ISEL - Instituto Superior de Engenharia de Lisboa

i-on Teams

Afonso Machado
Martim Francisco

Orientador: Pedro Félix

Relatório do projeto realizado no âmbito de Projecto e Seminário
Licenciatura em Engenharia Informática e de Computadores

setembro de 2022

Instituto Superior de Engenharia de Lisboa

i-on Teams

47217 Afonso Araújo Machado

46912 Martim Rodrigues Francisco

Orientador: Pedro Miguel Henriques Santos Félix

Relatório do projeto realizado no âmbito de Projeto e Seminário
Licenciatura em Engenharia Informática e de Computadores

setembro de 2022

Resumo

O presente documento descreve o desenvolvimento do projeto **i-on Teams**.

Atualmente, algumas unidades curriculares do Instituto Superior de Engenharia de Lisboa (ISEL) utilizam a plataforma **GitHub Classroom**, disponibilizada pelo **GitHub**, para gestão de grupos de trabalho, em que, cada grupo tem o seu repositório para o desenvolvimento das tarefas propostas. A existência desta plataforma facilita algumas tarefas do professor, tais como: formação de grupos; criação de repositórios para cada grupo de uma turma. Contudo existem lacunas no que a ferramenta tem para oferecer, e que serão faladas mais a frente no presente documento.

No ano letivo de 2020/21 foi realizado um projeto denominado **i-on Codegarten**, também no âmbito da unidade curricular de Projeto e Seminário e sob a alçada da iniciativa i-on. Este tinha um propósito semelhante ao i-on Teams, porém, também foram identificadas algumas lacunas no mesmo, tais como armazenamento de informação sensível no servidor e impossibilidade do docente obter cópias dos trabalhos, estas serão aprofundadas no decorrer deste relatório.

O i-on Teams tem como principal objetivo atender as lacunas identificadas tanto no **GitHub Classroom**, como no **i-on Codegarten** e assim otimizar a gestão, por parte do docente, das suas unidades curriculares.

O **i-on Teams** é constituido por uma aplicação desktop, uma aplicação web, e um serviço. A aplicação desktop, destinada ao uso exclusivo por parte do docente, permite que o mesmo possa gerir a formação de grupos, criar trabalhos e definir a sua data de entrega e gerir as entregas dos mesmos, entre outras funcionalidades. Já a aplicação web, desta vez destinada aos estudantes, serve para que os mesmos possam aderir a grupos de trabalho, associar a sua identificação na instituição de ensino ao seu utilizador do **GitHub** e consultar informações relativas as suas turmas, grupos e trabalhos. Por fim, o serviço, designado de i-on Teams Service, serve para a gestão dos dados necessários ao funcionamento do sistema, assim como a sua disponibilização a partir de uma API Web.

Palavras-chave: Github; GitHub Classroom; i-on Teams; desktop app; Web app; Web API;

Abstract

This document serves as a way to describe the implementation and development of the **i-on Teams** project.

Some courses at the Lisbon Institute of Engineering (ISEL) use the platform **GitHub Classroom**, integrated within **GitHub**, to manage classroom student teams, where each team has it's own repository for assignment completion. The existence of this platform facilitates some of the teachers management tasks, such as creating teams repositories, however some deficiencies have been identified on what can be achieved with this tool, these deficiencies will be described in greater detail later in the present report.

A similar project, with the name **i-on Codegarten**, was developed in the 2020/21 school year, also under the i-on initiative. But with this project there also were deficiencies identified, such as storing sensitive information in the server and the inability for the teacher to download copies of the student's work. Again, these will be described in greater detail later in this report.

The main objective of the i-on Teams project is to rectify the deficiencies identified in both **GitHub Classroom** and in **i-on Codegarten**, and with that, optimize the management of the teacher's classrooms.

The **i-on Teams** project is comprised of three main components, a desktop application, a web application and a service. The desktop application is to be used exclusively by the teacher and provides a means to manage teams, assignments and delivery dates, among other functionalities. The web application is destined for students and will be used for them to create and join teams, to associate their GitHub username to their institutional identity and for them to consult information about their classrooms, teams and assignments. Finally, the service, designated **i-on Teams Service**, will be used to host and manage data necessary for the system's functionality and will expose a Web API to be used by it's clients.

Key-words: Github; GitHub Classroom; i-on Teams; desktop app; Web app; Web API;

Agradecimentos

Quero começar por agradecer ao meu colega e amigo Martim Francisco por desenvolver este projeto comigo, não teria o orgulho que tenho no que foi alcançado, sem o seu apoio e dedicação.

