hypati67_aplicacoes*") and questions ("*hypati67_testeconhecimentos*") and all their related tables, for the same reasons as the ones explained in that chapter.

It will not access any of the back-office's specific tables ("*hypati67_questoes*") since it will only display non-quarantined questions.

The first extra database was created to allocate space for all the new tables related to the back-office, which contains a bunch of tables automatically generated by Strapi. If one were to use two Strapi instances on the same database, they would rewrite each other and soon become unusable, which is something that needs to be avoided in order to ensure that the applications work in parallel.

In the next section, it will be defined that Strapi will be the choice for the back-end of this application, just like the back-office. Therefore, there is a need to create a new database for it, in order to contain all the new tables necessary to implement the set requirements. After contacting the owners, the final name will be "*hypati67_qrq*" and will be initially empty, as expected, where it will be the new back-end's responsibility to manage, since it will have full permissions on it.

As it was the case previously, any database schema or image in the following sections will be a simplified version of the current database, since it is a complex database with many relationships and tables intertwined.

4.2.1 Favourites

One of the requirements is the rework of the favourites feature that the previous application provides.

In the "hypati67_testeconhecimentos" database, there is a table called 'favoritos' containing all the favourites of every user:

- **id** - auto-incremented primary key;
- **coduser** - foreign key to the users table;
- **codquestao** - foreign key to the questions table;
- **folder** - legacy column, every entry has the value 'root' and is currently unused;
- **nome** - legacy column, every entry has a string value and is currently unused;
- **datacriacao** - date when the favourite entry was created by the user.

As a reminder, the requirement was that instead of only having **one** list of favourites, there should be the possibility for the user to use an **arbitrary** number of lists, for better organization and efficiency.

Another important requirement is that the new application should not break previous *Hypatiamat* services, such as massively changing the tables' structure.

Thankfully, in this case it is rather easy to implement this feature by making use of one of the unused columns which is conveniently named 'folder' and its functionality would be the same as the required lists.

By allowing the user to create new folders, it is possible to save different favourite entries according to its folder, which means that the user could, in theory, create any number of folders they like.

Note that this would not break in any way the existing applications, as this field is unused and any of the 'root' entries would be considered the default folder for legacy accounts. The current "I Want To Solve Questions About..." would also work the same way as it currently does, it would simply list every folder as if they were a single list because it does not take this column into account, only the new version would show the division correctly.

4.2.2 Likes

The current version of the application currently has a feature that allows the user to add or remove likes on questions, allowing other users to see the approval rate of the question (higher is better).

The table is called 'likequestions' in "hypati67_testeconhecimentos" and it is very simple in nature, as it only needs to save which user placed the like and where:

- **id** - auto-incremented primary key;
- **user** - foreign key to the users table;

- **questionid** - foreign key to the questions table.

It does not need any changes nor additions, its structure will be kept for the new application as well.

4.3 TECHNOLOGY SELECTION

The technology selection was a straightforward process, as the reasons behind them are the same as the ones discussed at 3.3.

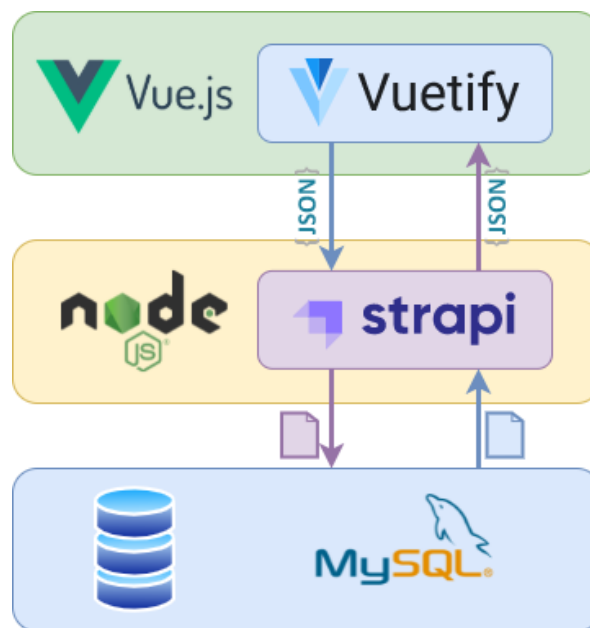


Figure 76: Technological stack for "I Want To Solve Questions About..."

In addition to the numerous benefits detailed in the aforementioned section, it will also allow a quicker and easier development of responsive interfaces for multiple devices, as Vuetify features an extensive library of built-in components that scale according to the screen size and resolution.

Another great benefit of keeping the same technological stack is that Vue also allows the developer to reuse interface components, as long as they do not have any pending dependency.

Not only that but also some configuration files from Strapi can also be reused, specifically the ones that relate to the questions and teachers tables, they do not change since the connection parameters are exactly the same, another great benefit of keeping the same back-end technology.

MySQL will also be kept as the only persistence layer, as the overhead of creating a new database server does not outweigh the performance improvements they could potentially bring for the same reasons as before.

4.4 LOGICAL DATA MODEL

The logical model below only adds four extra tables and relationships to the existing ones in "hyptati67_testeconhecimentos" (chapter 3.4) in order to fulfill all the set requisites.

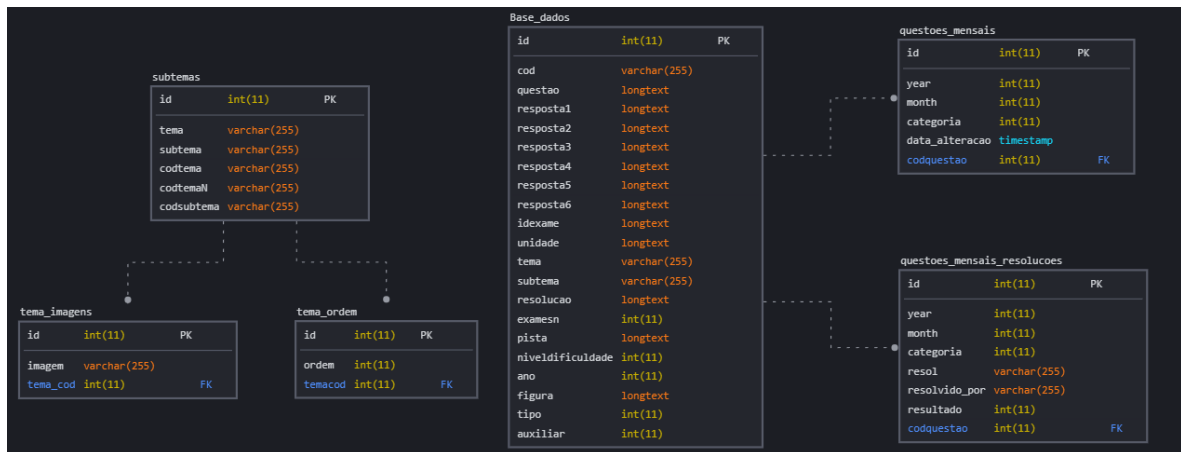


Figure 77: Logical data model for "hyptati67_qrq" (Simplified)

Starting with the themes, the front-end application needs to display them by the currently saved ordering system, which is represented by the "tema_ordem" table. Its structure is simplistic:

- **id** - auto-incremented primary key;
- **ordem** - integer from 1 to N, where N denotes the total number of different themes currently in the database and it represents the current order for each one;
- **temacod** - foreign key to the theme.

Each theme has an image associated to it, currently hard-coded in the original application, as seen before. By using a new table, it is possible to create or edit these links in the back-office, since every theme is supposed to have one. Therefore, the creation of the table "tema_imagens" is necessary for the new application to correctly display the user the intended picture for each case:

- **id** - auto-incremented primary key;

- **image** - name of the image, the *URL* is obtained by concatenating it to the back-end's static resources endpoint;
- **tema_cod** - foreign key to the theme.

The monthly questions, as the name implies, are questions that teachers highlighted for a particular month of a given year, organized by categories (3.1.1). The table "questoes_mensais" has all the necessary fields in order to save this data:

- **id** - auto-incremented primary key;
- **year** - year the question appears in;
- **month** - month the question appears in;
- **categoria** - the question's category;
- **data_alteracao** - date when the question was added to the monthly questions;
- **codquestao** - foreign key to the questions table.

The students have the option to submit their resolutions on the current rotation of questions, so that they can later have their position on the leaderboard updated. Therefore, they need to be persisted in a table, which in this case is "questoes_mensais_resolucoes":

- **id** - auto-incremented primary key;
- **year** - year the question appears in;
- **month** - month the question appears in;
- **categoria** - the question's category;
- **resol** - the answer the student gave;
- **resolvido_por** - foreign key to the student table;
- **resultado** - whether the answer is correct or not (saves computational efforts later);
- **codquestao** - foreign key to the questions table.

With the logical model fully complete, it is now possible to start the development of the back-end server, which is the next section.

4.5 BACK-END DEVELOPMENT

By using the same technology as the previous back-end, it is possible to reuse some components and configurations that were used in it.

As such, after creating a new Strapi instance and linking it to the new database so it can dump all its configuration and permissions tables, it is possible to start creating the collections and routes that will be needed for the "I Want To Solve Questions About" interface.

4.5.1 Initial Setup

The setup is identical to the first back-end server (3.5.1), where the main differences are related to the collections and consequently, the exposed *API* endpoints.

These collections link to their corresponding tables in the *Hypatiamat's* database servers:

- **School** - "escolas" in "hypati67_aplicacoes" (same as figure 14)
- **Question** - "Base_dados" in "hypati67_testeconhecimentos" (same as figure 18)
- **Teacher** - "professores" in "hypati67_aplicacoes" (same as figure 20)
- **Theme** - "subtemas" in "hypati67_testeconhecimentos" (same as figure 23)
- **Student** - "alunos" in "hypati67_aplicacoes"

10 fields

Ab	nome	Text
Ab	datanascimento	Text
Ab	turma	Text
Ab	pais	Text
123	numero	Number
Ab	codprofessor	Text
🔍	user	UID
Ab	email	Text
Ab	password	Text
Ab	escola	Text

Figure 78: Strapi Student Collection

- **Favourite** - "favoritos" in "hypati67_testeconhecimentos"

5 fields		
Ab	coduser	Text
Ab	codquestao	Text
Ab	folder	Text
Ab	nome	Text
📅	datacriacao	Timestamp

Figure 79: Strapi Favourite Collection

- **Like** - "likequestions" in "hypati67_testeconhecimentos"

2 fields		
Ab	user	Text
Ab	questionid	Text

Figure 80: Strapi Like Collection

- **Theme Order** - "tema_ordem" in "hypati67_qrq"

2 fields		
123	temacod	Number
123	ordem	Number

Figure 81: Strapi Theme Order Collection

- **Theme Image** - "tema_imagens" in "hypati67_qrq"

2 fields		
Ab	tema_cod	Text
Ab	imagem	Text

Figure 82: Strapi Theme Image Collection

- **Monthly Question** - "questoes_mensais" in "hypati67_qrq"

5 fields


123	year	Number
123	month	Number
123	categoria	Number
Ab	codquestao	Text
	data_alteracao	Timestamp

Figure 83: Strapi Monthly Question Collection

- **Monthly Question Resolution** - "questoes_mensais_resolucoes" in "hypati67_qrq"

7 fields

123	year	Number
123	month	Number
123	categoria	Number
Ab	codquestao	Text
Ab	resol	Text
Ab	resolvido_por	Text
123	resultado	Number

Figure 84: Strapi Monthly Question Resolution Collection

4.5.2 Creation of the API Endpoints

As it was the case with chapter 3.5.2, only the custom made routes will be mentioned, as Strapi generates default routes automatically.

Naturally, some of the routes for the same collection are not present in this back-end server, as they are intended for back-office use only.

Student

This one does not have any custom-made endpoints, only the default ones are used.

Documentation

- **GET** /documentacao - contains the Swagger documentation for the back-end;
- **GET** /documentation - exactly the same as the previous one.

Favourite

- **GET** /favoritos/utilizador - returns a list with all the favourites related to the user that sent the request;
- **POST** /favoritos/listas/cod - saves a question with code 'cod' on the user that sent the request's favourite list, which is identified in the request body.

Like

This one does not have any custom-made endpoints, only the default ones are used.

Monthly Question

- **GET** /questoes_mensais/mes - returns a list with all the monthly questions for the current month;
- **GET** /questoes_mensais/disponiveis - returns a list of with the history of monthly questions.

Monthly Question Resolution

- **GET** /questoes_mensais_resolucoes/mes - returns the monthly submission, if existent, of the current monthly questions;
- **GET** /questoes_mensais_resolucoes/todas - returns the history of submissions, if existent, of all the editions of monthly questions;
- **GET** /questoes_mensais_resolucoes/classificacoes/anosletivos - returns the leaderboard for all the school years that had monthly questions.

Question

- **GET** /perguntas/exames - returns a list of all the distinct tests or exams in the questions table;
- **GET** /perguntas/visiveis/tema/subtema - returns a list with all the questions that belong to the given theme and sub-theme, with all the text transformations applied;

- **GET** /perguntas/gostos/tema - returns a list of objects that contain the list of users that liked a question with a given code, for each theme;
- **GET** /perguntas/:cod - overrides the default '/perguntas/:id' since it is easier to access the question given its code instead of the auto-incremented identifier and applies the text transformations.

School

- **GET** /escolas/:cod - overrides the original '/escolas/:id' since it is easier to access the code that identifies the school instead of the database auto-incremented identifier.

Teacher

- **GET** /professores/:cod - overrides the default route with a new one since it is easier to access the teacher given their code instead of the auto-incremented identifier;

Theme

- **GET** /temas/agrupar - returns the list of all the themes and sub-themes grouped by the name of the theme itself, where the order is defined by the "tema_ordem" table;
- **GET** /temas/agruparImagens - returns a list of objects containing the themes with their respective image saved in the "tema_imagens" table.

4.5.3 Authentication

The authentication process is also done using *JWTs*, similar to the the back-office's back-end server (chapter 3.5.3).

The difference is on the response body after the user is authenticated. The back-office had to be blocked behind valid teacher accounts, however, *"I Want To Solve Questions About..."* is also aimed at students.

Therefore, in addition to the valid token displayed in that chapter, this is also an example of a valid student authentication result:

```

1 {
2   "token": {
3     "jwt": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7ImFncnVwYW1lbnRvIjoiaWw2Z2xhIGRvIEh5cGF0aWZtYXQsIEJyYWdhIiwiaWF0Ij0jQWMTgyLCJ1c2VyIjoiaHlwYXRpYTxiIiwiaWF0Ij0jZ29yZW5pbmVzcmVzc2FAbmV4aXN0ZS5wdCIjImVzY29sYSI6Imh5cGF0aWwEwMSIsInR5cGUj0jEwLCJ0cG9ueXBlijoieWx1bm8ifSwiaWF0Ij0jY1Njg4MTE2LCJleHAiOjE2NjUzMjE1MTZ9.ggdh02AM-pNnC706aZq1TDrlj10-GMADiy8a4Zw4GI4",
4     "expiresIn": 28800
5   }
6 }

```

```

5     },
      "user": {
7         "id": 40182,
          "user": "hypatia01",
9         "numero": 1,
          "nome": "Hypatia 01",
11        "datanascimento": "12/12/2012",
          "escola": "hypatia01",
13        "turma": "3A-21-1",
          "email": "agoraninteressa@nexiste.pt",
15        "password": "4297f44b13955235245b2497399d7a93",
          "codprofessor": "hprof2",
17        "pais": "Portugal",
          "confirmacao": 1,
19        "published_at": "2021-04-14T22:48:34.000Z",
          "created_at": null,
21        "updated_at": null,
          "type": "aluno"
23     }
  }
}

```

The front-end will then be able to differentiate between these two types of user accounts and change the interfaces accordingly.

