University of Brasília

Exact Sciences Institute
Computer Science Department

SkillSpace: Mixing social features with a course platform

Felipe Lima Vaz

Monograph submitted in partial fullfilment of
the requirements to the Computer Engineering Program

Advisor
Prof. Dr. José Edil Guimarães de Medeiros

Brasília
2022

University of Brasília

Exact Sciences Institute
Computer Science Department

SkillSpace: Mixing social features with a course platform

Felipe Lima Vaz

Monograph submitted in partial fullfilment of
the requirements to the Computer Engineering Program

Prof. Dr. José Edil Guimarães de Medeiros (Advisor)
ENE/UnB

Prof. Dr. Georges Daniel Amvame Nze
ENE/UnB

Prof. Dr. Daniel Guerreiro e Silva
ENE/UnB

Prof. Dr. João Luiz Azevedo de Carvalho
Coordinator of Computer Engineering Program

Brasília, Outubro 5, 2022

Agradecimentos

Agradeço primeiramente aos meus pais, Gleive e Roberto, pelo apoio e pelo carinho que sempre me deram ao longo de toda a minha vida.

Agradeço também ao meu irmão e à todos os meus primos que constituem a Plearn. Apesar de ligados por ações de rendimentos negativos, cada momento que passamos juntos me fez valorizar ainda mais a família que tenho.

Agradeço à minha namorada Steph por ter me encorajado em cada momento de insegurança e tristeza que tive nos últimos anos. Sou grato por sempre ter acreditado em mim, mesmo nos momentos que nem eu conseguia acreditar.

Agradeço aos amigos que conheci durante o curso, André, Danilo, Kalei, João, Eduardo, Otho e mais muitos outros por terem transformado dias desgastantes em dias mais leves nessa faculdade. Agradeço em especial ao André por ter topado esse projeto de graduação comigo.

Agradeço à Struct e à toda galera que esteve lá comigo. Aprendi mais lá do que no resto do curso todo e sou muito grato por isso.

Agradeço demais a todos os colegas de vôlei que conheci nesses últimos anos, especialmente ao grupo Vôlei Sem Panelinha e à atlética Binária. Sou muito grato por todos os campeonatos e jogos que pude jogar e sei que esse esporte me ajudou demais a não explodir durante os períodos mais intensos dos semestres.

Agradeço também ao Edil por ter me orientado durante todo esse projeto. Nesse pouco tempo que passamos juntos aprendi lições de organização com ele que pretendo levar para o resto da vida.

Agradeço finalmente à UnB que, apesar de ter me feito enlouquecer algumas várias vezes, me proporcionou alguns dos anos mais especiais da minha vida e que faço questão de nunca esquecer.

Abstract

This work seeks to propose a flexible and manageable platform that lets the user create a customizable course and also manage a community around it. In this platform, the administrators have total freedom to develop a course in the way they want by being able to create activities with different types of submissions, group activities in sections, classify the activities, and also link them with social aspects of the platform. In this platform, it is also possible to create online events, link them with activities, create posts for discussions, and suggest new content for the course. This way, we hope that this application can help build communities around the course theme where the users can share their knowledge with others and create discussions around it thus improving the overall experience on the platform.

Keywords: education, online platform, social network, web development

Resumo

Este trabalho busca propor uma plataforma flexível e gerenciável que permita o usuário criar um curso customizável e também gerenciar uma comunidade ao seu redor. Nesta plataforma, os administradores tem total liberdade para desenvolver um curso da maneira que quiserem seja criando atividades com diferentes tipos de submissões, agrupando atividades em categorias e estágios e até ligando as mesmas atividades a aspectos sociais da plataforma. Aqui também é possível criar eventos presenciais ou online, ligá-los às atividades, criar postagens para discussões e sugerir novos conteúdos para o curso. Dessa maneira, acredita-se que essa aplicação possa ajudar na construção de comunidades ao redor do tema de um curso, onde os usuários possam compartilhar conhecimento com os outros e assim melhorar a experiência geral na plataforma.

Palavras-chave: educação, plataforma online, rede social, desenvolvimento web

Contents

1 Introduction **1**

2 The Database Model **5**

2.1 Users and Permissions Models . . . 6

2.2 Activities Models . . . 8

2.3 Events Models . . . 11

2.4 Social Models . . . 13

3 The Backend Logic **16**

3.1 Routing . . . 16

3.2 Authentication . . . 18

4 Visual Prototype **20**

4.1 User Pages . . . 20

4.2 Activity Pages . . . 21

4.3 Event Pages . . . 23

4.4 Social Pages . . . 25

5 The Frontend Logic **30**

5.1 Technologies . . . 30

5.2 Initial Setup . . . 31

5.3 Routing . . . 31

5.4 API communication . . . 32

5.5 Authentication and sessions . . . 32

5.6 Implemented Application . . . 33

6 Conclusion **36**

6.1 Future Work . . . 37

References **38**

Appendix **39**

A API Routes **40**

List of Figures

2.1 Users and permissions related models description. 7
2.2 Activity related models description. 9
2.3 Non-linear learning branches built with stages 10
2.4 Event related models description. 12
2.5 Social related models description. 14

4.1 Visual prototype of SkillSpace's login page made on Figma. 21
4.2 Visual prototype of SkillSpace's two step signup page made on Figma. . . . 22
4.3 Visual prototype of the stages page made on Figma. 23
4.4 Visual prototype of the activities page made on Figma. 24
4.5 Visual prototype of the activities page made on Figma. 25
4.6 Visual prototype of the events page made on Figma. 26
4.7 Visual prototype of the event modal displayed on top of the events papge. . 27
4.8 Visual prototype of the social feed page made on Figma. 28
4.9 Visual prototype of the user's personal feed when accessed by himself on the left and when accessed by someone else on right. 28
4.10 Visual prototype of the chat page made on Figma. 29

5.1 Example code of route management using `react-router-dom`. 31

List of Tables

3.1 Table with the main packages installed in the Node.Js application. 17

A.1 Table with API routes with a brief description of each one of them. 43

Acronyms

API Application Programming Interface.

CRUD Create, Read, Update, Delete.

CSS Cascading Style Sheets.

DBMS Database Management System.

DOM Document Object Model.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

IFL Brasil Instituto de Formação de Líderes Brasil.

Js Javascript.

MVC Model-View-Controller.

MVP Minimum viable product.

PWA Progressive Web Apps.

URL Uniform Resource Locators.

Chapter 1

Introduction

When creating an online course, it is common to focus on the creation of the study content like texts, videos, activities, and other materials since this is what drives most people to the course in the first place. The main issue here is that a course that only provides activities and study content can only keep the user engaged while there is content to interact with. Thus, the user may stop accessing the course after the conclusion of all activities or even before due to the lack of variety in the course's content.

This problem was first presented to us when Professor Edil wanted to improve the platform that Instituto de Formação de Líderes Brasil (IFL Brasil) [1] uses to manage the study progress of its members. IFL is an organization that seeks to form independent leaders by guiding their associates through topics like liberalism, leadership, and management. To reach this objective, they have a platform [2] where the study content is divided into three trails, each focusing on one of the themes previously cited. This platform tracks the progress of the user in each trail, and every time a set of trails is finished, new trails are then unlocked so users can continue their progress on the course. In IFL there is also an event system where the users are encouraged to participate, discuss the subjects and deepen their knowledge of them. The main issue on the platform is that the platform lacked some type of interaction between the users since every interaction happened on a Whatsapp group created for discussions or during the aforementioned events, everything happening outside the platform itself. Another prominent issue was that the activities and trails were hardcoded into the site and required a programmer to code and change them, making it hard for regular administrators to add or remove new tasks.

To solve IFL's problems, we investigated popular course platforms on the internet like Udemy, Coursera, and Skillshare and found out that most of them focus only on the course creation aspect. Inside these platforms, social interactions occur mostly on simple forums or in areas dedicated to asking questions to the course creators. In most courses, these simple social sections have low user engagement due to the lack of incentive to use them. Thus, when the course creators want to build a community around its course, they usually rely on third-party systems with more robust social features like Discord, Whatsapp, or Telegram in the same way IFL did. This is done to make the users interact and discuss the content between themselves, which improves the learning experience and extends the time spent on the course. This social aspect also leads to the creation of an alive community that expands the course, making users dedicate more time to it and sometimes even spend time after finishing it. The issue that still lasts is that, by using third-party systems to manage the social aspect, the community and course platforms become distinct systems with limited integration between them. This may lead some users to be discouraged to engage with the rest of the community due to the need of using two platforms simultaneously. For example, after doing an activity and having an idea for a discussion, the user needs to first switch to the application where the community interacts to only then discuss what he had in mind.

In this work, we propose SkillSpace, a system inspired by IFL's platform that has flexible content creation systems and also has the features necessary to build and manage a community around the course created. By implementing both systems in the same application, we seek to create a very dynamic and interactive platform to improve the user's study experience and gather more people to engage in the course.

To create an engaged and integrated platform, SkillSpace is divided into three interconnected areas. The first one is the social area whose focus is to let users interact, discuss and create content linked to the course in the form of posts and comments. This area consists of a feed containing users' posts similar to the ones found on social networks like Instagram and Facebook and a chat area for users to interact directly. By adding these social features, we hope that the users can help each other progress in the course by creating helpful discussions around the themes, activities, and events. We also hope that they are encouraged to create posts with content that can further improve new users' learning experience like tutorials, related works, new materials, etc. At last, we expect that, by sharing their progress on the course and commenting on it, other users can also be inspired to advance the course.

The second area of the platform is the event section, created to engage the user in presential or online synchronous activities known as events. Here, the user can search for upcoming events on a calendar, confirm or deny their presence on it and send or receive feedback after its conclusion. This area serves as an intermediate between an activity and a social feature that can help users both learn by hearing others' perspectives and also bond with other people, thus creating a stronger community.

At last, the activities area is where the user spends their time doing the proposed tasks, studying the provided material, and thus advancing the course. This area consists of activities that are grouped in stages that need to be completed to advance in the course. These activities have different forms of submissions which may be linked with some social aspects of the platform like making a post or participating in an event. As can be seen, we sought to increase the variety of activity types to diversify the users' studies by creating these social-focused tasks. This way, users can experience a more dynamic course and be encouraged by others to push through the rest of the activities and stages.

With these three connected areas, it is possible to build a flexible and customizable course as well as an alive community without the need to rely on other platforms. Since SkillSpace focuses on the flexibility to create a study course, the whole event and social aspects can be disabled or even made completely optional to the users. This way, by using SkillSpace it is possible to create anything from traditional courses with only texts, videos, and written activities to a more modern approach with a community-driven focus.

To build SkillSpace, we opted for separating the backend and the frontend into two applications that communicate between themselves following an Model-View-Controller (MVC) architecture pattern. The service logic and the database interaction are encapsulated in a RESTful API [3] and all the visual elements of the frontend in a website that consumes the data from the API via requests. This approach permits the reuse of the backend logic in case we ever want to develop new applications like mobile apps, PWAs, or even unique websites for each course. Another positive aspect of this approach is that it helps isolate the frontend from accessing directly the database thus improving the isolation of the MVC layers, and easing future maintenance in the project. The backend was built in Node.Js [4] with the Express.Js [5] framework on top of it while the frontend was built using ReactJs [6]. We chose to use ReactJs as our frontend library due to its ease to create visual components and reuse them on other pages, reducing the development time considerably, and also because we already had previous experience using it. About Node.Js, we selected it because it is easy to learn and simple to adapt. Since we were used to using Ruby on Rails [7], a more robust and complete framework, for building RESTfull APIs we opted to switch to a more flexible and simple approach using Node.Js and Express.Js. However, considering the scalability of the platform, it would be interesting to further port the API to Ruby on Rails if the platform starts dealing with a high number of concurrent users. At last, this combination of frameworks was chosen to enable the code of both the API and the website to be written in javascript, facilitating its development and management.

We start the next chapter by introducing the modeling of SkillSpace's database and why some decisions were taken. In chapter 3, we dive a bit into the main logic that is behind the development of an API using Node.Js and Express.Js. In chapter 4, we present the visual prototype created to serve as a visual guide for the frontend development, as well as explain some decisions made to improve the user experience. Chapter 5 serves to describe the main logic of the React.js application. In chapter 6, the reader is presented with the developed platform, as well as comparing it to the initial idea presented throughout chapters 2 to 5. At last, we conclude with chapter 7 and provide some ideas for future improvements on the platform.

Chapter 2

The Database Model

When projecting SkillSpace we had the option to work with a full-stack application where all the models, controllers, and views are inside the same project or separate the models and controllers into a RESTfull API and the view layer into another application. We chose the latter due to the various advantages that come with the use of APIs. APIs are usually useful to abstract how the logic of an application is implemented and focus only on its inputs and outputs. Since APIs are isolated and independent of the applications using them, it is possible to have multiple applications sharing the same API. For example, you can simultaneously use the same API for a web page and a cellphone app without the need to build another one with different logic.

To build our API, we used Node.Js as our Javascript runtime environment which is used to execute Javascript applications. One of the main positive aspects of Node.Js in this application is to unify the language used on the client-side and server-side, since Javascript is also the language we use to write ReactJs code. To abstract part of the server-side logic and focus on the service we are trying to build, it is usually recommended to use Node.Js with a framework. In this project, the selected framework is Express.Js which provides some tools to help build web and mobile applications like routing, sessions, error handling, and middlewares. Apart from the cited features, Express.Js is relatively minimal with many features that are commonly present in other frameworks available as third-party packages. This means that Express.Js is light, simple to learn, and very customizable.

The API's main purpose is to query the database and process the received data to be used by the frontend application. Thus, the database is not part of the API itself, but it communicates with it through the queries. The selected Database Management System (DBMS) in this project was MySQL [8], due to our previous experience working with it. Any other relational DBMS could have worked in this project with the correct adjustments to the API code.