Quero também agradecer ao orientador do projeto o professor Pedro Félix, por apresentar-nos esta ideia, também pela sua constante disponibilização e apoio durante todo o projeto.

Por fim, quero agradecer a toda a minha família, amigos e à minha namorada por me acompanharem e apoiarem durante todo o percurso académico. O mérito é partilhado com eles, sem eles não teria alcançado o mesmo.

Obrigado.

setembro de 2022, Afonso Machado

Em primeiro lugar quero agradecer ao meu colega Afonso Machado por me ter acompanhado no realizar deste projeto, sem duvida que sem o apoio, a determinação e o companheirismo dele, o resultado final e o sentimento de missão cumprida não teriam sido os mesmos.

Quero também deixar uma nota de agradecimento ao nosso orientador Pedro Félix por nos ter apresentado esta ideia e acompanhado durante todo o processo, a sua ajuda e profundo conhecimento da área foram determinantes para que este projeto ficasse concluído com a qualidade que eu considero que ficou.

Como não poderia deixar de ser, quero também agradecer a toda a minha família e a minha namorada Rita Carneiro por me terem proporcionado a oportunidade de ingressar no ISEL e a força e apoio emocional necessários para concluir não só este projeto final, como todos os outros projetos e unidades curriculares no decorrer da Licenciatura que fica agora concluída.

Por fim queria deixar duas notas de agradecimento, uma à Maria Antunes, autora do logótipo do projeto, que se mostrou muito disponível para fazê-lo quando lhe propusemos a ideia, e a outra ao meu grupo de amigos que tive a oportunidade de conhecer no ISEL, e que foram determinantes na resolução de muitos problemas técnicos e bloqueios por vezes sentidos no desenvolvimento do projeto.

setembro de 2022, Martim Francisco

Índice

Lista de Figuras

Listings

Capítulo 1

Introdução

O presente capítulo introduz e contextualiza o projeto **i-on Teams**, descrevendo a sua motivação, as principais funcionalidades e características, e a estrutura do documento.

1.1 Contexto e Motivação

O projeto **i-on Teams** insere-se na iniciativa **i-on** [1] que consiste no desenvolvimento de sistemas e aplicações de apoio a atividades académicas no **ISEL** [2]. Um dos principais objetivos desta iniciativa é também o desenvolvimento e disponibilização *open-source* dos sistemas e aplicações por ela produzidos.

O **i-on Teams** pretende facilitar a gestão das unidades curriculares do docente, através da centralização da gestão de grupos e trabalhos práticos de uma ou várias turmas.

Atualmente no ISEL, é utilizado o GitHub Classroom [3] em muitas unidades curriculares, onde, na sua componente prática, é necessária a partilha de código com o docente e entre os elementos do grupo, caso de um grupo se trate. Cada aluno tem a sua conta particular, que pode ou não estar associada ao email do ISEL, com esta o aluno entra numa equipa e consequentemente tem acesso ao código desenvolvido. Apesar das vantagens que o aparecimento do GitHub Classroom tiveram, algumas lacunas são sentidas tanto por alunos como por docentes durante a sua utilização académica intensiva. As principais lacunas encontradas são descritas de seguida:

- Falta de associação entre o aluno e o seu nome de utilizador no GitHub [4];
- Dificuldade na verificação de entregas, mesmo realizando uma *tag* no repositório como representação da entrega o docente tem de, individualmente para cada grupo, confirmar a data e hora que esta foi feita;
- O acesso ao código dos vários grupos é complexo, pois sempre que o docente deseja consulta-lo necessita de fazer um pull ou navegar no website do GitHub individualmente para cada grupo;

- Falta de centralização de informação considerada relevante, como por exemplo, o enunciado do trabalho/tarefa proposta pelo docente e a data de entrega do mesmo.

No ano letivo de 2020/21, foi desenvolvido um projeto com uma motivação e propósito semelhante chamado **i-on Codegarten**. Contudo, neste também foram encontradas algumas lacunas na sua implementação, as principais sendo:

- Armazenamento de informação sensível no servidor, mais concretamente o token do GitHub do professor que é capaz de alterar as informações de todas as organizações, turmas, grupos, etc.;
- Impossibilidade do docente obter cópias dos trabalhos. O docente continua ter de individualmente obter o código do grupo.