4.5.4 Protecting the API

In chapter 3.5.4 it was revealed that every single *API* endpoint was protected by the authentication procedure.

However, in this new application the user is not only teachers but also students and they may opt to use the limited unauthenticated features that the website provides, such as solving at most 5 questions daily.

By using Strapi's default "Public" role, any endpoint that is added to it will **not** require the user to be authenticated.

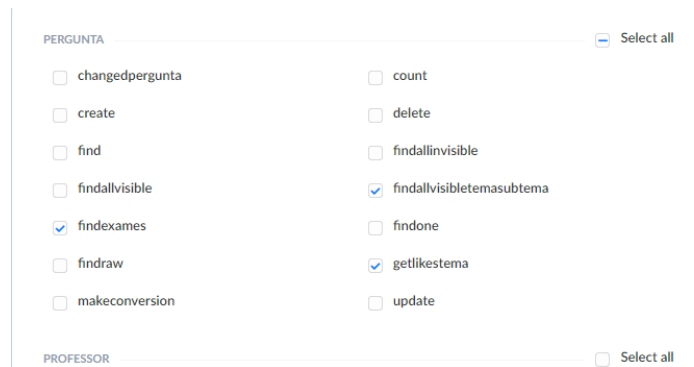


Figure 85: Strapi 'Public' Role - Questions API

In contrast, the "Authenticated" role will require the user to add their *JWT* authentication token to the request, such as the monthly questions that the free users will not be able to access unless they create an account.

This effectively prevents the free users, even if they somehow manipulate the interface to bypass the built-in functionality of blocking their access to the monthly questions, from consuming the *API* endpoints directly and getting that information.

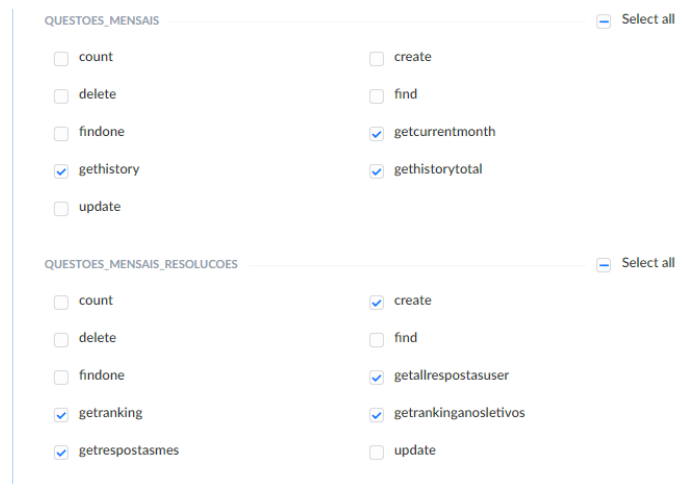


Figure 86: Strapi 'Authenticated' Role - Monthly Questions API

This marks the end of the back-end development and the creation of the *API* endpoints, which will be consumed by the final front-end application to complete the new version of "I Want To Solve Questions About...".

4.6 FRONT-END DEVELOPMENT

4.6.1 Initial Setup

The new interface will be built using Vue.js, making it possible to re-use the back-office's front-end components previously developed.

In addition, João Vieira also developed a digital keyboard that will be useful for open ended questions, where some symbols such as the infinity and square root are built-in with the correct formatting (Vieira, 2021).



Figure 87: First iteration of the digital keyboard (João Vieira)

However, the keyboard itself does not have a fitting color theme, making the characters harder to read. For example, the red button with black text can only be properly read if squinting or giving it a lot of attention. The same can be said for the green and yellow buttons, however the effects are much more diminished than the previous case.



Figure 88: Second iteration of the digital keyboard

By making the characters white on the keys that have the aforementioned background colors, it becomes much clearer and easier to read at a glance, which the students will appreciate in the long run. It was slightly enlarged for better readability aswell.

This is the version that will be used in this application, with João and Hypatiamat's permission.

4.6.2 Creation of the Interfaces

The application itself is very similar to the previous back-office, in terms of its design.

This is intended as every single *Hypatiamat* service, as mentioned before, needs to maintain consistency with all the others. Keeping the green and white color palette and the navigation bar on the left is the only way to achieve that, as all the others applications follow these exact same rules.

Every single interface went through a large number of iterations, where the final product was approved by the *Hypatiamat* team for production.

Authentication

The first front-end application had to force the user to authenticate before attempting to use the application, something that is not needed here.

“I Want To Solve Questions About...” does not lock the user behind an authentication wall, it just **incentivizes** them to do it, due to the question blocking requirement.

Therefore, it makes sense to allow the user to login whenever they want to and allow them to see the website in its entirety. By using a login modal, the user is not redirected to a specific login page, allowing them to continue their uninterrupted workflow in the application.

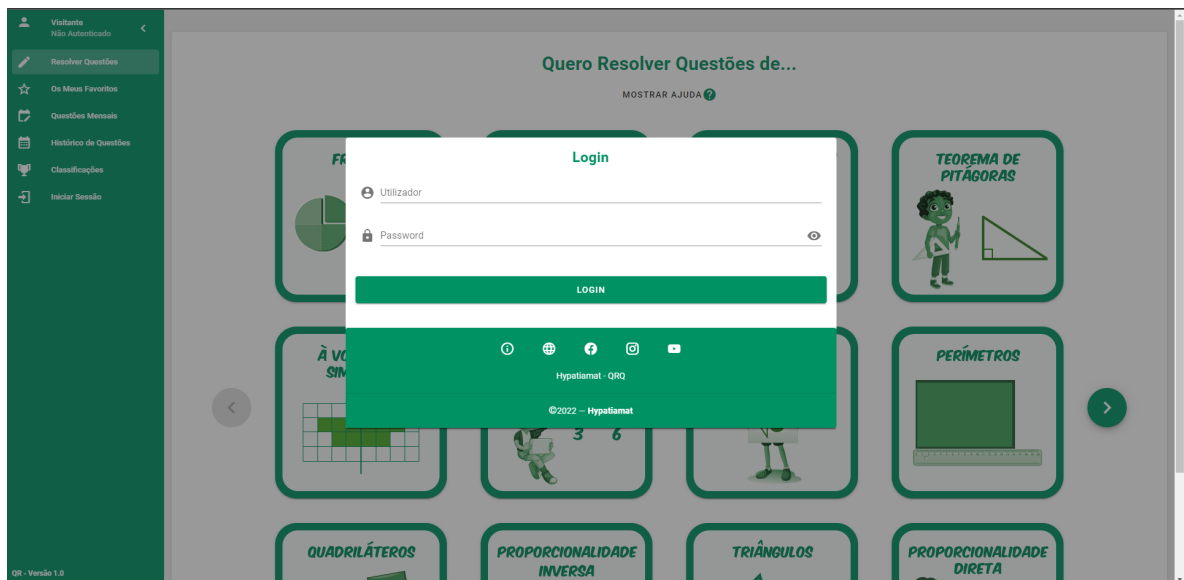


Figure 89: Authentication Interface

This modal blocks the entire website behind it and dims its color, emphasizing the authentication screen and giving the user the option to either quit it or input their login

data. Either choice will not redirect them out of the current page, it will simply remove the modal and the dimming effect from the screen.

Theme Selector

The theme selector is very similar to the current version of *Hypatiamat*, the only difference is that the images were reworked in other to fit the green and white color palette. The prototype (128) is also similar to the final version.

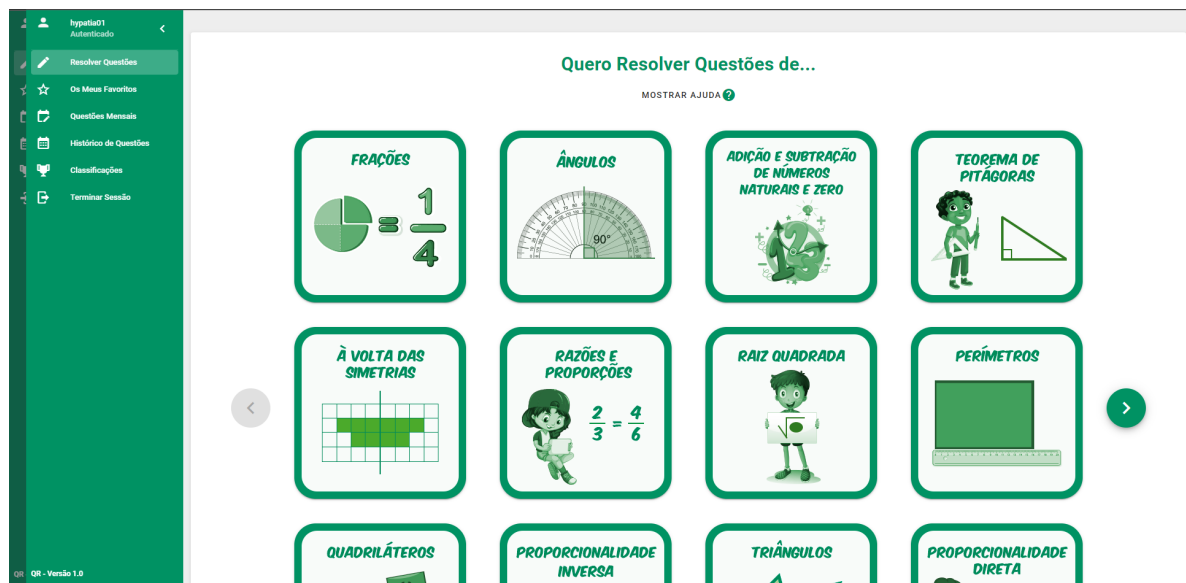


Figure 90: Theme Selector Interface

Every theme image also contains a rounded green border, for contrast and visual appeal. Note that the layout is not static, the themes shift around according to the screen size and resolution. For example, this is how it would look like in a tablet:

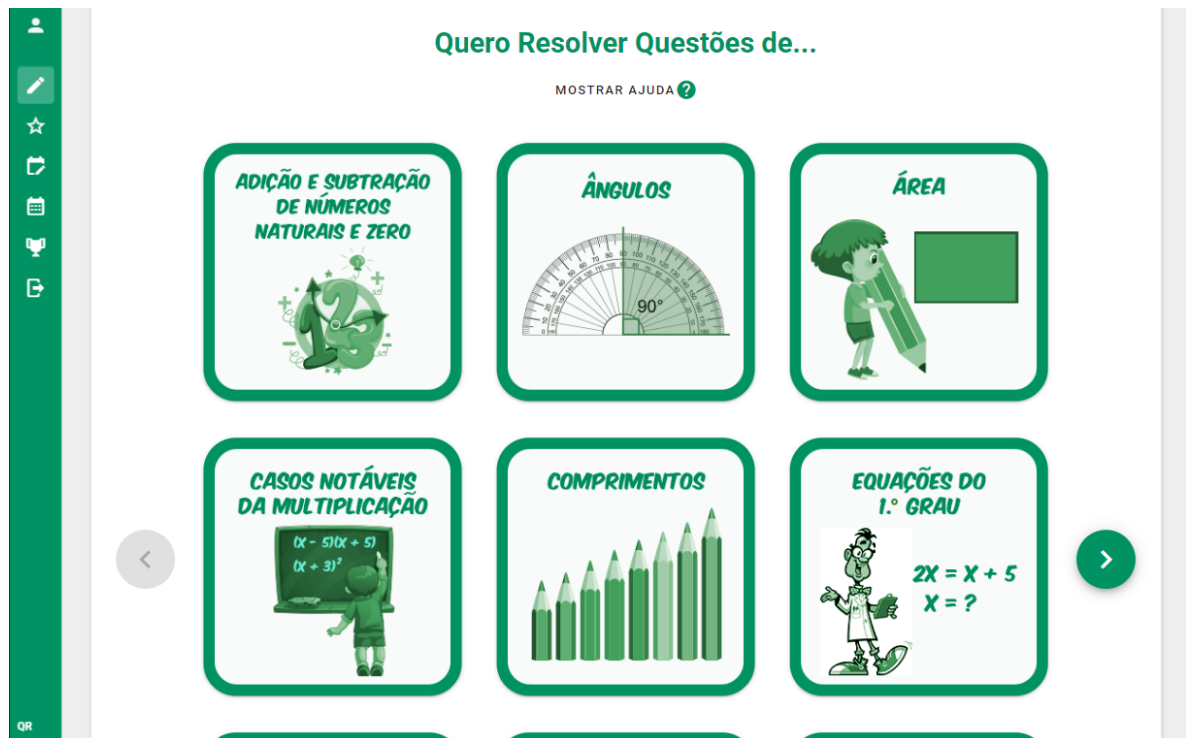


Figure 91: Theme Selector Interface (Tablet)

This way the application works seamlessly between devices, without the user having to worry about compatibility issues.

Question Solving

The final version of this interface is very similar to its prototype (figure 129), which in turn is based on the current version of the application. The main issue with the current live version is that the screen is too cluttered, which makes the sub-theme selector occupy too much space on the screen and rendering it not intuitive to use.



Figure 92: Current Question Solving Interface

Instead of having the sub-themes on the left side of the screen, it is better to list them **on top**, since the user must first choose one before being shown the questions themselves. It makes for a much cleaner interface, where the information is presented from top to bottom, there is no visual clutter in between.

