UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

LUIZ GUSTAVO BOURSCHEIT

APLICAÇÃO PARA GERENCIAMENTO DE EMPRÉSTIMOS DE EQUIPAMENTOS ELETRÔNICOS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2020

LUIZ GUSTAVO BOURSCHEIT

APLICAÇÃO PARA GERENCIAMENTO DE EMPRÉSTIMOS DE EQUIPAMENTOS ELETRÔNICOS

Trabalho de conclusão de curso, apresentado ao curso de especialização em Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para a obtenção do título de especialista.

Orientador: Vinícius Pegorini

PATO BRANCO
2020

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

TERMO DE APROVAÇÃO

APLICAÇÃO PARA GERENCIAMENTO DE EMPRÉSTIMOS DE EQUIPAMENTOS ELETRÔNICOS

por

LUIZ GUSTAVO BOURSCHEIT

Este trabalho de conclusão de curso foi apresentado em 09 de março de 2020, como requisito parcial para a obtenção do título de especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Vinicius Pegorini (orientador), Andreia Scariot Beulke e Beatriz Terezinha Borsoi, membros de banca. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Vinicius Pegorini
Prof. Orientador (UTFPR)

_____
Andreia Scariot Beulke
(UTFPR)

_____
Beatriz Terezinha Borsoi
(UTFPR)

_____
Vinicius Pegorini
Coordenador do curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

As instituições, em geral, precisam organizar suas informações para o obter o melhor desenvolvimento de suas atividades do dia-a-dia, portanto a utilização de um sistema que os auxilie nessa tarefa se faz necessário. Somado a isso, pelo fato de permitirem a universalização da informação, os sistemas projetados para serem utilizados na web ganharam muita força nos últimos anos. Tendo em vista esses fatores, o presente trabalho foi desenvolvido com o objetivo de implementar um sistema web para controle e acompanhamento de empréstimos de equipamentos do Departamento Acadêmico de Elétrica da UTFPR, câmpus Pato Branco. O sistema desenvolvido traz benefícios ao referido departamento: além de uma gestão mais organizada, destaca-se a segurança que o sistema trará para o controle de empréstimo de equipamentos. Pode ser citado também a facilidade dos seus usuários terão, uma vez que eles poderão utilizar o sistema de maneira *online*. Para a construção do projeto, foram utilizadas tecnologias que auxiliam em diversos aspectos do desenvolvimento, dentre as quais pode-se destacar as linguagens Java e TypeScript, além dos *frameworks* Spring e Angular.

Palavras-chave: API REST. SPA. Spring. Angular. Empréstimos.

ABSTRACT

The institutions usually need to organize their information to obtain the best execution of their daily activities, and the use of a system that helps them in this task is necessary. Moreover, since they allow the universalization of information, web systems received a lot of attention in recent years. Consequently, this work was developed with the goal of implement a web system for controlling and monitoring equipments loans from the Electrical Department of UTFPR, campus Pato Branco. The system developed brings several benefits: in addition to a better organized management, the system brings improved security to the department. Also, the comfort for the users can be mentioned, since everyone can interact with the system online. For the success of the project, different technologies were used to assist in several points of the development, among which the Java and TypeScript languages can be highlighted, in addition to the Spring and Angular frameworks.

Keywords: API REST. SPA. Spring. Angular. Loans.

LISTA DE FIGURAS

Figura 1 : Diagrama de casos de uso do projeto 22
Figura 2 : Diagrama de entidades e relacionamentos do projeto 23
Figura 3 : Página de autenticação 28
Figura 4 : Página "Esqueci minha senha" 29
Figura 5 : Página "Quero me cadastrar" 30
Figura 6 : Página *Home* dos alunos e professores 31
Figura 7 : Página *Home* dos administradores e laboratoristas 31
Figura 8 : Página da lista de fornecedores 32
Figura 9 : Página de cadastro de fornecedores 33
Figura 10 : Página da lista de equipamentos 33
Figura 11 : Página de cadastro de equipamentos 34
Figura 12 : Página da lista de entradas 34
Figura 13 : Página de cadastro de entradas 35
Figura 14 : Página de lançamento de equipamentos nas entradas 35
Figura 15 : Página de lista de empréstimos 36
Figura 16 : Página de cadastro de empréstimos 37
Figura 17 : Página de lançamento de equipamentos nos empréstimos 38
Figura 18 : Página de lançamento de equipamentos durante encerramento de empréstimo 38
Figura 19 : Página de cadastro de baixas de estoque 39
Figura 20 : Página do relatório de ficha de estoque 39
Figura 21 : Página com o resultado do relatório de ficha de estoque 40
Figura 22 : Página da lista de usuários da aplicação 40
Figura 23 : Página de cadastro de usuários 41
Figura 24 : Página para alterar a senha do usuário 41

LISTA DE QUADROS

Quadro 1 : Materiais e tecnologias do projeto 13
Quadro 2 : Requisitos funcionais do projeto 19
Quadro 3 : Requisitos não funcionais do projeto 21
Quadro 4 : Tabela "uf" 24
Quadro 5 : Tabela "cidade" 24
Quadro 6 : Tabela “fornecedor” 25
Quadro 7 : Tabela "usuario" 25
Quadro 8 : Tabela "equipamento" 26
Quadro 9 : Tabela "entrada" 26
Quadro 10 : Tabela "entradaitem" 26
Quadro 11 : Tabela "saida" 27
Quadro 12 : Tabela "saidaitem" 27
Quadro 13 : Tabela "estoque" 28

LISTAGEM DE CÓDIGOS

Listagem 1 : Arquivo "pom.xml" 42
Listagem 2 : Arquivo "application.properties" 45
Listagem 3 : Classe do *setup* da API 45
Listagem 4 : Classe "Saida" com mapeamento objeto-relacional 46
Listagem 5 : Classe de repositório "SaidaData" 47
Listagem 6 : Classe base para os serviços do back-end 48
Listagem 7 : Classe de serviço "SaidaService" 49
Listagem 8 : Classe controladora "SaidaController" 54
Listagem 9 : Classe que valida um usuário e gera um *token* 56
Listagem 10 : Classe que solicita a autenticação de um usuário e armazena o *token* 58
Listagem 11 : Classe que intercepta as requisições e adiciona o *token* 58
Listagem 12 : Classe que contém a configuração para o funcionamento do interceptador 59
Listagem 13 : Classe base dos *services* da aplicação cliente 59
Listagem 14 : Classe *service* que interage com o *endpoint* de saídas 61
Listagem 15 : HTML para exibição dos empréstimos cadastrados 64
Listagem 16 : Classe que carrega os empréstimos e interage com o HTML 66

LISTA DE SIGLAS E ABREVIATURAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	*Application Programming Interface*
CRUD	*Create, Read, Update e Delete*
DAELE	Departamento Acadêmico de Elétrica da UTFPR
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
JMS	*Java Message Service*
JPA	*Java Persistence API*
JSON	*JavaScript Object Notation*
REST	*Representational State Transfer*
RF	*Requisito Funcional*
RNF	*Requisito Não Funcional*
SPA	*Single Page Application*
SQL	*Structured Query Language*
UI	*User Interface*
UTFPR	Universidade Tecnológica Federal do Paraná
XML	*Extensible Markup Language*

SUMÁRIO

1 **INTRODUÇÃO** 10
1.1 CONSIDERAÇÕES INICIAIS 10
1.2 OBJETIVOS 11
1.2.1 Objetivo Geral 11
1.2.2 Objetivos Específicos 11
1.3 JUSTIFICATIVA 11
1.4 ESTRUTURA DO TRABALHO 12
2 **MATERIAIS E MÉTODO** 13
2.1 MATERIAIS 13
2.2 MÉTODO 15
3 **RESULTADOS** 17
3.1 ESCOPO DO SISTEMA 17
3.2 MODELAGEM DO SISTEMA 19
3.3 APRESENTAÇÃO DO SISTEMA 28
3.4 IMPLEMENTAÇÃO DO SISTEMA 42
4 **CONCLUSÃO** 69
5 **REFERÊNCIAS** 70

1 INTRODUÇÃO

Este capítulo apresenta a proposta do trabalho, iniciando com as principais considerações, seguido dos objetivos gerais e específicos, passando pela justificativa e é finalizado com a estrutura do texto.

1.1 CONSIDERAÇÕES INICIAIS

Em um mundo com alta troca de informações entre pessoas e organizações, essas procuram ter sempre disponível a informação de forma rápida, íntegra e confidencial. Para que isto seja possível, é importante que as organizações possuam sistemas de informação e tecnologias de informação (PIMENTA; QUARESMA, 2016). Um sistema de informação é definido por Rodrigues (2002), como um conjunto de procedimentos, atividades, pessoas e tecnologias envolvidas na coleta de dados relevantes, armazenamento enquanto necessário, processamento dos dados e na disponibilização de informação a quem necessite da mesma.

Nos ambientes computacionais, existem diferentes maneiras de disponibilizar um sistema de informação, dentre as quais podem ser destacadas os sistemas *desktop*, isto é, aqueles projetados para serem executados localmente em diferentes estações de trabalho e os sistemas *web*, que tem como premissa a disponibilidade do sistema de maneira única, em qualquer dispositivo com acesso à Internet.

Com o avanço da utilização da Internet, aumentou de maneira significativa o desenvolvimento de aplicações *web* devido à portabilidade, acessibilidade e facilidade na utilização e implantação (CAMARGO, 2009). Zanetti Junior (2003) lembra que sistemas *web* são caracterizados pela universalização do acesso às redes de computadores e pela utilização de sistemas de padrões abertos para comunicação.

Diante desse cenário, identificou-se a viabilidade do desenvolvimento de um sistema de informação, projetado para o ambiente *web,* que será implantado no Departamento Acadêmico de Elétrica da UTFPR, a fim de aperfeiçoar os processos do departamento, principalmente no que diz respeito ao fornecimento de equipamentos por empréstimo.

1.2 OBJETIVOS

Nesta seção serão apresentados os objetivos geral e específicos do trabalho.

1.2.1 Objetivo Geral

Desenvolver uma aplicação para auxiliar no processo de empréstimo de materiais do Departamento Acadêmico de Elétrica da UTFPR junto aos alunos e professores.

1.2.2 Objetivos Específicos

- Permitir solicitações de empréstimos *online* por parte dos alunos e professores da UTFPR.
- Fornecedor um controle de estoque mais eficiente dos equipamentos do departamento.
- Oferecer maior rastreabilidade dos equipamentos do departamento.

1.3 JUSTIFICATIVA

Avaliando o processo de empréstimo atual que o departamento acadêmico de elétrica disponibiliza aos seus usuários, foi identificado que ele apresenta algumas deficiências e riscos na gestão do departamento, visto que o controle é realizado por meio de anotações manuais.

A implementação do projeto se justifica pela segurança, comodidade e facilidade que o *software* poderá trazer aos seus usuários. Enquanto alunos e professores poderão requerer equipamentos de maneira *online*, os colaboradores do departamento receberão instantaneamente as solicitações e poderão avaliar a disponibilidade dos equipamentos, por meio do *software*.

Por fim, destaca-se que o *software* foi construído para utilização na *web*. As tecnologias utilizadas na construção do produto, por sua vez, foram selecionadas como consequência deste fator, além da estabilidade e ampla utilização no mercado.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. O Capítulo 2 apresenta os materiais e o método utilizados para o desenvolvimento do sistema. No Capítulo 3 estão os resultados da realização do trabalho, a apresentação do projeto desenvolvido por meio de suas telas e partes do código. Ao final, no Capítulo 4, está a conclusão do projeto. Em seguida estão as referências utilizadas na composição do texto.

2 MATERIAIS E MÉTODO

Neste capítulo são elencadas as ferramentas e tecnologias utilizadas para a modelagem e o desenvolvimento do projeto, além do método de desenvolvimento do trabalho.

2.1 MATERIAIS

O Quadro 1 expõe os materiais e tecnologias adotadas para a implementação da aplicação desenvolvida como resultado do presente trabalho.

Quadro 1: Materiais e tecnologias do projeto

Material/Tecnologia	Versão	Disponível em	Aplicação
LucidChart		https://www.lucidchart.com/pages/pt	Ferramenta para desenvolvimento de diagramas *Unified Modeling Language*.
Spring Tool Suite	3.9.7. RELEASE	https://spring.io/tools	*Integrated Development Environment* (IDE).
Java	1.8.0_201	https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html	Linguagem de programação.
Apache Maven	3.6.0	http://maven.apache.org/download.cgi	Ferramenta para gerenciamento de dependências.
Spring Boot, Spring Data, Spring Security	2.1.8. RELEASE	https://spring.io/	*Framework* para desenvolvimento de projetos na linguagem Java.
JasperReports	6.10.0	https://www.jaspersoft.com/	Ferramenta para elaboração de relatórios.
Visual Studio Code	1.41.1	https://code.visualstudio.co	*Integrated Development*

		m/	*Environment* (IDE).
Angular	8.3.10	https://angular.io/	Framework para desenvolvimento *front-end*.
PrimeNG	8.0.4	https://primefaces.org/primeng	Suíte de componentes para elaboração e estilização de páginas *Hypertext Markup Language*.
Bootstrap	4.3.1	https://getbootstrap.com/	Biblioteca para estilização de páginas HTML.
MomentJS	2.24.0	https://momentjs.com/	Biblioteca JavaScript para manipulação de data/hora.
PostgreSQL	10.7	https://www.postgresql.org/	Banco de dados.
DBEaver	6.3.1	https://dbeaver.io/	Administrador de banco de dados.
SocjJS	0.3.19	https://www.npmjs.com/package/sockjs	Biblioteca JavaScript *WebSocket*.
StompJS	2.3.3	https://www.npmjs.com/package/@stomp/stompjs	Biblioteca JavaScript *WebSocket*.

Fonte: Autoria própria

Os materiais e as tecnologias constantes no Quadro 1 são de uso bastante difundido em projetos *web*. A seguir é apresentado um conteúdo mais aprofundado acerca de alguns dos itens listados. A escolha pelo destaque dessas tecnologias é por conta da ampla utilização no projeto em relação às demais.

O Spring surgiu em 2003 como uma resposta à complexidade das especificações iniciais do J2EE. Enquanto alguns consideram o JavaEE e o Spring concorrentes, o Spring é, na verdade, complementar ao padrão anterior. Seu modelo de programação integra-se com especificações

individuais do JavaEE, cuidadosamente selecionadas, tais como: API Servlet, Bean Validation, JPA, JMS, entre outras. Além disso, ele também suporta a especificação de injeção de dependências (SPRING, 2020).

Somado às características anteriores, Souza (2017) lembra que o *framework* permite que a maioria das suas configurações sejam feitas de maneira programática, ou seja, não há necessidade de criar nenhum arquivo *Extensible Markup Language* (XML) para que os módulos comecem a funcionar.

O Angular, por sua vez, trata-se de um *framework* utilizado para a construção de *Single Page Applications* (SPA), que utiliza basicamente a linguagem de marcação HTML e do TypeScript. A sua arquitetura é baseada em alguns conceitos fundamentais, tais como módulos e componentes (ANGULAR, 2020).

Os módulos, de acordo com Calado (2020), fornecem um contexto no qual os componentes são compilados, sendo que uma aplicação deve possuir, ao menos, um módulo raiz, que habilita a inicialização. Componentes, por sua vez, definem as páginas de visualização, que podem ser modificadas de acordo com a lógica da aplicação.

O PostgreSQL é um sistema de banco de dados objeto relacional, que usa e estende a linguagem *Structured Query Language* (SQL), sendo que sua origem é datada em 1986. Além de ser gratuito e de código aberto, é altamente extensível, permitindo, por exemplo, que sejam definidos tipos de dados e funções personalizadas. Outro fato importante é que ele é executado em todos os sistemas operacionais e suporta os conceitos de Atomicidade, Consistência, Isolamento e Durabilidade (ACID) (POSTGRESQL, 2020).

Por fim, o PrimeNG trata-se de uma coleção de componentes *open source* para *User Interface* (UI), criada e mantida pela empresa Primetek, que também é responsável pelo Primefaces, uma outra coleção de componentes para projetos que utilizam JavaServer Faces (PRIMENG, 2020). A sua utilização retira a necessidade de implementar alguns comportamentos para *tags* padrões do HTML, visto que os componentes já possuem suas principais ações bem definidas.

2.2 MÉTODO

O método escolhido para o desenvolvimento do projeto se caracteriza como iterativo e incremental. Basicamente, um processo iterativo se baseia em refinamentos do projeto de maneira progressiva, isto é, com ciclos. Em cada ciclo, há uma pequena construção do software, um

“pedaço”, caracterizando assim um processo incremental, e definindo o modelo proposto como um todo. A seguir são apresentadas as etapas do projeto.

a) Levantamento de requisitos – As propostas de requisitos foram obtidas a partir de reuniões entre o orientador deste trabalho e os responsáveis pelo DAELE. Nas reuniões, foram apresentados os procedimentos do departamento e na sequência esses processos foram repassados ao autor do projeto.

b) Análise e projeto – Nesta etapa os requisitos foram avaliados e modelados a fim de elaborar a estrutura do projeto. O diagrama de casos de uso, assim como o diagrama de entidades e relacionamentos foram montados, e algumas revisões dos requisitos foram necessárias.

c) Desenvolvimento – Inicialmente as tecnologias a serem utilizadas foram escolhidas. Depois, o projeto do *back-end* foi iniciado, com a disponibilização dos cadastros de equipamentos, fornecedores, entradas de estoque e empréstimos de equipamentos. Tendo os cadastros estruturados, foram codificados os trechos responsáveis pela segurança do projeto, isto é, os mecanismos de autenticação e autorização para acessar os recursos do *back-end*. Na sequência, o projeto *front-end* foi iniciado com a criação das telas que representam os cadastros do *software*. As demais funcionalidades foram sendo acrescentadas ao longo do desenvolvimento, com o objetivo de atender a todos os requisitos propostos.

d) Testes – Os testes foram realizados de maneira informal, com o objetivo de encontrar possíveis erros de codificação.

3 RESULTADOS

Este capítulo apresenta os resultados obtidos com o presente trabalho. Está dividido em escopo do sistema, modelagem, apresentação do sistema e de alguns dos principais códigos fontes desenvolvidos.

3.1 ESCOPO DO SISTEMA

A aplicação tem por meta melhorar a gestão do departamento acadêmico de elétrica da UTFPR no que diz respeito ao fornecimento de equipamentos para alunos e professores, centralizando em uma única aplicação todo o processo de solicitação, aprovação e acompanhamento de empréstimos. Além disso, controlar de maneira eficiente as movimentações de estoque dos equipamentos, desde o lançamento de entradas até as saídas e retornos de empréstimo, bem como as baixas de estoque. Para isso, a aplicação disponibiliza algumas ações para que esses objetivos sejam atingidos.

O processo inicia com o cadastro de usuários, que podem ser alunos, professores, laboratoristas ou administradores. Uma vez cadastrados, qualquer ação no sistema é realizada após um *login* prévio, sendo que alunos e professores terão a sua disposição apenas a possibilidade de solicitar empréstimos e acompanhar o andamento deles. Já os laboratoristas e administradores serão responsáveis por cadastrar os equipamentos do departamento, os fornecedores, além das movimentações de estoque por entrada, empréstimos e baixas. Ainda, para acompanhamento mais preciso dos saldos, poderão também visualizar relatórios de estoque dos equipamentos, além de dar manutenção no cadastro de novos usuários.