Assim sendo, foi decidido tomar o curso de uma nova implementação que conseguisse colmatar as lacunas identificadas.

1.2 Características e Funcionalidades

O sistema projeto é constituído por duas aplicações, sendo um delas *desktop* e outra *web*, e um serviço chamado **i-on Teams Service**. A aplicação *desktop*, destinada ao uso exclusivo por parte do docente, é utilizada para consulta e gestão de turmas, grupos e trabalhos práticos, de entre outras funcionalidades. Quanto à aplicação *web* é apenas para consulta por parte dos alunos, e também para a criação ou inscrição num grupo de uma turma. Ambas têm dois mecanismos de autenticação, o próprio do i-on Teams, com objetivo de recolher informação pessoal do utilizador (exemplo: número de aluno) e verificar o utilizador via email. O outro mecanismo é a OAuth2.0, do GitHub, que tem como objetivo obter o nome de utilizador. O i-on Teams Service é responsável pelo armazenamento e disponibilização de dados, e por garantir alguma lógica de negócio. O armazenamento de dados recorre a uma base de dados relacional e a disponibilização dos mesmos através da exposição de uma API Web.

Tal como referido anteriormente o sistema usa o **GitHub** para a autenticação de utilizadores, para a criação de repositórios, e para a gestão dos mesmos, através da sua Web API. Ao autenticar um utilizador ou docente será feita uma associação entre número, atribuído pela instituição, e o nome de utilizador do GitHub.

Através da aplicação desktop o docente tem acesso as seguintes funcionalidades:

- Listar todas as organizações, turmas, equipas/grupos, enunciados e entregas;
- Criar e/ou atualizar de organizações, turmas, enunciados, ou entregas;
- Aceitar ou negar um pedido de criação de uma nova equipa equipa;
- Gestão das entregas de todos os grupos pertencentes à turma que está a gerir;

- Possibilidade de criar notas privadas por equipa;
- Acesso a uma lista de alunos de uma turma ou de uma equipa;
- Acesso aos link dos repositórios do GitHub, de cada turma e de cada equipa:
- Possibilidade de fazer um *pull* dentro de uma turma, corresponde a fazer *pull* de todos os repositórios localmente.

Quanto à aplicação web, dedicada a uso por parte dos alunos, é possível:

- Listar todas as turmas, equipas, repositórios, enunciados e entregas;
- Listar o(s) professore(s) responsáveis pela turma;
- Acesso aos link dos repositórios do GitHub, de cada turma e de cada equipa.

A aplicação web permitirá também o download do instalador da última versão disponível da aplicação desktop.

Para todas as funcionalidades mencionadas anteriormente o professor, ou aluno, necessitam de estar inscritos na organização a que estas pertencem. As entregas são identificadas pela *tag* existente no GitHub, e a verificação é feita através da data da publicação *tag*.

1.3 Descrição da Estrutura do Documento

No capitulo 2 são descritos todos os aspetos da arquitetura do sistema.

Os capítulos 3 e 4 focam-se na implementação específica do sistema com destaque para as metodologias e tecnologias usadas, e para as decisões tomadas. O capítulo 3 refere-se especificamente à implementação do serviço **i-on Teams Service**, enquanto que o capítulo 4 foca-se na implementação das aplicações *web* e *desktop*.

Por fim, no capítulo 5 serão abordadas as conclusões sobre a implementação do projeto, assim como os desafios encontrados no decorrer da mesma. Este último capítulo inclui também aspetos da evolução futura do projeto.

Capítulo 2

Arquitetura

O presente capítulo descreve a arquitetura desenhada, começando por descrever os três principais componentes que a constituem. Na figura 2.1 é ilustrada a arquitetura do sistema e a interação entre os vários componentes.