All the modeling was made using the database diagram design tool dbdiagram.io [9]. The complete modeling is mainly divided into four main sections: User-related models, Activities-related models, Event-related models, and Social-related models. Although the next sections of the text are described in an isolated form, every section is very connected to the others to create a dynamic and interconnected platform.

2.1 Users and Permissions Models

Before describing any of the main features of the application, it is fundamental to describe first what is a user and what it will be capable to do on the platform. These models are represented in Fig. 2.1. Starting from the `User` model, it is defined by a name, an email, an encrypted password, a CPF, a phone number, a DDD, a birthdate, and an `is_active` boolean to define if the user is active on the platform. This boolean was created to avoid the need to delete a user and deal with the consequences of deleting or nullifying all models that he was related to. By disabling a user, all of its information remains on the database and no other model needs to be deleted. The `User` model also has two booleans to check if he is the owner of the site, an administrator, or a regular user. These booleans were created to better control the permissions that a user has on the platform. At last, the user model has a reference to an `Address` model that stores all the necessary information of its address like country, state, city, number, neighborhood, and street. This model has a relation of one-to-one with the `User` and was created mainly to not overcrowd the `User` model with too many fields.

Most features on SkillSpace require the user to be logged in. Non-logged users only have access to the login and signup features. To better interact with the platform, first, it is necessary to sign up providing all of the basic user information and some of the users' interests on the platform to create a more personalized experience. After the account creation, the user only needs to log in to the platform using the created e-mail and password.

With a user created and logged in, the person now has access to the core features of the platform which are the main course area, the social area, and the events area. Although having permission to see and access a lot of features, some of them are restricted only to administrators and to the owner, most of them being related to creating and editing the course's content like activities, stages, events, etc. To improve the permissions flexibilities and create multiple levels of administration, we created the `Permissions` model that is related to the user via the `AdminPermissions` associative table. This model lets the owner give one or more specific permissions to a user based on its needs at the moment which can help create low-level administrators like curators, monitors, and teachers.

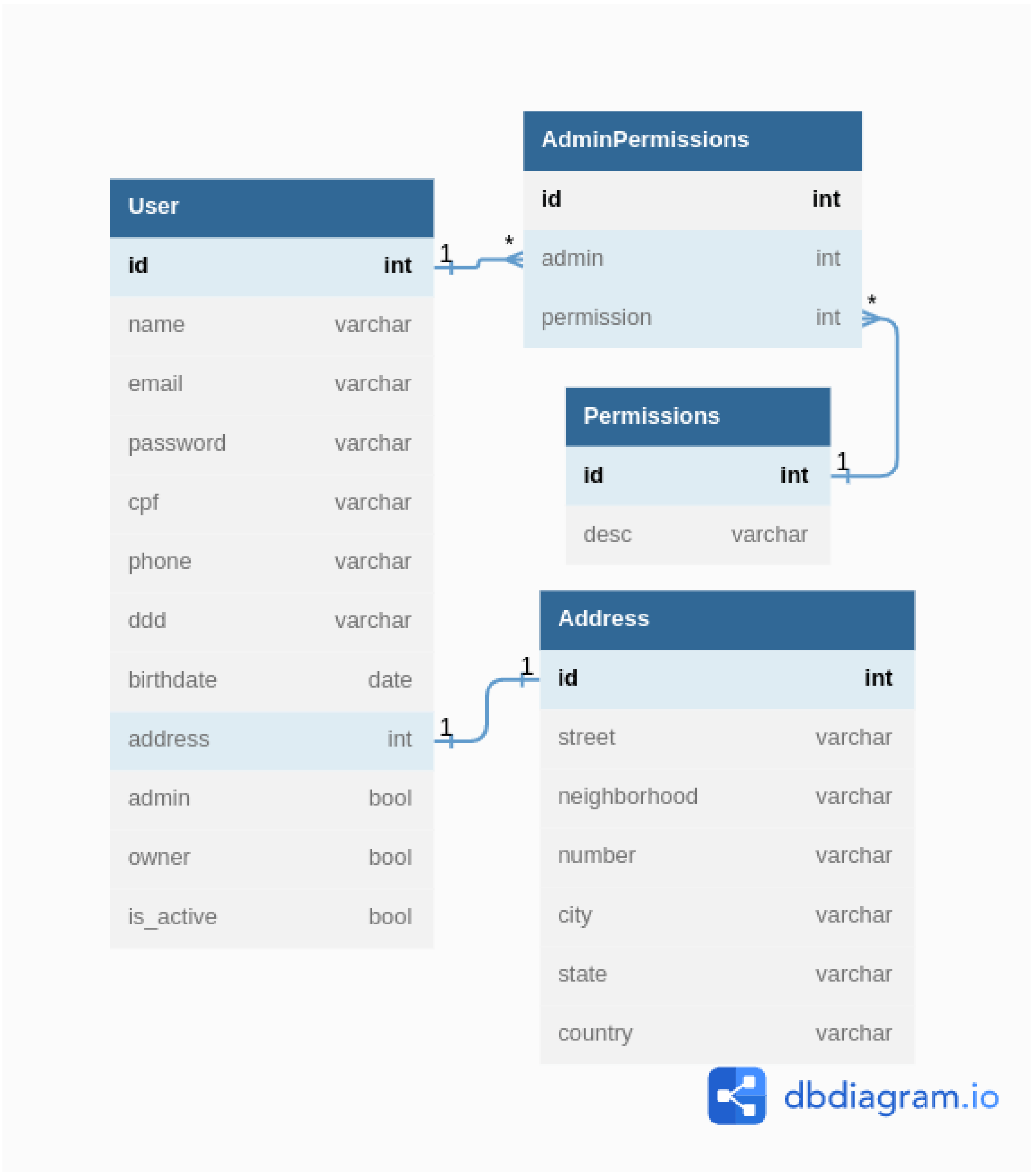


Figure 2.1: Users and permissions related models description.

To summarize the permissions system, at first, we have the user permission which limits what can be accessed as a logged regular user on the platform. These users can access the activities, posts, events, and profiles page and are limited to only creating new posts and editing the posts they created. The next level is the customized permissions that use the `AdminPermissions` model, which can be different for every user depending on the owner's needs. For example, the owner can grant user-specific permission to delete any post to create a kind of post moderator. These customizable permissions are divided into three categories: "modify", "show" and "delete". Each of these categories can be combined with a model to create a specific permission. The permission for the previous example, in this case, would be `delete_posts`. Next, we have the admin level permissions which lets the user create, edit, see and delete almost all elements of the platform. At last, we have the owner level which is the highest permission and is only granted to one user. The only difference between the admin and the owner is that the owner can change users' permissions as well as promote and demote them to regular administrators.

2.2 Activities Models

One of the differentials of SkillSpace is its flexible course creation system. To achieve this objective, the user can track their progress on the platform via the conclusion of activities and stages. To reach this goal, we proposed the database diagram model presented on Fig. 2.2 and that will be described in this section.

Before defining the `Activity` model, it is first interesting to describe what kinds of activities there are in SkillSpace. Here, an activity can be classified in two ways, type, and category. Activity types are pre-defined sets of activities that characterize how the activity must be submitted. There are currently four types of activities, each unique in the way they are evaluated and submitted. The first type is the theoretical one where the student need only to watch or read the associated material to check it as completed, therefore these activities are not scored. Contrary to this type, practical activities necessitate the user to submit some kind of material to be evaluated. This submission can be either text or files. Just like the theoretical type, practical activities can also have texts, videos, or other materials associated with them. The next set of activities is the event participation one. These are connected to an existing event already created on the platform. This way, the score of a student can be attached to its presence or absence in the event or a normal scoring system like the practical type. Event activities only exist during the event period. After the event, the activity will not appear anymore to students that didn't participate although they will still show to users that were present. At last, we have the social participation activities, which are linked to the social interactions of the user on the platform. This interaction can be creating a post or commenting on a post with a determined tag. In this activity, the post must be created via the activity submission page to connect it with the activity. This post will appear in the feed as a regular post to other users. Therefore, the `ActivityType` model is defined only by a name and a description.

In SkillSpace, an activity can also be categorized into categories. Different from activity types, a category is a classification that the admin creates for the activities to better organize the course. For example, in a gastronomy course, you may have one category for desserts and another for main dishes to help the user better find the knowledge he

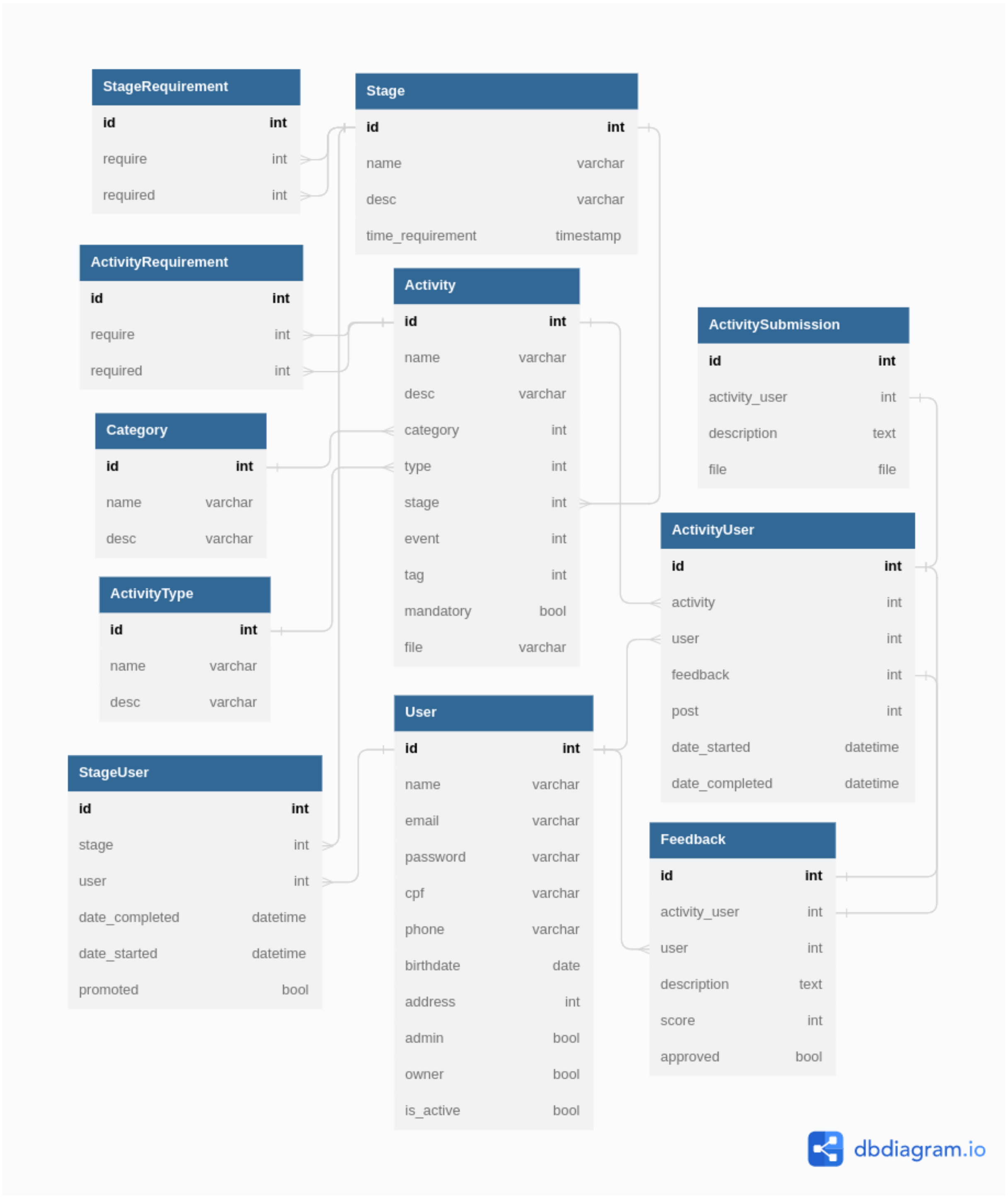


Figure 2.2: Activity related models description.

is looking for. Although different in purpose, these two classifications are defined in the same way on the database, with only a name, and a description.

After defining the different activities grouping logics, we can start describing the `Activity` model itself. It is defined first by a name, a description, and a mandatory

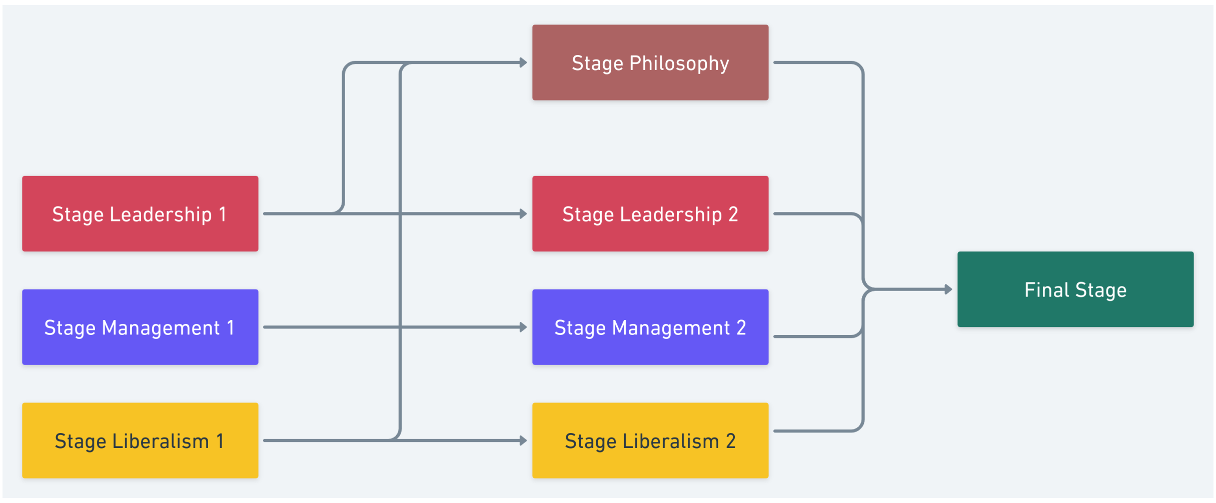


Figure 2.3: Non-linear learning branches built with stages .

boolean marking its conclusion as required or not to progress the course. Since activities in SkillSpace were built to integrate into all other sections of the site, the `Activity` model has reference to the `Category`, `ActivityType`, `Stage`, `Event`, and `Tag` models. The latter two are necessary only if the activity is classified as an event type or social type respectively. In the case of other types, these two fields remain empty.

