Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

SkillSpace: A learning environment with social capabilities

André Macedo P. Valle

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Orientador
Prof. José Edil Guimarães De Medeiros

Brasília
2022

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

SkillSpace: A learning environment with social capabilities

André Macedo P. Valle

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

Brasília, 05 de outubro de 2022

Agradecimentos

Agradeço à minha mãe por todo amor e cuidado que recebi durante toda minha vida, e por todo suporte que tive durante meu perído de graduação.

Agradeço também aos meus irmãos Marcella e Gabriel e ao meu sobrinho Samuel, que até mesmo em momentos mais distantes, sei que estão lá por mim.

Agradeço à minha avó Nadir e ao meu avô Edmundo por todo o amor e carinho que sempre deram à mim e a todos os seus netos. Sei que nem sempre consigo estar presente junto deles, em especial desde a pandemia, mas sempre sinto o amor deles perto de mim.

Agradeço às minhas tias e primos por todo o carinho que sempre recebi e por todos os momentos bons que tivemos com a família toda reunida.

Agradeço aos meu fiel grupo de amigos, Gabriel, Juliana, Luiz, Nathalia e Pedro, por tudo que vivemos juntos desde o ensino fundamental. Os momentos que tive com cada um estarão sempre guardados no meu coração.

Agradeço as amizades que fiz na UnB, Danilo, Felipe, João, Kalley, Eduardo, Hevelyn, Otho e tantos outros, por todas as experiências que compartilhamos e momentos que passamos juntos nessa graduação. Vocês definitivamente tornaram o processo de graduação mais fácil e divertido. E ao Felipe que realizou esse projeto junto comigo, agradeço em dobro. Se não fosse por ele, esse projeto nem sequer existia.

Agradeço à Struct e a todo mundo de lá por todo o conhecimento que obtive ao participar da empresa e por ter contríbuido por talvez a parte mais importante de todo o meu período na UnB. Agradeço em especial ao Arthur, Felipe, Venzi, Kayran e Xavier pelo tempo que passamos juntos como diretores da empresa.

Agradeço ao Edil por ter sido meu orientador nesse projeto e por todas as reuniões que tivemos nesse períodos e às dicas e aconselhamentos que recebi, que irei levar para além dessa monografia.

Por fim, agradeço à UnB e a todos os professores que tive durante minha graduação por todos os momentos inesquecíveis que vivenciei nesse período e por todo conhecimento que obtive.

Resumo

Esse trabalho tem como objetivo propor e projetar uma plataforma online capaz de prover aos usuários um ambiente social e de aprendizado. Essa plataform permite que seus administradores organizem um fluxo de aprendizado com a criação de atividades e disponibilização de material didático, classificando também essas ativdades em estágios para fins de organização da plataforma. Com o objetivo de aproximar os usuários, a plataforma também provê um ambiente de rede social, em que os conteúdos aprendidos podem ser debatidos e compartilhados, além de uma funcionalidade de eventos, para promover palestras e encontros entre os membros. Como objetivo final, é esperado que os organizadores que optarem por usar a plataforma consigam aumentar o engajamento com o curso e criar uma comunidade ao redor dos tópicos debatidos.

Palavras-chave: educação, plataforma online, rede social

Abstract

This work delves into designing and projecting an online platform capable of providing to its users a social and learning environment. This platform allows its administrators to organize a learning flow with the creation of activities and providing educational materials, classifying these activities into stages in order to better organize the platform. In order to get the users close to each other, the platform also provides a social network environment, in which the educational content provided can be discussed and shared, also featuring an event functionality, to provide lectures and meetings between the members. As the end goal, it is expected that the organizers that opt into using the platform are able to increase the participation with the course and create a community around the debated topics.

Keywords: education, online platform, social network

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface.

CORS Cross-Origin Resource Sharing.

CRUD Create, Read, Update and Delete.

CSS Cascading Style Sheets.

DOM Document Object Model.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

IFL Instituto de Formação de Líderes.

JS JavaScript.

JSON JavaScript Object Notation.

MVC Model-View-Controller.

NPM Node Package Manager.

SQL Structured Query Language.

UI User Interface.

URL Uniform Resource Locator.

Chapter 1

Introduction

The `Instituto de Formação de Líderes (IFL)` [1] from Brasília is an institution that focus on the formation of leaders, with the goal to instruct young people in Brazil, capable of putting in practice their leadership, entrepreneurship, and management abilities. To achieve these goals, they not only provide recommendations of books and articles to be read by its members, but organize events for them and create an environment where the members can share their thoughts and discuss about either the content and events provided by the organization, or any other relevant topic that they may find pertinent for the other members.

Generating this environment where each member can easily obtain the desired study materials and discuss it with other members, bridging the gap between the two, is a challenge. While `IFL` has an online platform [2] to register its members and provide the contents to be studied by them, the way they are presented may not be ideal for the institution needs. Not only there could be more flexibility on the way the components curriculum is structured, but there could also be more proactive exercises, influencing its members to write more about what they learned and practice their knowledge. When it comes to the social aspect, their platform lacks any meaningful way for members to interact with each other, with this interaction being left to happen on other social services, such as WhatsApp or Discord, and while these are good general use social platforms, they fail to create a unified learning environment for the members, that can easily reference the study materials and separate all the discussions happening into its relevant topics.

While the situation described until now is focused on `IFL`, a platform that manages to provide a learning environment for its users, with access to both a learning and a social space on the same place can be useful to several other institutions with the same goals as `IFL`, and also to organizations and individuals that desires to offer online courses, a modality that saw a great spike in popularity in recent years, specially after the pandemic. For this last public, that have mostly been selling courses through social networks like

Instagram or using platforms such as Hotmart and Udemy, which lack real social features besides a comment section, an attempt to create a social environment between their students is made by providing access to services such as Discord servers, which can fail to reach all the students due to being an external platform to where they are consuming the course, or advising the students to use the comment section these platforms have, which does not have the organization and resources you would expect from an online social environment.

One option one would have to solve this issue would be to use Moodle, a learning platform used in many educational environments such as universities, and, while Moodle does features a highly customize course structure and social features like chat rooms and forums, its use can be challenging to non-tech savvy users. For those setting up the course, Moodle may require the exploration of an extensive documentation to properly that may not be intuitive for some one just wanting to create or distribute a course, and for those actually using the platform, Moodle does not feature an intuitive user interface that is easily understandable by a tech illiterate, which the course may appeal to.

To solve this problem we envisioned the SkillSpace platform, which has the objective of bridging the gap between a social and a learning environment, featuring an activity center with every theoretical and practical activity that members in the platform should do, a social network-like feed where users can post their thoughts and engage in meaningful discussions with other people and an events area, where administrators can invite members to events and speeches organized by them.

Regarding the activity area, administrators will have the ability to register theoretical activities, representing content users should use to study, such as books, articles, videos, podcasts, or any other material relevant to the space, practical exercises where users can make their submissions and receive feedback from the organizers, and also social and events activities, that promotes the creation of posts on the social area and the participation in events, respectively. To better organize these activities, administrators can provide a flux which determinates the order activities should be done by defining requirements, with activities being locked until all are met, and with the creation of stages, which group various activities in the same level or topic together, while also allowing stages to be requirements for other stages to be unlocked, as with the activities.

The social area will have its focus on a post feed resembling the ones already made familiar by other social networks and platforms, with any user being able to create a new post on the space, therefore starting new discussions and conversations with its peers, and finding and interacting with posts made by other users, either by reacting to it using emojis or commenting on it, further engaging the discussion. The platform also has the goal of making the discussions started on it easily searchable and organized, by providing ways to search for older discussions and categorizing the posts by appropriate tags, set up by the administrators. We also acknowledge that it may be desirable by some to have private conversations and discussions with other users without having to resort to other services, where they might need to share private information such as phone numbers, for that reason, a chat feature is also available, allowing users to engage in private chat rooms inside the SkillSpace platform.

Lastly, the events functionality allows administrators to register events and speeches that will happen and invite the users to participate. The invitations can happen in a general way, inviting all users registered in the platform, or specifically selecting those that they think should participate. User then can find all the events that are bound to happen in the events area of the platform, both the ones they have been invited and other that are also scheduled to happen and state their intention of participating in the event. They can also leave feedback for the events they attended, which can then be used by the organizers to improve future events.

The scope of this work will focus on the implementation and organization of the SkillSpace platform, and all the parts needed to make it work. In chapter 2 the focus will be in designing the database that will store all relevant information on the platform, while in chapter 3 we will see the implementation of the back-end part of the application, focusing on the logic behind the implemented services, the database implementation and the creation of the API that will provide the service. In chapter 4 we will discuss the prototype of the UI that users will interact on the final product, which will have its implementation addressed on chapter 5.

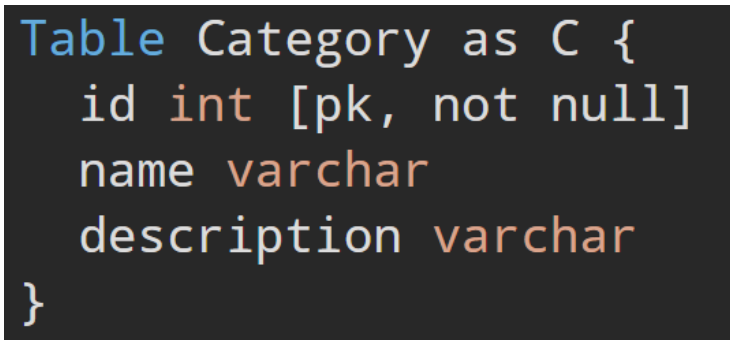
Chapter 2

Database Design

The development process of the SkillSpace platform began by designing and developing the back-end portion of the application. The first stage of the design process was to decide the structure of the database itself and how it would be organized to fulfill the needs of the desired system.

The first decision made in the development of the platform was about the type of database to be used, with the two options being relational and non-relational databases (also known as `SQL` and `NoSQL` databases, respectively) [3]. While relational databases features a more strict structure of the data, organizing it in tables with pre-defined fields and fixed relationships among the tables, non-relational databases allows for more flexibility while storing the information, with no pre-defined structure of what each data set stores and how it relates to other in the system. While non-relational databases are really useful for systems where the information being stored is not always predictable, also providing better scalability for the system, the decision to use relational databases was made due to the predictable nature of the SkillSpace platform design, having a low need for a flexible option like `NoSQL`, while reaping benefits from relational database, such as data integrity, due to the rigid nature of the table and its fields, and easier execution of complex queries that involves many of the tables and its relationships.

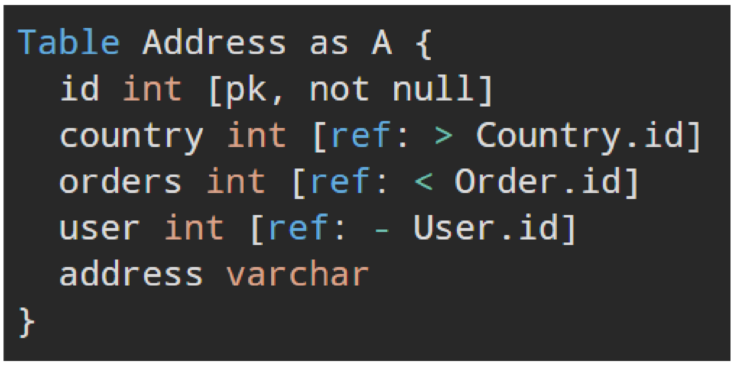
To properly design the database structure, the online tool `dbdiagram.io` [4] was used to plan the tables needed to match the proposed system criteria, alongside its fields and relationships. The tool features its own syntax language that is used to define all the elements present in the database. The definition of a `Table` using the tool syntax can be seen in Fig. 2.1, where we define the table name after the `Table` keyword, an alias for the table using the `as` keyword, and, inside the `Table` block, each field is defined, followed by its corresponding type. A field that is present in most of the tables designed is the `id`, which functions as the primary key of the table, with requirements of not being `null` nor repeating in any instance of the table.



```
Table Category as C {
  id int [pk, not null]
  name varchar
  description varchar
}
```

Figure 2.1: Table definition in the dbdiagram tool.

To define the relationships on the tool, a reference field is added inside the table definition, stating both the type of relationship between the tables and which field from which table is being referenced (usually the `id` field), with examples of three types of relationship being shown in Fig. 2.2. In the example figure, the `Address` table is in a many-to-one relationship with the `Country` table, by using the `>` operator, meaning that one instance of the `Country` can be referenced by multiple `Address` instances. The opposite is shown in the one-to-many relationship between `Address` and `Order`, with the use of the `<` operator, where one `Address` can connect to multiple `Orders`. The final relationship is the one-to-one, achieved using the `-` operator, where only one connection is possible for both sides of the relationship in each instance.

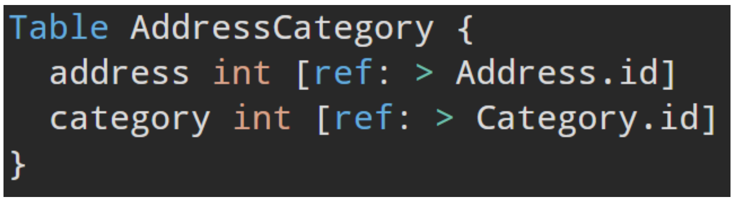


```
Table Address as A {
  id int [pk, not null]
  country int [ref: > Country.id]
  orders int [ref: < Order.id]
  user int [ref: - User.id]
  address varchar
}
```

Figure 2.2: Defining relationships on dbdiagram..

Many-to-many relationships, on the other hand, are defined by creating a join table between the two desired tables. This special table makes many-to-one references to each table being joined, with both working together as primary keys of the join table, removing the need of a separate `id` field on it, as it is shown in Fig. 2.3. These join tables may also feature more field than the tables references, if needed.

With the tables being defined using the tool syntax, dbdiagram also generates a visual design of the modeled database, which will be used during thorough this document to illustrate the tables being described.



```
Table AddressCategory {
  address int [ref: > Address.id]
  category int [ref: > Category.id]
}
```

Figure 2.3: Defining a join table on dbdiagram.

2.1 The User Model

The User model represents the instances of the registered users on the platform, containing its basic personal and login info, such as the email and a field to store the encrypted password. There is also an Address model in the database, associated with the User model in a one-to-one relationship, with its instances created alongside the users registrations. Some Boolean fields in the User model are used to determinate the role and status of the user on the system, these being the admin, owner and is_active fields.

The owner and admin fields, both represents administrators on the platform, with access to areas and configurations not available to regular users. The difference between them is, while the owner has full access to all the configurations available on the platform, the admin is limited by the permissions given to him. These permissions are defined by another model in the database with each instance having a unique name that represents the functionality associated with the permission, such as registering a new event or giving feedback to user submissions, for example. The Permission and User models are associated in a many-to-many association using a junction table, allowing one single administrator to have multiple permissions on the platform.

An overall view of the User, Address, Permission and AdminPermission models can be seen in Fig. 2.4, which features the models and its fields as it was designed in prototyping stages using the online tool dbdiagram.io.

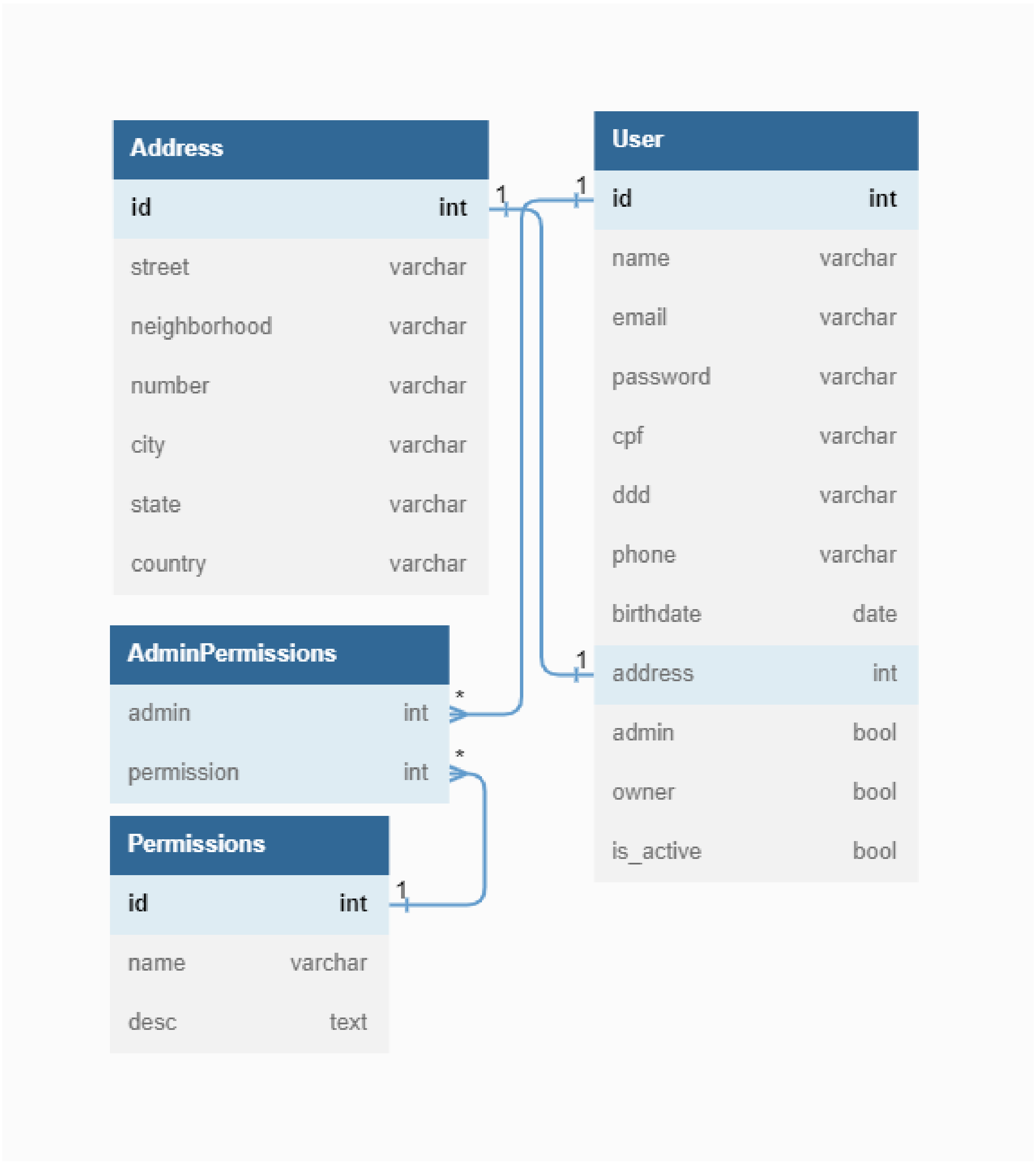


Figure 2.4: Diagram of the User Model and its associations.