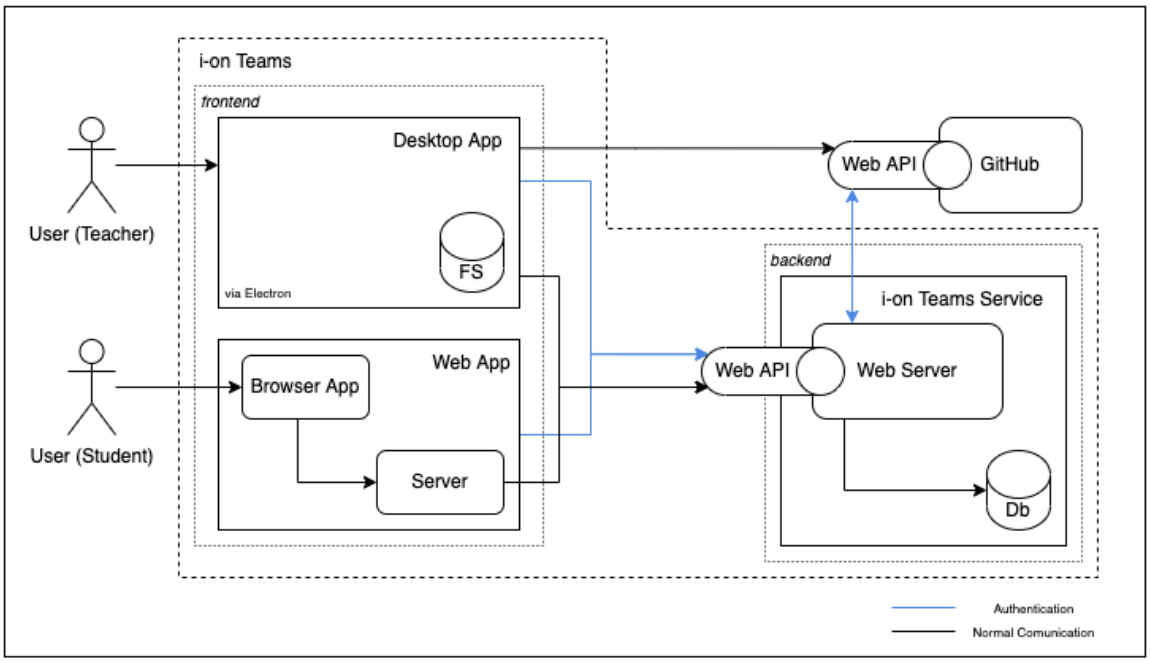

Figura 2.1: Arquitetura do sistema

Os elementos de *frontend*, isto é, as aplicações web ou desktop com as quais o utilizador pode interagir, identificadas na figura como **Desktop App** e **Web App**, comunicam diretamente com a Web API disponibilizada pelo serviço e, no caso da aplicação desktop, também com a API disponibilizada pelo GitHub.

O **i-on Teams Service** é responsável por implementar lógica de negócio, isto é, o fazer cumprir requisitos essenciais para o correto funcionamento do sistema, armazenar dados e disponibiliza-los aos seus clientes. Para isso este expõe uma Web API que disponibiliza

um conjunto de funcionalidades suportadas pelos dados armazenados numa base de dados relacional.

A API disponibilizada pelo **GitHub** é utilizada tanto para a autenticação, através de uma OAuth App[5], como para a correspondência entre recursos armazenados no serviço (turmas, repositórios, etc) e os recursos equivalentes do GitHub.

2.1 i-on Teams Service

O i-on Teams Service expõe uma Web API comum a todos os utilizadores, implementada com recurso à *framework* Spring Boot [6] através linguagem Kotlin [7], e contém a base de dados. Este irá ser hospedado na plataforma **Heroku**[8]. A escolha desta plataforma em detrimento de outras disponíveis para o mesmo efeito vem do conhecimento prévio e experiência que já tinha sido adquirida sobre a mesma.

Como Sistema de Gestão de Base de Dados (SGBD) foi escolhida uma base de dados relacional desenvolvida e implementada através de PostgreSQL [9], escolha feita com base no conhecimento de sistemas de informação adquirido anteriormente e, tendo também em conta, que o mesmo será hospedado na plataforma **Amazon RDS** [10], disponibilizada pela **AWS** (Amazon Web Services) [11], e que suporta totalmente SGBDs em PostgreSQL.

2.1.1 Estrutura da API

A API foi toda desenvolvida com recurso a *hypermedia*, permitindo a navegação entre recursos e o complementar da informação do recurso com outras informações utilizadas pelos clientes para que o mesmo seja apresentado na maneira mais apropriada. No que toca ao *media type* e ao corpo das respostas em caso de sucesso é diferente em cada tipo. Assim sendo, temos para o pedido

- GET: application/siren+json [12];
- POST e PUT: application/json [13];
- DELETE: sem corpo de resposta.

Quanto ás respostas em caso de erro, o *media type* da resposta é application/problem+json [14].

2.2 Aplicações

As duas aplicações desenvolvidas, *desktop* e *web*, têm como finalidade fornecer uma interface gráfica para interação com o sistema, que deverá ser utilizada por professores, no caso da aplicação *desktop*, e por alunos, na aplicação *web*.