As funcionalidades detalhadas da aplicação são destacadas a seguir:

1) Cadastro de Equipamento – Os laboratoristas e administradores deverão registrar todos os equipamentos fornecidos pelo departamento. Os campos necessários para esse cadastro foram definidos de acordo com as necessidades do departamento.
2) Cadastro de Fornecedor – Ficará também a cargo dos laboratoristas e administradores cadastrar os fornecedores dos materiais, sendo que tal cadastro se faz necessário para que o processo de entrada seja mais preciso.
3) Cadastro de Entradas – Etapa indispensável em um controle de estoque, tal funcionalidade só será disponível para os laboratoristas e administradores. Nesse cadastro, o usuário deverá

indicar o fornecedor, além dos materiais adquiridos com suas respectivas quantidades e valores. Ao concluir esse cadastro, uma movimentação de estoque é realizada, adicionando aos saldos de estoque as quantidades adquiridas na data informada.

4) Cadastro de Empréstimo – Principal funcionalidade do *software*, poderá ser acessada por qualquer tipo de usuário, principalmente alunos e professores. Dentre os campos necessários, o usuário precisará indicar qual a finalidade da solicitação e os equipamentos desejados. Os empréstimos poderão ter quatro *status* diferentes: “pendente”, “aprovado”, “encerrado” ou “reprovado”. Empréstimos oriundos de alunos e professores terão sempre como situação inicial o *status* “pendente”, sendo que apenas laboratoristas e administradores podem alterar essa situação. Depois da situação “pendente”, um empréstimo poderá ficar com o *status* “aprovado” ou “reprovado”. Caso seja aprovado, é realizada uma movimentação de estoque de saída e um *e-mail* é enviado ao usuário que cadastrou o empréstimo, confirmando a disponibilidade dos equipamentos. Do contrário, o usuário é notificado por *e-mail* sobre a rejeição do empréstimo. Por fim, quando o empréstimo for concluído, deverá o laboratorista ou administrador atribuir o *status* “encerrado” para ele, sendo que, nesse momento, uma movimentação de entrada de estoque é feita, visto que os materiais retornaram ao departamento. Enquanto o empréstimo estiver na situação “pendente”, tanto o usuário que requereu quanto um laboratorista ou um administrador poderão editá-lo. Quando o empréstimo estiver com a situação diferente de “pendente”, não se altera mais os seus dados.

5) Relatório de Estoque – Os gestores do departamento poderão, sempre que necessário, visualizar os estoques dos materiais por meio desse relatório. Para utilizá-lo, basta informar a data desejada e os equipamentos para consulta.

6) Cadastro de usuários – Os novos usuários poderão se registrar pela página inicial da aplicação, desde que sejam alunos ou professores. Os outros tipos de usuários só poderão ser registrados por algum outro desses tipos de usuários previamente cadastrados. Entre os campos necessários, destaca-se o e-mail, o número do registro acadêmico e, obviamente, a senha de acesso. Quando um aluno ou professor se cadastra pela página inicial, a efetivação desse cadastro depende de um laboratorista ou administrador. Tal ação foi adotada para impedir que usuários sem relação com o departamento se registrem de maneira desnecessária. Depois que um cadastro é efetivado, o usuário que se registrou receberá um *e-mail* confirmando a sua conclusão.

7) *Home* – A tela inicial da aplicação apresentará informações diferentes de acordo com o tipo do usuário: enquanto os alunos e professores visualizarão seus empréstimos solicitados, os

laboratoristas e administradores poderão analisar as novas solicitações de usuários, bem como os empréstimos pendentes e os equipamentos cujo estoque está próximo do fim.

3.2 MODELAGEM DO SISTEMA

Tendo em vista as diferentes ações e responsabilidades que existem na aplicação, o Quadro 2 apresenta os requisitos funcionais definidos para o sistema. Para facilitar a identificação, foi adotado o prefixo RF para os requisitos funcionais, seguido de numeração ordenada.

Quadro 2: Requisitos funcionais do projeto

Identificação	Nome	Descrição
RF01	Manutenção no cadastro de equipamentos	Usuários do tipo laboratorista ou administrador deverão ter a sua disposição as opções de cadastro, edição, exclusão e listagem de equipamentos.
RF02	Manutenção no cadastro de fornecedores	Usuários do tipo laboratorista ou administrador deverão ter disponíveis as opções de cadastro, edição, exclusão e listagem de fornecedores.
RF03	Manutenção no cadastro de entradas de estoque	Usuários do tipo laboratorista ou administrador deverão conseguir cadastrar, editar, remover e visualizar as entradas de estoque.
RF04	Manutenção no cadastro de empréstimos de equipamentos	Qualquer usuário poderá requerer um empréstimo. Solicitações de alunos e professores deverão passar por um processo de aprovação. Poderá um empréstimo ter quatro *status* diferentes: “pendente”, “aprovado”, “reprovado” e “encerrado”. O empréstimo só pode ser editado caso esteja com *status* “pendente”. Um empréstimo pode ser removido a qualquer momento.
RF05	Visualizar Estoques	Laboratorista e administradores poderão visualizar os estoques por meio de relatório

		ou listagem dos equipamentos com estoques próximos do fim.
RF06	Cadastro de usuários	Alunos e professores poderão se cadastrar sem a participação de um outro usuário pré-cadastrado. Laboratoristas e administradores dependem de um outro usuário para cadastrá-los.
RF07	Ativação/Inativação de usuários	Poderão os laboratoristas ou administradores ativar ou desativar os demais usuários da aplicação.
RF08	Atualizar senha de acesso	Qualquer usuário poderá, a qualquer momento, alterar sua senha de acesso.
RF09	Aprovação/Reprovação de usuários	Os laboratoristas e administradores poderão definir se um usuário será aceito na aplicação.
RF10	Aprovação/Reprovação de Empréstimos	Os laboratoristas e administradores poderão definir se um empréstimo poderá ser realizado ou não.
RF11	Solicitar nova senha de acesso	Qualquer usuário poderá requerer nova senha de acesso, caso a atual tenha sido perdida.
RF12	Notificações de aceite de usuário via *e-mail*	Novos usuários serão notificados via *e-mail* quando seus cadastros forem aprovados.
RF13	Notificações de aceite de empréstimo via *e-mail*	Os usuários serão notificados via *e-mail* quando seus empréstimos forem aprovados para retirada.
RF14	Movimentação de estoque nas entradas	Ao concluir uma entrada de estoque, os saldos dos equipamentos adquiridos devem sofrer acréscimo na data em que o processo ocorreu.
RF15	Movimentação de estoque nos	Quando um empréstimo estiver com a

	empréstimos	situação “aprovada”, os saldos de estoque dos equipamentos emprestados sofrerão decréscimo. Quando esse empréstimo for encerrado, o estoque é acrescido novamente.
RF16	Movimentação de estoque nas baixas	Quando uma baixa de estoque é feita, os saldos dos equipamentos informados sofrem decréscimo.

Fonte: Autoria própria

O Quadro 3, apresenta os requisitos não funcionais. Foi adotado o prefixo RNF para esses requisitos, seguido de numeração ordenada

Quadro 3: Requisitos não funcionais do projeto

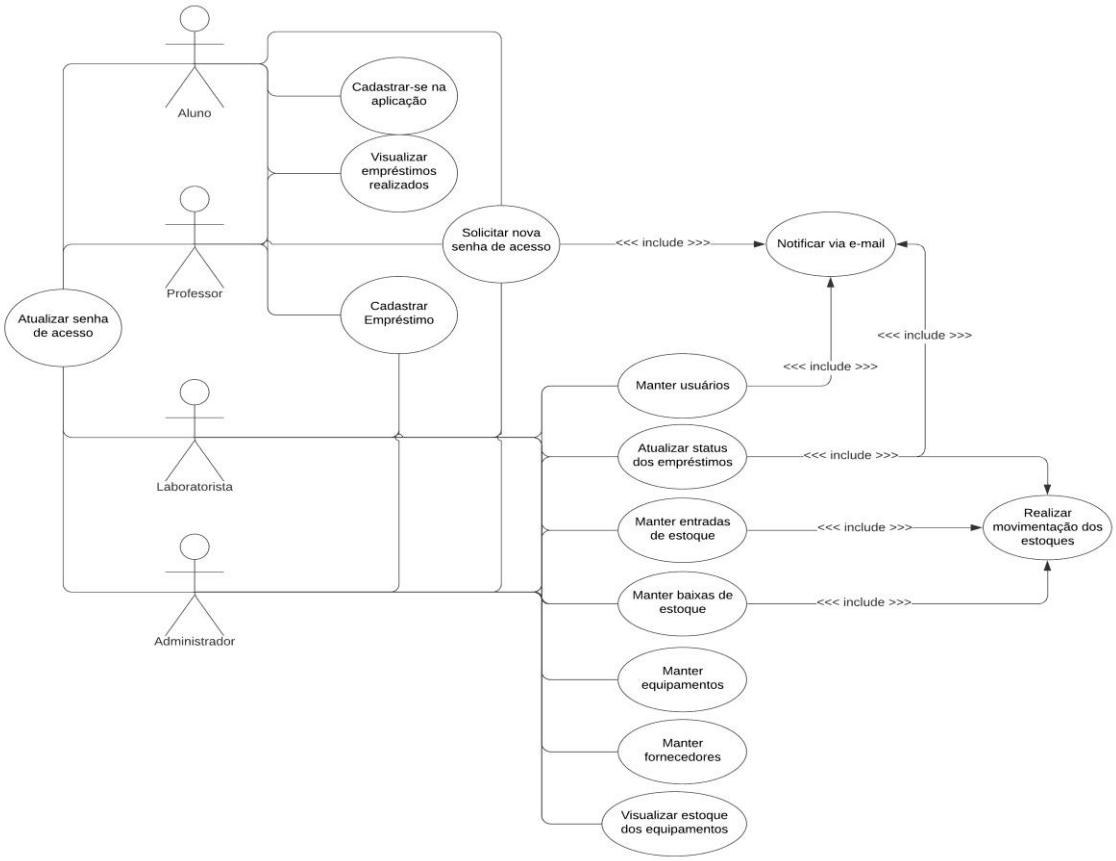
Identificação	Nome	Descrição
RNF01	Persistência de dados	Todos os cadastros serão persistidos em banco de dados PostgreSQL.
RNF02	Autenticação no sistema	Todas as ações no *software* deverão ser autenticadas, exceto o cadastro de alunos e professores. Um *login* prévio na aplicação é necessário, gerando um *token* com uma determinada vida útil. Enquanto o *token* não atinge seu tempo de expiração, não é necessário um novo *login*.
RNF03	Conta GMail para envio de *e-mails*	Os *e-mails* deverão ser enviados por uma conta GMail previamente informada no arquivo de configuração de aplicação.
RNF04	Envio de *e-mails* em segundo plano	Para não interferir negativamente na aplicação, gerando travamentos e afins, os *e-mails* que a aplicação gera serão enviados em um processo secundário.
RNF05	Notificação de novos usuários	Os laboratoristas e administradores devem conseguir visualizar instantaneamente as novas solicitações de usuários na página

		home da aplicação.
RNF06	Notificação de novos empréstimos	Os laboratoristas e administradores devem conseguir visualizar instantaneamente as novas solicitações de empréstimo na página *home* da aplicação.
RNF07	Notificação de atualizações nos empréstimos	Os alunos e professores devem conseguir visualizar instantaneamente as atualizações de *status* que os seus empréstimos sofrem.
RNF08	Ocultar menus da aplicação de acordo com a atribuição do usuário	Os usuários só terão a disposição os menus das ações que eles têm autorização de execução.

Fonte: Autoria própria

Com base nos requisitos apresentados anteriormente, em especial os funcionais, elaborou-se o diagrama de casos de uso apresentado na Figura 1, o qual apresenta as ações do sistema, separando os atores de acordo com a sua devida atribuição.

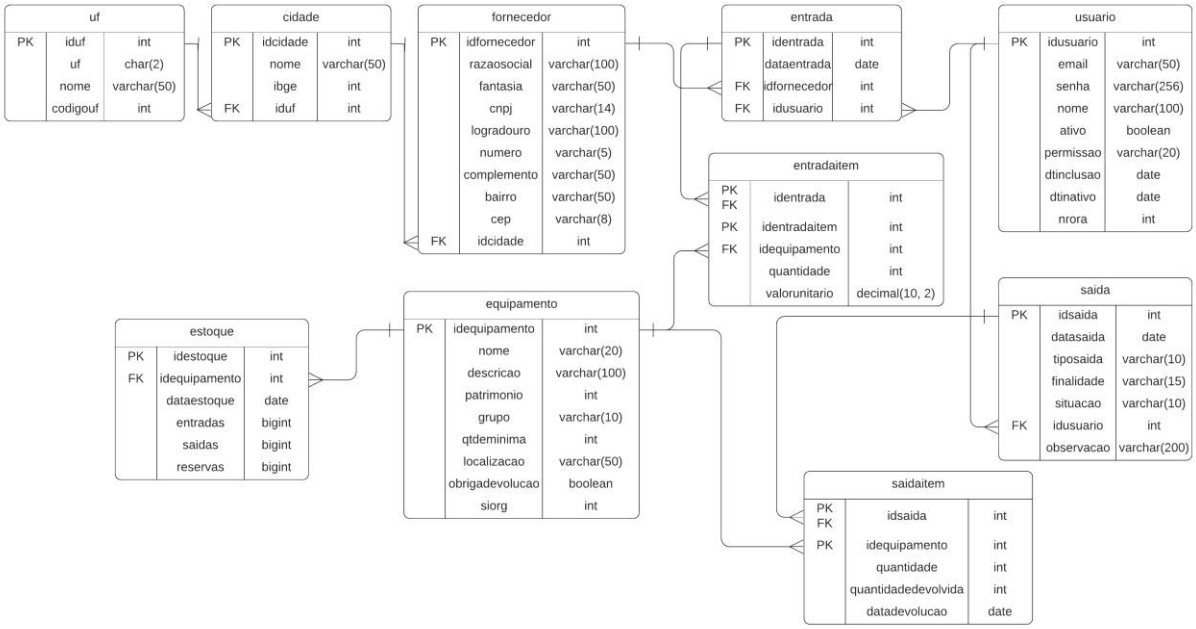
Figura 1: Diagrama de casos de uso do projeto



Fonte: Autoria própria

Destaca-se na Figura 2 o diagrama de entidades e relacionamentos do banco de dados.

Figura 2: Diagrama de entidades e relacionamentos do projeto



Fonte: Autoria própria

Para armazenamento dos usuários da aplicação, foi definida a tabela “usuario”. Dentre seus campos, destaca-se a presença do campo “permissao”, que será o responsável por indicar qual a atribuição do usuário: se é um aluno, professor, laboratorista ou administrador. Já os equipamentos controlados pelo departamento serão persistidos na tabela “equipamento”.

Para as movimentações de estoque de entrada, foi especificada a tabela “entrada”, que armazenará a data do processo, além do fornecedor e o usuário que realizou o cadastro. Quanto aos equipamentos da entrada, serão armazenados na tabela “entradaitem”, a qual possui relacionamento do tipo “*1 – n”* com a tabela anterior. Nela, serão persistidos os materiais adquiridos, bem como suas quantidades e valores. Ainda relacionado as entradas, existe na estrutura do banco a tabela “fornecedor”, com os campos característicos de uma instituição desse tipo.

Já para as movimentações de saída, criou-se a tabela “saida”, que contém o campo da data do cadastro, além da finalidade, a qual pode ser para aulas práticas ou projetos, o tipo da saída, podendo ser empréstimo ou baixa de estoque, situação (“pendente”, “aprovada”, “reprovada” ou “encerrada”) e o usuário que realizou o cadastro. Os materiais de saída, por sua vez, ficarão na tabela “saidaitem”, sendo que tal tabela possui relacionamento “*1 – N*” com “saida” e armazenam os

equipamentos e as quantidades, além da quantidade devolvida e data de devolução (quando a saída for do tipo “empréstimo”).

A tabela “estoque” tem por finalidade salvar as quantidades movimentadas em cada processo de entrada ou saída do *software*. Quando uma entrada é cadastrada, as quantidades dos equipamentos são lançadas no campo “entradas”. Já quando uma baixa de estoque é salva, as quantidades são persistidas no campo “saídas”. Quanto aos empréstimos, o processo de persistência nessa tabela varia de acordo com o *status* deles, isto é, quando a situação for “aprovado”, a quantidade dos equipamentos fica na coluna “reservas”, enquanto que quando a situação for “encerrado”, as quantidades vão para a coluna “entradas”.

O Quadro 4 apresenta a listagem dos campos da tabela “uf”.

Quadro 4: Tabela "uf"

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
iduf	Numérico	Não	Sim	Não
uf	Texto	Não	Não	Não
nome	Texto	Não	Não	Não
codigouf	Numérico	Não	Não	Não

Fonte: Autoria própria

O Quadro 5 exibe os campos presentes na tabela “cidade”.

Quadro 5: Tabela "cidade"

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
idcidade	Numérico	Não	Sim	Não
nome	Texto	Não	Não	Não
ibge	Numérico	Sim	Não	Não
iduf	Numérico	Não	Não	Sim

Fonte: Autoria própria

O Quadro 6, por sua vez, lista os campos da tabela “fornecedor”.

Quadro 6: Tabela “fornecedor”

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
idfornecedor	Numérico	Não	Sim	Não
razaosocial	Texto	Não	Não	Não
fantasia	Texto	Sim	Não	Não
cnpj	Texto	Não	Não	Não
logradouro	Texto	Não	Não	Não
numero	Texto	Não	Não	Não
complemento	Texto	Sim	Não	Não
bairro	Texto	Não	Não	Não
cep	Texto	Não	Não	Não
idcidade	Numérico	Não	Não	Sim

Fonte: Autoria própria

O Quadro 7 apresenta a listagem dos campos da tabela “usuario”.

Quadro 7: Tabela "usuario"

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
idusuario	Numérico	Não	Sim	Não
email	Texto	Não	Não	Não
senha	Texto	Não	Não	Não
nome	Texto	Não	Não	Não
ativo	Booleano	Não	Não	Não
permissao	Texto	Não	Não	Não
dtinclusao	Data	Sim	Não	Não
dtinativo	Data	Sim	Não	Não
nrora	Numérico	Sim	Não	Não

Fonte: Autoria própria

Para exibir os detalhes da tabela “equipamento”, o Quadro 8 listará seus campos.

Quadro 8: Tabela "equipamento"

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
idequipamento	Numérico	Não	Sim	Não
nome	Texto	Não	Não	Não
descricao	Texto	Sim	Não	Não
patrimonio	Numérico	Sim	Não	Não
grupo	Texto	Não	Não	Não
qtdeminima	Numérico	Não	Não	Não
localizacao	Texto	Sim	Não	Não
obrigadevolucao	Booleano	Não	Não	Não
siorg	Numérico	Sim	Não	Não

Fonte: Autoria própria

O Quadro 9 apresenta os campos presentes na tabela “entrada”.

Quadro 9: Tabela "entrada"

Campo	Tipo	Nulo	Chave primária	Chave Estrangeira
identrada	Numérico	Não	Sim	Não
dataentrada	Data	Não	Não	Não
idfornecedor	Numérico	Não	Não	Sim
idusuario	Numérico	Não	Não	Sim

Fonte: Autoria própria

Com relação a tabela “entradaitem”, os campos são listados no Quadro 10. Ela possui relação “*1 – N*” com a tabela “entrada”.