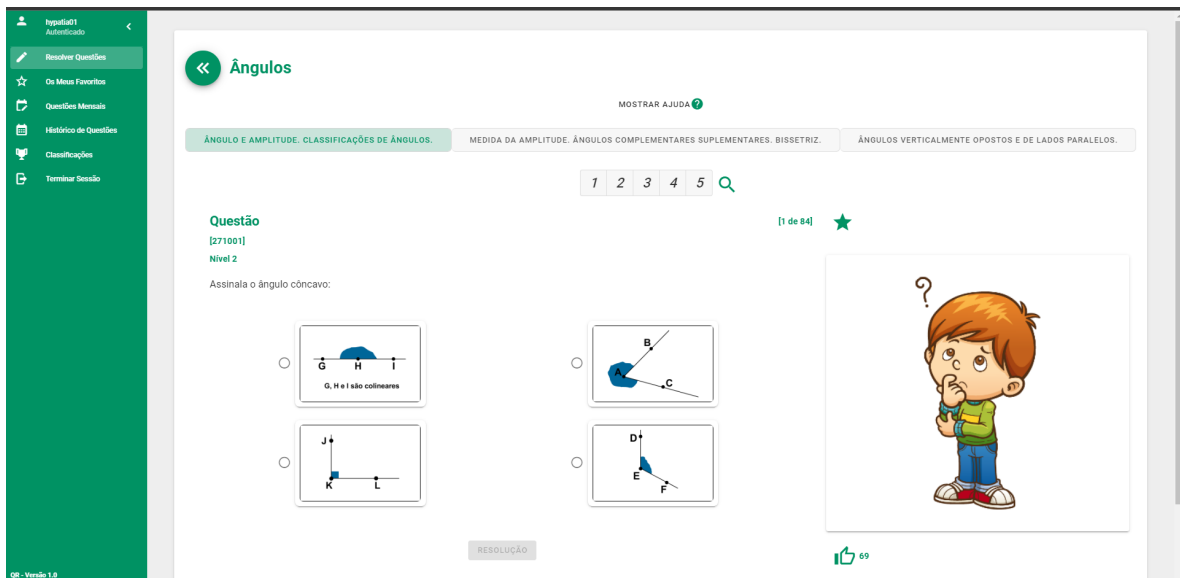


Figure 93: Question Solving Interface

At the very top of the page, there is the back button represented by two left chevrons with the theme's name next to it, allowing the user to quickly return to the theme selector if they want to.

The difficulty selector itself was slightly reworked from the original version.

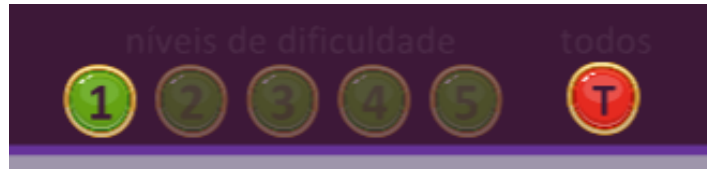


Figure 94: Current Question Solving Interface (Difficulty Selector)

The numbers 1 to 5 represent difficulty levels and the character 'T' represents any difficulty. All of these buttons can be toggled, filtering any question that is not of that difficulty level. By taking a look at these definitions, the 'T' character is redundant since if all the buttons are toggled, it essentially means that the user wants the questions of all difficulties.



Figure 95: Question Solving Interface (Difficulty Selector)

This simplifies the interface a bit and makes it more intuitive, the user simply selects all the difficulty levels they want, where they are all selected by default.

Next up, regarding the original search functionality (figure 75), it needed a rework, as it was previously mentioned. By placing a magnifying glass next to the difficulty selector (it works as a type of search, in a way, so it makes sense to group them together) that opens up a search modal.

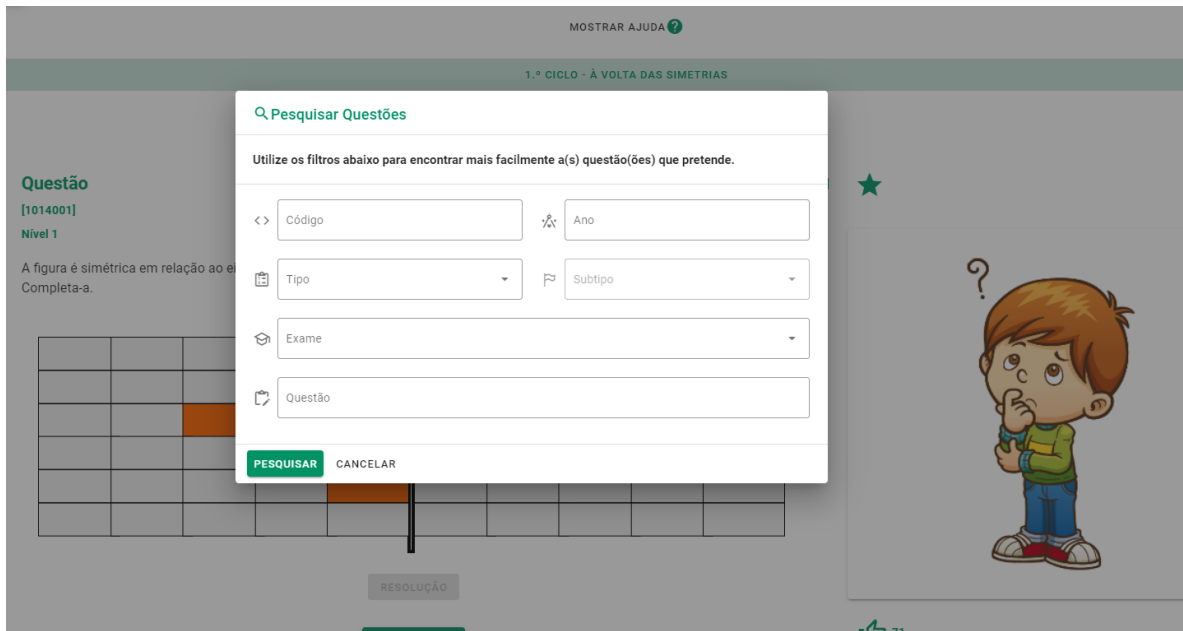


Figure 96: Question Solving Interface (Search)

The filters work exactly the same as the ones on the back-office, the user simply has to write anything they might remember on the question(s) and, if it/they belong(s) to that theme and sub-theme, it/they will appear in the same page seamlessly.

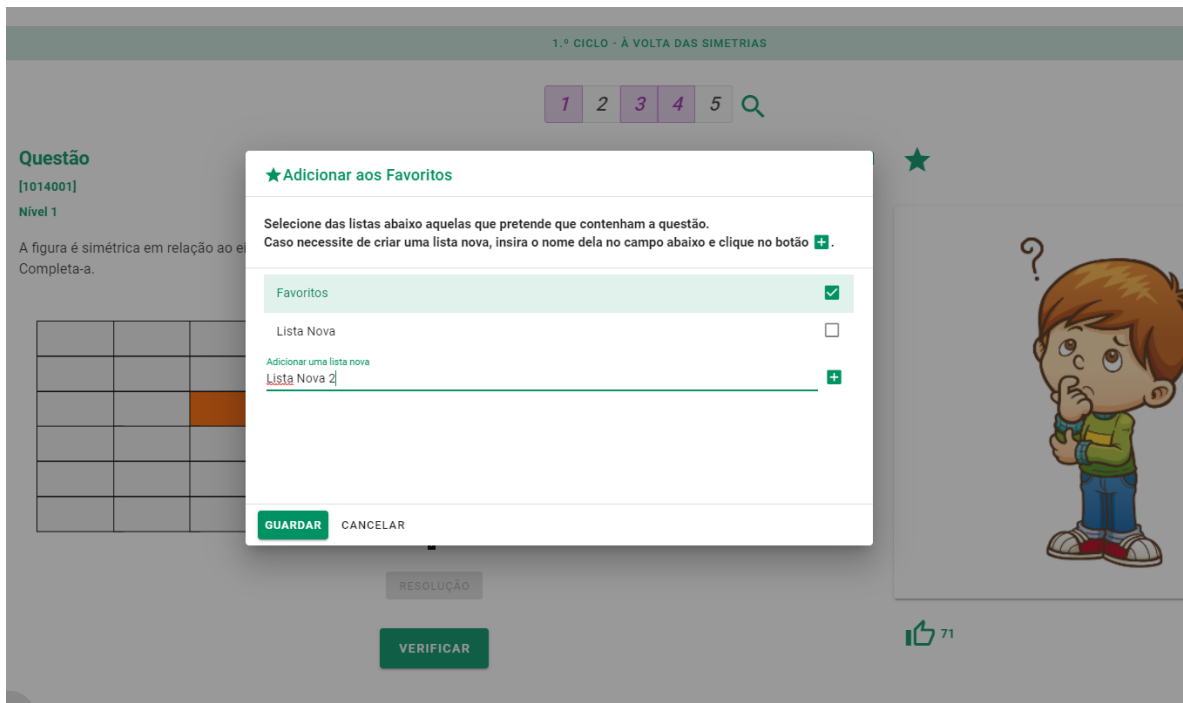


Figure 97: Question Solving Interface (Favourites)

The favourites (figure 74) also needed to be reworked due to their poor functionality and lack of customizability.

Instead of allowing the user to only use a single list, they can now create multiple lists on the fly and add the question to any of them. This menu is accessed by clicking the green star button on top of the image, just like the original version. This interface is also very similar to YouTube and its playlists, which served as direct inspiration for this interface and it is an efficient way of solving the issue.

The like button below the image was just slightly tweaked to fit the green and white theme better, its functionality and flow are kept the exact same.

The interface where the question actually appears and allows the user to solve it just has to take four different types of question into account. Every single question hides its resolution, if it has one, and only enables it if the user has gotten the wrong answer three times in a row. For every answer the user submits, it can either be correct (green) or incorrect (red) and it is displayed accordingly, along with a sound for better audiovisual feedback.

- Open-Ended - the different types of open-ended questions are not visually different, they are just background checks. What matters is the single input field that the user interacts with, via the digital keyboard. If the user submits a correct answer, the keyboard is disabled.

Questão [29 de 39]

[13035]

Nível 1

Escreva uma fração que represente a parte (do todo) pintada na figura.

1/5

-	,	0	1	2	3	4	5	6	7	8	9	Apagar			
;	()	{	}	[]	-∞	+∞	x	y	+	×			
-	0	1	2	3	4	5	6	7	8	9	a	b	/	-	=

RESOLUÇÃO

VERIFICAR

Figure 98: Question Solving Interface (Open-Ended) - Incorrect

Questão

[29 de 39]

[13035]

Nível 1

Escreve uma fração que represente a parte (do todo) pintada na figura.

The interface shows a green progress bar at the top with the fraction 1/4 written inside it. Below the bar are two buttons: a green 'RESOLUÇÃO' button and a grey 'VERIFICAR' button.

Figure 99: Question Solving Interface (Open-Ended) - Correct

- Multiple Choice - the text interface shows the answers using rows, whereas the images are divided in columns. This way, the screen space is saved more efficiently, as text answers can be long and images are always of the same size.

Questão

[32 de 39]

[13069]

Nível 2

As figuras são todas constituídas por quadrados geometricamente iguais. As figuras em que a parte sombreada não é metade de toda a figura são:

The interface shows a list of six radio button options. The first option, 'B, D e E', is selected and highlighted with a green background. The second option, 'A e B', is unselected. The third option, 'B e C', is selected and highlighted with a red background. The fourth option, 'B e D', is unselected. The fifth option, 'A, D e E', is unselected. The sixth option, 'A e D', is unselected. Below the list are two buttons: a green 'RESOLUÇÃO' button and a grey 'VERIFICAR' button.

Figure 100: Question Solving Interface (Multiple Choice) - Text

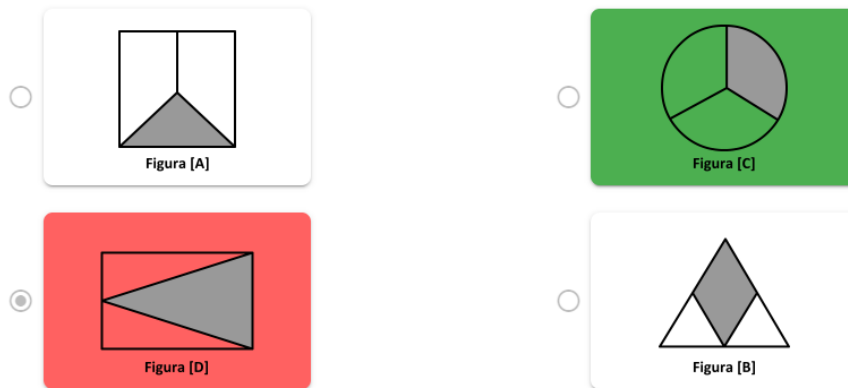
Questão

[34 de 39]

[13071]

Nível 2

Qual das opções seguintes mostra uma figura cuja parte sombreada a cinzento corresponde a $\frac{1}{3}$ da própria figura?



RESOLUÇÃO

VERIFICAR

Figure 101: Question Solving Interface (Multiple Choice) - Images

- True or False - very similar to the text multiple choice questions (two options).

Questão

[12 de 39]

[13018]

Nível 2

A Flora gastou metade do seu dinheiro na compra de um livro sobre animais. O Tomás comprou um livro sobre o corpo humano, gastando $\frac{1}{4}$ do seu dinheiro. Será possível os dois livros terem custado o mesmo?

- Sim
- Não

RESOLUÇÃO

VERIFICAR

Figure 102: Question Solving Interface (True or False)

- Symmetry - the grid may have two orientations, but they all behave similarly. If the user submits anything that does not exactly match the picture, it will be considered incorrect.

– Vertical

Questão [1 de 26]

[1014003]

Nível 1

A figura é simétrica em relação ao eixo representado.
Completa-a.

RESOLUÇÃO

VERIFICAR

Figure 103: Question Solving Interface (Symmetry) - Vertical, Incorrect

Questão [1 de 26]

[1014003]

Nível 1

A figura é simétrica em relação ao eixo representado.
Completa-a.

RESOLUÇÃO

VERIFICAR

Figure 104: Question Solving Interface (Symmetry) - Vertical, Correct

- Horizontal

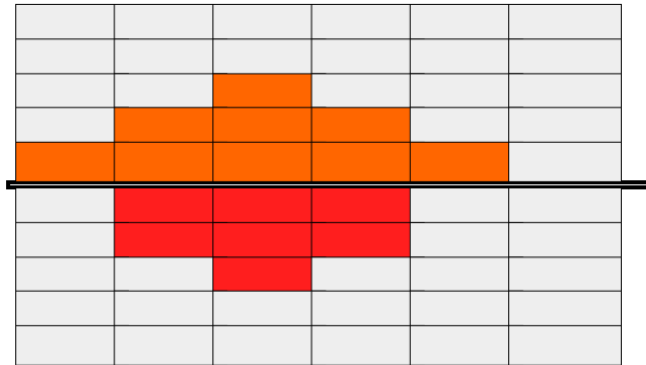
Questão

[49 de 60]

[1014049]

Nível 1

A figura é simétrica em relação ao eixo representado.
Completa-a.



RESOLUÇÃO

VERIFICAR

Figure 105: Question Solving Interface (Symmetry) - Horizontal, Incorrect

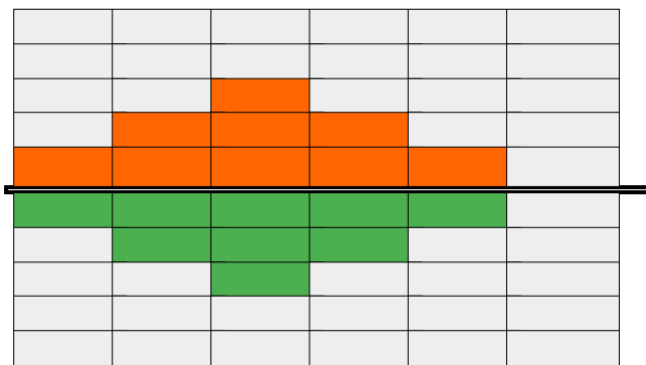
Questão

[49 de 60]

[1014049]

Nível 1

A figura é simétrica em relação ao eixo representado.
Completa-a.