2.2 The Activities Model

In the center of the activities area from the SkillSpace platform is the `Activity` model, which represents the core content and tasks the platform administrators believe the registered users should consume and partake. The model, then, stores the necessary information to describe and identify the activity, such as `name` and `description`. A join table is also created between the `Activity` and `User` models, to identify the activities which the user started and finished, with new entries of the join table being created when the user starts an activity, which is then updated when the user finishes it, registering both started and finished dates to create a track for the time the user used to complete the activity. Also associated with the activity model are the `Category` and `ActivityType` tables, the first one representing categories that administrators can create to better categorize the activities into topics that makes sense for their platform, while the second one represents the types that each activity can assume on the platform, with the four available being defined directly on the database.

2.2.1 Theoretical Activities

As the name implies, theoretical activities are the ones which only features a material to be consumed by the user, such as a book, article, or video. Administrators can directly upload the file to the system and make it available for users to download or directly link it on the description field. Being only a theoretical activity, users can start and finish the activity without submitting anything.

2.2.2 Practical Activities

As far as registering the activity goes, practical activities (also referred as exercises) can feature everything the theoretical ones have, including file uploads for material to be used by the user, the key difference here is that these exercises require a submission to be sent from the user when finalizing the activity, which comprises of either a text written directly from the platform or a submission file, and is stored in the `ActivitySubmission` table in the database, which is directly linked to the `Activity` and `User` join table. Once submitted, its vital for the user to be able to receive feedback on the work they have done, allowing them to improve on their skills and acknowledge where they are doing right or wrong. For that purpose, an administrator with the proper privilege can access the submission and provide the user with feedback of their work, which can include a text detailing the feedback with its praises and corrections and an optional numbered score. These feedbacks are stored in a separate model, `ActivityFeedback` also linked to the `Activity` and `User` join table.

2.2.3 Social Activities

The social activities exist to further encourage users to start discussions on the social area of the platform, as such, to finalize this type of activity the user is required to make a post on the platform directly from the activity page. Moreover, administrators can also specify a tag that the post should be a part of to finish the activity.

2.2.4 Events Activities

The last type of activities available are the events ones, which requires the participation in an event on the website to be completed. These are the only activities that are not finalized by the user themselves, instead being marked as complete once an administrator confirms the user presence on an event while they have already started the activity on their dashboard.

2.2.5 Activities Requirements

While registering the activities on the platform, the administrators might require that users complete some activities before starting another. For that reason, the `ActivityRequirement` table on the database stores a relationship of requirement between two activities. Users are then unable to start a certain activity if all the requirements for it are not yet met.

In Fig. 2.5, we are able to see the schematics of the `Activity` model and its associations as described above. In the figure, there is also the representation of other models that, while not being strictly a part of the activities area, are connected to it, such as `Stage`, `User`, `Event`, `Tag` and `Post`. The fields of these models were omitted for organization purposes and the details of each one will be better described in the following sections.

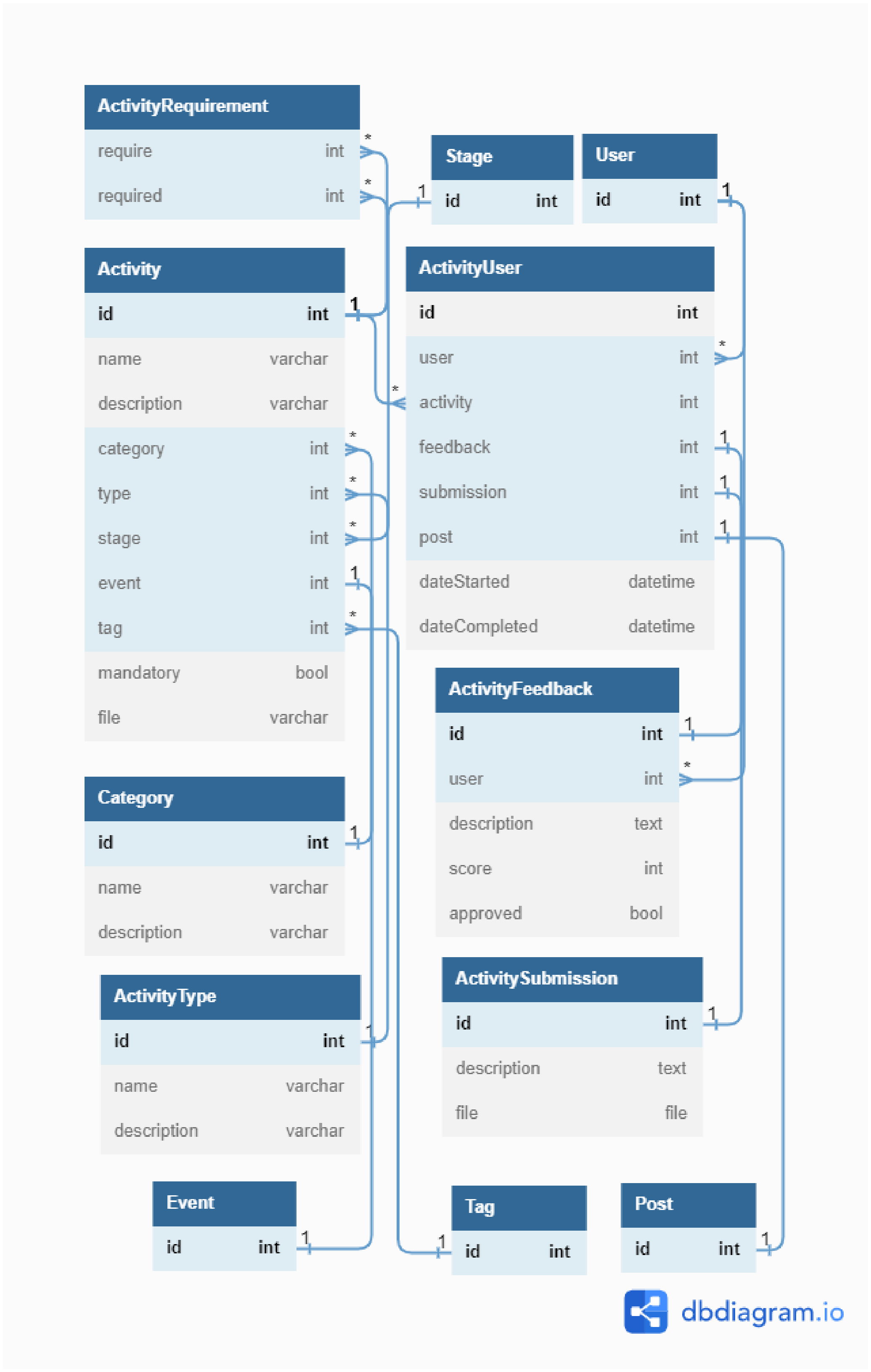


Figure 2.5: Diagram of the Activity Model and its associations.

2.3 Grouping the Activities into Stages

To allow administrators to create a better flux of the activities in the platform, SkillSpace allows the activities to be inserted onto stages created by the admins. The idea of the stages is to group together activities that either are on the same difficulty levels (creating *Beginner* and *Advanced* stages for instance) or that are under the same topics (such as a *Mathematics* and *Chemistry* stage). Such as with the activities, there is also a join table between the `Stage` and `User` models, to indicate its start and completion just like it was done in the activities. Also similar is the presence of a `StageRequirement` model, that essentially locks a stage from being started if its requirements are still not met by the user.

To finalize a stage the user must meet two conditions according to the configuration of the stage. Administrators can determinate a certain time requirement that users must meet from the day they started the stage to finalize it. This configuration, although optional, may be used for cases where the planners of the platform may want to avoid students progressing on the stages too fast, creating a big discrepancy between users that started studying at the same time. The second condition to finalize a stage is having completed all the mandatory activities inside that stage, allowing administrators to determinate which ones are essential to be done as the user progress on their studies and which ones are seen as extra activities to be performed. The information of which activities are mandatory is stored in a Boolean field on the activities themselves.

The diagram from Fig. 2.6 represents the organization of the `Stage` model and its associations on the database, omitting the fields from the `User` for organization purposes.

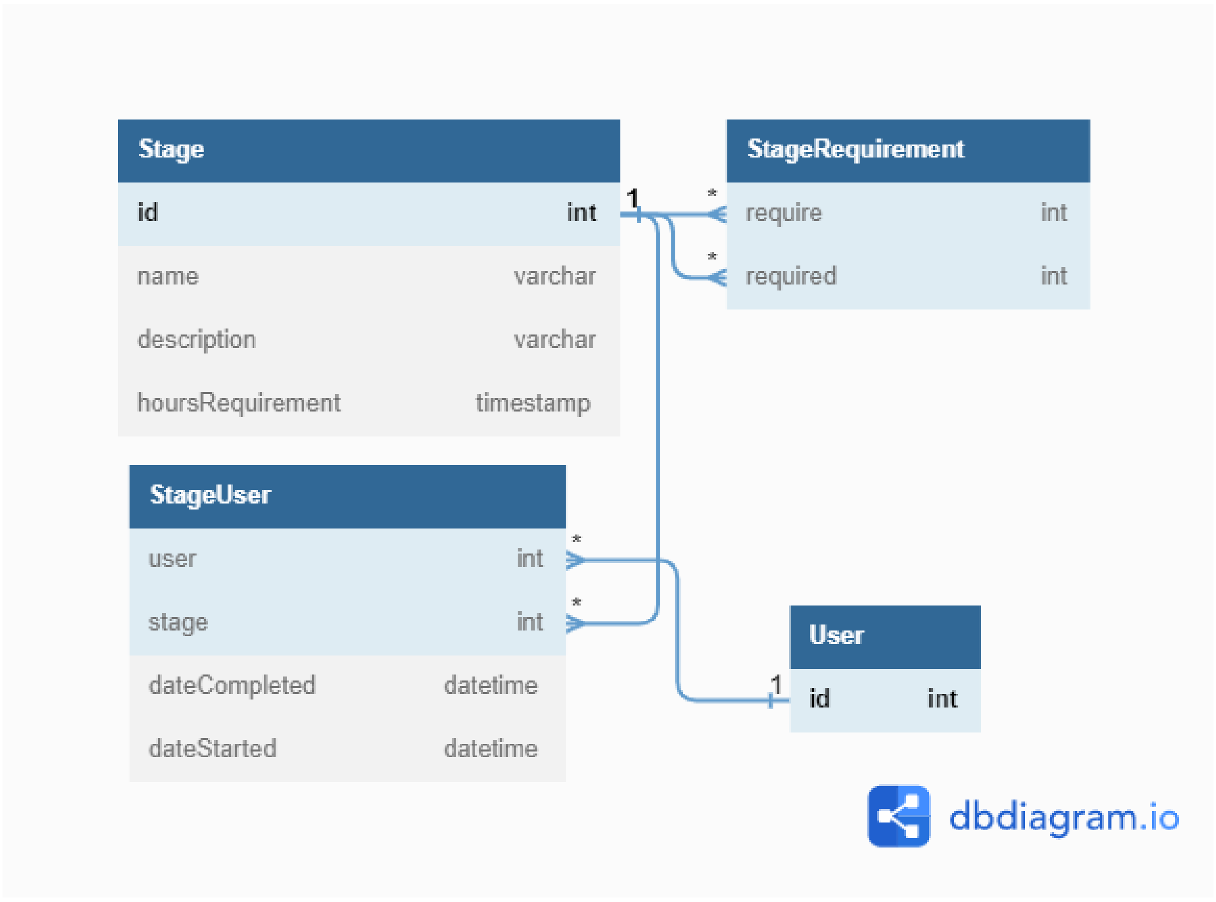


Figure 2.6: Diagram of the Stage Model and its associations.

2.4 Events

To allow the platform community to come together in events, such as speeches and lectures, the SkillSpace platform comes with functionality to its administrators manage and sent invite to these events and to users to view these invites and information about the events and provide feedback for the ones they attended. In the back-end, the center of this functionality is the `Event` model, which contains the basic information about the event, such as a description and its date, also storing information about whether it is a remote event in a Boolean field and the link to the event (for the case it is remote). The administrators may also want to provide information about invited speakers that will participate in the event, which is stored in a separate table in the database containing information about these participants to be shown to the users on the website. The `InvitedSpeakers` and the `Event` tables are connected through a join table and the invited participants are not required to be registered on the platform.

To manage the invitations, a join table between `User` and `Event`, called `Guest`, is created, and the entries of this table are created when an invitation is sent by the administrators or the user declares their intention of participation in the event after visualizing it on the platform. This join table also stores whether the guest is the organizer of the event, their intent of participation, through enum values, and their presence, which is updated after the event happens. A second join table between `Event` and `User` is the `EventFeedback`, which stores the feedback for a specific event that the user attended. The user may leave a text with their feedback and a numbered score. They may also choose to not let their name appear to the administrators while reviewing the feedbacks, through a Boolean anonymous field.

Fig. 2.7 shows the `Event` and `InvitedSpeaker` models, alongside its join tables. The `User` model represented in the figure, as usual, have its fields omitted for organization purposes.

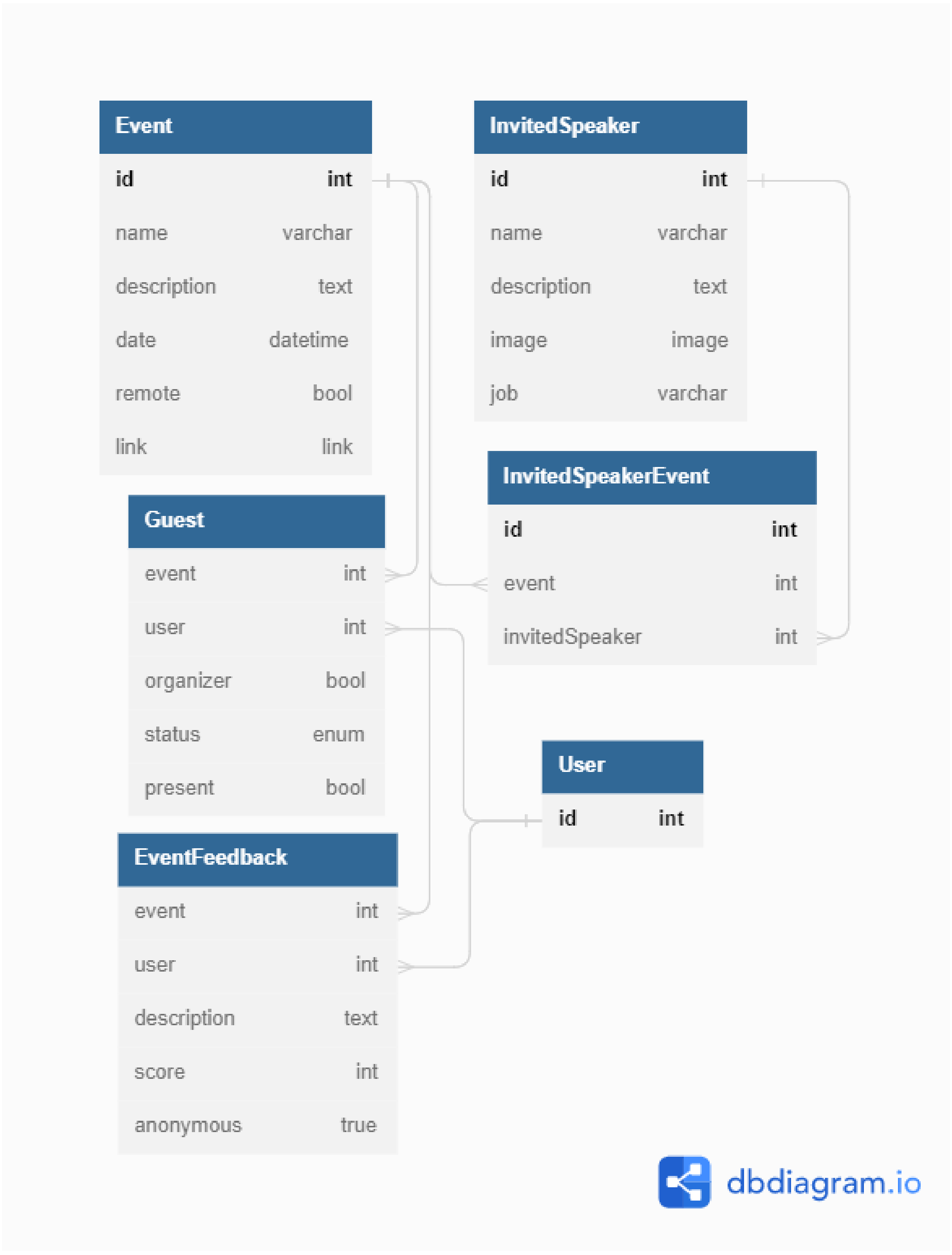


Figure 2.7: Diagram of the Event Model and its associations.

2.5 The social area of the platform

The final main area of the platform is the social area, with its main part being the `Post` model, that represents posts made by the users to be displayed in the front-end just like a social media platform, storing the details about the post alongside a reference to the user that created it. To better organize the topics of conversation that the post falls to, users can select tags to be inserted in the post. These tags are stored on a separate table and are created beforehand by the administrators, representing topics of relevance to the specific Space it belongs to. The `Tag` and `Post` models are connected through a join table to allow a single post to be a part of multiple tags, while another join table is also created, connecting the **`Tag`** and the `User`, allowing users to select which topics they are more interested in, allowing for future customized posts recommendations on their feed.

Its also fundamental to allow interactions among all users of the platform to the posts published on the website. To achieve that, the `Post` table also features a parent field, which references itself, representing which post the new one is responding to. A null value on this field indicates that the post is a parent post while a valid reference represents that it is a comment on another post. Another way of interacting with the posts is by reacting to it, through a join table between the `Post` and the `User`, containing an `enum` value with types of reaction, ranging from *like* and *love* to *sad* and *angry*. In Fig. 2.8 we are able to see an overview of the `Post` model and its associations as described above.