Quadro 10: Tabela "entradaitem"

Campo	Tipo	Nulo	Chave primária	Chave estrangeira

identrada	Numérico	Não	Sim	Sim
identradaitem	Numérico	Não	Sim	Não
idequipamento	Numérico	Não	Não	Sim
quantidade	Numérico	Não	Não	Não
valorunitario	Numérico	Não	Não	Não

Fonte: Autoria própria

Para listar os campos da tabela “saida”, tem-se o Quadro 11.

Quadro 11: Tabela "saida"

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**
idsaida	Numérico	Não	Sim	Não
datasaida	Data	Não	Não	Não
tiposaida	Texto	Não	Não	Não
finalidade	Texto	Não	Não	Não
situacao	Texto	Não	Não	Não
idusuario	Numérico	Não	Não	Sim
observacao	Texto	Sim	Não	Não

Fonte: Autoria própria

A tabela “saidaitem”, por sua vez, tem seus campos listados no Quadro 12.

Quadro 12: Tabela "saidaitem"

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**
idsaida	Numérico	Não	Sim	Sim
idequipamento	Numérico	Não	Sim	Sim
quantidade	Numérico	Não	Não	Não
quantidadedevolvida	Numérico	Sim	Não	Não
datadevolucao	Data	Sim	Não	Não

Fonte: Autoria própria

Para finalizar a estrutura do banco, o Quadro 13 demonstra os campos da tabela “estoque”.

Quadro 13: Tabela "estoque"

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
idestoque	Numérico	Não	Sim	Não
idequipamento	Numérico	Não	Não	Sim
dataestoque	Data	Não	Não	Não
entradas	Numérico	Sim	Não	Não
saidas	Numérico	Sim	Não	Não
reservas	Numérico	Sim	Não	Não

Fonte: Autoria própria

3.3 APRESENTAÇÃO DO SISTEMA

Como o *software* foi construído para ser acessível no ambiente *web*, é necessário que o usuário utilize um navegador. Ao iniciar a aplicação pela primeira vez, o usuário visualizará a página de autenticação. Nela, além do mecanismo de acesso ao *software*, existem também *links* que permitem com que o usuário se cadastre ou solicite uma nova senha de acesso, caso seja necessário. Essa página é a única que não requer nenhum tipo de autorização para acessá-la.

Outro ponto que merece destaque é que essa página não será exibida a cada vez que o usuário acessar a aplicação. Ao concluir o processo de autenticação, é gerado um *token*, mecanismo que identifica o usuário que está acessando o *software*, que permite que ele navegue pela aplicação durante um tempo pré-determinado (definido pelo desenvolvedor do projeto) ou encerre a página do navegador e acesse novamente, sem ter que se identificar toda vez.

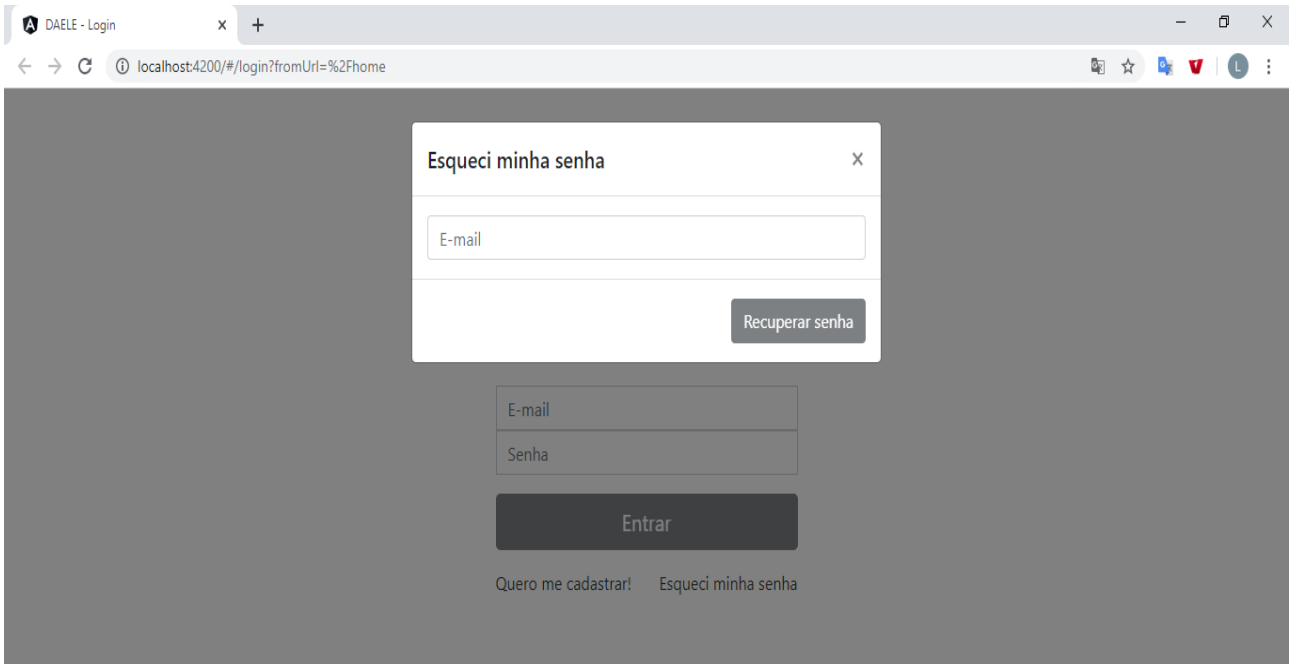
A Figura 3 apresenta a página de autenticação e os demais *links*.

Figura 3: Página de autenticação

Fonte: Autoria própria

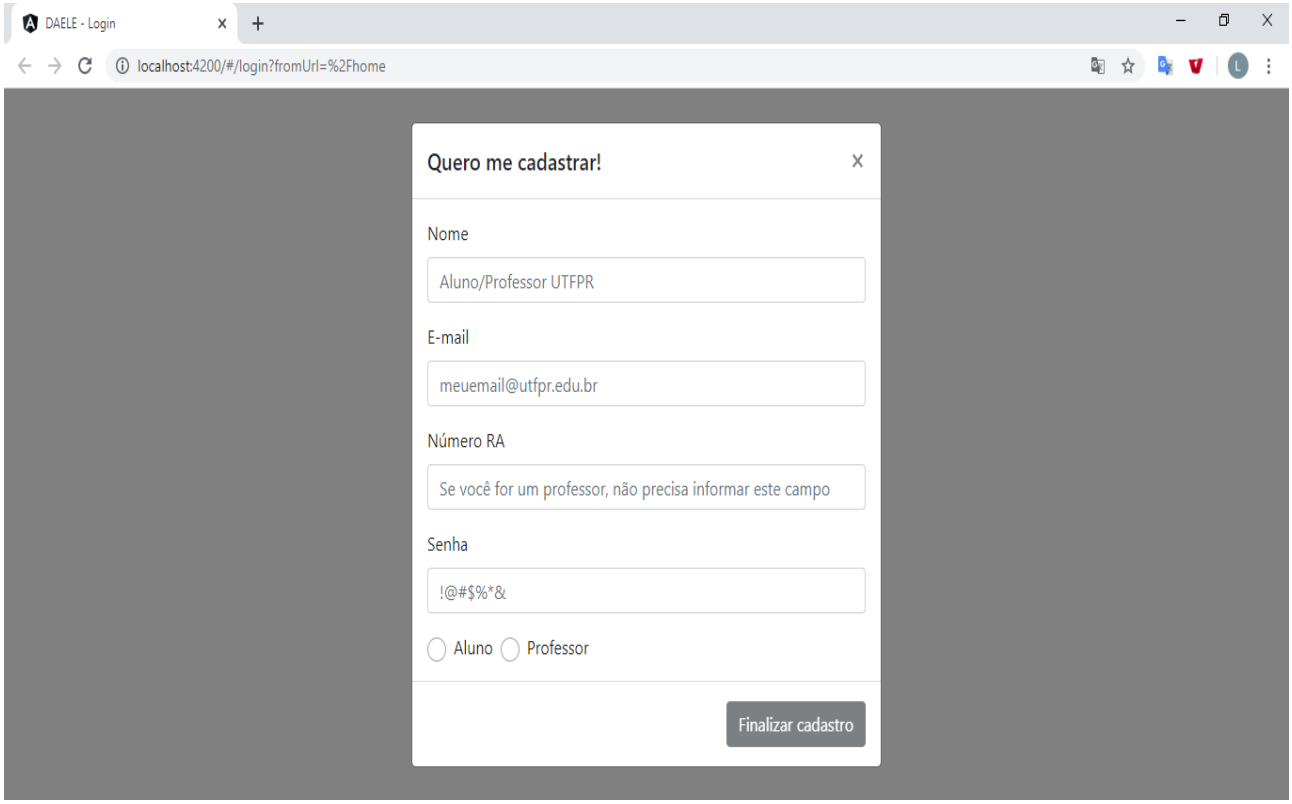
As Figuras 4 e 5 mostram os outros *links* da página em funcionamento, sendo que a primeira exibe o resultado ao clicar sobre a opção “Esqueci minha senha” e a segunda apresenta o item “Quero me cadastrar” em execução.

Figura 4: Página "Esqueci minha senha"



Fonte: Autoria própria

Figura 5: Página "Quero me cadastrar"



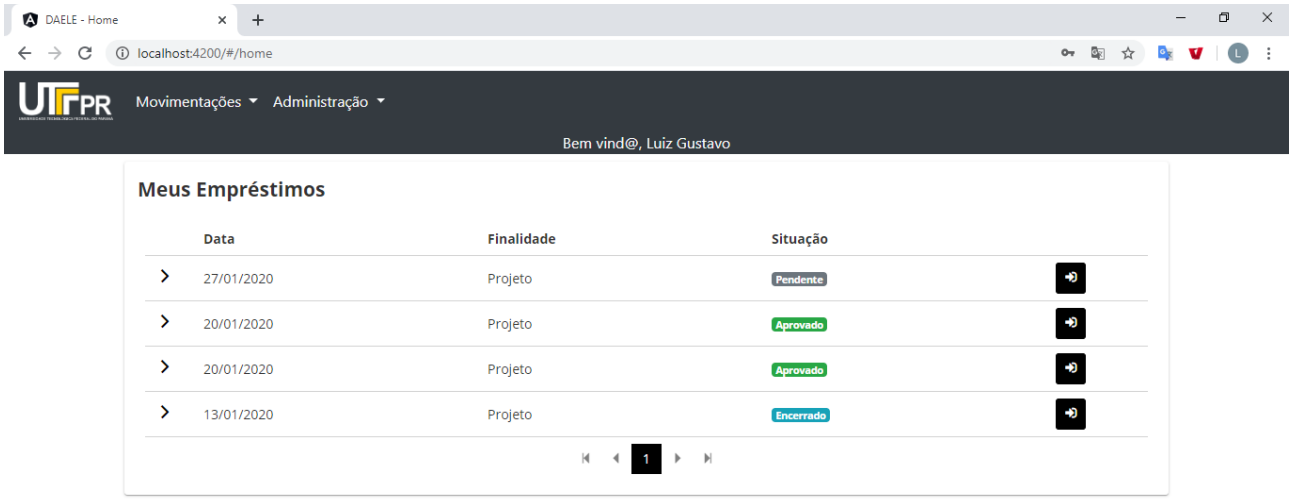
Fonte: Autoria própria

Ambas as telas são apresentadas no formato *modal*, isto é, a página de autenticação permanece ao fundo, inacessível, enquanto as outras telas são apresentadas e até que sejam concluídas ou fechadas.

Ao acessar a aplicação, a página exibida ao usuário será a *home*, assim como o menu posicionado no topo. O conteúdo da *home* será carregado de acordo com o tipo de atribuição do usuário que está acessando: usuários do tipo “aluno” ou “professor” visualizarão os seus empréstimos cadastrados, além de uma lista de menus mais reduzida, enquanto que os laboratoristas e administradores terão alguns *dashboards* de avaliação, como a lista de usuários pendentes, os empréstimos pendentes e os equipamentos com estoque próximo do fim. Quanto aos menus, todos da aplicação ficarão disponíveis.

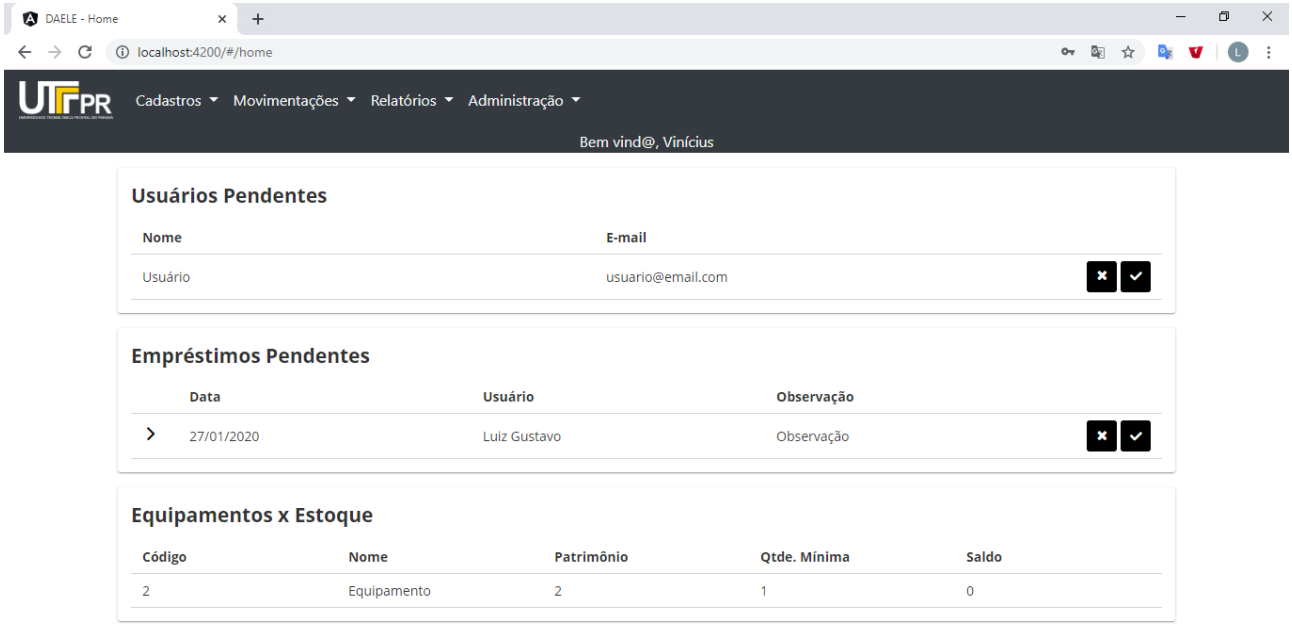
A Figura 6 exibe a *home* para os alunos e professores, enquanto a Figura 7 mostra a *home* dos laboratoristas e administradores.

Figura 6: Página *Home* dos alunos e professores



Fonte: Autoria própria

Figura 7: Página *Home* dos administradores e laboratoristas



Fonte: Autoria própria

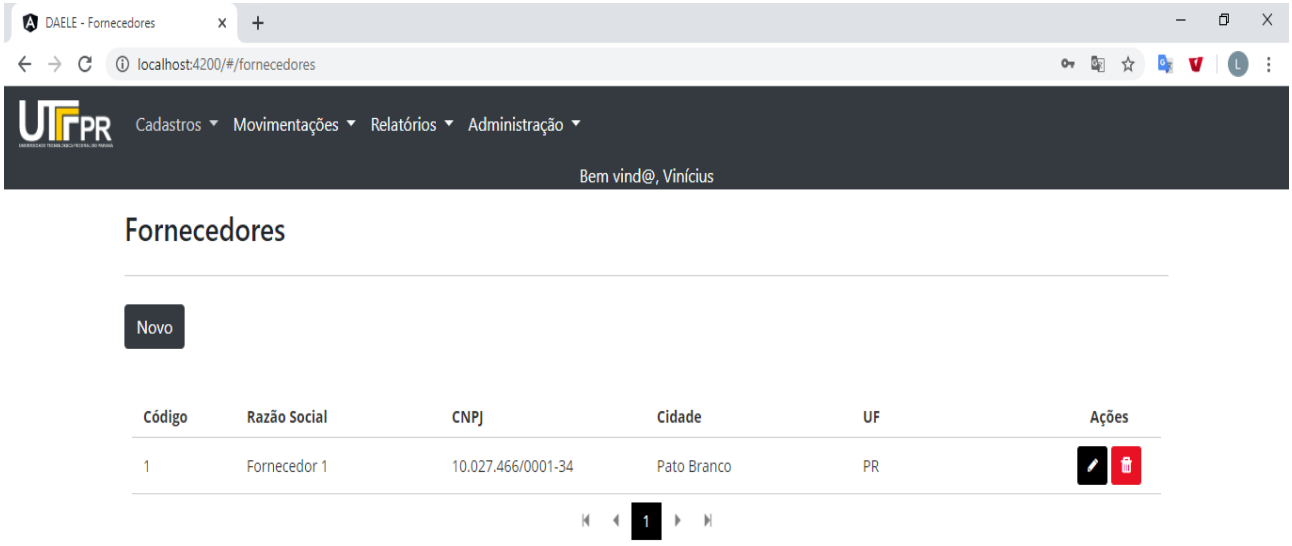
Para acessar os demais recursos do *software*, os usuários deverão navegar pelos menus, sendo que cada item de menu apresenta uma tela de listagem, seguida de um formulário para manutenção dos dados. A exceção fica para a rotina de alteração de senha, que apenas apresenta um *modal* para inserção da nova senha, seguida de um *logout*.

Todas as listagens presentes na aplicação são carregadas seguindo o padrão *lazy load* e com registros paginados, isto é, limitando as consultas em valores estabelecidos (para essa aplicação, definiu-se que as consultas retornarão, no máximo, dez registros por requisição). Essa opção foi adotada para evitar gargalos de desempenho e melhorar a experiência do usuário. Ainda, será por meio das listagens que as opções de edição e remoção de registros ficarão disponíveis.

No que diz respeito aos formulários, todos foram estruturados conforme a necessidade de cada cadastro, com validações padrão e customizadas em seus campos. O modelo de estrutura varia entre *reactive form* e *template-driven form*, sendo que todo cadastro manipula uma entidade do banco de dados e só pode ser salvo se atender a todas as validações especificadas.