RESOLUÇÃO

VERIFICAR

Figure 106: Question Solving Interface (Symmetry) - Horizontal, Correct

After the user chooses to solve the question or not, they can move onto the previous or next question by clicking the green buttons with chevrons on the bottom of the page.



Figure 107: Question Solving Interface - Next and Previous Buttons

Instead of cluttering the screen, it is better to place them at the bottom, making the page a lot clearer and better looking.

Favourites

The favourites page is very similar to the original mock-up (figure 130), as it simply needs to list all the user's favourites folders and their corresponding questions.

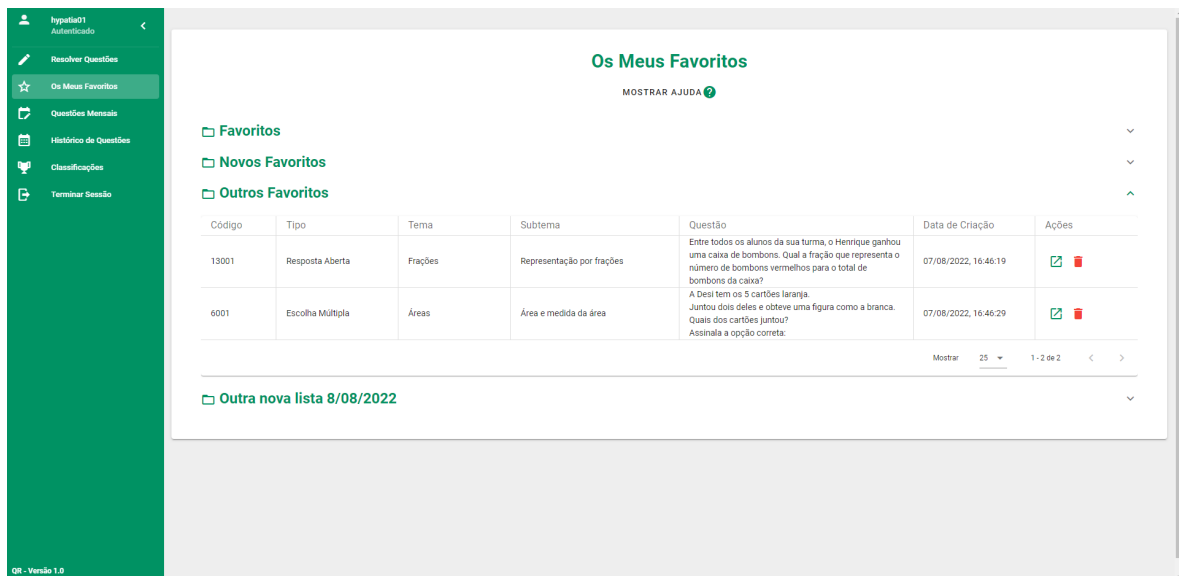


Figure 108: Favourites Interface

To make it visually more interesting, a folder icon was added to the items and the text was shifted from the center to the left.

The colors were slightly altered aswell in order to better fit the application's theme.

Each list has a table, with the same features as the ones mentioned in the back-office. However, these ones lack the preview function because they have a redirect button. Instead of viewing them inside the favourites page, the user is redirected to the question solving screen with that question opened. Next to that button is delete button, represented by a trashcan icon, responsible for removing that entry from that particular list.

This way the user has total control of their favourite's contents, multiple lists and a way to preview the questions they added.

Monthly Questions

The monthly questions are also very similar to the mock-up (figure 131), since it was already developed for the back-office and a general idea of its look was already in place.

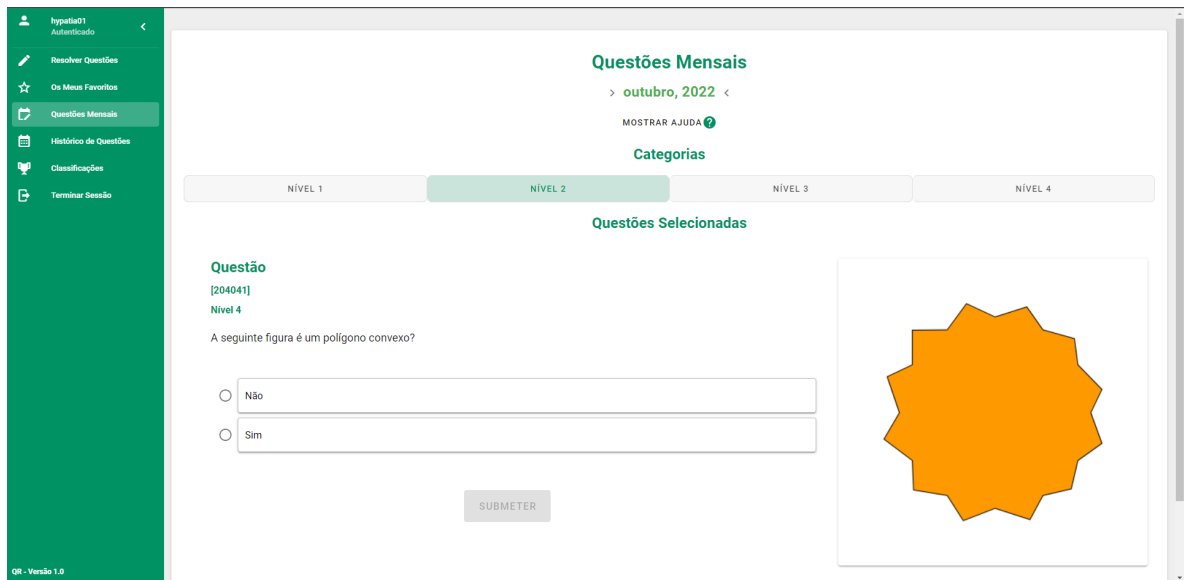


Figure 109: Monthly Questions Interface

The category selector is exactly the same as the one in the back-office and the question solving screen is exactly the same as the one already seen in the sections above, the only major difference is a title marking the current month and year, for a cleaner look.

The user may only submit their answers once for each question until the the end of the month, which is when the questions will reset, and these answers may only be to questions supported by their school grade (chapter 3.1.1).

Once the reset happens, the questions will be moved to the history of monthly questions, where the user can check if they got the answer right or wrong.

Note that these questions do not have a resolution button, since the point of these monthly questions is to incentivize the students to think about their answers before submitting, because if they do not get them right, they do not move up on the leaderboard.

Monthly Questions History

This interface is very similar to the previous one and to its prototype (132), since the mock-up was done after the development of the back-office, the monthly questions history interface happened to very close to its final form.

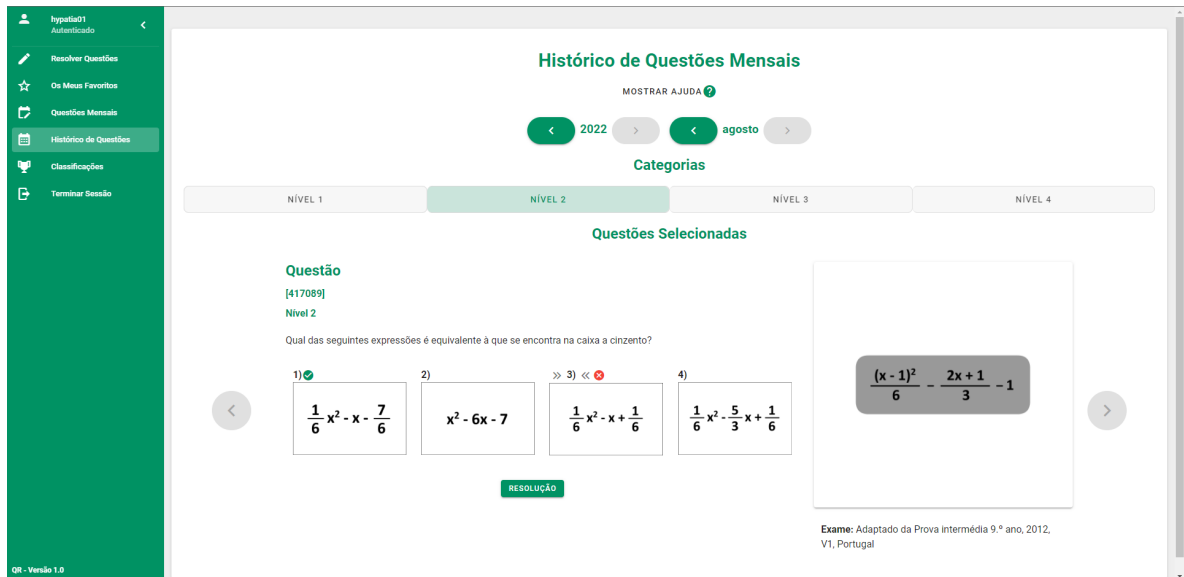


Figure 110: Monthly Questions Interface

The question preview screen is a slight rework on the one used at the back-office. The only addition is a red cross for the wrong answers that the user submitted.

This way, the users can quickly view the correct answer, what their answer was and the resolution for it, in case they need extra help.

Leaderboards

The leaderboards had some improvements done to it in relation to its mock-up (figure 133), mainly focusing on presentation and gamification for better student engagement.

The leaderboards are divided by school year and category, where the user can see where they place in relation to their peers, according to their monthly questions performance.

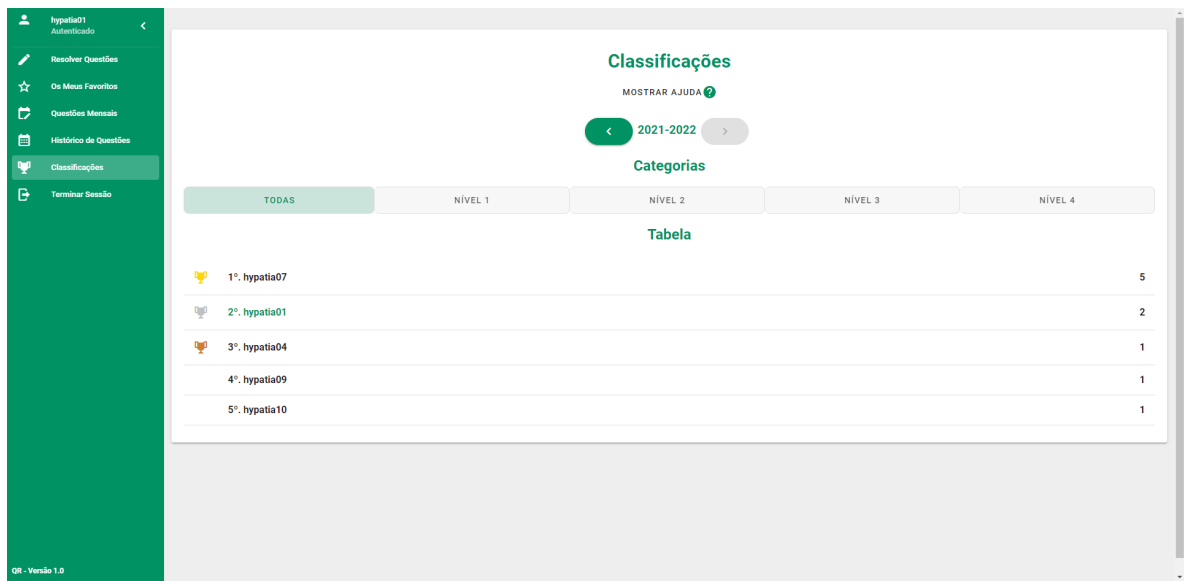


Figure 111: Leaderboards Interface

The first major change was the addition of bronze, silver and gold trophies for the third, second and first places, respectively. This adds a bit of a competition between the students, so that they have a reason to reach the number one spot.

The next was the addition of the score itself, having it hidden makes no sense because the user ends up knowing if they got the question wrong or right, it is better to simply show them as a differentiating factor.

4.6.3 Saving the Blocked User State

One of the requisites for this application was that unauthenticated users get blocked after viewing five different questions daily.

In order to achieve this, there are two different possible approaches:

- client-sided - keep a local counter on the browser and if it reaches the daily limit, block the user's access to the interface;
- server-sided - save the user's IP address after the limit has been reached, needs constant polling after a question is loaded to ensure the limit has not been reached yet.

The second option is not ideal for this particular use case, despite it being the safer approach, making the client-sided approach the better fit.

The first reason is due to security risks not applying for a question solving application, the authentication is optional and the questions are free to begin with, there is no need to even consider it when everything is publicly available.

The second reason is that the target audience are primarily young students, many of them with limited technological knowledge, let alone understanding how a full-stack web application works. Therefore, implementing complex server-sided solutions is a much worse strategy than implementing simple and yet effective client-sided blocking strategies, which can be easily done with Vue and Vuex.

In addition to the user state that was developed for the back-office, that will be reused for this application, there also needs to be a new module dedicated to this blocking procedure.

The first variable that needs to be taken into account is the viewed questions list, which needs to work as a set, ensuring that no repeated elements are contained within it.

The second is the limit itself, defaulted to and it is only used to limit the set's size and verify whether the cap has been reached or not.

The last one is the date when the block will be removed, if it exists. Since this information is persisted in the local storage, when restoring the Vuex timer instance it needs the information to restart it.

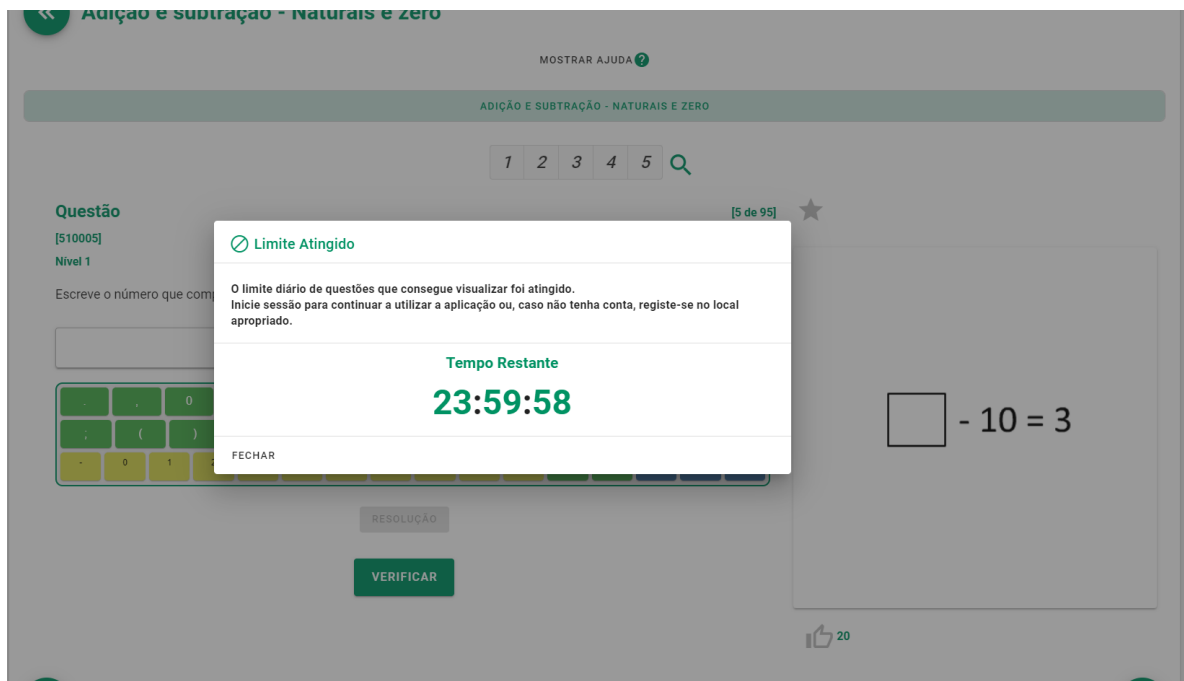


Figure 112: Timer Interface

The flow is also simplistic, whenever the user views a question, it is added to the set. If the limit has been reached, it simply blocks the user from clicking the previous and next buttons and displays the blocking modal instead, visible in the picture above.