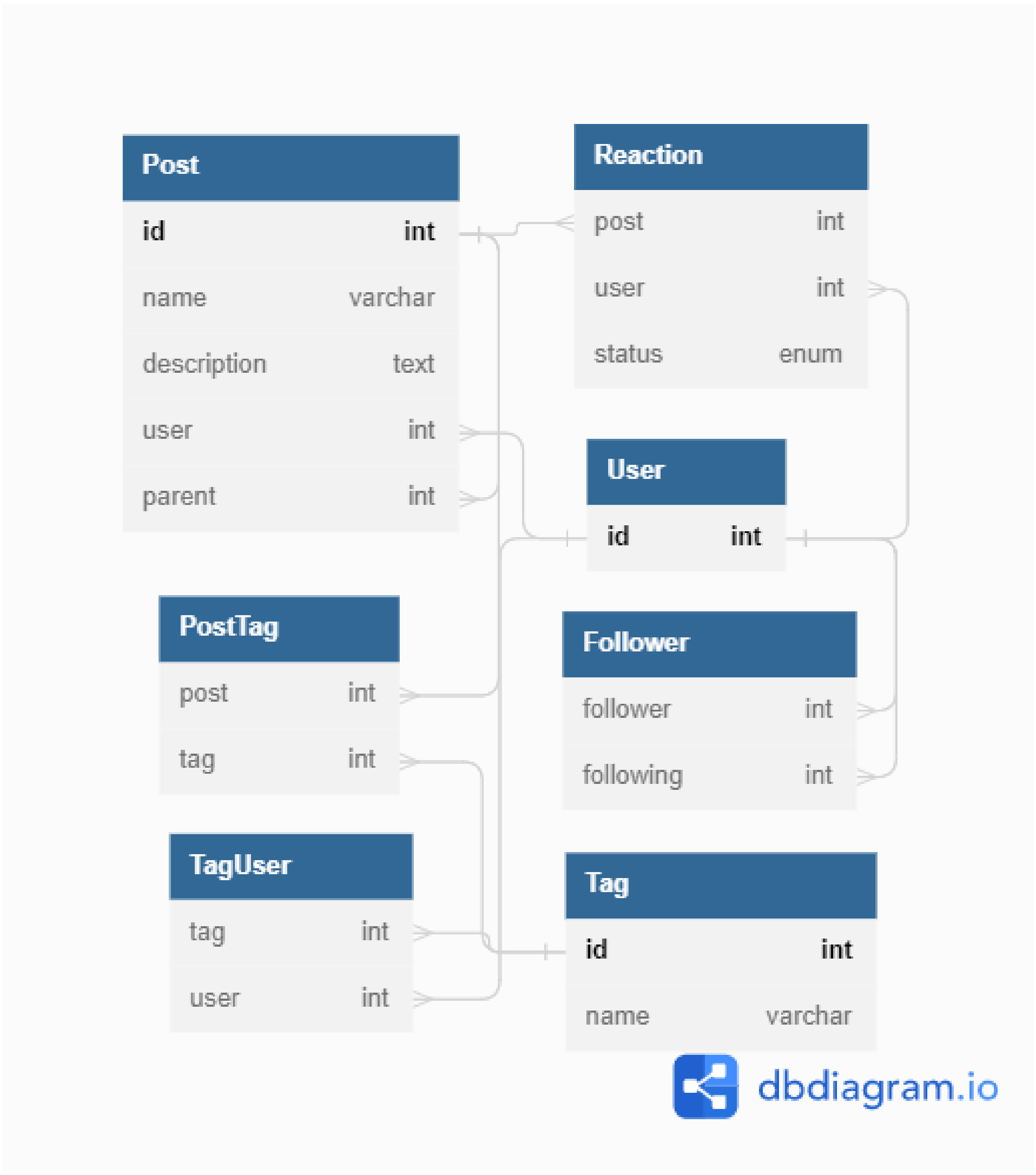


Figure 2.8: Diagram of the Follower and Post models and its associations.

The social area of the website also features a more direct connection between the users, with a `Follower` table allowing a user to follow another one, which allows us to prioritize posts published by people the user is following on their feed. We can also see this model in Fig. 2.8 above. This direct communication between users is more evident in the messaging function on the platform, that allows multiple users to communicate with each other via the SkillSpace website itself, without requiring them to exchange personal contact information with each other if they don't wish to. This messaging feature is composed by three main tables: the first one is the `Chat` table, that only stores an identification number for the instance, automatically generated by the back-end. The second table is a join between the `User` and `Chat` and is used to represent all the participants of the chat that are able to view and send messages to it. Finally, the `Message` table references the `Chat` it is from, the `User` that sent it and all other information required by the message, such as the message itself and a timestamp from when it was sent. All the models from the messaging feature can be seen in Fig. 2.9 below.

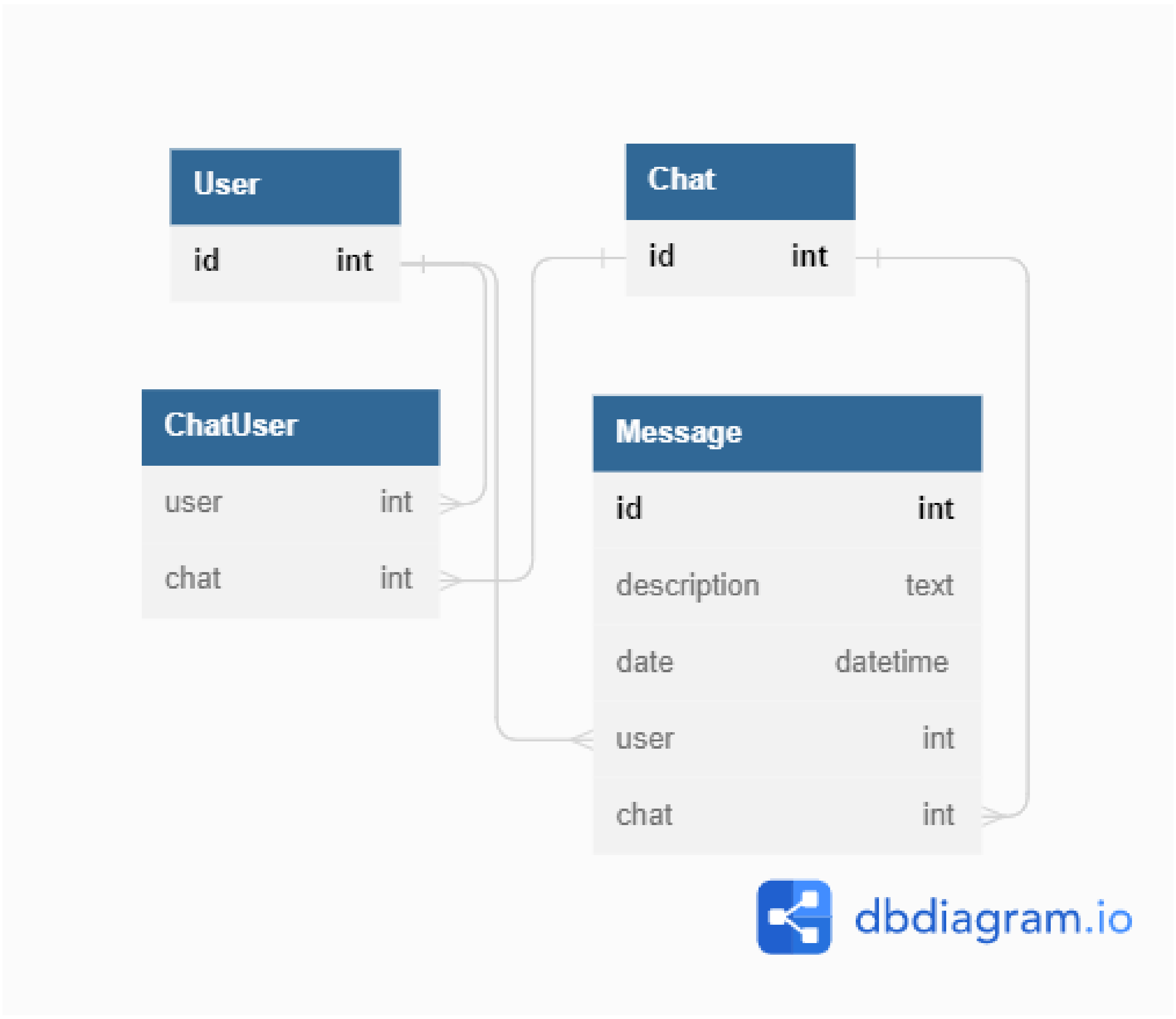


Figure 2.9: Diagram of the models from the messaging feature.

With a complete design of the database that represents the SkillSpace platform, alongside all tables and its relationships, the next step in the development is to proper implement these definitions into code and create the back-end logic that will properly use the defined database to provide the SkillSpace functionalities.

Chapter 3

Back-End Development

To provide a functional website, the back-end was implemented as an API developed using the `Node.js` [5] environment linked to a `MySQL` [6] database to store and organize the necessary information for the application. The decision of making the back-end separated from the front-end, by making it as an API, was to make it as an independent part of the system, allowing the development of multiple different front-ends in the future if needed, such as mobile applications and different websites that uses the same business logic.

3.1 API Technologies

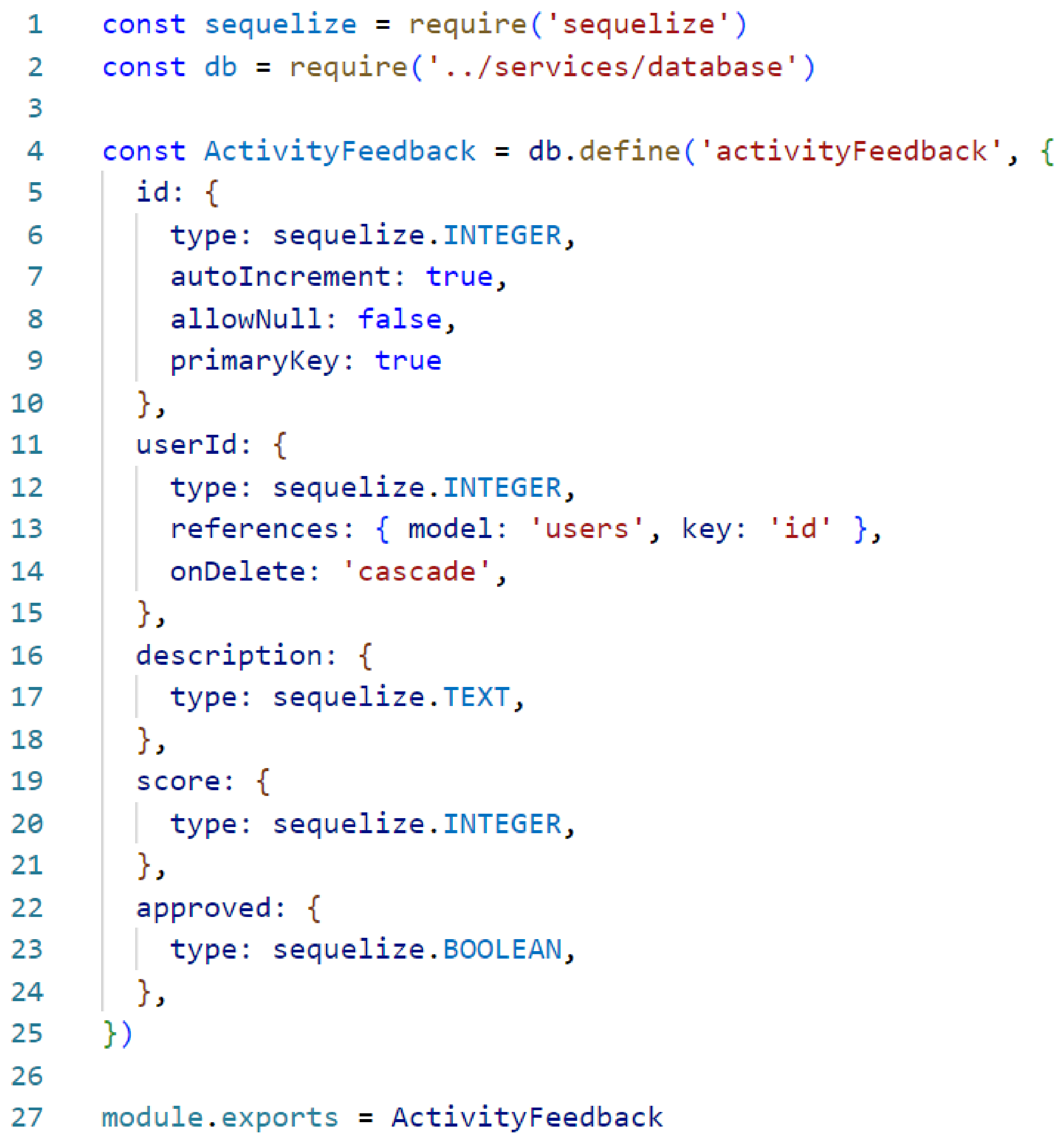
`Node.js` is a runtime environment that allows the execution of JavaScript code outside the web browser. As such, it allows for the development of web applications, such as web servers and APIs, using an already popular language in web development. For the SkillSpace API development, several options of languages and frameworks were available to choose from, like Ruby, Python and PHP. The choice of using `Node.js` (and JavaScript for that matter) came from the high availability of tutorials, guides, and overall references for it on the internet, the high number of open-source packages and tools available using the technology package manager (`NPM` [7]) and the possibility to use the same programming language across the API and front-end development. The use of the `Express` [8] framework was also chosen to further help on the development process. In our API development, the `Express` framework particularly helps while managing the different HTTP requests received and its routes, also providing the ability to configure middleware's to be executed in those requests. Being the most popular web framework for `Node.js`, the community support is unparalleled among other alternatives available for `Node.js` development.

Working alongside `Express` as middlewares, two other node packages were used to further configure the developed API, `CORS` [9] and `body-parser` [10]. The `CORS` package adds Cross-Origin Resource Sharing [11] to the project, which allows the API to inform web browsers outside domains which the server may request resources to, by setting up the appropriate headers on all requests and responses made by the API. For development purposes, CORS on this project is currently configured to allow access to all outside domains, a configuration that should be changed when deploying the site into production. Finally, the body-parser package is used to parse the body content of all incoming requests to JSON format, making it available in all upcoming steps of the request-response cycle.

As the connection with the database is another essential part for the API, Sequelize [12], another popular Node.js package, was chosen to further help development. This package provides an abstraction layer in the connection between the API and the database, allowing the use of the database with JavaScript code instead of SQL queries. As Sequelize works with multiple relational database management system, the decision of which one to use didn't make much difference, with MySQL chosen for the previous familiarity with the tool.

3.2 Defining Models and Migrations

Using the MVC [13] architecture as reference, Sequelize was used to write the application models, composed by the tables stored in the database along all its fields and relationship references. After having the models defined, Sequelize provides a JavaScript object that represents it and is accessible thorough the rest of the project code. This model object features the methods that are used to replace the SQL queries, such as methods to create a new instance on the database and to find stored instances. An example of defining a model using Sequelize can be seen in Fig. 3.1. In this example, it is possible to see that each of the defined fields has a determined type to it, which is chosen among the options available for the MySQL database. Other constraints and options are possible for each field, such as requiring the field to be filled, by not allowing nulls, and not allowing repeated field values across the instances of the table, using the unique option. The id field is also set as the primary key of the table and, with the autoIncrement option set to true, the database takes the responsibility of setting the id on each instance using a value that increases on every instance creation of the table, maintaining the constraint of uniqueness required by a private key. The references fields are also indicated when defining a model, stating which model and field is being referenced. By using the onDelete and onUpdate options, we can define what will happen with the table instance when a referenced table is deleted or modified, respectively, being able to choose to cascade the action to the table referencing it, restricting the action to be done, setting the field to null or doing nothing.

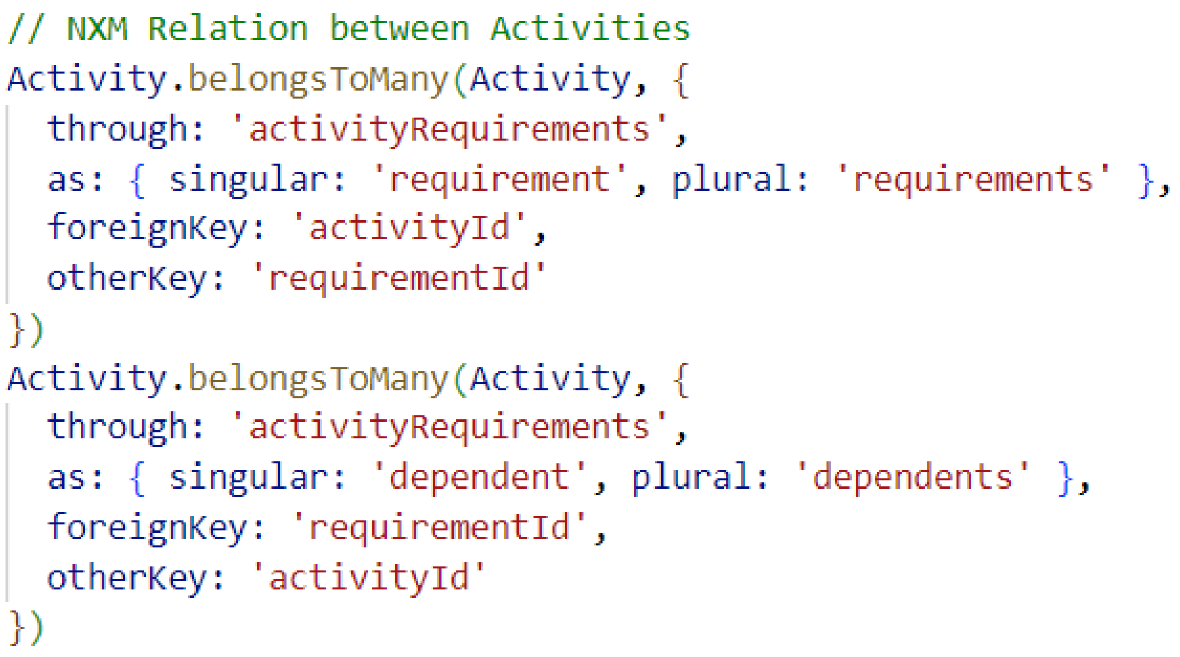


```javascript
const sequelize = require('sequelize')
const db = require('../services/database')

const ActivityFeedback = db.define('activityFeedback', {
  id: {
    type: sequelize.INTEGER,
    autoIncrement: true,
    allowNull: false,
    primaryKey: true
  },
  userId: {
    type: sequelize.INTEGER,
    references: { model: 'users', key: 'id' },
    onDelete: 'cascade',
  },
  description: {
    type: sequelize.TEXT,
  },
  score: {
    type: sequelize.INTEGER,
  },
  approved: {
    type: sequelize.BOOLEAN,
  },
})

module.exports = ActivityFeedback
```

Figure 3.1: Example of Model definition using the Sequelize package.

While `Sequelize` allows syncing with the database using only the models definition, the decision of using migrations, another feature available in the package, was made to track the changes made to the database as the development progressed. Each migration represents the changes made to the database, being it a new table or changes to a table's field, and features up and down rules that dictates how to perform the migration and how to undo it, respectively. The use of migrations helps while working in the model development as a team and avoids the database to be always reconstructed when multiples changes are made to it.