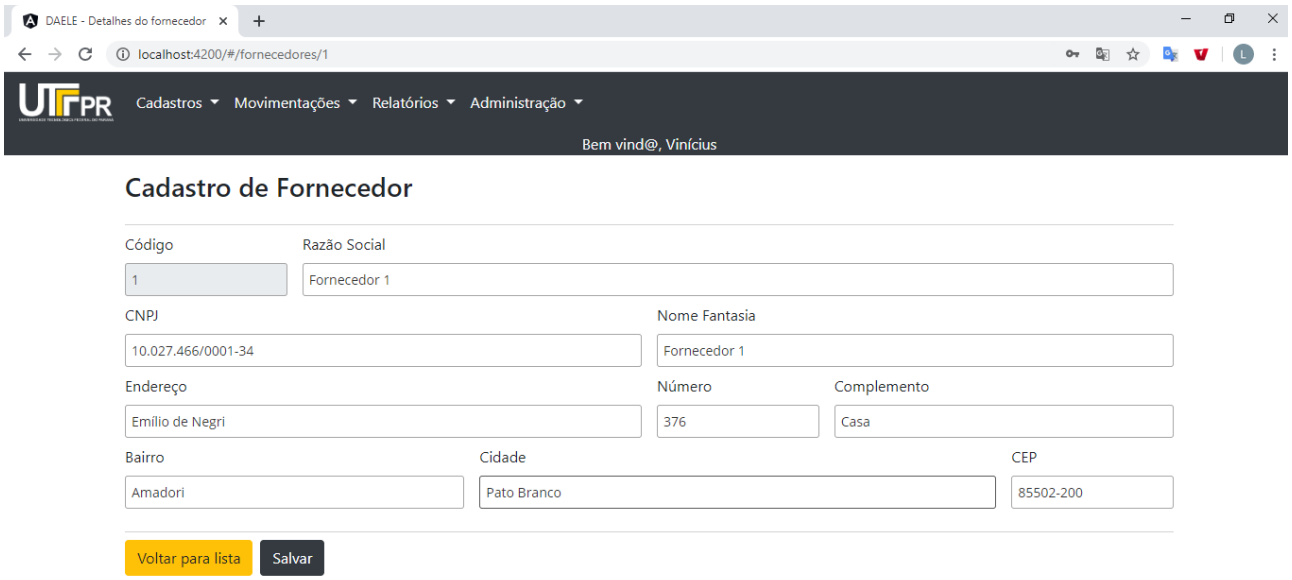
Ao clicar sobre o menu “Cadastros” (apenas laboratoristas e administradores têm esse menu visível), duas opções são exibidas: o cadastro de fornecedores e o cadastro de equipamentos. Para o cadastro de fornecedores, o resultado da listagem e do formulário são apresentados nas Figuras 8 e 9.

Figura 8: Página da lista de fornecedores



Fonte: Autoria própria

Figura 9: Página de cadastro de fornecedores

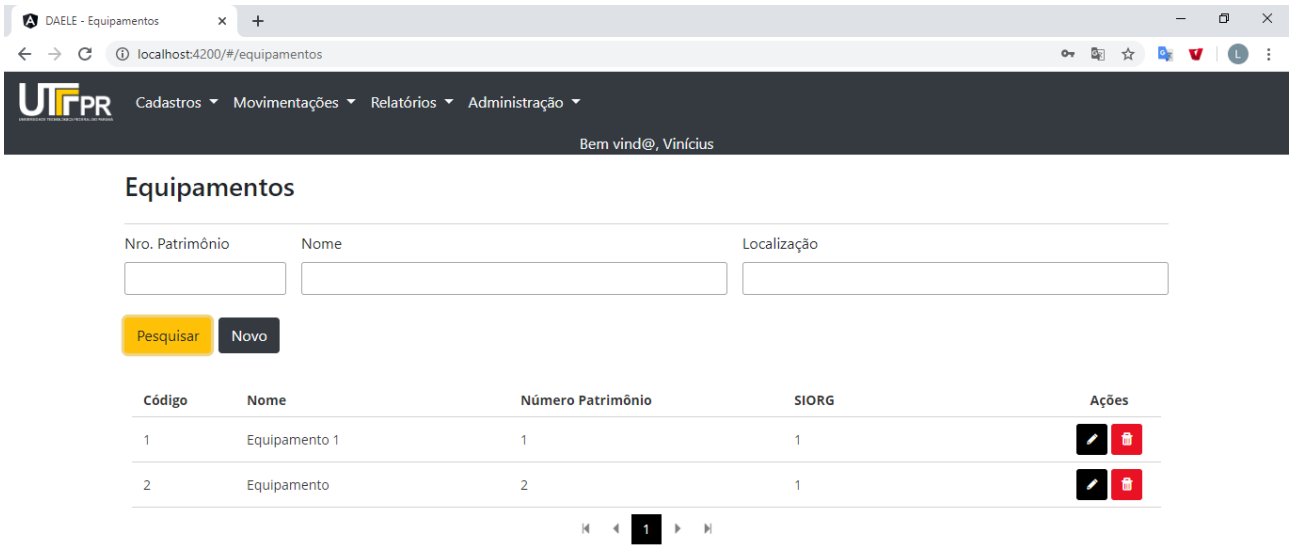


Fonte: Autoria própria

Como pode ser visto na Figura 9, o campo que representa o código do fornecedor não pode ser editado no formulário de manutenção, visto que ele é gerenciado apenas pela aplicação. Esse comportamento se repetirá em todos os outros cadastros.

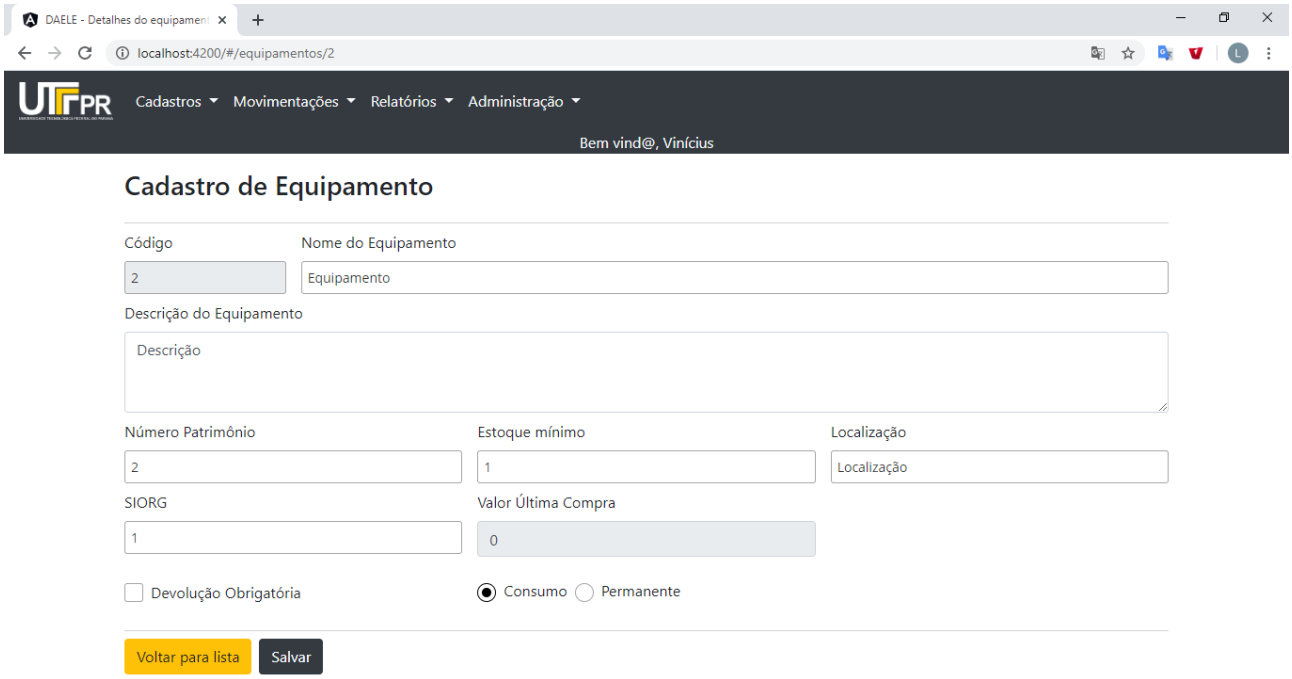
A outra opção de cadastro existente é a de equipamentos. Ao acessá-la, o usuário visualizará a página conforme as Figuras 10 e 11. No caso da listagem de equipamentos, além do carregamento ser *lazy load* e paginado, ele dependerá de uma ação de filtro explícita do usuário, visto que o número de registros pode ser grande.

Figura 10: Página da lista de equipamentos



Fonte: Autoria própria

Figura 11: Página de cadastro de equipamentos

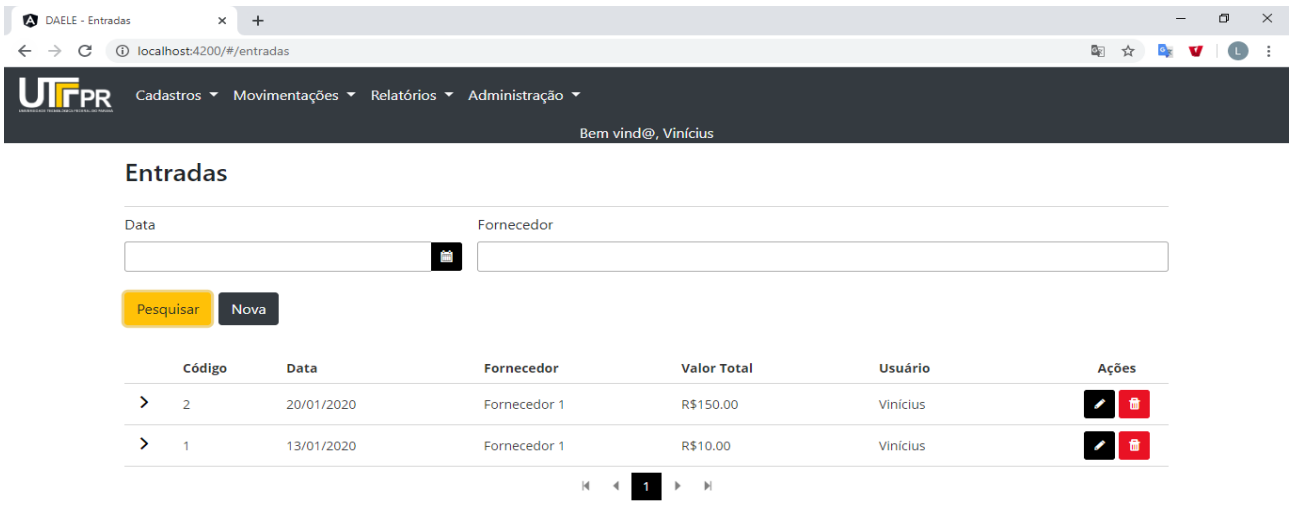


Fonte: Autoria própria

Clicando sobre o menu “Movimentações”, os usuários terão disponíveis três opções: entradas, empréstimos e baixas de estoque. Com relação às entradas e baixas de estoque, apenas os laboratoristas e administradores podem visualizá-las. A opção de empréstimos é acessível para todos.

Caso o usuário opte por acessar a página de entradas de estoque, ele visualizará a tela conforme a Figura 12.

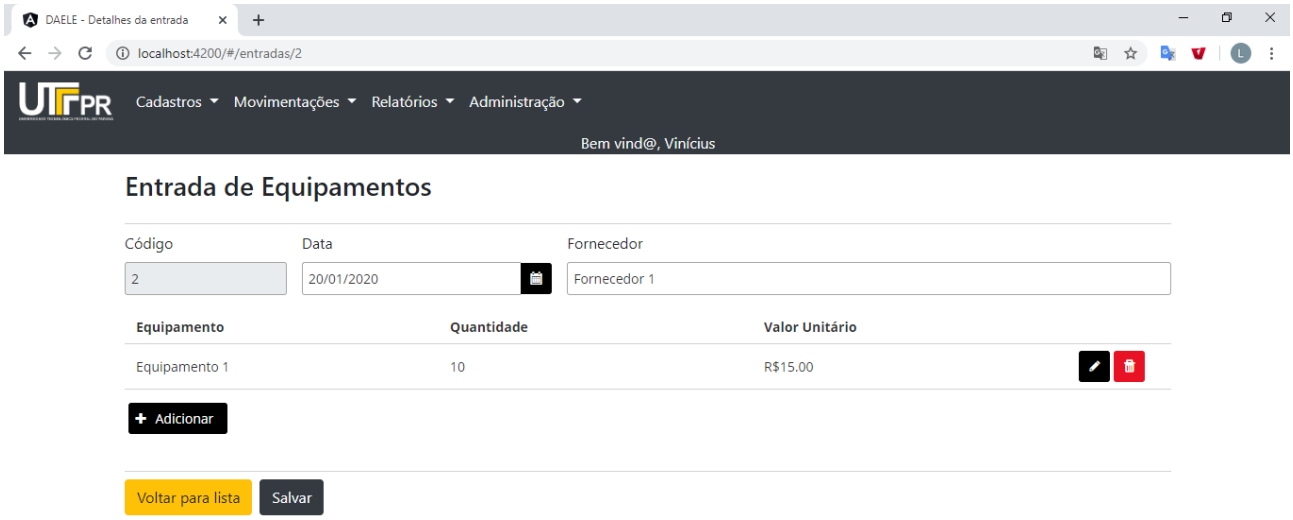
Figura 12: Página da lista de entradas



Fonte: Autoria própria

Semelhante ao que acontece com a lista de equipamentos, as entradas também devem ser filtradas antes de serem exibidas. O campo “Fornecedor” é do tipo *autocomplete*, isto é, a medida em que o usuário digita nele, as opções de fornecedores vão sendo listadas para seleção. O formulário associado a esse cadastro é exibido na Figura 13.

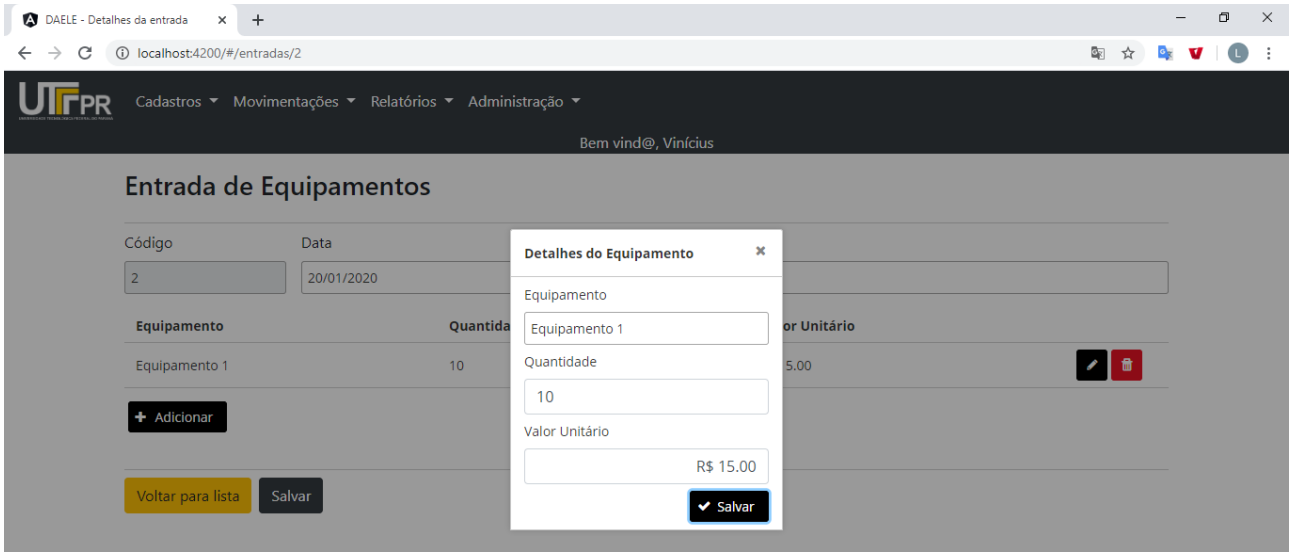
Figura 13: Página de cadastro de entradas



Fonte: Autoria própria

No formulário de entrada, o campo “Fornecedor” também é do tipo *autocomplete* e o botão “Adicionar” revela ao usuário um outro formulário para inserção dos equipamentos. Essa tela é exibida no formato *modal* e é representada pela Figura 14. Ao salvar, a tabela de equipamentos presente no formulário de entrada é atualizada.

Figura 14: Página de lançamento de equipamentos nas entradas

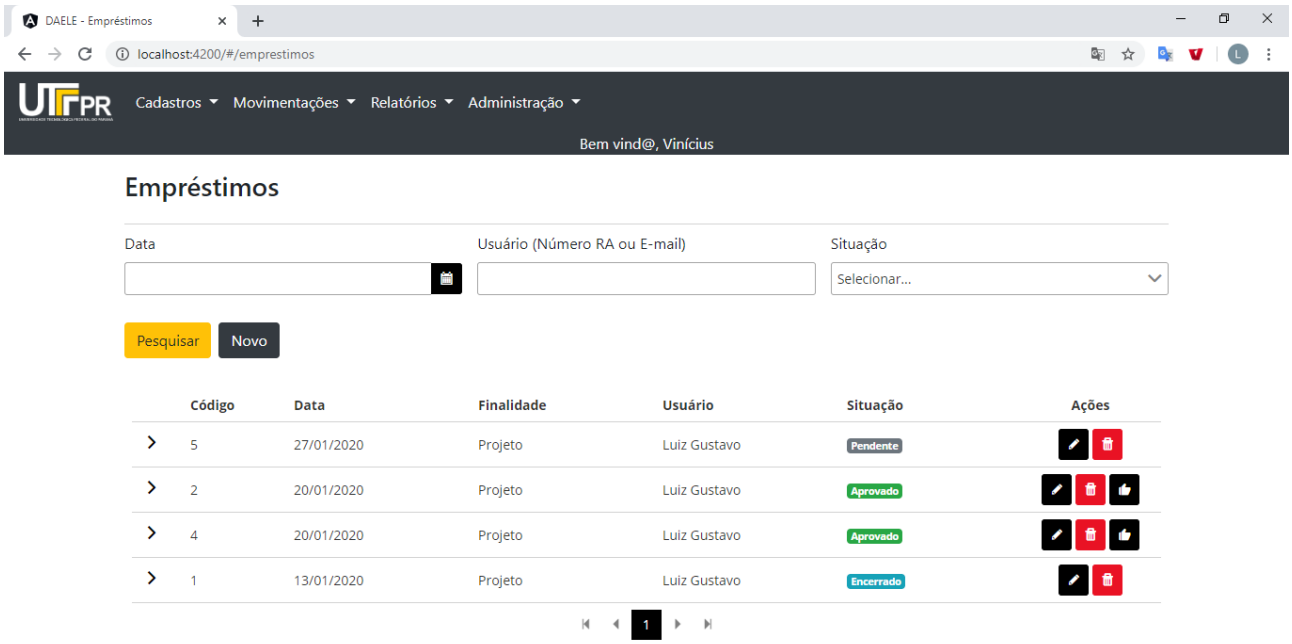


Fonte: Autoria própria

Com relação opção de menu “Empréstimos”, seu comportamento varia conforme o tipo de usuário. Os laboratoristas e administradores visualizam uma página de listagem com filtro, enquanto os alunos e professores são redirecionados para o formulário de cadastro. Esse comportamento foi adotado principalmente para evitar a redundância de ações no sistema, já que alunos e professores têm à disposição a lista de seus empréstimos na *home* da aplicação. Inclusive, a descrição desse menu varia entre “Empréstimos” e “Novo Empréstimo”.