This concludes the development of the front-end server, all that remains is deploying both applications, which is the focus of the next chapter.

DEPLOYMENT

The *Hypatiamat* services are hosted in *PTServidor*, a portuguese company that provides web services and domains.

Since all the new micro-services on the platform need their own domain and address, they are all managed by an instance of a NGINX server acting as a reverse proxy for all of them. In this particular case, 4 new domains need to be created, namely the two back-end and front-end servers for each application. Evidently, these are web applications developed taking only *HTTP* requests in consideration, so, it is only necessary that these domains provide these services, resulting in the following *URLs* being made available:

- **Question Submission Back-Office**
 1. <https://apibckqr.hypatiamat.com> - back-end server;
 2. <https://bckqr.hypatiamat.com> - front-end server,
- *"I Want To Solve Questions About..."*
 1. <https://apiqr.hypatiamat.com> - back-end server;
 2. <https://qr.hypatiamat.com> - front-end server.

Naturally, tests were done before being placed in production, in order to ensure that the applications' existence would not cause any damage, especially to the databases.

Rolling out the current application versions into a production build was made easier by the GitHub repositories where their source code was hosted. Essentially, there is one for each server and there is a copy from all of them saved locally, so they can be easily updated after any future change.

The deployment process for these entities (back-end and front-end) differs from one another, so they will be explained in different sections.

5.1 BACK-END SERVERS

There are many ways to deploy an *API* server, however, *Hypatiamat* is currently using PM2, a process manager package for production usage, for all its micro-services built with Node.js.

This tool allows for better maintenance and its features include a *CLI* interface, auto-start scripts, load balancing, logging, and many others, which is a great way to ensure that the applications work flawlessly while they are live.

The deployment process is identical to both back-end servers, and they both include creating a new PM2 instance for each one.

Whenever a new production build is ready to be deployed, the first step is updating the repositories to the most recent version. On a typical Node.js server, it would suffice to execute the following command on the file that contains the code related to starting the *HTTP* server:

```
pm2 start server.js
```

However, this particular file does not exist in Strapi and, as such, PM2 can not start a new instance on it. However, due to popular demand, Strapi's creators already sorted this issue in previous versions, but it has to be done manually by the developer. The solution is creating a new JavaScript file inside the server's root folder, where the Strapi server instance is manually started, which has the exact same behaviour as if it was started in the command line using *NPM* or Yarn.

```
1 const strapi = require('strapi');  
3 strapi().start();
```

This way, the previous command can now be executed using this file (assuming it was named 'server.js', just like the example above), and the server can now be a PM2 instance and thus benefit from its features and interfaces. After being created, it will be given a new identifier which, as an example, let it be the number 0 going forward.

PM2 does not copy the current files to a new folder in order to create a new server instance, it simply saves the folder where they are located. This way, whenever the back-end needs to be updated, it is not necessary to completely remove the previous version from PM2, updating the local files to the most recent version and restarting the instance is the most efficient way to complete the process:

```
1 pm2 restart 0
```

This process is relatively simple to execute and can be done in a manner of minutes, which is a great time-saver considering PM2 and Strapi do all the heavy-lifting in the background.

5.2 FRONT-END SERVERS

Both front-end applications were built using Vue.js, which works in a vastly different manner than Strapi. A feature it provides is compiling the entire build as **static** files, meant to be served by another *HTTP* server. By executing the following command inside the project's root folder, a new folder named 'dist' will be created that contains these files:

```
1 npm run build
```

When examining the folder structure, it has multiple folders divided by the data types used in the application, such as 'css', 'img', 'js' and 'media' and, for each one, it contains multiple files related to the folder they are in. For example, inside 'js' there is the entire website's functionalities and inside 'css' there are the entire Vue.js and custom user-made *CSS* files. There is also one single short 'index.html' file where it has these folders linked correctly in the header for the browser to fetch when appropriate.

```
1 <!DOCTYPE html>
  <html lang="en">
3
  <head>
5     <meta charset="utf-8">
     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width,initial-scale=1">
     <link rel="icon" href="/favicon.ico">
9     <title>Hypatiamat - Quero Resolver Quest es de...</title>
     <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto
11 :100,300,400,500,700,900">
     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@mdi/font@latest/
css/materialdesignicons.min.css">
     <link href="/css/app.1f9a827.css" rel="preload" as="style">
13     <link href="/css/chunk-vendors.3d7720a4.css" rel="preload" as="style">
     <link href="/js/app.290ebf01.js" rel="preload" as="script">
15     <link href="/js/chunk-vendors.68f663a2.js" rel="preload" as="script">
     <link href="/css/chunk-vendors.3d7720a4.css" rel="stylesheet">
17     <link href="/css/app.1f9a827.css" rel="stylesheet">
  </head>
19
  <body><noscript><strong>We're sorry but frontend doesn't work properly without
    JavaScript enabled. Please enable it to
21     continue.</strong></noscript>
     <div id="app"></div>
23     <script src="/js/chunk-vendors.68f663a2.js"></script>
     <script src="/js/app.290ebf01.js"></script>
25 </body>
```

```
27 </html>
```

As mentioned previously, *Hypatiamat* relies on NGINX as a reverse proxy, so, whenever a user attempts to access <https://qr.hypatiamat.com>, it simply returns the *HTML* file and everything else is the user's responsibility. Assuming the user is on a typical browser, once they have the 'index.html', it will scan the links inside the *HTML* file's headers and send requests to fetch them from the NGINX server and, once that is complete, it saves them locally until they are changed (caching), which avoids repeating this costly process in the near future. The following code is a simple NGINX configuration parameter that allows this process to happen:

```
1 location / {
2     try_files $uri $uri/ /index.html;
3 }
```

After repeating this process to both front-end servers, they are now ready to be used in production and can fully consume the *API* endpoints on the back-end servers in order to fully render the website's pages with the necessary data.

5.3 IMAGE SERVER REDIRECTS

The questions in the database, as seen in previous chapters, have two optional fields that contain the *URL* to two images, namely 'resolucao' and 'figura' (see figure 11).

The current *Hypatiamat* image server is essentially a static file server, where the directory it operates on contains all the images for the applications and is updated **manually**, whenever a new image must be added.

This is where a major problem emerges, and that is that if some application wants access to add a new image to this folder, it must somehow have permissions to make **local** changes (in the same machine) to it, because there is no upload endpoint due to the nature of these types of servers and security concerns.

This is important especially on the back-office, where editing, adding or removing the images is a core feature and if the situation was left the way it is only removing would be possible, since that would entail simply clipping the *URL* from the fields.

So, the possibility of using that static server is completely out of the question. The only alternative is creating a new one where the back-office has total control on its contents and this is easily achievable using Strapi's built-in upload manager. By default, Strapi has

endpoints that can be used to upload *HTML* form data and if by chance it contains any file, it will be stored both in the database (only its metadata) and in the server's physical drives, specifically in the public folder inside Strapi, where they will behave as static resources and served as such. This way, it is possible to edit, remove or add images that are owned by that Strapi instance.

The solution then is pretty simple, right? If a new image needs to be added, simply save it in the back-office's back-end server and use its *URL* in the field(s). No, this is not quite true and that has to do with how the *URLs* are stored.

Starting with the 'figura' field, an example of its contents could be "perimetrosN2/perimimg43.png". This means that the address is not fully saved, only the folder and file name for that particular image, since originally it was assumed that there would only be one address where the images would be stored, namely <https://www.hypatiamat.com/imagens/>. By concatenating these two elements together, the final *URL* that contains the image is <https://www.hypatiamat.com/imagens/perimetrosN2/perimimg43.png>.

This also happens in the 'resolucao' field where an example of its contents could be "perimetrosN2/solperimetros3.png". For this one, it also has a permanent address by default and that is <https://www.hypatiamat.com/imagens/propresolucao>, where it only differs from the previous case on its "propresolucao" extension. The concatenation results in <https://www.hypatiamat.com/imagens/propresolucao/perimetrosN2/solperimetros3.png> and it is where the image is located at.

Consequently, for every other *Hypatiamat* application, this means that any concatenated *URL* would lead to the places mentioned above and if the image is not on that static server, it would simply be considered as non-existent when it clearly is. They all follow this rule, so attempting to change the format in which the information is saved or storing the new image *URLs* from the back-office in their entirety are solutions that would not work, they would absolutely break every other application and that is something that must be avoided at all costs.

Consider that a new submission was created and a new image successfully added to the back-office's static server and the information that is stored in the database would be the following instead:

- "bckqr/1.png" - for the 'figura' column;
- "bckqr_r/2.png" - for the 'resolucao' column.

Naturally, this by itself does not solve anything, the result is still the same as before. However, if **redirects** are introduced, that is a different story. Note that no other folder or file exists with these particular names ("bckqr" and "bckqr_r") in that static server so there are no conflicting routes.

If the NGINX instance is configured in a way such that any incoming requests containing these two keywords are automatically redirected to the back-office instead, this problem

would be fixed for any application, whether new or old, because this redirection process happens in the background and works as if nothing changed in the static image server.

The picture below shows a brief overview on what this redirection process looks like:

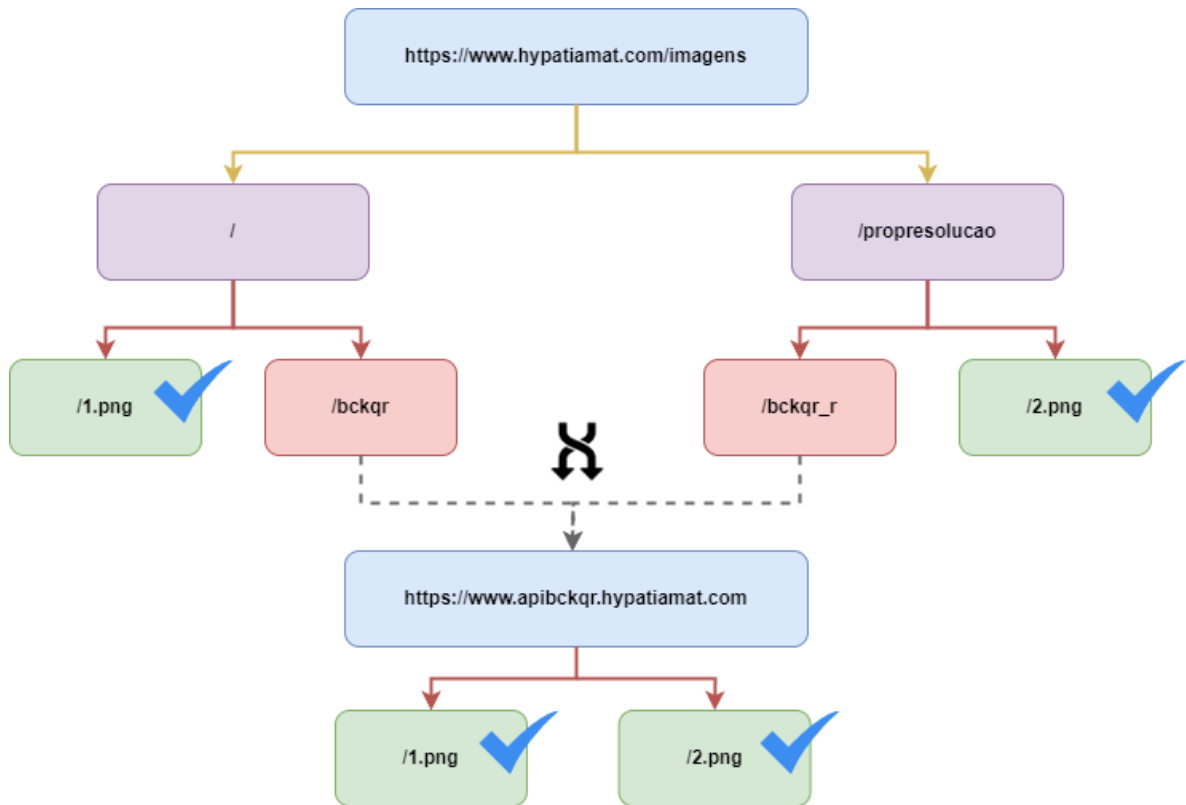


Figure 113: Image Server Redirects Diagram

After this implementation, all applications were tested and they worked correctly with this new method of serving images, leaving the teachers the possibility of executing *CRUD* operations on any image at will and without breaking any existing application.

This deployment method brought forth some challenges, related to how the NGINX instance operates in the background, causing troubles in the Strapi's default documentation endpoints, which will be the focus of the next chapter.

DOCUMENTATION

The *API* endpoints might be used by other developers, in the future, for integration purposes in their applications or scripts. Thus, creating a proper documentation page for both back-end applications where the routes, parameters, responses types, response bodies and so on is important in this regard.

Once again, Strapi makes this entire process easier and quicker since it has a downloadable plugin that automatically integrates Swagger in it. Swagger uses the OpenAPI specification for creating an interface where it is possible to send RESTful requests to consume the *API* endpoints without any external tool, effectively creating a quick and friendly test environment.

On a side note, this paper will not go into a lot of detail on how Swagger operates, as that can be a dissertation by itself, only how it works when integrated with Strapi and how to properly tune it to the application's needs.

As soon as the plugin is installed, it will be generated inside every folder that contains the *API* routes (see chapter 3.5.2) a new sub-folder called 'documentation'. It has sub-folders aswell, one for each documentation version, which is an useful feature for more complex applications, where the default one is 1.0.0 and the only one that will be used for both back-ends.

The version folder contains a single *JSON* file named after the Strapi collection, i.e 'es-cola.json', which is where the route information generated on the install is stored. Naturally, this file is incomplete sometimes, it may not have the request body types, status codes or even request response types, it is up to the developer to complete it. Since these files are generated every time the server is restarted, any change is not kept if done inside them, so, Strapi defined a way for the developer to override these default values on their own volition. To achieve this, a new folder named 'overrides' needs to be created inside that version folder, containing a *JSON* file named exactly the same as the previous one and containing the fields that need to be updated, added or removed.