When a user starts an activity, an instance of the `ActivityUser` model is created to track the progress of the user on that activity. This model contains the dates that the user started and finished the task, references to the `User`, `Activity`, `Post`, and `Feedback` models. The reference to a post is only necessary in case the activity is a social one and is used to record the post that the user created to complete it. Very connected to this model, we have the `Feedback` model responsible to record the teacher or administrator's score and review of the user's work. This model has a reference for the `ActivityUser` in question, to the user that reviewed the activity, a description, a score, and an approved boolean to mark if the student completed the task correctly.

In case the course manager wants to create some sort of order for the activities to be concluded, it is possible to set some requirements for an activity. This is represented by the `AcivityRequirement` model, which only contains the reference to the required activity and to the activity that requires it. This way, it is possible to create a linear path for the activities to be executed, thus improving the organization of the course.

Another feature of SkillSpace that helps to organize the student's progress on the course is the existence of stages. Stages are a way of grouping activities of similar difficulty and theme into sections that need to be concluded to unlock more tasks. Differently from categories, which have only visualization and organizing purposes, stages exist to pace the student's progress blocking them from accessing complex and advanced activities before completing the basic ones. Each stage can have other stages as requirements. Thus, it is possible to create linear or non-linear learning branches each with its stages and activities,s as can be seen in Fig. 2.3. This way, the admin can customize its course to even create sub-courses inside of it just by creating new and isolated learning branches. This feature comes to complement the activities requirement, and with both combined, it is possible to craft any type of course being it completely linear or with multiple branching paths.

A user can be in more than one stage simultaneously if he has the previous stage concluded. To conclude a stage, it is necessary to complete and get the minimum score on all the mandatory activities inside it. Some stages also have a time requirement to be concluded, which is linked to the time that has passed since the user created the account on the platform. Another way to finish a stage is to be promoted by an administrator. This resource can be used if a more advanced user joins the platform and want to skip some stages. In this case, it is necessary to contact an administrator in advance and the stage will be tagged as “promoted” on the user stages page. It is also important to notice that when the admin is creating the course, there needs to be a final stage with no activities to indicate that the user has finished the course. All the stages are required to unlock this one. At last, we can finally define the `Stage` model as having a name, a description, and an optional time requirement.

2.3 Events Models

To further improve the events system used by IFL Brasil’s platform, we propose the models presented in Fig. 2.4. This modeling was chosen to enable the creation of online or presential events and link them with an activity, for example. This way, it is possible to create more dynamic and interactive activities for the students such as group discussions, debates, or presentations.

To achieve this objective, we define the `Event` model on the database by a name, a description, a date to mark when it will happen, a remote boolean marking it as remote if true and presential otherwise, and a link containing either an online meeting link, in case of remote presentations or a Google Maps link for the presential location of the event.

To create an event it is necessary to have the necessary permission or be the main administrator of the platform. In other words, regular users do not have access to its creation. On the event creation page, the user has to make some decisions. The first one is to decide whether the event will be attached to an activity or not. If so, a new activity will appear in the students’ learning flow on the selected stage, which may be classified as an obligatory task or not. If the newly created event isn’t attached to an activity,

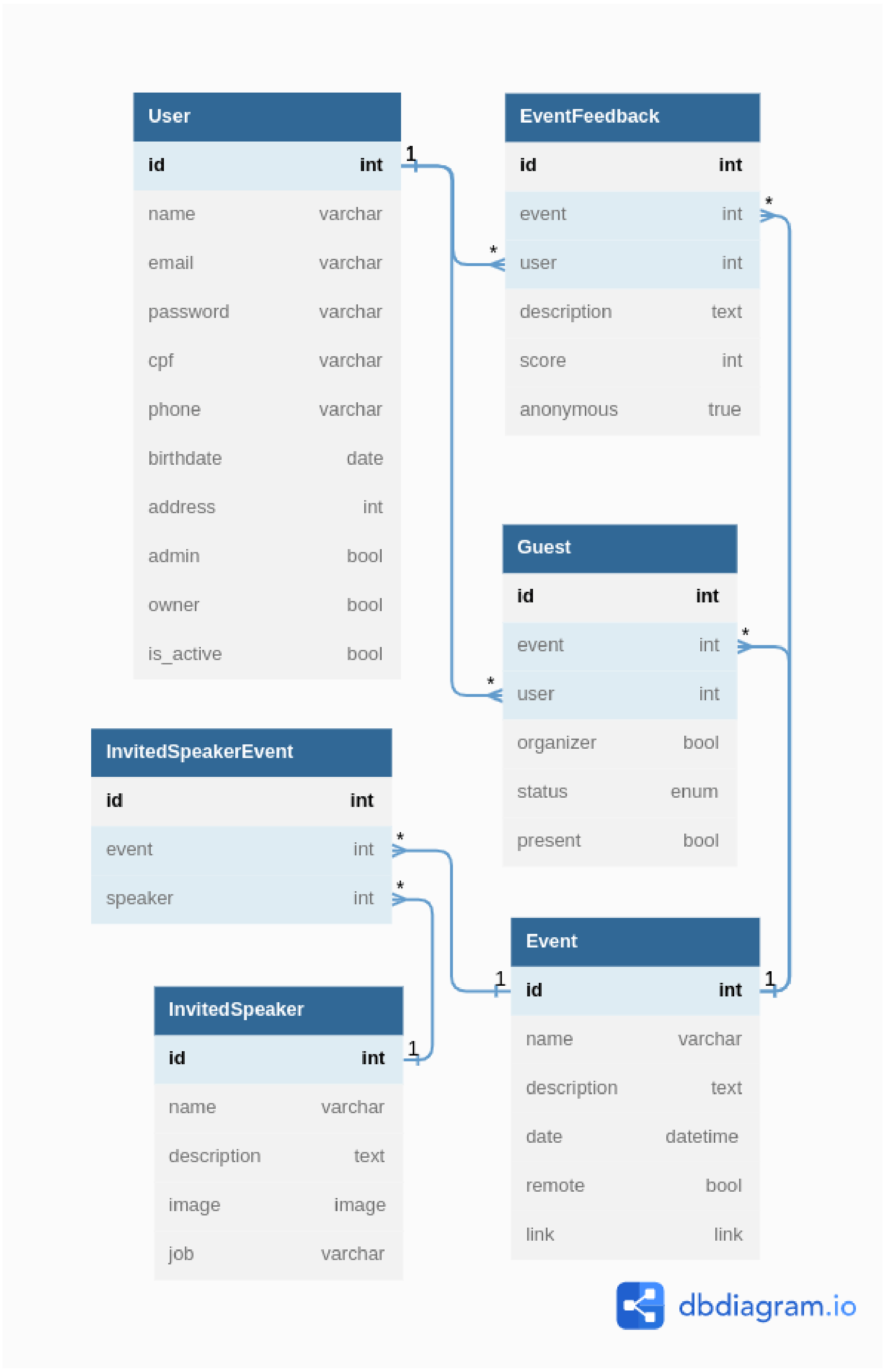


Figure 2.4: Event related models description.

the creator can pass a list of guests to be invited to the event. This is defined in the database as the `Guest` model, which contains references for a user of the platform and the created event, a present boolean to better organize who was indeed present on the day of the event, and an organizer boolean that marks the guest as one of those responsible to manage the event or a regular student. The `Guest` model also has a status where the user can select some options depending on its intention to participate in the event like: “Confirmed”, “Not Confirmed”, "Have interest", and “Not answered”.

An event is not limited to users on the platform, if the admin wants some outside person to also be a guest, it is possible to set it as an invited speaker. To do so, it is necessary to first register it on the administrator dashboard. The `InvitedSpeaker` model consists of a name, a simple description, a photo, and a job. These pieces of information will be displayed on the event’s details page to catch the users’ attention if the speaker is someone notorious or renowned. It is also interesting to comment that, after the invited speaker is registered on SkillSpace, it can be used in other events without the need of registering it again. This is possible by the creation of the associative table `InvitedSpeakerEvent`.

At the end of the event, the admin marks manually the users that indeed were present in the event, which may be considered when grading the student’s activity. At last, an event can be reviewed by the present guests via feedback. The `EventFeedback` has reference to the user who wrote it and the event, a description, a score, and a boolean anonymous field in case the student does not want to be recognizable when sending the feedback. The feedback is not related to any sort of scoring for the students and exists solely for the organizers to improve the next events based on the guest’s opinions.

2.4 Social Models

One of Skillspace’s main objectives is to improve users’ learning by creating an environment of discussions and exchange of ideas. To achieve this goal, the last section of SkillSpace that needs to be described is probably the most important one to keep the course alive in the long term. This section is the social one, whose models are described in Fig. 2.5. It consists of a post and a chat system for the users to connect with others and share the knowledge learned on the platform.

In Skillspace, the core of social interaction is done by discussions via posts and comments on users’ feeds. The `Post` model is defined by a name, a description, an optional attachment file, a reference to the user who created it, and a reference to the parent post. To create discussions, it is necessary to have a way to reply to other users which is possible by creating comments on a post. These comments also use the `Post` model, the only difference being that its parent field references the post in which the comment was made. In case the post isn’t a comment/response to another one, its parent reference remains empty. By reusing the same model for posts and responses, it becomes possible to create multiple post threads, thus better organizing the discussions. There is also one more type

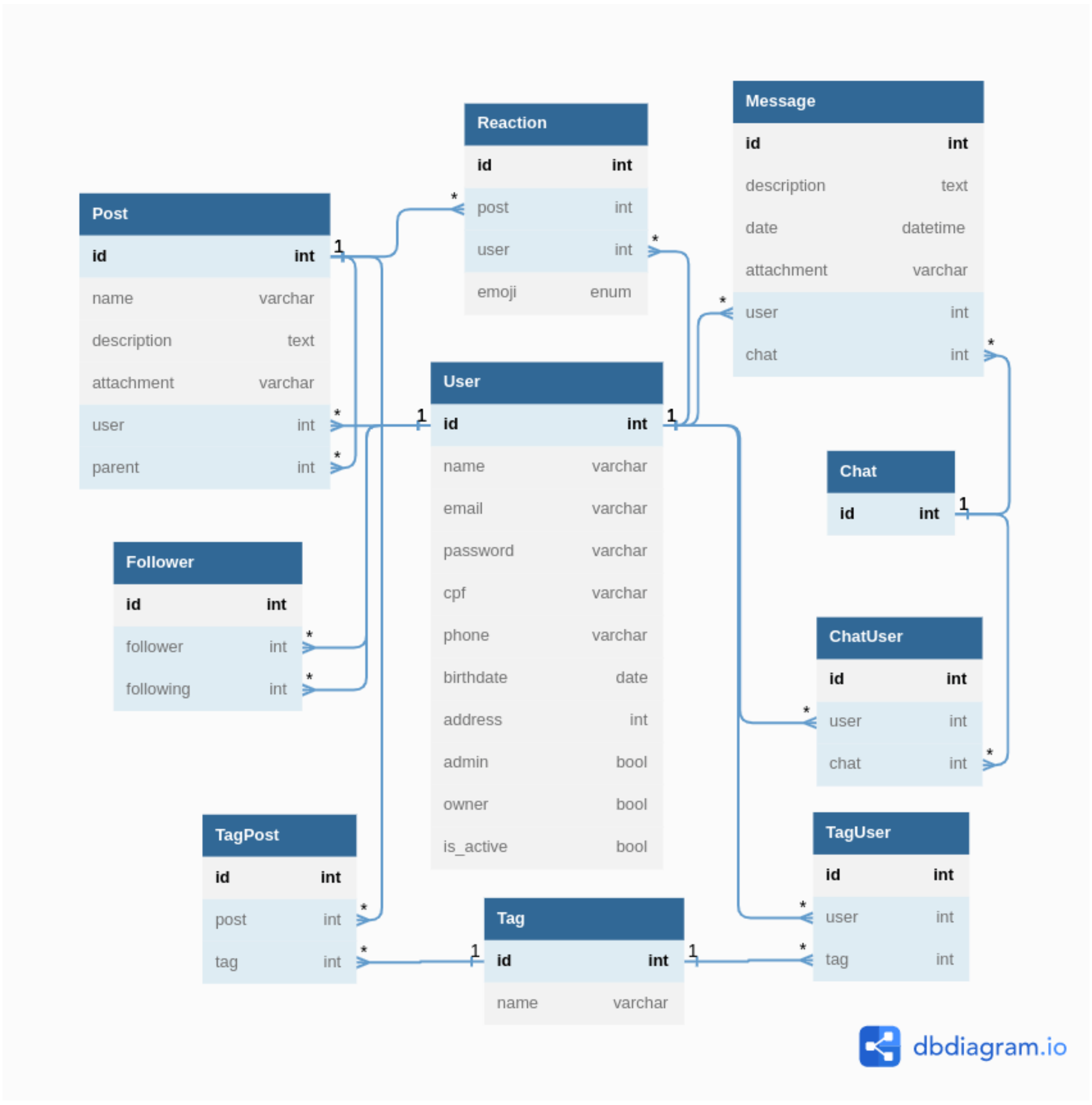


Figure 2.5: Social related models description.

of interaction related to posts which is the reactions. Any user can react to someone's post with an emoji which can help other users to see how is the general reception to its post without the need to read the reply posts first. The `Reaction` model is described as an `enum` with the options being the existing emojis or icons of the platform and references to the user who reacted and to the reacted post.