A Figura 15 apresenta a listagem de empréstimos

Figura 15: Página de lista de empréstimos

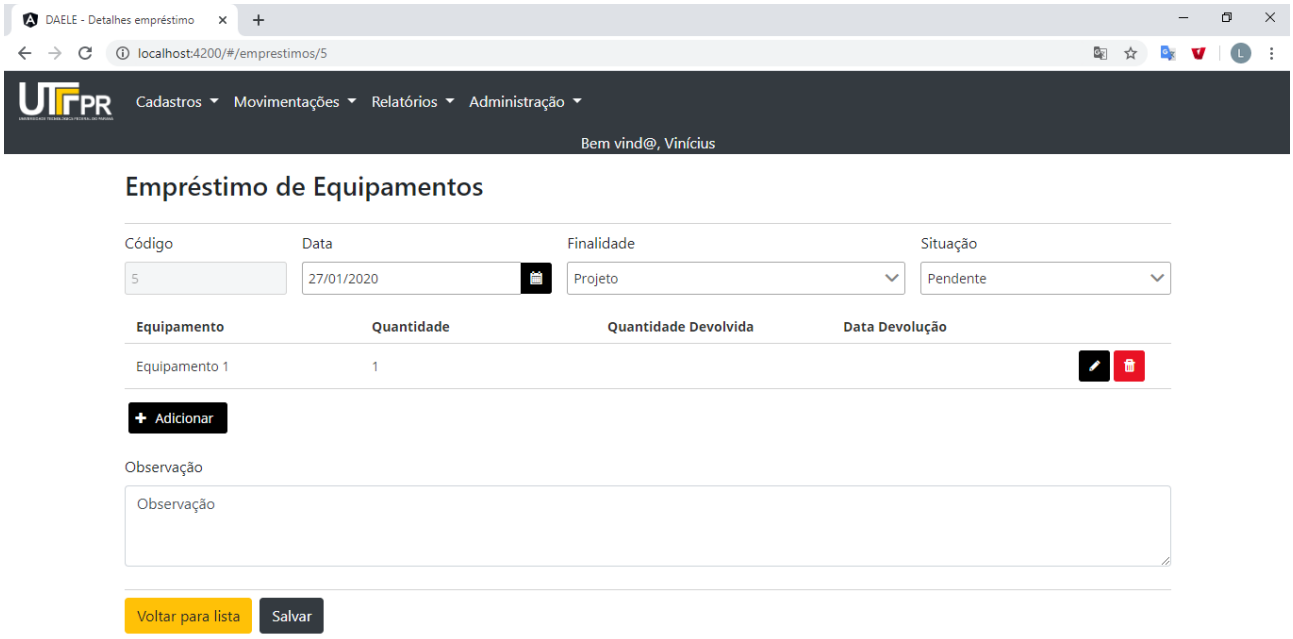


Fonte: Autoria própria

Na listagem, poderá o usuário realizar, além das opções tradicionais de edição e exclusão, o encerramento do empréstimo. Tal botão fica disponível apenas se o empréstimo estiver com a situação “aprovado”.

O formulário de cadastro de um empréstimo é apresentado na Figura 16.

Figura 16: Página de cadastro de empréstimos



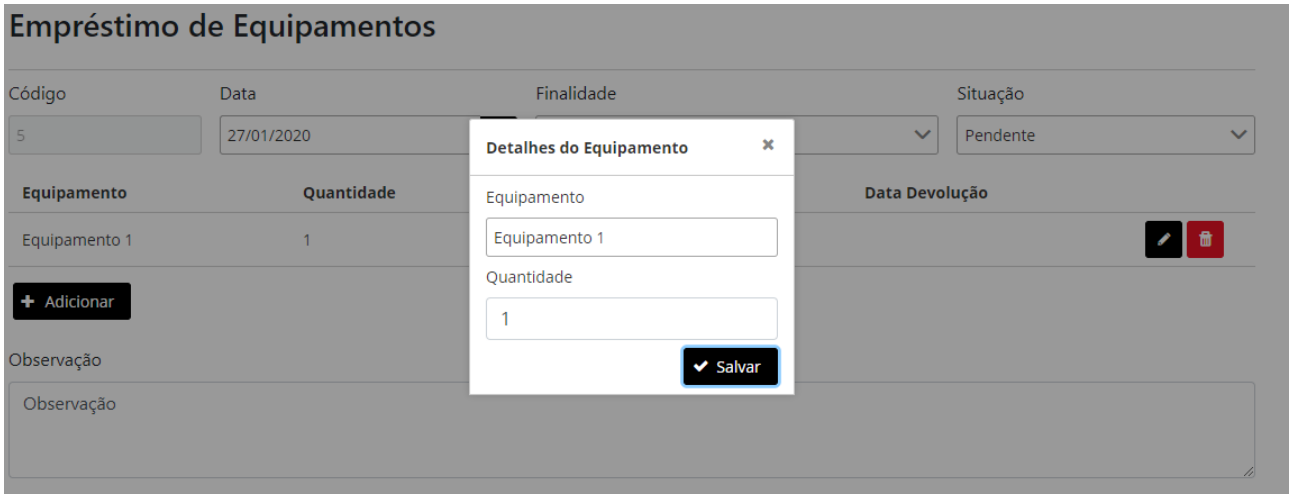
Fonte: Autoria própria

Este formulário possui comportamentos que variam conforme o tipo do usuário e a situação atual do empréstimo. As principais características são listadas a seguir:

- Alunos e professores visualizam o campo “Situação” desabilitado, enquanto laboratoristas e administradores têm permissão para editá-lo.
- Um empréstimo só pode ser editado se estiver com a situação “pendente”.
- Um aluno ou professor não pode visualizar um empréstimo de outro aluno ou professor. Apenas laboratoristas e administradores têm essa permissão.
- Para manter a consistência, as opções do campo “Situação” variam conforme a situação atual do empréstimo. Com isso evita-se, por exemplo, que um empréstimo passe da situação “aprovado” para “reprovado”, ou de “reprovado” para “pendente” novamente.

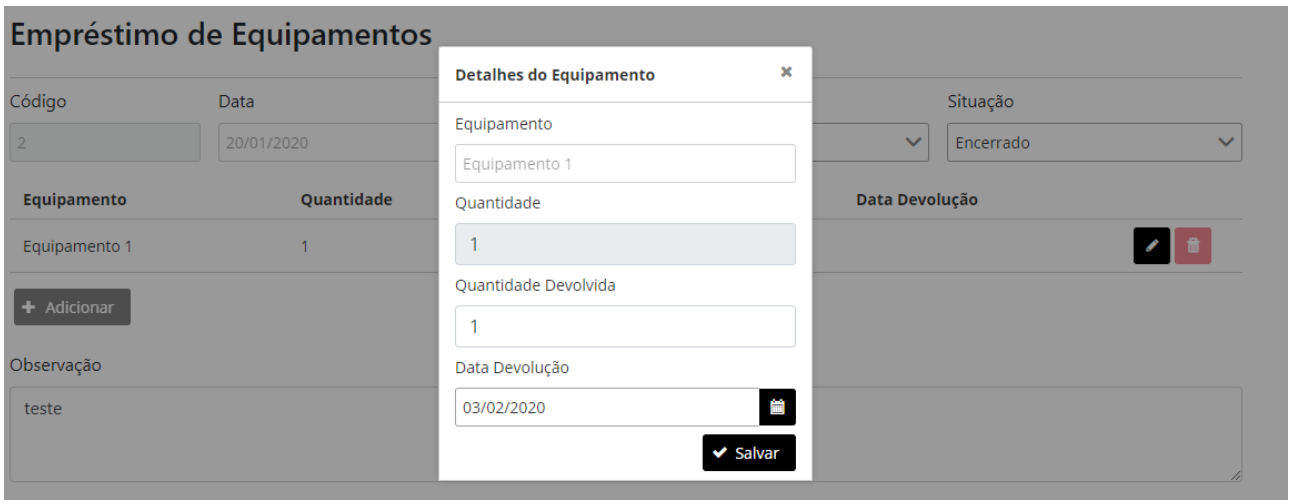
Com relação ao lançamento dos equipamentos no empréstimo, o *software* apresenta um formulário que varia conforme a situação do empréstimo: se estiver “pendente” ou “aprovado”, o formulário fica conforme a Figura 17, porém se o empréstimo estiver durante um encerramento, a Figura 18 representa a tela do sistema.

Figura 17: Página de lançamento de equipamentos nos empréstimos



Fonte: Autoria própria

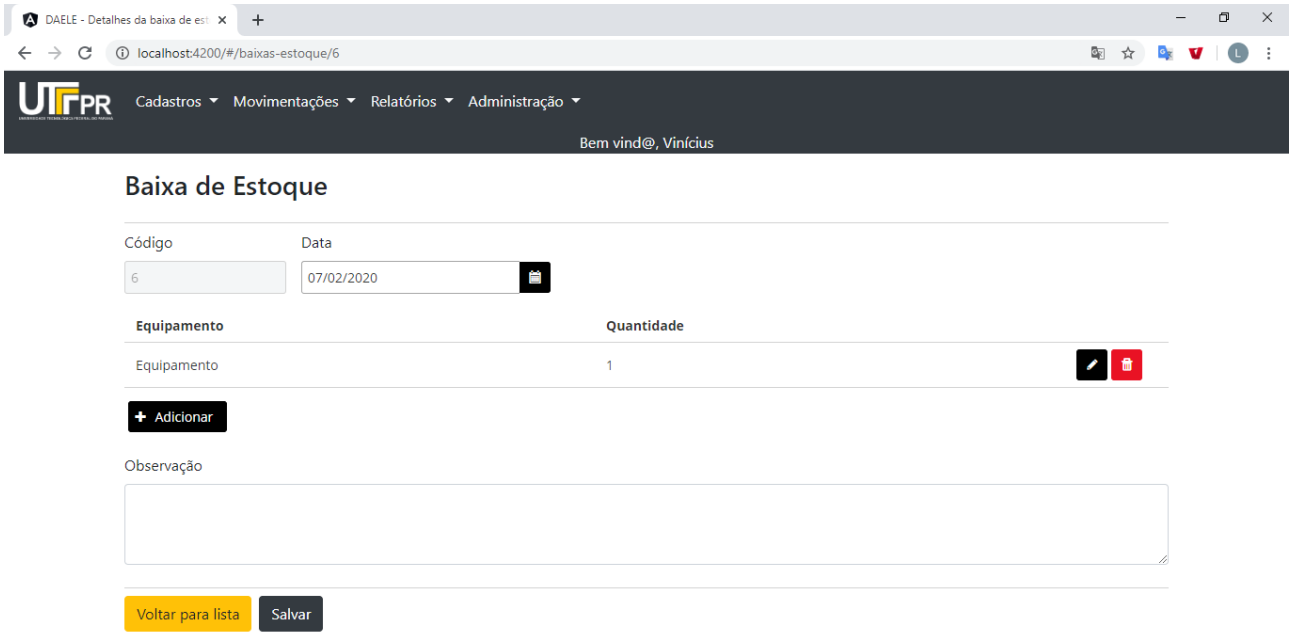
Figura 18: Página de lançamento de equipamentos durante encerramento de empréstimo



Fonte: Autoria própria

Outra movimentação de saída que existe no *software* é a baixa de estoque. Ela deve ser utilizada quando determinados equipamentos não serão mais disponibilizados para empréstimos devido a falhas, perdas e afins. O formulário para preenchimento segue o mesmo modelo da página de empréstimos, porém com campos a menos, e é apresentado na Figura 19.

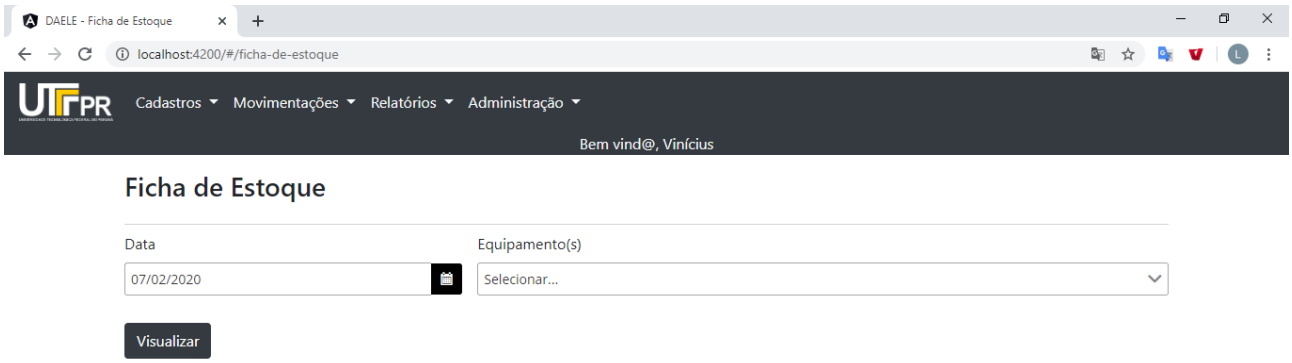
Figura 19: Página de cadastro de baixas de estoque



Fonte: Autoria própria

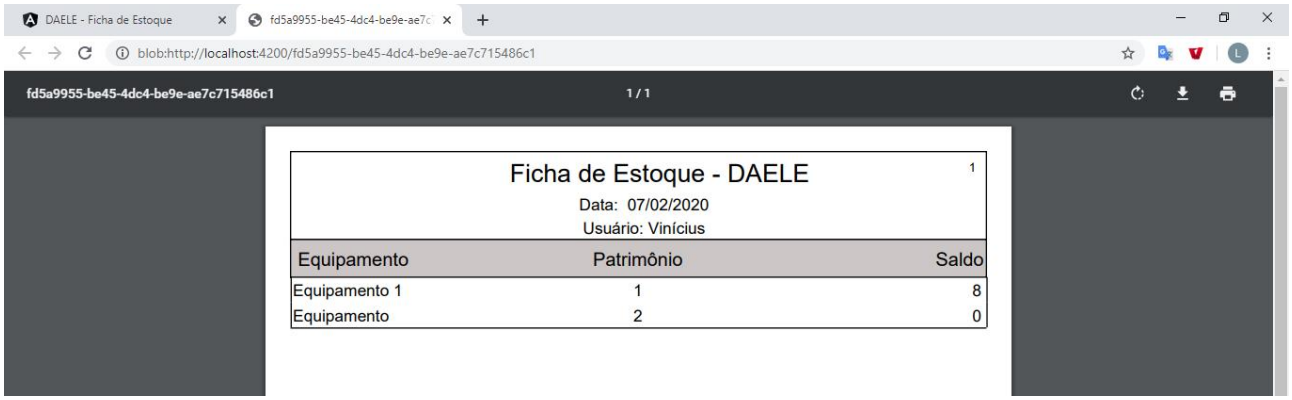
Para acompanhamento dos saldos dos equipamentos, o software disponibiliza um relatório denominado “Ficha de Estoque”, acessível pelo menu “Relatórios” (apenas para laboratoristas e administradores). Ao visualizar a página, o usuário deve inserir a data base para consulta e os itens que deseja analisar. A Figura 20 demonstra o estado inicial da página e a Figura 21 apresenta o relatório gerado.

Figura 20: Página do relatório de ficha de estoque



Fonte: Autoria própria

Figura 21: Página com o resultado do relatório de ficha de estoque

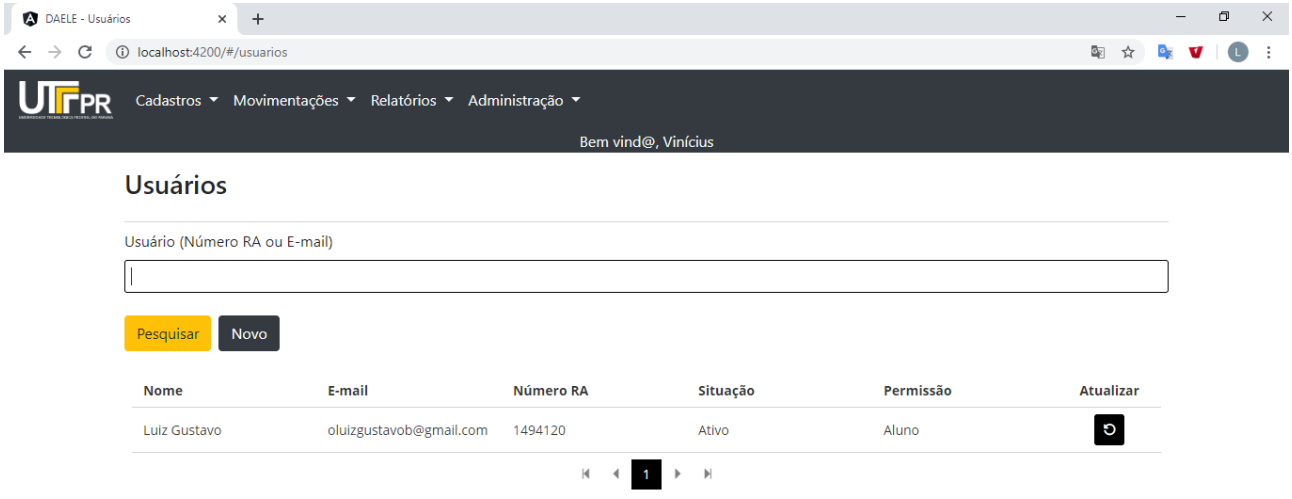


Fonte: Autoria própria

A respeito da manutenção dos usuários, a aplicação disponibiliza, através do menu "Administração", a opção para esse tipo de cadastro. Esse cadastro, diferente dos demais, não permite que registros sejam excluídos, apenas inseridos. Para as situações de alteração, o *software* limita essa ação apenas ao campo "ativo", podendo um laboratorista ou administrador alterar esse *status* de um outro usuário através de um botão na listagem.

Ao abrir o menu e carregar os dados em tela, o usuário perceberá que seu registro não será carregado na página. Isso foi definido para que um usuário não consiga ativar ou inativar a si mesmo. A Figura 22 mostra a funcionalidade em execução.

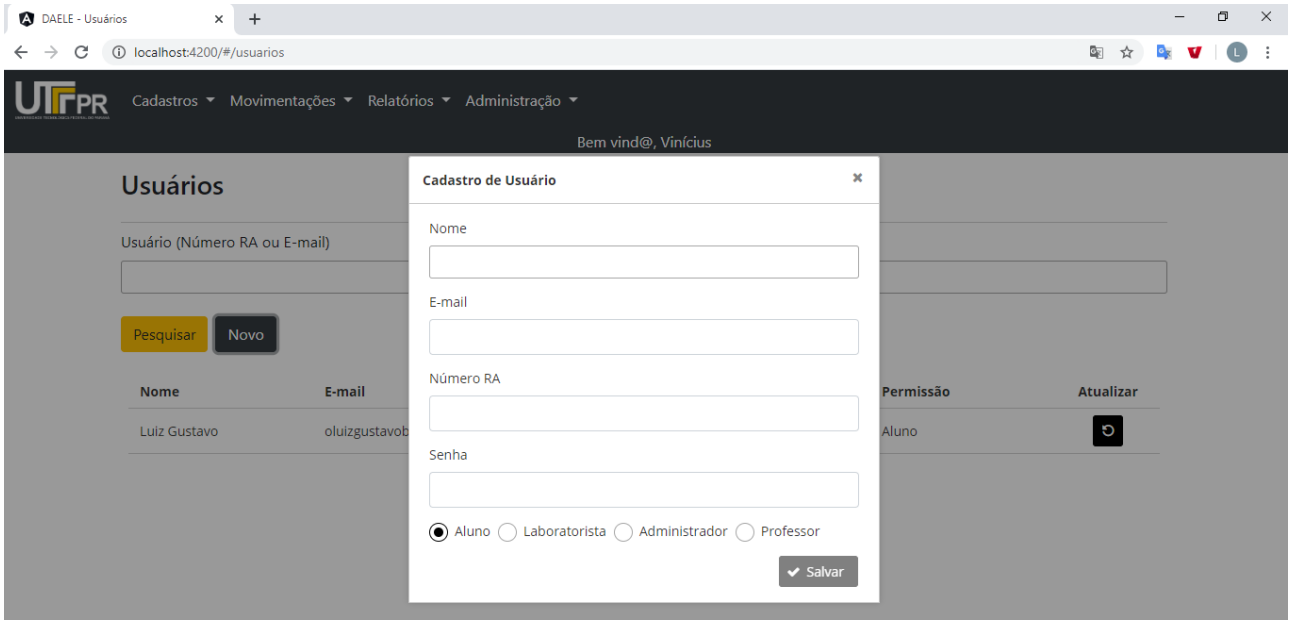
Figura 22: Página da lista de usuários da aplicação



Fonte: Autoria própria

Para cadastrar um novo usuário, o formulário de preenchimento é apresentado no formato *modal* sobre a listagem, conforme a Figura 23. Nome, e-mail, senha e o tipo do aluno são campos obrigatórios, enquanto o número do registro acadêmico só deve ser informado para os alunos.

