Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Portal Aberto: Plataforma de acesso ao acervo de Psicologia Anomalística do Instituto de Psicologia da USP

Renan Costa Laiz
Renan Tiago dos Santos Silva

Monografia Final

mac 499— Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Marco Dimas Gubitoso

São Paulo
24 de Dezembro de 2021

Agradecimentos

Ao professor Dr. Marco Dimas Gubitoso, orientador deste trabalho, que mesmo lutando contra um câncer nos acolheu e nos auxiliou durante o desenvolvimento do projeto.

Ao professor Dr. Wellington Zangari, cliente deste trabalho, que esteve conosco desde o início do desenvolvimento até a entrega da plataforma.

Aos nossos amigos, que nos deram apoio, motivação e energia para a conclusão deste trabalho.

Às nossas famílias, que sempre estiveram conosco, nos apoiando e provendo todo o suporte para chegarmos até aqui.

Resumo

Renan Costa Laiz Renan Tiago dos Santos Silva. **Portal Aberto: Plataforma de acesso ao acervo de Psicologia Anomalística do Instituto de Psicologia da USP**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

Em uma conversa entre o professor Dr. Marco Dimas Gubitoso, docente do Instituto de Matemática e Estatística (IME-USP) e orientador deste trabalho, e o professor Dr. Wellington Zangari, docente do Instituto de Psicologia (IP-USP) e cliente deste projeto, surgiu a ideia de se criar uma plataforma *web* para as obras de um novo acervo do Instituto de Psicologia, o que acabou por tornar-se o objetivo deste trabalho. A plataforma desenvolvida neste projeto tem o intuito de divulgar e gerenciar as obras do acervo, que conta com mais de 8 mil títulos de livros e outros tipos de mídias, como CDs, DVDs, slides, revistas, entre outros, relacionados à religiosidade popular e psicologia anomalística, área da psicologia que estuda experiências anômalas. Nesta plataforma, o usuário vai poder buscar pelas obras, ver os metadados destas, assim como avaliar e comentar os itens. Neste trabalho, o objeto de interesse foram os livros, mas a plataforma foi projetada para receber facilmente os outros tipos de mídias no futuro. No desenvolvimento da plataforma foram utilizados *frameworks*, com *Ruby on Rails* para o *back-end* e *Vue.JS* para o *front-end*, assim como um banco de dados não-relacional, o *MongoDB*. *Softwares* para organização de tarefas e código também foram utilizados, utilizando-se o *Jira* e o *GitLab*, nos quais foi possível aplicar os conhecimentos de métodos ágeis que os alunos aprenderam durante a graduação, como o conceito de *Sprint*, *Kanban* e *Peer Code Review*. A plataforma foi muito bem recebida pelo professor Wellington, na qual foram desenvolvidas as principais funcionalidades que foram acordadas durante o desenvolvimento do projeto. A realização deste projeto possibilitou que os alunos pudessem colocar em prática conhecimentos adquiridos durante a graduação, assim como experienciar as etapas do desenvolvimento de um *software* do seu início até a sua entrega.

Palavras-chave: Desenvolvimento Web. Psicologia Anomalística. Metodologias Ágeis.

Abstract

Renan Costa Laiz Renan Tiago dos Santos Silva. **Portal Aberto: Access platform for the Anomalistic Psychology collection of the Institute of Psychology at USP**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2021.

In a conversation between Marco Dimas Gubitoso, Ph.D. professor at the Institute of Mathematics and Statistics (IME-USP) and supervisor of this work, and Wellington Zangari, Ph.D. professor at the Institute of Psychology (IP-USP) and client of this project, the idea of creating a web platform for the items of a new collection of the Institute of Psychology came up, which became the goal of this work. The platform developed in this project aims to disseminate and manage the items of the collection, which has more than 8 thousand titles of books and other types of media, such as CDs, DVDs, slides, magazines, among others, related to popular religiosity and anomalistic psychology, an area of psychology that studies anomalous experiences. On this platform, the user will be able to search the items, view their metadata, as well as rate and comment on the items. In this work, the object of interest was the books, but the platform was designed to easily accommodate other types of media in the future. In the development of the platform, frameworks were used, such as Ruby on Rails for the back-end and Vue.JS for the front-end, as well as a non-relational database, MongoDB. Software for organizing tasks and code was also used, using Jira and GitLab, in which it was possible to apply the knowledge of agile methods that the students learned during graduation, such as the concept of Sprint, Kanban and Peer Code Review. The platform was very well received by professor Wellington, in which the main functionalities that were agreed upon during the development of the project were delivered. The realization of this project allowed the students to put into practice their knowledge acquired during graduation, as well as experience the stages of developing a software from the beginning to delivery.

Keywords: Web Development. Anomalistic Psychology. Agile Methodologies.

Sumário

Capítulo 1

Introdução

O Instituto de Psicologia da USP (IP-USP) criou um acervo composto por uma biblioteca e um museu. A biblioteca compreende cerca de 8 mil títulos de livros, mas também conta com CDs, DVDs, fitas, revistas e slides. O museu é formado por centenas de objetos, sobretudo ligados à religiosidade popular e psicologia anomalística, ramo da psicologia que estuda experiências anômalas.

De acordo com o Prof. Dr. Wellington Zangari, professor do Instituto de Psicologia e cliente do projeto deste trabalho de conclusão de curso, o acervo trata-se de um "Biblio-Museu da Crença", composto por materiais ligados principalmente aos seguintes temas: crenças religiosas, crenças paranormais, hipnose e arte mágica.

Algumas das obras presentes no acervo são extremamente raras e, para fins de preservação, não podem ser retiradas do local, de forma que elas ficarão disponíveis apenas para consultas físicas. Como consequência, a exposição das obras ao público tornou-se limitada.

1.1 Objetivo

O objetivo deste projeto é a criação de uma plataforma *web* para reunir as informações do acervo em um único lugar, no qual usuários poderão consultar os itens presentes no acervo, assim como poderão usar a plataforma como uma ferramenta para avaliar e comentar as obras. Mais do que isso, a plataforma tem o intuito de ser uma ferramenta de novas descobertas para usuários interessados na temática do acervo, assim como uma ferramenta que promova a visibilidade dessas obras. Em um primeiro momento, e sendo objeto de interesse deste trabalho, a plataforma irá lidar apenas com os livros do acervo. No entanto, a construção da plataforma já foi pensada para se adicionar os outros tipos de itens que o acervo possui.

1.2 Público Alvo

O público alvo dessa plataforma é qualquer pessoa que se interesse pelos assuntos das obras do acervo, seja ela membro da comunidade USP ou não. A plataforma servirá não apenas como um chamariz para o Biblio-Museu da Crença, mas também como uma ferramenta de catalogação e gerenciamento digital do acervo para seus administradores.

1.3 Motivação

Antes mesmo da escolha do tema deste trabalho de conclusão de curso, ficou decidido entre os integrantes do grupo que o projeto deveria estar relacionado com a universidade, de modo que o trabalho pudesse ser utilizado pela comunidade USP de alguma forma.

Tendo isso em mente e tendo conversado com o professor Dr. Marco Dimas Gubitoso, orientador deste trabalho, este lembrou-se do novo acervo do IP-USP. Ao apresentar a proposta de uma plataforma para o acervo, esta foi muito bem recebida pelo professor Dr. Wellington Zangari, dando início, assim, a este trabalho de conclusão de curso.

1.4 Organização do Trabalho

Este documento está organizado em 8 capítulos. O primeiro sendo uma introdução a respeito do que se trata este trabalho de conclusão de curso. O segundo descreve as tecnologias e como estas estão aplicadas ao contexto da plataforma. O terceiro aborda a metodologia que foi empregada pelos integrantes para o desenvolvimento do projeto. O quarto capítulo refere-se à organização da plataforma, como as telas e funcionalidades desta. O quinto capítulo aborda a arquitetura da plataforma, descrevendo assuntos mais técnicos do projeto. O sexto capítulo aborda os resultados atingidos neste trabalho. Por fim estão o sétimo e oitavo capítulos, descrevendo como executar a plataforma e concluindo o que foi abordado neste documento, respectivamente.

Capítulo 2

Tecnologias

Solicitado pelo professor Wellington, a aplicação deste trabalho deveria ser acessada a partir de um navegador *web*, disponibilizando páginas na linguagem *HTML (Hypertext Markup Language)* para apresentação de dados para os usuários finais. Com isso em mente, o primeiro mês de desenvolvimento do projeto foi utilizado sobretudo para o estudo e definição das tecnologias e ferramentas que seriam utilizadas na plataforma.

Com a definição de que seria desenvolvida uma aplicação *web*, surgiu uma decisão a ser tomada: a aplicação deveria prover o processamento de dados junto com as páginas de apresentação *(HTML)* ou deveria separar a camada de processamento de dados e regra de negócio em uma aplicação e a camada de apresentação em outra?

A primeira vantagem que se observa ao se ter uma única aplicação que contém tanto o processamento de dados quanto a apresentação é que o tempo de desenvolvimento é reduzido drasticamente, se considerar que, ao implementar uma funcionalidade naquela aplicação, já é o que o usuário final irá enxergar. A manutenção do código e detecção de *bugs* também é facilitada, pois quaisquer erros que forem detectados estarão com certeza nessa aplicação.

Por outro lado, ao separar a aplicação responsável por processar e fornecer os dados e prover a segurança da plataforma (chamado de *back-end*) e uma aplicação que irá consumir os dados e prover a camada de apresentação (chamada de *front-end*), ganha-se uma vantagem muito interessante: a possibilidade de se ter diferentes aplicações *front-end* para consumir e apresentar os mesmos dados. No futuro, com a evolução desse projeto, a ideia é que os desenvolvedores criem uma aplicação para *smartphone* que possa consumir os mesmos dados da aplicação *back-end*.

A possibilidade de expandir o projeto foi a principal motivação para se escolher separar a camada de processamento da de apresentação, porém uma outra vantagem dessa abordagem é que a separação do *front-end* do *back-end* é algo muito utilizado pela comunidade de desenvolvedores atualmente, de forma que há diversos *frameworks* e ferramentas interessantes a serem utilizadas.

Desta forma, o *back-end* neste trabalho compreende um *framework web*, sendo utilizado *Ruby on Rails* e um banco de dados, sendo utilizado o *MongoDB*. Já o *front-end*, que nesse

trabalho utilizou-se o f*ramework Vue.JS*, teve sobretudo a função de prover uma interface gráfica e interativa para o usuário final da aplicação. Essas duas frentes serão melhor abordadas ao longo do capítulo.

Para auxiliar no desenvolvimento, tanto do *back-end* quanto do *front-end*, foram utilizadas ferramentas para o gerenciamento de tarefas e organização de código, garantindo uma maior rapidez na construção da plataforma, assim como uma melhor qualidade do resultado entregue.

2.1 Back-End

Durante o planejamento do projeto, foi decidido que o *back-end* implementaria uma arquitetura *REST* (Fielding, 2000), tornando a plataforma em um sistema *RESTful*. A arquitetura *REST (Representational State Transfer)* implementa um conjunto uniforme e predefinido de operações sem estado (onde cada operação não carrega dados de operações anteriores), no qual requisições são feitas à uma *API (Application Programming Interface)*.

Mais do que o fato da arquitetura *REST* ser comumente utilizada em serviços *web*, a escolha desta arquitetura para o projeto em questão teve como motivação o fato de ser uma padronização, fazendo com que seja possível que as *APIs* que sejam consumidas por outras aplicações de *front-end* de maneira similar, possibilitando que no futuro novas aplicações possam ser integradas à plataforma, como no caso de um aplicativo *mobile*, por exemplo.

Existindo diversos *frameworks* que são comumente utilizados como *API*, como *Spring Boot*, *Django* e *Ruby on Rails*, ficou decidido entre os membros da equipe que seria utilizado este último, junto com o banco de dados *MongoDB*. A decisão para utilizar essas duas ferramentas será descrita com mais detalhes a seguir.

2.1.1 Ruby on Rails

A escolha do *Ruby on Rails* como *framework* no *back-end* se deu por diversos motivos, como por exemplo a documentação oficial do *framework*, sendo esta bem detalhada e com diversos exemplos, ajudando a acelerar a curva de aprendizado da equipe em assuntos ainda não conhecidos. Mais do que isso, *Ruby on Rails* é um *framework* consolidado no desenvolvimento de *software web*, fazendo com que seja mais fácil achar bons profissionais que ficarão encarregados de manter a plataforma e adicionar novas funcionalidades no futuro. Além disso, é um *framework* constantemente atualizado desde 2004, com *bugs* sendo corrigidos e novas funcionalidades sendo adicionadas periodicamente.

Houve outras motivações para a escolha do *Ruby on Rails*, como paradigmas e funcionalidades do próprio *framework*. Sendo amplamente utilizado como *API* por desenvolvedores de *software*, o próprio *Ruby on Rails* possui a opção de utilizar o *framework* como *API*, desabilitando ferramentas que não serão utilizadas, como as *views*, por exemplo. Além disso, é possível usar *scaffolding* neste *framework*, o que nada mais é do que comandos que geram automaticamente partes da aplicação, fazendo com que o desenvolvimento de *software* seja mais rápido e menos suscetível a erros.

Por fim, algo que teve bastante importância na escolha do *Rails* foi o fato do *framework*, assim como a linguagem de programação *Ruby*, ser utilizada nas disciplinas ministradas pelo professor Marco Dimas Gubitoso, fazendo com que os integrantes da equipe já tivessem uma certa familiaridade com as ferramentas. Mais do que isso, sendo o professor um grande conhecedor dessas ferramentas, eventuais dificuldades poderiam ser rapidamente sanadas, fazendo com que a entrega fosse mais rápida e com maior qualidade.

2.1.2 MongoDB

Com os dados referentes às obras do acervo, assim como os dados pessoais dos usuários da plataforma possuindo uma necessidade de armazenamento, utilizou-se o banco de dados *MongoDB* para esse propósito.