Similar to the categories present in the activities section, in the social area, there are tags to group posts with similar themes. The `Tag` model is defined only by a name and can be related to multiple posts. It is also important to notice that each post can have one or more tags to define its theme and to further facilitate finding specific posts just like hashtags work on other social networks. The main difference to conventional social platforms is that only administrators can create new tags to make the discussion spaces more controlled.

In SkillSpace, posts are displayed for the user in two ways. The first one is the personal feed, which serves as a storing place for old users' posts, where every time a user creates a new post, it is immediately displayed there. This feed can also be visited by other users. On the other hand, there is also a social feed that shows posts related to the user's interests. This is done via the aforementioned `Tag` model. Users can follow tags related to their interests to receive in their social feed new posts created by other users marked with that tag. This is described in the database via the `UserTag` associative model, meaning that a user can follow multiple tags with no problem. Other types of posts will appear on the user's social feed, which are the posts published by the people that are being followed by the user. Similar to most common social networks, users can follow each other to keep up with someone's content. This is described by the `Follower` model on the database that contains a reference to the user that is following, and to the one that is being followed. This follower feature is very important to create circles inside the platform, where people with the same interests can gather to bond and discuss it. At last, to diversify the posts presented to the user, the social feed will also display some posts indirectly linked to the user, like posts from people that goes to the same events as the user or posts of people that are connected to the users that he is following. This last feature is optional and can be disabled on the settings.

Another option of interaction apart from the post system is to interact directly with other users using the chat feature. In Skillspace it is possible to create chats between one or more users where it is possible to send messages, images, or files. Each chat is a set of messages sent between a group of users. On the other hand, the `Message` model is defined by a description with the message text, a date to mark when it was sent, the sender identification, and an attachment field for images and files. Any user can send private messages to another user via the chats section, since there is no friends system in the platform to restrict the users that are possible to initiate a conversation.

We have now described how the database of the platform is modeled and how each model connects creating an integrated application. It was also possible to describe the reasons behind some of the decisions and explain the overall logic of the platform. Now, the next step in the development of SkillSpace is to create the API. The next chapter details some of the steps necessary to create the backend logic using the selected technologies.

Chapter 3

The Backend Logic

Starting a project in Express.Js is a very simple task, it is only necessary to import the method `express` and make it listen on a port of the computer. After that, every time this port is accessed through the browser or other application, a request is sent to the application which may answer with a response. But with this simple setup, nothing is returned yet. For it to work, it is first necessary to define some middlewares first.

Middlewares are functions that have access to the request, the response, and to the next function in the middleware of the cycle. Express.Js works by calling middleware functions in a defined order until the response is ready to be sent back to the client. It is common practice to define a lot of initial settings as middleware to be executed before the server starts listening to requests. Some of these settings come from packages outside of Express.Js that may be installed via a package manager like yarn or npm. The most important ones are described in Table 3.1

3.1 Routing

When building a web application in Express.Js and many other frameworks, one of the first steps that need to be taken is defining its routes. Routes are mechanisms that map an URL and an HTTP method to a specific code that will handle the request the client just made. Usually, in an API, there are different routes to access or manipulate each resource, where a resource can be an element of the database or a resource from another API for example.

In an Express.Js application, routes are managed as a type of middleware where it is only executed if the URL and HTTP method matches its reference. In this case, it calls the function that was set in this middleware. To better organize the code and enforce an MVC code pattern in SkillSpace, we isolated all methods related to one resource into a controller. Thus, methods related to the basic CRUD operations like creating, editing,

Table 3.1: Table with the main packages installed in the Node.Js application.

Package	**Description**
aws-sdk [10]	Package that simplifies the usage of AWS services on the application. Used here to set the communications with Amazon's cloud storage to store images and files.
bcryptjs [11]	Package used to encrypt user-sensitive data like passwords.
body-parser [12]	Used for parsing incoming request bodies in a middleware before being handled. Used here to parse JSON data.
cors [13]	Package added to permit Cross-Origin Resource Sharing between the API and the frontend since both use the same address.
dotenv [14]	Package useful for defining environment variables
express-validator [15]	Package used to define middlewares that check data sent on the request and validated it before being processed in a controller.
jsonwebtoken [16]	Package used to create JSON Web Tokens (JWTs) to manage user sessions and authentication.
multer [17]	This is a Node.js middleware for handling multipart/form-data, used in image/file uploads.
multer-s3 [18]	Extends multer to work with Amazon's S3 when uploading images and files.
mysql2 [19]	Package used to communicate with a mysql2 managed database.
sequelize [20]	Package that abstracts part of the query logic to the database through a promise-based Object-relational mapping approach.

accessing, and deleting a resource are found in most controllers. These routes follow a pattern by starting with the model name on the URL, followed by the method that is being called, and an id to find the instance of that model. For example, the `Tag` resource has a route `/tag` for the index, `/tag/:id` to show a specific tag, `/tag/create` for creation a tag, `/tag/update/:id` for updating a tag, and `/tag/delete/:id` for deleting a tag. Note that any element that starts with a colon like `:id` describes a dynamic parameter that needs to be passed to access the desired resource. Some resources also have more specific routes and controller methods to deal with particular behaviors, for example, the user controller has login and logout methods. For a more complete description of the API's routes, please refer to Table A.1. Please, notice that standard CRUD routes like the ones described in the previous example had to be simplified on the table to reduce its size.

3.2 Authentication

Since we describe our API as a REST API, one very important constraint that we need to deal with is the stateless one. This constraint defines that the server shouldn't have to remember the previous client requests to process the next, all information necessary to execute a request, should be contained in the request itself. This complicates the concept of user sessions on the backend since it is not possible to track which users are logged at any time. This way, to secure our application as a REST API and still be able to deal with user authentication, we use Javascript Web Tokens [21] (JWTs). These tokens consist of a header containing basic information like the type of token and the signing algorithm being used, the payload that usually contains sensitive user data like the user email, name, permission info, etc. At last, there is a signature to verify if the message wasn't changed along the way. All these three sections are encrypted using the signature info that usually comes with a secret.

The way these tokens are used to authenticate a user in an API is very simple. At first, the user sends a log-in request to the server with a password and an identification like an e-mail or username. The server will check if the password and identification match and, if so, will build the JWT based on the user's information. After that, the API sends back the token to the client to store it somewhere (usually in a cookie). At last, the user needs to send this newly created token in the header of every subsequent request to ensure it has access to some resources.

To facilitate the permission handling in SkillSpace, we created some middleware to be called before accessing the controller methods. There is one middleware to check if the user is authenticated, one to check if it is an administrator, one to check if it has specific permission related to that route, and one to check if it is the owner.

After covering the essential logic behind our API, we've now concluded describing the backend since it is defined only by models and controllers. Starting from the next chapter, we dive into the more visual and user-interactive part of the application, the frontend. There we discuss how the decisions taken in these last two chapters reflect on the visual interface with which the user interacts.

Chapter 4

Visual Prototype

Before developing the frontend of a website, it is usually interesting to develop a non-functional visual prototype first. This prototype works as a first draft of the site's style and serves as a reference to build the real website in the end. By prototyping in advance, it is possible to have a component-based approach to the site's visual elements, for example by creating sections of pages that work very similarly and that may be reused later. Another advantage of early prototyping is to plan the pages' flow. For example, when creating the user's initial flow on a website, first he has to start on the homepage, next he accesses a signup page, and lastly, he enters the login page. Just formalizing which pages belong to each system of the application, makes it a lot easier to ground the planned ideas for the site.

In this step of development, it is common to draw the first draft on a piece of paper, or editing software to create the initial flow of the site's pages. To prototype SkillSpace, we made use of a graphic editor called Figma [22] since it has componentization features that help to reuse parts of the site and align with ReactJs's idea of componentization. It is also important to comment that not all planned pages were prototyped due to time constraints. The main pages that were left out are the majority of the administrator dashboard pages, since we focused on the pages that made part of the regular student experience.

4.1 User Pages

The first pages prototyped for SkillSpace on Figma were the user creation and authentication ones. They are represented in Fig. 4.1 and Fig. 4.2 respectively and consist of a login page and a signup page. The login page is very straightforward, it has an e-mail and a password field. The signup is a little more elaborated and is defined by two steps. The first step is where the main information about the user is provided like name, e-mail, password, cellphone, birthdate, document number, and address information. In the second step, the user is presented with a list of topics that might interest him on the platform. By selecting some tags in advance, we desire to help improve the user's initial experience in the social area by presenting posts on the chosen subjects.

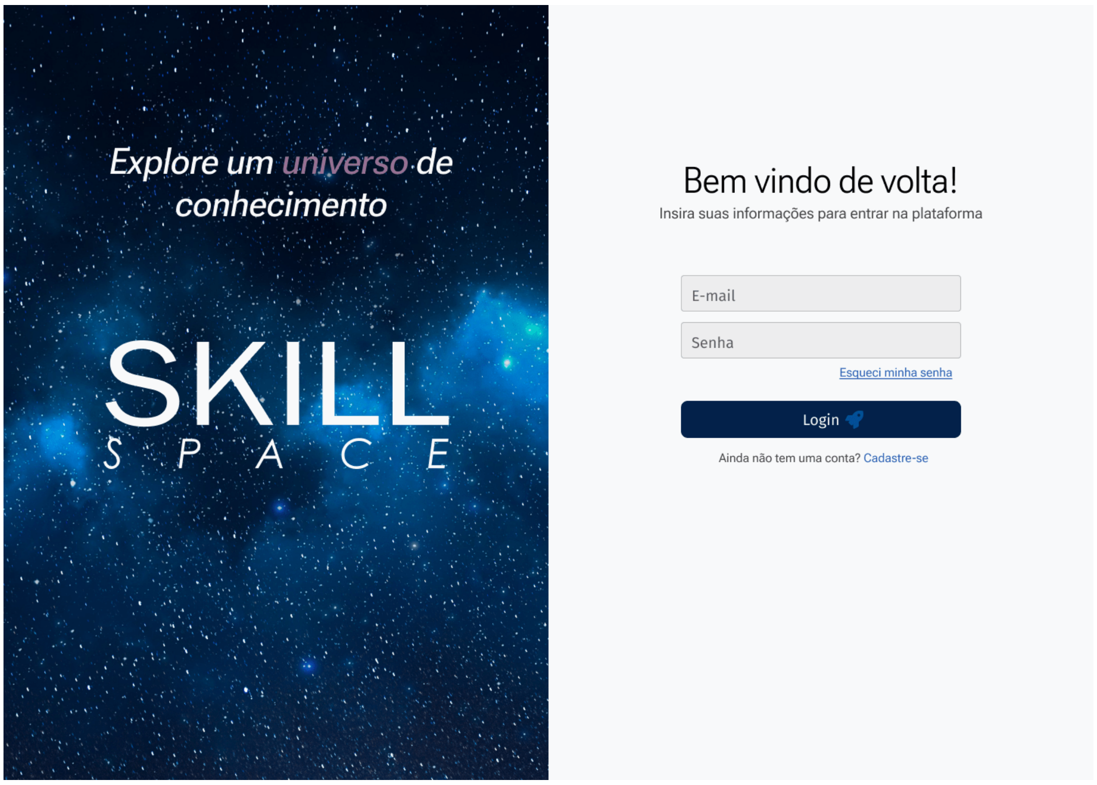

Explore um universo de conhecimento
SKILL
S P A C E
Bem vindo de volta!
Insira suas informações para entrar na plataforma
E-mail
Senha
Esqueci minha senha
Login
Ainda não tem uma conta? Cadastre-se

Figure 4.1: Visual prototype of SkillSpace's login page made on Figma. .

4.2 Activity Pages

On the activities section of the platform, there are pages for stages, activities, and submissions. The first page of this section is displayed when the user accesses the activities button on the sidebar. This page displays all the stages present on the site, as well as the overall progress of the user on the platform and is presented on Fig. 4.3. The stages on this page provide information about how many activities it has, how many have been completed, and what are the requirements to unlock blocked stages. The state of the stage is represented by its color, where gray stages are locked, light blues have been started, and dark blues are completed. At last, it is possible to filter stages by name or by their completion state.