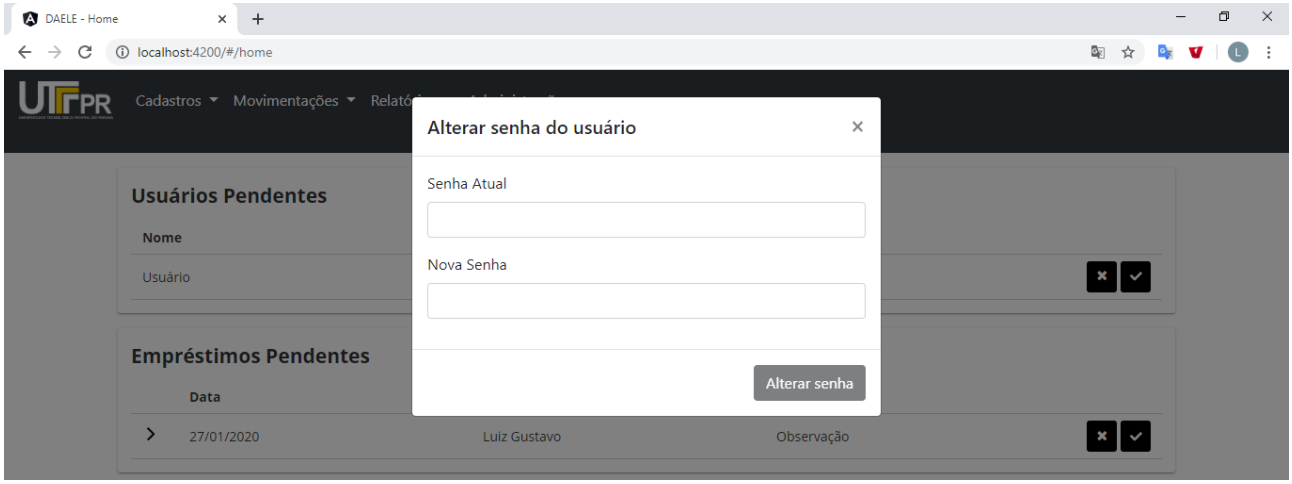
Figura 23: Página de cadastro de usuários



Fonte: Autoria própria

Por fim, poderá o usuário a qualquer momento redefinir sua senha de acesso na aplicação. Para isso, deverá acessar o menu “Administração”, escolher a opção “Alterar senha de acesso” e um formulário no formato *modal* é apresentado. Como valores, o *software* espera que o usuário informe a sua senha atual e a nova senha, conforme a Figura 24.

Figura 24: Página para alterar a senha do usuário



Fonte: Autoria própria

3.4 IMPLEMENTAÇÃO DO SISTEMA

A construção da aplicação foi dividida em dois projetos, sendo uma *Application Programming Interface* (API) Java no *back-end*, desenvolvida sob o *framework* Spring e o *client-side* implementado com Angular. Para a comunicação entre os dois projetos, foi estruturada uma arquitetura REST, com o *client-side* enviando requisições HTTP para consumir os recursos da API, por meio dos seus principais verbos: GET, POST, PUT, PATCH e DELETE. Nas requisições que necessitam de um conteúdo em seu corpo, deve ser utilizada a notação JSON para os dados.

O modelo de arquitetura REST foi escolhido pois tende a consumir menos recursos de rede e apresentar facilidades em sua utilização. Somado a isso, o *framework* Spring apresenta recursos que facilitam bastante a construção de uma API do gênero. Por outro lado, a adoção do Angular no *client-side* se deve ao fato de ele ser desenhado sob o conceito de SPA, isto é, *Single Page Application*, que consiste em um modelo no qual apenas uma página é criada e seu conteúdo é carregado sob demanda, variando de acordo com as recursos acessados pelo usuário. Essa estratégia, além de apresentar maior produtividade no que diz respeito ao consumo de banda, apresenta um *design* mais dinâmico e fluído ao usuário.

Com relação ao desenvolvimento da API, outra tecnologia de bastante importância adotada foi o Maven. Essa ferramenta foi utilizada para automatizar o gerenciamento das dependências do projeto, isto é, caso seja necessário utilizar recursos de bibliotecas ou *frameworks*, ele auxilia na inclusão dessas tecnologias auxiliares no projeto. Além disso, ele também possui recursos para geração de *builds*, mas essa funcionalidade não foi utilizada.

Para o *setup* da aplicação *back-end*, inicialmente foram configuradas as dependências do projeto em um arquivo chamado “pom.xml”. É esse arquivo que o Maven gerencia para identificar as dependências necessárias ao projeto. A Listagem 1 apresenta o conteúdo do “pom.xml” na API.

Listagem 1: Arquivo "pom.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.1.8.RELEASE</version>
		<relativePath />
	</parent>

	<groupId>br.edu.utfpr.pb</groupId>
```

```
	<artifactId>emprestimos-labs</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>jar</packaging>
	<name>emprestimos-labs</name>

	<properties>
		<java.version>1.8</java.version>
		<maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
		<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-cache</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-websocket</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
		</dependency>
		<dependency>
			<groupId>org.postgresql</groupId>
			<artifactId>postgresql</artifactId>
		</dependency>
		<dependency>
			<groupId>org.flywaydb</groupId>
			<artifactId>flyway-core</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
		<dependency>
			<groupId>org.apache.commons</groupId>
			<artifactId>commons-lang3</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-security</artifactId>
		</dependency>
		<dependency>
			<groupId>io.jsonwebtoken</groupId>
			<artifactId>jjwt</artifactId>
			<version>0.9.0</version>
		</dependency>
		<dependency>
			<groupId>org.apache.commons</groupId>
```

```
                <artifactId>commons-text</artifactId>
                <version>1.6</version>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-mail</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-configuration-processor</artifactId>
                <optional>true</optional>
            </dependency>
            <dependency>
                <groupId>net.sf.jasperreports</groupId>
                <artifactId>jasperreports</artifactId>
                <version>6.10.0</version>
            </dependency>
            <dependency>
                <groupId>net.sf.jasperreports</groupId>
                <artifactId>jasperreports-functions</artifactId>
                <version>6.10.0</version>
            </dependency>
            <dependency>
                <groupId>net.sf.jasperreports</groupId>
                <artifactId>jasperreports-fonts</artifactId>
                <version>6.10.0</version>
            </dependency>
        </dependencies>

        <build>
            <finalName>emprestimos-labs</finalName>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>

        <repositories>
            <repository>
                <id>jr-ce-releases</id>
                <url>http://jaspersoft.artifactoryonline.com/jaspersoft/jr-ce-
releases</url>
            </repository>
        </repositories>
</project>
```

Fonte: Autoria própria

Destaca-se na Listagem 1 a presença das dependências dos projetos Spring Boot, Spring Security e Spring Data JPA, que foram utilizados, respectivamente, para configuração da aplicação, para segurança e para manipulação com o banco de dados.

Outro arquivo importante nesse projeto é o “application.properties”. Normalmente, projetos que utilizam o Spring Boot tendem a ter configurações reduzidas, concentradas no próprio código Java por meio de anotações ou em arquivos bem específicos, sendo que o “application.properties” é

o principal deles. Nele, são indicadas uma série de informações, como o banco de dados da aplicação, além de valores a serem manipulados pelo código Java. A Listagem 2 mostra o arquivo “application.properties” do projeto da API.

Listagem 2: Arquivo "application.properties"

```
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.datasource.url=jdbc:postgresql://localhost:5432/emprestimoslabs
spring.datasource.username=postgres
spring.datasource.password=postgres
spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.platform=postgresql
spring.datasource.initialization-mode=always
spring.jpa.hibernate.ddl-auto=none
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.flyway.enabled=true
spring.flyway.baseline-on-migrate=true
spring.flyway.baseline-version=0.0.0
emprestimos-labs.seguranca.jwt-secret=emprestimos-labs-utfpr-secret
emprestimos-labs.seguranca.jwt-expiration=3600000
emprestimos-labs.seguranca.cors-origin-enable=http://localhost:4200
emprestimos-labs.email.host=smtp.gmail.com
emprestimos-labs.email.port=587
emprestimos-labs.email.sender=remetente@email.com
emprestimos-labs.email.username=remetente

emprestimos-labs.email.password=senha
```

Fonte: Autoria própria

Tendo o “pom.xml” e o “application.properties” configurados, o *setup* da API fica disponível, e quem faz isso é a classe do projeto que contém a anotação @SpringBootApplication. Ao executá-la, o Spring Boot utiliza um contêiner embarcado para subir a aplicação (nesse projeto, o Tomcat). Na Listagem 3 consta o código da classe responsável pelo *setup*.

Listagem 3: Classe do *setup* da API

```
package br.edu.utfpr.pb.emprestimoslabs;

/* imports omitidos */

@SpringBootApplication
@EnableConfigurationProperties(EmprestimosLabsApiProperty.class)
@EnableCaching
public class EmprestimosLabsApplication {

	public static void main(String[] args) {
		SpringApplication.run(EmprestimosLabsApplication.class, args);
	}
}
```

Fonte: Autoria própria

Como a API disponibiliza recursos que, na maior parte dos casos, realizam interações com o banco de dados, todas as tabelas do banco foram mapeadas para classes de modelo no código Java. Dessa maneira, o número de sentenças SQLs é reduzido, visto que o Spring Data JPA abstrai as principais operações *Create, Read, Update e Delete* (CRUD) e utiliza essas classes mapeadas para manipular as tabelas do banco de dados.

A Listagem 4 apresenta o mapeamento da tabela “saida” do banco de dados para a classe “Saida” do projeto.

Listagem 4: Classe "Saida" com mapeamento objeto-relacional

```java
package br.edu.utfpr.pb.emprestimoslabs.entity;

/* imports omitidos */

@Entity
@Table(name = "SAIDA")
@Data
@NoArgsConstructor
@EqualsAndHashCode(of = {"idSaida"})
public class Saida implements Serializable, EntidadeBD {

	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "saida_id")
	@SequenceGenerator(name="saida_id", sequenceName="saida_id", allocationSize=1)
	@Column(name = "idsaida")
	private Long idSaida;

	@Column(name = "datasaida", nullable = false)
	private LocalDate data;

	@Enumerated(EnumType.STRING)
	@Column(name = "tiposaida", nullable = false)
	private TipoSaida tipoSaida;

	@Enumerated(EnumType.STRING)
	@Column(name = "finalidade", nullable = false)
	private FinalidadeSaida finalidade;

	@Enumerated(EnumType.STRING)
	@Column(name = "situacao")
	private SituacaoSaida situacao = SituacaoSaida.PENDENTE;

	@ManyToOne
	@JoinColumn(name = "idusuario", referencedColumnName = "idusuario")
	private Usuario usuario;

	@Column(name = "observacao", length = 200, nullable = true)
	private String observacao;

	@OneToMany(mappedBy = "idSaidaItem.saida")
	@Valid
	private List<SaidaItem> itens = new ArrayList<>();

	@Override
```

```
	@JsonIgnore
	public Long getId() {
		return idSaida;
	}

	@PrePersist
	private void prePersist() {
		if (this.tipoSaida == TipoSaida.BAIXA) {
			this.finalidade = null;
			this.situacao = SituacaoSaida.ENCERRADA;
		}
	}
}
```

Fonte: Autoria própria

Para a manipulação efetiva no banco de dados, foram definidas duas camadas no projeto: a camada de repositórios e a camada de serviços. A primeira tem como objetivo abstrair as operações com o banco de dados, definindo os seus principais métodos (inserção, atualização, remoção e consultas por exemplo), enquanto a segunda contém as regras de negócio e as validações a serem aplicadas.

Na definição das classes da camada de repositórios, o Spring fornece uma interface base denominada de “JpaRepository” na qual todas as operações básicas de um CRUD já estão implementadas. Isso significa que, basta o desenvolvedor herdar sua classe de repositório dessa interface que ele fica dispensado de implementar as operações básicas. Somado a isso, destaca-se a necessidade da utilização da anotação @Repository, indicando ao Spring que essa deve ser uma classe com o ciclo de vida gerenciado por ele. A Listagem 5 apresenta a classe de repositório das saídas.

Listagem 5: Classe de repositório "SaidaData"

```
package br.edu.utfpr.pb.emprestimoslabs.data;

/* imports omitidos */

@Repository
public interface SaidaData extends JpaRepository<Saida, Long> {

	Page<Saida> findByUsuarioEqualsOrderByDataDesc(Usuario usuario, Pageable
pageable);

	@Query("SELECT s FROM Saida s WHERE s.tipoSaida = 'EMPRESTIMO' AND s.situacao =
'PENDENTE' ORDER BY s.idSaida")
	List<Saida> findEmprestimosPendentes();
}
```

Fonte: Autoria própria

As classes de serviço, além das funções já destacadas no texto, também servem para realizar a interação com os repositórios, e, assim como eles, são gerenciados pelo Spring. Para isso, a anotação @Service deve ser utilizada.

Sabendo que todos os serviços do projeto possuem comportamentos em comum, foi estruturada uma classe para ser utilizada como base de todos os serviços da aplicação. Ela recebeu o nome de “CrudServiceImpl” e possui o código exibido na Listagem 6.

Listagem 6: Classe base para os serviços do back-end

```
package br.edu.utfpr.pb.emprestimoslabs.service.generic;

/* imports omitidos */

public abstract class CrudServiceImpl<T, ID extends Serializable> implements
CrudService<T, ID> {

	protected abstract JpaRepository<T, ID> getData();

	@Override @Transactional(readOnly = false, propagation = Propagation.REQUIRED)
	public abstract T update(ID id, T entidade);

	@Override
	@Transactional(readOnly = false, propagation = Propagation.REQUIRED)
	public T save(T entidade) {
		return getData().save(entidade);
	}

	@Override
	@Transactional(readOnly = false, propagation = Propagation.REQUIRED)
	public void delete(ID id) {
		getData().deleteById(id);
	}

	@Override
	@Transactional(readOnly = false, propagation = Propagation.REQUIRED)
	public void delete(T entidade) {
		getData().delete(entidade);
	}

	@Override
	@Transactional(readOnly = true, propagation = Propagation.REQUIRED)
	public T findById(ID id) {
		return getData().findById(id).orElse(null);
	}

	@Override
	@Transactional(readOnly = true, propagation = Propagation.REQUIRED)
	public List<T> findAll() {
		return getData().findAll();
	}

	@Override
	@Transactional(readOnly = true, propagation = Propagation.REQUIRED)
	public List<T> findAllSorted(Sort sort) {
		return getData().findAll(sort);
	}
```

```
	@Override
	@Transactional(readOnly = true, propagation = Propagation.REQUIRED)
	public Page<T> findAllPaginated(Pageable pageable) {
		return getData().findAll(pageable);
	}

	@Override
	@Transactional(readOnly = true)
	public long count() {
		return getData().count();
	}
}
```

Fonte: Autoria própria

Já a Listagem 7 mostra a classe “SaidaService” codificada. Ela herda os métodos da classe “CrudServiceImpl”, além de implementar comportamentos específicos ao processo de saída de equipamentos (empréstimos e baixas de estoque).

Listagem 7: Classe de serviço "SaidaService"

```
package br.edu.utfpr.pb.emprestimoslabs.service;

/* imports omitidos */

@Service
public class SaidaService extends CrudServiceImpl<Saida, Long> {

	@Autowired
	private SaidaData saidaData;
	@Autowired
	private SaidaItemData saidaItemData;
	@Autowired
	private EstoqueService estoqueService;
	@Autowired
	private EquipamentoService itemService;
	@Autowired
	private EmailService emailService;
	@PersistenceContext
	private EntityManager manager;
	@Autowired
	private SimpMessagingTemplate message;

	@Override
	protected JpaRepository<Saida, Long> getData() {
		return saidaData;
	}

	@Transactional(readOnly = false, propagation = Propagation.MANDATORY)
	private void atualizarEstoque(Saida saida, boolean isUpdate) {
		for (SaidaItem itemSaida : saida.getItens()) {
			Equipamento item =
itemService.findById(itemSaida.getIdSaidaItem().getEquipamento().getIdEquipamento());
			switch (saida.getSituacao()) {
				case ENCERRADA:
					if (saida.getTipoSaida() == TipoSaida.BAIXA) {
						estoqueService.atualizarEstoqueSaida( item,
saida.getData(),
							itemSaida.getQuantidade() *
```

```
(isUpdate ? -1 : 1) );
                                } else {
                                        estoqueService.atualizarEstoqueEntrada( item,
itemSaida.getDataDevolucao(),
                                                                itemSaida.getQuantidadeDevolvida()
* (isUpdate ? -1 : 1) );
                                }
                                break;
                        case APROVADA:
                                estoqueService.atualizarEstoqueReservas( item,
saida.getData(),
                                                        itemSaida.getQuantidade() * (isUpdate ? -
1 : 1) );
                                break;

                        default:
                                break;
                        }
                }
        }

        @Override
        @Transactional(readOnly = false, propagation = Propagation.REQUIRED)
        public Saida update(Long id, Saida saida) {
                Saida saidaAtual = saidaData.findById(id).orElseThrow(() -> new
EmptyResultDataAccessException(1));

                Saida saidaAntiga = new Saida();
                BeanUtils.copyProperties(saidaAtual, saidaAntiga, "idSaida");

                if (!saida.getSituacao().equals(saidaAntiga.getSituacao())) {
                        atualizarEstoque(saidaAntiga, true);
                        Saida saidaAjuste = new Saida();
                        BeanUtils.copyProperties(saida, saidaAjuste, "idSaida", "situacao");
                        saidaAjuste.setSituacao(saidaAntiga.getSituacao());
                        atualizarEstoque(saidaAjuste, false);
                } else {
                        atualizarEstoque(saidaAntiga, true);
                }

                BeanUtils.copyProperties(saida, saidaAtual, "idSaida", "itens");
                saidaAtual = saidaData.save(saidaAtual);
                saidaAtual.getItens().clear();
                for (SaidaItem item: saida.getItens()) {
                        SaidaItem itemSaida = new SaidaItem();
                        itemSaida.getIdSaidaItem().setSaida(saidaAtual);

        itemSaida.getIdSaidaItem().setEquipamento(item.getIdSaidaItem().getEquipamento())
;
                        itemSaida.setQuantidade(item.getQuantidade());
                        itemSaida.setQuantidadeDevolvida(item.getQuantidadeDevolvida());
                        itemSaida.setDataDevolucao(item.getDataDevolucao());
                        saidaAtual.getItens().add(itemSaida);
                }
                saidaItemData.saveAll(saidaAtual.getItens());

                atualizarEstoque(saidaAtual, false);

                return saidaAtual;
        }
```

```
	@Override
	@Transactional(readOnly = false, propagation = Propagation.REQUIRED)
	public Saida save(Saida saida) {
		saida.setUsuario(UsuarioAutenticado.get());
		saida = saidaData.save(saida);
		for (SaidaItem item : saida.getItens()) {
			item.getIdSaidaItem().setSaida(saida);
		}
		saidaItemData.saveAll(saida.getItens());
		atualizarEstoque(saida, false);
		return saida;
	}