Durante o planejamento de quais ferramentas seriam utilizadas na plataforma, surgiu o seguinte questionamento: "seria melhor a utilização de um banco de dados relacional ou um orientado a documentos para o armazenamento dos dados?". Um banco de dados relacional possui todas as informações armazenadas em tabelas, de modo que elas relacionam-se entre si a partir de dados em comum. Mais do que isso, ao construir um banco de dados relacional é preciso a criação de um *schema*, isto é, uma organização formal e rígida de como as tabelas estão organizadas e como os dados se relacionam dentro do banco. Já um banco de dados orientado a documentos encapsula dados em conjuntos em determinados formatos, no caso do *MongoDB*, no formato *JSON (JavaScript Object Notation)*, sem a necessidade de se definir um *schema*, no qual é possível a criação apenas de chaves de interesse de um conjunto.

Dentre as vantagens em se utilizar um banco de dados relacional, pode-se destacar a facilidade em representar e recuperar relacionamentos entre os dados armazenados nas tabelas. Por outro lado, em um banco de dados orientado a documentos, não é necessário definir o *schema* dos dados armazenados, flexibilizando os tipos de dados que é possível armazenar dentro de uma mesma coleção.

Levando-se em consideração que não há muitos relacionamentos entre os dados armazenados na plataforma e considerando o fato de que o acervo possui diversos tipos de itens, como livros, CDs, revistas, slides, entre outros, sendo necessário armazenar esses itens com diferentes campos, decidiu-se então empregar um banco de dados não relacional na plataforma. Sendo o *MongoDB* compatível com *Ruby on Rails* e sendo um dos bancos de dados não relacionais mais utilizados em desenvolvimento de *software*, decidiu-se pela sua utilização.

No entanto, o *Ruby on Rails* utiliza o conceito de *Convention Over Configuration (CoC)* (Hansson, s.d.), sendo um modelo de desenvolvimento de *software* que encoraja a utilização de ferramentas pré-estabelecidas pelos criadores de um *framework*. No caso do *Ruby on Rails*, a convenção é utilizar bancos de dados relacionais. Durante a instalação do *framework* e criação de uma aplicação com ele, lhe é fornecido um banco de dados relacional padrão chamado *SQLite*. Esse modelo, por mais que encoraje a utilização de determinadas ferramentas, também permite que os desenvolvedores configurem a aplicação de acordo com o seu interesse, no entanto, pelo fato de a comunidade de desenvolvedores geralmente aderir à convenção, o suporte e quantidade de dúvidas solucionadas é geralmente

menor quando não se adere. Por esse motivo, a escolha de se utilizar o *MongoDB* como banco de dados gerou maior esforço em buscar informações a respeito de sua instalação e configuração.

2.2 Front-End

Sendo o *front-end* a aplicação que interage diretamente com o usuário, foram considerados elementos de *UX (User Experience)*, que são elementos que tem o papel de possibilitar uma boa usabilidade da plataforma para o usuário, e *UI (User Interface)*, que tem a finalidade de gerar uma interface esteticamente agradável e receptiva para o usuário.

Um exemplo de *UX* foi a escolha de se construir uma *SPA (Single-Page Application)*, isso é, uma aplicação *front-end* em que a medida que o usuário muda de página, apenas parte dessas páginas são renderizadas e outras partes se mantém, uma vez que possuem elementos em comum entre elas (como um menu de navegação, por exemplo). Assim, uma página não precisa ser carregada do zero à medida que um usuário interage com a plataforma, fazendo com que o usuário tenha uma interface muito mais rápida à disposição. Isso foi possível utilizando *Vue.JS*, que será abordado em mais detalhes na próxima subseção.

Como exemplo de *UI*, pode ser citado a escolha de se utilizar uma biblioteca de componentes visuais conhecida como *Element-Plus*, biblioteca na qual os componentes (botões, menus, ícones, *layout* de página, entre outros) seguem aspectos estéticos em comum, possibilitando uma interface mais uniforme e agradável para o usuário.

2.2.1 Vue.JS

Dentre os diferentes tipos de *frameworks* e linguagens de programação para aplicações de *front-end*, decidiu-se pela escolha do *Vue.JS* por uma série de razões, como por exemplo a extensa e bem detalhada documentação do *framework*, facilitando o desenvolvimento e fazendo com que a curva de aprendizado dos integrantes da equipe fosse mais rápida. Mais do que isso, *Vue.JS* é um *framework* constantemente atualizado pelos colaboradores do projeto e é uma ferramenta amplamente utilizada pela comunidade de desenvolvedores, sendo esses dois aspectos muito importantes para a manutenção de código e adição de novas funcionalidades no futuro.

2.3 Ferramentas de Organização

Com o rápido desenvolvimento da computação ao longo dos anos, no qual tanto o *hardware* quanto o *software* ficaram cada vez mais complexos, tornou-se fundamental a utilização de ferramentas para auxiliar e organizar as tarefas dos desenvolvedores responsáveis por tais projetos, fazendo com que estes pudessem ser implementados de maneira mais rápida e com maior qualidade. Com isso em mente, neste trabalho utilizou-se duas importantes ferramentas difundidas entre desenvolvedores de *software*: o *Jira* e o *GitLab*, que serão abordadas em mais detalhes a seguir.

2.3.1 Jira

O *Jira* é um *software web* utilizado sobretudo para organização de tarefas de um projeto, principalmente para aqueles que possuem duas ou mais pessoas colaborando em conjunto para o seu desenvolvimento.

Com o *Jira* foi possível saber qual integrante da equipe estava responsável por determinada tarefa, evitando que uma mesma tarefa fosse feita de maneira desnecessária ao mesmo tempo pelos membros do time. Também foi possível saber o andamento das tarefas, as quais poderiam demandar mais tempo e energia do que o planejado que, ao ficarem em evidência, foi possível saber a real complexidade dessas tarefas e se seria necessário o auxílio de outro membro nesta. Mais do que isso, ao usar o *Jira* foi possível ter uma noção de quantas tarefas ainda tinha-se pela frente, sendo que estas ficavam no *backlog* à espera de um responsável por elas.

Ao utilizar o *Jira*, foi possível aplicar metodologias ágeis, como o conceito de *Sprint* e *Kanban*, práticas de desenvolvimento ágil que tiveram enorme importância na conclusão deste trabalho e que serão melhor exploradas no próximo capítulo.

2.3.2 GitLab

Similar ao tão conhecido *GitHub*, o *GitLab* é uma ferramenta de versionamento de código, servindo sobretudo para a organização do código fonte da plataforma deste trabalho.

O *GitLab* permitiu que os membros da equipe mantivessem o código em um lugar seguro, servindo como um *backup*, caso algo acontecesse com os computadores pessoais utilizados para o desenvolvimento. Também teve o papel de manter o código sempre atualizado, sincronizando o código dos repositórios locais das máquinas para os repositórios remotos. Mais do que isso, também foi um facilitador para revisão de código e resolução de conflitos.

Capítulo 3

Metodologia

A partir da escolha do tema, foi necessário o estudo das tecnologias que seriam empregadas neste trabalho, assim como a forma que o desenvolvimento do projeto seria realizado, utilizando-se principalmente de metodologias ágeis que foram aprendidas durante a graduação.

3.1 Métodos Ágeis

A equipe teve um primeiro contato com metodologia ágil durante a graduação, mais especificamente durante as disciplinas de Técnicas de Programação e Laboratório de Métodos Ágeis. Durante essas disciplinas, foi perceptível os ganhos que tais conceitos tiveram sobre o desenvolvimento dos *softwares* criados, tais como: maior clareza e organização das tarefas do projeto, melhor percepção do tempo que as tarefas levariam, maior rapidez e melhor qualidade de código entregue. Não somente isso, aplicando métodos ágeis, os integrantes da equipe mantinham-se motivados e ao mesmo tempo era possível aprender e trocar conhecimentos com os colegas de equipe. A fim de usufruir de tais ganhos mencionados, foi acordado entre os membros deste trabalho que tais metodologias seriam aplicadas.

Durante o desenvolvimento da plataforma levou-se em consideração o *Agile Manifesto* (*Agile Manifesto* 2001), manifesto com princípios para criação de *softwares* ágeis assinado por diversas pessoas ao redor do mundo. No desenvolvimento também utilizou-se técnicas e ferramentas de metodologia ágil para promover a entrega contínua e o contato constante com o professor Wellington, garantindo que o desenvolvimento da plataforma estivesse de acordo com o esperado pelo cliente.

As entregas de código deste projeto foram realizadas utilizando *Scrum*, sendo um conjunto de técnicas e processos, tendo como principal técnica o desenvolvimento do projeto dividido em ciclos de atividades com reuniões frequentes com o cliente, as chamadas *Sprints*. Neste trabalho foram organizadas *Sprints* com duração de duas semanas na qual os membros da equipe reuniam-se para definir as tarefas que precisariam serem feitas nesse meio tempo. Após certo número de *Sprints*, foram realizadas reuniões com os professores Marco e Wellington para que fosse alinhado o que havia sido feito, assim como ajustar o desenvolvimento para novas ideias ou mudanças de projeto.

As *Sprints* foram realizadas utilizando o *software Jira*. Ao criar uma *Sprint*, é necessário que as tarefas a serem feitas estejam bem definidas. Assim, antes do início da *Sprint* eram criadas uma série de tarefas a serem feitas durante o desenvolvimento do projeto, essas tarefas, chamadas de *tickets*, possuíam um título auto-explicativo, no qual era marcado se essa tarefa seria desenvolvida no *back-end ([BE])*, no *front-end ([FE])* ou em ambos *([BE] + [FE])*. A fim de ficar o mais claro possível, algumas tarefas também continham uma breve descrição sobre o que se tratava.

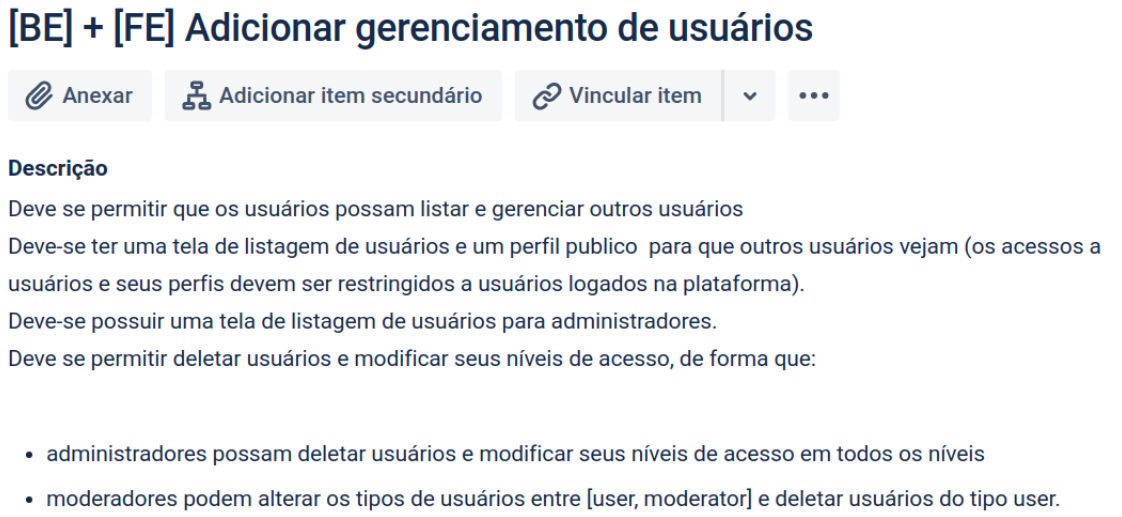


Figura 3.1: *Exemplo de um ticket no Jira*

Essas tarefas eram adicionadas em um primeiro momento à uma lista chamada de *backlog*, sendo um estágio anterior ao início da *Sprint* contendo diversas tarefas relativas ao projeto. Concluído o processo de criação das tarefas, os integrantes da equipe reuniam-se de modo a escolher quais dessas tarefas entrariam na próxima *Sprint* a ser realizada.

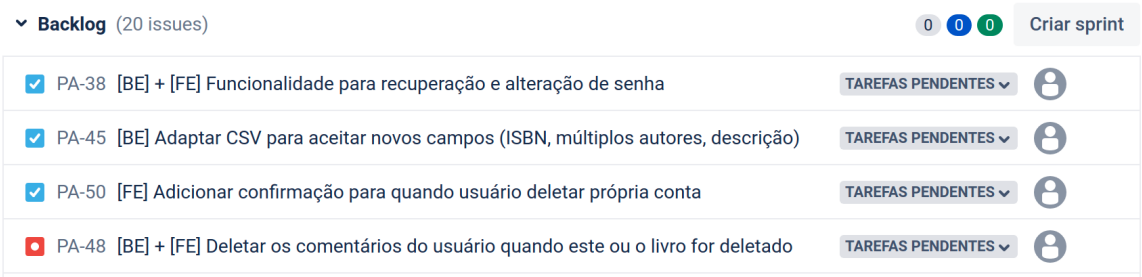


Figura 3.2: *Lista de backlog no Jira durante o desenvolvimento do projeto*

Ao serem escolhidas, as tarefas eram então movidas para a nova *Sprint* que seria realizada. Para realizar as *Sprints* utilizou-se o método *Kanban*, um método que tem o intuito de organizar o fluxo e a organização das tarefas, separando as tarefas em categorias e modificando-as de categoria conforme estas avançam no projeto. Neste projeto as tarefas foram divididas em basicamente quatro categorias: *"To Do"* (a ser feito), *"In Progress"* (em progresso), *"Testing/Code Review"* (testagem e revisão de código) e *"Done"* (completas). Assim, ao iniciar a *Sprint*, as tarefas começavam inicialmente em *"To Do"*. À medida que um integrante da equipe começava a desenvolver a tarefa, essa era movida para *"In Progress"*.

Terminado o desenvolvimento da tarefa, essa era então movida para *"Testing/Code Review"* para que o outro membro da equipe pudesse entender e aprovar o que foi feito. Sendo uma tarefa aprovada na etapa de revisão e testagem, essa era então submetida para *"Done"*.

Figura 3.3: *Quadro Kanban da Sprint "Hello World 10"no Jira*

Outro método ágil empregado no desenvolvimento do projeto, desta vez mais focado na qualidade de código, foi o *Peer Code Review*. Este método tem o intuito de que os integrantes da equipe possam entender, testar e revisar o código de outro integrante do time, de tal modo que erros possam ser percebidos e corrigidos, assim como observar pontos de melhoria no código.