Ambas as aplicaçãos fazem uso da *framework* de CSS **Semantic UI** [15], mais concretamente a sua implementação para React [16]. Utilizam também o **Webpack** [17] como bundler, que para além de disponibilizar todas as dependencias necessárias para a execução dos scripts, transforma o código num ficheiro muito pouco *human readable* mas que traz vantagens de um ponto de vista da eficiência pois reduz o tamanho do script executado pelo *browser*. Foi também utilizado o compilador do Typescript [18] para transformar o código Typecript em código Javascript que o *browser* consegue compreender.

2.2.1 Aplicação Web

A aplicação web é uma *Single Page Applcation (SPA)*, o que significa que a aplicação apenas tem de ser carregada uma vez e todo o seu conteúdo é obtido de forma dinâmica, através do código JavaScript que corre no navegador do utilizador como *script* da página. A aplicação irá realizar pedidos ao serviço à medida das suas necessidades por forma a disponibilizar ao utilizador a informação que este requer. O desenvolvimento da aplicação web foi realizado com recurso à linguagem de programação TypeScript e à biblioteca ReactJS [19]. Na figura 2.2 está representada a arquitetura da aplicação web, assim como os seus componentes.

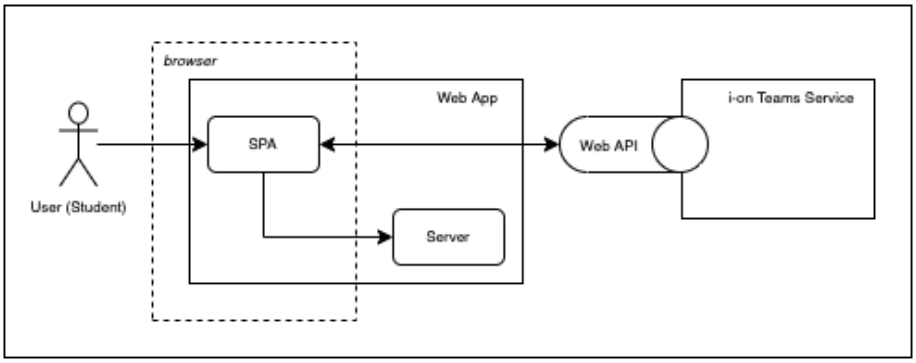

Figura 2.2: Arquitetura e componentes da aplicação web

A aplicação é composta por dois principais componentes, uma componente cliente e uma componente servidora.

A componente servidora é responsável por disponibilizar a componente cliente ao browser do utilizador, para que este a possa executar, assim como disponibilizar ficheiros estáticos que sejam necessários para o correto funcionamento da mesma.

Já a componente cliente tem como responsabilidade a apresentação de uma interface gráfica com a qual o utilizador pode interagir, assim como a comunicação direta com o serviço para obtenção dos dados a apresentar. Sendo esta componente uma SPA, esta apenas tem de ser carregada uma vez e todas as suas representações serão obtidas de forma dinâmica, consoante os *inputs* do utilizador.

2.2.2 Aplicação Desktop

A aplicação *desktop*, à semelhança da aplicação *web*, também é, na sua essência, uma SPA. Porém, dadas as limitações de ser uma aplicação *desktop*, e para que fosse possível a utilização de tecnologias encontradas na *web*, com as quais já existia mais conhecimento prévio, esta possui alguns importantes detalhes de arquitetura que a distinguem da aplicação *web*.

Esta aplicação foi então desenvolvida com recurso à *framework* **Electron** [20], que permite utilizar tecnologia *web* para desenvolver aplicações *desktop*, multiplataforma. Este último ponto é particularmente importante, pois permitiu a construção de uma única aplicação que depois seria compilada para os diferentes sistemas operativos, não sendo então necessário construir uma aplicação nativa a cada sistema operativo (ex. Windows, MacOS, Linux, etc.).

A arquitetura da aplicação e os seus componentes estão representados na figura 2.3.