	@Override
	@Transactional(readOnly = false, propagation = Propagation.REQUIRED)
	public void delete(Long id) {
		Saida saida = saidaData.findById(id).orElseThrow(() -> new
RuntimeException("Saída não encontrada"));
		this.delete(saida);
	}

	@Override
	@Transactional(readOnly = false, propagation = Propagation.REQUIRED)
	public void delete(Saida saida) {
		for (SaidaItem item : saida.getItens()) {
			saidaItemData.delete(item);
		}
		super.delete(saida);
		atualizarEstoque(saida, true);
		if (!saida.getSituacao().equals(SituacaoSaida.REPROVADA)) {
			message.convertAndSend(Queue.ATUALIZAR_ESTOQUE, "update");
		}
	}

	@Transactional(readOnly = false, propagation = Propagation.REQUIRED)
	public void updateSituacao(Long id, SituacaoSaida situacao) {
		Saida saida = saidaData.findById(id).orElseThrow(() -> new
RuntimeException("Saída não encontrada"));
		saida.setSituacao(situacao);

		if (situacao.equals(SituacaoSaida.ENCERRADA)) {
			saida.getItens().forEach(item -> {
				item.setQuantidadeDevolvida(item.getQuantidade());
				item.setDataDevolucao(LocalDate.now());
			});
		}

		saida = saidaData.save(saida);
		atualizarEstoque(saida, false);

		message.convertAndSend(String.format(Queue.NOTIFICA_USUARIO,
saida.getUsuario().getEmail()), "emprestimo");
		message.convertAndSend(Queue.ATUALIZAR_ESTOQUE, "update");

		switch (situacao) {
			case APROVADA:
				emailService.enviarAprovacaoDeEmprestimo(saida);
				break;
			case REPROVADA:
```

```
                        emailService.enviarReprovacaoDeEmprestimo(saida);
                        break;
                default:
                        break;
                }
        }

        @Transactional(readOnly = true, propagation = Propagation.REQUIRED)
        public Page<SaidaDto> getEmprestimosDoUsuario(Pageable pageable) {
                return
saidaData.findByUsuarioEqualsOrderByDataDesc(UsuarioAutenticado.get(), pageable)
                        .map((saida) -> new SaidaDto(saida));
        }

        @Transactional(readOnly = true, propagation = Propagation.REQUIRED)
        public List<SaidaDto> getEmprestimosPendentes() {
                return saidaData.findEmprestimosPendentes()
                        .stream()
                        .map((saida) -> new SaidaDto(saida))
                        .collect(Collectors.toList());
        }

        @Transactional(readOnly = true, propagation = Propagation.REQUIRED)
        public Page<SaidaDto> findByDataAndUsuarioAndSituacao(SaidaFiltro saidaFiltro,
Pageable pageable) {
                List<SituacaoSaida> situacoes = null;

                if (saidaFiltro.getSituacao() == null) {
                        situacoes = Arrays.asList(
                                        SituacaoSaida.PENDENTE,
                                        SituacaoSaida.APROVADA,
                                        SituacaoSaida.ENCERRADA,
                                        SituacaoSaida.REPROVADA);
                } else {
                        situacoes = Arrays.asList(saidaFiltro.getSituacao());
                }

                String qry = "SELECT s FROM Saida s WHERE s.tipoSaida = 'EMPRESTIMO'";

                if (saidaFiltro.getData() != null) {
                        qry += " AND s.data = :data";
                }

                if (saidaFiltro.getNrora() != null && saidaFiltro.getNrora() > 0) {
                        qry += " AND s.usuario.nrora = :nrora";
                }

                qry += " AND s.situacao in :situacao";
                qry += " ORDER BY s.data DESC";

                TypedQuery<Saida> query = manager.createQuery(qry, Saida.class);

                if (qry.contains(":data")) {
                        query.setParameter("data", saidaFiltro.getData());
                }

                if (qry.contains(":nrora")) {
                        query.setParameter("nrora", saidaFiltro.getNrora());
                }
```

```
            query.setParameter("situacao", situacoes);

            List<Saida> emprestimos = query.getResultList();

            int paginaAtual = pageable.getPageNumber();
            int totalRegistrosPorPagina = pageable.getPageSize();
            int primeiroRegistroDaPagina = paginaAtual * totalRegistrosPorPagina;

            List<SaidaDto> emprestimosDtoComPaginacao = query
                        .setFirstResult(primeiroRegistroDaPagina)
                        .setMaxResults(totalRegistrosPorPagina)
                        .getResultList()
                        .stream()
                        .map(saida -> new SaidaDto(saida))
                        .collect(Collectors.toList());

            return new PageImpl<SaidaDto>(emprestimosDtoComPaginacao, pageable,
emprestimos.size());
      }

      @Transactional(readOnly = true, propagation = Propagation.REQUIRED)
      public Page<SaidaDto> findBaixasDeEstoque(Pageable pageable) {
            String qry = "SELECT s FROM Saida s WHERE s.tipoSaida = 'BAIXA' ORDER BY
s.idSaida";

            TypedQuery<Saida> query = manager.createQuery(qry, Saida.class);

            List<Saida> baixas = query.getResultList();

            int paginaAtual = pageable.getPageNumber();
            int totalRegistrosPorPagina = pageable.getPageSize();
            int primeiroRegistroDaPagina = paginaAtual * totalRegistrosPorPagina;

            List<SaidaDto> baixasDtoComPaginacao = query
                        .setFirstResult(primeiroRegistroDaPagina)
                        .setMaxResults(totalRegistrosPorPagina)
                        .getResultList()
                        .stream()
                        .map(saida -> new SaidaDto(saida))
                        .collect(Collectors.toList());

            return new PageImpl<SaidaDto>(baixasDtoComPaginacao, pageable,
baixas.size());
      }
}
```

Fonte: Autoria própria

Com os serviços e repositórios criados, foram implementados os controladores, que fazem parte da camada responsável por receber as requisições da aplicação cliente, interpretá-las, redirecioná-las às devidas classes de serviço e encaminhar as respostas de volta para a aplicação que as consome. Um controlador é identificado pela anotação @RestController e seus métodos devem expôr os recursos aos quais estão aptos a responder, bem como os verbos HTTP que devem ser utilizados.

A Listagem 8 demonstra o código da classe controladora das saídas.

Listagem 8: Classe controladora "SaidaController"

```
package br.edu.utfpr.pb.emprestimoslabs.controller;

/* imports omitidos */

@RestController
@RequestMapping("/saidas")
public class SaidaController {

	@Autowired
	private SaidaService saidaService;
	@Autowired
	private EquipamentoService equipamentoService;
	@Autowired
	private ApplicationEventPublisher publisher;
	@Autowired
	private SimpMessagingTemplate message;

	@GetMapping(value = { "", "/" })
	@PreAuthorize("hasRole('ADMIN') or hasRole('LABORATORISTA')")
	public ResponseEntity<List<SaidaDto>> findAll() {
		List<Saida> saidas = saidaService.findAll();
		List<SaidaDto> saidasDto = saidas
				.stream()
				.map(saida -> new SaidaDto(saida))
				.collect(Collectors.toList());
		return ResponseEntity.ok().body(saidasDto);
	}

	@GetMapping("/page")
	@PreAuthorize("hasRole('ADMIN') or hasRole('LABORATORISTA')")
	public Page<SaidaDto> findAllPaginated(Pageable pageable) {
		Page<Saida> saidas = saidaService.findAllPaginated(pageable);
		Page<SaidaDto> saidasDto = saidas.map(saida -> new SaidaDto(saida));
		return saidasDto;
	}

	@GetMapping("/{id}")
	public ResponseEntity<SaidaDto> findById(@PathVariable("id") Long id) {
		Saida saida = saidaService.findById(id);
		return ResponseEntity.ok().body(new SaidaDto(saida));
	}

	@PostMapping
	public ResponseEntity<SaidaDto> insert(@RequestBody @Valid SaidaDto saidaDto,
HttpServletResponse response) {
		Saida saida = saidaDto.toEntity(equipamentoService);
		saida = saidaService.save(saida);
		if (saida.getSituacao().equals(SituacaoSaida.PENDENTE)) {
			message.convertAndSend(Queue.NOVOS_EMPRESTIMOS, "update");
		} else if (!saida.getSituacao().equals(SituacaoSaida.REPROVADA)) {
			message.convertAndSend(Queue.ATUALIZAR_ESTOQUE, "update");
		}
		publisher.publishEvent(new NovoRecurso(this, response, saida.getId()));
		return ResponseEntity.status(HttpStatus.CREATED).body(new
SaidaDto(saida));
	}
```

```
	@PutMapping("/{id}")
	public ResponseEntity<SaidaDto> edit(@PathVariable("id") Long id,
			@Valid @RequestBody SaidaDto saidaDto) {
		Saida saida = saidaDto.toEntity(equipamentoService);
		saida = saidaService.update(id, saida);
		if (!saida.getSituacao().equals(SituacaoSaida.REPROVADA)) {
			message.convertAndSend(Queue.ATUALIZAR_ESTOQUE, "update");
		}
		return ResponseEntity.ok(new SaidaDto(saida));
	}

	@DeleteMapping("/{id}")
	public void delete(@PathVariable("id") Long id) {
		saidaService.delete(id);
	}

	@GetMapping("/filter")
	@PreAuthorize("hasRole('ADMIN') or hasRole('LABORATORISTA')")
	public ResponseEntity<Page<SaidaDto>> findByDataAndUsuarioAndSituacao(SaidaFiltro
saidaFiltro, Pageable pageable) {
		Page<SaidaDto> emprestimos =
saidaService.findByDataAndUsuarioAndSituacao(saidaFiltro, pageable);
		return ResponseEntity.ok(emprestimos);
	}

	@PatchMapping("/{id}/situacao")
	@ResponseStatus(HttpStatus.NO_CONTENT)
	public void updateSituacao(@PathVariable("id") Long id, @RequestBody @NotNull
UpdateSituacaoSaidaForm situacaoForm) {
		SituacaoSaida situacao = SituacaoSaida.toEnum(situacaoForm.getSituacao());
		saidaService.updateSituacao(id, situacao);
	}

	@GetMapping("/meus-emprestimos")
	@PreAuthorize("hasRole('ALUNO') or hasRole('PROFESSOR')")
	public Page<SaidaDto> getEmprestimosDoUsuario(Pageable pageable){
		Page<SaidaDto> emprestimos =
saidaService.getEmprestimosDoUsuario(pageable);
		return emprestimos;
	}

	@GetMapping("/emprestimos-pendentes")
	@PreAuthorize("hasRole('ADMIN') or hasRole('LABORATORISTA')")
	public ResponseEntity<List<SaidaDto>> getEmprestimosPendentes() {
		List<SaidaDto> emprestimos = saidaService.getEmprestimosPendentes();
		return ResponseEntity.ok(emprestimos);
	}

	@GetMapping("/baixas")
	@PreAuthorize("hasRole('ADMIN') or hasRole('LABORATORISTA')")
	public ResponseEntity<Page<SaidaDto>> findBaixasDeEstoque(Pageable pageable) {
		Page<SaidaDto> baixas = saidaService.findBaixasDeEstoque(pageable);
		return ResponseEntity.ok(baixas);
	}
}
```

Fonte: Autoria própria

De maneira geral, a API é protegida para garantir que apenas os usuários da aplicação possam consumir os seus recursos. Isso significa que a sua utilização depende de uma autenticação prévia, isto é, a validação de um usuário e senha existentes. Ao realizar essa ação, a API retorna um *token* para o cliente que garante a sua autorização a determinados métodos por um tempo pré-determinado. Esse *token* deve ser armazenado localmente no lado cliente e sempre que a aplicação consumir algum *endpoint* da API, deve enviá-lo na requisição.

A Listagem 9 apresenta o código que valida o usuário e senha e gera o *token*.

Listagem 9: Classe que valida um usuário e gera um *token*

```
package br.edu.utfpr.pb.emprestimoslabs.security.filter;

/* imports omitidos */

public class AuthenticationFilter extends UsernamePasswordAuthenticationFilter {

	private AuthenticationManager manager;
	private JwtUtil jwtUtil;

	public AuthenticationFilter(AuthenticationManager manager, JwtUtil jwtUtil) {
		setRequiresAuthenticationRequestMatcher(new
AntPathRequestMatcher("/auth/**", "POST"));
		this.manager = manager;
		this.jwtUtil = jwtUtil;
		setAuthenticationFailureHandler(new JWTAuthenticationFailureHandler());
	}

	@Override
	public Authentication attemptAuthentication(HttpServletRequest request,
HttpServletResponse response)
			throws AuthenticationException {
		try {
			Credenciais credenciais = new
ObjectMapper().readValue(request.getInputStream(), Credenciais.class);
			UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(
					credenciais.getUsername(), credenciais.getPassword());
			return manager.authenticate(authentication);
		} catch (IOException ex) {
			throw new RuntimeException(ex);
		}
	}

	@Override
	protected void successfulAuthentication(HttpServletRequest request,
HttpServletResponse response, FilterChain chain,
			Authentication auth) throws IOException, ServletException {
		Usuario usuario = (Usuario) auth.getPrincipal();
		String token = jwtUtil.gerarToken(usuario);
		response.addHeader("Authorization", token);
	}

	public static class Credenciais {

		private String username;
		private String password;
```

```
		public Credenciais() {}

		public String getUsername() {
			return username;
		}

		public void setUsername(String username) {
			this.username = username;
		}

		public String getPassword() {
			return password;
		}

		public void setPassword(String password) {
			this.password = password;
		}
	}

	private class JWTAuthenticationFailureHandler implements
AuthenticationFailureHandler {

		@Override
		public void onAuthenticationFailure(HttpServletRequest request,
HttpServletResponse response,
				AuthenticationException exception) throws IOException,
ServletException {
			String mensagem = "";

			if (exception.getCause() instanceof InactiveUserException) {
				mensagem = exception.getMessage();
			} else {
				mensagem = "E-mail ou senha inválido(s)";
			}

			response.setStatus(401);
			response.setContentType("application/json");
			response.getWriter().append(json(mensagem));
		}

		private String json(String mensagem) {
			long date = new Date().getTime();
			return "[{\"timestamp\": " + date + ", " + "\"status\": 401, " +
"\"error\": \"Não autorizado\", "
					+ "\"mensagemUsuario\": \"" + mensagem + "\"}]";
		}
	}
}
```

Fonte: Autoria própria

A aplicação cliente recebe esse *token* e armazena-o em *local storage*. A Listagem 10 apresenta o código que solicita a autenticação de um usuário na API e em caso de sucesso, é solicitado ao *service* “UsuarioService” o armazenamento local do *token* gerado.

Listagem 10: Classe que solicita a autenticação de um usuário e armazena o *token*

```
/* imports omitidos */

@Injectable({
  providedIn: 'root'
})
export class AuthService {

  constructor(private httpClient: HttpClient, private usuarioService: UsuarioService) { }

  authenticate(username: string, password: string) {
    return this.httpClient.post(environment.api_auth, { username, password }, { observe: 'respons
e'} )
      .pipe(
        tap(resp => {
          const token = resp.headers.get('Authorization');
          this.usuarioService.setToken(token);
        })
      );
  }
}
```

Fonte: Autoria própria

O *token* é sempre enviado nas requisições HTTP para a API. Para lidar com esse cenário, foi criada uma classe que intercepta todas as requisições da aplicação cliente e adiciona a informação do *token*, além do tipo do conteúdo a ser enviado e consumido. O código dessa classe é exibido na Listagem 11.

Listagem 11: Classe que intercepta as requisições e adiciona o *token*

```
/* imports omitidos */

@Injectable({
  providedIn: 'root'
})
export class AuthInterceptorService implements HttpInterceptor {

  constructor(private router: Router, private tokenService: TokenService) { }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    if (this.tokenService.isTokenExpired()) {
      this.tokenService.removeToken();
      this.router.navigate(['/login']);
      return;
    }

    req = req.clone({
      headers: req.headers
        .append('Content-Type', 'application/json')
        .append('Accept', 'application/json')
    });

    if (this.tokenService.hasToken()) {
      const token = this.tokenService.getToken();
      req = req.clone({
        headers: req.headers.append('Authorization', 'Bearer ' + token)
      });
```

```
    }

    return next.handle(req);
  }
}
```

Fonte: Autoria própria

Para o código da Listagem 11 ter efeito no projeto, foi necessário configurar um *provider* na classe “AppModule”, conforme a Listagem 12.

Listagem 12: Classe que contém a configuração para o funcionamento do interceptador

```
/* imports omitidos */

@NgModule({
  declarations: [ AppComponent ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    CommonModule,
    ToastModule,
    MenuModule,
    LoadingModule,
    AppRoutingModule
  ],
  providers: [
    MessageService,
    {
      provide: HTTP_INTERCEPTORS,
      useClass: AuthInterceptorService,
      multi: true
    },
    {
      provide: ErrorHandler,
      useClass: ErrorHandlerService
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Fonte: Autoria própria

A fim de manter o projeto organizado, a comunicação da aplicação cliente com a API foi isolada em *services*, que possuem em suas estruturas componentes que lidam com requisições HTTP. Tais serviços tiveram seus comportamentos comuns abstraídos em uma classe base, facilitando a sua reutilização.

A Listagem 13 mostra a classe base dos *services*.

Listagem 13: Classe base dos *services* da aplicação cliente

```
/* imports omitidos */

export abstract class CrudService<T, ID> {
```

```
  constructor(protected url: string, protected http: HttpClient) {}

  protected getUrl(): string {
    return this.url;
  }

  save(id: ID, t: T): Observable<T> {
    if (id) {
      return this.edit(id, t);
    } else {
      return this.insert(t);
    }
  }

  insert(t: T): Observable<T> {
    const url = `${this.getUrl()}`;
    return this.http.post<T>(url, t);
  }

  edit(id: ID, t: T): Observable<T> {
    const url = `${this.getUrl()}/${id}`;
    return this.http.put<T>(url, t);
  }

  findAll(): Observable<T[]> {
    const url = `${this.getUrl()}`;
    return this.http.get<T[]>(url);
  }

  findAllPaginated(page: number, size: number): Observable<Page<T>> {
    const url = `${this.getUrl()}/page`;

    let params = new HttpParams();
    params = params.set('page', page.toString());
    params = params.set('size', size.toString());

    return this.http.get<Page<T>>(url, {params});
  }

  findById(id: ID): Observable<T> {
    const url = `${this.getUrl()}/${id}`;
    return this.http.get<T>(url);
  }

  delete(id: ID) {
    const url = `${this.getUrl()}/${id}`;
    return this.http.delete(url);
  }

  count() {
    const url = `${this.getUrl()}/count`;
    return this.http.get(url);
  }
}
```

Fonte: Autoria própria

A Listagem 14, por sua vez, apresenta o código do *service* que interage com o *endpoint* de saídas da API.