To properly interact with the relationships on the database, `Sequelize` also requires each relationship to be explicitly defined outside the model, informing the type of relationship that happens between the two models, as can be seen in Fig. 3.2, where a many-to-many relationship is defined using a join table. On this project, it was decided to create a middleware to store all the associations and to make it available to all requests made to the API, centralizing the relationships in one single file of the code. In the association definition, besides the `onDelete` and `onUpdate` options described before, which are also accessible here, we are able to define the join table to be used on many-to-many relationships, using the `through` option, and also define aliases for the association, which is specially useful when the table references the same table in more than one field.



```
// NXM Relation between Activities
Activity.belongsToMany(Activity, {
  through: 'activityRequirements',
  as: { singular: 'requirement', plural: 'requirements' },
  foreignKey: 'activityId',
  otherKey: 'requirementId'
})
Activity.belongsToMany(Activity, {
  through: 'activityRequirements',
  as: { singular: 'dependent', plural: 'dependents' },
  foreignKey: 'requirementId',
  otherKey: 'activityId'
})
```

Figure 3.2: Example of association definition using the Sequelize package.

All this knowledge on defining the models, migrations and associations was used to properly configure the database described on the previous section of this document, and with all configured, we can then use these models to properly create the methods that will access it to provide the SkillSpace functionalities in the controller.

3.3 Creating the Controller

In a MVC architecture, the `Controller` provides the interface between the `Model` (database) and the `View` (front-end), also defining the rules of this access, implementing the business logic. In the SkillSpace project, several controller files are created each representing a model or functionality of the API, containing methods specialised in executing a specific function of the platform.

3.3.1 Controller Methods

The CRUD acronym stands for `Create, Read, Update and Delete` and defines the set of functions that provides this functionalities to a specific model in the system, which, for the SkillSpace API is provided by the functions `create`, responsible of creating an instance of the model in the database, `show`, which returns all the information from one specific instance, `index`, which retrieves information from all instances of a model in the database, `update`, that updates and instance, and `delete`, to remove an instance from the database. Among the models of the platform, `Tag`, `Stage`, `Post`, `InvitedSpeaker`, `Event`, `Category`, `Address`, `ActivityType` and `Activity` features full CRUD implementation with all the five methods described above. The `Permission` model, on the other hand, features only the index method, since the creation of its instances should be done directly on the database to tie to the business logic. Most of the remaining models are from join tables, and are indirectly accessed and created as needed inside methods from others models controllers, and some may not feature all CRUD functionalities when not needed.

Besides the `index`, `show`, `create`, `update` and `delete` methods, the controllers also features other functions that provides more specific functionalities needed by the system. These methods are described in the Table A.1.

3.3.2 Managing the User Access and Privileges

The `User` controller, instead of the usual `create` method, features a `signup` one, that besides creating an `User` instance, also create instances of `Address` and `TagUser`. The `signup` method also handles the encryption of the password before storing in the database, using a hash function, that receives the user password and generates a predictable hash that can only be retrieved by using the original password. To further increase security, a salt (a randomly generated string) is used alongside the hash function to produce different hashes even when the original password is the same. This salted hash function is not implemented by hand, instead, the `bcrypt.js` [14] package is used to achieve the desired result. This package, in its hash function, also allows for a `salt round` parameter, which controls how much time is needed to calculate a single hash, with higher values increasing the time needed to calculate a single hash, while making it more difficult for brute-force attacks to succeed. The standard value for the `salt rounds` in the `bcrypt.js` package is `10`, and, since this value has not been updated by the package in at least 6 years, other values were tested in order to select the appropriate one. In Fig. 3.3 the evolution of the time each request takes in relation to each `salt round` value is shown. To plot this graph, five requests under the same processing conditions were run for each `salt round` value, using the mean value of the five values in the plot and computing the standard deviation across the values, also shown in the graph. The value `12` was then chosen to be used on the project as the `salt round`, being the higher value such that the request time was still in an acceptable range to not disturb the platform users, being lower than 500 ms.

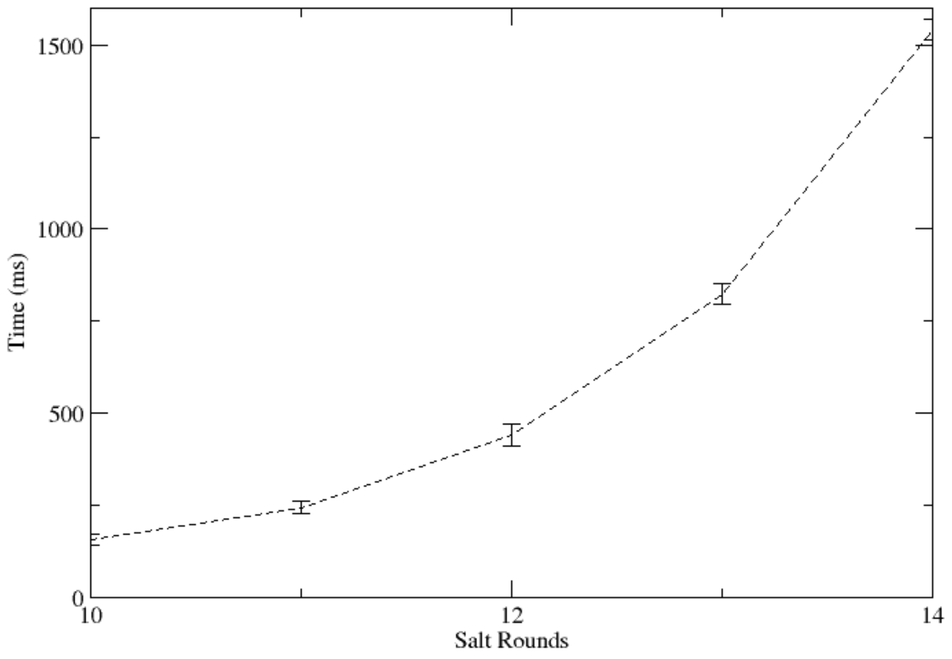


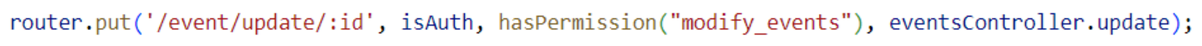
Figure 3.3: Graph comparing the `salt round` value to the time each request takes.

To allow the authenticated user to make requests to the API from the front-end side, without sending the login information on each request, the `login` method uses the `jsonwebtoken` [15] package to generate a `JSON Web Token object` [16] containing the user information, which can be sent on the header of the API requests as a mean to authenticate the user making those requests. These tokens are encrypted using a key only available on the API, securing malicious parties of reading the token information or generating false tokens, and also have an expiration time of 3 hours, avoiding old tokens to be retrieved and used by hackers.

The controller also implements a restriction functionality to each method based on the role of the user on the website and the privileges that the administrator holds, which is done in the router level using middlewares. The first middleware available just verifies if the user is properly authenticated by reading the token on the request header, without regards to the user role or status, also adding the user information retrieved on the token to the request information, to be used by future middlewares and methods. Two other middlewares, `is-admin` and `is-owner`, analyses the retrieved user from the previous middleware to see if it's a admin and a owner on the platform, respectively. The final authorization middleware receives the name of a permission and analyses if the user making the request is either an owner or have the proper permission associated with him, and, if not, restricts the user access to the desired method.

3.3.3 Mapping the Controller into Routes

In order to make the methods of the controller available for the users of the API, we need to map each method into a route, which is the URL endpoint from which the method is accessed. The routes can be of 5 different types, `GET`, when the endpoint only retrieves a information with no modifications on the database, `POST`, when resources are created, `DELETE`, for methods that delete resources, `PUT`, when a resource is completely modified by the request and `PATCH`, for when only a set of changes are applied to a resource. An example of a route definition using `Express` can be seen in Fig. 3.4, where the `router` is an object generated by `Express` with methods for each one of the route types described before. Theses methods takes as the first parameter the URL endpoint from which the route will be accessed and all the middlewares that should be used by the request (such as the authorization middlewares from the previous subsection), with the final parameter being the method that the route redirects into.



```
router.put('/event/update/:id', isAuth, hasPermission("modify_events"), eventsController.update);
```

Figure 3.4: Example of Route definition using Express.

3.4 Handling File Uploads

In several areas of the Skill Space application, we require the upload of files by the user to correctly serve the proposed functionality, such as allowing file submissions upon finishing practical activities with the work the student has done. But file uploading and saving is drastically different from storing text data in a database, with database management systems, like MySQL used in this project, not having support for this functionality. To add this functionality in our project, we decided to use the node package *Multer* [17], which does two things to support file upload: it first allows us to receive and decode multipart forms from the POST requests made to the API. What this means is that instead of the usual encoding used on requests, `application/x-www-form-urlencoded`, which simply concatenates the form information using the '&' operator (ex: `name=Maria&age=21`), the `multipart/form-data` groups each key-value pair into its own section, each with its own header, content type and content disposition, allowing the submission of more complex information, such as files, with the trade-off of having more overload of data per information sent. The second way the *Multer* package helps is by providing helper methods that handles saving the files to the desired destination on the computer. Finally, to link the uploaded files to its respective databases entries, a string field is added to the required tables storing the path and filename used in the upload. There is only one problem with the configuration provided above, the files can only be stored locally on the machine running the API. While this behavior is acceptable in a locally development environment, its incredible inefficient on a production one, since the machines used to host an API service are not correctly optimized to deliver big file data to users, also causing more intense traffic on the application, which can hinder its use by the users. The solution is to store the files in an appropriate cloud tool optimized for this use, such as Amazon S3 service, a part of their Amazon Web Services package. This functionality is added to the application with the use of two more node packages, the Amazon provided *aws-sdk* [18], which allows the direct communication between our application and the Amazon services, and *Multer-S3* [19], which builds upon *Multer* functionalities to allow the S3 service to be used as storage option. With the proper configuration of this package, we're able to store all files related to the project in a bucket on the S3 servers.

With both the `Model` and `Controller` configured, alongside the definition of the API endpoints through the routes, we can start projecting the final part of the MVC architecture, the `View`, which is the front-end of our application. We begin by presenting a prototype of it in the chapter that follows.

Chapter 4

Non-Functional Prototype

With the advancements in web development, the development of User Interfaces got harder, with the need to dedicate a bigger amount of time to code a web page from scratch and style it accordingly. This task is made harder by constant testing and changes made to a UI interface during development, to make sure the pages feel not only intuitive but beautiful to look at, and, due to the non-intuitive nature of web development, making all these tests directly on code is highly inefficient. For instance, even making small visual changes, such as moving a button to the other side of the screen, can prove to be a not so trivial task and, when doing it only for testing purposes, it can represent a wasted time on your development schedule. To avoid doing the entire front-end development from scratch, it is necesary to create a prototype of the pages, with a clear definition of the design and the elements placement on the page, allowing for an easier testing and validation of the pages structure. For this purpose, the online tool `Figma` [20] was chosen to prototype the website, with all the pages needed being first created on the tool, and then translated into code, in an already validated design. `Figma` allows to easily create design prototypes, using basic shapes (like squares and circles), importing icons to be used on the website and structuring the overall look and feel of the pages in an easy and intuitive way, which also allows elements to be quickly changed for testing purposes. `Figma` also allows to create components, that once created and styled, can be reused across different prototypes. Moreover, after designing your prototype, the tool allows for a quick glance on the `CSS` code that can be used on the web page implementation, even though this code is usually non responsible for different screen sizes than the one it was designed for.

4.1 Login and Sign Up Pages

As means of accessing the system and registering into the platform, the `Login` and `Sign Up` pages are the first the user will be given access to. The login page, seen in Fig. 4.1, features a simple form with options to the user input its login credentials, as its expected for this type of page. The sign up page, is divided into two parts, the first one, seen in Fig. 4.2, features the form with all the inputs needed to register into the system, the second part, seen in Fig. 4.3, however, features options for the user to select all topics (`Tags`) that are in the user interest, to allow the platform to recommend `Posts` according to the user preference right out of the register step.

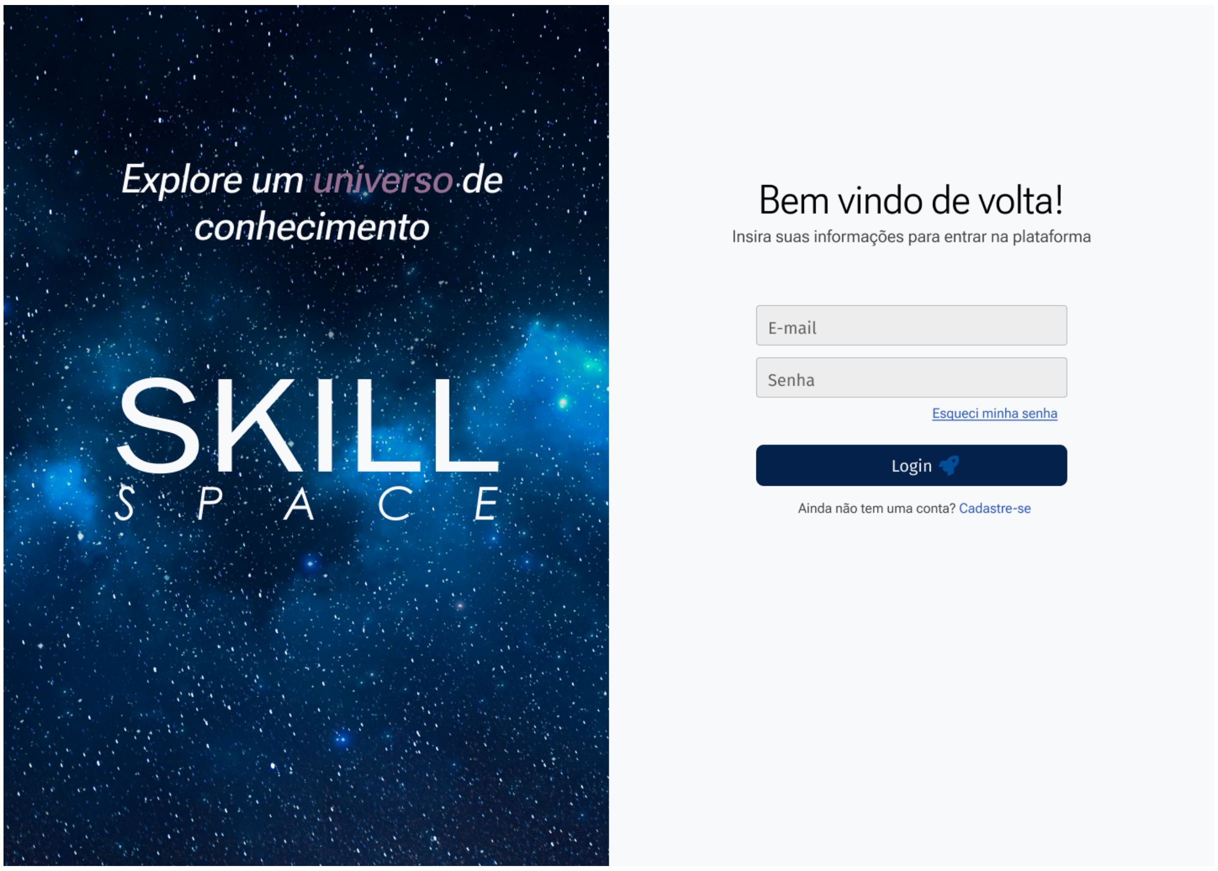

Explore um universo de conhecimento
SKILL
SPACE
Bem vindo de volta!
Insira suas informações para entrar na plataforma
E-mail
Senha
Esqueci minha senha
Login
Ainda não tem uma conta? Cadastre-se

Figure 4.1: Prototype of the `Login` page.

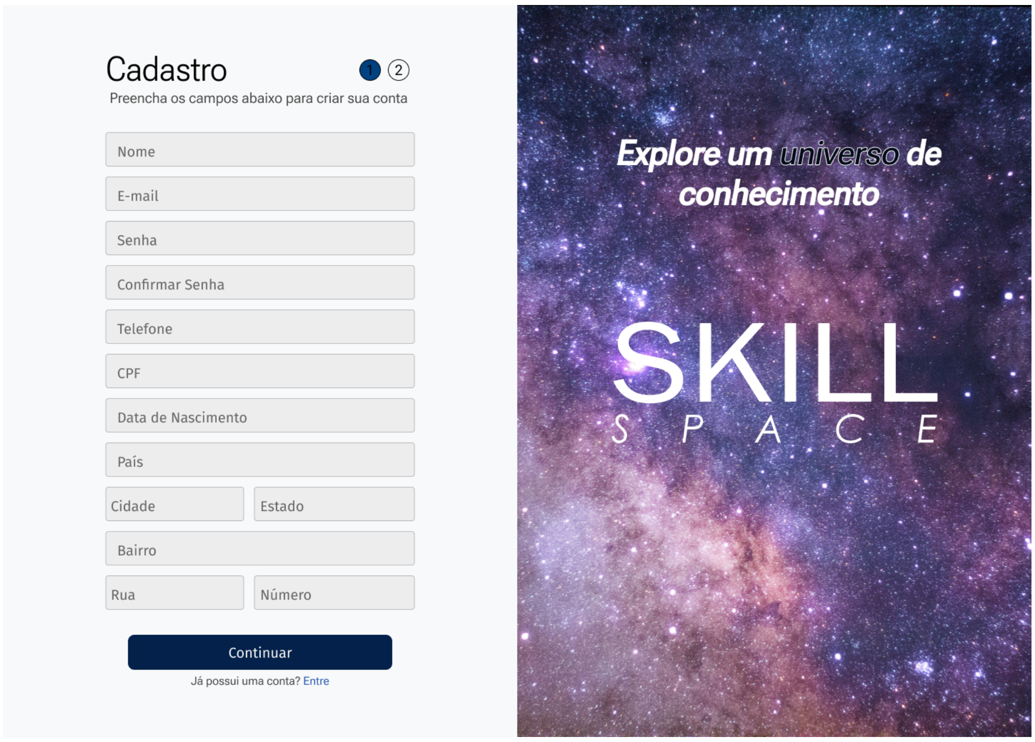


Figure 4.2: Prototype of the first step in the `Register` page.

