



SIG2E - Sistema Informático para Gestão de Espaços e Equipamentos

Projeto P20

Relatório Final da Unidade Curricular de Projeto

Licenciatura em Engenharia de Informática, Redes e Telecomunicações
Departamento de Eng. Eletrónica e Telecomunicações e de Computadores
Instituto Superior de Engenharia de Lisboa

Alunos:

André Santos, n.º 48705

João Teixeira, n.º 48710

Orientadores:

Prof. Tiago Miguel Braga da Silva Dias

Prof. Diogo Cardoso

Julho de 2023

Resumo

Este relatório visa demonstrar uma possível implementação para a reserva de equipamentos e laboratórios no DEETC do ISEL através da criação de um sistema de gestão. No relatório são levantados os problemas que ocorrem atualmente e são propostas soluções fundamentadas para a sua solução.

Índice

1. Introdução.....	1
2. Descrição do Projeto	2
3. Tarefas a Realizar	3
4. Seleção das Tecnologias	7
4.1 Base de dados.....	7
4.2 Servidor	8
4.3 API.....	10
4.4 Aplicação Web.....	10
5. Implementação do Projeto	12
5.1 <i>Backend</i>	12
5.1.1 Base de Dados	12
5.1.2 Servidor	13
5.1.3 API	14
5.2 <i>Frontend</i>	16
5.2.1 Componentes	19
6. Conclusão.....	21
6.1 Reflexão crítica	21
6.2 Trabalho Futuro	23
7. Referências	24
Apêndice I – Manual de Documentação da API	28
Equipamentos:	28
Bancadas:	31
Laboratórios:	33
Avarias:	36
Reservas:	42
Espaços:	46
Armazéns:	47
Utilizadores:	48
Apêndice II – Manual de Descrição Da UI	51
1. Navegação.....	51
2. Autenticação	52

3. Reservas.....	52
4. Equipamentos.....	53
5. Espaços	55
6. Avarias.....	59

Índice de Figuras

Figura 1- Diagrama da arquitetura do SIG2E	2
Figura 2 - Diagrama de caso de uso de autenticação e laboratórios	4
Figura 3 - Diagrama de caso de uso de equipamentos	5
Figura 4 - Diagrama de caso de uso da gestão dos utilizadores	6
Figura 5 - Diagrama do modelo ER SIG2E	13
Figura 6 - Organização dos ficheiros do Backend	14
Figura 7 - Ficheiros da diretoria routes	15
Figura 8 - Fluxo de Utilização da Aplicação Web	17
Figura 9 - Diagrama Funcionalidades Autenticadas	18
Figura 10 - Estruturação Base de Código Frontend	19
Figura 11 - Barra de Navegação Lateral Expandida.....	51
Figura 12 - Barra de Navegação Lateral Escondida	51
Figura 13 - Barra de Navegação de Topo.....	51
Figura 14 - Menu Utilizador não Autenticado	52
Figura 15 - Menu Utilizador Autenticado	52
Figura 16 - Formulário Autenticação	52
Figura 17 - Tabela das Reservas	53
Figura 18 - Criar Reserva.....	53
Figura 19 - Tabela dos Equipamentos	54
Figura 20 - Detalhes de um Equipamento	54
Figura 21 - Adicionar Equipamento Móvel	55
Figura 22 - Adicionar Equipamento de Bancada	55
Figura 23 - Visualização Laboratório e Sala Armazenamento.....	56
Figura 24 - Visualização de Horários	56
Figura 25 - Visualização de Bancadas.....	57
Figura 26- Adicionar Laboratório.....	58
Figura 27 - Adicionar Sala Armazenamento	58
Figura 28 - Campo de Horário.....	58
Figura 29 - Visualização de Avarias.....	59

Figura 30 - Formulário de Notificação de uma Avaria.....	60
--	----

Índice de Tabelas

Tabela 1 - Características das bases de dados SQL e NoSQL.....	8
--	---

Lista de acrónimos

ACID - Atomicidade, Consistência, Isolamento e Durabilidade (do inglês, Atomicity, Consistency, Isolation, Durability)

API - Interface de Programação de Aplicação (do inglês, Application Programming Interface)

BASE - Basicamente Acessível, Estado Suave e Eventualmente Consistente (do inglês, Basically Available, Soft State and Eventually Consistent)

BD - Base de dados

CSS - Cascading Style Sheet

DEETC - Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

ER - Entidade-Relação

HTML - Linguagem de Marcação de Hipertexto (do inglês, HyperText Markup Language)

HTTP - Protocolo de Transferência de Hipertexto (do inglês, Hypertext Transfer Protocol)

HTTPS - Protocolo de Transferência de Hipertexto Seguro (do inglês, Hypertext Transfer Protocol Secure)

ISEL - Instituto Superior de Engenharia de Lisboa

IPL - Instituto Politécnico de Lisboa

JMV - Máquina Virtual Java (do inglês, Java Virtual Machine)

JSON - Notação de Objeto JavaScript (do inglês, JavaScript Object Notation)

LEIRT - Licenciatura em Engenharia Informática, Redes e Telecomunicações

NoSQL - Não Somente SQL (do inglês, Not Only SQL)

RBAC - Controlo de Acesso Baseado em Funções (do inglês, Role-Based Access Control)

REST - Transferência de Estado Representacional (do inglês, Representational State Transfer)

SIG2E - Sistema Informático para Gestão de Espaços e Equipamentos

SGBD - Sistema de Gestão de Base de Dados

SOAP - Protocolo Simples de Acesso a Objetos (do inglês, Simple Object Access Protocol)

SQL - Linguagem de Consulta Estruturada (do inglês, Structured Query Language)

UI - Interface do Utilizador (do inglês, User Interface)

URI - Identificador Único de Recurso (do inglês, Uniform Resource Identifier)

UML - Linguagem de Modelagem Unificada (do inglês, Unified Modeling Language)

UX - Experiência do Utilizador (do inglês, User Experience)

XML - Linguagem Extendida de Marcação (do inglês, Extensible Markup Language)

Instituto Superior de Engenharia de Lisboa

1. Introdução

As atividades de natureza laboratorial e/ou experimental implementam estratégias de ensino que visam facilitar a compreensão e a consolidação de determinados conteúdos programáticos, promovendo uma melhor visão e profundidade do conteúdo lecionado através da interligação dos conteúdos técnicos aos teóricos e motivando a autonomia dos/as alunos/as para o seu estudo. Acresce ainda que estas atividades também potenciam nos/as estudantes o desenvolvimento de competências a nível da comunicação, da argumentação e do diagnóstico de problemas. O trabalho experimental em laboratório é, portanto, um fator crucial para a formação de novos/as engenheiros/as [1].

Infelizmente, as medidas implementadas nos últimos anos para combater a pandemia por COVID-19 afastaram os/as alunos/as dos espaços físicos de ensino, com significativo prejuízo para a sua formação, não só ao nível da sua componente prática e experimental, como também nos seus hábitos de trabalho [2]. Assim, no atual momento, é muito importante desenvolver estratégias e implementar metodologias que incentivem os/as alunos/as a voltar a usar os laboratórios e os seus equipamentos, não só durante as aulas programadas, mas também no seu tempo livre.

Atualmente, existem diversos fatores que condicionam e desmotivam a utilização em regime livre dos laboratórios do DEETC, desde logo os procedimentos não automatizados e de carácter presencial para a reserva dos espaços e equipamentos, a dificuldade em aceder à informação sobre os horários disponíveis ou o desconhecimento da disponibilidade dos equipamentos. Este projeto visa combater estas dificuldades, enquadrando-se numa estratégia para promoção do trabalho em laboratório junto dos/as alunos/as dos cursos de engenharia informática do Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores (DEETC) do ISEL, em que se inclui o curso de Licenciatura em Engenharia Informática, Redes e Telecomunicações (LEIRT).

2. Descrição do Projeto

Este projeto tem como objetivo o desenvolvimento de um sistema informático para gestão e reserva dos espaços e equipamentos de laboratório afetos ao DEETC do ISEL, designado por Sistema Informático para Gestão de Espaços e Equipamentos (SIG2E).

O sistema a desenvolver será composto por dois elementos: uma aplicação de gestão, designada por *Backend*, e uma aplicação web, designada por *Website*. A aplicação de gestão será a componente que irá fornecer lógica ao sistema, sendo responsável pelo armazenamento e processamento dos dados sobre os espaços e os equipamentos geridos no sistema e os seus utilizadores. A aplicação *web* será a interface de utilização que permitirá aos utilizadores interagirem com o sistema, através de um explorador de *Internet*, possibilitando assim a sua gestão e utilização.

A Figura 1 ilustra a ligação entre os dois elementos do SIG2E, onde o *Frontend* é implementado pela aplicação *web* e o *Backend* implementa a aplicação de gestão, composta por uma base de dados (BD), uma Interface de Programação de Aplicação (em inglês, *Application Programming Interface* - API) e um servidor. A BD é o componente que irá organizar e armazenar a informação de modo a esta ser consistente e de acesso rápido e eficiente, permitindo ainda através de *queries* um processamento personalizado. Através da API será possível estabelecer a comunicação entre o *Frontend* e o *Backend* usando rotas no servidor e recorrendo ao Protocolo de Transferência de Hipertexto (em inglês, *Hypertext Transfer Protocol* - HTTP) [3], um dos protocolos mais utilizados devido à sua fiabilidade, simplicidade e suporte. O servidor executará todas as funcionalidades do sistema, incluindo o cumprimento das regras lógicas do algoritmo, o acesso à informação da BD e o processamento dos pedidos recebidos pela API.

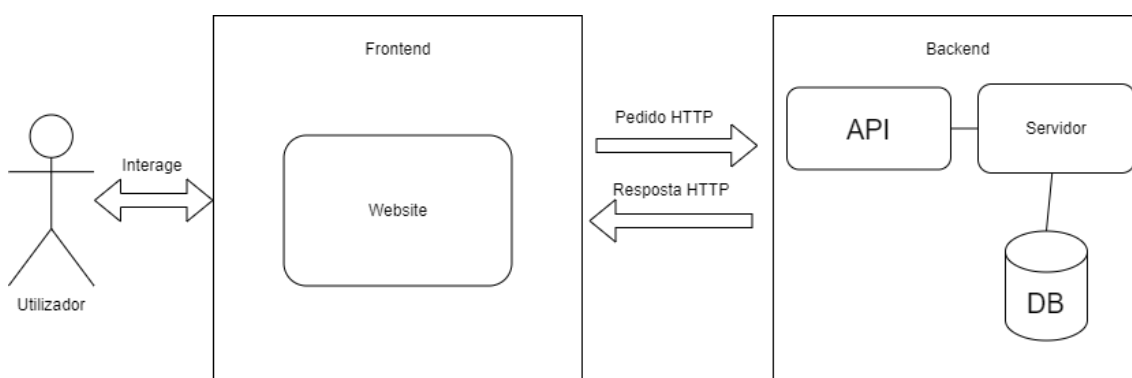


Figura 1- Diagrama da arquitetura do SIG2E

3. Tarefas a Realizar

A Figura 2, Figura 3 e Figura 4 apresentam, usando diagramas UML do tipo caso de uso [4], o contexto e os requisitos funcionais básicos para o desenvolvimento do SIG2E, bem como o seu modelo de fluxo de eventos.

Em cada diagrama estão representadas as funcionalidades que o sistema deve implementar, bem como as suas relações. À esquerda estão representados os tipos de utilizadores possíveis, i.e., Aluno, Professor, Responsável de Laboratório ou Administrador, em que cada um possui um conjunto de permissões que são herdadas hierarquicamente, seguindo um sistema de Controlo de Acesso Baseado em Funções (em inglês, *Role-Based Access Control* - RBAC). Por exemplo, um Responsável de Laboratório terá todas as permissões de um Professor, que por sua vez terá também todas as permissões de um Aluno. No entanto, um Responsável de Laboratório não conseguirá exercer as funções de um Administrador. Finalmente, os atores à direita do bloco SIG2E representam as entidades responsáveis por executar as funcionalidades.

Na Figura 2 representam-se as funcionalidades suportadas pelo sistema relacionadas com a autenticação e a gestão dos espaços, doravante designados por laboratórios. No que diz respeito à autenticação, todos os utilizadores podem fazer o Log In, que se pretende ser tratado pela entidade dos Serviços de Autenticação do ISEL, existindo ainda um Log In administrativo para Administradores. Esta autenticação deverá ainda ser segura pelo qual deverá ser utilizado HTTPS na comunicação de modo a encriptar os dados ponto a ponto. Quanto aos laboratórios, todos os utilizadores vão poder dispor das funcionalidades de consultar e requisitar laboratórios, prolongar ou cancelar a requisição e de notificar falhas. As funcionalidades de adicionar, alterar e remover laboratórios, assim como a verificação de falhas, alteração da condição dos laboratórios (caso existam avarias que impossibilitem o seu uso) ou a atualização da pontuação dos utilizadores (se foi feita uma comunicação correta das avarias ou se o equipamento foi retornado em bom estado de funcionamento) estão restritas ao responsável de laboratório. Todas as funcionalidades relacionadas com o laboratório são executadas pela entidade *Backend*.