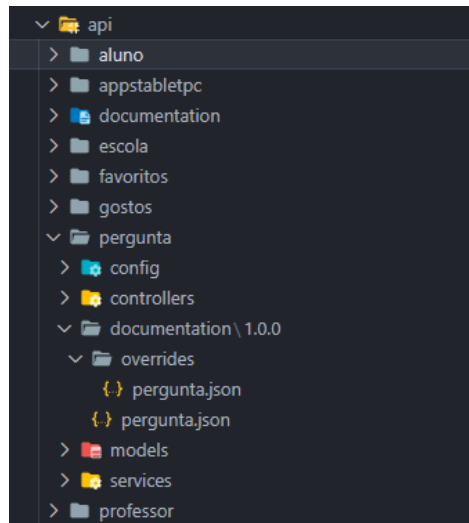


Figure 114: Documentation Folder Example

The following *JSON* excerpt is taken from the 'overrides/pergunta.json' override file, where the response body of one particular route was defined to contain an array of questions (the data type is automatically defined in the original document) instead of the default generated one.

```

1 {
2   (...)
3   "/perguntas/diretas": {
4     "get": {
5       "deprecated": false,
6       "description": "Retorna as perguntas no formato original.",
7       "responses": {
8         "200": {
9           "description": "response",
10          "content": {
11            "application/json": {
12              "schema": {
13                "type": "array",
14                "items": {
15                  "$ref": "#/components/schemas/Pergunta"
16                }
17              }
18            }
19          }
20        },
21        (...)
22      }
23    }
24  }

```

The plugin then compiles the information on all of these *JSON* files and renders them as a single *HTML* file, which Strapi serves as a static resource when the documentation endpoint is accessed, located in 'extensions/documentation/public/index.html'. It contains the link to other static resources, namely the Swagger framework JavaScript, *CSS* and other files, which can then be downloaded separately by the client's browser, if they are using one.

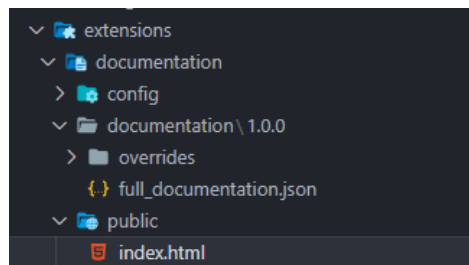


Figure 115: Documentation Final HTML File

If everything works correctly, after accessing the '/documentation/v1.0.0' route on the back-end, the documentation should now be properly visible and ready to provide its services to the user accessing it.

However, due to the way NGINX is configured, this does not work due to the static resources the *HTML* file links to not being public since they are installed as *NPM* modules. The solution for this is moving all these files to the correct public folder and swapping the links inside the *HTML* to the new ones. Since this file is reset every time the server restarts, there is also the need to create a new *API* endpoint that provides the modified *HTML* file with the correct links.

This new documentation *API* endpoint will be located at both '/documentacao' and '/documentation', for better flexibility. The JavaScript code below is the asynchronous function called when these are accessed and it is as simple as doing a quick find and replace before sending it to the user:

```

module.exports = {
  2   index: async ctx => {
      let html = fs.readFileSync('extensions/documentation/public/index.html', '
      utf8')

      4   html = html.replace(/plugins\/documentation/g, 'swagger') // only the links
      will be affected

      6   return html
      8   }
};

```

The "swagger" replacement relates to the folder 'public/swagger' inside the Strapi project root, where the aforementioned files are stored. After this is complete, the documentation is now ready to be used in its entirety.

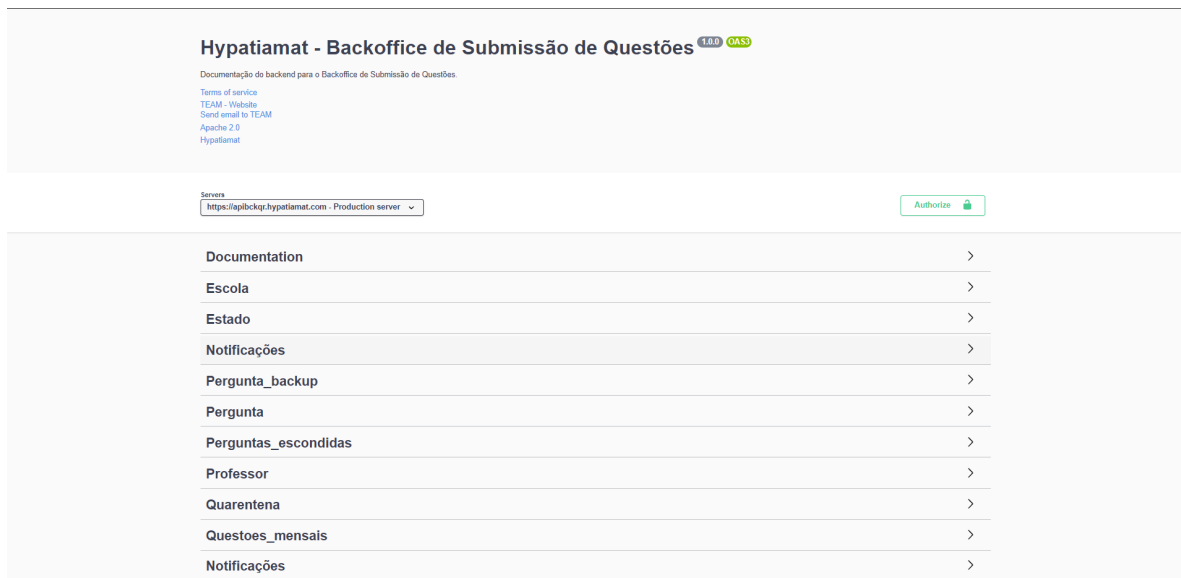


Figure 116: Documentation for the back-office



Figure 117: Documentation for the "I Want To Solve Questions About" application

Since the back-end servers are deployed for production, they currently have a public *URL* that may be accessed for public use and Swagger offers ways for users to quickly test the *API* endpoints. However, some of these routes are locked by the *JWT* authentication layer

and may not work if the requests are sent via unauthenticated sources. In order to unlock them, the user must first obtain a valid token, otherwise they can only use the public routes.

They can be accessed in the following links:

- **Question Submission Back-Office:**
 - <https://apibckqr.hypatiamat.com/documentation>
 - <https://apibckqr.hypatiamat.com/documentacao>

- *"I Want To Solve Questions About...":*
 - <https://apiqr.hypatiamat.com/documentation>
 - <https://apiqr.hypatiamat.com/documentacao>

With complete and readily accessible documentation, any developer that requires any of these routes should hopefully have a better time figuring out the details behind every request and response, which should save them time they can use developing other features.

This marks the end of the development for both applications, but there are some conclusions to be taken from all the work done before, analyzed in the next chapter.

CONCLUSION

This dissertation focused on the development of two full-stack applications, each with their own unique quirks and challenges.

The complex nature of this task was not seen as an impediment but as a challenge, as stagnation is never a good way to approach life and something that should not be taken lightly especially in the information technology fields.

From the technology selection to implementing all of the requisites demanded by the *Hypatiamat* team, this paper goes deep into every topic in order to ensure that every minute detail is fully explained and understood.

Deploying the applications was also an interesting challenge because of the static image server that exists as the foundation for *Hypatiamat's* existing applications, it was a great way to learn NGINX and proxying incoming client requests.

They both made it into production and are available under the *Hypatiamat* domain for public use, currently being used and tested by the teachers and students alike.

The following sections contain what was the outcome of the project and its consequences, future features that would be great additions and would complement the applications well and, finally, the closing thoughts and personal notes on the development of the project.

7.1 OUTCOME

The development of these two applications, quoting the owner himself, "**exceeded expectations**".

The back-office was approved with flying colors by every member that has tried it, where they praised how clean-looking and intuitive the interface is and the bug-free environment they experienced.

The "*I Want To Solve Questions About...*" application was also approved unanimously by them and it was seen as a major improvement over the existing one, with tons of new features and refinements done to the user experience.

Now the staff teachers do not have to worry about breaking the existing applications or making mistakes due to managing the data manually inside the database, they can simply

open the website and they will be guided step by step in order to ensure that everything aligns correctly and seamlessly. Using the two-step verification process, they can ensure that no question is approved if it has a mistake of any kind and they can always be removed or edited if they do on a later date. Automating this process is not only a time-saver but also prevents them from burning out due to the tedious process of doing everything manually.

Around September 2022, the new school year will begin and that means that a new wave of students and existing ones will hit the website and use it profusely and in depth, which is when it will be properly tested. This does not mean the application was not tested while in development and before being placed in production, it is both easier and quicker to find mistakes or errors when the application is being used by a larger number of people. The *Hypatiamat* team expects it to hold well on this larger scale, which is another great goal accomplished.

This means that now thousands of Portuguese-speaking students will now be able to develop their Maths skills, for free, which is both impressive in terms of scale and laudable due to its objective of fighting the low success rate in Maths in order to ensure a brighter future for them, and all credit goes to *Hypatiamat* in that regard.

Overall, since all the requested features were implemented, along with some extra features deemed important when developing and considering that no complaints emerged for the final builds of both applications, it is hard to consider this project anything but a **resounding success**.

7.2 FUTURE WORK

No software piece is ever perfect, this is proven time and time again in this industry where no matter the headcount of developers working on a project, there is always something not working correctly or lacking in features. Most of the time it is not due to incompetence or lack of skill, it is due to tight deadlines and the competition to be the first to enter the market and gain a substantial margin on the remaining competitors.

This dissertation's projects are no different in that regard. Despite them having all the requested features and some extra ones that felt necessary and useful, they still have a high potential for expansion since they only scratch the surface of the gamification features that they can support. Unfortunately, there is not enough time to implement them, but it is an interesting exercise to look at what these applications could achieve in a future revision.

Starting with the back-office, the first possibility for improvement is on the themes and sub-themes selection menus when creating a new submission. The way it currently works is it simply fetches **existing** data from the database and only allows the user to select from those lists, ultimately not giving them the freedom to edit or add new themes/sub-themes. Adding a new interface that operates on a tree view (think of a file explorer but with

only one level of nested elements) using the taxonomic classification of each theme and sub-theme would be a great solution for this problem as it would allow the user to visualize the alterations they are doing on them.

Additionally, another great feature that could even be its own separate project is exporting and importing the questions data in different formats. For example, this would give the user the possibility to automate the creation of repetitive questions using other external scripts or applications that would be saved in an easily parsable text file such as *XML* or *JSON*. Naturally this would have to be made user-friendly and that is where most of the time would be dedicated to.

For the *"I Want To Solve Questions About..."* application, the main areas for improvement relate to giving the user systems to track their progression. This could be done using achievements or milestones, i.e, number of questions solved, daily login streaks, monthly questions completion ratio and so on. This would also give the leaderboards page a purpose other than vanity, as it would reward users in a more permanent manner for their daily/monthly presence.

Another thing the application lacks at the moment is audiovisual feedback. Currently it only plays a sound whenever the user introduces a right or wrong answer. Since it is primarily aimed at young people, it is important to keep them engaged with audiovisual stimuli and that includes notifying them when they skip to the next question, when the question-blocking timer begins, when it ends, etc. Whenever a pop-up appears that shows the user completed a question successfully, it should be more satisfying with animations, slide-ins, fade-outs and the sorts.

Naturally, there are infinitely more features that can be added but these are the ones that fit the projects better and would effectively improve the usability and dynamism of them as a whole.

7.3 CLOSING THOUGHTS

The projects required a continuous study and research as they were very demanding in terms of technical expertise and knowledge, which was an humbling experience and great for personal development, both as a software engineer and a person.

Full-stack development requires extensive knowledge of front-end, back-end and database management systems. Having total control of the outcome as a solo developer is both terrifying, since the end product might not be anywhere near what was originally envisioned, and ensuring since you have total control of how every piece of the stack turns out in the end and it does not depend on other teams. This project highlights that with enough practice and study, amazing results can be achieved.

It is a great relief that not only the project as a whole was a success, but also that it actually contributes to something meaningful. Having a positive impact on the world, no matter how little or marginal it is, is an amazing achievement that brings more satisfaction than any monetary compensation would.

Hopefully this project serves as inspiration for any future developer that wishes to delve a bit into full-stack engineering and its intricacies. For them and to any other, never forget: **persistence is key.**

BIBLIOGRAPHY

- Cloudflare. What do client side and server side mean? client side vs. server side. <https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/>. (Accessed on 17/01/2022).
- Hypatiamat. Hypatiamat: Apresentação. <https://hypatiamat.com/apresentacao.php>. (Accessed on 28/09/2021).
- Marin Kaluža and Bernard Vukelic. Comparison of front-end frameworks for web applications development. *Zbornik Veleučilišta u Rijeci*, 6:261–282, 01 2018. doi: 10.31784/zvr.6.1.19.
- Arpana Karanjit. Mean vs. lamp stack. https://repository.stcloudstate.edu/csit_etds/11, Maio 2016. (Accessed on 20/01/2022).
- Marca. Colégio marista apresenta programa de internacionalização do ensino. <https://www.metropoles.com/conteudo-especial/colégio-marista-apresenta-programa-de-internacionalizacao-do-ensino>, Novembro 2021. (Accessed on 24/01/2022).
- Catarina Martins. Guia para um exame sem ansiedade. <https://observador.pt/2014/05/19/guia-para-um-exame-sem-ansiedade/>, Maio 2014. (Accessed on 18/01/2022).
- MDN. Window.localStorage, 09 2022. URL <https://html.spec.whatwg.org/multipage/web-storage.html#dom-localstorage-dev>. Accessed on 07/10/2022.
- Miracle Onyenma. Understanding and using relations in strapi, 05 2022. URL <https://strapi.io/blog/understanding-and-using-relations-in-strapi>. Accessed on 27/08/2022.
- Ricardo Pinto. As aplicações hipermédia podem promover o sucesso escolar e a autorregulação da aprendizagem? análise da eficácia de uma aplicação hipermédia. <http://hdl.handle.net/1822/35846>, Dezembro 2014. (Accessed on 21/01/2022).
- Md Rahman. Cloud computing technology. 12 2021. URL https://www.researchgate.net/publication/357071610_Cloud_Computing_Technology. (Accessed on 22/01/2022).
- Redação. Escola básica de prado realizou o ii campeonato de cálculo mental hypatiamat de prado. <https://semanariov.pt/2019/06/06/escola-basica-de-prado-realizou-o-ii-campeonato-de-calculo-mental-hypatiamat-de-prado/>, Junho 2019. (Accessed on 19/01/2022).

- Statista. Market share held by leading desktop internet browsers in the united states from january 2015 to december 2021. <https://www.statista.com/statistics/272697/market-share-desktop-internet-browser-usa/>, 2021. (Accessed on 03/02/2022).
- Swoop. Token-based authentication: How to optimize your website, 06 2020. URL <https://swoopnow.com/token-based-authentication/>. Accessed on 07/09/2022.
- João Vieira. Hypatiamat: Especificação e implementação de um componente para gestão de trabalhos de casa, 2021.
- Ruihan Wang and Zongyan Yang. Sql vs nosql: A performance comparison. <https://www.cs.rochester.edu/courses/261/fall2017/termpaper/submissions/06/Paper.pdf>, 2017. (Accessed on 01/02/2022).
- Matthew West. *Requirements Specification*, pages 187–196. 12 2011. ISBN 9780123751065. doi: 10.1016/B978-0-12-375106-5.00015-4.