O código fonte deste trabalho está dividido em três repositórios no *GitLab*, sendo um para o *back-end*, outro para o *front-end* e outro para para o *deploy* (implantação do projeto). Em todos os repositórios a *branch* principal é chamada de *master*, enquanto que ao ser desenvolvido o código relativo a um *ticket* no *Jira*, criou-se *branches* que foram padronizadas como feature/PA-<número-do-ticket>_<breve_descrição> (como por exemplo, feature/PA-21_create_authentication), que após revisão, são mescladas na *branch* principal através de uma operação de *merge*.

Capítulo 4

Organização da Plataforma

A plataforma está organizada em quatro tipos de usuários: aqueles que não estão logados, *Users*, *Moderators* e *Administrators*. As funcionalidades que podem ser acessadas por um usuário vai depender do seu nível de acesso.

4.1 Níveis de Acesso

Por se tratar de uma plataforma *web* que apresenta dados para o usuário e permite a interação destes com a plataforma, é necessário que haja uma hierarquia de usuários, cada um possuindo determinados acessos e permissões dentro da plataforma.

Há três tipos de usuários cadastrados na plataforma, sendo eles:

- ***User***: todo usuário recém criado, é o usuário que vai consumir a informação presente na plataforma, vai poder buscar por livros, ver os detalhes, comentar ou avaliar um livro, assim como gerenciar suas informações pessoais.
- ***Moderator***: tem o papel de gerenciar os dados da plataforma, é o usuário que vai poder adicionar e editar livros e categorias na plataforma.
- ***Administrator***: é o usuário com todas as permissões da plataforma. Em conjunto com o *Moderator*, administra os dados dos itens, assim como pode administrar os usuários, podendo mudar a hierarquia de um usuário (de *User* para *Moderator*, por exemplo), assim como deletar usuários da plataforma.

Vale ressaltar que todo usuário registrado na plataforma vai ser do tipo *User*, podendo ser alterado para *Moderator* ou *Administrator* somente com a permissão de um *Administrator*. Também é importante dizer que a hierarquia de usuários é cumulativa, ou seja, o *Moderator* tem permissão de *User* e mais algumas, já o *Administrator* tem permissão tanto de *User* quanto de *Moderator* e mais algumas adicionais. Usuários não cadastrados na plataforma poderão buscar por livros e ver os seus detalhes, mas não poderão comentar ou avaliar as obras.

4.2 Cadastro de Usuário

Ao entrar na página inicial da plataforma, haverá dois botões no canto superior direito, o primeiro para entrar na plataforma, o segundo para fazer o cadastro de usuário.

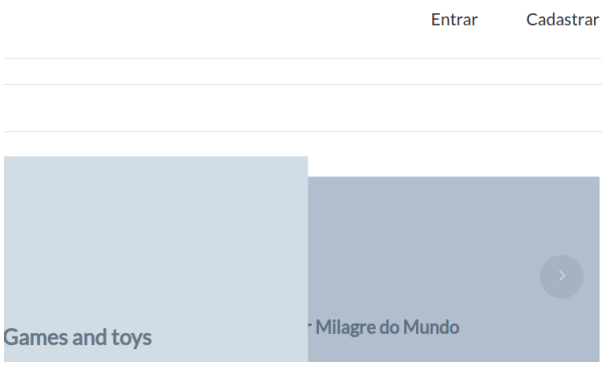


Figura 4.1: *Botões "Entrar"e "Cadastrar"no canto superior direito da página principal*

Caso seja a primeira vez do usuário na plataforma, este deve clicar no botão “Cadastrar”, levando para uma página onde o usuário colocará alguns dados, como *username*, *email*, nome, data de nascimento, gênero e sua nova senha.

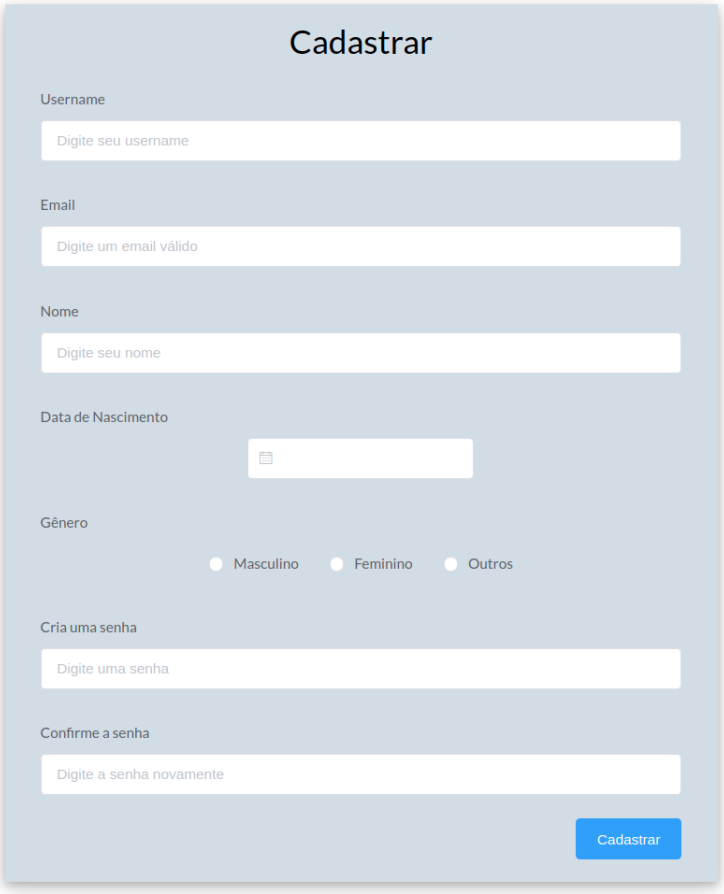


Figura 4.2: *Campos para cadastro de um novo usuário na plataforma*

Após digitar os dados solicitados e clicar em “Cadastrar”, caso os dados não apresentem problemas, como *email* e *username* já cadastrados ou confirmação de senha diferente da senha, o usuário será redirecionado para uma tela de sucesso, solicitando que o usuário abra o *email* que foi cadastrado.

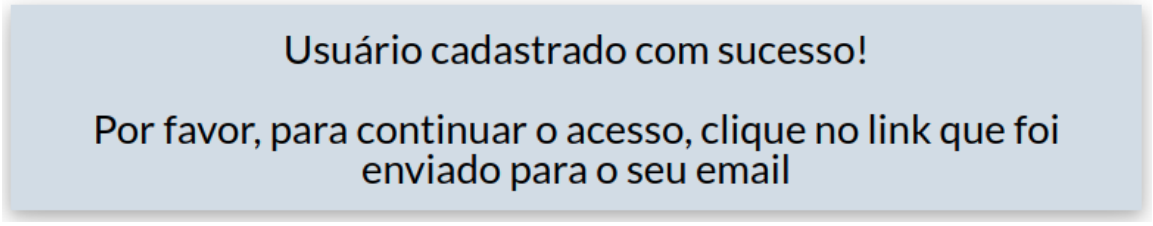


Figura 4.3: *Mensagem informando que o usuário foi cadastrado com sucesso e pedindo para que o usuário confirme seu email na plataforma*

Ao abrir o *email*, o usuário verá uma nova mensagem da plataforma pedindo para que o *email* seja confirmado. Dessa maneira, o endereço de *email* será utilizado para fazer o acesso do usuário na plataforma nos próximos acessos e a confirmação tem o intuito de assegurar a autenticidade do usuário registrado.

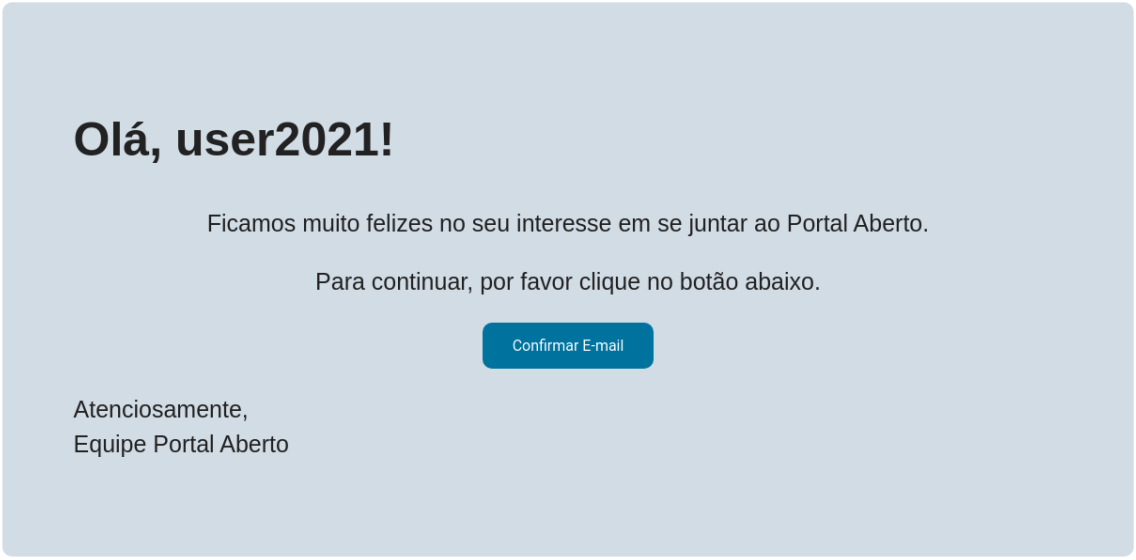


Figura 4.4: *Email recebido pelo usuário user2021 ao cadastrar na plataforma*

Ao clicar no botão, o usuário será redirecionado para uma página afirmando que o *email* foi confirmado. Podendo o usuário, então, logar na plataforma.

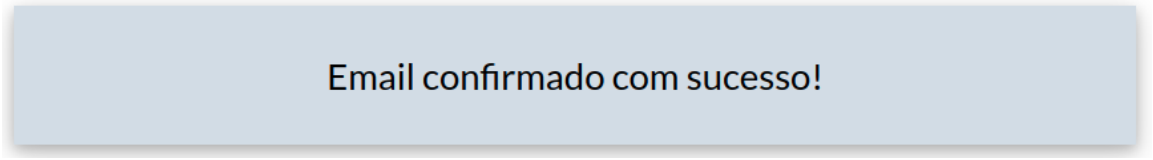


Figura 4.5: *Mensagem informando que o email do usuário foi confirmado com sucesso*

4.3 Acesso para Usuário Cadastrado

Feito o cadastro de usuário como descrito na seção anterior, o usuário então poderá clicar no botão “Entrar” no canto superior direito da tela, sendo solicitado o *email* e senha cadastrados.

Figura 4.6: *Campos de email e senha para acesso à plataforma na página de login*

Sendo o acesso bem sucedido, o usuário será redirecionado para a *homepage* da plataforma.

4.4 Telas e Funcionalidades

Nesta subseção será descrito com mais detalhes e exemplos as telas visíveis para os determinados tipos de usuários, assim como as permissões que cada tipo possui.

4.4.1 Para User

Busca por Livros

Logo ao entrar na página principal da plataforma, o usuário depara-se com todos os livros do acervo dispostos em uma lista infinita, sendo esta uma lista renderizada conforme o usuário desliza a página para baixo. Os itens estão ordenados de maneira decrescente pela avaliação dada pelos usuários a ele.

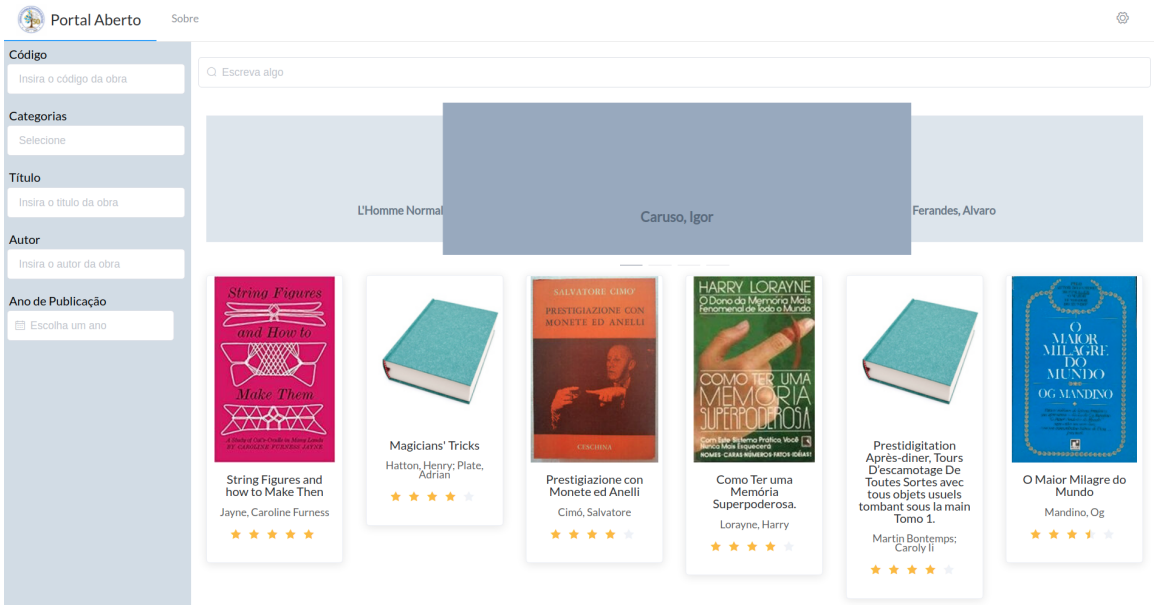


Figura 4.7: *Página principal da plataforma*

Cada item na lista possui a imagem do livro associado, o título da obra, o nome dos autores e a média da avaliação dada pelos usuários da plataforma para determinado livro.

Figura 4.8: *Exemplo de um item na página principal da plataforma*

A página principal também conta com um elemento visual chamado de carrossel, sendo este um elemento que destaca as últimas obras que foram carregadas na plataforma para que o usuário sempre tenha acesso rápido às obras mais recentes do acervo.