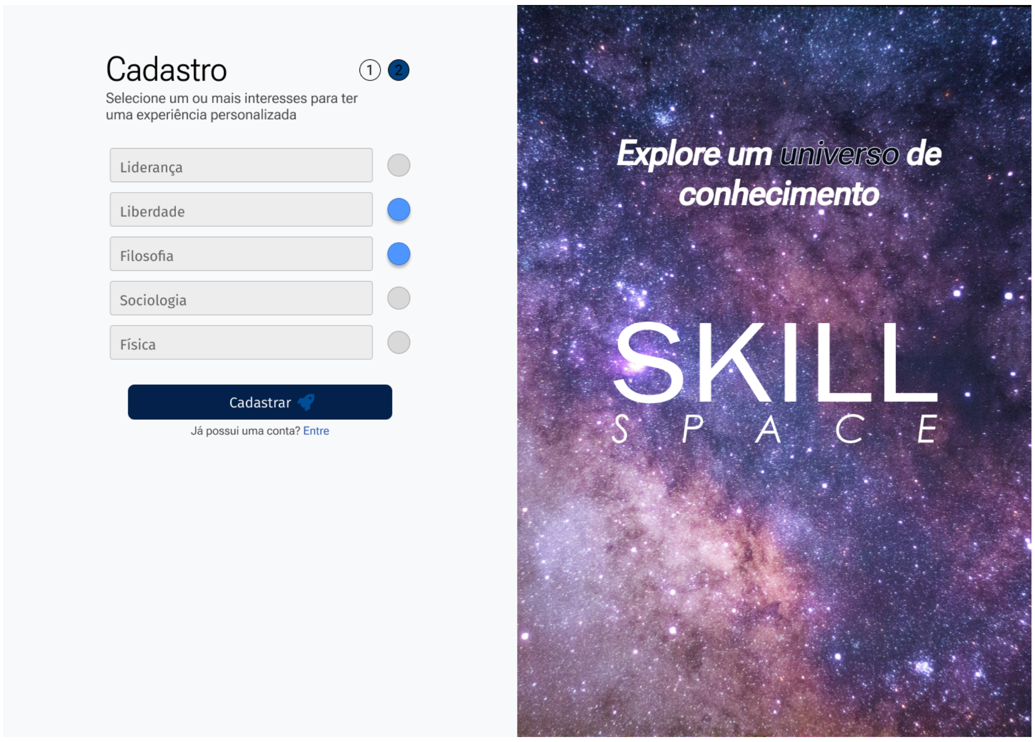

Cadastro
1 2
Selecione um ou mais interesses para ter uma experiência personalizada
Liderança
Liberdade
Filosofia
Sociologia
Física
Cadastrar
Já possui uma conta? Entre
Explore um universo de conhecimento
SKILL
SPACE

Figure 4.3: Prototype of the second step in the `Register` page.

4.2 The Navigation Sidebar

As a means of easily navigating through the site multiple functionalities, upon login the user will see on all the platform pages a sidebar on the left side of the screen containing shortcuts to all different sections of the platforms into six main icons, Activities, Events, Posts, Messages, Help and Configurations, with each icon lighting up representing where in the site the user is in the moment. In the bottom of the sidebar, the users name and profile picture will be seen, serving as a link to the user own profile. To improve user experience, the sidebar may be collapsed, taking less space on the screen, but with all the shortcuts still easily accessed. The prototype design for the sidebar can be seen in Fig. 4.4.

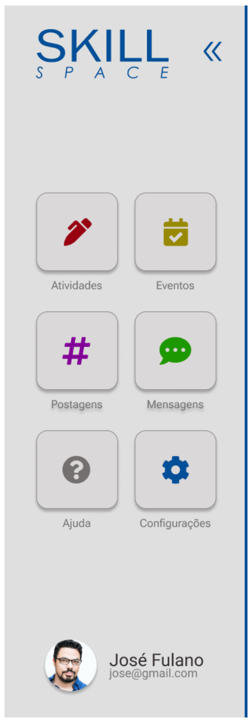


Figure 4.4: Prototype of the navigation sidebar.

4.3 Stages and Activities Area

For the activities section of the website, the user will first be presented with a list of all the stages on the platform, as can be seen in Fig. 4.5. The stages are displayed in cards indicating its name and information regarding the number of activities on the stage and how many were already completed by the user. The cards also indicates if the activity is blocked due to requirements not being met, by showing a lock icon in the top right corner of the card. They can also be filtered by searching for a stage name or choosing to show only completed, unlocked, or not unlocked stages. This page also features general statics about the completion of stages by the user, such as the percentage of progress throughout the whole course.

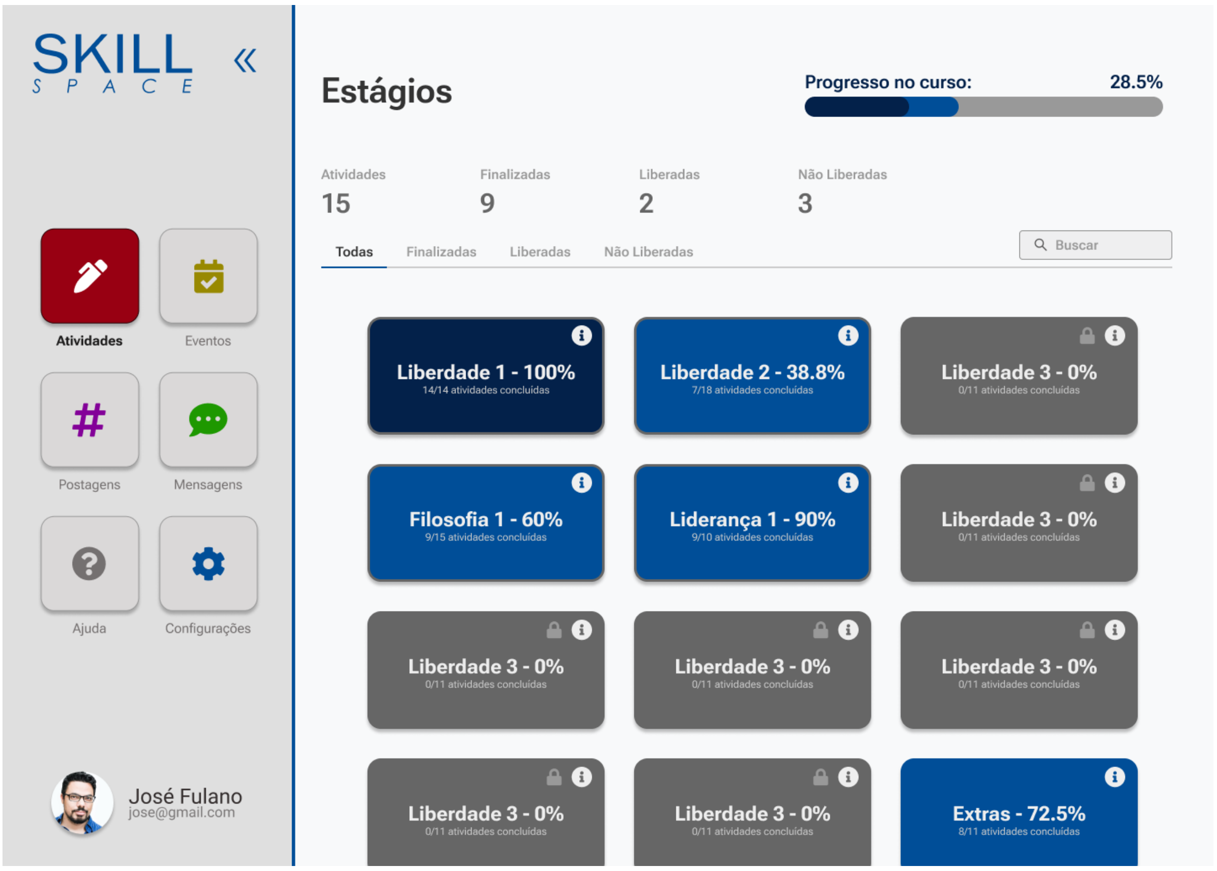
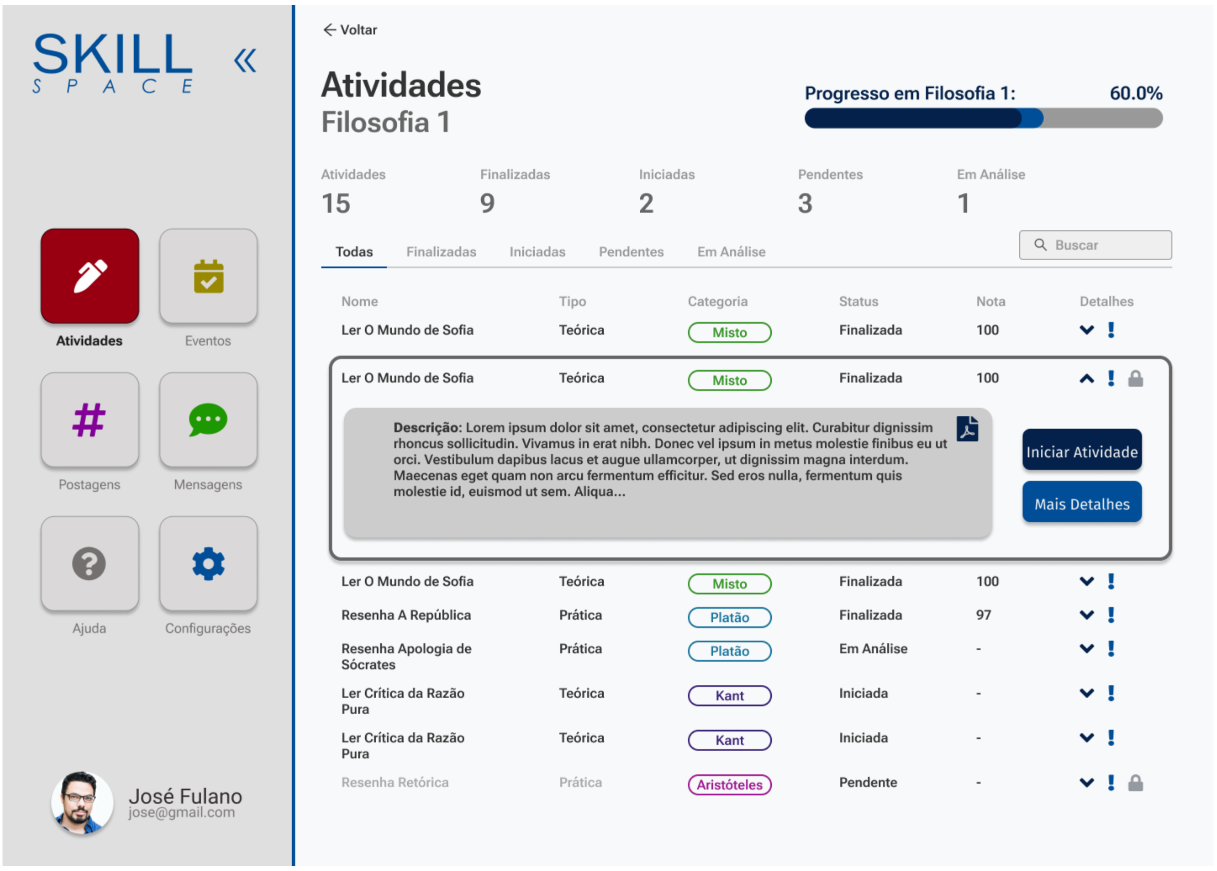


Figure 4.5: Prototype of the `Stages` page.

When choosing a stage on the page, the user will be taken to the table in Fig. 4.6, with all the activities on the stage, once again containing general information about the progress of the stage alongside filter and search options. The table itself shows at a quick glance the basic information about the activity, such as its name, type, category, status, and grade (if applicable). Icons may also appear on the far-right column of the table indicating whether the activity is locked due to missing requirements and if the activity is mandatory to completing the stage. Upon clicking on any activity on the table, it is expanded to show its description and links to access the activity itself, which leads us to the last page on the activities section, the activity details.



Atividades
Filosofia 1

Progresso em Filosofia 1: 60.0%

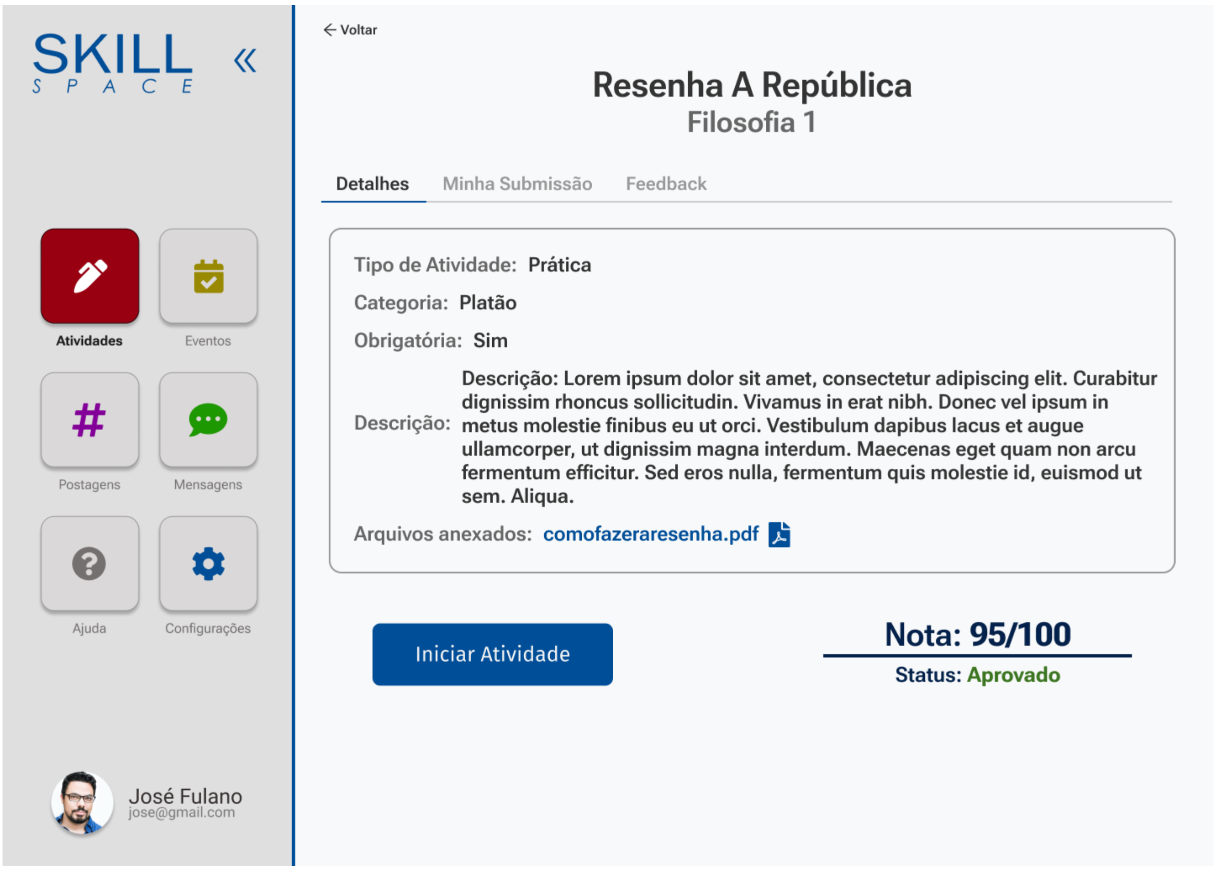
Atividades	Finalizadas	Iniciadas	Pendentes	Em Análise
15	9	2	3	1

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 4.6: Prototype of `Activities` index page.

In Fig. 4.7 the activity details page is shown, containing all the information regarding the activity, including files that may be attached to it, alongside a button to properly start or finish the activity. If the activity requires a submission, we can also see tabs on the top area to visualize the provided submission and the feedback provided to it. Also in this case, we will be able to see the score of the submission throughout all the tabs.



Resenha A República
Filosofia 1

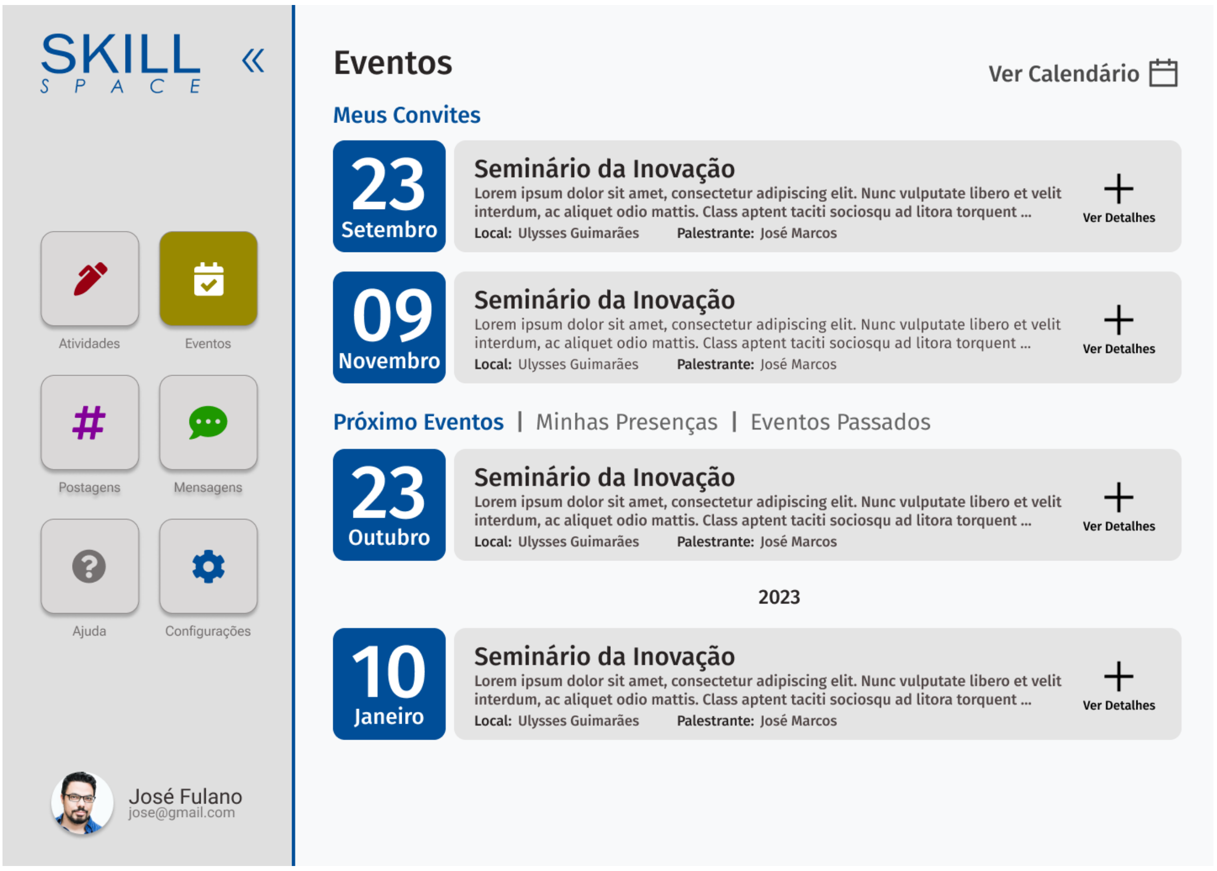
Tipo de Atividade: Prática
Categoria: Platão
Obrigatória: Sim
Descrição: Descrição: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur dignissim rhoncus sollicitudin. Vivamus in erat nibh. Donec vel ipsum in metus molestie finibus eu ut orci. Vestibulum dapibus lacus et augue ullamcorper, 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

Figure 4.7: Prototype of `Activity` details page.