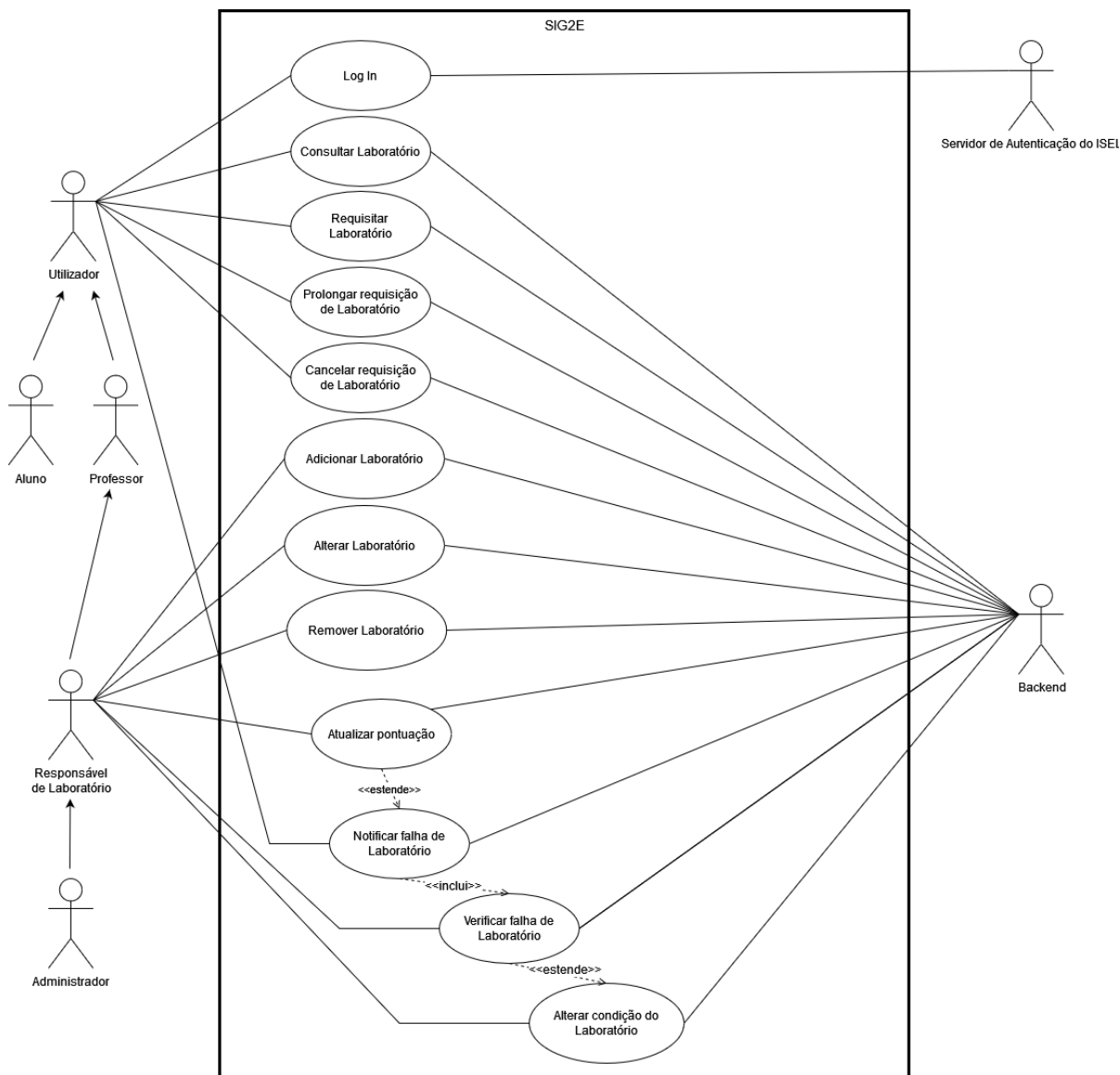


Figura 2 - Diagrama de caso de uso de autenticação e laboratórios

Na Figura 3 estão representadas as funcionalidades do sistema relacionadas com a gestão dos equipamentos. Das funcionalidades apresentadas é possível verificar que a consulta de equipamentos, a requisição e o prolongamento da requisição, o seu cancelamento e a notificação de falhas estão disponíveis para todos os utilizadores do sistema. As restantes funcionalidades, i.e., adição, remoção ou alteração de características dos equipamentos, assim como a verificação de falhas e a alteração da condição dos equipamentos (se apresenta avarias) ou a atualização das pontuações dos utilizadores apenas podem ser realizadas pelo Responsável de Laboratório ou pelo Administrador. Tal como as sucede com a gestão dos laboratórios, todas as funcionalidades relacionadas com a gestão de equipamentos são executadas pela entidade de *Backend*.

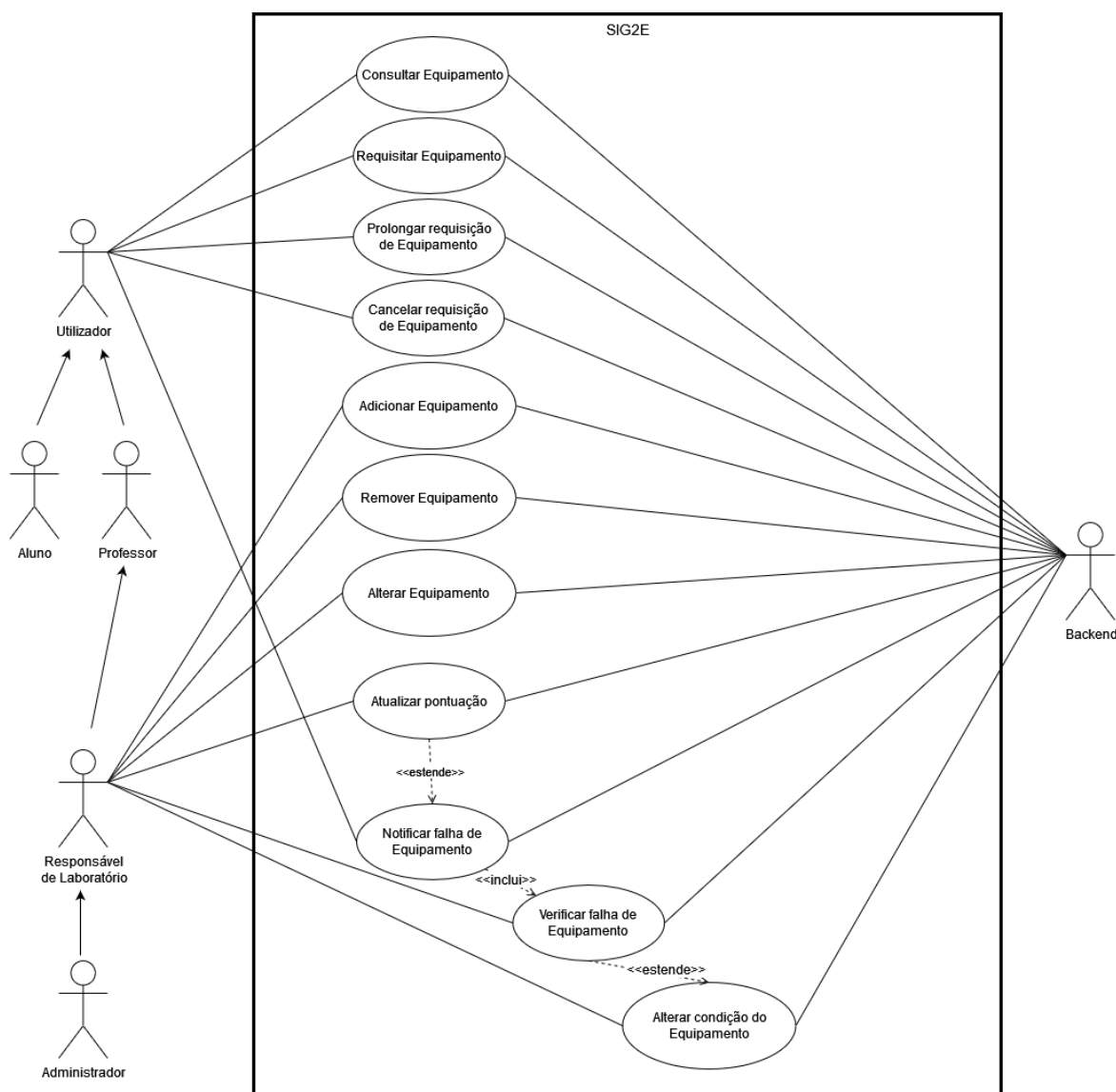


Figura 3 - Diagrama de caso de uso de equipamentos

Na Figura 4 estão representadas as funcionalidades do sistema relacionadas com a gestão dos utilizadores. Das funcionalidades apresentadas, o Responsável de Laboratório consegue apenas autorizar e suspender utilizadores, estando as funcionalidades de adição e remoção de utilizadores e de gestão das suas permissões disponíveis apenas para os administrados do sistema. À semelhança das outras funcionalidades do sistema, todas as operações relacionadas com a gestão dos utilizadores são executadas pelo *Backend*.

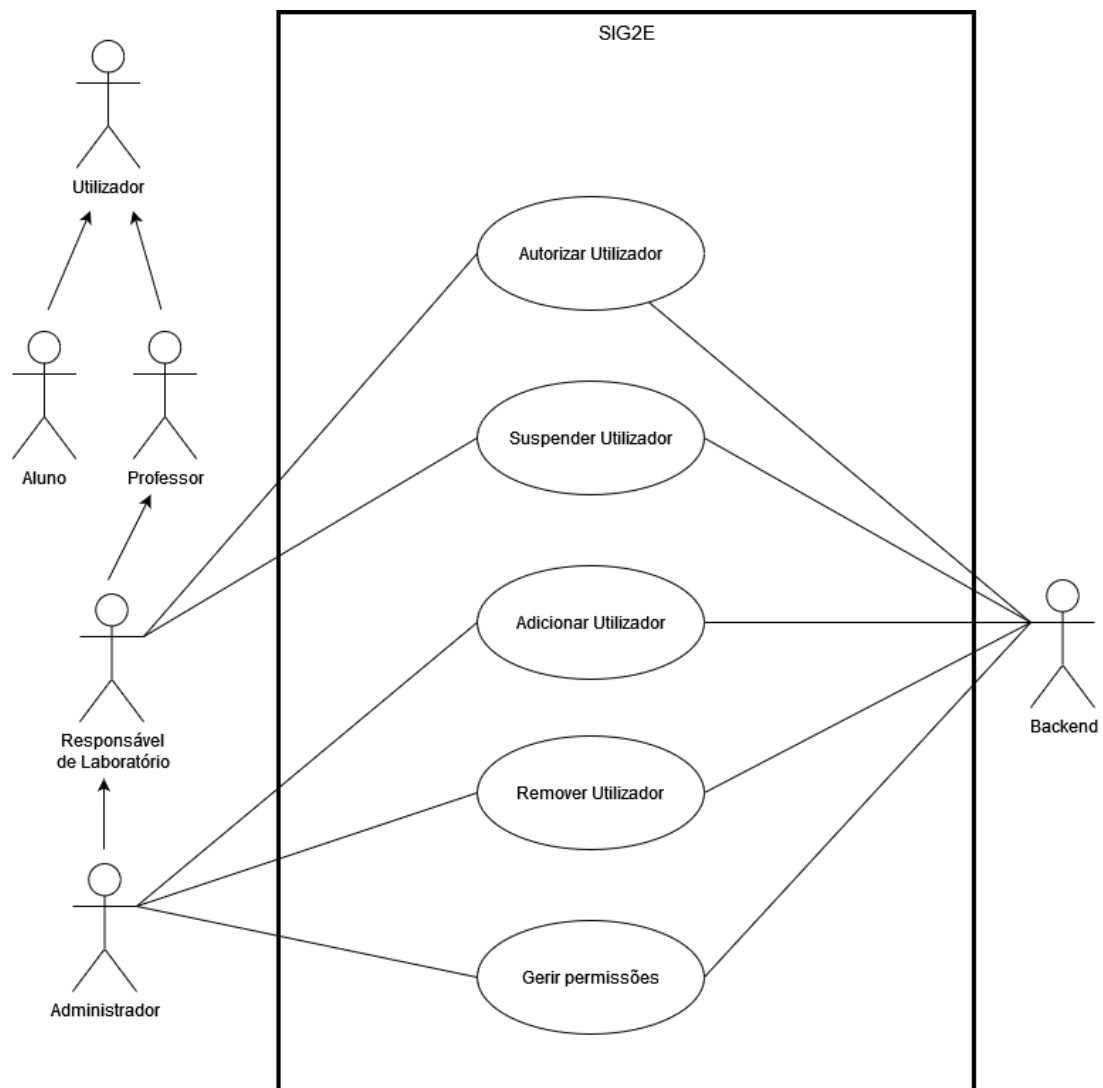


Figura 4 - Diagrama de caso de uso da gestão dos utilizadores

4. Seleção das Tecnologias

Neste capítulo, analisam-se os requisitos de cada um dos componentes que compõem a arquitetura do SIG2E, i.e. a base de dados, o servidor, a API e a aplicação web, com o objetivo de selecionar as tecnologias mais adequadas para sua implementação

4.1 Base de dados

Um sistema de gestão de base de dados (SGBD) é responsável pela gestão de uma ou mais bases de dados, sendo o seu principal objetivo gerir o acesso e processamento dos dados. Estas BD podem ser do tipo relacional, como não relacional.