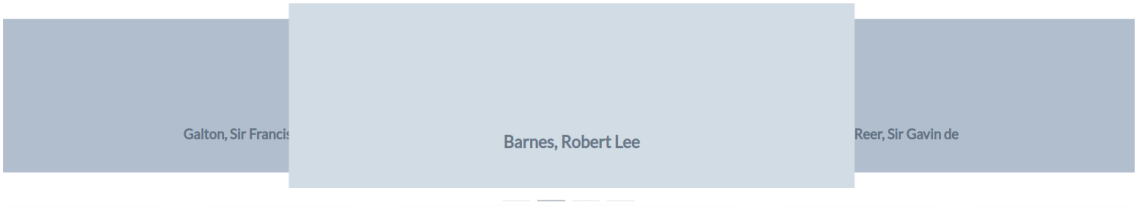


Figura 4.9: *Carrossel com as últimas obras que foram carregadas na plataforma*

Na coluna do lado esquerdo da página principal, o usuário vai poder filtrar os livros de acordo com determinados parâmetros, como código, título, nome de autores, ano de publicação ou até mesmo por categoria, sendo que nesta última, à medida que o usuário escreve o termo, aparecerá as sugestões de categorias que o usuário pode selecionar. Também é possível filtrar os livros por uma busca textual genérica, isto é, ao buscar por um texto, serão retornados livros cujo título ou autores contenham o texto desejado, por exemplo. Esta busca pode ser realizada através da barra de escrita acima do carrossel. Em ambas as formas de busca, à medida que o usuário digita, os livros vão sendo carregados de maneira automática.

Dada a quantidade de livros e a falta de informação sobre alguns deles, é possível que alguns livros não possuam capa ou até mesmo título. No entanto, todos os livros possuem uma identificação única, que é o código deste no acervo.

Detalhes da Obra

Ao clicar em uma obra na página principal, o usuário será redirecionado para os detalhes dessa obra, podendo ver todos os metadados associados à ela, tais como: capa, título, descrição, código no acervo, categorias que o item se enquadra, editora, edição, cidade e ano de publicação.

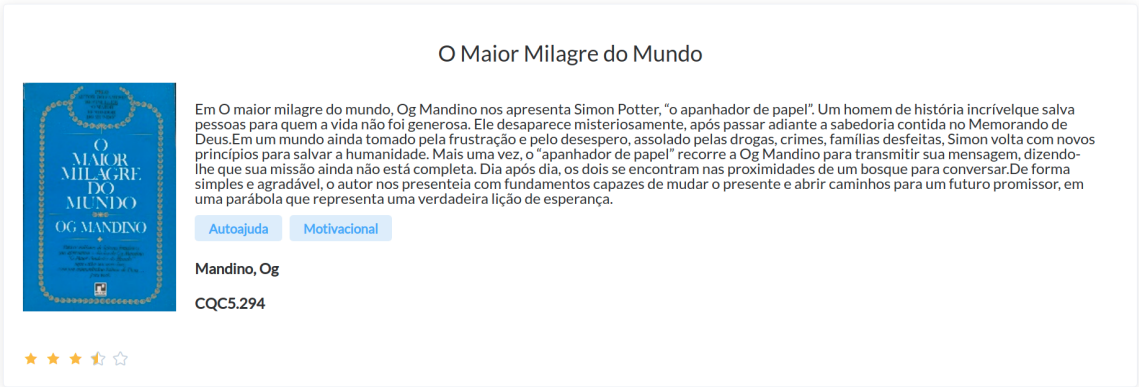


Figura 4.10: *Detalhes de uma obra da plataforma*

Nesta página de detalhes, o usuário vai poder avaliar uma obra de 1 a 5 estrelas clicando sobre elas. Ao clicar, a página irá retornar a média das avaliações daquela obra atualizada. O usuário também vai poder fazer comentários em relação à obra, podendo ver também os comentários de outros usuários da plataforma.

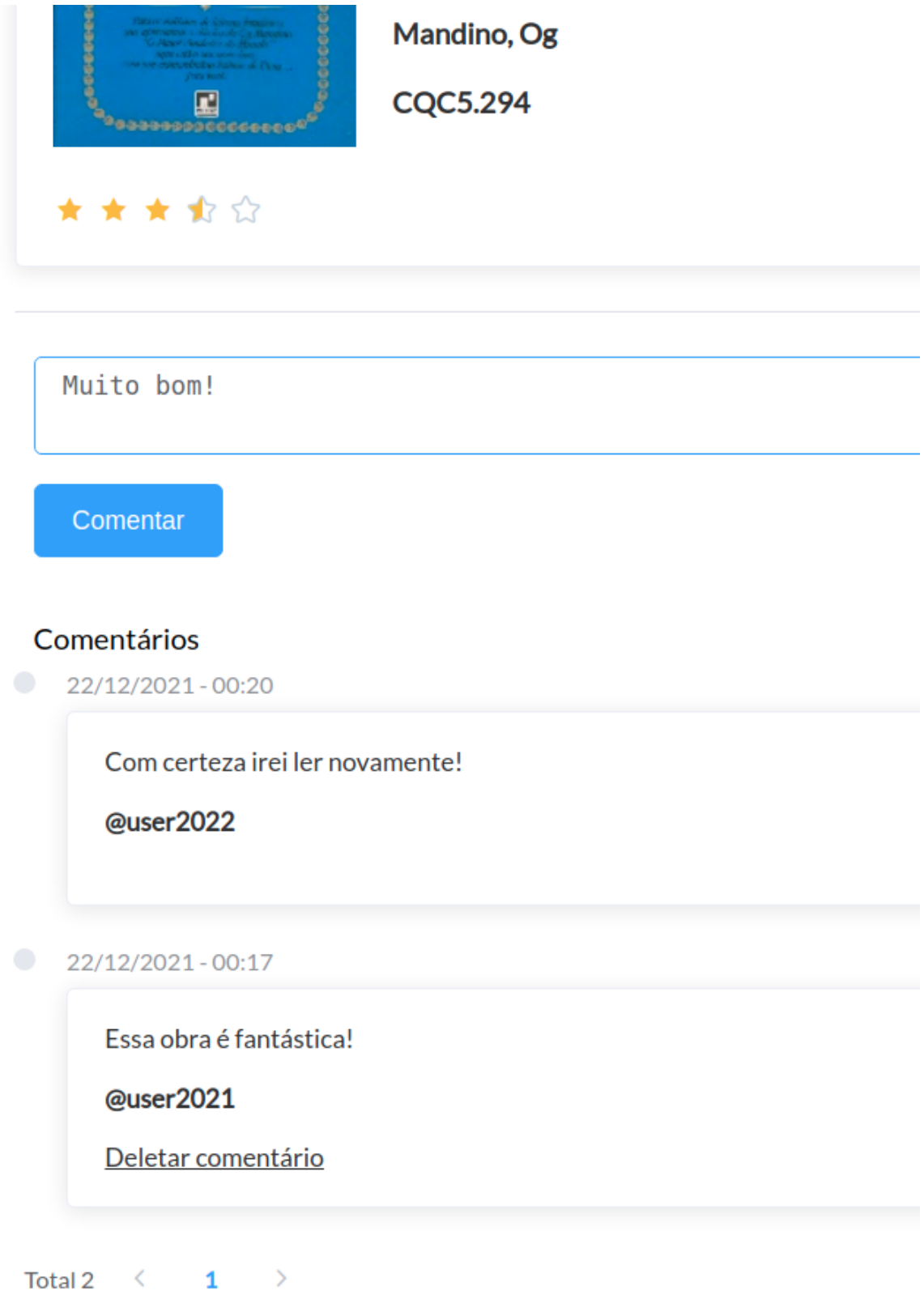


Figura 4.11: *Seção de comentários nos detalhes de um item*

Gerenciamento de Informações e Configurações de Perfil

Ao passar o *mouse* sobre a engrenagem no canto superior direito e clicar em "Dados Pessoais", o usuário vai poder verificar e alterar as informações de perfil, sendo possível alterar o nome, a imagem de perfil, data de nascimento e sexo. Por questões de identificação dos usuários, não é possível alterar o *username* do usuário.

Figura 4.12: *Página contendo os dados pessoais do usuário*

Ao selecionar a opção "Meus Comentários" no menu do lado esquerdo da página, o usuário vai poder ver todos os comentários feitos por ele, assim como quando e em que obra esses comentários foram feitos.

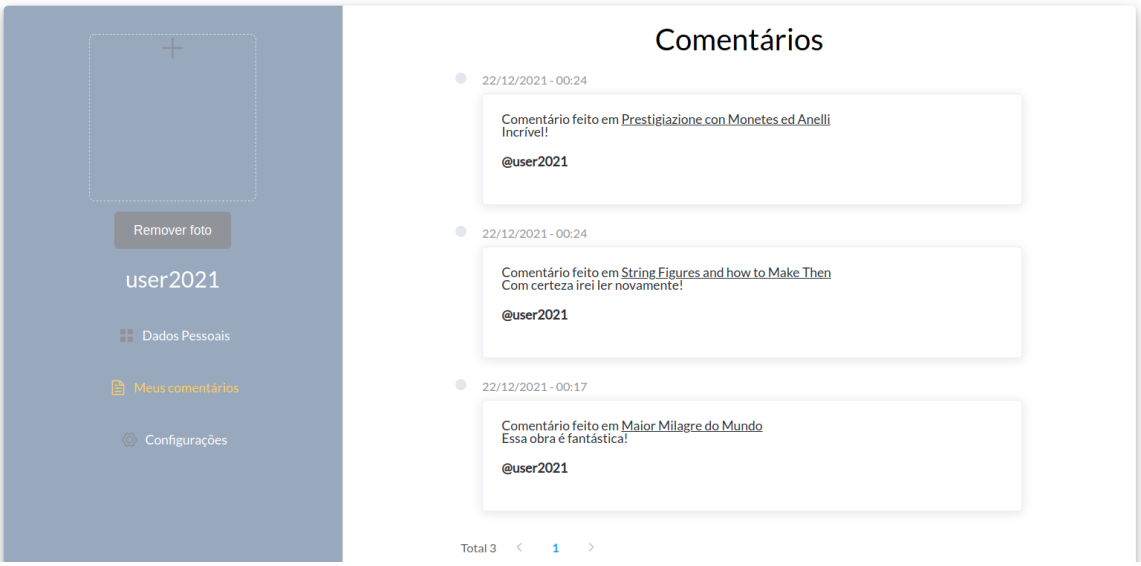


Figura 4.13: *Página contendo os comentários feitos por um usuário*

Por fim, ao clicar em "Configurações" no menu do lado esquerdo, o usuário vai poder deletar seu usuário ou modificar a sua senha.

Figura 4.14: *Página contendo as configurações do usuário*

4.4.2 Para Moderator

Filtrar Livros

Na barra horizontal de navegação, localizada na parte superior da plataforma, o usuário *Moderator* vai ter acesso a um *dropdown* rotulado de “Administração”, ao passar o *mouse* sobre esse *dropdown*, dois itens serão listados: "Livros"e "Categorias".

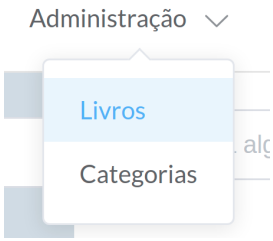


Figura 4.15: *Dropdown "Administração"para o usuário Moderator*

Ao clicar na opção “Livros”, o usuário será redirecionado para uma página contendo todas as obras da plataforma no formato de lista, podendo ser aplicadas as operações de deleção ou edição sobre cada uma das obras listadas. Mais do que isso, é possível filtrar os livros, semelhante ao filtro da *homepage*, mas agora filtrando os livros em um formato de lista.

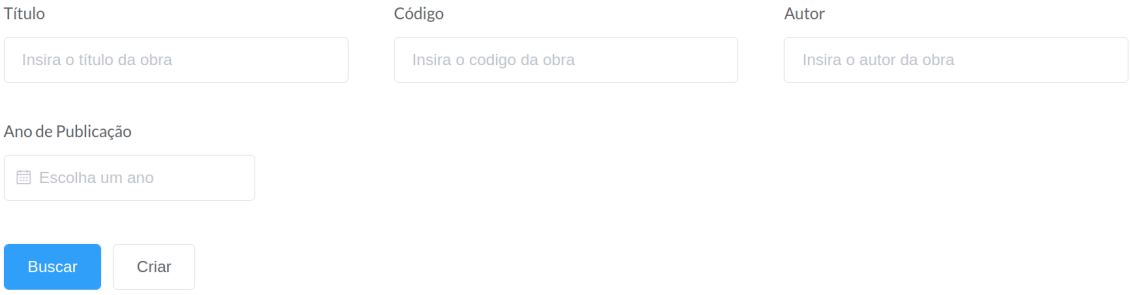


Figura 4.16: *Campos de busca para filtrar os livros que aparecem listados na página*

Código	Título	Autor(es)	Editora	Ano	Operações
CQC5.483	A Ecologia Mental do Espaço Interior: Uma Perspectiva Arquetípica para a Educação Ambiental	Verdade, Marisa Moura		1995	Detalhar Deletar
CQC5.482	Sofrimento e Cura	Gilot, Laura Boggio	Paulinas	1998	Detalhar Deletar
CQC5.481	Psicanálise de Joje (Coleção "O Poder da Mente Humana")	Gratton, Henri	Loyola		Detalhar Deletar

Figura 4.17: *Lista contendo os livros da plataforma*

Editar ou Deletar Livros

Caso o *Moderator* queira realizar a deleção de uma obra da plataforma, basta clicar na opção "Deletar", ao clicar, aparecerá um aviso ao usuário confirmando a deleção de determinada obra.

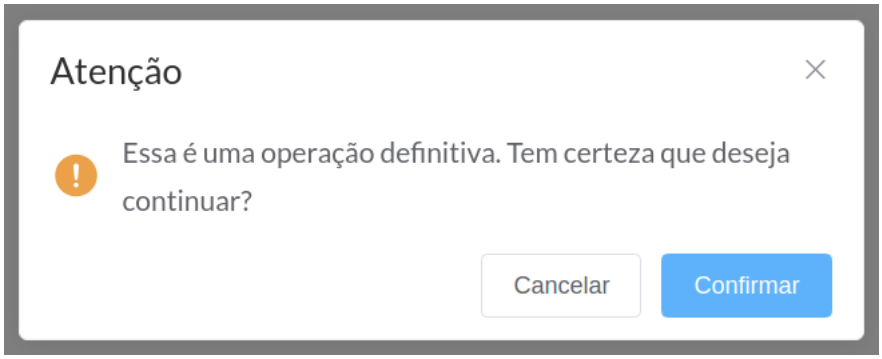


Figura 4.18: *Mensagem de confirmação para deleção de obras*

Caso o *Moderator* queira editar uma obra, basta clicar na opção "Detalhar", sendo então redirecionado para uma página contendo os detalhes da obra na qual terá os campos relativos à obra editáveis, podendo serem substituídos por outros valores. Ao terminar as modificações, o *Moderator* deverá clicar em "Editar" e as alterações realizadas serão salvas.