4.4 Events Area

The events area is composed of one single page featuring multiple tabs and modals. Upon first entering the page, the user is greeted with the list in Fig. 4.8, containing the upcoming events on the platform, each showing its most important information upfront, such as the date, location, and description. The page places a particularly bigger focus on events which the user have been explicitly invited to, with these appearing on top of the page. After that, the page shows all upcoming events in ascending order by its date. The user may also use the tabs on the page to visualize events he has been to and to see all past event on the platform.



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
Ver Detalhes

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
Ver Detalhes

Próximo Eventos | Minhas Presenças | Eventos Passados

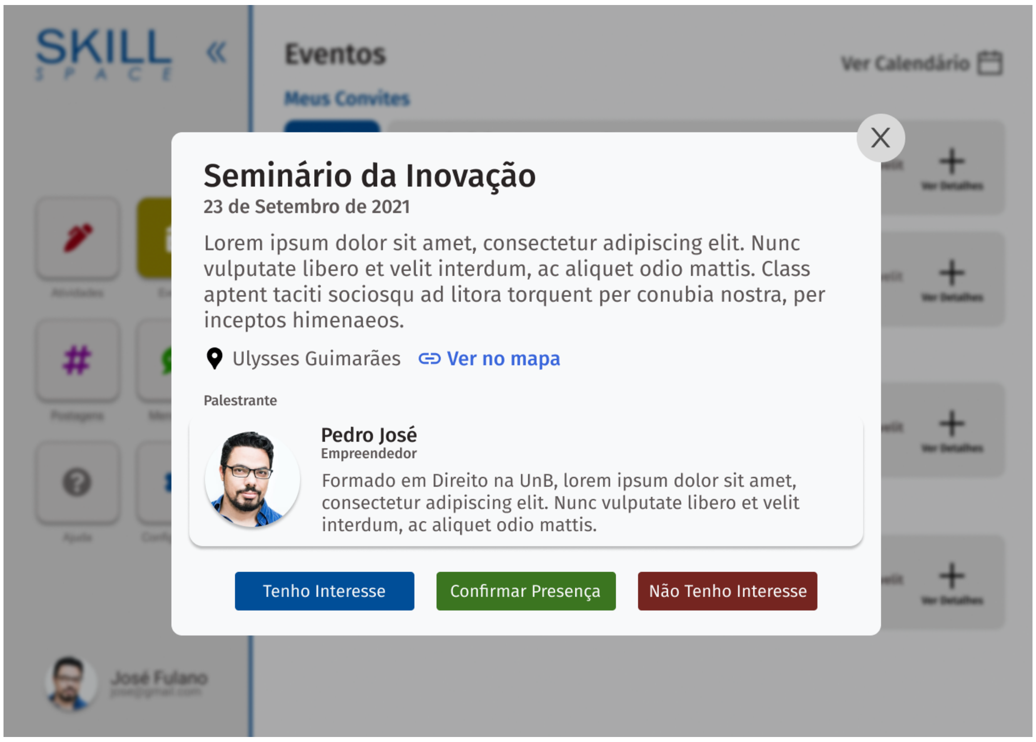
23 Outubro — Seminário da Inovação
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc vulputate libero et velit interdum, ac aliquet odio mattis. Class aptent taciti sociosqu ad litora torquent ...
Local: Ulysses Guimarães Palestrante: José Marcos
Ver Detalhes

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
Ver Detalhes

Figure 4.8: Prototype of `Events` index page.

Upon clicking to see more details of the event, the modal in Fig. 4.9 appears on the page showing all its details, and the details of the invited speakers to it, if any. This modal also features options to the users state their intent of participating in the event if it is an upcoming one. If the event was one that the user participated, an option to give feedback to the event will appear, which opens the modal in Fig. 4.10, that allows the user to write a feedback, grade it and choose if the feedback should be anonymous or not.


Seminário da Inovação

23 de Setembro de 2021

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.

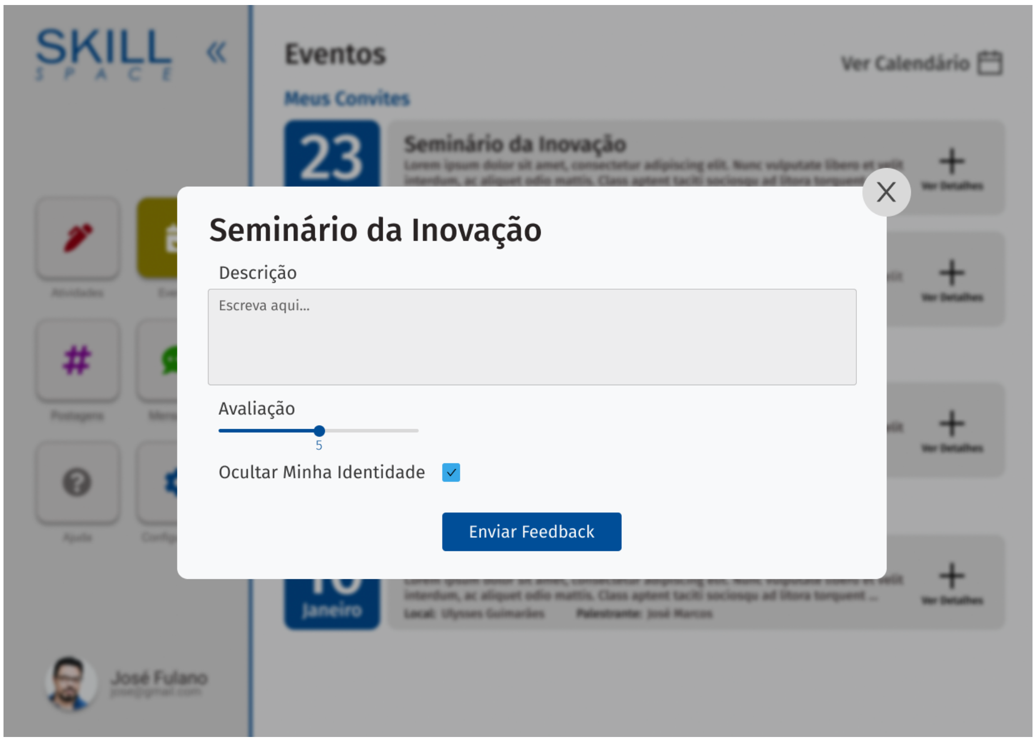
Ulysses Guimarães Ver no mapa

Palestrante

Pedro José
Empreendedor

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

Figure 4.9: Prototype of the Event details modal.


Seminário da Inovação

Descrição

Escreva aqui...

Avaliação

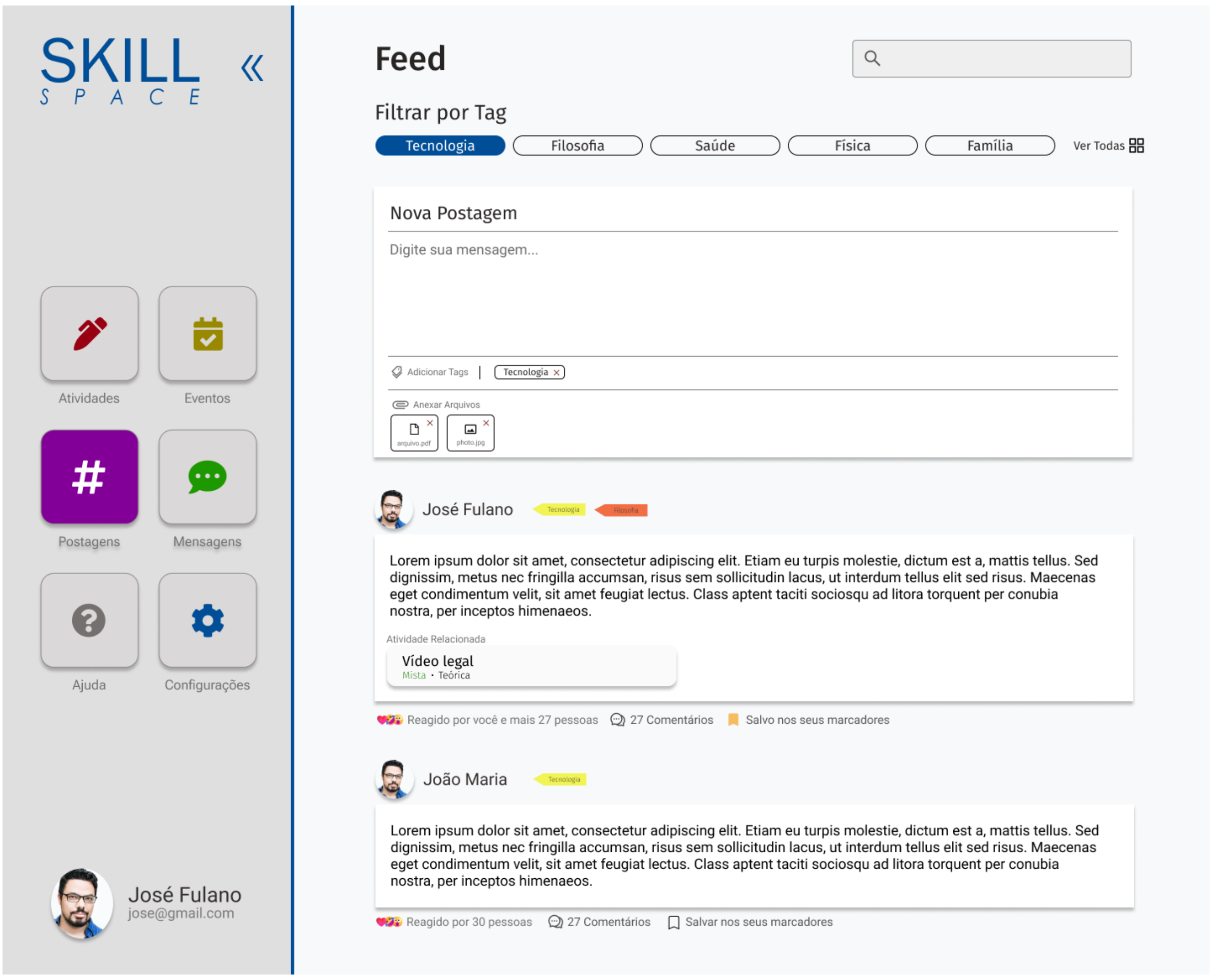
5

Ocultar Minha Identidade ✓

Figure 4.10: Prototype of the feedback submission modal.

4.5 Social Feed and Messages Capabilities

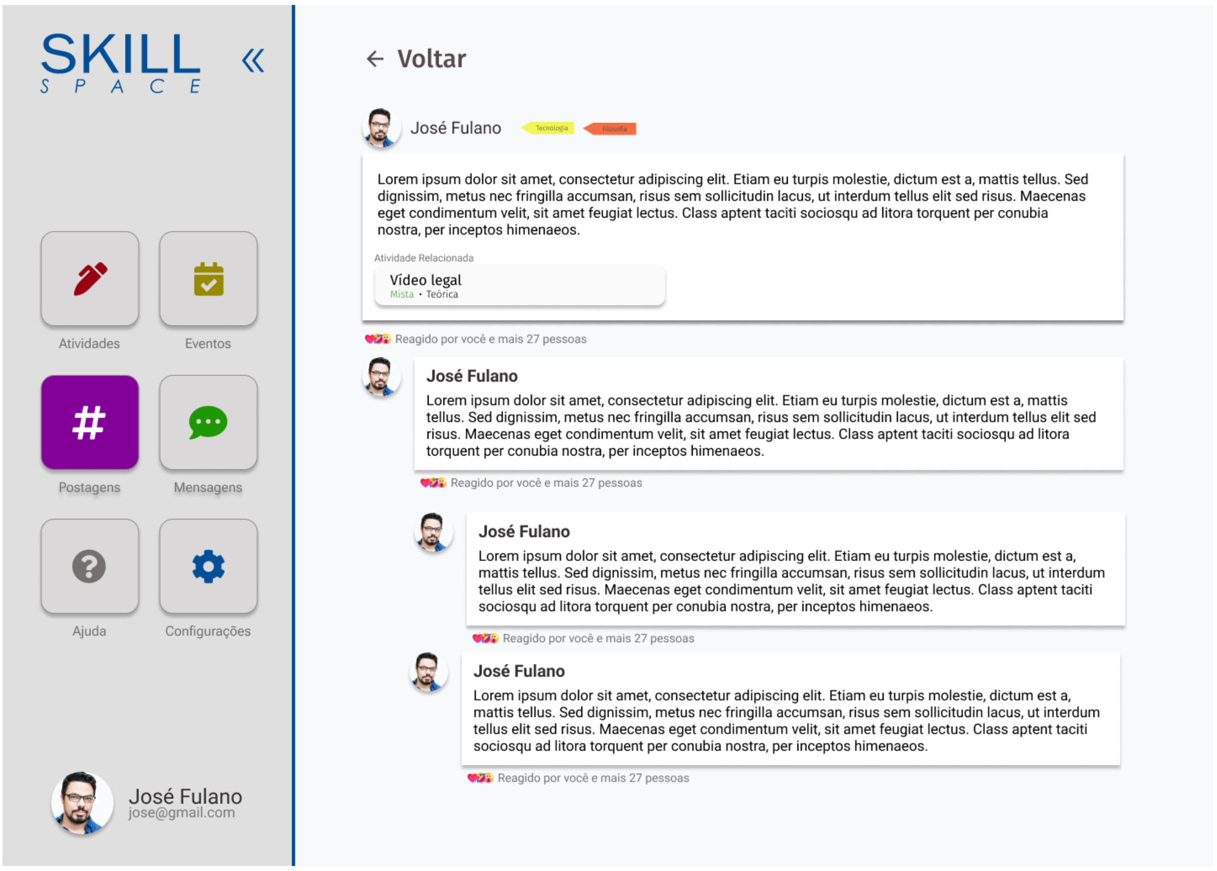
The social area features a post feed like the ones found in social networks like Facebook, LinkedIn, and Twitter, with a field to write a new post on top and the posts itself in an infinite scroll view, as can be seen in Fig. 4.11. When writing a new post, the user can, besides writing the text it features, add tags that fits on the topic being written and include files to be linked with the post. When viewing posts on the feed, the user is able to visualize the user that created the post, its content and the quantity of reaction and comments. When clicking in a post, the page in Fig. 4.12 will appear, with the user being able to visualize all the comments made to that post and click on its reaction to view details as to how many of each reaction was given to the post. Still in the feed, the users are also able to find a filter on top to browse for posts inside a specific tag, instead of featuring all posts of the platform.



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.

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.

Figure 4.11: Prototype of `Feed` page.



José Fulano

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

José Fulano

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 você e mais 27 pessoas

José Fulano

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 você e mais 27 pessoas

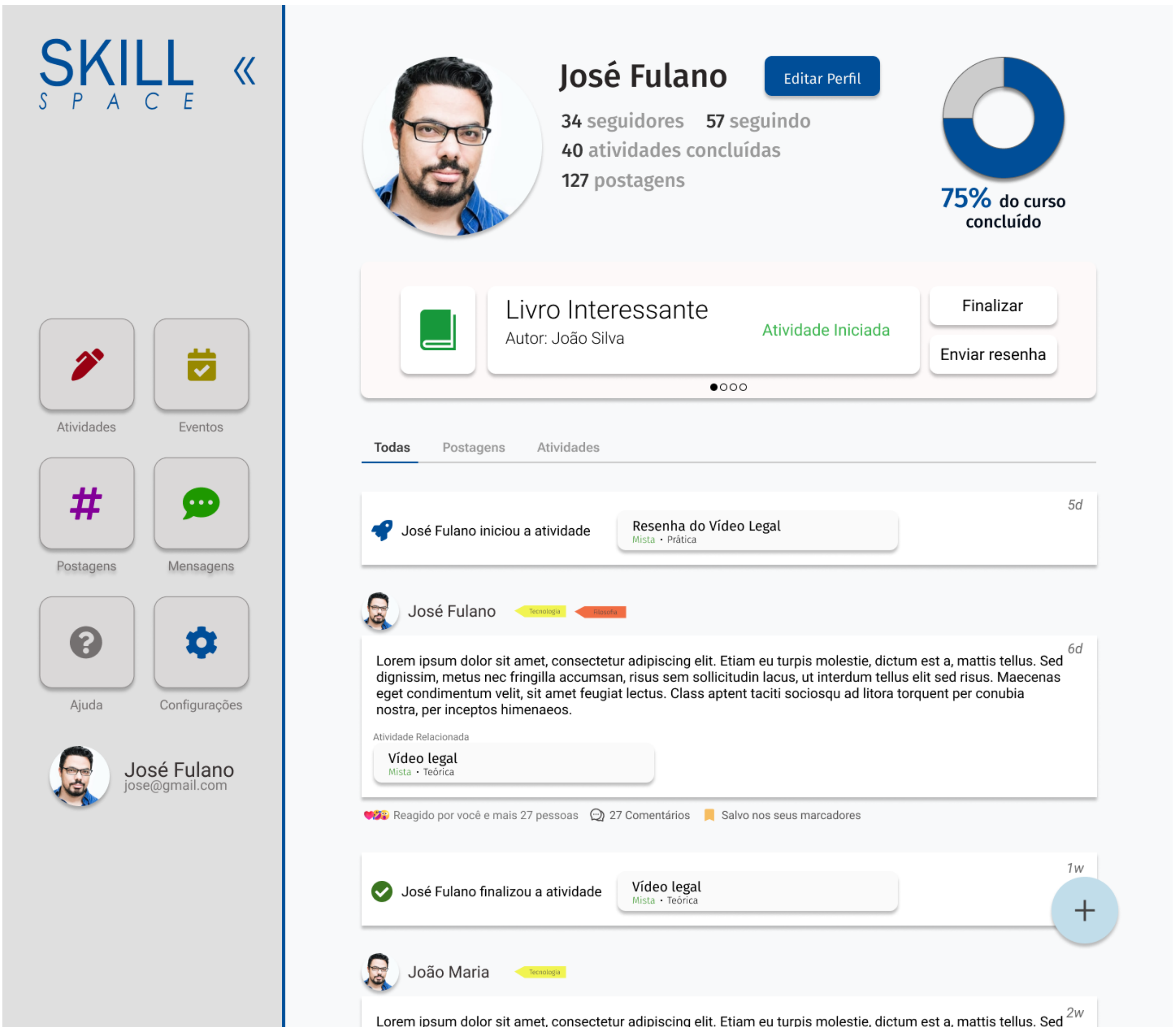
José Fulano

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 você e mais 27 pessoas

Figure 4.12: Prototype of `Post` comments page.