As bases de dados relacionais, ou mais frequentemente bases de dados de consulta estruturada (em inglês, *Structured Query language*, SQL), são sistemas de armazenamento de informação estruturada baseadas no modelo relacional [5], sendo este tipo de sistemas constituídos por entidades e relações. Uma entidade (tabela) é um elemento caracterizado por atributos (colunas) e que apresenta os seus valores em énplo (linhas, ou ainda, em inglês, tuple). A relação é o modo como as entidades (tabelas) se associam. As principais vantagens deste tipo de sistema são a sua ferramenta de *queries* SQL para consulta e processamento de dados e as suas características de atomicidade, consistência, isolamento e durabilidade (ACID), garantindo assim uma maior fiabilidade e integridade da informação [6]. O princípio de atomicidade garante que uma transação (uma ou várias mudanças na BD) é tratada por inteira, sendo esta apenas bem-sucedida ou malsucedida. A consistência assegura a integridade dos dados, ou seja, todos os dados de uma transação devem seguir as regras de integridade da tabela. O isolamento certifica que diferentes transações a correr em modo paralelo não têm efeito umas sobre as outras. Por fim, a durabilidade garante que a informação deve persistir permanentemente, mesmo em casos de erros ou quebras de energia.

No entanto, estes sistemas relacionais apresentam dificuldades na escalabilidade pois possuem um modelo de dados restrito e apresentam complicações na distribuição do processamento por diferentes máquinas (também conhecido como escalonamento horizontal, em inglês, *Horizontal Scaling*), possuindo assim tipicamente um escalonamento vertical (em inglês, *Vertical Scaling*) [7] [8]. Devido a esta dificuldade foram criadas bases de dados não relacionais, também conhecidas por NoSQL [9].

As bases de dados NoSQL são sistemas de armazenamento de informação que utilizam diferentes tipos de documentos, sendo estes geralmente em formato JSON [10] (do inglês, *JavaScript Object Notation*) e em relações de chave-valor. Ao contrário dos modelos relacionais, geralmente, as bases de dados NoSQL seguem os princípios BASE (basicamente acessível, estado suave e eventualmente consistente) que não garantem a consistência nem fiabilidade dos dados. O princípio de basicamente acessível diz que a informação está sempre disponível ao espalhá-la pelos diversos nós de um *cluster* [5]. Estado suave significa que o modelo de dados não garante consistência da informação e que esta pode ser alterada ao longo do tempo. Por

fim, a base de dados é eventualmente consistente, ou seja, é possível realizar operações antes da informação se tornar consistente. Devido a este tipo de sistemas não utilizarem um modelo de dados restrito possuem uma escalabilidade horizontal superior aos dos sistemas relacionais [11].

A Tabela 1 apresenta, de um modo sucinto, as principais diferenças entre bases de dados relacionais e não relacionais.

Base de dados SQL	Base de dados NoSQL
Sistema de gestão de base de dados relacional	Sistema de gestão de base de dados não relacional
Escalabilidade Vertical	Escalabilidade Horizontal
Modelo de dados Restrito	Modelo de dados Flexível
Transações ACID	Transações BASE

Tabela 1 - Características das bases de dados SQL e NoSQL

No sistema SIG2E os atributos e as entidades estarão intrinsecamente relacionados, por exemplo, um equipamento (entidade) tem sempre de ter um identificador único e um estado de utilização (atributos) e pode estar a ser ocupado por um utilizador (entidade) que também terá atributos como identificador único e nome. Deste modo, devido à natureza e baixo volume de dados que se espera armazenar, a opção mais indicada são bases de dados relacionais, não sendo assim necessário considerar soluções de escalabilidade horizontal.

Das diferentes opções existentes no mercado de sistemas de gestão de base de dados relacionais, neste trabalho foi utilizado o PostgreSQL, devido à sua natureza gratuita e de código aberto e à sua grande comunidade ativa, possuindo ainda funcionalidades bastante competitivas com os restantes tipos de base de dados disponíveis.

4.2 Servidor

O servidor é a componente do sistema que implementa a aplicação responsável pelo processamento dos pedidos da aplicação web, implementando a lógica da aplicação, interagindo com a base de dados e disponibilizando as rotas para a API.

Para a criação do servidor web foram analisadas diversas linguagens de programação e tecnologias que atendessem às necessidades do projeto, deste modo descartando o uso de linguagens com *frameworks* que não disponibilizam nativamente ou via bibliotecas os recursos necessários para a realização do projeto. Além disso, embora Java e PHP ainda sejam extensamente utilizados, existem linguagens mais modernas e dinâmicas que oferecem uma gama mais ampla de recursos e ferramentas de desenvolvimento. Outro critério utilizado na seleção das linguagens foi a curva de aprendizagem, tendo sido descartadas linguagens que apresentam um alto nível de dificuldade na sua aprendizagem, como Ruby. De entre as restantes linguagens analisadas, JavaScript e Python evidenciaram-se como as mais relevantes por possuírem *frameworks* que agilizam o processo de implementação e serem mais compatíveis com as necessidades do projeto. Após considerar as *frameworks* Django [12] e Flask [13] para Python, e o Express [14] e o Deno [15] para JavaScript, descartou-se o Python devido a experiência prévia com a tecnologia JavaScript e por esta ser uma linguagem que pode ser utilizada tanto no desenvolvimento do servidor como da aplicação web. No entanto, devido à sua natureza dinâmica, o JavaScript é mais propício a problemas no desenvolvimento do código, uma vez que não tem um analisador estático [16]. Por esta razão, optou-se por utilizar TypeScript [17], uma linguagem que estende JavaScript adicionando tipificação.

TypeScript é uma linguagem de *scripting*, ou seja, os programas são interpretados em tempo de execução, nativamente pelo navegador. Consequentemente, para a sua execução no servidor surge a necessidade de dispor de um ambiente de execução, que é um sistema que permite a execução de código fora de um navegador web e é responsável pela gestão dos recursos do sistema, como o acesso à memória e rede. No caso de JavaScript existem três principais ambientes de execução: NodeJS [18], Deno [19] e Rhino [20]. Entre estas opções o NodeJS destaca-se pela sua ampla adoção e suporte ativo. O Deno, por sua vez, é conhecido pela sua segurança e suporte nativo ao TypeScript, embora seja uma ferramenta mais recente e possua uma comunidade menos ativa que o NodeJS. Por fim, o Rhino oferece integração com Java, sendo este executado na máquina virtual Java (em inglês, Java Virtual Machine, JVM). Uma vez que existem mais bibliotecas e *frameworks* e também devido a experiência prévia com a tecnologia, o ambiente de execução adotado para o servidor é NodeJS [21].

Para disponibilizar os recursos do servidor via uma API existem diferentes *frameworks* que podem ser utilizadas, tal como Express [14], Hapi [22] e Koa [23]. Express foi desenvolvido para auxiliar na criação de aplicações web, fornecendo um conjunto de recursos que agilizam o processo de desenvolvimento, desde declaração de rotas a manipulação de pedidos e respostas [24]. O Hapi foi criado para resolver problemas de performance do Express, armazenando em cache os pedidos para respostas mais rápidas [25]. O Koa é uma opção mais leve computacionalmente pois não inclui componentes não estritamente necessários [26]. Neste trabalho optou-se por usar Express dado que as outras duas *frameworks* são suas derivadas e porque é a que tem mais suporte e documentação disponível.

4.3 API

Conforme apresentado no Capítulo 2, o modo de comunicação entre o *Frontend* e o *Backend* do SIG2E é definido por uma API, que estabelece o formato de dados da comunicação e a forma de acesso aos seus métodos.

Alguns dos tipos de API mais conhecidos são baseados nos protocolos SOAP [27] (do inglês, *Simple Object Access Protocol*) ou REST [27] (em inglês, *Representational State Transfer*).

As API baseadas no protocolo SOAP permitem a exposição de recursos do servidor que são invocados pelo cliente por meio de mensagens no formato XML (do inglês, *Extensible Markup Language*). Essas mensagens são compostas por um envelope que encapsula os detalhes da mensagem, incluindo o corpo e o cabeçalho, este último sendo opcional. O corpo da mensagem contém a informação a ser transmitida para o destinatário, que pode ser uma invocação de recursos por parte do cliente ou uma resposta por parte do servidor. O cabeçalho contém informações adicionais relacionadas à mensagem e ao processamento da mesma, como meta dados, informações de autenticação, configurações de segurança, entre outros [28].

As API baseadas no protocolo REST pressupõem que as comunicações são sem estado, o que significa que o servidor não armazena informações sobre os clientes. Em vez disso, são os próprios clientes que, para cada pedido, enviam as informações necessárias para que o servidor possa interpretar o pedido adequadamente. A forma como o cliente invoca os pedidos ao servidor é por meio de métodos HTTP, como GET, PUT, POST e DELETE, utilizando um indicador de recurso único (em inglês, *Uniform Resource Identifier - URI*) que identifica o recurso solicitado. Ao contrário do protocolo SOAP, os dados trocados nas mensagens de uma API REST podem assumir vários formatos, como XML, JSON, texto em claro, entre outros.

No desenvolvimento do SIG2E optou-se por implementar uma API que implementa o protocolo REST, pois estas requerem menos processamento do que as baseadas no protocolo SOAP, consumindo uma menor largura de banda e sendo mais simples de desenvolver [29].

4.4 Aplicação Web

Uma aplicação web é um programa informático disponibilizado aos utilizadores através de um navegador web (em inglês, *browser*) e que é entregue através da rede por um servidor web. Nesta transação, que geralmente utiliza o protocolo HTTP, o navegador envia pedidos a um servidor web, que os processa e envia as correspondentes respostas de volta ao cliente para serem apresentadas no navegador. Estas aplicações consistem em páginas web e que são comumente construídas através das linguagens HTML, CSS (do inglês, *Cascading Style Sheets*) e Javascript. A linguagem HTML é utilizada para definir a estrutura e o conteúdo estático das páginas, sendo este o elemento base da aplicação web. No entanto, para melhorar a aparência e apresentação visual das páginas, é utilizada a linguagem CSS, que é responsável pela definição de estilo e disposição dos elementos. Assim, HTML e CSS trabalham em conjunto para

tornar a aplicação web mais atrativa e fácil de usar. Por fim, de modo a criar páginas dinâmicas é possível executar código Javascript para atualizar o conteúdo das páginas HTML em tempo real sem ter de atualizar toda a página, sendo assim aumentada a interatividade.

Devido ao processo de criação de páginas web ser algo moroso e suscetível a erros, ao longo dos anos foram surgindo *frameworks* que auxiliam e agilizam o processo de construção de aplicações web através da utilização de recursos pré-construídos e testados, como componentes expansíveis para o domínio da aplicação (por exemplo, existir um botão genérico que pode ser utilizado para construir botões específicos à aplicação), oferecendo assim abstração. Por razões semelhantes, também existem *frameworks* de CSS para customizar a apresentação dos documentos que foram gerados pela *framework* de construção de elementos da página.

Relativamente à geração dos elementos que compõem as páginas web, as *frameworks* mais relevantes atualmente são Angular [30], React [31] e Vue [32]. Neste projeto optou-se por utilizar React devido à sua atualidade, características de reutilização e ainda possuir uma comunidade bastante ativa, sendo esta a *framework* de *Frontend* mais utilizada atualmente [33].

Desta decisão resulta que as opções mais viáveis para customizar a apresentação dos documentos gerados com a *framework* React, i.e., o estilo da página, são aquelas com suporte à *framework* escolhida, sendo Bootstrap-React [34] e MUI [35] exemplos destas. Neste caso, escolheu-se utilizar MUI devido aos seus elementos pré-construídos apresentarem um aspeto mais moderno.

5. Implementação do Projeto

Neste capítulo descreve-se a implementação dos componentes do SIG2E apresentados na Figura 1, dividido em duas secções: *Backend* e *Frontend*. Nestas secções são abordados os detalhes da implementação mais relevantes e a organização do código.

5.1 *Backend*

Tal como referido no Capítulo 2 o *Backend* é constituído pela BD, a API e o Servidor. A implementação iniciou-se pela BD de modo a estabelecer a estrutura dos dados e devido à sua independência de implementação perante outros componentes. A API e o Servidor foram construídos simultaneamente tendo como foco o desenvolvimento de cada funcionalidade em paralelo, dando prioridade à finalização de uma funcionalidade.

5.1.1 Base de Dados

Um dos componentes fundamentais do SIG2E é a BD que armazena toda a informação do sistema. A projeção da BD foi realizada através do modelo entidade relação (ER) [36] que representa graficamente as entidades e relações de um sistema de informação, facilitando a sua identificação de um modo simples e concreto [37]. Para a criação do modelo ER foi necessário identificar as entidades do sistema bem como os seus atributos e chaves.