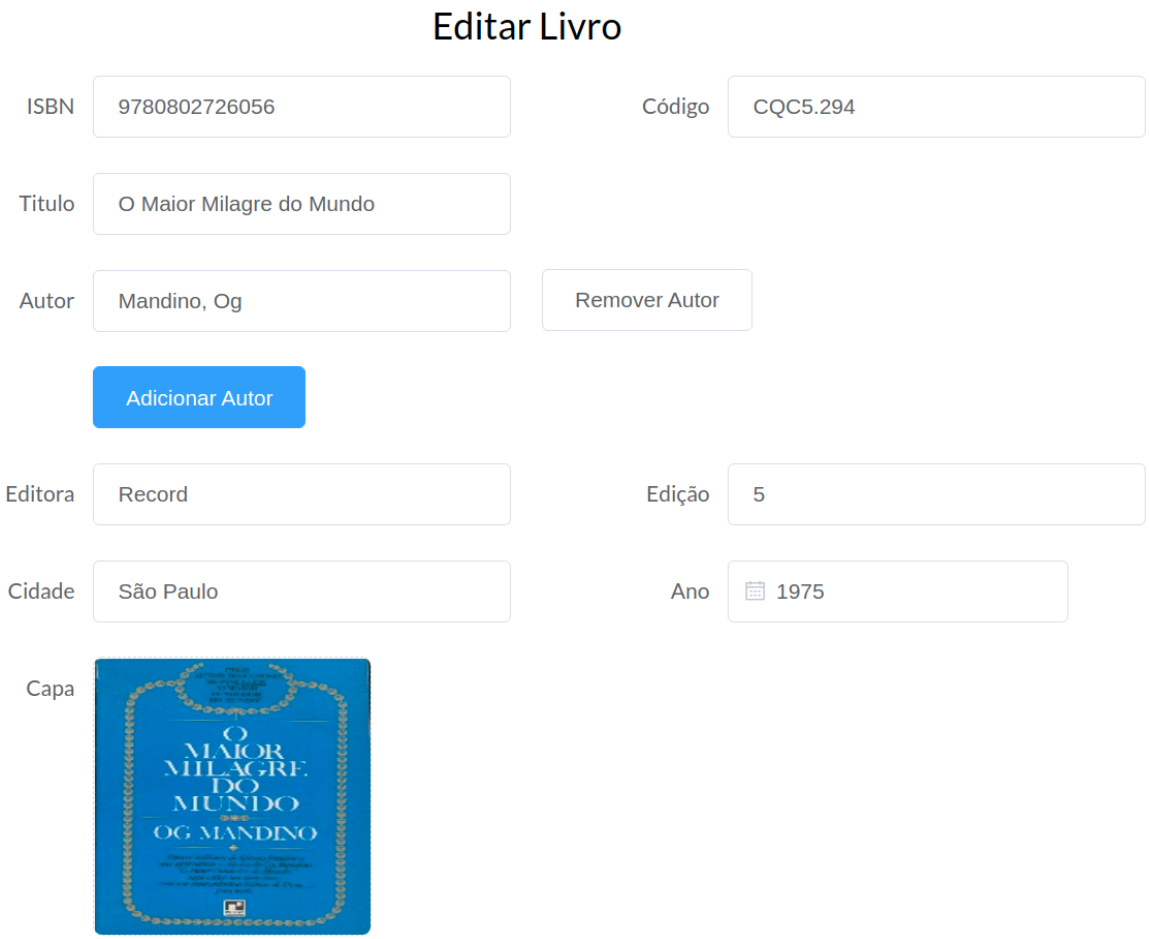


Figura 4.19: *Exemplo de alguns campos de um livro na página de edição*

Carregar Livros

O *Moderator* ao passar o *mouse* sobre o *dropdown* "Administração" e escolher por livros, verá que na página há o botão "Criar". Ao clicar sobre o botão, o usuário será redirecionado para uma página possuindo dois botões: "Inserir dados" e "A partir de um arquivo".

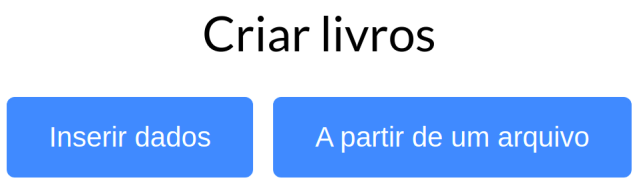


Figura 4.20: *Botões para criar livros individualmente ou a partir de um arquivo*

I. Criar Item Individualmente

Ao escolher o primeiro botão, o usuário será redirecionado a uma página com diversos campos relativos à uma obra em branco, sendo eles: *ISBN (International Standard Book Numbering)*, código, título, autor, editora, edição, cidade, ano, capa, descrição e categoria.

Dado que as obras são muito diversas e algumas podem não conter determinadas informações, como título ou descrição, por exemplo, é possível que os campos sejam deixados em branco, com exceção do campo "código", sendo este um campo único e obrigatório para a correta identificação de uma obra.

A página para criação de um item conta com o campo "*ISBN*", uma identificação internacional para a identificação de obras a partir de um código numérico. Se o usuário *Moderator* possuir este código, basta digitá-lo que a plataforma, utilizando a *API* do *Google Books*, irá perguntar se o usuário deseja que alguns dos campos em branco sejam preenchidos.

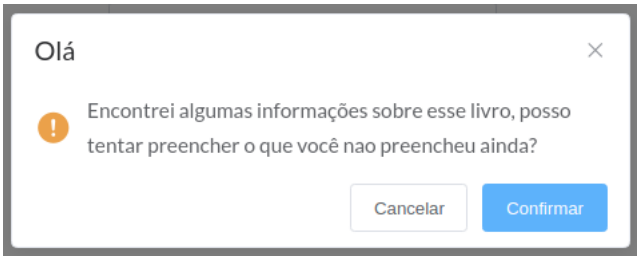


Figura 4.21: *Mensagem sugerindo informações para a obra após o preenchimento do ISBN*

Ao confirmar a sugestão, diversos campos em branco serão preenchidos, não apenas facilitando o trabalho de adicionar um novo item, como também sugerindo informações que talvez o acervo não possua, agregando informações às obras.

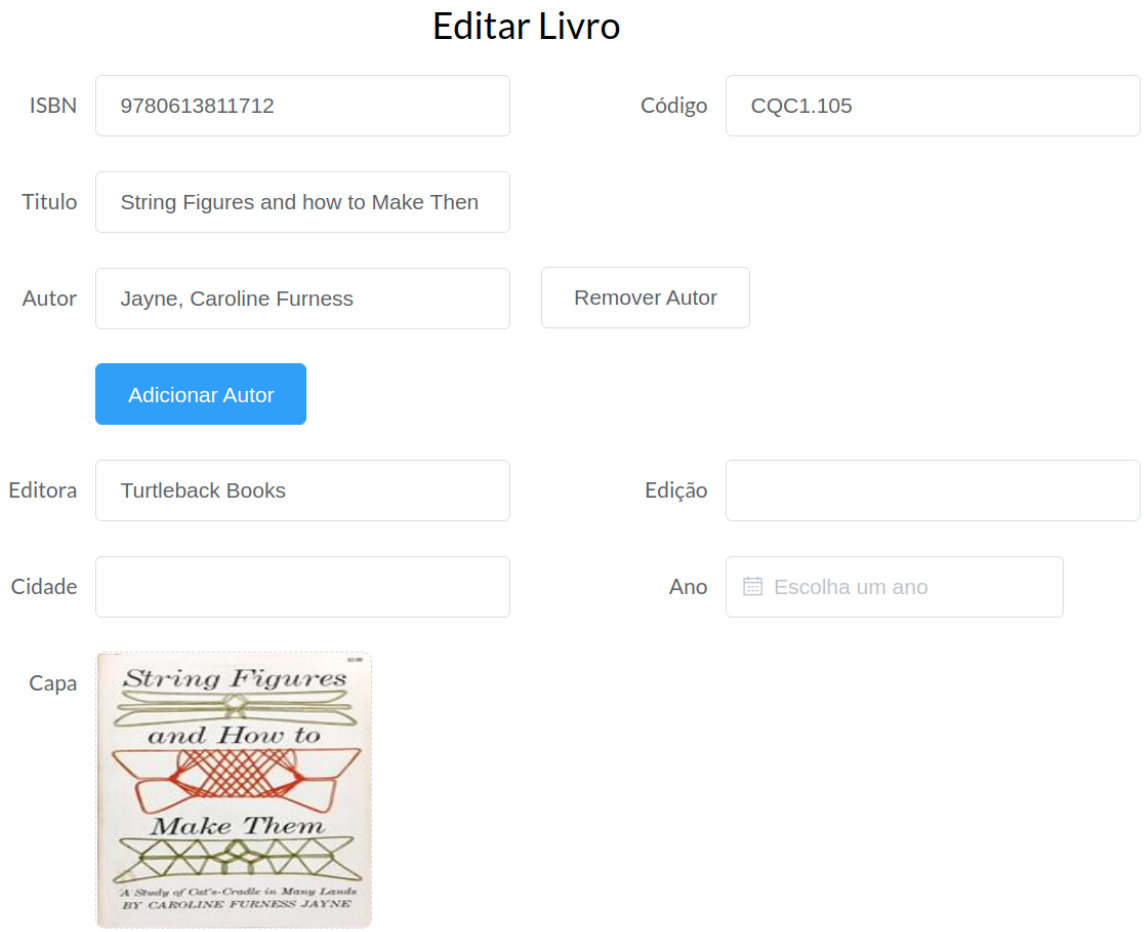


Figura 4.22: *Exemplo de alguns campos preenchidos após a inserção do ISBN pelo usuário*

Também é possível adicionar múltiplos autores a uma mesma obra clicando no botão "Adicionar Autor", que irá criar um novo campo para o cadastro do mesmo. O campo "Categorias" também possui uma particularidade que, ao digitar alguma categoria, como "Ficção", irá aparecer sugestões de acordo com o que foi digitado, na qual o usuário deverá clicar sobre a sugestão para relacionar o livro à categoria escolhida. Ao definir as categorias que os livros poderão ser incluídos, é possível manter a plataforma mais organizada, evitando categorias repetidas ou com erros de digitação.

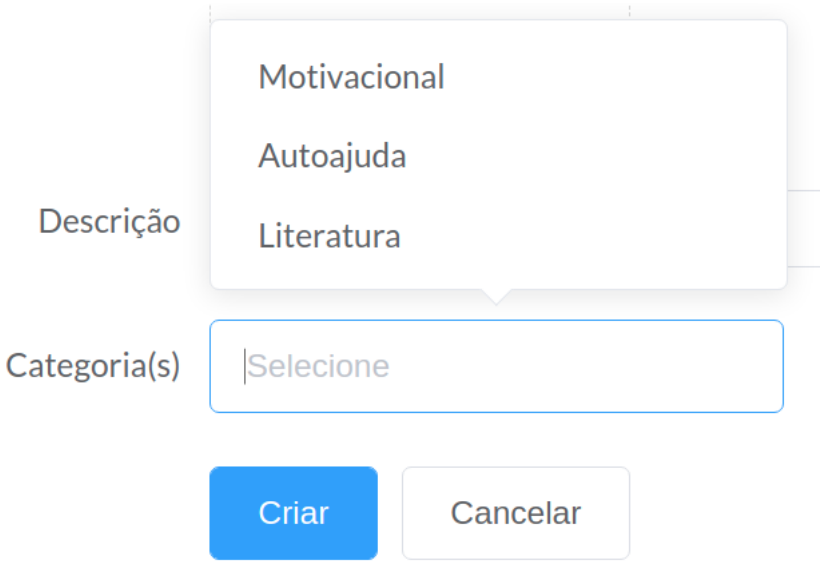


Figura 4.23: *Categorias para o usuário vincular com uma obra*

II. Criar Itens a Partir de um Arquivo

Clicando no segundo botão, "A partir de um arquivo", é possível que diversos livros sejam carregados de uma só vez na plataforma a partir de um arquivo, basta o usuário carregar o documento. Para tanto, os dados dos livros devem estar em um arquivo no formato *CSV (Comma Separated Values)* e possuindo a seguinte header:

codigo isbn titulo autores editora edicao cidade ano descricao

Assim, cada obra vai ocupar uma linha do arquivo, sendo que o campo "autores", que admite múltiplos autores, deve ter os nomes dos autores separados por ponto e vírgula dentro da célula correspondente.

codigo	isbn	titulo	autores	editora	edicao	cidade	ano	descricao
CQC2.1579		Parapsicologia cientifica	Puncernau, Ricardo	Producciones Editoriales	1 ed.		1976	
CQC2.158	9788515004478	A luz que cura: Oração pelos doentes	Sanford, Agnes	Loyola		São Paulo	1985	A luz que cura é
CQC2.1580		How to Develop Clairvoyance	Butler, Walter Ernest	HarperCollins Distribution Services			1968	
CQC2.1581		Em Busca dos Estraterrestres	Kaiser, Andreas Faber	Editora Três			1971	
CQC2.1582	9780116703064	What is Stonehenge?	Atkinson, John Copland	HM Stationery Office			1962	

Figura 4.24: *Exemplo de um documento CSV contendo obras a serem carregadas na plataforma*

Ao submeter o arquivo *CSV* na plataforma e clicar em "Criar", um *feedback* será retornado ao usuário, mostrando quais obras foram criadas com sucesso na plataforma e

quais falharam. Obras possuindo um mesmo código irão retornar com falha pela plataforma, por exemplo.

Código da Obra	Status
CQC1.1	Item criado com sucesso!
CQC1.10	message: Validation of Book failed. summary: The following errors were found: Code is already taken r esolution: Try persisting the document with valid data or remove the validations.
CQC1.100	Item criado com sucesso!
CQC1.101	Item criado com sucesso!
CQC1.102	message: Validation of Book failed. summary: The following errors were found: Code is already taken r esolution: Try persisting the document with valid data or remove the validations.