When entering in a member profile page, the user will be able to see all posts made by that member and updates on the activities progress for that member. From this page, the user can also follow the member and see overall statistics of its follower and following count and of its posting and activities information. If a user enters in its own profile, beside his own posts it will be shown an overall progress on the activities and cards featuring started activities that the user can continue doing, which can be seen in Fig. 4.13.



José Fulano
jose@gmail.com

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

Livro Interessante
Autor: João Silva
Atividade Iniciada

José Fulano iniciou a atividade Resenha do Vídeo Legal
Mista • Prática
5d

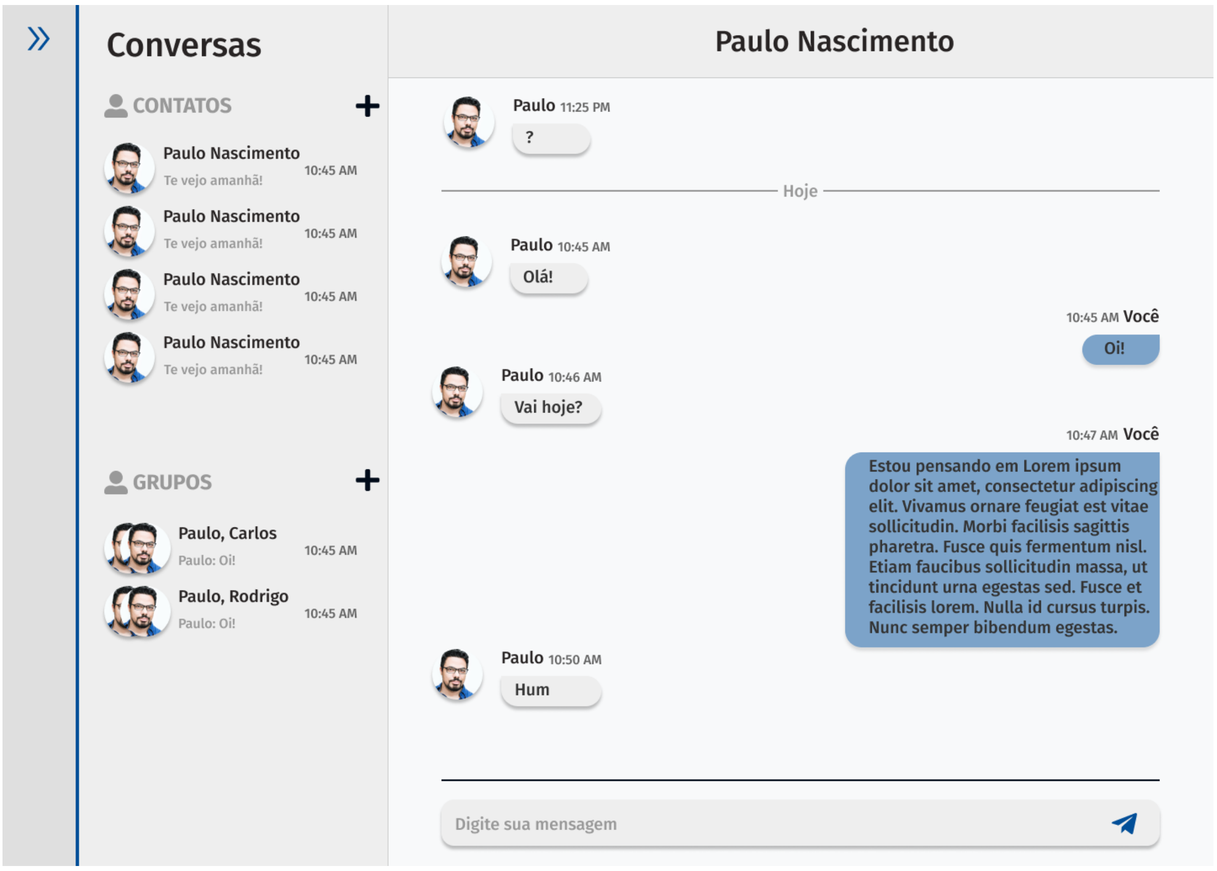
José Fulano
6d
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eu turpis molestie, dictum est a, mattis tellus. Sed dignissim, metus nec fringilla accumsan, risus sem sollicitudin lacus, ut interdum tellus elit sed risus. Maecenas eget condimentum velit, sit amet feugiat lectus. Class 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

José Fulano finalizou a atividade Vídeo legal
Mista • Teórica
1w

João Maria
2w
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam eu turpis molestie, dictum est a, mattis tellus. Sed

Figure 4.13: Prototype of the user own `Profile` page.

As the last section of the social area, the user can partake on chats with other users or chat groups featuring multiple users. The chat area is like those found in several other messaging services and can be seen in Fig. 4.14, with speech bubbles representing each participant in the chat with timestamps for each message.



Paulo Nascimento — Te vejo amanhã! — 10:45 AM

Paulo Nascimento — Te vejo amanhã! — 10:45 AM

Paulo Nascimento — Te vejo amanhã! — 10:45 AM

Paulo Nascimento — Te vejo amanhã! — 10:45 AM

Paulo, Carlos — Paulo: Oi! — 10:45 AM

Paulo, Rodrigo — Paulo: Oi! — 10:45 AM

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 4.14: Prototype of the `Chat` page.

4.6 The Administrator Dashboard

The administrator dashboard is planned as the main area to administrators manage the space, by registering events, activities, stages, and tags, and visualizing the users submissions and providing feedback to them. The admin area appears as a separate one from the rest of the platform, featuring a different sidebar, seen in Fig. 4.15, that, instead of showing the areas of the site, provides links to every single option on the dashboard, categorized by the functionality it belongs to in collapsible buttons.

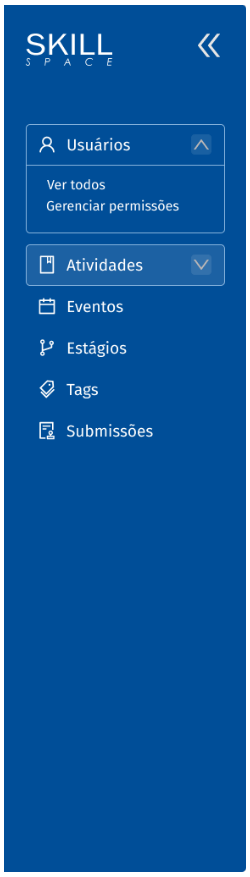
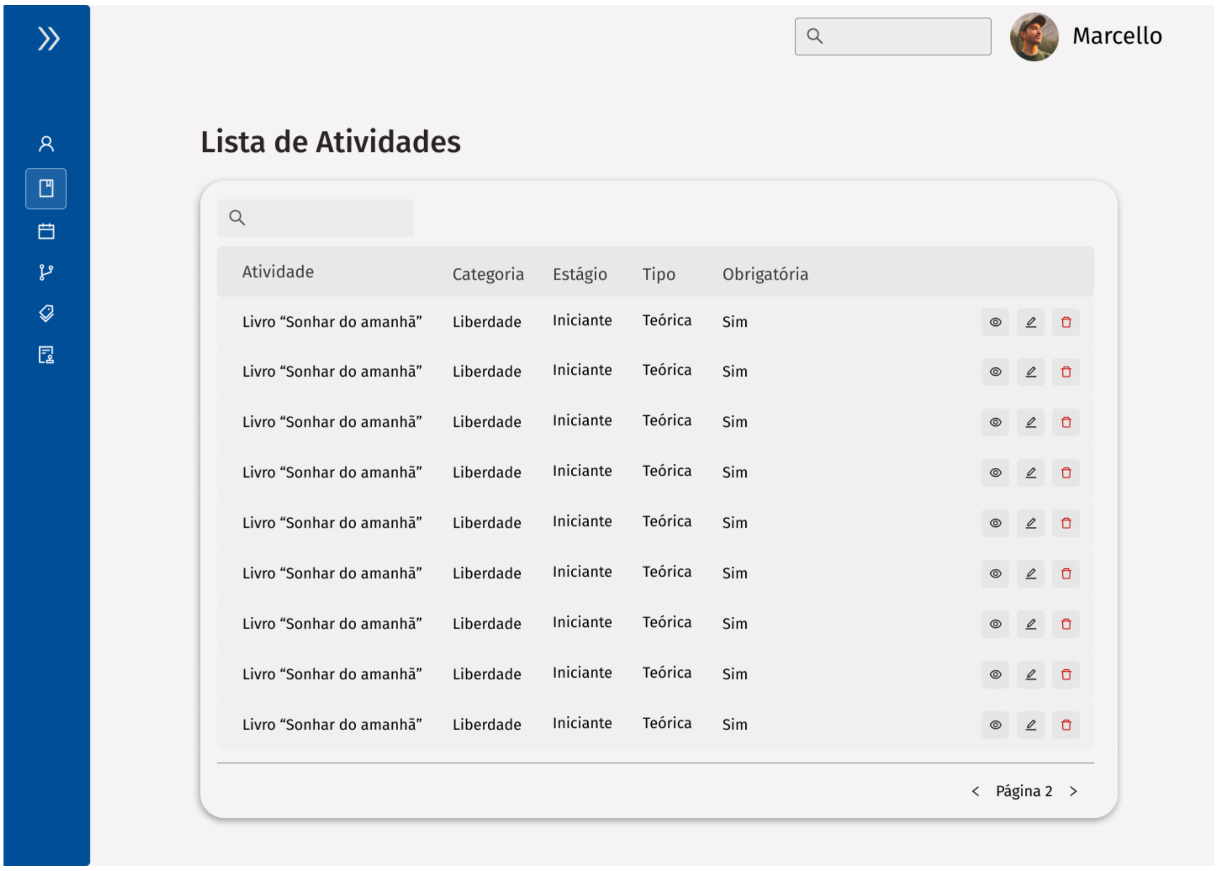


Figure 4.15: Prototype of the Admin navigation sidebar.

While these pages were not implemented on the final version of the front-end of the website, three types of pages were designed on Figma to represent its functionalities. The first type of page are the indexes, seen in Fig. 4.16, which are tables that features all instances of a certain model registered on the database, showing each of its fields, options to view, delete or edit them and a search field to easily find the instance that the administrator is looking for. The second type of page is shown in Fig. 4.17 and is called the visualization page, that simply features a more detailed view on the item registered in the database, which may feature information hidden from the table due to size constraints. Finally, the forms in Fig. 4.18 are the pages used to create or edit information on the platform, each featuring all the fields required to create an instance of the desired model on the database.

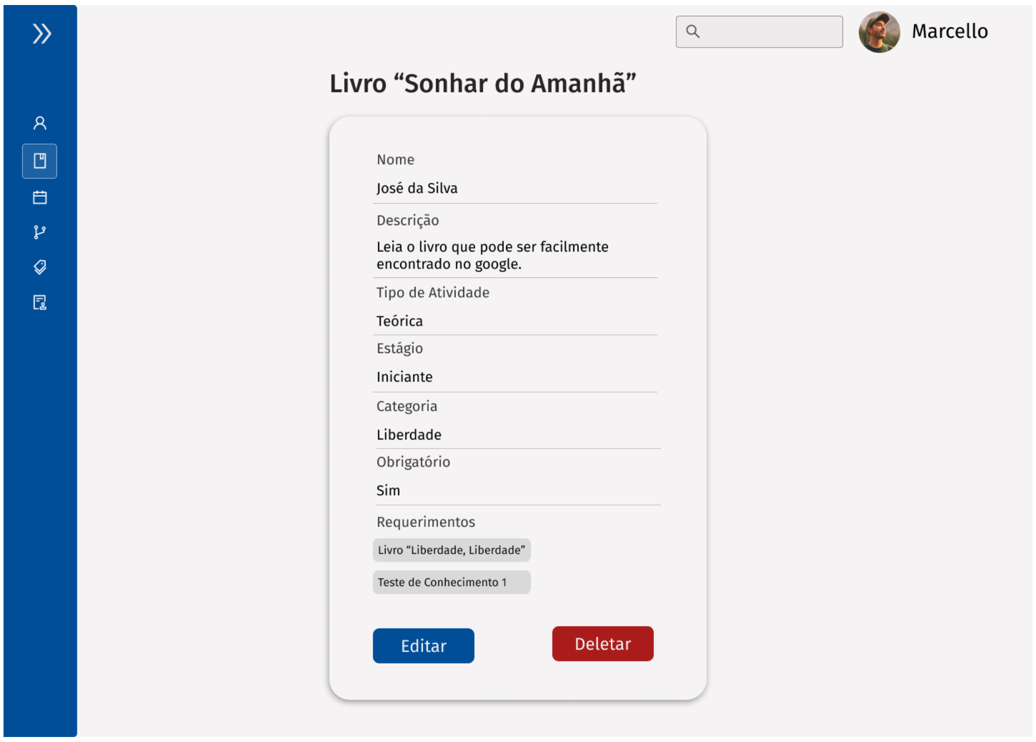

Marcello

Lista de Atividades

Atividade	Categoria	Estágio	Tipo	Obrigatória
Livro “Sonhar do amanhã”	Liberdade	Iniciante	Teórica	Sim
Livro “Sonhar do amanhã”	Liberdade	Iniciante	Teórica	Sim
Livro “Sonhar do amanhã”	Liberdade	Iniciante	Teórica	Sim
Livro “Sonhar do amanhã”	Liberdade	Iniciante	Teórica	Sim
Livro “Sonhar do amanhã”	Liberdade	Iniciante	Teórica	Sim
Livro “Sonhar do amanhã”	Liberdade	Iniciante	Teórica	Sim
Livro “Sonhar do amanhã”	Liberdade	Iniciante	Teórica	Sim
Livro “Sonhar do amanhã”	Liberdade	Iniciante	Teórica	Sim
Livro “Sonhar do amanhã”	Liberdade	Iniciante	Teórica	Sim

Figure 4.16: Prototype of the Index pages from the Admin dashboard.

With the prototype of the web site made using `Figma`, the next and final step of the platform development is to actually implement theses pages into a functional site, also connecting the created front-end to the back-end described in the previous chapter.


Marcello

Livro “Sonhar do Amanhã”

Nome
José da Silva

Descrição
Leia o livro que pode ser facilmente encontrado no google.

Tipo de Atividade
Teórica

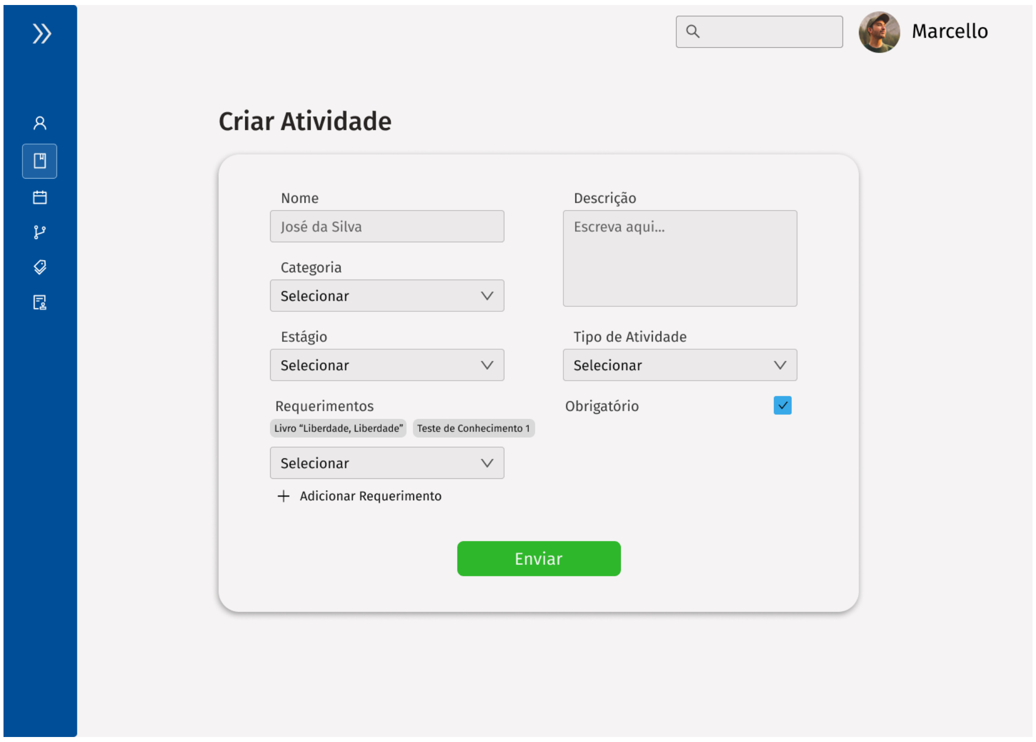
Estágio
Iniciante

Categoria
Liberdade

Obrigatório
Sim

Requerimentos
Livro “Liberdade, Liberdade”
Teste de Conhecimento 1

Figure 4.17: Prototype of the Visualization pages from the Admin dashboard.


Marcello

Criar Atividade

Nome: José da Silva

Descrição: Escreva aqui...

Categoria: Selecionar

Estágio: Selecionar

Tipo de Atividade: Selecionar

Requerimentos: Livro “Liberdade, Liberdade” | Teste de Conhecimento 1

Selecionar

Obrigatório ✓

Figure 4.18: Prototype of the Form pages from the Admin dashboard.

Chapter 5

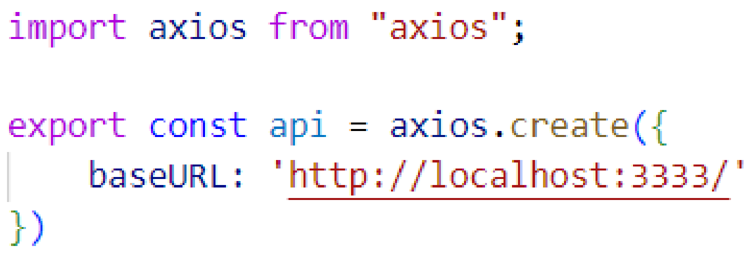
Front-End Development

The basis for any modern web page is the combination of three languages: `HTML`, describing the content and organization of a web page; `CSS`, providing styling for the pages; and `JavaScript` (`JS`), which adds a layer of interactivity on the website. While these three languages alone provide the tools to create a modern website, many other technologies and frameworks can be used to either help the front-end development or improve the website capabilities. One of these frameworks is `ReactJS` [21], a `JavaScript` library that allow developers to build user interface based on UI components. These components are standalone UI elements that can be coded once and easily reused across the whole website. For instance, let's consider you have a button on your website, with its own `HTML`, `CSS`, and `JS` code, that will be used across different pages on the site, instead of copying and paste its entire code on each instance of its usage, `ReactJS` allows developers to easily create a component for the button and use it on the entire site. This also helps if you decide to update your button, either its style or behavior, a simple change in the component and it will reflect across all pages that uses it, without needing to manually change every instance like you would have using a copy-and-paste method.