MOCK-UPS - QUESTION SUBMISSION BACK-OFFICE

- Dashboard

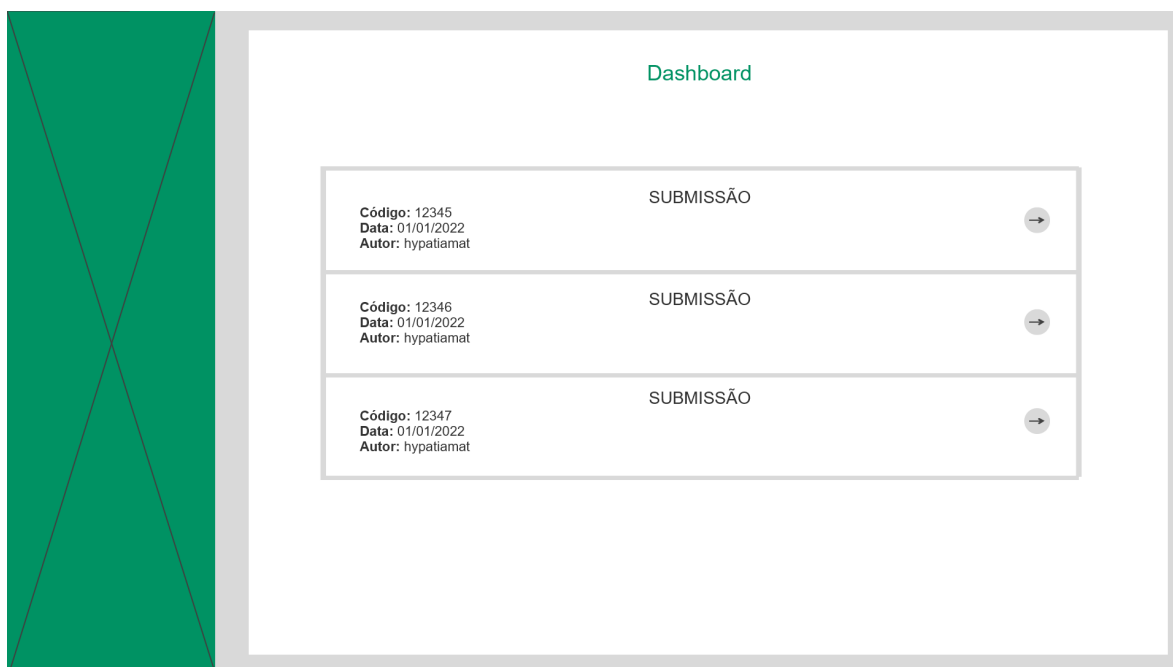


Figure 118: Question Submission Back-Office - Dashboard Mockup

- New Submission Form

Formulário de Questão

Questão

Tema Subtema

Exame

Ano Dificuldade Tipo

Resposta 1

Resposta 3

Resposta 5

Resposta 6

Figure 119: Question Submission Back-Office - New Submission Form

- Approved Questions

Lista de Questões Aprovadas

Tema Subtema

Mostrar mais...

ID	Descrição
20113	A seguinte figura...
(PRÉ-VISUALIZADOR)	
40215	Para a seguinte equação...
60215	Para a seguinte equação...
60216	Indique quais...
83410	Das seguintes figuras...