After clicking on a stage, the student is redirected to the section's activities page, which is displayed on Fig. 4.4. This page contains information about the user's progress

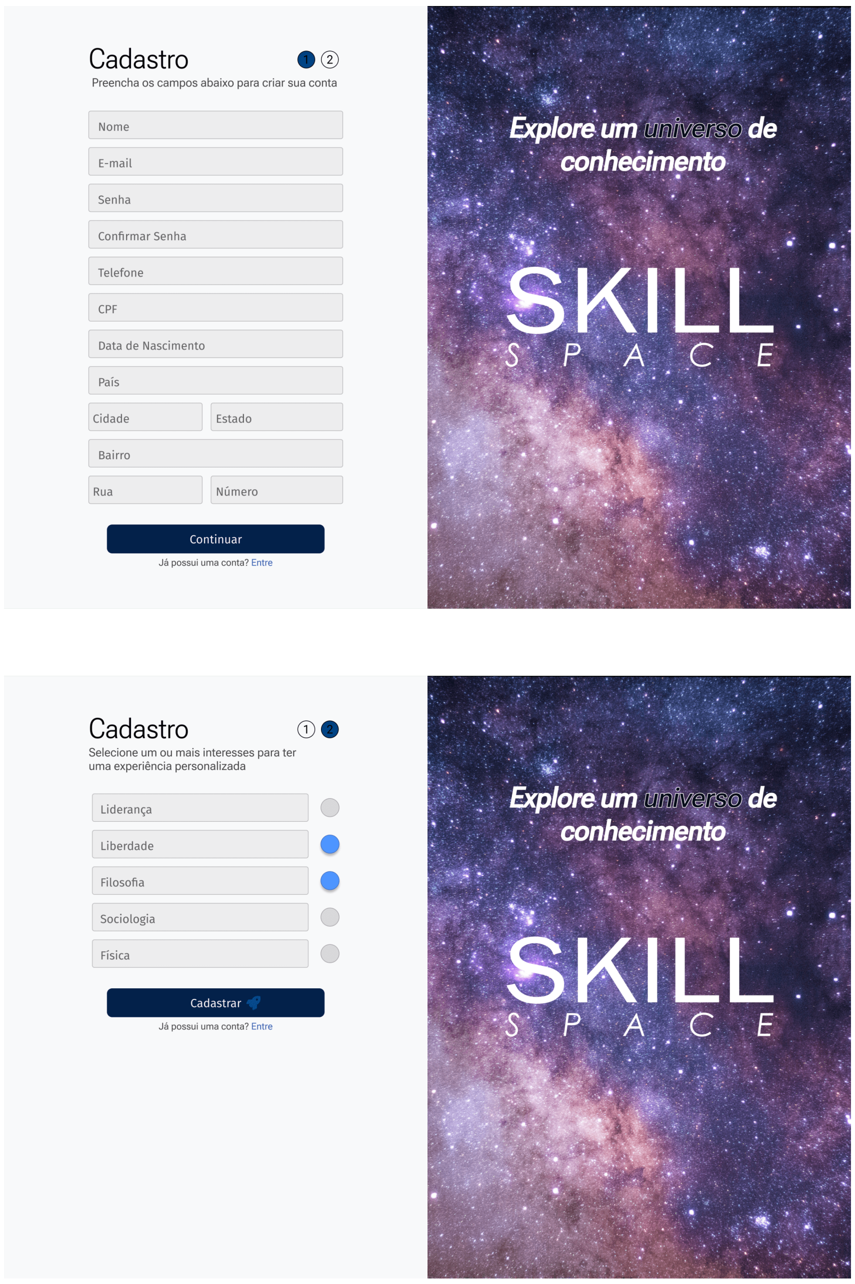

Cadastro
1
2
Preencha os campos abaixo para criar sua conta
Nome
E-mail
Senha
Confirmar Senha
Telefone
CPF
Data de Nascimento
País
Cidade
Estado
Bairro
Rua
Número
Continuar
Já possui uma conta? Entre
Explore um universo de conhecimento
SKILL
SPACE
Cadastro
1
2
Selecione um ou mais interesses para ter uma experiência personalizada
Liderança
Liberdade
Filosofia
Sociologia
Física
Cadastrar
Já possui uma conta? Entre
Explore um universo de conhecimento
SKILL
SPACE

Figure 4.2: Visual prototype of SkillSpace's two step signup page made on Figma. .

in the stages, similar to the progress information displayed on the stages page since the same component was used in both places. The main information present here is the list of activities which contains the name, type, category, status, and grade of the activity. In the last column of the activities table, there may be one to three icons. The downwards icon is the one present in all activities, and when clicked, it expands the activity row with its description, the files attached to it, and buttons to start, finish, and detail the activity. The other two icons are the exclamation mark, which denotes that the activity

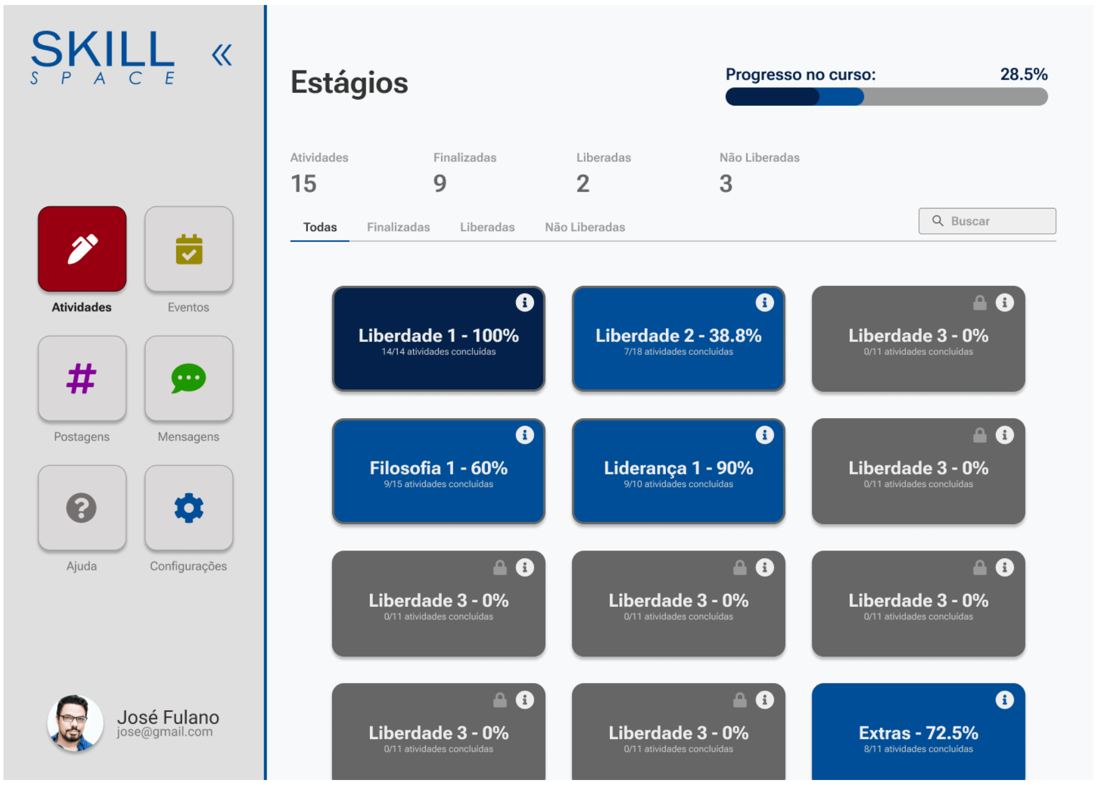


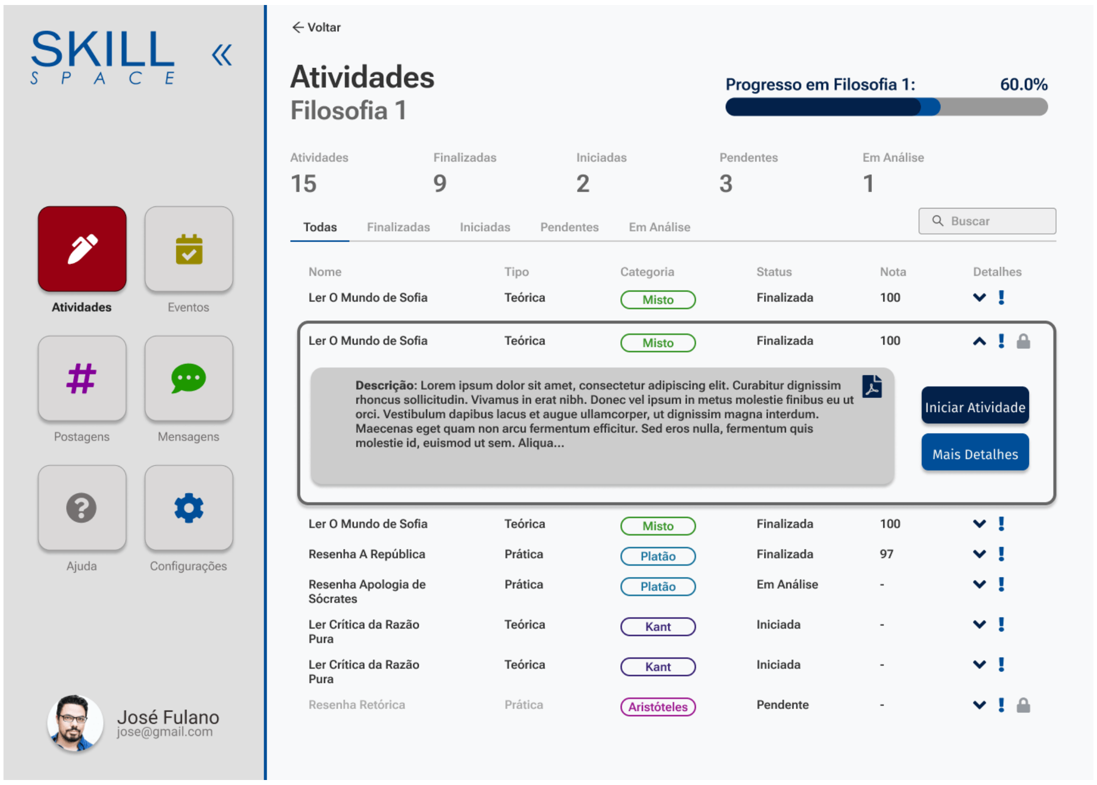
Figure 4.3: Visual prototype of the stages page made on Figma. .

is obligatory to finish that section and a lock icon which indicates that the activity is locked and may need another task to be completed before starting it.

At last, when the user clicks the more details button of activity, it redirects to the activity's dedicated page, which contains three tabs as described on Fig. 4.5. The first describes the activity's information, the next display the user's submission details, and the last has information about the teacher's feedback. We chose to display all the information in tabs to not overcrowd the activity's page with information. At all tabs, the user has access to the buttons to start or finish the activity as well as its grade if the activity is already reviewed by a teacher. When the user clicks the finish activity button, a submission modal is displayed for the user to write the description text and upload any necessary files for the activity.

4.3 Event Pages

The event-related pages of SkillSpace are very few, since the use of modals made it possible to fit the core of event features on a single page. It is possible to see in Fig. 4.6 that there is a page listing all the events as well as providing the most important information about it, like the date, the description, the type of event, and the invited speakers if they are



Atividades
Filosofia 1

Atividades	Finalizadas	Iniciadas	Pendentes	Em Análise
15	9	2	3	1

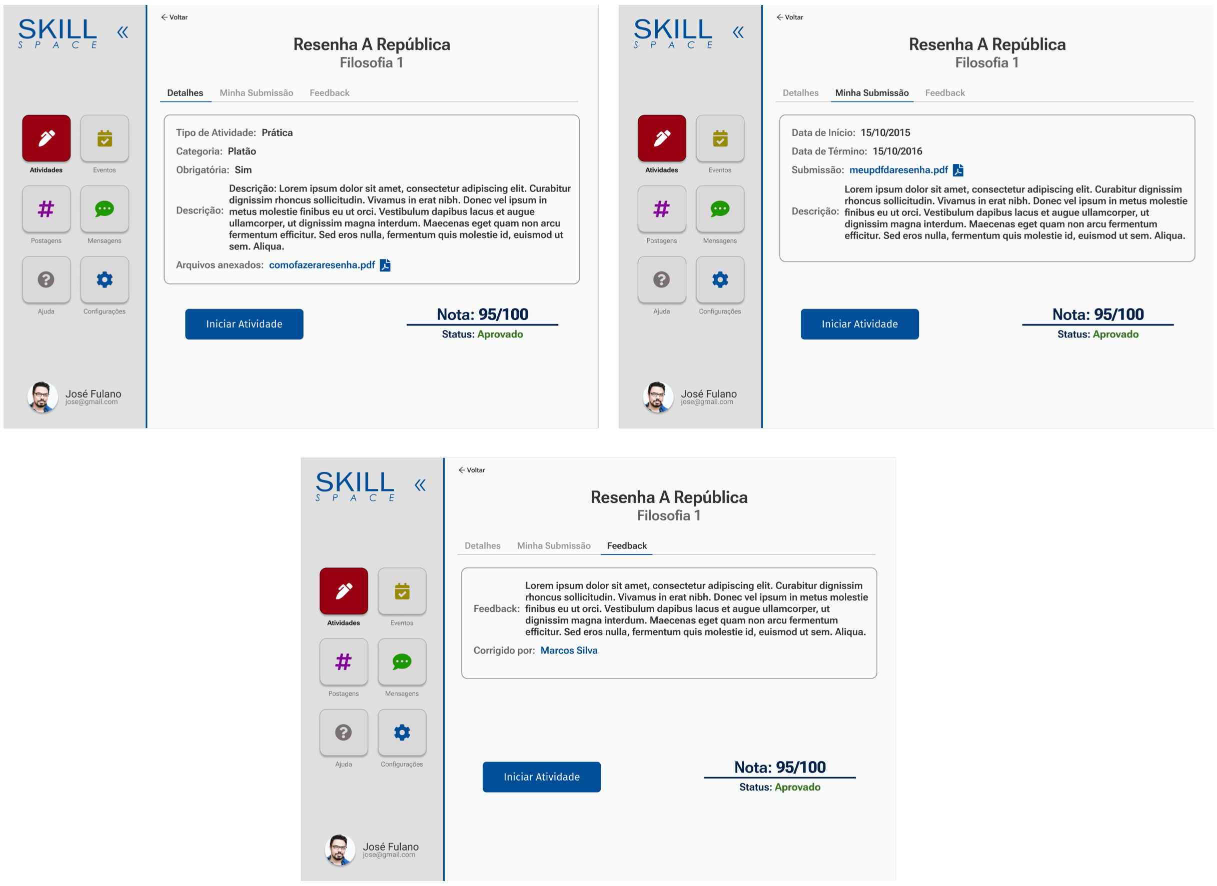
Nome	Tipo	Categoria	Status	Nota	Detalhes
Ler O Mundo de Sofia	Teórica	Misto	Finalizada	100	
Ler O Mundo de Sofia	Teórica	Misto	Finalizada	100	
Ler O Mundo de Sofia	Teórica	Misto	Finalizada	100	
Resenha A República	Prática	Platão	Finalizada	97	
Resenha Apologia de Sócrates	Prática	Platão	Em Análise	-	
Ler Crítica da Razão Pura	Teórica	Kant	Iniciada	-	
Ler Crítica da Razão Pura	Teórica	Kant	Iniciada	-	
Resenha Retórica	Prática	Aristóteles	Pendente	-	

Descrição: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur dignissim rhoncus sollicitudin. Vivamus in erat nibh. Donec vel ipsum in metus molestie finibus eu ut orci. Vestibulum dapibus lacus et augue ullamcorper, ut dignissim magna interdum. Maecenas eget quam non arcu fermentum efficitur. Sed eros nulla, fermentum quis molestie id, euismod ut sem. Aliqua...