O SIG2E visa gerir equipamentos e espaços, entendendo-se que nessa gestão os gestores podem realizar a adição, eliminação e alteração de equipamentos e de espaços e que os utilizadores apenas podem fazer a sua requisição e outras operações relacionadas. Os espaços referenciam-se a armazéns de equipamentos ou aos laboratórios do ISEL, caracterizados por terem bancadas de trabalho. Uma bancada de trabalho pode ou não ter equipamentos acoplados, pelo qual a sua requisição poderá implicar também a requisição desses equipamentos. Um equipamento requisitado pode ser móvel ou fixo, caso esteja associado a uma bancada. Uma requisição pode ser feita para múltiplos equipamentos e/ou bancadas, possuindo ainda uma prioridade conforme previsto no regulamento das instalações laboratoriais do DEETC do ISEL, e.g., uma reserva de aula é mais importante que uma reserva em tempo livre, pelo qual será necessário indicar o motivo da reserva e a sua duração. Após a obtenção do espaço ou equipamento, este pode apresentar problemas que condicionam a sua utilização, porquanto poderá ser necessário apresentar uma notificação de avaria. Estas características podem ser observadas no modelo ER do SIG2E da Figura 5

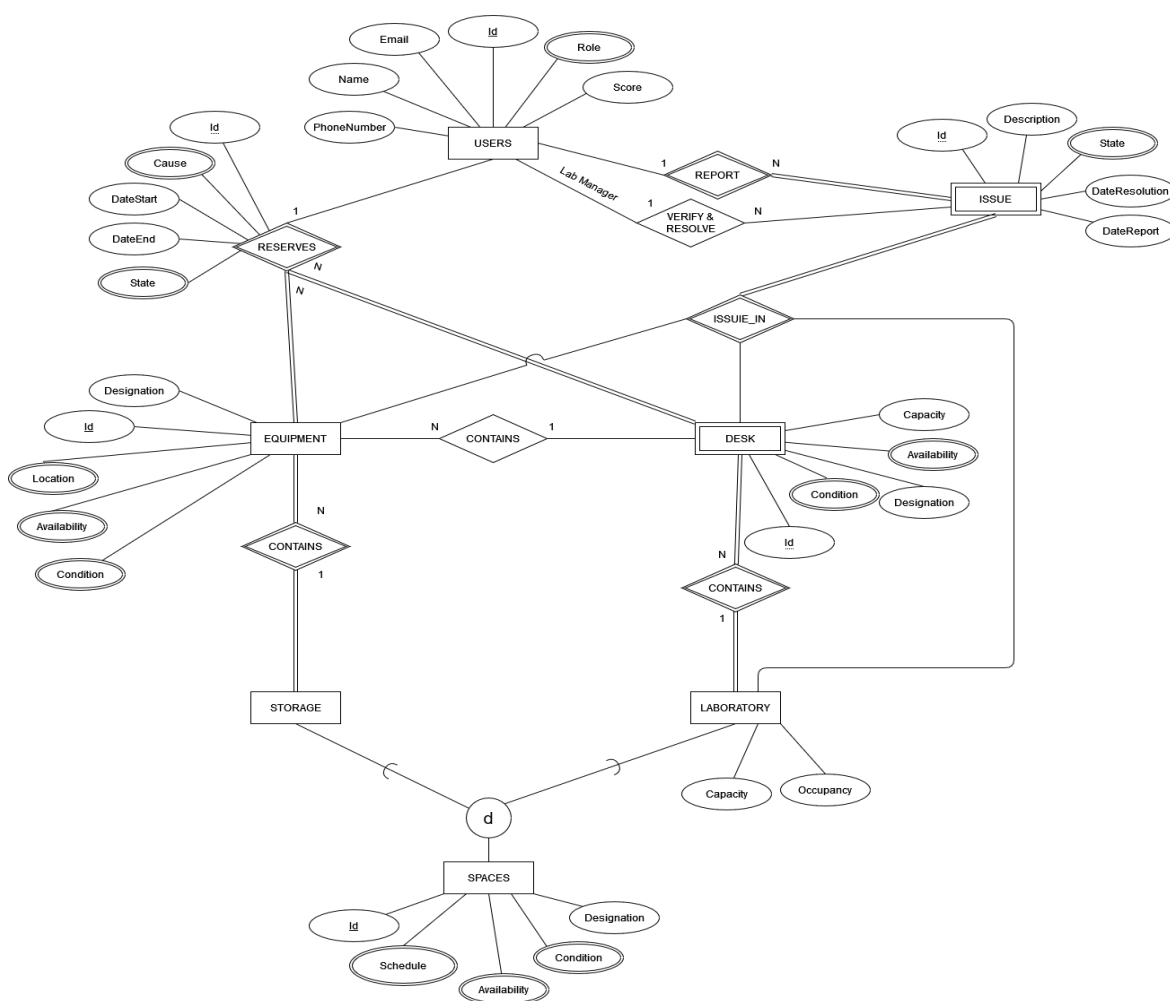


Figura 5 - Diagrama do modelo ER SIG2E

Após a conclusão do modelo ER realizou-se a transformação para o modelo físico seguindo as regras do modelo relacional. Para esta transformação, foram criadas tabelas correspondentes às entidades do modelo ER com colunas correspondentes aos atributos. Neste modelo de dados, os atributos são tipificados e existe o conceito de chaves primárias e chaves estrangeiras. Uma chave primária segue os mesmos princípios de chave do modelo ER e as chaves estrangeiras são elementos que fazem referência a outra tabela de modo a criar uma relação.

5.1.2 Servidor

Outro componente fundamental do sistema é o servidor, é aqui que se encontra a lógica do sistema uma vez que é através deste que todas as ações relacionadas com os equipamentos e laboratórios são analisadas e processadas. Como se pode observar na Figura 6, o servidor está organizado dentro da pasta /src em várias subpastas, cada uma com um propósito diferente.

Na diretoria /configs está armazenado o ficheiro relativo à configuração do acesso do servidor à base de dados. A diretoria /context contém um único ficheiro com a representação em objeto das várias entidades do modelo ER, assim como os seus atributos; estas representações em objeto

tornam-se essenciais dado que a linguagem usada na construção do servidor ser de natureza tipificada. Na diretoria */data* estão armazenados os vários ficheiros referentes a cada entidade que existe no modelo ER da base de dados, e em cada um destes ficheiros existem diferentes funções, cada uma composta por uma *querie SQL* usada para adicionar ou obter dados da base de dados. É na diretoria */services* que se encontra toda a lógica da aplicação, contendo esta também ficheiros referentes às várias entidades do modelo ER, e em cada um destes ficheiros estão as funções que são invocadas pela API, aquando a invocação das funções e dependendo das funções invocadas vão ser feitas diferentes verificações, analisando se os argumentos que a função recebe estão no formato que é esperado e se os dados estão em conformidade, caso não estejam o servidor vai lançar erros, com uma mensagem de forma que a API possa tratar dos mesmos. Na pasta */utils* estão armazenados vários ficheiros que contém funções importadas por vários ficheiros de modo a reduzir a quantidade de código repetido, como por exemplo a função *QueryHandler* que recebe uma *SQL querie* e faz o pedido á base de dados, sendo usada por todos os ficheiros que estão contidos na pasta *data*. Por último o ficheiro *index.ts* contém a inicialização do servidor, no porto 8080.



Figura 6 - Organização dos ficheiros do Backend

5.1.3 API

A API é a componente que completa o *Backend*, disponibilizando rotas que permite ao *Frontend* comunicar com o *Backend* através de pedidos HTTP. A disponibilização das rotas é feita através de *routers* criados pelos diferentes ficheiros que estão contidos na diretoria */routes*, como se pode observar na Figura 7, para serem depois importados e usados no ficheiro *index.ts* aquando da criação do servidor.

Nos diversos ficheiros, contidos na diretoria */api*, está a ser feito a interpretação do pedido que chega, e consequentemente a chamar o servidor para fazer o processamento. No fim do processamento por parte do servidor é aqui que se gera e envia a resposta HTTP para o cliente.

No que diz respeito aos equipamentos, está a ser disponibilizado recursos para fazer diferentes tipos de listagens, indo de encontro às necessidades do *frontend* para quando não é necessário obter toda a informação referente aos equipamentos. Além disso, é possível adicionar, editar e eliminar equipamentos.

Em relação às bancadas, os recursos disponibilizados permitem fazer a listagem, adição, alteração e eliminação de bancadas. Para os laboratórios a API está a disponibilizar rotas para obter, adicionar, editar e apagar.

No caso dos armazéns estão disponíveis recursos para fazer a criação e para apagar. Já para os espaços, existem recursos que permitem a listagem, adição, alteração e remoção.

No que diz respeito às reservas, há recursos para a listar de todas as reservas, ou apenas as reservas dos utilizadores, equipamentos ou bancadas. Além disso, é possível adicionar, alterar, cancelar ou apagar reservas.

Em relação às avarias, existem recursos para fazer diferentes tipos de listagens, assim como para adicionar, editar, marcar uma avaria como verificada ou resolvida e para apagar.

Por último, para o caso dos utilizadores, existem recursos que permitem a listagem, assim como a adição, edição, alteração da pontuação e remoção.



Figura 7 - Ficheiros da diretoria routes

5.2 *Frontend*

No Capítulo 2, o *Frontend* é descrito como a interface de utilização do SIG2E. Para a sua implementação, utilizou-se a *framework* React, conforme discutido na secção 4.4 e optou-se por seguir a abordagem recomendada na documentação [38], e começou-se por criar *mockups* da aplicação, tendo de seguida dividir os elementos em diferentes peças, também chamadas de componentes. Este modelo tem a vantagem de abstrair os componentes, tornando-os assim em peças mutáveis e reutilizáveis pelas diversas páginas, bem como facilitando a sua recomposição. A Figura 8 representa as diversas funcionalidades da aplicação web que são discutidas nas seguintes secções. A Figura 9 mostra as permissões necessárias para aceder a cada um dos recursos.