Figure 120: Question Submission Back-Office - Approved Questions

- Active Submissions List

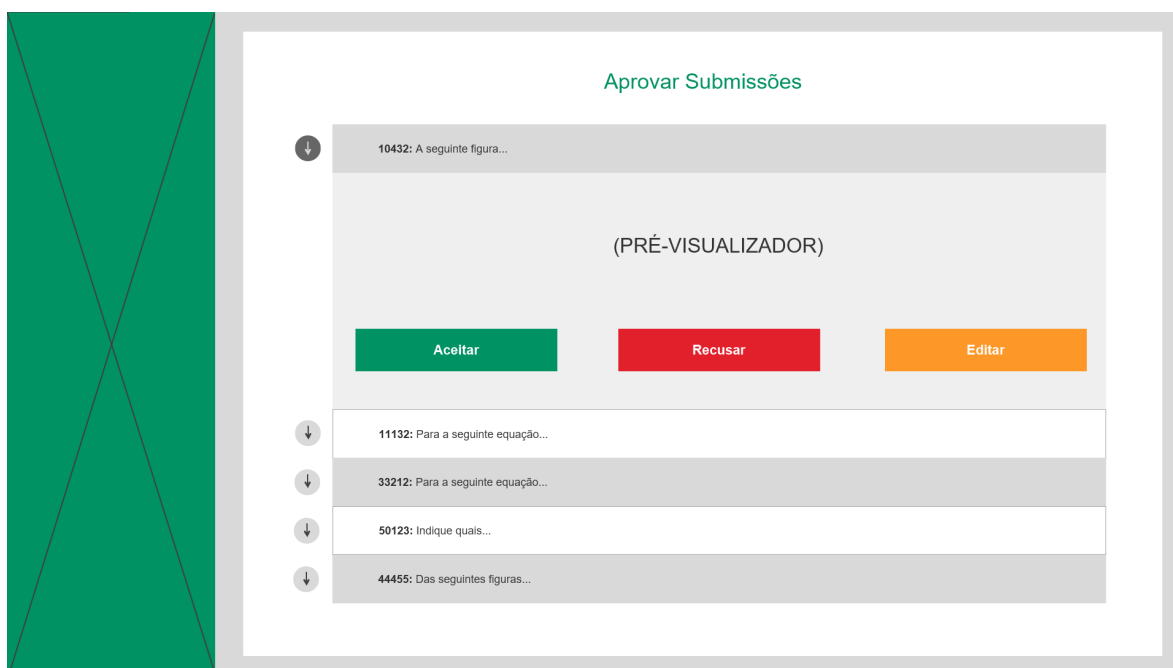


Figure 121: Question Submission Back-Office - Active Submissions List

- Hidden Questions

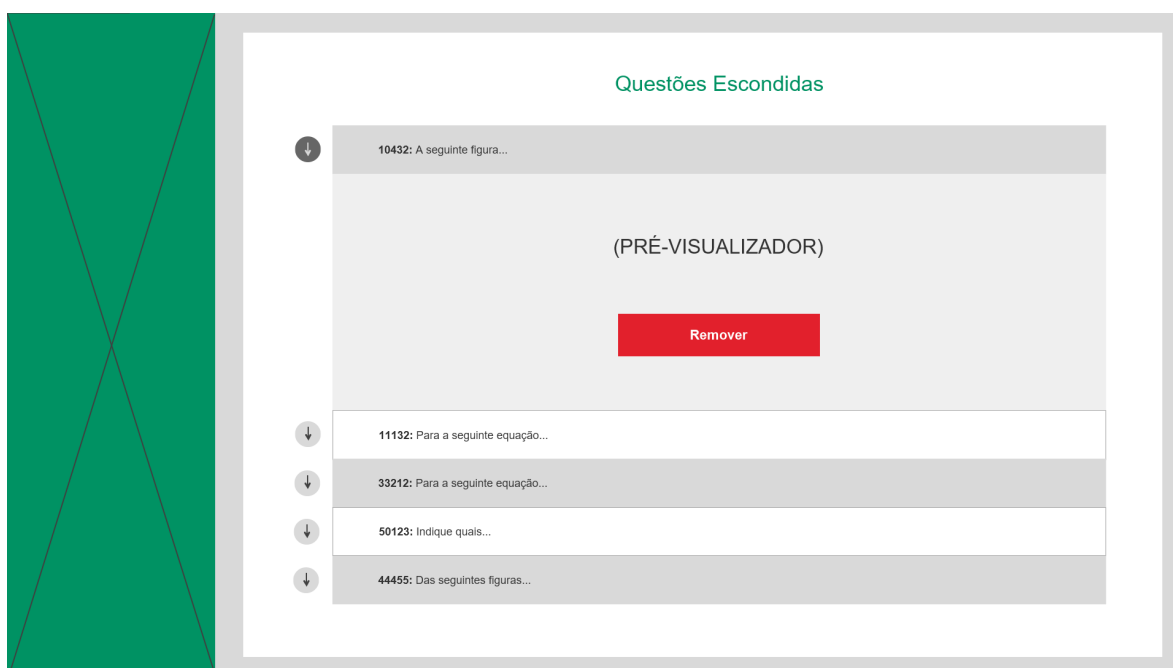


Figure 122: Question Submission Back-Office - Hidden Questions

- Submission History

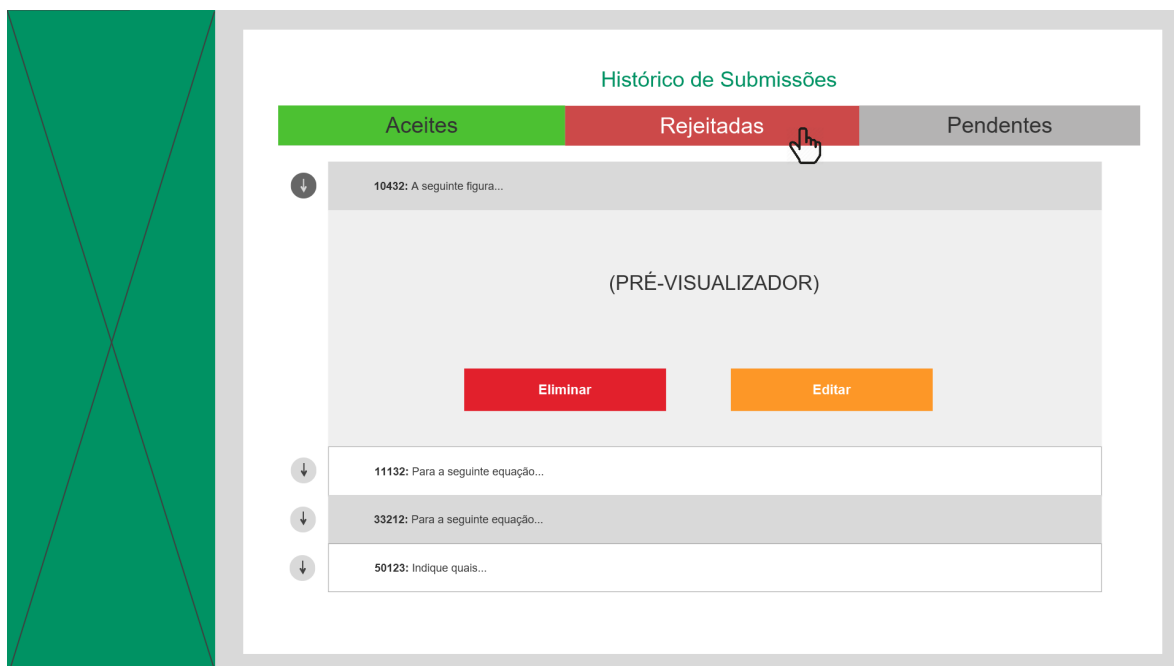


Figure 123: Question Submission Back-Office - Submission History

- Order Themes

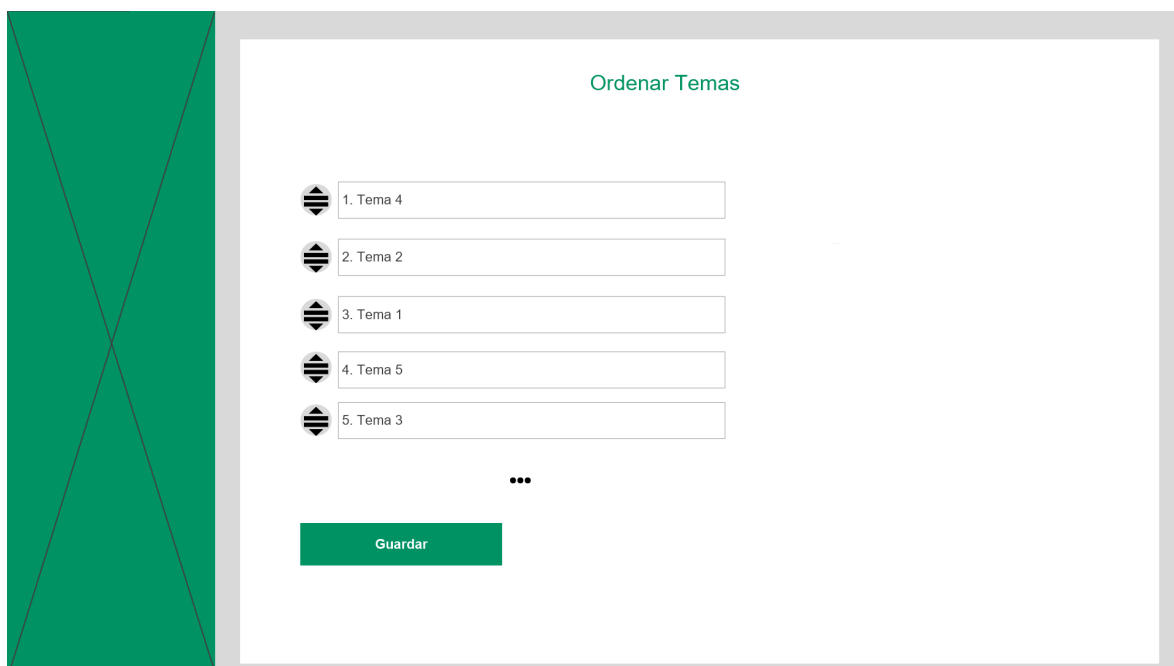


Figure 124: Question Submission Back-Office - Order Themes

- Change Theme Images

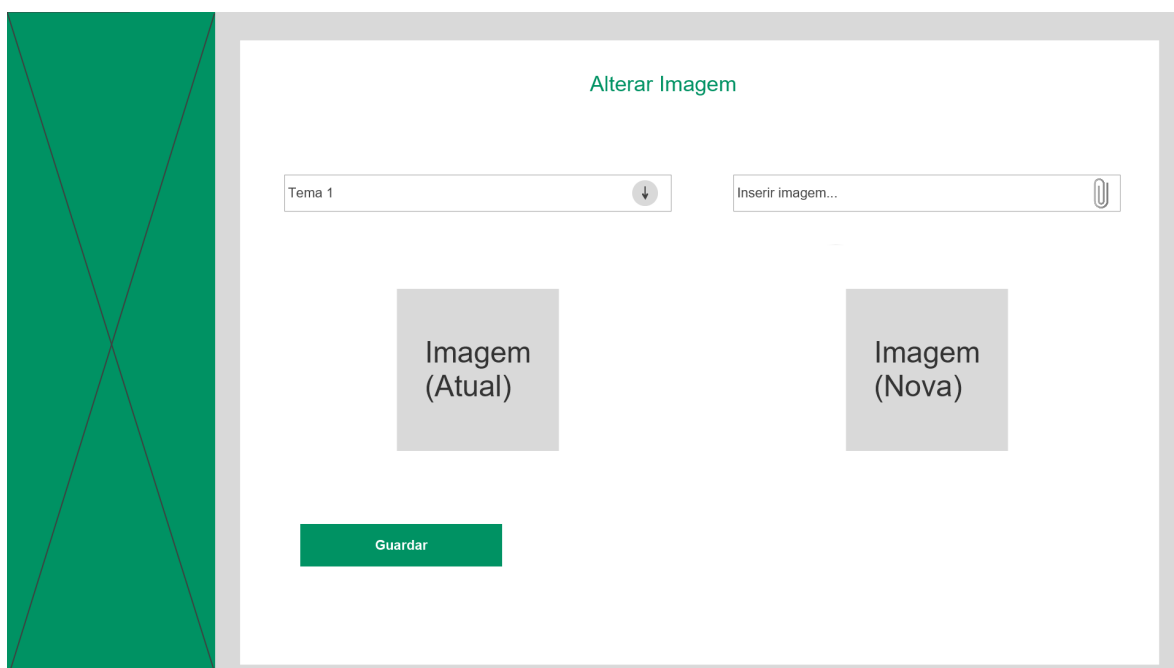


Figure 125: Question Submission Back-Office - Change Theme Images

- Current Monthly Questions

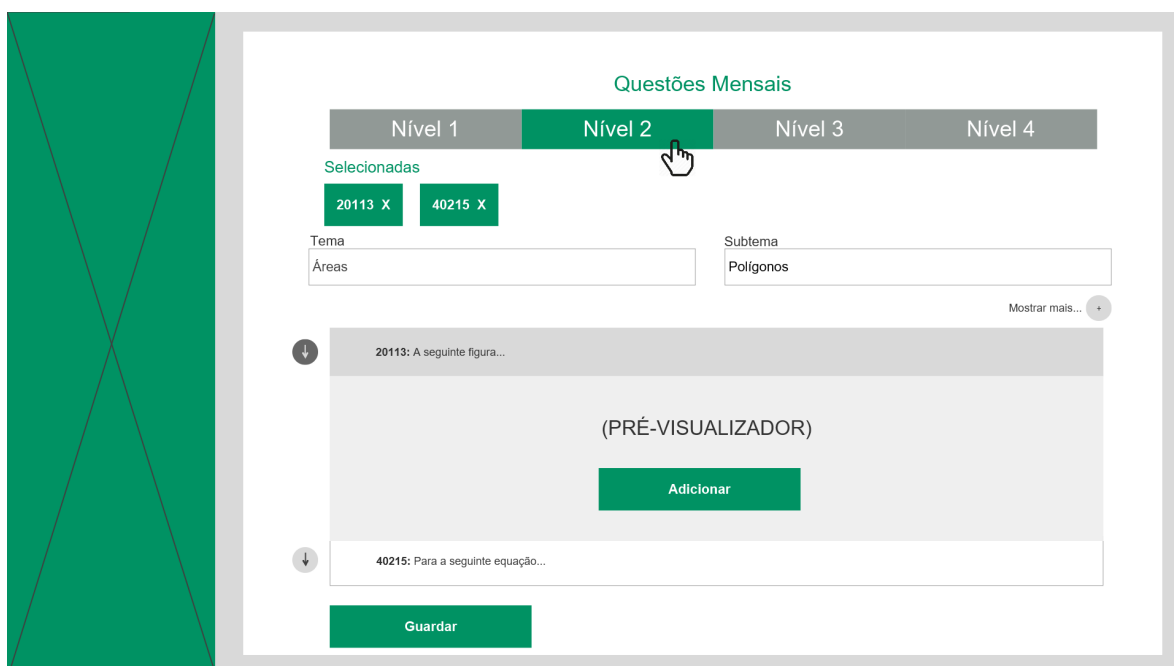


Figure 126: Question Submission Back-Office - Current Monthly Questions

- Monthly Questions History

The image shows a web interface titled "Questões Mensais". At the top, there are two dropdown menus for "Ano" and "Mês". Below these are four tabs for difficulty levels: "Nível 1", "Nível 2" (which is highlighted in green and has a mouse cursor over it), "Nível 3", and "Nível 4". The main content area displays a list of questions. The first question is "20113: A seguinte figura..." and the second is "40215: Para a seguinte equação...". Below the list is a green button labeled "Guardar".

Figure 127: Question Submission Back-Office - Monthly Questions History

MOCK-UPS - "I WANT TO SOLVE QUESTIONS ABOUT..."

- Theme Selector

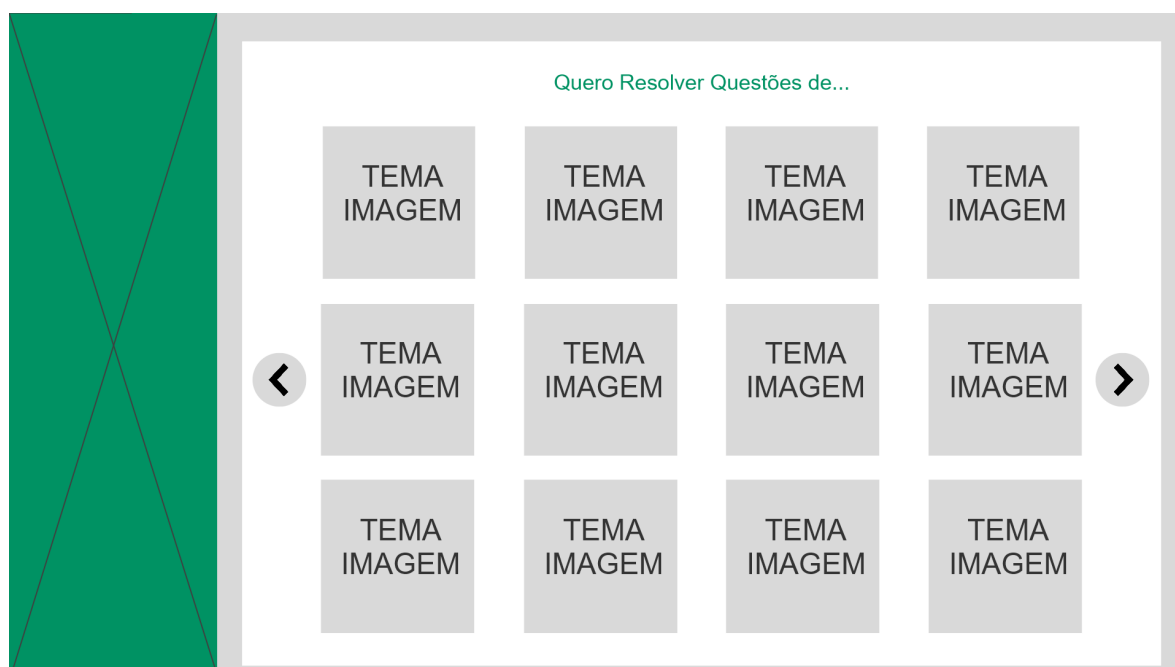


Figure 128: "I Want To Solve Questions About..." - Theme Selector

- Question Solving Screen

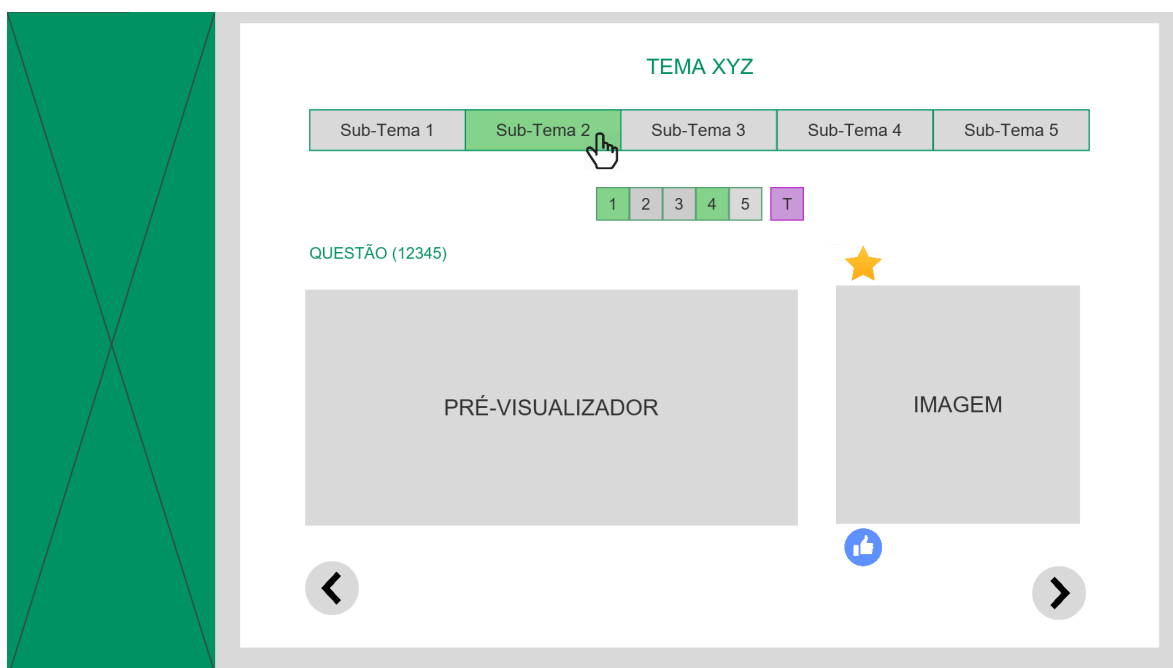


Figure 129: "I Want To Solve Questions About..." - Question Solving Screen

- Favourites

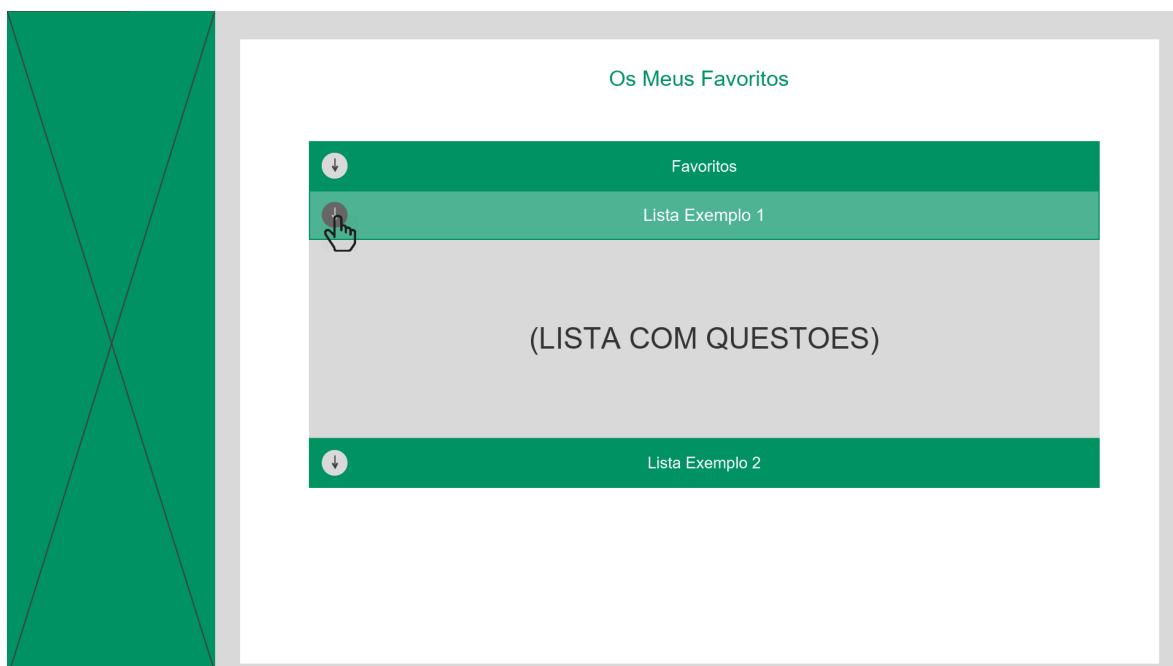


Figure 130: "I Want To Solve Questions About..." - Favourites

- Monthly Questions



Figure 131: "I Want To Solve Questions About..." - Monthly Questions

- Monthly Questions History



Figure 132: "I Want To Solve Questions About..." - Monthly Questions History

- Leaderboards

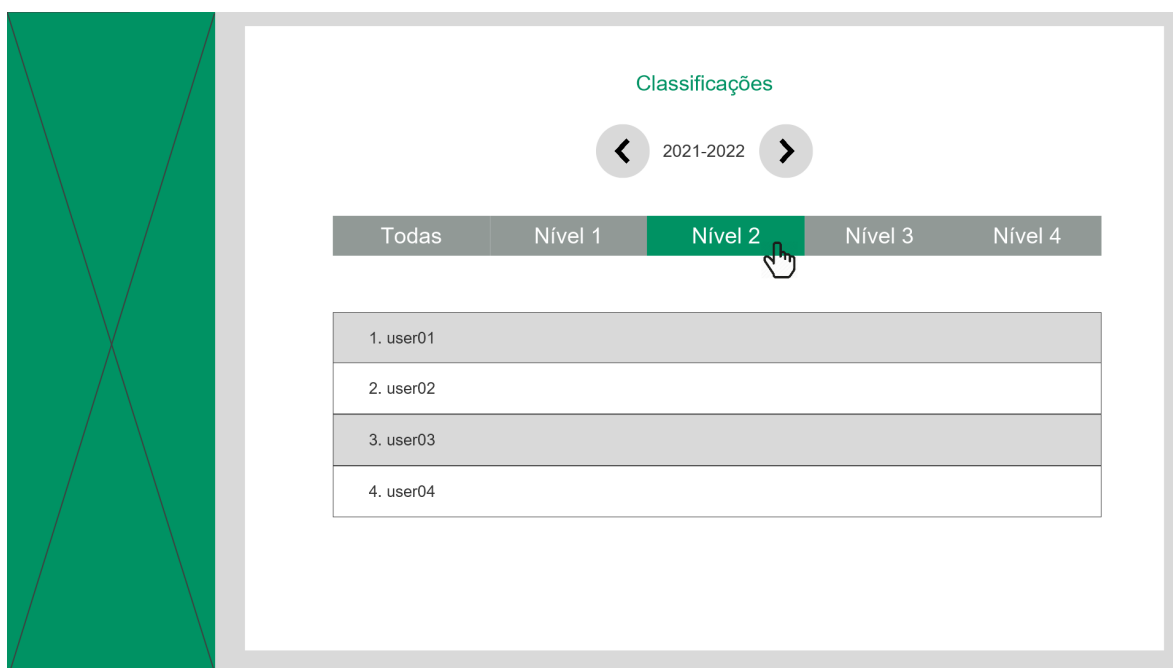


Figure 133: "I Want To Solve Questions About..." - Leaderboards