Figure 4.4: Visual prototype of the activities page made on Figma. .

present. This page also has a list of event invitations that were sent to the current user. These events are highlighted by being at the top of the page, since they are the ones that need the user's attention to confirm or deny their presence in it. Finally, in the last section of the page, all the events of the platform are listed in chronological order, which the user can then filter by future or past events.

The core features that the normal user can interact with come when he clicks on the more details button of the event. This displays a modal, which is shown in Fig. 4.7, with all the information related to the event, which includes the link to the online event or a google maps location in case it is an in-person event. The modal also displays more information about the guest speakers, which can be used to lure more students to the event. This way, the administrator can use the presence of a renowned speaker to catch the users' interest in the event. At the end of the page, there will be some buttons for the user to confirm, deny or declare interest in being present at the event. It is important to notice that these buttons are only displayed in case the user was invited to the event, and it hasn't happened yet. When the user changes its status about an event, its color changes to reflect the new status, being blue related to having interest, green to confirmed, and red to denied. After the conclusion of the event, the user is then provided with an evaluation button to send some feedback to the organizers about the event. This feedback also has a



Resenha A República
Filosofia 1

Tipo de Atividade: Prática
Categoria: Platão
Obrigatória: Sim
Descrição: Descrição: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur dignissim rhoncus sollicitudin. Vivamus in erat nibh. Donec vel ipsum in metus molestie finibus eu ut orci. Vestibulum dapibus lacus et augue ullamcorper, ut dignissim magna interdum. Maecenas eget quam non arcu fermentum efficitur. Sed eros nulla, fermentum quis molestie id, euismod ut sem. Aliqua.
Arquivos anexados: comofazeraresenha.pdf

Nota: 95/100
Status: Aprovado

Resenha A República
Filosofia 1

Data de Início: 15/10/2015
Data de Término: 15/10/2016
Submissão: meupdfdaresenha.pdf
Descrição: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur dignissim rhoncus sollicitudin. Vivamus in erat nibh. Donec vel ipsum in metus molestie finibus eu ut orci. Vestibulum dapibus lacus et augue ullamcorper, ut dignissim magna interdum. Maecenas eget quam non arcu fermentum efficitur. Sed eros nulla, fermentum quis molestie id, euismod ut sem. Aliqua.

Nota: 95/100
Status: Aprovado

Resenha A República
Filosofia 1

Feedback: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur dignissim rhoncus sollicitudin. Vivamus in erat nibh. Donec vel ipsum in metus molestie finibus eu ut orci. Vestibulum dapibus lacus et augue ullamcorper, ut dignissim magna interdum. Maecenas eget quam non arcu fermentum efficitur. Sed eros nulla, fermentum quis molestie id, euismod ut sem. Aliqua.
Corrigido por: Marcos Silva

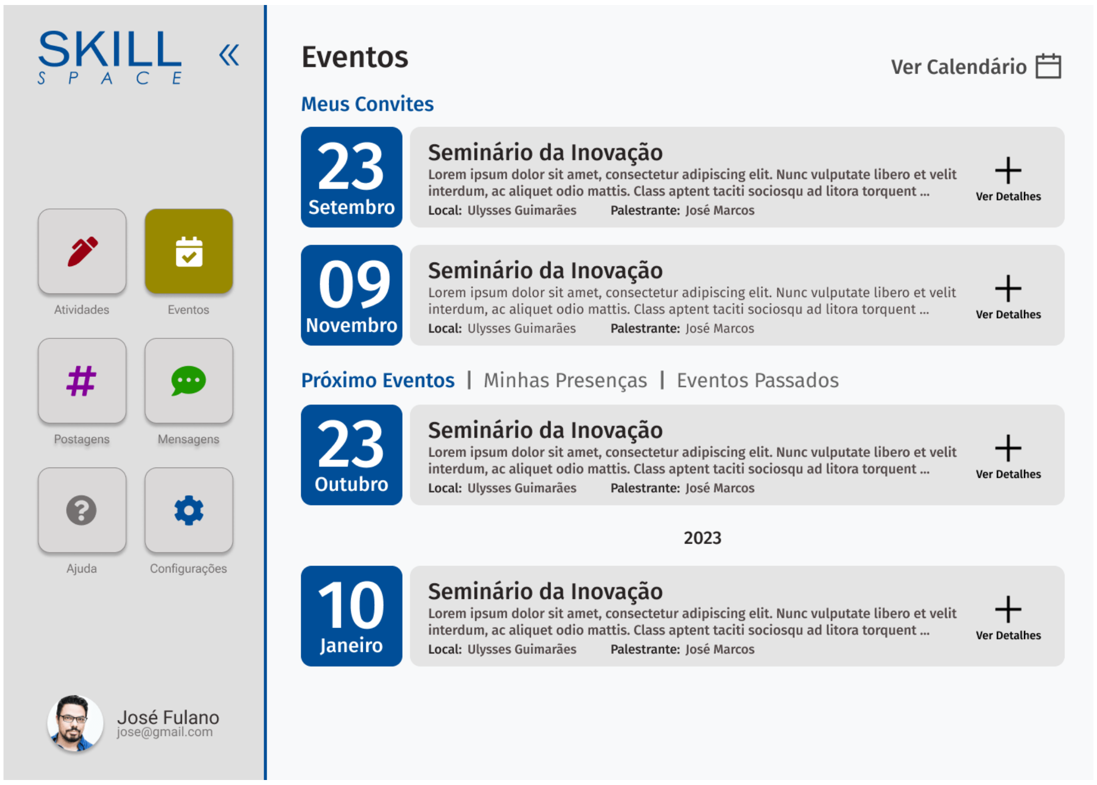
Nota: 95/100
Status: Aprovado

Figure 4.5: Visual prototype of the activities page made on Figma. .

checkbox to mark if the user wants the feedback to be anonymous or not. This was made to preserve the user's identity in case it does not feel comfortable sharing an opinion. This may occur when the user desire to express some negative opinion or critic. At last, we also planned a calendar page to display all the events in a calendar format. This page however was not prototyped due to it being not that necessary for the minimum product viable and due to time constraints.

4.4 Social Pages

The last pages prototyped are the ones that relate to user interactions. This section consists of the pages related to the user's post feed and the chat feature. Starting from the feed page, which is described in Fig. 4.8, we have a list of every post published by the people followed by the user, the posts marked with tags that interest the user, and the user's posts in chronological order. Each post displays the owner's basic information, the text of the post, and its tags. If a user wants to discuss any specific activity or event



Eventos

Meus Convites

23 Setembro
Seminário da Inovação
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc vulputate libero et velit interdum, ac aliquet odio mattis. Class aptent taciti sociosqu ad litora torquent ...
Local: Ulysses Guimarães Palestrante: José Marcos

09 Novembro
Seminário da Inovação
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc vulputate libero et velit interdum, ac aliquet odio mattis. Class aptent taciti sociosqu ad litora torquent ...
Local: Ulysses Guimarães Palestrante: José Marcos

Próximo Eventos | Minhas Presenças | Eventos Passados

23 Outubro
Seminário da Inovação
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc vulputate libero et velit interdum, ac aliquet odio mattis. Class aptent taciti sociosqu ad litora torquent ...
Local: Ulysses Guimarães Palestrante: José Marcos

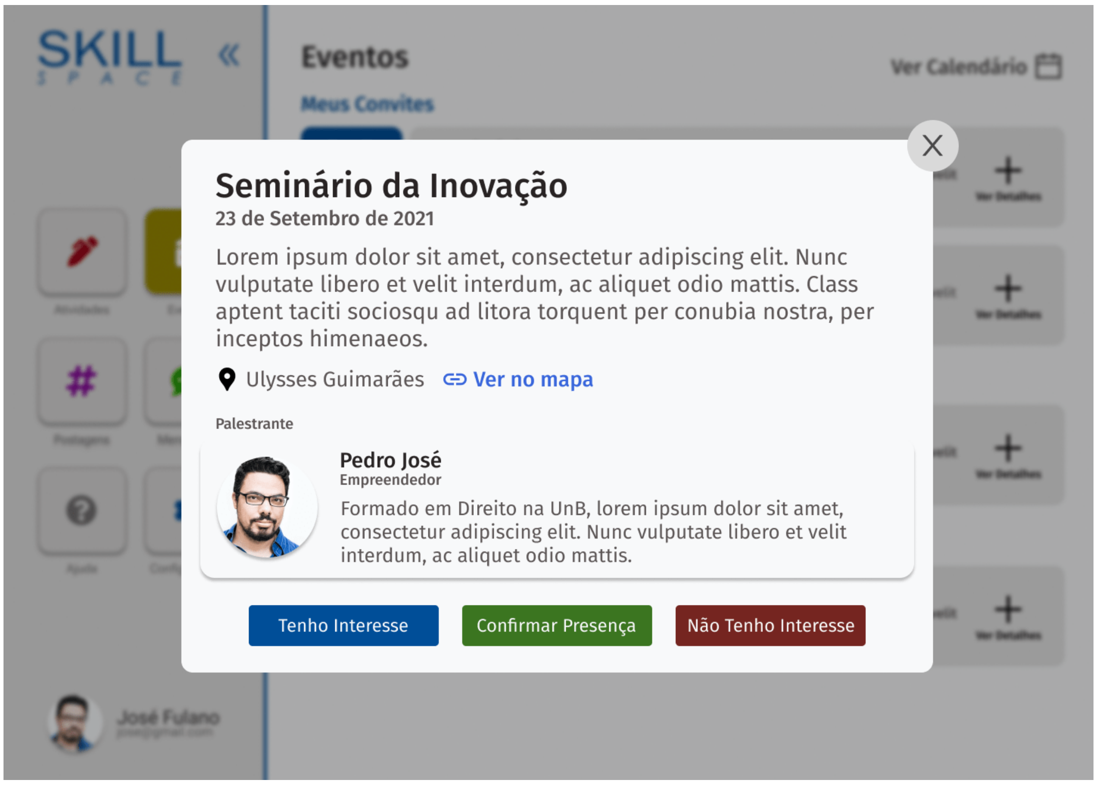
2023

10 Janeiro
Seminário da Inovação
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc vulputate libero et velit interdum, ac aliquet odio mattis. Class aptent taciti sociosqu ad litora torquent ...
Local: Ulysses Guimarães Palestrante: José Marcos

Figure 4.6: Visual prototype of the events page made on Figma. .

on the platform, it can link it to a post that creates a link to the referenced activity or event. This helps engage the user's in the platform by generating discussions about the platform's content. At the top of the page, there is a form to create a new post. We chose to display it at the top of the list of posts to encourage and facilitate the creation of new posts, thus increasing the interactivity of the platform. There is also a filter and a search function on the top to help users find posts more easily. Besides every post, there are some buttons to add a reaction to it, see the comments, and save it to read it afterward. When the user clicks on the comments button, a page specific to the post is loaded where it is possible to see all the comments in the form of threads. There, the user can then post a comment responding to a post or another comment.

Similar to the post feed, every user has a profile page as it is represented on Fig. 4.9 containing the posts he published and the activities he has started or concluded. There is a filter option to display only posts or activities too. In case, the user enters their profile page, the information about the course's completion is displayed as well as a button to edit its profile. On the other hand, if he enters the profile page of another user, buttons to follow, and to chat with the person are displayed. On both pages, some general information is displayed about the user like its following, follower, posts, and activities completed count.



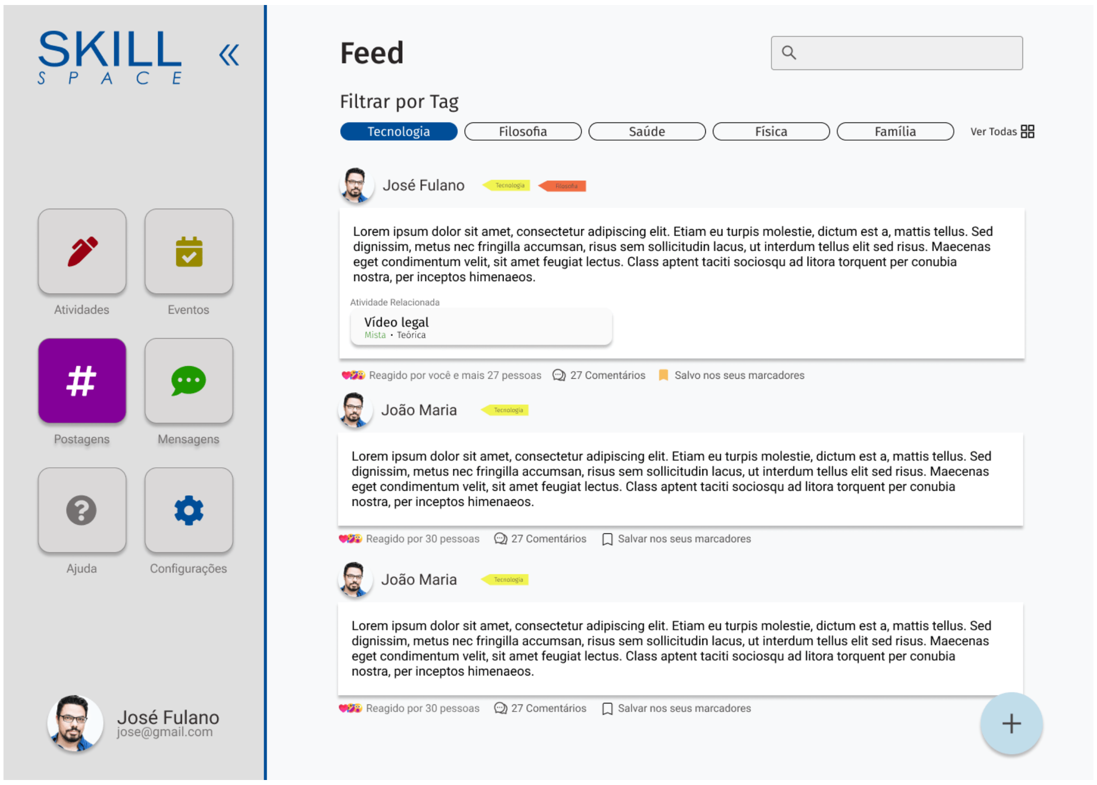
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc vulputate libero et velit interdum, ac aliquet odio mattis. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

Formado em Direito na UnB, lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc vulputate libero et velit interdum, ac aliquet odio mattis.