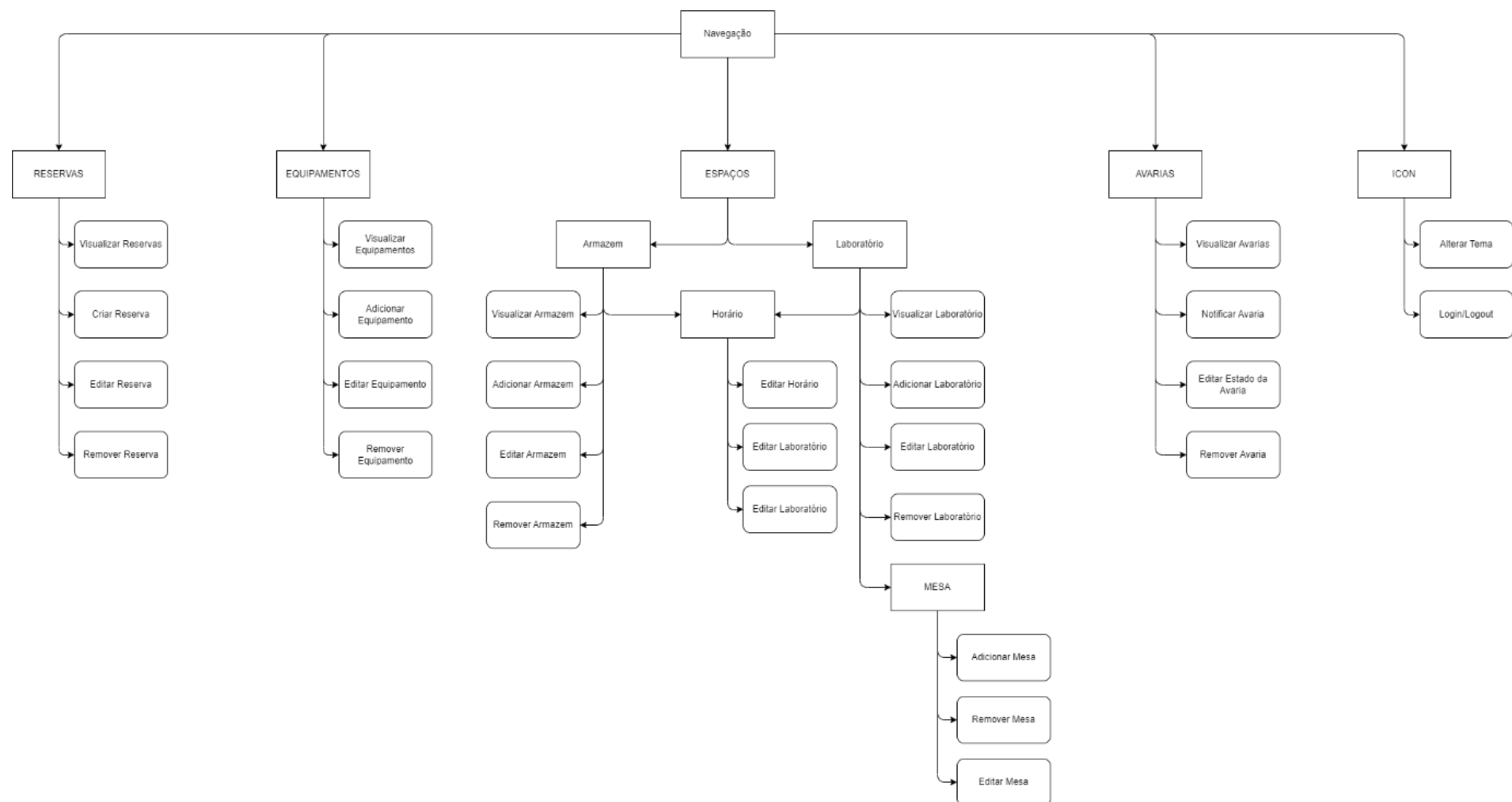


Figura 8 - Fluxo de Utilização da Aplicação Web

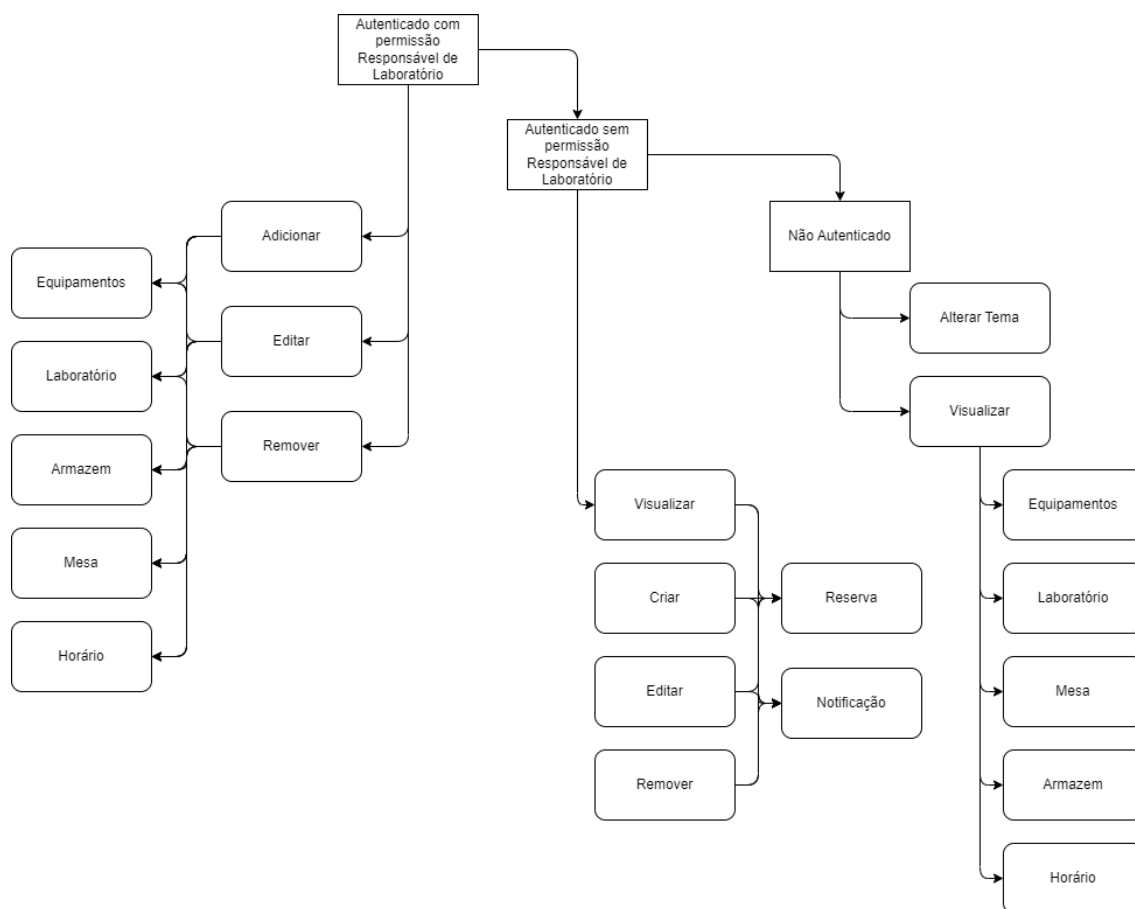


Figura 9 - Diagrama Funcionalidades Autenticadas

Todo o código desenvolvido para implementar o *Frontend* do SIG2E é disponibilizado num repositório na plataforma GitHub [39]. Neste repositório, a pasta raiz tem o nome */src* e o ponto de execução é o ficheiro *index.tsx*, conforme se mostra na Figura 10, que invoca o componente *App* em *App.tsx* de modo a inicializar a aplicação web.

No componente *App* são instanciados os componentes de navegação e as rotas através da extensão *React-Router* [40] de modo a possibilitar que o utilizador consiga alterar a página em exibição. O código dessas páginas está presente na diretoria */pages*.

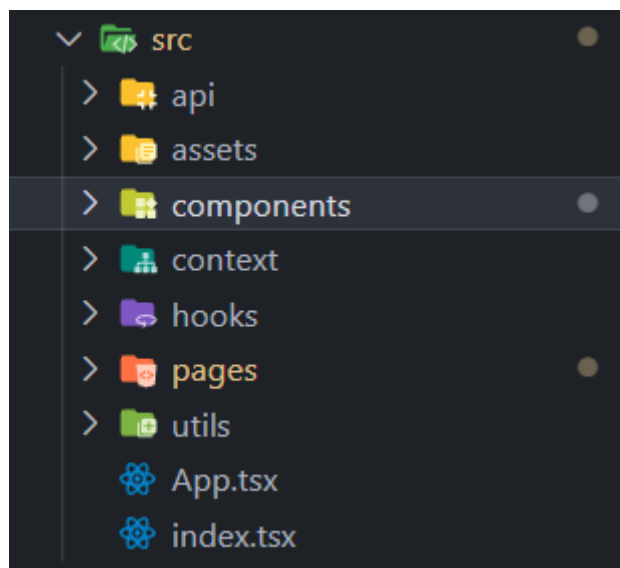


Figura 10 - Estruturação Base de Código Frontend

Uma página é constituída por um ou mais componentes e esses componentes podem, ou não, ser utilizados na construção de diferentes páginas. Para melhorar organizar o código fonte, a diretoria *componentes* armazena todos os componentes genéricos da implementação.

A pasta *assets* contém os elementos visuais não provenientes da *framework* de CSS, tal como imagens e CSS customizado. A diretoria *utils* contém funções utilitárias, enquanto a diretoria *contexto* os tipos que dão contexto à aplicação.

Por fim a comunicação com o *Backend* é conseguida usando métodos para chamadas à API e, de seguida, de modo a manter os resultados obtidos nas posteriores recomposições foram criados hooks [41]. Como estes hooks são utilizados em diversas páginas, decidiu-se criar a diretoria *hooks* centralizando assim o seu código.

5.2.1 Componentes

Com a criação das *mockups* levantadas no Apêndice II – Manual de Descrição Da UI, chegou-se à conclusão de que existem componentes partilhados por diferentes páginas, sendo estes tabelas, *popups* com formulários (em MUI, Dialogs) e linhas e listas colapsáveis. As tabelas e os *popups* já se encontram previamente construídos pela *framework* seleciona (*MUI*), no entanto foram criados componentes tendo estes como base de modo a facilitar a sua utilização e atribuindo-lhe novas funcionalidades. Nas tabelas foi alterado o alinhamento e colocado em negrito os cabeçalhos, adicionando ainda às diferentes células os *icons* para aceder a funcionalidades de editar e remover elementos da tabela. Para os diálogos foi realizada uma implementação contendo as características comuns (cabeçalho e botões para cancelar e submeter) e a capacidade de aparecer e desaparecer do ecrã, sendo o restante conteúdo do formulário recebido por argumento e possibilitando assim cada página de possuir diferentes conteúdos. Por fim foram ainda criados itens de lista e células de tabela com capacidade de

expansão e contração.

6. Conclusão

Neste documento foi apresentada uma solução para o problema atual na reserva de espaços e equipamentos do DEETC do ISEL. O documento foi dividido em quatro principais secções, onde foi descrito o problema, a arquitetura proposta, o estudo e justificação das tecnologias a utilizar e as implementações realizadas. Estas duas últimas secções foram ainda divididas em capítulos correspondentes às componentes do projeto, a BD, servidor, API, aplicação web.

As quatro componentes principais do foram todas estudadas e foi produzida a documentação justificativa do processo de seleção de cada um. Foi também documentado a projeção da BD (Digrama de Entidade-Relacionamento) e a sua implementação. O servidor foi na sua grande maioria implementado, assim como a API, estando apenas em falta a gestão de permissões através de *RBAC*. A componente *Frontend* ficou com a sua base implementada, no entanto ficou com funcionalidades por concluir, especificamente a autenticação, as reservas e as notificações de falha. Apesar destas não terem sido implementadas, as componentes base necessárias para a implementação foram criadas, faltando a integração de pedidos com a API e certos detalhes de UI/UX (do inglês, *user interface/ user experience*).

Ainda assim, através do protótipo do sistema desenvolvido os utilizadores podem consultar todos os equipamentos, bem como as bancadas, laboratórios, reservas e avarias reportadas. As restantes funcionalidades que estavam planeadas e que não foram possíveis de concluir a tempo da entrega do projeto, tais como adicionar, editar ou apagar informação do sistema, pecam apenas pela falta de integração dos pedidos do website à API e de alguns detalhes, uma vez que os componentes necessários para tal estão prontos para implementar a aplicação.

Relativamente à componente de autenticação, que estava planeada para ser implementada em último, não foi iniciada porque os autores concordaram que apenas faria sentido restringir o acesso às funcionalidades quando estas estivessem totalmente implementadas, o que acabou por não acontecer devido a atrasos no desenvolvimento do projeto. Quanto ao desenvolvimento de testes unitários e de integração, face ao tempo existente foi decidido dar prioridade à implementação deixando os testes para serem desenvolvidos posteriormente. Esta decisão foi tomada uma vez que foi necessário realizar alterações nas implementações das componentes durante o processo de desenvolvimento, e desta forma, evitou-se a constante modificação dos testes à medida que as implementações eram ajustadas. Embora os testes sejam importantes, a abordagem adotada permitiu concentrar os esforços em concluir as funcionalidades principais.

Todo o código e documentos produzidos ao longo do projeto podem ser consultados no seguinte repositório na plataforma GitHub, *mediante* pedido para visualização: [39]

6.1 Reflexão crítica

No geral, considera-se que o projeto demonstra a capacidade dos autores em criar um sistema funcional, e as limitações impostas pelo tempo não diminuem os esforços e progressos alcançados. O sistema de gestão de equipamentos e laboratórios desenvolvido pelos alunos

oferece uma base sólida para futuras melhorias e pode ser expandido conforme necessário.

Em conclusão, o projeto evidenciou a capacidade de os autores desenvolverem um sistema informático para gerir espaços e equipamentos. Durante esse processo, foram postos em prática a gestão e planeamento do projeto, como a escolha das tecnologias mais adequadas para o projeto, assim com a utilização e interligação das mesmas com todo o sistema.

Como resultado da execução do projeto, os autores adquiriram capacidades de pesquisa e decisão, como de planeamento e organização. Além disso, foi possível adquirir competências técnicas no domínio das tecnologias usadas, uma vez que algumas nunca tinham sido usadas por nenhum dos autores, por exemplo, a linguagem *TypeScript* para a construção do servidor nem nenhum paradigma *single-page* com a *framework React* para a construção do website.

Os resultados alcançados demonstram o esforço, a dedicação, e a persistência dos autores ao longo do projeto, além de fornecer uma experiência valiosa que contribuirá para o desenvolvimento de software no futuro. O projeto serviu também como um marco importante na jornada académica dos autores

6.2 Trabalho Futuro

Conforme já foi referido, até à data de entrega deste relatório não foi possível realizar todo o trabalho proposto, estando em falta os testes unitários e de integração, a integração da API na aplicação web, a adaptação das páginas consoante as permissões do utilizador, a autenticação, a página de notificação de avarias e detalhes de UI/UX.

Para realizar os testes unitários e de integração poder-se-á utilizar *frameworks* como JEST [42] ou Mocha [43], devendo-se assegurar que cada rota é testada de forma a verificar todos os casos possíveis de erro. Enquanto para os testes unitários, cada função de teste deve testar uma funcionalidade do sistema de forma individual, nos testes de integração, cada função de testes deve executar várias funcionalidades do sistema de forma sequencial e depois verificar o resultado.

Para realizar a autenticação dos utilizadores através do sistema de autenticação single-sign-on (SSO/IdP) do IPL, será necessário estudar o protocolo SAML [44] e coordenar com a equipa do DSIC/IPLNeta entrada em produção desta forma de autenticação. Para a autenticação administrativa será ainda necessário seguir normas de segurança como o armazenamento do *hash* das palavras-chave concatenadas com um *salt* [45]. Ambos estes processos devem utilizar HTTPS de modo a enviar informação sensível encriptada, garantindo assim a sua confidencialidade ponto-a-ponto.

Em relação à página web, é necessário adaptar o conteúdo da página perante as permissões do utilizador autenticado. Isto pode ser feito através de renderização condicional dos componentes autenticados. A página de notificação de avarias deve seguir uma implementação semelhante à das outras e conforme o *mockup* apresentado no Apêndice II – Manual de Descrição Da UI. Por fim, a UI tem de ser refinada de modo a oferecer uma melhor experiência de utilização ao utilizador como a indicação de erros do lado do servidor e nas reservas serem indicados os espaços temporais disponíveis e indisponíveis.

7. Referências

- [1] “<https://journals.sagepub.com/doi/10.1177/0020720916689103?icid=int.sj-full-text.similar-articles.7>,” [Online].
- [2] “<https://www.sciencedirect.com/science/article/abs/pii/S1749772821000087>,” [Online].
- [3] D. Gourley, HTTP: The Definitive Guide: The Definitive Guide.
- [4] M. Fowler, “ACM Digital Library,” 1 10 2003. [Online]. Available: <https://dl.acm.org/doi/10.5555/861282>.
- [5] E. F. Codd, The Relational Model for Database Management: Version 2, Addison-Wesley, 1990.
- [6] B. Medjahed, M. Ouzzani e A. Elmagarmid, 2009. [Online]. Available: <https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1107&context=ccpubs>.
- [7] A. P. R. R. Ragul R, “IJITEE,” 03 2020. [Online]. Available: <https://www.ijitee.org/wp-content/uploads/papers/v9i5/E2418039520.pdf>.
- [8] A. H. A. Hinai, “DiVA,” 31 08 2016. [Online]. Available: <https://www.diva-portal.org/smash/get/diva2:957015/FULLTEXT01.pdf>.
- [9] P. Sadalage e M. Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.
- [10] L. Bassett, Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON.
- [11] J. Pokorny, “ResearchGate,” 12 2011. [Online]. Available: https://www.researchgate.net/publication/221237715_NoSQL_Databases_a_step_to_data_base_scalability_in_Web_environment.
- [12] “Django,” [Online]. Available: <https://www.djangoproject.com/start/overview/>.
- [13] “Flask,” [Online]. Available: <https://flask.palletsprojects.com>.
- [14] “NPM,” [Online]. Available: <https://www.npmjs.com/package/express>.
- [15] “Deno,” [Online]. Available: <https://deno.land/manual@v1.32.3/introduction#feature->

highlights.