Figura 4.25: *Feedback após o carregamento de um CSV com as obras, mostrando quais foram bem sucedidas e quais deram erro, assim como o motivo do erro*

Criar Categorias

Para cada obra, pode-se vincular categorias à ela, de modo que o usuário possa buscar livros por uma mesma categoria. Por exemplo, é possível vincular a categoria "Ficção"para determinadas obras e um usuário ao buscar pela categoria “Ficção”, vai ter essas obras como retorno. Para que isso seja possível, é necessário que o usuário *Moderator* passe o *mouse* sobre a seção “Administração” da barra de navegação e selecione “Categorias”.

Ao selecionar a opção “Categorias”, o usuário é redirecionado para uma página semelhante aquela de listagem de livros, na qual vai ser possível ver todas as categorias da plataforma listadas.

Categoria	Operações
Drama	Detalhar Deletar
Religião	Detalhar Deletar
Motivacional	Detalhar Deletar

Figura 4.26: *Lista com alguns exemplos de categorias*

Ao clicar em “Criar” dentro da página, o usuário vai ser redirecionado para uma página na qual será possível criar uma nova categoria, vinculando-se à uma obra e servindo como filtro de busca para os usuários.

Editar ou Deletar Categorias

Na listagem das categorias, o usuário também pode escolher aquelas categorias que deseja editar, basta clicar em “Detalhar” na mesma linha da categoria e então o usuário vai ser redirecionado para uma tela na qual vai poder alterar o nome da categoria. Caso queira deletar, basta que o usuário clique no botão “Deletar” na listagem das categorias ou clique no botão “Deletar” dentro dos detalhes desta. Ao clicar no botão de deleção, uma confirmação será feita para o usuário, podendo cancelar a deleção ou continuar com a operação. É importante que o usuário verifique se a categoria pode ser deletada antes de efetuar a operação, pois uma vez deletada, a categoria será deletada de todos os livros que ela está vinculada.

4.4.3 Para Administrator

Mudar Nível de Permissão de Usuários

Todo usuário ao se cadastrar na plataforma será criado como *User*, sendo que a plataforma sempre terá pelo menos um usuário *Administrator* ativo. Em vista disso, cabe ao *Administrator* mudar o nível de acesso de determinado usuário na plataforma, como de *User* para *Moderator*, por exemplo. Para isso, o usuário *Administrator* deve passar o mouse sobre a seção “Administração” na barra de navegação principal da plataforma e clicar em “Usuários”.

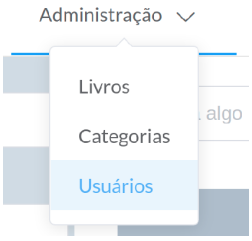


Figura 4.27: *Dropdown "Administração"para o usuário Administrator*

O usuário *Administrator* será então redirecionado para uma página contendo uma lista de todos os usuários da plataforma, sendo possível filtrar pelo nome do usuário, *email* ou nível de permissão (*User*, *Moderator* ou *Administrator*).

Para mudar o nível de permissão de um usuário, o *Administrator* deve localizar este na lista e clicar em “Detalhar”, na qual será redirecionado para uma página contendo as informações de perfil deste usuário, onde terá a opção de alterar seu nível de permissão. Ao final do processo, basta clicar em “Editar” para que o novo nível de acesso passe a valer para aquele usuário.

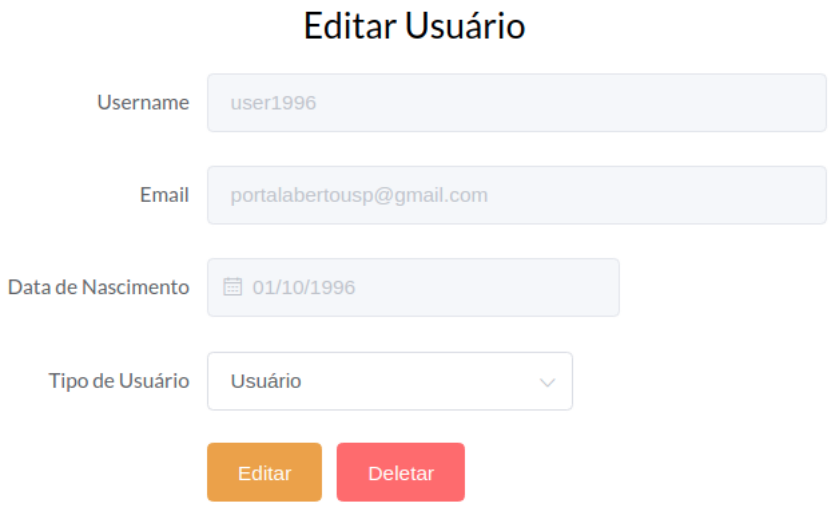


Figura 4.28: *Campos para edição de um usuário pelo Administrator*

Deletar Usuários

Para deletar usuários da plataforma, o *Administrator* pode tanto clicar em "Deletar" na listagem de usuários quanto clicar em "Deletar" nos detalhes do usuário de interesse.

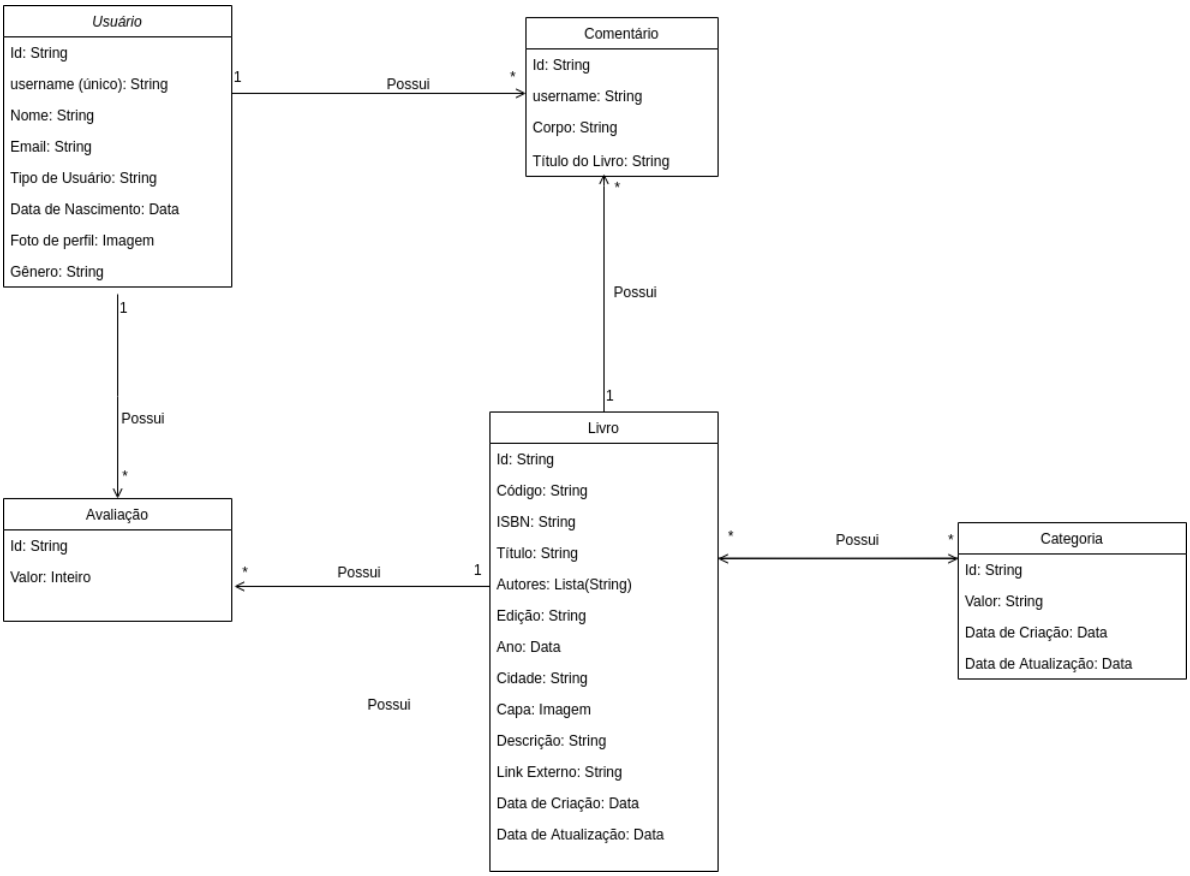
Capítulo 5

Arquitetura

A seguir, se apresentará decisões arquiteturais dos pontos de vista do banco de dados e das *APIs* que provêm acesso aos dados ao *front-end*.

5.1 Banco de Dados

As entidades presentes no banco de dados podem ser vistas na imagem abaixo, assim como os seus relacionamentos:



As *APIs* disponibilizadas pela aplicação, junto com as permissões associadas a elas são

as seguintes:

5.1.1 Livros

- GET /books/id

  Serve para disponibilizar os dados de um livro através do id. Para acessar esse *endpoint* não é necessário estar logado.

- GET /books

  Serve para disponibilizar uma lista paginada de livros que pode ser filtrada por título, código, autores, ano e categorias. Caso mais de um filtro seja solicitado, a filtragem se comporta como uma intersecção, de forma que só sejam listados livros que se adequem aos dois ou mais filtros selecionados. Para acessar esse endpoint não é necessário estar logado.

- POST /books

  Serve para criar um livro. Para isso, deve receber no corpo da chamada um *JSON* com dados válidos do novo livro. Esse *endpoint* é restrito a usuários dos tipos *Moderator* e *Administrator*.

- PUT /books/id

  Serve para atualizar um livro existente através do seu id. Para isso, deve receber no corpo da chamada um *JSON* com os novos dados a serem atribuídos ao livro existente. Esse *endpoint* é restrito a usuários dos tipos *Moderator* e *Administrator*.

- DELETE /books/id

  Serve para deletar um livro existente através do seu id. Esse *endpoint* é restrito a usuários dos tipos *Moderator* e *Administrator*.

- POST /books/csv

  Serve para criar uma lista de livros. Para isso, deve receber em seu corpo um arquivo com dados dos livros em formato *CSV*. Esse *endpoint* é restrito a usuários dos tipos *Moderator* e *Administrator*.

- PUT /books/csv

  Serve para atualizar uma lista existente de livros. Para isso, deve receber em seu corpo um arquivo com os novos dados dos livros existentes no formato *CSV*. Esse *endpoint* é restrito a usuários dos tipos *Moderator* e *Administrator*.

5.1.2 Comentários

- GET /books/id/comments

  Serve para disponibilizar uma lista paginada de comentários associados ao livro com o id id. Para acessar esse endpoint não é necessário estar logado.

- GET /users/id/comments

  Serve para disponibilizar uma lista paginada de comentários feitos pelo usuário com o id id. Para acessar esse *endpoint* não é necessário estar logado.

- POST /books/id/comments

  Serve para criar um comentário feito pelo usuário logado, associado ao livro com o id id. Para acessar esse *endpoint*, basta estar logado.

- DELETE /books/id/comments

  Serve para deletar um comentário associado ao livro com o id id. Esse *endpoint* é restrito ao usuário que fez o comentário ou a usuários dos tipos *Moderator* e *Administrator*.

5.1.3 Avaliações

- PUT /books/id/ratings

  Se não existe uma avaliação feita pelo usuário logado associada ao livro de id id, é criada uma nova avaliação, caso contrário, a avaliação existente é atualizada. Para acessar esse *endpoint*, basta estar logado.

- DELETE /books/id/ratings

  Serve para deletar uma avaliação feita pelo usuário logado, associada ao livro de id id. Para acessar esse *endpoint*, basta estar logado.

5.1.4 Categorias

- GET /books/id/tags

  Serve para disponibilizar uma lista (paginada ou não) de categorias pertencentes ao livro por id. Para acessar esse *endpoint*, não é necessário estar logado.

- GET /tags

  Serve para disponibilizar uma lista (paginada ou não) de categorias disponíveis na plataforma que podem ser filtradas pelo valor. Para acessar esse *endpoint*, não é necessário estar logado.

- GET /tags/id

  Serve para disponibilizar os dados de uma categoria através do id. Para acessar esse *endpoint* não é necessário estar logado.

- POST /tags

  Serve para criar uma nova categoria. Para isso, deve receber no corpo da chamada um *JSON* com dados válidos da nova categoria. Esse *endpoint* é restrito a usuários dos tipos *Moderator* e *Administrator*.

- PUT /tags/id

  Serve para atualizar uma categoria existente através de seu id. Para isso, deve receber no corpo da chamada um *JSON* com os dados atualizados da categoria. Esse *endpoint* é restrito a usuários dos tipos *Moderator* e *Administrator*.

- DELETE /tags/id

  Serve para deletar uma categoria através de seu id. Esse *endpoint* é restrito a usuários dos tipos *Moderator* e *Administrator*.

5.1.5 Uploads

- POST /uploads/images

  Serve para realizar o *upload* de uma imagem que será utilizada como capa de algum livro. Para isso, deve receber a imagem no corpo de sua chamada. Esse *endpoint* é restrito a usuários dos tipos *Moderator* e *Administrator*.

- GET /uploads/images/id

  Serve para disponibilizar uma imagem existente através do seu id para poder ser visualizado no *front-end*.

5.1.6 Usuários

- POST /registrations

  Serve para criar um novo usuário. Para isso, deve receber no corpo da chamada um *JSON* com os dados do novo usuário. Esse *endpoint* é utilizado para se registrar na aplicação, portanto, para acessá-lo, não é necessário estar logado.

- POST /registrations/confirm_email

  Após se criar um novo usuário, a plataforma envia um *email* com um *token* de confirmação para o *email* informado no cadastro. Esse *token* deve ser enviado no corpo dessa chamada para confirmar que o *email* é acessível pelo usuário e ele possa então se logar na plataforma.

- POST /sessions

  Serve para se logar à plataforma. Para isso, deve receber no corpo da chamada um *JSON* com as informações de acesso, a saber, o *email* e a senha. Não é necessário estar logado para acessar o *endpoint*.

- GET /logged_in

  Serve para disponibilizar as informações do usuário logado. Não é necessário estar logado para acessar o *endpoint*, porém caso não esteja, o retorno será do erro de usuário não encontrado. Esse comportamento se deve à necessidade do *front-end* saber se o usuário está logado ou não ao acessar a plataforma.

- DELETE /logout

Serve para se deslogar da plataforma, desta forma é necessário estar logado para acessar o *endpoint*.