Figure 4.7: Visual prototype of the event modal displayed on top of the events pagge. .

At last, there is the chat page that works similarly to any conventional chat application and is represented on Fig. 4.10. It has a sidebar containing every person the user has ever talked with as well as a group feature to create a conversation with more than one person. When the user clicks on one of its contacts or one of its groups, the messages are displayed in chronological order, and the user can then write new messages to that person. This screen was added to keep every conversation between users inside the platform itself, this way there is no need to open another application and exit SkillSpace to start a conversation.

With the visual prototype of the platform completed, developing the actual frontend of the application becomes a much simpler task. Since there is no need to think about designing anymore, the focus becomes on coding the proposed visual onto a website, creating the logic behind it, and connecting with the previously created API. Describing how all these steps come together is what will happen in the following chapter.


SKILL SPACE «

Atividades · Eventos · Postagens · Mensagens · Ajuda · Configurações

José Fulano
jose@gmail.com

Feed

Filtrar por Tag

Tecnologia · Filosofia · Saúde · Física · Família · Ver Todas

José Fulano — Tecnologia · Filosofia

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eu turpis molestie, dictum est a, mattis tellus. Sed dignissim, metus nec fringilla accumsan, risus sem sollicitudin lacus, ut interdum tellus elit sed risus. Maecenas eget condimentum velit, sit amet feugiat lectus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

Atividade Relacionada
Vídeo legal
Mista • Teórica

Reagido por você e mais 27 pessoas · 27 Comentários · Salvo nos seus marcadores

João Maria — Tecnologia

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eu turpis molestie, dictum est a, mattis tellus. Sed dignissim, metus nec fringilla accumsan, risus sem sollicitudin lacus, ut interdum tellus elit sed risus. Maecenas eget condimentum velit, sit amet feugiat lectus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

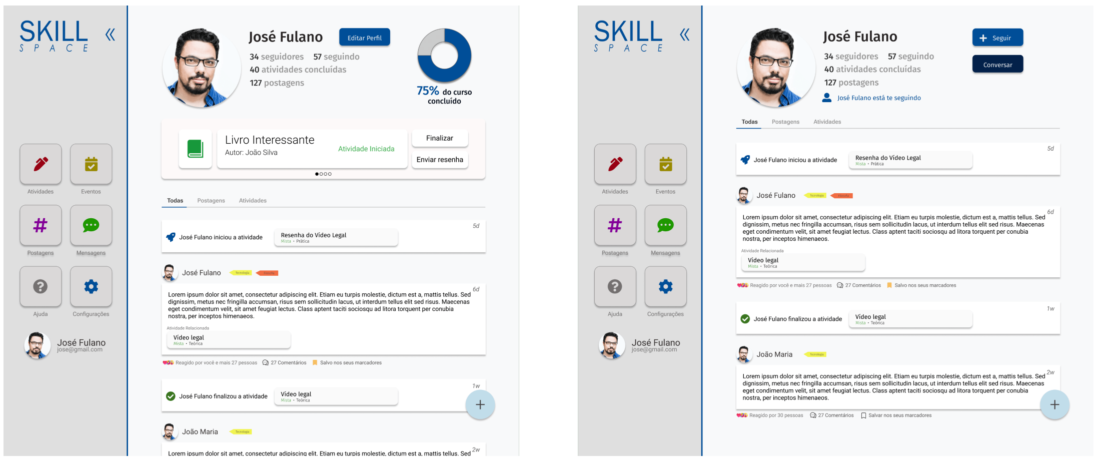
Reagido por 30 pessoas · 27 Comentários · Salvar nos seus marcadores

João Maria — Tecnologia

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eu turpis molestie, dictum est a, mattis tellus. Sed dignissim, metus nec fringilla accumsan, risus sem sollicitudin lacus, ut interdum tellus elit sed risus. Maecenas eget condimentum velit, sit amet feugiat lectus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

Reagido por 30 pessoas · 27 Comentários · Salvar nos seus marcadores

Figure 4.8: Visual prototype of the social feed page made on Figma. .


José Fulano · Editar Perfil
34 seguidores · 57 seguindo
40 atividades concluídas
127 postagens
75% do curso concluído

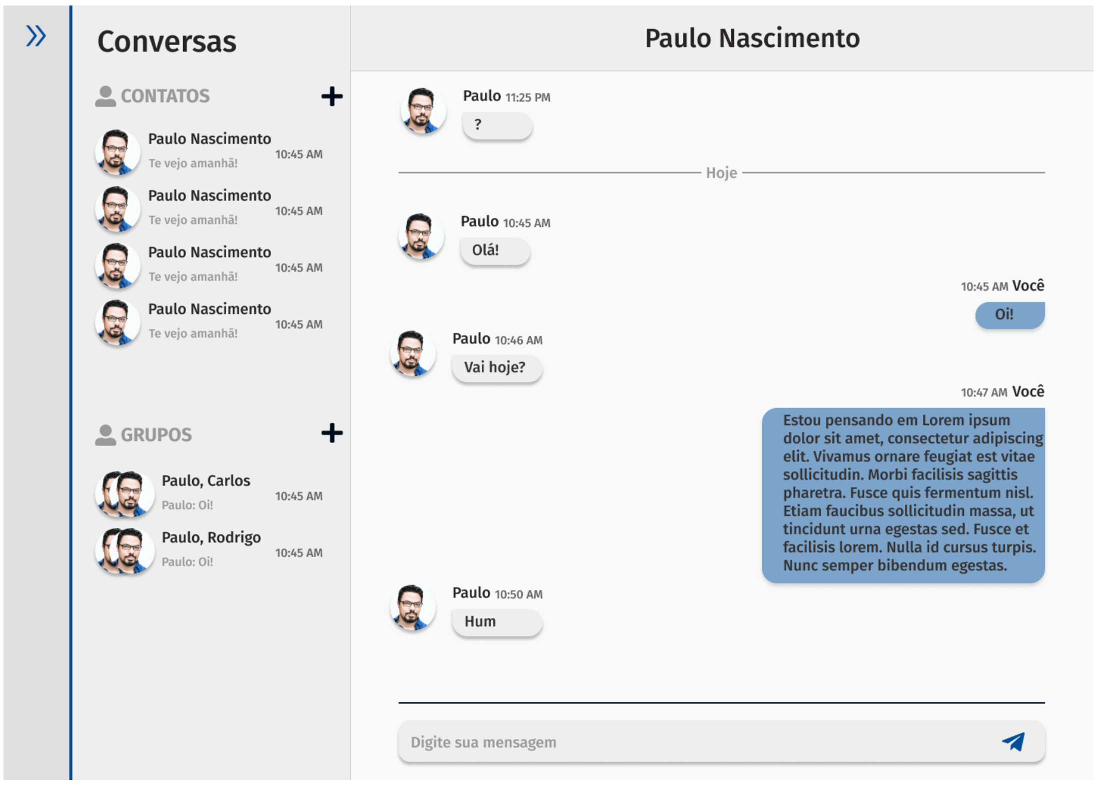
Livro Interessante
Autor: João Silva
Atividade Iniciada · Finalizar · Enviar resenha

Todas · Postagens · Atividades

José Fulano iniciou a atividade — Resenha do Video Legal
José Fulano finalizou a atividade — Vídeo legal

José Fulano · Seguir · Conversar
José Fulano está te seguindo

Figure 4.9: Visual prototype of the user's personal feed when accessed by himself on the left and when accessed by someone else on right. .



Paulo Nascimento 10:45 AM
Te vejo amanhã!

Paulo Nascimento 10:45 AM
Te vejo amanhã!

Paulo Nascimento 10:45 AM
Te vejo amanhã!

Paulo Nascimento 10:45 AM
Te vejo amanhã!

Paulo, Carlos 10:45 AM
Paulo: Oi!

Paulo, Rodrigo 10:45 AM
Paulo: Oi!

Paulo 11:25 PM
?

Paulo 10:45 AM
Olá!

10:45 AM Você
Oi!

Paulo 10:46 AM
Vai hoje?

10:47 AM Você
Estou pensando em Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus ornare feugiat est vitae sollicitudin. Morbi facilisis sagittis pharetra. Fusce quis fermentum nisl. Etiam faucibus sollicitudin massa, ut tincidunt urna egestas sed. Fusce et facilisis lorem. Nulla id cursus turpis. Nunc semper bibendum egestas.

Paulo 10:50 AM
Hum

Figure 4.10: Visual prototype of the chat page made on Figma. .

Chapter 5

The Frontend Logic

In this chapter, we describe some important concepts about the selected frontend technologies like authentication, routing, hooks, and componentization. This chapter also covers how our two applications, the API and the website, communicate with each other. At last, at the end of this chapter, the reader should have a complete understanding of how the models, controllers, and views of SkillSpace work.

5.1 Technologies

To develop a website, it is usually necessary to deal with three fronts: structure, style, and dynamic functionalities. These three fronts are rendered by a browser using respectively HTML, CSS, and Javascript (Js) code. Even in frameworks that use other programming languages like Ruby on Rails, Lavarel, and Django, the final code handled to the browser is purely HTML, CSS, and Js.

To build SkillSpace's website, we use ReactJs as our frontend library to unify the programming languages used in the project as Javascript. Apart from this reason, ReactJs is also an easy language to learn and to maintain the project afterward. This happens because ReactJs allows for elements to be componentized. Any visual elements can be converted into a function that returns HTML, and Js code thus providing a way to reuse any element and logic as a component. This is possible due to ReactJs code being written in JSX, a syntax extension to Js that permits HTML code to be handled inside regular Js code. Another advantage of using ReactJs is that the project can become a Single-Page Application (SPA). This way, the application works by rendering only a single HTML page, and all HTML, CSS, and Js necessary to render the page is handled in fragments by the server as they are needed instead of sending the whole page. This leads to very fast loading times since due to componentization, some sections of the pages may not need to be reloaded after changing the page like a user navigation bar or a footer for example.

5.2 Initial Setup

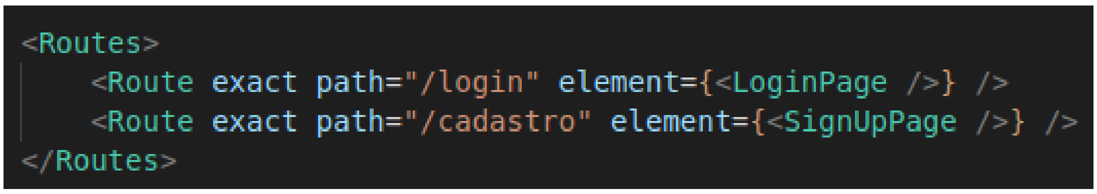
To start a ReactJs application, the first step is to create the project. To do so, a developer can either configure it from the ground up, creating all of its files, and adding the basic packages manually, or use the `create-react-app` to set it up for you. When using the second option, a project with all the basic files, configurations, and a simple folder structure is created. Usually, all the source code is put in the `src` folder, the `public` folder contains the files that will be read by the browser and the `package.json` contains the packages installed in the application.

The core files of a ReactJs project are its index.js and index.html. The index.js file is responsible for manipulating the DOM of the index.html page and injecting other elements into its root div. Thus, every page, component, and feature is rendered and executed on the index.html page by manipulating its DOM. Changing the URL on the website only changes what components are rendered in that situation.

5.3 Routing

Now that the basis of a ReactJs project is defined, we need to deal with rendering different "pages" depending on what the client requests. This works similarly to the routing logic on the API, it is necessary to define a router to select the page based on the current URL. To facilitate the creation of this router, we used the `react-router-dom` [23] package that provides features and components to manage the path and what is being rendered.

To create the router, we need to import the Routes component from `react-router-dom`. This component is responsible to render the first Route component that matches the current location. The Route component, which is also imported from `react-router-dom`, is the most important component of this package. It has two main parameters, one defines the path that is necessary to trigger it and the other is the element that will be rendered when its path matches the current URL. The element it renders is usually the component of a specific page. This way, we can build the HTML, CSS, and JS of every page as a JSX component that is loaded when its route is matched. An example can be seen in Fig. 5.1.



```
<Routes>
    <Route exact path="/login" element={<LoginPage />} />
    <Route exact path="/cadastro" element={<SignUpPage />} />
</Routes>
```

Figure 5.1: Example code of route management using `react-router-dom`. .

5.4 API communication

Now that the navigation of the site is working, we need to start working on the management of dynamic resources. Since our ReactJs application is a frontend pure project, it is necessary to rely on the previously created API to access and manage the database's resources. To do this, every time a page or a feature needs something from the API, it executes an HTTP call to it via the `axios` [24] package and returns a promise containing the response sent by the API.

Now that we have the resources in hand via the `axios` request, there is still one problem remaining, the API response is not received at the same time the page is loaded causing the page to render without the resources required. To solve this problem, it is necessary to use two ReactJs hooks which are a feature in this library that allows the use of states and other ReactJs core features without writing classes. The first hook is useEffect, which is used to create side effects that are triggered at some point after the component is rendered. In the API calls context, we use this hook to define the API call function to trigger right after the page is loaded. Therefore, the request can be processed asynchronously while the page is already rendered.

The next hook necessary to deal with this asynchronous data is useState. This hook is responsible for defining state variables that can be changed and re-rendered after the page is loaded. When this hook is called, it returns the state variable with an initial value and a setter for it, which needs to be called every time the state needs to be changed. This way, by using useState it is possible to set the data sent by the API after the page is loaded, which will make ReactJs re-render the necessary components in which the variable storing this data is used. By using a combination of all the cited tools, it is possible to render pages that deal with asynchronous API calls without the need to wait for the API response to arrive to render the page.