- [16] Z. Gao, C. Bird e E. T. Barr, “Microsoft,” 09 2017. [Online]. Available: <https://www.microsoft.com/en-us/research/wp-content/uploads/2017/09/gao2017javascript.pdf>.
- [17] “TypeScript,” [Online]. Available: <https://www.typescriptlang.org/>.
- [18] “NodeJS,” [Online]. Available: <https://nodejs.org/en/about>.
- [19] “<https://deno.land/>,” [Online].
- [20] “GitHub,” [Online]. Available: <https://github.com/mozilla/rhino>.
- [21] G. Jadhav e F. Gonsalves, “IRJET,” 06 2020. [Online]. Available: <https://www.irjet.net/archives/V7/i6/IRJET-V7I61149.pdf>.
- [22] “NPM,” [Online]. Available: <https://www.npmjs.com/package/@hapi/hapi>.
- [23] “NPM,” [Online]. Available: <https://www.npmjs.com/package/koa>.
- [24] F. Copes, The Express Handbook, 2018.
- [25] “Hapi,” [Online]. Available: https://hapi.dev/tutorials/caching/?lang=en_US.
- [26] “KoaJS,” [Online]. Available: <https://koajs.com/>.
- [27] A. Soni e V. Ranga, “ResearchGate,” 07 2019. [Online]. Available: https://www.researchgate.net/publication/335419384_API_Features_Individualizing_of_Web_Services_REST_and_SOAP.
- [28] G. M. Waleed e R. B. Ahmad, “ResearchGate,” 12 2008. [Online]. Available: https://www.researchgate.net/publication/224386981_Security_protection_using_simple_object_access_protocol_SOAP_messages_techniques.
- [29] F. Halili e E. Ramadani, “ResearchGate,” 28 02 2018. [Online]. Available: https://www.researchgate.net/publication/323456206_Web_Services_A_Comparison_of_Soap_and_Rest_Services.
- [30] “<https://angular.io/>,” [Online].
- [31] “<https://react.dev/>,” [Online].

- [32] "<https://vuejs.org/>," [Online].
- [33] M. Bauer, "DiVa," 07 2021. [Online]. Available: <https://uu.diva-portal.org/smash/get/diva2:1591850/FULLTEXT01.pdf>.
- [34] "<https://react-bootstrap.netlify.app/>," [Online].
- [35] "<https://mui.com/>," [Online].
- [36] R. E. a. S. Navathe, Fundamentals of Database, USA: Addison-Wesley Publishing Company, 2010.
- [37] A. Badia, 2001. [Online]. Available: https://www.researchgate.net/publication/220415410_Entity-Relationship_modeling_revisited.
- [38] "<https://react.dev/learn/thinking-in-react>," [Online].
- [39] "<https://github.com/jmpTeixeira02/sig2e>," [Online].
- [40] "<https://reactrouter.com/en/main>," [Online].
- [41] "<https://react.dev/reference/react>," [Online].
- [42] Jest, "Jest," [Online]. Available: <https://jestjs.io/>.
- [43] Mocha, "Mocha," [Online]. Available: <https://mochajs.org/>.
- [44] "https://www.researchgate.net/publication/221609828_Formal_analysis_of_SAML_20_web_browser_single_sign-on," [Online].
- [45] "https://www.academia.edu/35342322/Overview_Of_Web_Password_Hashing_Using_Salt_Technique," [Online].
- [46] "Visual Paradigm," [Online]. Available: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>.
- [47] "Adatum," [Online]. Available: <https://www.sciencedirect.com/science/article/abs/pii/S1749772821000087>.
- [48] S. Journals. [Online]. Available: <https://journals.sagepub.com/doi/10.1177/0020720916689103?icid=int.sj-full-text.similar->

articles.7.

- [49] J. Makkonen, “Aalto University,” 12 11 2017. [Online]. Available: https://aaltodoc.aalto.fi/bitstream/handle/123456789/29224/master_Makkonen_Joni_2017.pdf.
- [50] K. R. Srinath, “IRJET,” 12 2017. [Online]. Available: <https://www.irjet.net/archives/V4/i12/IRJET-V4I1266.pdf>.
- [51] “Tech Insider,” Sun Microsystems, 1995. [Online]. Available: <https://www.tech-insider.org/java/research/acrobat/9503.pdf>.
- [52] “Deno,” [Online]. Available: <https://deno.com/runtime>.
- [53] D. Marinescu e M. Kaufmann, Cloud Computing – Theory and Practice, 2nd Edition, 2017.

Apêndice I – Manual de Documentação da API

Equipamentos:

GET /equipment – Obtém todos os equipamentos

- Resultado: Array de equipamentos. Cada equipamento é um objeto JSON composto por:
 - id: inteiro
 - deskId: null | inteiro
 - location: string
 - equipmentnr: inteiro
 - designation: string
 - availability: booleano
 - condition: string

GET /equipment/desk/:deskid – Obtém todos os equipamentos contidos na bancada passada em deskid

- Parâmetros:
 - Path:
 - deskid: inteiro
- Resultado: Array de equipamentos. Cada equipamento é um objeto JSON composto por:
 - id: inteiro
 - deskId: null | inteiro
 - location: string
 - equipmentnr: inteiro
 - designation: string
 - availability: booleano
 - condition: string

GET /equipment/name/:name – Obtém todos os equipamentos que tenham o nome igual valor passado no parâmetro name

- Parâmetros:
 - Path:
 - name: string
- Resultado: Array de equipamentos. Cada equipamento é um objeto JSON composto por:
 - id: inteiro
 - deskId: null | inteiro
 - location: string
 - equipmentnr: inteiro

- designation: string
- availability: booleano
- condition: string

GET /equipment/distinct – Obtém todos os nomes diferentes dos equipamentos

- Resultado: Array de strings com os nomes dos equipamentos que existem.

GET /equipment/distinct/count – Obtém todos os nomes diferentes dos equipamentos e o número de equipamentos com o mesmo nome

- Resultado: Array de objetos JSON. Cada objeto é composto por:
 - designation: string
 - num: inteiro

GET /equipment/separated – Obtém todos os equipamentos separados pelo nome

- Resultado: Objeto JSON. Cada atributo é a designação dos equipamentos e o valor é um array de equipamentos. Cada equipamento é por sua vez um objeto JSON composto por:
 - id: inteiro
 - deskId: null | inteiro
 - location: string
 - equipmentnr: inteiro
 - designation: string
 - availability: booleano
 - condition: string

GET /equipment/:id – Obtém o equipamento passado no id

- Parâmetros:
 - Path:
 - id: inteiro
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - deskId: null | inteiro
 - location: string
 - equipmentnr: inteiro
 - designation: string
 - availability: booleano
 - condition: string

POST /equipment – Adiciona um equipamento

- Parâmetros:
 - Body:
 - location: string
 - deskid: inteiro

- designation: string
- condition: string
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - deskId: null | inteiro
 - location: string
 - equipmentnr: inteiro
 - designation: string
 - availability: booleano
 - condition: string

PUT /equipment/:id – Editar o equipamento passado no id

- Parâmetros:
 - Path:
 - id: inteiro
 - Body:
 - deskid: string
 - location: string
 - equipmentnr: inteiro
 - designation: string
 - availability: booleano
 - condition: string
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - deskId: null | inteiro
 - location: string
 - equipmentnr: inteiro
 - designation: string
 - availability: booleano
 - condition: string

DELETE /equipment:id – Apagar o equipamento passado no id

- Parâmetros:
 - Path:
 - id: inteiro
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - deskId: null | inteiro
 - location: string
 - equipmentnr: inteiro
 - designation: string

- availability: booleano
- condition: string

Bancadas:

GET /desk – Obtém todas as bancadas

- Resultado: Array de bancadas. Cada bancada é um objeto JSON composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro
 - designation: string
 - capacity: inteiro
 - availability: booleano
 - condition: string

GET /desk/lab/:labid – Obtém todas as bancadas do laboratório passado no labid

- Parâmetros:
 - Path:
 - labid: inteiro
- Resultado: Array de bancadas. Cada bancada é um objeto JSON composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro
 - designation: string
 - capacity: inteiro
 - availability: booleano
 - condition: string

GET /desk/separated – Obtém todas as bancadas separadas pelo laboratório

- Resultado: Objeto JSON, cada atributo é o id do laboratório e o valor é um array de bancadas. Cada bancada é por sua vez um objeto JSON composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro
 - designation: string
 - capacity: inteiro
 - availability: booleano
 - condition: string

GET /desk/:id – Obtém a bancada passada no id

- Parâmetros:
 - Path:
 - id: inteiro
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro
 - designation: string
 - capacity: inteiro
 - availability: booleano
 - condition: string

POST /desk - Adiciona uma bancada

- Parâmetros:
 - Body:
 - labid: inteiro
 - designation: string
 - capacity: inteiro
 - condition: string
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro
 - designation: string
 - capacity: inteiro
 - availability: booleano
 - condition: string

PUT /desk/:id – Editar a bancada passada no id

- Parâmetros:
 - Path:
 - desk: inteiro
 - Body:
 - labid: inteiro
 - designation: string
 - capacity: inteiro
 - condition: string
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro

- designation: string
- capacity: inteiro
- availability: booleano
- condition: string

DELETE /desk/:id – Apaga a bancada passada no id

- Parâmetros:
 - Path:
 - id: inteiro
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro
 - designation: string
 - capacity: inteiro
 - availability: booleano
 - condition: string

Laboratórios:

GET /lab – Obtém todos os laboratórios

- Resultado: Array de laboratórios. Cada laboratório é um objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de dois índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano
 - spaceid: inteiro
 - capacity: inteiro
 - occupancy: inteiro
 - condition: string
 - desks: array de objetos de bancadas. Cada objeto JSON é composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro
 - designation: string
 - capacity: inteiro
 - availability: booleano

- condition: string

GET /lab/:id – Obtém o laboratório passado no id

- Parâmetros:
 - Path:
 - id: inteiro
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de dois índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano
 - spaceid: inteiro
 - capacity: inteiro
 - occupancy: inteiro
 - condition: string
 - desks: array de objetos de bancadas. Cada objeto JSON é composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro
 - designation: string
 - capacity: inteiro
 - availability: booleano
 - condition: string

POST /lab – Adicionar um laboratório

- Parâmetros:
 - Body:
 - schedule: string de um array de 7 índices e em cada índice em array de 2 índices de strings
 - designation: string
 - capacity: inteiro
 - occupancy: inteiro
 - availability: booleano
 - condition: string
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de

dois índices de strings

- designation: string
- spacetype: string
- availability: booleano
- spaceid: inteiro
- capacity: inteiro
- occupancy: inteiro
- condition: string
- desks: array de objetos de bancadas. Cada objeto JSON é composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro
 - designation: string
 - capacity: inteiro
 - availability: booleano
 - condition: string

PUT /lab/:id – Editar o laboratório passado no id

- Parâmetros:
 - Path:
 - Id: inteiro
 - Body:
 - schedule: string de um array de 7 índices e em cada índice em array de 2 índices de strings
 - designation: string
 - capacity: inteiro
 - occupancy: inteiro
 - availability: booleano
 - condition: string
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de dois índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano
 - spaceid: inteiro
 - capacity: inteiro
 - occupancy: inteiro

- condition: string
- desks: array de objetos de bancadas. Cada objeto JSON é composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro
 - designation: string
 - capacity: inteiro
 - availability: booleano
 - condition: string

DELETE /lab/:id – Apagar o laboratório passado no id

- Parâmetros:
 - Path:
 - Id: inteiro
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de dois índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano
 - spaceid: inteiro
 - capacity: inteiro
 - occupancy: inteiro
 - condition: string
 - desks: array de objetos de bancadas. Cada objeto JSON é composto por:
 - id: inteiro
 - labid: inteiro
 - desknr: inteiro
 - designation: string
 - capacity: inteiro
 - availability: booleano
 - condition: string

Avarias:

GET /issue – Obter todas as avarias

- Resultado: Array de avarias. Cada avaria é um objeto JSON composto

por:

- id: inteiro
- useridreport: string
- useridverification: string
- useridresolution: string
- equipmentid: null | inteiro
- deskid: null | inteiro
- labid: null | inteiro
- description: string
- state: string
- datereport: data
- dateverification: data
- dateresolution: data

GET /issue:id – Obtém a avaria passada no id

- Parâmetros:
 - Path:
 - id: inteiro
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - useridreport: string
 - useridverification: string
 - useridresolution: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - labid: null | inteiro
 - description: string
 - state: string
 - datereport: data
 - dateverification: data
 - dateresolution: data

GET /issue/equipment/:equipmentid – Obtém as avarias do equipamento passado no parâmetro equipmentid

- Parâmetros:
 - Path:
 - equipmentid: inteiro
- Resultado: Array de avarias. Cada avaria é um objeto JSON composto por:
 - id: inteiro
 - useridreport: string
 - useridverification: string