- POST /update_profile

  Serve para atualizar os dados do usuário logado. Para isso, deve receber no corpo da chamada um *JSON* com os dados a serem atualizados. É necessário estar logado para acessar esse *endpoint*.

- DELETE /delete_own_user

  Serve para deletar os dados do usuário logado. É necessário estar logado para acessar esse *endpoint*.

- PATCH /recover_password

  Serve para enviar o acesso para recuperação de senha ao *email* do usuário, para isso, deve-se enviar no corpo da chamada o *email* utilizado no cadastro na plataforma. Não é necessário estar logado para acessar esse *endpoint*.

- POST /reset_password

  Serve para atualizar a senha através da recuperação, para isso deve-se enviar no corpo da chamada o *token* de acesso enviado para o *email* do usuário, além da nova senha. Não é necessário estar logado para acessar o *endpoint*.

- GET /users

  Serve para disponibilizar uma lista paginada dos usuários da plataforma, podendo ser filtrada por *email*, *username* e tipo de usuário. Para acessar esse *endpoint*, é necessário estar logado na plataforma.

- GET /users/id

  Serve para listar os dados do usuário com id id. Para acessar esse *endpoint*, é necessário estar logado na plataforma.

- PUT /users/id

  Serve para atualizar o tipo do usuário de id id, para isso, deve receber o novo tipo de usuário no corpo da chamada. Esse *endpoint* é restrito a usuários dos tipos *Moderator* e *Administrator*. Os usuários do tipo *Administrator* não possuem restrição quanto aos tipos de usuário que podem alterar.

- DELETE /users/id

  Serve para deletar o usuário de id id. Esse endpoint é restrito a usuários do tipo *Administrator*. Os usuários do tipo *Administrator* não possuem restrição quanto aos tipos de usuário que podem deletar.

Capítulo 6

Resultados

6.1 Visão dos Desenvolvedores

De maneira geral, as funcionalidades solicitadas pelo cliente foram entregues no tempo planejado, pôde-se ter contato com tecnologias novas durante o processo, além de se desenvolver a aplicação de maneira que permita a expansão do projeto por outros alunos e docentes. Houve algumas funcionalidades propostas pelos desenvolvedores que não foram implementadas, a saber:

- Interação entre os comentários dos usuários, onde um usuário não apenas poderia comentar nos livros, porém poderia responder comentários de outros usuários e avaliar os comentários, o que iria afetar a ordem de exibição dos comentários, sendo ordenados de maneira decrescente pela avaliação.

- Implementação de localização, onde a plataforma poderia ser visualizada em inglês ou outro idioma, aumentando assim, o alcance e acessibilidade do acervo.

- Implementação do conceito de "lista de favoritos" para cada usuário, onde este poderia adicionar suas obras favoritas, o que iria influenciar a ordem de listagem das obras na página inicial, de modo que os livros que pudessem ser mais interessantes para ele iriam aparecer primeiro na listagem.

Essas funcionalidades são exemplos do que pode se dar de continuidade no projeto no futuro.

6.2 Visão do Cliente

O texto a seguir é uma carta escrita pelo cliente (Prof. Dr. Wellington Zangari) descrevendo suas impressões sobre a versão final do Portal Aberto e do processo de desenvolvimento deste.

São Paulo, 15 de dezembro de 2021.

Tem essa carta a intenção de informar minha experiência e minha avaliação relacionada ao processo de construção do que estamos chamando de Portal Aberto, realizado pelos alunos Renan Costa Laiz e Renan Tiago dos Santos Silva.

A ideia do projeto surgiu de uma conversa entre mim e o Prof. Marco Dimas Gubitoso (Gubi para nós) que, infelizmente, está em cuidados paliativos, vitimado que foi por um câncer cerebral. Na conversa eu o informava de minha alegria pela conquista de um grande acervo de livros e revistas na especialidade do laboratório que coordeno no Instituto de Psicologia, laboratório que mantém grupos de estudo nos quais o Prof. Marco participava até recentemente. Mas também comentava que, por se tratar de acervo antigo, constituído desde a década de 1970, toda sua catalogação estava em papel, em centenas de folhas com codificação rudimentar e impossível capacidade de realização de buscas. Naquele momento eu já havia recrutado participantes de meu laboratório para digitarem os dados em uma tabela Excel. No mesmo momento, o Prof. Marco disse que poderia ajudar, melhorando as possibilidades permitidas no Excel. Pouco tempo depois entrou em contato informando que havia conversado com dois alunos que poderiam fazer de minha dificuldade um trabalho de conclusão de curso no qual buscariam por soluções dessa dificuldade.

No primeiro briefing que passei aos dois alunos, minha expectativa era apenas de ter um sistema computadorizado que me permitisse fazer buscas por autores ou títulos, além da inserção de novos itens e que pudesse ser utilizado remotamente por visitantes da internet. Os alunos mantiveram, a partir de então, conversas tanto com o Prof. Marco quanto comigo e, ao longo de alguns meses, o sistema foi evoluindo ao ponto que chegou atualmente.

Quero fazer constar que o que Portal Aberto foi sendo desenvolvido muito em função de sugestões e soluções que os próprios alunos trouxeram. Acompanhei várias reuniões nas quais os alunos trouxeram sugestões que os levaram, com a ajuda do Prof. Marco, a estudarem novas ferramentas e aplicações que permitiriam sua exequibilidade. Compreendo, portanto, que ao longo do processo, os alunos foram obrigados a desenvolver novas habilidades. Dou alguns exemplos de funcionalidades que eram imprevistas para mim e que foram implementadas dada as sugestões dos alunos. O Portal permite, para muito além do que eu apresentei no briefing que passei, a operação por múltiplos usuários, com diferentes status de funções e tarefas; permite que usuários possam apresentar comentários e realizar discussões entorno dos títulos de livros; permite que as capas dos livros sejam inseridas, usufruindo de uma interface atraente e amigável; permite que os dados de um livro sejam inseridos de múltiplas formas, inclusive por meio da inserção do número do ISBN, o que reduz consideravelmente o tempo para o trabalho de inserção e maior acurácia no processo; permite praticidade na realização de backup; permite simplicidade no upload e atualização do banco de dados; permite ampliação das funcionalidades no futuro. São apenas alguns exemplos do quanto o produto final superou, em muito, minha solicitação inicial.

Para finalizar, quero dizer de minha satisfação com o Portal Aberto e, como professor, também preocupado com a questão da formação técnico-cientifica, permito-me dizer que os alunos se empenharam muito em trazer aportes importantes para o produto, mesmo que isso lhes tenha custado ainda mais trabalho. Nesse sentido, considero que o desempenho

dos alunos nesse trabalho me surpreendeu e me satisfez plenamente.

Atenciosamente,

Wellington Zangari.

Capítulo 7

Deploy

Para realizar o *deploy* da aplicação, optou-se por utilizar o conceito de *containers*, que são uma forma de encapsular aplicações com todas as suas dependências de maneira isolada de outros processos. Isso permite que os desenvolvedores possam realizar o *deploy* da aplicação em qualquer servidor *Linux* sem se preocupar em instalar quaisquer dependências. Para implementar os *containers*, foi utilizado o *Docker*, que é a plataforma responsável por executar cada *container*, junto com o *Docker Compose*, que é responsável por organizar *containers* que devem ser executados de maneira conjunta.

Desta forma, a divisão dos *containers* ficou da seguinte forma: um *container* para a aplicação *front-end*, um para a aplicação *back-end* e um para o banco de dados.

7.1 Como Executar a Aplicação Localmente

Para executar a aplicação localmente, é necessário primeiro ter instalado os programas *Git*, *Docker* e *Docker Compose*, além de garantir que as portas 80, 3000 e 27017 estejam disponíveis. Com esses programas instalados, deve-se acessar a página do *GitLab* do Portal Aberto através do endereço gitlab.com/portal-aberto. Ao acessar essa página, se é apresentado com os repositórios do Portal, como pode ser visto abaixo:

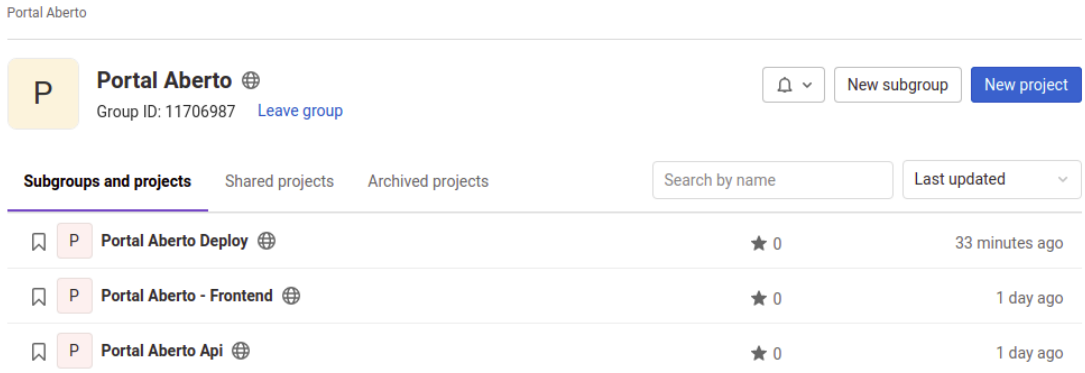


Figura 7.1

Nela, pode-se visualizar todos os repositórios do projeto. Para executar a aplicação, será necessário que todos sejam baixados no mesmo diretório do computador. Para baixar cada um dos repositórios, é necessário acessar a página de cada um deles, clicando no nome na lista apresentada acima, sendo redirecionado para a página do repositório desejado, como pode ser visto no exemplo abaixo:

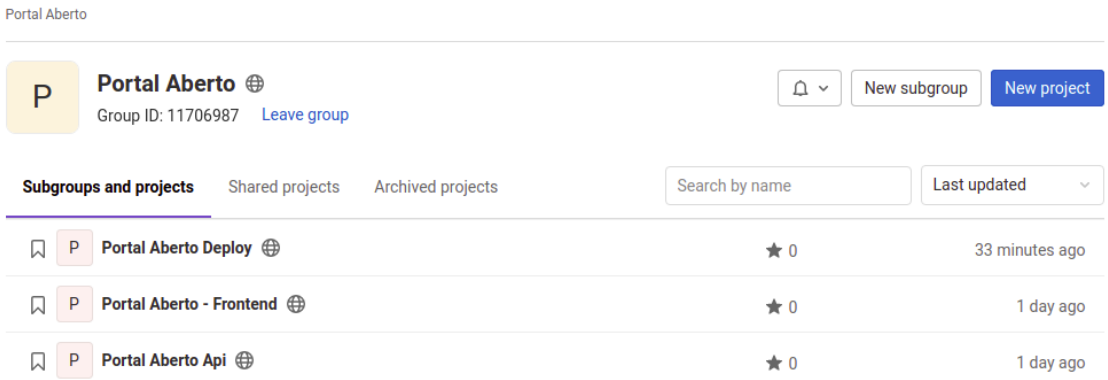


Figura 7.2

Nela, deve-se clicar no botão *clone* e copiar o endereço *HTTPS*, como visto na imagem abaixo:

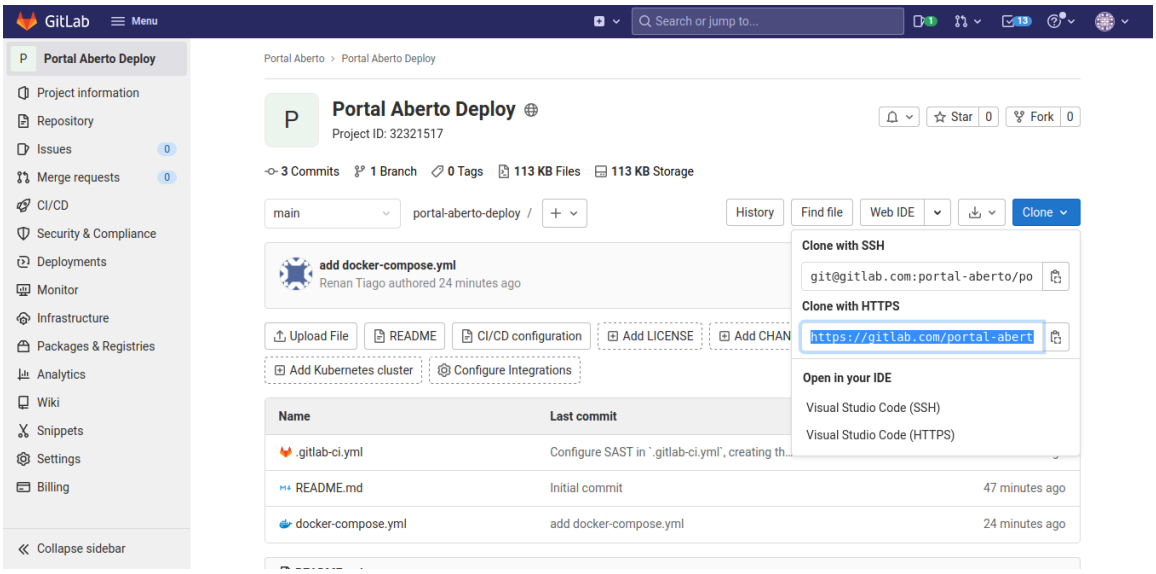


Figura 7.3

Com esse link, deve-se executar o comando `git clone <link>` no diretório desejado, clonando-o para seu computador, conforme visto na imagem abaixo:

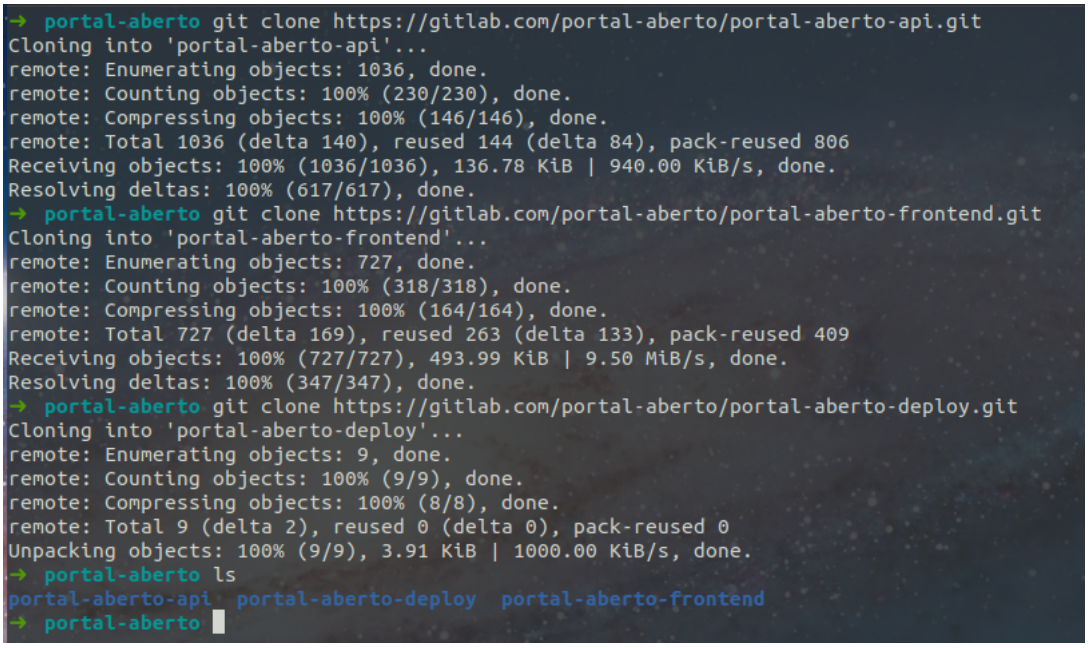


Figura 7.4

Deve se então acessar o diretório portal-aberto-deploy e executar o comando docker-compose up. Com isso, a aplicação estará disponível no endereço *localhost* do computador. Dependendo das configurações da máquina, é necessário a utilização do comando *sudo*.