`ReactJS` also creates a `Virtual DOM` [22] while rendering you web pages, that stores a data-structure cache of the page and computes the difference with the real `DOM` [23] to only change in the page the elements that are needed, avoiding the browser to recalculate the `CSS` style and render the entire page again, in a process called reconciliation. For example, if your website features a `navbar` containing navigation shortcuts that should appear on every page on your site, the `ReactJS` library, instead of rendering the `navbar` in every page change, will instruct the browser to only load the actual changes in the page, saving resources and improving the overall performance while browsing the website. This `Virtual DOM` feature is also useful while using `states` on `ReactJS`, that are special types of variables that React keeps watching for changes on it to be rendered on the website. When using traditional `JavaScript` variables, if their values are updated while the site is being used, either by an API request or user interaction, the visual components are not changed unless explicit requested to. Using `ReactJS states` makes this update on the elements automatic, and, using the reconciliation feature, only the components that use the updated `state` changes, without the need to reload what remained equal.

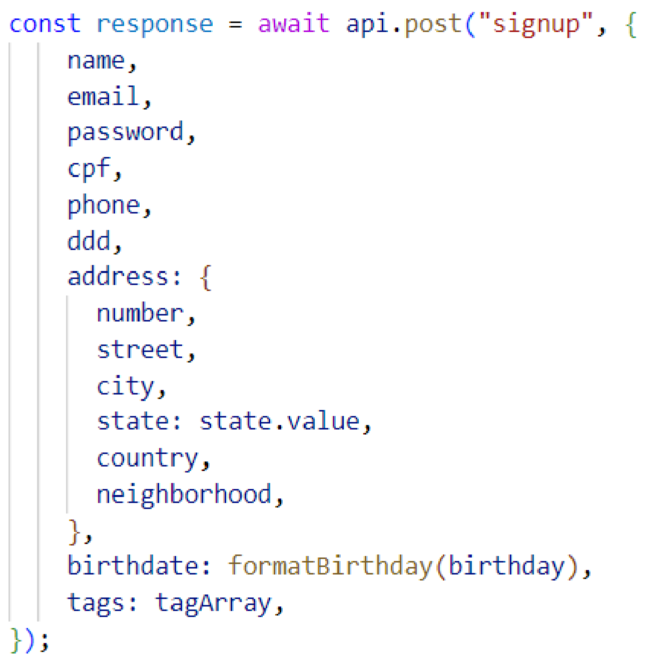
5.1 Making API Requests

To actually make the implemented pages functional, we need a way to communicate the front-end pages to the API, which, in this project, it is done using the `Axios` [24] package for `ReactJS`. `Axios` allows the creation of an object instance with some main configurations used on the requests, such as the base URL for use on those requests, as can be seen in the code snippet in Fig. 5.1. This instance gives access to all request type methods, such as `get`, `post`, `pacth`, `put` and `delete`. Then, the created `Axios` instance can be imported to each desired page on the web site, in order for a request to be made. The `get` requests are usually made inside another `ReactJS` feature, the `useEffect hooks`, which contains code to be run in each page render, since it is needed to load data from the API to properly display the desired pages. Other types of methods, instead, may be used outside the `useEffect hook`, being called when needed or requested by the user. In Fig. 5.2, an example of a `post` request is shown, where both the API endpoint and the request body is specified to the method.



```
import axios from "axios";

export const api = axios.create({
    baseURL: 'http://localhost:3333/'
})
```

Figure 5.1: Code snippet of the creation of an `Axios` instance.

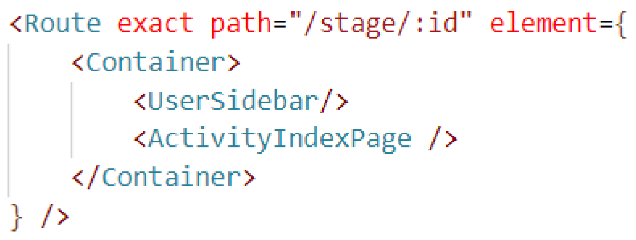


```
const response = await api.post("signup", {
    name,
    email,
    password,
    cpf,
    phone,
    ddd,
    address: {
      number,
      street,
      city,
      state: state.value,
      country,
      neighborhood,
    },
    birthdate: formatBirthday(birthday),
    tags: tagArray,
});
```

Figure 5.2: Code snippet of a `post` request using `Axios`.

5.2 Navigating Through the Front-End Pages

To proper map each created page component into a route in the application, the `React Router` package [25] was used. This package allows for the creation of a `Router` component, containing all the `routes` definition inside, which is imported in the main `ReactJS` file, `App.js`. The routes definition, as seen in Fig. 5.3, defines the URL path for the route access and the component element, or the page, mapped to the route. This is also the location where fixed components, such as the sidebar, are called. `React Router` also provides access to the components to a `useNavigate hook`, which can be used to redirect to other pages of the application when needed.



```
<Route exact path="/stage/:id" element={
    <Container>
        <UserSidebar/>
        <ActivityIndexPage />
    </Container>
} />
```

Figure 5.3: Code snippet of a `route` definition using `React Router`.

5.3 Managing The User Context

Since most of the pages from the web site requires the user to be logged in, it is essential for our front-end application to be able to store and retrieve the users credentials during runtime, also providing access to this information across all pages of the platform. To achieve this goal, `ReactJS` provides the ability to use `contexts`, which are special components that creates a way to pass data through the component tree without manually providing `props` to each page or component that requires the context. This functionality is then used to create the `user context`, that stores the `user state`, with all the user information retrieved upon login, the `login` and `sign_out` methods and the functionality to retrieve the user data if its lost upon the page reload.

The `login` method receives the user `email` and `password` and makes a request to the appropriate route on the API, logging in the user. Upon receiving the API response, the function also sets the `user state` and set `Axios Authorization` header with the retrieved `JSON Web Token`, to be used on all future requests from the platform. The user data and `Token` are also stored in the web browser `Cookies` using the `js-cookie` [26] `ReactJS` package, to be retrieved in case the `user state` is lost during page reloads. To secure the user information, this `Cookies` is encrypted using the `crypto-js` [27] package, with a key stored in the project environment variables. The `sign_out` method, in contrast, unset the `user state`, the `Cookies` and the `Axios` headers.

As previously stated, the `user state` may be lost during page reloads, or when the user closes the page and gets back in a latter moment. To handle this situation, the `user context` also uses the `ReactJS useEffect hooks` feature. The `useEffect` code inside the `context` simply access the stored `Cookies` and use the retrieved information to set the `user state` and `Authorization` headers. Since the `user context` is available across all pages on the website, this `hook` is prepared to run across all the pages as well.

5.4 Achieved Implementation

Following the implementation details previously described in this document, a functional prototype of the SkillSpace platform was developed, featuring the functionalities deemed essential to demonstrate the platform capabilities. Due to time constraints and hindrances found during the application development, some of the intended functionalities were not able to be implemented on time, which will be better discussed on the next section of this chapter.

Regarding the back-end development, most of the intended functionalities were successfully developed, with the entire of the projected database design being implemented alongside API endpoints to manipulate and manage these models. It was successful in the development of authentication measures for the user model, alongside endpoint protections to limit its accesses from non-authorized personnel, such as common users and administrators without the necessary clearance to the functionality. The connection with third-party services, like Amazon S3, was also made to allow a reliable and efficient storage of files uploaded to the system.

Regarding the use of the `Figma` software to design the web pages, its use was proven to be essential once the actual front-end development started. Not only it provided a quick way to plan and test different designs and prototypes to the required page, also gathering feedback after each page was done, the prototypes provided a more streamlined development once the pages were implemented on `ReactJS`, also helping with the provided `CSS` code on the inspect feature of `Figma`. One area where the prototyping was cut short was while designing the admin dashboard pages. Since most of it was intended to be very similar to each other, only the three pages were created, one of each of the three types described on Section 4, to avoid wasting time by doing a huge number of pages that would be similar to each other.

The front-end development, being the last step in the development process, was the most affected by the time constraints mentioned, but it was still able to make a usable demonstration of all the platform functionalities from the user standpoint. By that, the user can navigate on the platform using the sidebar and exploring the social, activities and events areas of the web site. In the developed prototype, it is possible for the user to navigate across the platform stages, start and finish activities and track its overall progress on them. In the events area, the user can freely see all the upcoming events and state their intent of participation in them. Finally, the users can also create posts to the platform and navigate across the ones created by other members of the Space. The code for the implemented prototype can be found in the `GitHub` repositories for the back-end [28] and the front-end [29].

5.5 What was left from the scope?

On the initial stage of this project, while gathering information about the project requirements, it was intended to feature more functionalities on the platform relating to the administration management, such as financial management for paid spaces, featuring options to directly connect to payment providers and managing invoices, and also more analytics visualizations for each functionality of the platform, from graphs with most popular posts and tags on the social area to statistics about the general evolution of the users on the activities. There was also a plan to include gamification options to the users, that would reward them with badges and exclusives customization options for their evolution on the activities, participation on events and interactions in the social area. These features were already not planned to be implemented on the initial scope of the platform due to concerns with the time constraints of the project and the overall complexity that these features, that were not considered essential for the platform use, would take to be implemented.

Besides that, some features that were on the initial scope, most from the front-end, were not implemented due to time constraints, the most prominent one being the administrator dashboard. Even though the sidebar for the dashboard was created as a component in `ReactJS`, with its code being latter reused while creating the user sidebar, the pages itself were not developed, thus not being possible to register items to the database from the website. These features, however, does exist on the back-end as endpoints accessible from the API, thus being possible to make direct requests to it to simulate what should be possible from the admin dashboard.

Also left from the front-end implementation was the `Chat` page, another functionality that, although working on the back-end, has no means of accessing it from the web platform. Some other functionalities, such as a calendar view for the events, the ability to provide feedback for the events and a way to finish `Social` activities from the `Activity Page`, also falls under the category of left out implementations on the front-end that are available to use through the API.

As far as functionalities that were limited by not being implemented on the back-end, one would be the ability to finish `Events` activities, due to the logic of finishing this type of activity being quite different from the other type available. The creation of personalized `feed` suggestions based on the user followed users and tags was also not implemented on time due to the high complexity to develop the feature. Finally, deemed not essential for the prototype, there is also no way for a user to regain its access to the platform in case of forgetting the password.

Chapter 6

Conclusion

This project started with the goal of creating a unified educational platform to allow institutions, companies and even individuals to create a space where its members, clients or followers could learn with the provided material in a course-like learning environment, participate in events promoted by the space organizers and interact with each other, building a unified, collaborative, and secure space for these educational means, all in one single platform.

The developed application, while not complete with all intended functionalities, shows a working prototype with that initial vision implemented, with the main functionalities working as intended. Administrators can use the designed platform to design simple or complex courses while users can easily navigate through it, while also explore the community features available to them. The developed prototype code can also continue to grow in order to become a full-fledged tool or product to be used by organizations of different kinds to help them achieving their needs and goals.

The intention of this project is that institutions and providers of educational courses, after starting to use the SkillSpace application, could see an increase in the engagement of the users, transforming online learning experiences from an almost impersonal experience to one where they can find support and build a trustable network among its peers with similar interests and objectives, improving their overall satisfaction and retaining them in the institution.

6.1 Future Work

In future revisions of the developed application described in this document, many new features and improvements are planned to be made. Besides overall improvements and validations on functionalities already on the website, the first step for future versions would be to allow administrators to properly manage the platform from the web site, by implementing the administrator dashboard that is missing from the current version of the software. It is also important to complete the left out functionalities described in Section 5.5.

Another step would be to implement better statistics and analytic functionalities to help administrators manage the website, billing options with the integration of third-party payment providers and gamification options for the users, to further increase their participation on the platform. Besides that, the API developed was made with the intent to be used with multiple different front-ends, and in future works, it would be interesting to develop a SkillSpace mobile application, to better fit in the way users nowadays consume and use internet content. Furthermore, since the web front-end was done using `ReactJS`, part of its code could be reused on the mobile development, by using frameworks like React Native and Ionic.

References

[1] *IFL Brasil's institutional website.* `https://iflbrasil.com.br/`, visited on 2022-27-09. 1

[2] *IFL Brasil's platform to manage associated member's progress.* `https://iflportal.azurewebsites.net/`, visited on 2022-27-09. 1

[3] Bartholomew, Daniel: *SQL vs. NoSQL.* `https://dl.acm.org/doi/fullHtml/10.5555/1883478.1883482`, visited on 2022-27-09. 4

[4] *dbdiagram.io official website.* `https://dbdiagram.io/home`, visited on 2022-27-09. 4

[5] *Node.js official website.* `https://nodejs.org/en/`, visited on 2022-27-09. 19

[6] *MySQL official website.* `https://www.mysql.com/`, visited on 2022-27-09. 19

[7] *NPM official website.* `https://www.npmjs.com/`, visited on 2022-27-09. 19

[8] *Express.js official website.* `https://expressjs.com/`, visited on 2022-27-09. 19

[9] *cors package repository.* `https://github.com/expressjs/cors`, visited on 2022-27-09. 19

[10] *body-parser package repository.* `https://github.com/expressjs/body-parser`, visited on 2022-27-09. 19

[11] *CORS documentation on MDN Web Docs.* `https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS`, visited on 2022-27-09. 19

[12] *Sequelize package repository.* `https://github.com/sequelize/sequelize`, visited on 2022-27-09. 20

[13] Pop, Dragos Paul and Adam Altar: *Designing an MVC model for rapid web application development.* Procedia Engineering, 69:1172–1179, 2014, ISSN 1877-7058. `https://www.sciencedirect.com/science/article/pii/S187770581400352X`, 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013. 20

[14] *bcryptjs package repository.* `https://github.com/dcodeIO/bcrypt.js`, visited on 2022-27-09. 23

[15] *jsonwebtoken package repository.* `https://github.com/auth0/node-jsonwebtoken`, visited on 2022-27-09. 24

[16] *JSON Web Token official standard website.* `https://jwt.io/`, visited on 2022-27-09. 24

[17] *multer package repository.* `https://github.com/expressjs/multer`, visited on 2022-27-09. 26

[18] *aws-sdk package repository.* `https://github.com/aws/aws-sdk-js`, visited on 2022-27-09. 26

[19] *multer-s3 package repository.* `https://github.com/anacronw/multer-s3`, visited on 2022-27-09. 26

[20] *Figma official website.* `https://www.figma.com/design/`, visited on 2022-27-09. 27

[21] *ReactJS official website.* `https://reactjs.org/`, visited on 2022-27-09. 43

[22] *Virtual DOM documentation on ReactJS official website.* `https://reactjs.org/docs/faq-internals.html`, visited on 2022-27-09. 43

[23] *DOM documentation on MDN Web Docs.* `https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction`, visited on 2022-27-09. 43

[24] *Axios package repository.* `https://github.com/axios/axios`, visited on 2022-27-09. 44

[25] *React Router official website.* `https://reactrouter.com/en/main`, visited on 2022-27-09. 45

[26] *js-cookie package repository.* `https://github.com/js-cookie/js-cookie`, visited on 2022-27-09. 46

[27] *crypto-js package repository.* `https://github.com/brix/crypto-js`, visited on 2022-27-09. 46

[28] Vaz, Felipe L. and André M. P. Vale: *SkillSpace back end repository.* `https://github.com/andremacedopv/skill-space-api`, visited on 2022-27-09. 47

[29] Vaz, Felipe L. and André M. P. Vale: *SkillSpace front end repository.* `https://github.com/andremacedopv/skill-space-front`, visited on 2022-27-09. 47

Appendix A

Non-standard methods in controller

Method	**Controller**	**Description**
`addRequirements`	Activities	Adds new requirements for an activity
`addDependents`	Activities	Add a dependency for an activity
`requirements`	Activities	Return all requirements of an activity
`dependents`	Activities	Return all dependents of an activity
`start`	ActivityUsers	Start an Activity
`userIndex`	ActivityUsers	Retrieves all activities started by the user
`mySubmission`	ActivityUsers	Get the user submission for an activity
`userSubmission`	ActivityUsers	Allow the administrator to get the user submission of an activity
`editSubmission`	ActivityUsers	Allows the user to edit a submission
`indexSubmissions`	ActivityUsers	Retrieves all submissions made by users
`giveFeedback`	ActivityUsers	Allows the administrator to provide feedback for a submssion

`editFeedback`	ActivityUsers	Allows the administrator to edit a feedback made for a submssion
`pendingFeedbacks`	ActivityUsers	Retrieves all submissions pending of feedback
`myChats`	Chats	Retrieves all chats by an user
`setInvites`	Events	Send invites to users for an event
`invites`	Events	Get all invites for an event
`myInvites`	Events	Get all invites sent to an user
`createFeedback`	Events	Send feedback to an event
`feedbacks`	Events	Get all feedbacks for an event
`follow`	Followers	Follow an user
`followers`	Followers	Retrieves all followers of an user
`following`	Followers	Retrieves all followings of an user
`followerCount`	Followers	Retrieves the amount of users following an account
`followingCount`	Followers	Retrieves the amount of users an account follows
`confirmPresence`	Guests	Confirm intention of participation on an event
`comments`	Posts	Retrieves all comments from a post
`react`	Reactions	React to a post
`removeReaction`	Reactions	Remove the reaction to a post
`userReactions`	Reactions	Retrieves all posts the user reacted to
`postReactions`	Reactions	Retrieves all reactions to a post
`reactionCount`	Reactions	Retrieves the reaction count by type of reaction for a post
`startStage`	Stages	Start a stage
`finishStage`	Stages	Finish a stage
`myStages`	Stages	Retrieves all stages started by the user

`myStageActivities`	Stages	Retrieves all activities from a stage started by the user
`follow`	Tags	Follow a tag
`toggleFollow`	Tags	Toggle following a tag
`unfollow`	Tags	Unfollow a tag
`followedTags`	Tags	Retrieve all followed tags
`signup`	Users	Create an account on the platform
`login`	Users	Login on the platform
`profile`	Users	Get the profile information of the logged user
`promote`	Users	Promote an user to admin
`demote`	Users	Demote an user from admin
`updatePermissions`	Users	Update the permissions of an admin
`deactivate`	Users	Deactivate an user
`activate`	Users	Activate an user
`updateProfilePicture`	Users	Update or remove a user profile picture

Table A.1: List of non-standard methods in the project controllers