- useridresolution: string
- equipmentid: null | inteiro
- deskid: null | inteiro
- labid: null | inteiro
- description: string
- state: string
- datereport: data
- dateverification: data
- dateresolution: data

GET /issue/desk/:deskid – Obtém todas as avarias da bancada passada no parâmetro deskid

- Parâmetros:
 - Path:
 - deskid: inteiro
- Resultado: Array de avarias. Cada avaria é um objeto JSON composto por:
 - id: inteiro
 - useridreport: string
 - useridverification: string
 - useridresolution: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - labid: null | inteiro
 - description: string
 - state: string
 - datereport: data
 - dateverification: data
 - dateresolution: data

GET /issue/lab/:labid – Obtém todas as avarias do laboratório passado no parâmetro labid

- Parâmetros:
 - Path:
 - labid: inteiro
- Resultado: Array de avarias. Cada avaria é um objeto JSON composto por:
 - id: inteiro
 - useridreport: string
 - useridverification: string

- useridresolution: string
- equipmentid: null | inteiro
- deskid: null | inteiro
- labid: null | inteiro
- description: string
- state: string
- datereport: data
- dateverification: data
- dateresolution: data

GET /issue/user/:userid – Obtém todas as avarias reportadas pelo utilizador passado no parâmetro userid

- Parâmetros:
 - Path:
 - userid: string
- Resultado: Array de avarias. Cada avaria é um objeto JSON composto por:
 - id: inteiro
 - useridreport: string
 - useridverification: string
 - useridresolution: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - labid: null | inteiro
 - description: string
 - state: string
 - datereport: data
 - dateverification: data
 - dateresolution: data

POST /issue – Adicionar uma avaria

- Parâmetros:
 - Body:
 - useridreport: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - labid: null | inteiro
 - description: string
 - datereport: data
- Resultado: Objeto JSON composto por:

- id: inteiro
- useridreport: string
- useridverification: string
- useridresolution: string
- equipmentid: null | inteiro
- deskid: null | inteiro
- labid: null | inteiro
- description: string
- state: string
- datereport: data
- dateverification: data
- dateresolution: data

PUT /issue/:id – Editar a avaria passada no parâmetro id

- Parâmetros:
 - Path:
 - id: inteiro
 - Body:
 - useridreport: string
 - useridverification: string
 - useridresolution: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - labid: null | inteiro
 - description: string
 - state: string
 - datereport: data
 - dateverification: data
 - dateresolution: data
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - useridreport: string
 - useridverification: string
 - useridresolution: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - labid: null | inteiro
 - description: string
 - state: string
 - datereport: data

- dateverification: data
- dateresolution: data

PUT /issue/:id/verify – Alterar o estado da avaria passada no id para verificado

- Parâmetros:
 - Path:
 - id: inteiro
 - Body:
 - useridverification: string
 - dateverification: data
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - useridreport: string
 - useridverification: string
 - useridresolution: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - labid: null | inteiro
 - description: string
 - state: string
 - datereport: data
 - dateverification: data
 - dateresolution: data

PUT /issue/:id/resolve – Alterar o estado da avaria passada no id para resolvido

- Parâmetros:
 - Path:
 - id: inteiro
 - Body:
 - useridresolution: string
 - dateresolution: data
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - useridreport: string
 - useridverification: string
 - useridresolution: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - labid: null | inteiro
 - description: string

- state: string
- datereport: data
- dateverification: data
- dateresolution: data

DELETE /issue:id – Apagar a avaria passada no parâmetro id

- Parâmetros:
 - Path:
 - id: inteiro
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - useridreport: string
 - useridverification: string
 - useridresolution: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - labid: null | inteiro
 - description: string
 - state: string
 - datereport: data
 - dateverification: data
 - dateresolution: data

Reservas:

GET /reservation/:id – Obtém todas as reservas

- Resultado: Array de reservas. Cada reserva é um objeto JSON composto por:
 - id: inteiro
 - userid: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - cause: string
 - datestart: data
 - dateend: data
 - state: string

GET /reservation/:id – Obtém a reserva passada no id

- Parâmetros:
 - Path:
 - id: inteiro

- Resultado: Objeto JSON composto por:
 - id: inteiro
 - userid: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - cause: string
 - datestart: data
 - dateend: data
 - state: string

GET /reservation/user/:userid – Obtém todas as reservas do utilizador passado no parâmetro userid

- Parâmetros:
 - Path:
 - userid: string
- Resultado: Array de reservas. Cada reserva é um objeto JSON composto por:
 - id: inteiro
 - userid: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - cause: string
 - datestart: data
 - dateend: data
 - state: string

GET /reservation/equipment/:equipmentid – Obtém todas as reservas do equipamento passado no parâmetro equipmentid

- Parâmetros:
 - Path:
 - equipmentid: inteiro
- Resultado: Array de reservas. Cada reserva é um objeto JSON composto por:
 - id: inteiro
 - userid: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - cause: string
 - datestart: data
 - dateend: data

- state: string

GET /reservation/desk/:deskid – Obtém todas as reservas da bancada passada no parâmetro deskid

- Parâmetros:
 - Path:
 - deskid: inteiro
- Resultado: Array de reservas. Cada reserva é um objeto JSON composto por:
 - id: inteiro
 - userid: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - cause: string
 - datestart: data
 - dateend: data
 - state: string

POST /reservation – Adicionar uma reserva

- Parâmetros:
 - Body:
 - userid: string
 - equipmentid: inteiro
 - deskid: inteiro
 - cause: string
 - datestart: daa
 - dateend: data
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - userid: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - cause: string
 - datestart: data
 - dateend: data
 - state: string

PUT /reservation/:id – Editar uma reserve passada no id

- Parâmetros:

- Path:
 - id: inteiro
 - Body:
 - userid: string
 - equipmentid: inteiro
 - deskid: inteiro
 - cause: string
 - datestart: daa
 - dateend: data
 - state: string
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - userid: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - cause: string
 - datestart: data
 - dateend: data
 - state: string

PUT /reservation/:id/cancel – Cancelar a reserva passada no id

- Parâmetros:
 - Path:
 - id: inteiro
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - userid: string
 - equipmentid: null | inteiro
 - deskid: null | inteiro
 - cause: string
 - datestart: data
 - dateend: data
 - state: string

DELETE /reservation/:id – Apagar a reserva passada no id

- Parâmetros:
 - Path:
 - id: inteiro
- Resultado: Objeto JSON composto por:

- id: inteiro
- userid: string
- equipmentid: null | inteiro
- deskid: null | inteiro
- cause: string
- datestart: data
- dateend: data
- state: string

Espaços:

GET /space – Obtém todos os espaços

- Resultado: Array de espaços. Cada espaço é um objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de dois índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano

GET /space:id – Obtém o espaço passado no id

- Parâmetros:
 - Path:
 - id: inteiro
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de dois índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano

POST /space – Adicionar um espaço

- Parâmetros:
 - Body:
 - schedule: string de um array de 7 índices e em cada índice em array de 2 índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano

- Resultado: Objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de dois índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano

PUT /space/:id – Editar o espaço passado no id

- Parâmetros:
 - Path:
 - id: string
 - Body:
 - schedule: string de um array de 7 índices e em cada índice em array de 2 índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de dois índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano

DELETE /space/:id – Apagar o espaço passado no id

- Parâmetros:
 - Path:
 - id: string
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de dois índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano

Armazéns:

POST /storage– Adicionar um armazém

- Parâmetros:
 - Body:
 - schedule: string de um array de 7 índices e em cada índice em array de 2 índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de dois índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano

DELETE /storage/:id – Apagar o armazém passado no id

- Parâmetros:
 - Path:
 - id: inteiro
- Resultado: Objeto JSON composto por:
 - id: inteiro
 - schedule: array com 7 índices, cada índice contém um array de dois índices de strings
 - designation: string
 - spacetype: string
 - availability: booleano

Utilizadores:

GET /user – Obtém todos os utilizadores

- Resultado: Array de utilizadores. Cada utilizador é um objeto JSON composto por:
 - id: string
 - email: string
 - name: string
 - role: string
 - phonenumber: string
 - score: null | inteiro

GET /user/:id – Obtém o utilizador passado no id

- Parâmetros:

- Path:
 - id: string
- Resultado: Objeto JSON composto por:
 - id: string
 - email: string
 - name: string
 - role: string
 - phonenumber: string
 - score: null | inteiro

POST /user – Adicionar um utilizador

- Parâmetros:
 - Body:
 - id: string
 - email: string
 - name: string
 - role: string
 - phonenumber: string
 - score: null | inteiro
- Resultado: Objeto JSON composto por:
 - id: string
 - email: string
 - name: string
 - role: string
 - phonenumber: string
 - score: null | inteiro

PUT /user/:id – Editar o utilizador passado no id

- Parâmetros:
 - Path:
 - id: string
 - Body:
 - id: string
 - email: string
 - name: string
 - role: string
 - phonenumber: string
 - score: null | inteiro
- Resultado: Objeto JSON composto por:
 - id: string

- email: string
- name: string
- role: string
- phonenumber: string
- score: null | inteiro

PUT /user/:id/score – Alterar a pontuação do utilizador passado no id

- Parâmetros:
 - Path:
 - id: string
 - Body:
 - score: null | inteiro
- Resultado: Objeto JSON composto por:
 - id: string
 - email: string
 - name: string
 - role: string
 - phonenumber: string
 - score: null | inteiro

DELETE /user/:id – Apagar o utilizador passado no id

- Parâmetros:
 - Path:
 - id: string
- Resultado: Objeto JSON composto por:
 - id: string
 - email: string
 - name: string
 - role: string
 - phonenumber: string
 - score: null | inteiro

Apêndice II – Manual de Descrição Da UI

1. Navegação

Para a navegação na aplicação web foram desenvolvidas duas barras de navegação, uma lateral e uma de topo. A barra lateral é expansível e disponibiliza o acesso às diferentes funcionalidades do SIG2E, enquanto a barra de topo possibilita ao utilizador autenticar-se, alterar o tema da página e retornar à página raiz.

Na Figura 11 apresenta-se a imagem da barra de navegação lateral quando o utilizador coloca o seu cursor em cima desta, realizando a sua expansão, e na Figura 12 quando o cursor se encontra noutro local. Em ambas estas barras, quando o utilizador clica num dos itens reserva, equipamentos, espaços ou avarias, a página em exibição é alterada para a correspondente ao item selecionados.

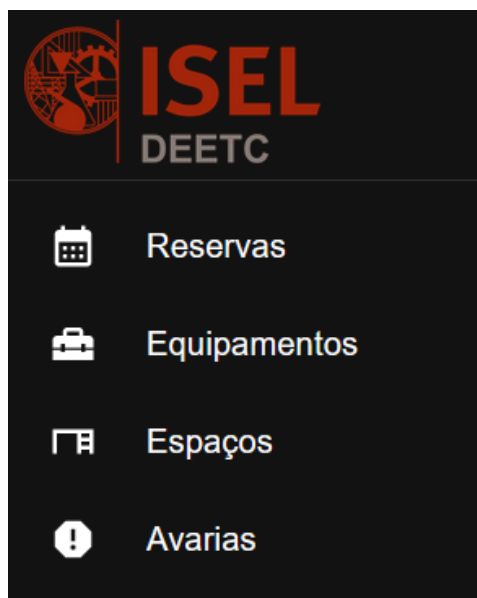


Figura 11 - Barra de Navegação Lateral Expandida

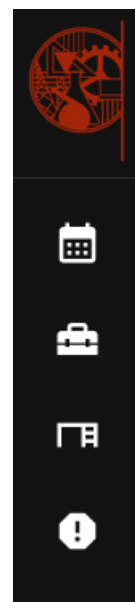


Figura 12 - Barra de Navegação Lateral Escondida

A barra de topo é constituída por um elemento responsável por levar o utilizador de volta à página raiz e por um *icon* clicável, como se pode ver na Figura 13. O *icon* clicável pode apresentar a imagem de utilizador no caso de este estar autenticado (Figura 15), ou uma imagem padrão no caso contrário (Figura 14). Ao clicar neste *icon* é possível realizar a autenticação e alterar o tema de exibição, conforme se ilustra na Figura 14 e Figura 15.



Figura 13 - Barra de Navegação de Topo

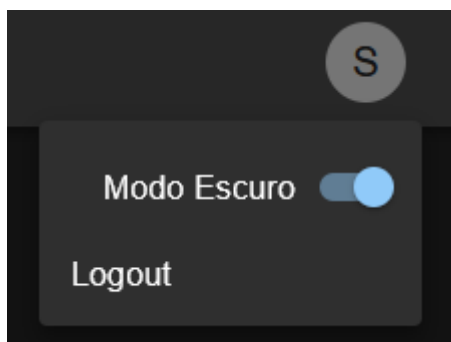


Figura 15 - Menu Utilizador Autenticado

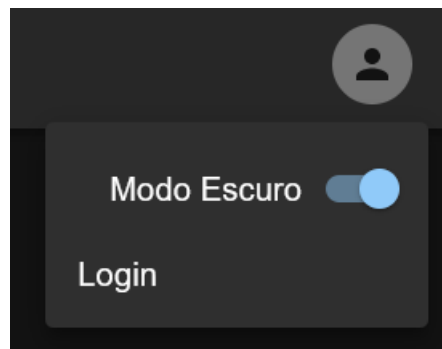


Figura 14 - Menu Utilizador não Autenticado

2. Autenticação

A autenticação pode ser realizada de dois modos, através de uma conta criada administrativamente no SIG2E ou através da autenticação ISEL. Tal como referido na secção 1. Navegação, a autenticação pode ser realizada ao clicar no botão de *Login* presente no *ícon* do utilizador não autenticado, que por sua vez abrirá um formulário semelhante ao da Figura 16

A dark-themed login form titled 'LOGIN' in white. It contains two input fields: 'Nome de Utilizador' and 'Palavra Passe'. Below these is a button with a circular icon on the left and the text 'Autenticar com ISEL' on the right.