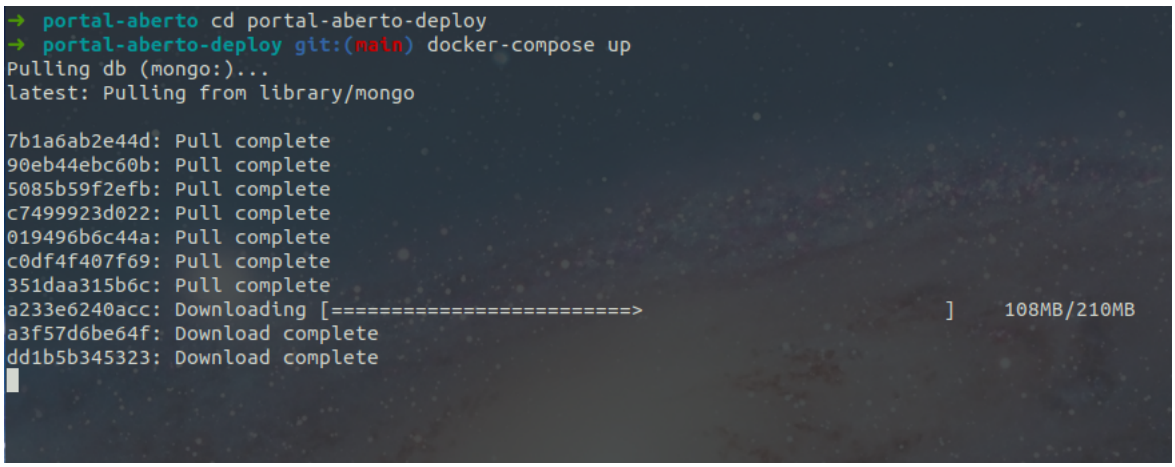


Figura 7.5

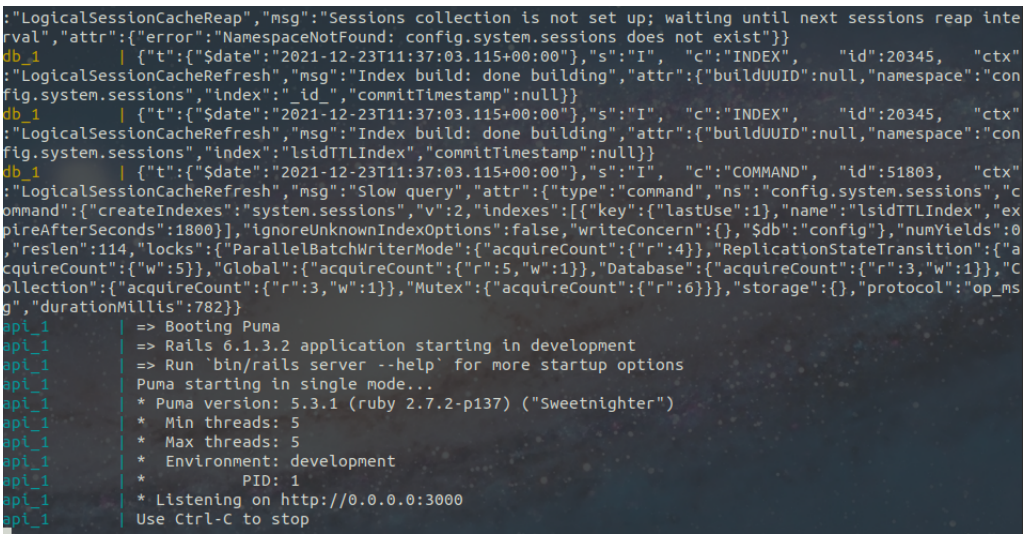


Figura 7.6: *Aplicação sendo executada*

Para que os usuários possam se cadastrar através da plataforma, é necessário que um *email* esteja cadastrado na aplicação do back-end, vai ser esse *email* que irá ser utilizado para mandar o *email* de confirmação para o usuário quando este se cadastra na plataforma.

No diretório da aplicação do *back-end*, as alterações devem ser feitas no arquivo config/environments/development.rb, no qual deve ser incluído o protocolo de *email* que vai ser utilizado e as configuraçõs necessárias, incluindo o endereço de *email* e senha deste.

```
config.action_mailer.delivery_method = :smtp
config.action_mailer.smtp_settings = {
  address: 'smtp.gmail.com',
  port: 587,
  domain: 'localhost:3000',
  user_name: 'portalabertousp@gmail.com',
  password: '',
  authentication: 'plain',
  enable_starttls_auto: true
}
```

Figura 7.7: *Exemplo da configuração de email na plataforma no arquivo development.rb omitindo a senha do mesmo*

No caso do *Gmail*, é possível a utilização de um *App Password*, um código que é criado que pode ser utilizado ao invés de se utilizar a própria senha do *email*. Esse código deve ser criado nas configurações da conta, sendo gerado apenas uma vez, sendo importante que o administrador da plataforma o guarde em um lugar seguro.

Com a aplicação sendo executada, deve-se criar o primeiro usuário através da página de cadastro. Pelo fato de o usuário ser criado como o tipo *User*, o próximo passo depois

da criação é alterar o usuário no banco de dados para que seja um *Administrador* e tenha todas as permissões. Para isso, deve-se acessar o banco de dados através dos seguintes comandos utilizando o ID do *container* do *MongoDB*:

```
→ portal-aberto-deploy git:(main) docker container ls
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS     NAMES
c9668046ec57   portal-aberto-deploy_frontend   "serve -s dist -l 80"    4 minutes ago   Up 4 minutes             portal-aberto-deploy_frontend_1
5643a7dd348f   portal-aberto-deploy_api        "bash -c 'rm -f tmp/…"   4 minutes ago   Up 4 minutes             portal-aberto-deploy_api_1
f63a19a13a48   mongo                          "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes             portal-aberto-deploy_db_1
→ portal-aberto-deploy git:(main) docker exec -it f63a19a13a48 mongo
```

Figura 7.8

Após isso, deve-se atualizar os usuários existentes para o tipo *Administrator*.

```
> use portal_aberto_api_development
switched to db portal_aberto_api_development
> db.users.updateMany({}, {$set:{"user_type": "admin"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Figura 7.9

Caso a configuração do email não tenha sido feita, a plataforma irá falhar ao tentar criar um novo usuário, porém para garantir que o usuário funcione, basta alterar o valor do atributo `email_confirmed` para `true` no banco de dados.

```
> db.users.updateMany({}, {$set: {email_confirmed: true}})
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
```

Figura 7.10

Com isso, a aplicação vai ser acessível localmente, e se terá um usuário *Administrator*. Em um primeiro momento a plataforma deve aparecer sem os livros, que devem ser adicionados na seção de Administração por um *Administrator* ou *Moderator*.

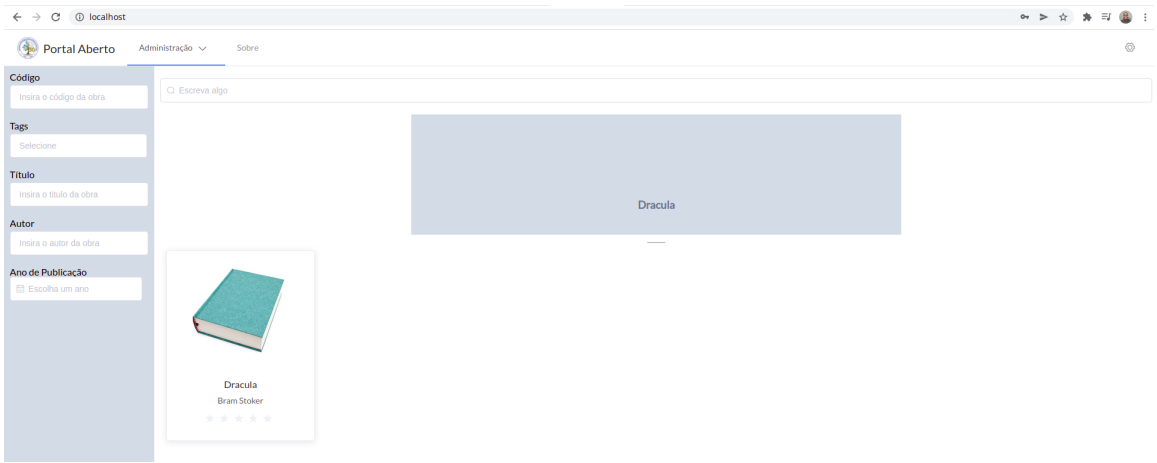


Figura 7.11

Capítulo 8

Conclusão

Durante o desenvolvimento deste projeto, pôde-se observar que o desenvolvimento e entrega de um sistema de *software* envolve diversas disciplinas e habilidades além da capacidade de se escrever código, e para que o projeto seja bem sucedido, é necessário que haja método e organização.

O primeiro aspecto que se faz necessário é que haja comunicação clara e constante entre os responsáveis pelo desenvolvimento e o cliente interessado na aplicação. Nesse ponto, é importante que o desenvolvedor saiba expressar suas ideias sem ser desnecessariamente técnico, uma vez que nem todo cliente terá o conhecimento para tal. Também é necessário que o cliente saiba expressar o que deseja da aplicação para que não haja desencontros em relação ao que é entregue e o que é esperado. A comunicação entre a dupla de desenvolvimento, o professor Marco e o professor Wellington foi tão eficaz que não apenas permitiu que os desenvolvedores entregassem o que era crucial para o cliente, mas propusessem *features* que pudessem ser úteis, expandindo ainda mais o uso da plataforma.

Outro ponto extremamente importante para além do código é o gerenciamento do projeto em si. Utilizar ferramentas, e metodologias de desenvolvimento ágil existentes como *Kanban* e *Scrum* foi importante para garantir que cada integrante do time sempre tivesse algo a fazer, além de apresentar constantemente evoluções do projeto para o cliente e garantir que as expectativas estavam sendo alcançadas (ou reavaliadas quando necessário).

Ao desenvolver uma aplicação, há diversas tecnologias que se pode escolher para utilizar na implementação. A escolha de uma opção em detrimento de outra pode determinar o tempo necessário para entregar, a satisfação do cliente, ou mesmo a viabilidade do projeto como um todo. Como descrito em *"No Silver Bullet - Essence and Accident in Software Engineering" - Frederick P. Brooks, Jr.* (Brooks-Jr., 1987), não há uma única forma de se desenvolver, tecnologia ou técnica que em si garanta melhora na produtividade, confiabilidade e simplicidade. Desta forma, é importante que o time de desenvolvimento seja aberto a diversas possibilidades e as avalie de maneira racional a fim de escolher as melhores para o contexto do projeto e das pessoas envolvidas nele. Ao longo do desenvolvimento da plataforma, pôde-se observar vantagens e desvantagens das escolhas tomadas

no início do projeto, e por isso o tempo levado para estudar e escolher as tecnologias do projeto foi necessário.

O fato de o projeto ser feito em equipe e pensado para expansão futura por outros desenvolvedores torna aspectos do código como facilidade de manutenção e cobertura de testes bastante importantes. Tão importante quanto escrever um código performático, é garantir que outros desenvolvedores possam ler e compreender com facilidade o que está sendo executado. Também contribuindo para a facilidade de manutenção, uma cobertura considerável de testes é importante para que outros desenvolvedores se sintam confortáveis em fazer alterações na aplicação no futuro sem se preocupar que a aplicação deixe de funcionar de maneira inesperada em algum momento.

O que se pôde observar ao longo da evolução do projeto é que não apenas essas habilidades e disciplinas foram necessárias para uma boa condução, mas elas também evoluíram e seus efeitos foram mais visíveis ao longo do tempo. Com o passar do tempo no projeto, a comunicação entre desenvolvedores e cliente se tornava mais clara e objetiva, a organização e gerenciamento do projeto se adaptava melhor ao time, as vantagens e desvantagens das escolhas tecnológicas se tornaram mais aparentes, e a preocupação com a qualidade do código se fazia mais necessária com o aumento da complexidade da aplicação.

Referências

[*Agile Manifesto* 2001] *Agile Manifesto*. 2001. URL: https://agilemanifesto.org/ (citado na pg. 9).

[Brooks-Jr. 1987] Frederick P. Brooks-Jr. *No Silver Bullet - Essence and Accidents of Software Engineering*. 1987. URL: http://www.cs.unc.edu/techreports/86-020.pdf (citado na pg. 47).

[Fielding 2000] Roy Thomas Fielding. "Architectural Styles and the Design of Network-based Software Architectures". Tese de dout. University of California, 2000. URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (citado na pg. 4).

[Hansson s.d.] David Heinemeier Hansson. *The Rails Doctrine - Convention over Configuration*. URL: https://rubyonrails.org/doctrine#convention-over-configuration (citado na pg. 5).