Listagem 14: Classe *service* que interage com o *endpoint* de saídas

```
/* imports omitidos */

@Injectable({
  providedIn: 'root'
})
export class SaidaService extends CrudService<Saida, number> {

  constructor(http: HttpClient,
              private messageService: MyMessageService) {
    super(environment.api_saidas, http);
  }

  findById(id: number): Observable<Saida> {
    const url = `${this.getUrl()}/${id}`;
    return this.http.get<Saida>(url).pipe(
      map(saida => {
        saida.data = moment(saida.data).toDate();
        saida.itens.map(item => {
          if (item.dataDevolucao) {
            item.dataDevolucao = moment(item.dataDevolucao).toDate();
          }
        });
        return saida;
      })
    );
  }

  findEmprestimosByFiltros(saidaFiltro: EmprestimoFiltro): Observable<Page<Saida>> {
    const url = `${this.getUrl()}/filter?`;
    let params = new HttpParams();
    params = params.set('page', saidaFiltro.page.toString());
    params = params.set('size', saidaFiltro.size.toString());

    if (saidaFiltro.data) {
      params = params.set('data', moment(saidaFiltro.data).format('YYYY-MM-DD'));
    }

    if (saidaFiltro.usuario) {
      params = params.set('nrora', saidaFiltro.usuario.nrora.toString());
    }

    if (saidaFiltro.situacao) {
      params = params.set('situacao', saidaFiltro.situacao.toString());
    }

    return this.http.get<Page<Saida>>(url, { params }).pipe(
      map((resp: Page<Saida>) => {
        resp.content.map(saida => {
          saida.data = moment(saida.data).toDate();
          saida.itens.map(item => {
            if (item.dataDevolucao) {
              item.dataDevolucao = moment(item.dataDevolucao).toDate();
            }
          });
        });
```

```
      return resp;
    })
  );
}

getEmprestimosDoAluno(page: number, size: number): Observable<Page<Saida>> {
  let params = new HttpParams();
  params = params.set('page', page.toString());
  params = params.set('size', size.toString());

  const url = `${this.getUrl()}/meus-emprestimos?`;
  return this.http.get<Page<Saida>>(url, { params }).pipe(
    map((resp: Page<Saida>) => {
      resp.content.map(saida => {
        saida.data = moment(saida.data).toDate();
        saida.itens.map(item => {
          if (item.dataDevolucao) {
            item.dataDevolucao = moment(item.dataDevolucao).toDate();
          }
        });
      });
      return resp;
    })
  );
}

getEmprestimosPendentes(): Observable<Saida[]> {
  const url = `${this.getUrl()}/emprestimos-pendentes`;
  return this.http.get<Saida[]>(url).pipe(
    map(saidas => {
      saidas.map(saida => {
        saida.data = moment(saida.data).toDate();
        saida.itens.map(item => {
          if (item.dataDevolucao) {
            item.dataDevolucao = moment(item.dataDevolucao).toDate();
          }
        });
      });
      return saidas;
    })
  );
}

updateSituacao(idSaida: number, situacao: SituacaoSaida) {
  const url = `${this.getUrl()}/${idSaida}/situacao`;
  const situacaoForm = {
    situacao: situacao.valueOf()
  };
  return this.http.patch(url, situacaoForm);
}

save(id: number, saida: Saida): Observable<Saida> {
  if (saida.tipoSaida === TipoSaida.baixa) {
    saida.situacao = SituacaoSaida.encerrada;
    saida.finalidadeSaida = null;
  }

  saida.data = moment(saida.data).add(environment.moment_fuso_horario, 'hours').toDate();
  saida.itens.map(item => {
    if (item.dataDevolucao) {
      item.dataDevolucao = moment(item.dataDevolucao).add(environment.moment_fuso_horario, 'hou
```

```
rs').toDate();
      }
    });

    if (id) {
      return super.edit(id, saida);
    } else {
      return super.insert(saida);
    }
  }

  getBaixasDeEstoque(page: number, size: number): Observable<Page<Saida>> {
    const url = `${this.getUrl()}/baixas`;
    let params = new HttpParams();
    params = params.set('page', page.toString());
    params = params.set('size', size.toString());
    return this.http.get<Page<Saida>>(url, {params}).pipe(
      map((resp: Page<Saida>) => {
        resp.content.map(saida => saida.data = moment(saida.data).toDate());
        return resp;
      })
    );
  }

  podeEncerrarEmprestimo(saida: Saida): boolean {
    let retorno = true;

    for (const item of saida.itens) {
      if (!item.quantidadeDevolvida || !item.dataDevolucao) {
        retorno = false;
        this.messageService.showMessage('info',
          'Informe a quantidade devolvida e a data de devolução do ' + item.equipamento.nome);
        break;
      }
    }

    return retorno;
  }
}
```

Fonte: Autoria própria

Foram codificadas também classes do tipo *component*, que possuem *templates* HTML associados a elas e são capazes de utilizar os *services* para requisições a API e apresentarem os resultados na página.

A Listagem 15 mostra o código HTML para exibição dos empréstimos cadastrados. A tela é dividida em um formulário para filtro e uma tabela para exibição dos empréstimos. No formulário de filtro, foi utilizada a diretiva “formGroup”, caracterizando ele como um formulário reativo e todos os seus *inputs* possuem a propriedade “formControlName” valorizada, para que seja possível obter os conteúdos informados. Dentro do formulário, foram utilizados alguns componentes da biblioteca PrimeNG, como “p-calendar”, “p-autoComplete” e “p-multiSelect”. O primeiro apresenta um *input* no formato de um calendário, o segundo trata-se de um *dropdown* que filtra seu conteúdo

a medida que um texto é inserido no componente e o terceiro também é um *dropdown*, porém com conteúdo já definido e que permite a seleção de mais de um registro. Para apresentação do resultado, foi utilizado o componente "p-table", também da biblioteca PrimeNG, que permite a utilização dos recursos de paginação e *lazy load*.

Listagem 15: HTML para exibição dos empréstimos cadastrados

```
<div class="container">
  <div class="mt-2">
    <div class="p-grid pt-2 pb-2">
      <div class="p-col-12">
        <h3 class="font-weight">Empréstimos</h3>
      </div>
    </div>

    <div class="separador"></div>

    <form [formGroup]="formFiltro" (submit)="findEmprestimos()">
      <div class="p-grid pt-2 pb-2">
        <div class="p-col-4">
          <label for="data">Data</label>
          <p-calendar id="data" name="data" formControlName="data" [locale]="pt"
            [dateFormat]="dateFormat" [inline]="false" [readonlyInput]="false" [showIcon]="true"
            class="ui-fluid">
          </p-calendar>
        </div>

        <div class="p-col-4">
          <label for="usuario">Usuário (Número RA ou E-mail)</label>
          <p-
autoComplete id="usuario" name="usuario" formControlName="usuario" [suggestions]="usuarios"
            (completeMethod)="loadUsuarios($event)" field="nome" [minLength]="3" class="ui-
fluid">

            <ng-template let-usuario pTemplate="item">
              <div class="ui-helper-
clearfix">{{ usuario.nome }} | {{ usuario.email }} | {{ usuario.nrora }}</div>
            </ng-template>
          </p-autoComplete>
        </div>

        <div class="p-col-4">
          <label for="situacao">Situação</label>
          <p-multiSelect id="situacao" formControlName="situacao" defaultLabel="Selecionar..."
            [options]="situacoes" [virtualScroll]="true" [filter]="true" selectedItemsLabel="{0}
itens selecionados"
            class="ui-fluid">
          </p-multiSelect>
        </div>
      </div>

      <div class="pt-2 pb-2">
        <button type="submit" class="btn btn-warning mr-2">Pesquisar</button>
        <button type="button" [routerLink]="['novo']" class="btn btn-dark">Novo</button>
      </div>
    </form>
  </div>

  <div class="mt-3 mb-3">
```

```
    <p-table #tableEmprestimos styleClass="p-2" [value]="emprestimos" [lazy]="true"
      [paginator]="true" [rows]="10" [totalRecords]="totalRecords" (onLazyLoad)="loadLazy($event)
"
      [lazyLoadOnInit]="false" dataKey="idSaida" [responsive]="true">

      <ng-template pTemplate="header">
        <tr>
          <th class="col-expand"></th>
          <th field="idSaida" class="col-codigo">Código</th>
          <th field="data">Data</th>
          <th field="finalidadeSaida">Finalidade</th>
          <th field="usuario">Usuário</th>
          <th field="situacao">Situação</th>
          <th class="text-center col-acoes-emprestimos">Ações</th>
        </tr>
      </ng-template>

      <ng-template pTemplate="body" let-emprestimo let-expanded="expanded">
        <tr>
          <td>
            <a href="" [pRowToggler]="emprestimo" style="color: black;">
              <span [ngClass]="expanded ? 'pi pi-chevron-down' : 'pi pi-chevron-right'"></span>
            </a>
          </td>
          <td>{{ emprestimo.idSaida }}</td>
          <td>{{ emprestimo.data | date:'dd/MM/y'  }}</td>
          <td>{{ emprestimo.finalidadeSaida | finalidadeSaida }}</td>
          <td>{{ emprestimo.usuario.nome }}</td>
          <td>
            <span [ngClass]="getBadgeClass(emprestimo.situacao)">
              {{ emprestimo.situacao | situacaoSaida }}
            </span>
          </td>
          <td class="text-center col-acoes-emprestimos">
            <button pButton (click)="edit(emprestimo.idSaida)" pTooltip="Editar"
              icon="pi pi-pencil" style="width: 32px;">
            </button>

            <button pButton (click)="remove(emprestimo.idSaida)" pTooltip="Excluir" icon="pi pi-
trash"
              class="ui-button-danger ml-1" style="width: 32px;">
            </button>

            <button pButton type="button" *ngIf="emprestimoAprovado(emprestimo.situacao)"
              (click)="encerrarEmprestimo(emprestimo.idSaida)" pTooltip="Encerrar Empréstimo"
              icon="pi pi-thumbs-up" class="ml-1" style="width: 32px;">
            </button>
          </td>
        </tr>
      </ng-template>

      <ng-template pTemplate="rowexpansion" let-emprestimo>
        <tr>
          <td [attr.colspan]="7">
            <tr class="p-grid">
              <th class="p-col-6 text-center">Equipamento</th>
              <th class="p-col-6 text-center">Quantidade</th>
            </tr>
            <tr *ngFor="let item of emprestimo.itens" class="p-grid">
              <td class="p-col-6 text-center">{{ item.equipamento.nome }}</td>
              <td class="p-col-6 text-center">{{ item.quantidade }}</td>
```

```
            </tr>
          </td>
        </tr>
      </ng-template>

      <ng-template pTemplate="emptymessage">
        <tr>
          <td [attr.colspan]="7">Nenhum empréstimo para ser exibido.</td>
        </tr>
      </ng-template>
    </p-table>
  </div>
</div>

<p-confirmDialog></p-confirmDialog>
```

Fonte: Autoria própria

Para carregar os dados no HTML anterior, foi implementado o código da Listagem 16. Nesse código são definidas algumas variáveis que permitem a interação com os elementos visuais da tela. O *decorator* @ViewChild faz referência a tabela de visualização, enquanto "formFiltro" referencia o formulário que filtra os resultados. Para construção desse formulário, foi necessário injetar um objeto do tipo FormBuilder, que pelo seu método "group()" retorna uma instância de FormGroup. Para aplicar um filtro de fato, é realizada uma busca no banco de dados, isto é, uma requisição HTTP para a API, e essa lógica foi encapsulada na classe "SaidaService", pelo método "findEmprestimosByFiltros()". Essa pesquisa depende de um objeto do tipo "EmprestimoFiltro", que é obtido a partir da função "getRawValue()" executada sobre o formulário de filtro.

Listagem 16: Classe que carrega os empréstimos e interage com o HTML

```
/* imports omitidos */

@Component({
  selector: 'app-emprestimos-lista',
  templateUrl: './emprestimos-lista.component.html'
})
export class EmprestimosListaComponent extends CalendarBase implements OnInit {

  @ViewChild('tableEmprestimos', {static: false}) table: Table;
  formFiltro: FormGroup;
  saidaFiltro: EmprestimoFiltro = new EmprestimoFiltro();
  emprestimos: Saida[] = [];
  usuarios: Usuario[] = [];
  situacoes: SelectItem[] = [];
  totalRecords = 0;
  private currentPage = 0;
  private maxRecords = 10;

  constructor(private formBuilder: FormBuilder,
              private saidaService: SaidaService,
              private router: Router,
              private usuarioService: UsuarioService,
              private confirmationService: ConfirmationService,
              private messageService: MyMessageService) {
```

```
    super();
  }

  ngOnInit(): void {
    this.formFiltro = this.formBuilder.group({
      data: [],
      usuario: [],
      situacao: []
    });
    this.carregarSituacoes();
  }

  private carregarSituacoes() {
    const situacoesTemp = [];

    situacoesTemp.push(new SituacaoSaidaDescricao(SituacaoSaida.pendente, 'Pendente'));
    situacoesTemp.push(new SituacaoSaidaDescricao(SituacaoSaida.aprovada, 'Aprovado'));
    situacoesTemp.push(new SituacaoSaidaDescricao(SituacaoSaida.encerrada, 'Encerrado'));
    situacoesTemp.push(new SituacaoSaidaDescricao(SituacaoSaida.reprovada, 'Reprovado'));

    situacoesTemp.map(sit => this.situacoes.push({label: sit.descricao, value: sit.situacao}));
  }

  private findEmprestimosByFiltros() {
    this.saidaService.findEmprestimosByFiltros(this.saidaFiltro).subscribe(
      (resp) => {
        this.totalRecords = resp.totalElements;
        this.emprestimos = resp.content;
      }
    );
  }

  findEmprestimos() {
    this.saidaFiltro = this.formFiltro.getRawValue() as EmprestimoFiltro;
    this.saidaFiltro.page = this.currentPage;
    this.saidaFiltro.size = this.maxRecords;
    this.findEmprestimosByFiltros();
  }

  loadLazy(event: LazyLoadEvent) {
    this.currentPage = event.first / event.rows;
    this.maxRecords = event.rows;
    this.saidaFiltro.page = this.currentPage;
    this.saidaFiltro.size = this.maxRecords;
    setTimeout(() => this.findEmprestimosByFiltros(), 200);
  }

  loadUsuarios(event) {
    this.usuarioService.findByNroraOrEmail(event.query).subscribe(
      (resp) => this.usuarios = resp
    );
  }

  edit(idEmprestimo: number) {
    this.router.navigate(['/emprestimos', idEmprestimo]);
  }

  remove(idEmprestimo: number) {
    this.confirmationService.confirm({
      header: 'Deseja realmente excluir o registro?',
      message: 'Essa ação não poderá ser desfeita.',
```

```
      acceptLabel: 'Excluir',
      rejectLabel: 'Cancelar',
      accept: () => {
        this.saidaService.delete(idEmprestimo).subscribe(
          () => {
            this.table.reset();
            this.messageService.showMessage('success', 'Empréstimo removido com sucesso!');
          }
        );
      }
    });
  }

  getBadgeClass(situacao: SituacaoSaida): string {
    return getBadgeClass(situacao);
  }

  encerrarEmprestimo(idEmprestimo: number) {
    this.confirmationService.confirm({
      header: 'Empréstimos DAELE',
      message: 'Deseja realmente encerrar este empréstimo?',
      acceptLabel: 'Encerrar',
      rejectLabel: 'Cancelar',
      accept: () => {
        this.saidaService.updateSituacao(idEmprestimo, SituacaoSaida.encerrada).subscribe(
          () => {
            this.table.reset();
            this.messageService.showMessage('success', 'Empréstimo atualizado com sucesso!');
          }
        );
      }
    });
  }

  emprestimoAprovado(situacao: SituacaoSaida): boolean {
    return situacao === SituacaoSaida.aprovada;
  }
}
```

Fonte: Autoria própria

4 CONCLUSÃO

O objetivo deste trabalho foi o desenvolvimento de um sistema para o ambiente *web*, que permitisse o controle mais seguro e eficaz dos empréstimos dos equipamentos do DAELE, tendo em vista que a rotina atual para esse tipo de processo contém riscos no gerenciamento do departamento. Para atingir esse objetivo, foram estruturados dois projetos distintos, sendo que um deles tem a meta de fornecer uma interface simples e amigável para utilização, e o outro disponibiliza um servidor apto a atender requisições para persistência de dados, consultas e regras de negócio em geral.

Uma série de tecnologias foram utilizadas para que fosse possível atender aos requisitos propostos, destacando principalmente os *frameworks* Spring e Angular. A escolha dessas tecnologias se mostrou acertada, uma vez que ambos possuem uma documentação de fácil acesso e entendimento, além de serem bastante produtivos. O Spring, por exemplo, contém recursos que otimizam o tempo de desenvolvimento na construção de APIs, já que com poucos passos é possível disponibilizar *endpoints* cujas entradas de dados são facilmente convertidas em classes, permitindo suas manipulações. A injeção de dependências também é outra característica que destaca o *framework*.

Em relação ao Angular, os principais benefícios visualizados foram em relação a grande quantidade de funcionalidades que ele disponibiliza. Primeiramente, a existência da ferramenta Angular CLI facilitou a criação do projeto *front-end*, pois necessita de poucos comandos via terminal para seu funcionamento. Além da criação do projeto, o Angular CLI permite também a criação de classes específicas no projeto, tais como componentes, módulos e serviços. A fácil configuração para atender diferentes cenários também é um ponto que motiva a sua utilização, como a criação de interceptadores de requisições HTTP, validações customizadas de campos de formulários, entre outras.

Por fim, o desenvolvimento do projeto ocorreu conforme o planejado e possibilitou um ganho considerável de conhecimento, devido à necessidade de constantes consultas às documentações das tecnologias utilizadas, bem como implementações de tecnologias até então desconhecidas pelo autor. Como trabalho futuro, espera-se aprimorar as regras de negócio da aplicação, tornando o projeto ainda mais seguro e eficaz, além de aperfeiçoar as interfaces.

5 REFERÊNCIAS

ANGULAR. **Angular**. Disponível em <https://angular.io/guide/architecture>. Acesso em: 13 fev 2020.

CALADO, Ricardo. **Um overview sobre o framework Angular.** Disponível em <https://blog.geekhunter.com.br/um-overview-sobre-o-framework-angular/>. Acesso em: 13 fev. 2020.

CAMARGO, Danieli A. de Oliveira. ESTUDO DAS TÉCNICAS DE DESENVOLVIMENTO WEB E VALIDAÇÃO DESTE ESTUDO COM UM PORTAL PARA RECICLADORES E PRODUTORES DE RECICLADOS. **Anuário da Produção de Iniciação Científica Discente.** Vol. XII, Nº 14, Ano 2009.

PIMENTA, Alexandre Manuel Santareno; QUARESMA, Rui Filipe Cerqueira. A SEGURANCA DOS SISTEMAS DE INFORMAÇÃO E O COMPORTAMENTO DOS USUÁRIOS. **JISTEM J. Inf.Syst. Technol. Manag.** São Paulo, v. 13, n. 3, p. 533-552, Dec. 2016.

POSTGRESQL. **PostgreSQL**. Disponível em <https://www.postgresql.org/about/>. Acesso em: 13 fev. 2020.

PRIMEFACES. **PrimeNG.** Disponível em <https://www.primefaces.org/primeng/#/>. Acesso em: 13 fev. 2020.

RODRIGUES, Luis António da Silva. **Arquiteturas dos sistemas de informação**. Lisboa: FCA Editora de Informática. Disponível em <https://recipp.ipp.pt/bitstream/10400.22/4930/1/Disserta%c3%a7%c3%a3o.PDF>. Acesso em: 13 fev. 2020.

SOUZA, Alberto. **Spring MVC: domine o principal framework web Java**. Editora Casa do Código, 2017.

SPRING. **Spring.** Disponível em <https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html#overview>. Acesso em 13 fev. 2020.

ZANETI JUNIOR, Luiz Antonio. **Sistemas de informação baseados na tecnologia web: um estudo sobre seu desenvolvimento**, 2003. Dissertação (Mestrado em Administração) - Faculdade de Economia, Administração e Contabilidade, Universidade de São Paulo, São Paulo, 2003.