Figura 16 - Formulário Autenticação

3. Reservas

A página de reservas permite a realização de novas reservas e a consulta de pedidos de reserva, conforme se ilustra na Figura 17. No entanto, as reservas existentes são mostradas de forma diferente em função do perfil do utilizador autenticado. Os utilizadores com permissões de gestor têm acessos a todas as reservas registados no sistema, enquanto os outros utilizadores apenas

têm acesso aos seus pedidos de reserva.

RESERVAR					
Equipamento	Laboratório	Mesa	Início	Fim	Estado
Placa SDP16 1	F.0.46	5	2023-04-26T09:00:00.000Z	2023-04-26T11:00:00.000Z	active
Placa SDP16 2	F.0.46	2	2023-04-26T09:00:00.000Z	2023-04-26T11:00:00.000Z	canceled
Placa SDP16 1	F.0.46	1	2023-04-26T09:00:00.000Z	2023-04-26T11:00:00.000Z	concluded
Placa SDP16 2	F.0.46	5	2023-04-26T09:00:00.000Z	2023-04-26T11:00:00.000Z	pending

Figura 17 - Tabela das Reservas

Para criar uma reserva é utilizado um formulário, representado na Figura 18, que apresenta ao utilizador os equipamentos, laboratórios e mesas disponíveis e pede a introdução dos dados relativos à data e ao horário pretendido para a reserva.

Adicionar Reserva

Equipamentos

Laboratório

Mesa

Data

Início

Fim

CANCELAR

ADICIONAR

Figura 18 - Criar Reserva

4. Equipamentos

Na página de equipamentos, é possível o utilizador consultar todos os equipamentos existentes no sistema e, caso este possua permissões, pode ainda aceder às funcionalidades de adicionar, editar e remover equipamentos.

A página mostrar os equipamentos registados no sistema usando um formato de uma tabela, conforme se apresenta na Figura 19. Nesta tabela, cada linha discrimina o tipo de equipamento, o número de equipamentos disponíveis para utilização e a quantidade total de equipamentos existentes. Ao clicar no tipo de equipamento é possível obter informação mais detalhada, tal como a disponibilidade e a localização de cada um dos equipamentos do tipo selecionado, conforme se ilustra na Figura 20. Caso o utilizador possua permissões, ao colocar o cursor sobre a linha aparecerão as opções para editar e remover a linha selecionada.


Equipamento	Disponíveis	Total	
▼ Placa SDP16	24	24	 
▼ Placa ATB	14	19	
▼ Placa uLICx	12	12	
▼ Placa DE10-Lite	12	12	
▼ Fonte de alimentação DF-1731SB	16	16	
▼ Fonte de alimentação Velleman PS 613	16	16	

Figura 19 - Tabela dos Equipamentos



Equipamento		Disponíveis	Total
^ Placa SDP16		24	24
Numero do Equipamento	Localização	Disponibilidade	 
1	Sala F.0.49	Disponível	
2	Sala F.0.49	Disponível	
3	Sala F.0.49	Disponível	
4	Sala F.0.49	Disponível	
5	Sala F.0.49	Disponível	
6	Sala F.0.49	Disponível	
7	Sala F.0.49	Disponível	
8	Sala F.0.49	Disponível	

Figura 20 - Detalhes de um Equipamento

A adição de equipamentos ao sistema é feita com base num formulário. Neste formulário o utilizador poderá seleccionar se tenciona adicionar um equipamento móvel, conforme se mostra na Figura 21, ou um equipamento de bancada, conforme indicado na Figura 22. Neste caso, o equipamento é um campo livre com sugestões, isto significa que o utilizador poderá seleccionar

um dos valores disponibilizados ou escrever o seu. No entanto, os campos de Condição, Localização e Bancada são campos com valor pré-definido, ou seja, o utilizador apenas pode seleccionar um dos valores disponibilizados no campo.

The screenshot shows a dark-themed form titled 'Adicionar Equipamento'. Under the heading 'Tipo de Equipamento', there are two radio buttons: 'Móvel' (selected) and 'De bancada'. Below this, there are three input fields: a text box for 'Equipamento', a dropdown menu for 'Condição', and a dropdown menu for 'Localização'. At the bottom right, there are two buttons: 'CANCELAR' and 'ADICIONAR'.

Figura 21 - Adicionar Equipamento Móvel

The screenshot shows a dark-themed form titled 'Adicionar Equipamento'. Under the heading 'Tipo de Equipamento', there are two radio buttons: 'Móvel' and 'De bancada' (selected). Below this, there are four input fields: a text box for 'Equipamento', a dropdown menu for 'Condição', a dropdown menu for 'Laboratório', and a dropdown menu for 'Bancada'. At the bottom right, there are two buttons: 'CANCELAR' and 'ADICIONAR'.

Figura 22 - Adicionar Equipamento de Bancada

5. Espaços

Na página de espaços, à semelhança dos equipamentos, o utilizador pode consultar todos os espaços registados no sistema, podendo estes ser laboratórios ou locais de armazenamento e, caso este possua permissões pode ainda aceder às funcionalidades de adicionar, editar e remover espaços. A página mostra os espaços organizados em duas tabelas, uma para os laboratórios e outra para os restantes espaços, conforme ilustrado na Figura 23. Na tabela relativa aos laboratórios, cada linha apresenta informação sobre a lotação atual, lotação máxima e disponibilidade para novas reservas. Na tabela relativa aos outros espaços, cada linha dos armazéns apenas informa sobre a disponibilidade atual desse espaço.

Ambas as tabelas possuem elementos expansíveis que possibilitam mostrar informação mais detalhada sobre os espaços. Ao clicar num espaço do tipo laboratório é possível obter informação sobre o seu horário de funcionamento e a sua organização em bancadas, conforme se ilustra nas Figura 24 e Figura 25, respetivamente. As bancadas são identificadas por número e possuem uma capacidade máxima e disponibilidade. Caso o utilizador possua permissões adequadas, ao colocar o cursor sobre uma linha de qualquer das duas tabelas, aparecerão as opções para editar e remover a linha selecionada, conforme se pode observar nas Figura 23, Figura 24 e Figura 25. Selecionando essa linha, poderá ainda adicionar novos horários ou bancadas para o espaço representado. A adição de um novo espaço é feita selecionando o botão no topo da página representado na Figura 23.

ADICIONAR ESPAÇO			
Laboratório	Lotação Atual	Lotação Máxima	Disponibilidade
▼ F.0.46	0	21	Disponível
▼ F.0.47	0	16	Disponível
▼ F.1.09	0	15	Disponível
Sala	Disponibilidade		
▼ F.0.49	Disponível		

Figura 23 - Visualização Laboratório e Sala Armazenamento

^ Horários			
ADICIONAR HORÁRIO			
Dia da Semana	Hora de abertura	Hora de fecho	
Segunda-Feira	08:00	20:00	✎ 🗑
Terça-Feira	08:00	20:00	
Quarta-Feira	08:00	20:00	
Quinta-Feira	08:00	20:00	
Sexta-Feira	08:00	20:00	
Sabado	08:00	20:00	
Domingo	08:00	20:00	

Figura 24 - Visualização de Horários

Laboratório	Lotação Atual	Lotação Máxima	Disponibilidade
^ F.0.46	0	21	Disponível
v Horários ^ Bancadas ADICIONAR BANCADA			
Numero da bancada	Capacidade máxima	Disponibilidade	
1	2	Disponível	 
2	2	Disponível	
3	2	Disponível	

Figura 25 - Visualização de Bancadas

A adição de equipamentos ao sistema é feita com base num formulário. Neste formulário o utilizador poderá selecionar se tenciona adicionar um equipamento móvel, conforme se mostra na Figura 21, ou um equipamento de bancada, conforme indicado na Figura 22. Neste caso, o equipamento é um campo livre com sugestões, isto significa que o utilizador poderá selecionar

Para adicionar um espaço, o utilizador terá de preencher um dos formulários apresentados nas Figura 26 e Figura 27. Para ambos os formulários, será necessário especificar a localização da sala, a sua condição atual e as horas de abertura e fecho para os diferentes dias da semana. Caso se pretenda definir horários de funcionamento distintos para os vários dias da semana, é possível clicar no botão “+” para adicionar um novo campo com dias da semana e horas de abertura e fecho. Para a adição de um laboratório é possível indicar a quantidade e capacidade das bancadas tal como a sua disponibilidade. À semelhança dos horários. É também possível adicionar novos campos para bancadas diferentes.

Para adicionar um horário a um espaço ou uma bancada a um laboratório utilizam-se formulários similares, mas que apenas incluem os elementos necessários para realizar a ação desejada.

Adicionar Espaços

Tipo de Espaço

☐ Laboratório ☒ Sala

Localização

Condição

Horários

S T Q Q S S D

☐ ☐ ☐ ☐ ☐ ☐ ☐

Abertura Fecho

- +

CANCELAR ADICIONAR

Figura 27 - Adicionar Sala Armazenamento

Adicionar Espaços

Tipo de Espaço

☒ Laboratório ☐ Sala

Localização

Condição

Horários

S T Q Q S S D

☐ ☐ ☐ ☐ ☐ ☐ ☐

Abertura Fecho

- +

Bancadas

Quantidade Capacidade

Disponibilidade

- +

CANCELAR ADICIONAR

Figura 26- Adicionar Laboratório

Abertura

hh:mm

00 00

01 05

02 10

03 15

04 20

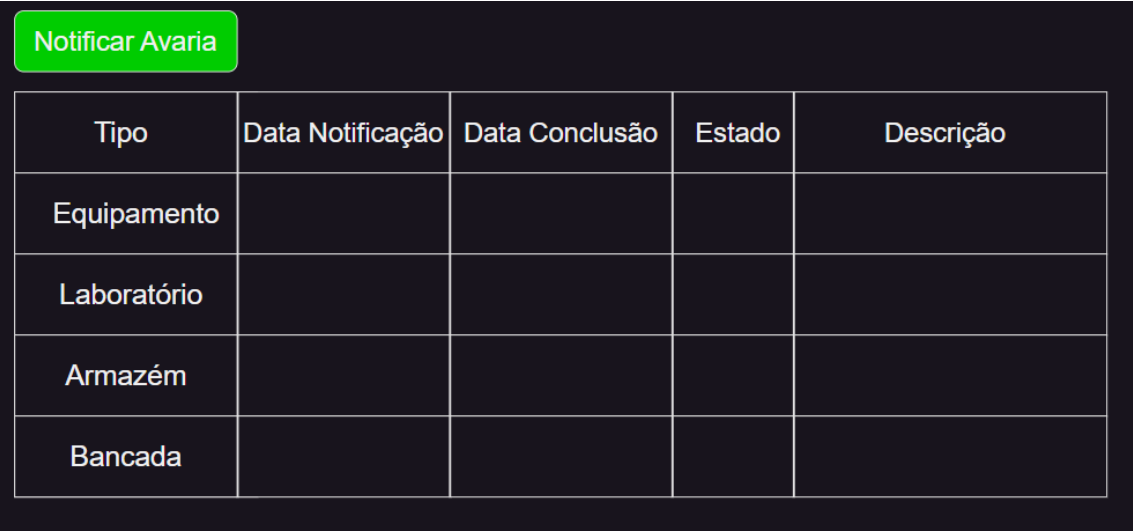
Figura 28 - Campo de Horário

Para adicionar um horário a um espaço ou uma bancada a um laboratório os formulários serão similares ao de adicionar um espaço, no entanto possuirá apenas os elementos necessários para

realizar a ação desejada.

6. Avarias

Conforme indicado na Figura 5, uma avaria é caracterizada pelo tipo de item aplicável, i.e. um equipamento, espaço ou bancada, por uma descrição, pelo seu estado atual e por datas de notificação e de conclusão. Na página de avarias representa-se todos estes dados usando uma tabela, conforme se mostra na Figura 29.



Tipo	Data Notificação	Data Conclusão	Estado	Descrição
Equipamento				
Laboratório				
Armazém				
Bancada				

Figura 29 - Visualização de Avarias

Para notificar uma avaria o utilizador tem de especificar a natureza do item (equipamento, laboratório, armazém ou bancada). Com isto surgirá no formulário os campos para selecionar o item com a anomalia como o número de bancada e designação de laboratório ou o número de equipamento e designação de equipamento. Por último deverá ainda descrever a anomalia observada. A Figura 30 exemplifica o formulário previamente explicado antes da seleção do tipo de item a notificar. Com isto, a tabela ficará preenchida com os valores anteriores sem a data de conclusão e com o estado. Quando o responsável por verificar a avaria alterar o seu estado para concluída, a data de conclusão será alterada para a data de modificação do estado.

Notificar Avaria

Item a notificar ▼

Descrição do Problema

Cancelar

Notificar

Figura 30 - Formulário de Notificação de uma Avaria