5.5 Authentication and sessions

Different from REST APIs, where it is not recommended to store sessions between requests, in a ReactJs application it is. This happens because at all times the application needs to track if a user is authenticated and who he is to render the correct elements on the page. Authentication is fairly simple on the frontend, being only necessary to call the login API route with the correct identifier and password, and then retrieve the user's JWT from its response. The problem here lies in storing this JWT and creating a session for that user. It is not possible to save such data in a state variable since it only retains the value inside the specific component it was defined. For example, it would lose the

JWT after switching pages. The solution here is to use another important ReactJs hook called "useContext". A Context provides a way to share information between multiple components inside the same component tree without the need to pass on this information to each child, working similarly to global variables. This way, it is possible to define a context that contains all of our routes, making it possible to access the user's JWT and any other context methods and variables on all of the pages.

After creating a session using the described hook, there is still one problem that remains. When the client refreshes the page or re-opens the browser, any context variables are lost and need to be set again since they are not persistent. To solve this problem cookies can be used to store the data on the user's browser. Cookies are small data fragments that can be stored on the client's browser to be retrieved after. Therefore, by storing the JWT as a cookie, it is possible to retrieve it if necessary at any time a user not authenticated access a page to re-create the context. If the user does not have this specific cookie stored, the context is not created and thus the user is not authenticated. At last, these session cookies normally have some expiration time or condition to make the user log in again and create a new token, improving its security.

5.6 Implemented Application

After defining the database modeling, the logic behind the used technologies, and the visual prototype, it is time to describe what was implemented and what was left to do. Since the scope of the project was rather extensive, we noted in the initial steps that would not be possible to implement all features in time. Therefore, we tried to build instead a Minimum viable product (MVP) of SkillSpace focused solely on the user-related features like the basic events, activities, and post logic. Thus it would be possible to present a usable website from the perspective of a regular user. The whole code of the project can be accessed from the GitHub repositories [25] [26] where it is also possible to access the site and the API deployed on Heroku and Netlify's cloud service.

Starting from the API, most of it was possible to implement in time due to it being the first part of the project itself. All the presented tables and relations were implemented using the `sequelize` package. Their controllers were also implemented, which contain all the necessary methods to show, list, create, update, and delete the data. In some controllers, as was mentioned in Chapter 3, other methods were also implemented to help deal with the data. All the planned routes were also implemented, but there is a possibility that, as the frontend part progresses, more routes and methods may be created. The authentication system, as well as the permissions system, are working together with the routing. We also managed to implement the file storing system, being possible to store files and images via local storage or store it on AWS's cloud storage.

On the API side, not much was left out of the final implementation. The main features that were left behind were the ones that required the creation of jobs that are executed apart from the requests the users send in the background of the application. One case that needs a job is the notification system that was originally planned to notify the user that something happened. Some examples of notifications are a teacher posting feedback on a user's activity submission, an event invite arriving or the receiving a message. Another part that was left incomplete was the creation of event-type activities. It is possible to create activities of this type but not yet possible to link the activity itself to an event, which should also send the invite to the marked users automatically.

The website part of SkillSpace was the one that needed to cut corners to deliver a usable product in time. The focus of it was on the user-related features to execute tasks, interact in the social area and participate in events. Because of this, the administrative part of the project was not implemented on the frontend even though its methods had been implemented on the API.

Starting from the authentication-related pages, it was possible to implement both the signup and the login pages with all the authentication and session logic working properly with useContext and cookies. The only part that was left out was the forgot password page and logic since they were not fundamental to the user experience. After logging in, the user sidebar is displayed although some of the icons are not linked with an actual page like the chat, help, and settings icons.

In the event section of the platform, the event index page and the event details modal were implemented. The calendar page was left out for this version since it is only another way to display events. The event modal's logic to confirm or deny the presence of an event is working properly. At last, the feedback modal also wasn't implemented due to time constraints.

In the activities section, we were able to implement the core pages of the user's course experience. It is possible to access the stages index page, the activities index page inside a section, and the details page of an activity, which displays the option to start and finish an activity. The activity submission feature is also working although only for the theoretical and practical activity types. The event and social activities had to be left out due to having a mostly different logic of completion in the frontend compared to the other two.

At last, since the main feature of the social section is the post feed feature, we had to leave the chat section out of this build. Other than that, the main features related to the feed were all implemented and works properly like the user page with its feed, the main feed with other users' feeds, the post creation feature, and the logic behind reacting and commenting on other users' posts. The main thing left out here is which posts are displayed in the social feed. Previously it was said that this feed displays the followed user's posts, the posts which contain tags that interest users, and some posts that are indirectly related to the user's interests to diversify the presented content. The last type had to be left out due to its higher complexity.

With the described implemented features, a regular user can experience all the main features that were proposed in this platform. The user can execute multiple types of activities, engage in discussions with other users about them via a post, and even participate in events to interact directly with people about the studied topics. All these features are integrated into a single platform which keeps the user engaged on the course and extends its use time by giving more variety of activities to be done.

Chapter 6

Conclusion

Creating a course while maintaining a community around it is no simple task. It takes more than just the study material to engage users in the platform for it to start developing a community. By providing the means of engaging the user with dynamic activities, and social-related features on the same platform, it becomes a lot easier to create a community and retain the user's interest in the course for a longer time.

In this work, it was possible to project, design, and develop a platform that implements a flexible course creation system with social activities and interactions. This should increase users' engagement by creating a community around the themes studied in the course. By using all three main sections of SkillSpace, activities, events, and social, it is possible to develop any type of course from it being more traditional to a more community-driven approach that relies not only on its content but on its users to increase the course's value.

All the features presented in SkillSpace should help the user feel more engaged to progress its studies, and even attract new users to the platform. This system should be of great use for course creators that have problems maintaining a community after the course's creation. By having other users interact and discuss the studied themes, even after there is no content left to do, some of them may remain on the platform posting new content, participating in events, and engaging with new users. Despite SkillSpace not being projected to work in school or the academic environment due to its lack of classes, it could still be used by teachers to manage a single class. We hope that by using our platform, people feel more engaged while studying, thus improving their learning experience.

6.1 Future Work

The greatest problem in the developed platform was not being able to implement all the features that were planned initially, especially the administrative ones since it leaves us with just the student's side of the platform. We hope that all incomplete features presented in chapter 6 could be completed in a near future. It is also important to point out that during all phases of the project, no surveys were conducted to gather the general public's opinion about the created features. It should be interesting to conduct surveys to further comprehend what works and what doesn't with the platform before creating a course in SkillSpace for real.

It would also be interesting to further improve this platform with more features that encourage the student to keep studying. One way to do that should be by adding a gamification system to the platform. This way, the student would receive points, badges, or titles that can be shown to other users to point out how much he dedicated to his studies. This could lead some users to treat its studies like a game of sorts thus encouraging them to spend more time with it. Another way is by creating more variety of activity types for example an exam-based type where the answers to the questions need to be done in real-time on the platform.

Another feature that could be interesting is to improve the platform's overall design by applying some user experience and user interface design knowledge to the already created features. This could make the site more intuitive and easy to use which could lead to more users using it. At last, we hope that by leaving the code of the project public, people can use it as a base to build even better and more complete platforms.

References

[1] *IFL Brasil's institutional website.* https://iflbrasil.com.br/, visited on 2022-25-09. 1

[2] *IFL Brasil's platform to manage associated member's progress.* https://iflportal.azurewebsites.net/, visited on 2022-25-09. 1

[3] Fielding, Roy Thomas: *Introduction of the representational state transfer (rest) architectural style.* https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, visited on 2022-25-09. 3

[4] *Node.js's official website.* https://nodejs.org/en/, visited on 2022-25-09. 3

[5] *Express.js's official website.* https://expressjs.com/, visited on 2022-25-09. 3

[6] *React's official website.* https://reactjs.org/, visited on 2022-25-09. 3

[7] *Ruby on Rails's official website.* https://rubyonrails.org/, visited on 2022-25-09. 3

[8] *MySQL's official website.* https://www.mysql.com/, visited on 2022-25-09. 5

[9] *dbdiagram.io's official website.* https://dbdiagram.io/home, visited on 2022-25-09. 6

[10] *aws-sdk node.js package repository.* https://github.com/aws/aws-sdk-js, visited on 2022-25-09. 17

[11] *bcryptjs's node.js package repository.* https://github.com/dcodeIO/bcrypt.js, visited on 2022-25-09. 17

[12] *body-parser's node.js package repository.* https://github.com/expressjs/body-parser, visited on 2022-25-09. 17

[13] *cors's node.js package repository.* https://github.com/expressjs/cors, visited on 2022-25-09. 17

[14] *dotenv's node.js package repository.* https://github.com/motdotla/dotenv, visited on 2022-25-09. 17

[15] *express-validator's node.js package repository.* https://www.npmjs.com/package/express-validator, visited on 2022-25-09. 17

[16] *jsonwebtoken's node.js package repository.* `https://github.com/auth0/node-jsonwebtoken`, visited on 2022-25-09. 17

[17] *multer's node.js package repository.* `https://github.com/expressjs/multer`, visited on 2022-25-09. 17

[18] *multer-s3's node.js package repository.* `https://github.com/anacronw/multer-s3`, visited on 2022-25-09. 17

[19] *mysql2's node.js package repository.* `https://github.com/sidorares/node-mysql2`, visited on 2022-25-09. 17

[20] *sequelize's node.js package repository.* `https://github.com/sequelize/sequelize`, visited on 2022-25-09. 17

[21] Jones, Michael B., John Bradley, and Nat Sakimura: *Json web token (jwt).* RFC, 7519:1–30, 2015. 18

[22] *Figma's official website.* `https://www.figma.com/design/`, visited on 2022-25-09. 20

[23] *React Router's official website.* `https://v5.reactrouter.com/`, visited on 2022-25-09. 31

[24] *Axios package's repository.* `https://github.com/axios/axios`, visited on 2022-25-09. 32

[25] Vaz, Felipe L. and André M. P. Vale: *SkillSpace backend repository.* `https://github.com/andremacedopv/skill-space-api`, visited on 2022-27-09. 33

[26] Vaz, Felipe L. and André M. P. Vale: *SkillSpace frontend repository.* `https://github.com/andremacedopv/skill-space-front`, visited on 2022-27-09. 33

Appendix A

API Routes

Resource	**Route(s)**	**Description**
-	/signup	Creates a new user
-	/login	Log a user in the platform
/profile	Show and update routes	Standard CRUD methods for the `User` model for the logged user
/profile	/update/picture	Update the logged user's profile picture
/user	Show and index routes	Standard CRUD methods for the `User` model for admins
/user	/promote/:id	Promotes a user to administrator
/user	/demote/:id	Demotes a user from the administrator level
/user	/permissions/:id	Returns a user's `AdminPermissions`
/user	/activate/:id	Changes the user's status to active
/user	/deactivate/:id	Changes the user's status to deactivated
/address	CRUD routes	Standard CRUD methods for the `Address` model
/event	CRUD routes	Standard CRUD methods for the `Event` model
/event	/:guests/:id	Returns the event's guests

/event	/invite/:id	Invites users to an event
/event	/my_invites	Returns the logged user's event invites
/event/:event_id/guest	Update, show and delete routes	Standard CRUD methods for the `Guest` model
/event/:event_id/guest	/presence	Confirms presence of the logged user on an event
/event/:id/feedback	Create and index routes	Standard CRUD methods for the `EventFeedback` model
/invited_speaker	CRUD routes	Standard CRUD methods for the `InvitedSpeaker` model
/activity	CRUD routes	Standard CRUD methods for the `Activity` model
/activity/types	CRUD routes	Standard CRUD methods for the `ActivityType` model
/activity/category	CRUD routes	Standard CRUD methods for the `Category` model
/activity/submission	Create, Update and show routes	Standard CRUD methods for the `ActivitySubmission` model
/activity	/user	Returns the logged user's activities started or finished
/activity	/start/:id	Changes an ActivityUser's status to started
/activity	/finish/:id	Changes an ActivityUser's status to finished
/activity/manager/feedback	/pending	Returns the list of `ActivitySubmissions` that have pending feedback
/activity/manager/feedback	/:act_id/:user_id	Create an `ActivityFeedback` for a user's submission
/activity/manager/feedback	/edit/:act_id/:user_id	Edits an `ActivityFeedback` for a user's submission
/activity/requirements	Create, and index routes	Standard CRUD methods for an activity's required activities

/activity/dependents	Create, and index routes	Standard CRUD methods for an activity's dependent activities
/stage	CRUD routes	Standard CRUD methods for the `Stage` model
/stage/my	Index and show routes	Standard CRUD methods for `StageUser`'s of the logged user
/stage	/start/:id	Changes an `StageUser`'s status to started
/stage	/finish/:id	Changes an `StageUser`'s status to finished
/post	CRUD routes	Standard CRUD methods for the `Post` model
/post	/feed	Returns the list of posts of a user's social feed
/post	/feed/user/:id	Returns the list of posts of a user's personal feed
/post	/:id/comments	Returns the comments of a `Post` model
/post/reaction	Create, delete and show routes	Standard CRUD methods for the `Reaction` model
/post/reaction	/count/:id	Returns the reactions count of a post
/tag	CRUD routes	Standard CRUD methods for the `Tag` model
/tag	/follow/toggle/:id	Creates or destroy an `UserTag`
/tag	/followed	Lists the user's followed tags
/follow	/:id	Follows a user by creating a new `Follower` model
/follower	/:id	Returns an user's followers
/follower	/count/:id	Returns an user's followers count
/following	/:id	Returns the user's that another user is following
/follower	/count/:id	Returns an user's following count

/chat	Index and show routes	Standard CRUD methods for the `Chat` model
/chat	/my	Returns the list of chats that contains the logged user
/message	Create, update and delete routes	Standard CRUD methods for the `Message` model
/permission	Index route	Returns the list of permissions in the platform

Table A.1: Table with API routes with a brief description of each one of them.