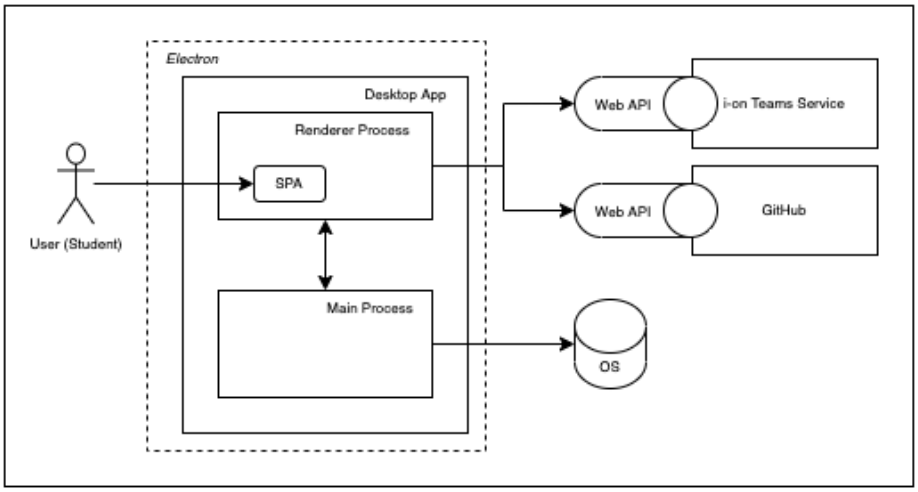

Figura 2.3: Arquitetura e componentes da aplicação desktop

Na aplicação *desktop* existem três principais componentes, um *main process*, um *renderer process* e uma SPA.

O Electron herda a sua *multi-process architecture* [21] do Chromium [22], presente também em browsers modernos populares (ex. Google Chrome, Microsoft Edge, etc.). Este tipo de modelo de processos resulta num nível de isolamento entre os conteúdos apresentados ao utilizador, executados pelo *renderer process*, e as funções essenciais da aplicação, executadas pelo *main process*.

No caso específico do Electron, e consequentemente da aplicação *desktop*, o *main process* é utilizado para executar a aplicação em si, e por criar uma ponte entre a mesma e o sistema operativo, através das diversas APIs que os Sistemas Operativos modernos disponibilizam.

O *renderer process* é o responsável por executar a janela da aplicação e, por sua vez, executar a SPA.

Por fim, a SPA tem a responsabilidade de representar graficamente os dados a serem apresentados ao utilizador, assim como a ligação às web APIs do GitHub e do serviço para obtenção desses mesmos dados.

2.3 Autenticação

A autenticação dos utilizadores é realizada através do serviço e de uma **OAuth App**. O serviço é responsável por iniciar os fluxos de autenticação, tanto para login como para registo de novos utilizadores, e pelo armazenamento e atribuição de utilizadores e sessões. Já o GitHub, através do protocolo OAuth 2.0 [23] é responsável por fornecer a identidade dos utilizadores.

Em ambos os casos de autenticação o serviço será responsável por indicar à aplicação, no caso da **web app**, ou ao browser, no caso da **desktop app**, o *endpoint* para comunicar com o **Authorization endpoint** do servidor do GitHub. Findo o processo de autenticação do github este gera um **code** que será enviado para o serviço através do *endpoint* indicado à OAuth App para *callback*. No caso da aplicação *desktop*, dada a sensibilidade do *token* gerado, o serviço nunca o armazena, assim sendo o *token* só é gerado a pedido da própria aplicação ao fazer um pedido para um *endpoint* específico para o efeito.

Em ambos os casos o processo de autenticação é dado como concluído depois da obtenção do *token*, que no caso dos docentes terá autorização para fazer pedidos de escrita e de leitura para os repositórios e organizações onde o docente se encontra, e no caso do aluno apenas poderá fazer pedidos de leitura.

2.3.1 Aplicação Desktop

Nas figuras 2.4 e 2.5 estão apresentados os esquemas de autenticação da aplicação desktop, respetivamente, *register* e *login*.

A figura ilustra a importância do browser na autenticação da aplicação desktop. Este é utilizado como intermediário entre a aplicação e os serviços com os quais é preciso comunicar para que a autenticação seja bem sucedida. Isto deve-se ao facto de ser mais conveniente para o utilizar realizar este fluxo num browser onde pode ser armazenada informação relativa ao GitHub para que o utilizador não precise de iniciar sessão no GitHub a cada vez que pretender iniciar sessão no i-on Teams. No fim do processo de autenticação o browser entra em contacto com a aplicação para trazê-la a foco e para enviar as informações necessárias à correta conclusão do processo.

Após concluído o fluxo de autenticação no browser cabe à aplicação desktop entrar em contacto com o serviço para que lhe seja atribuída a informação que permite autenticar os pedidos subsequentes, que é armazenada num *cookie*.

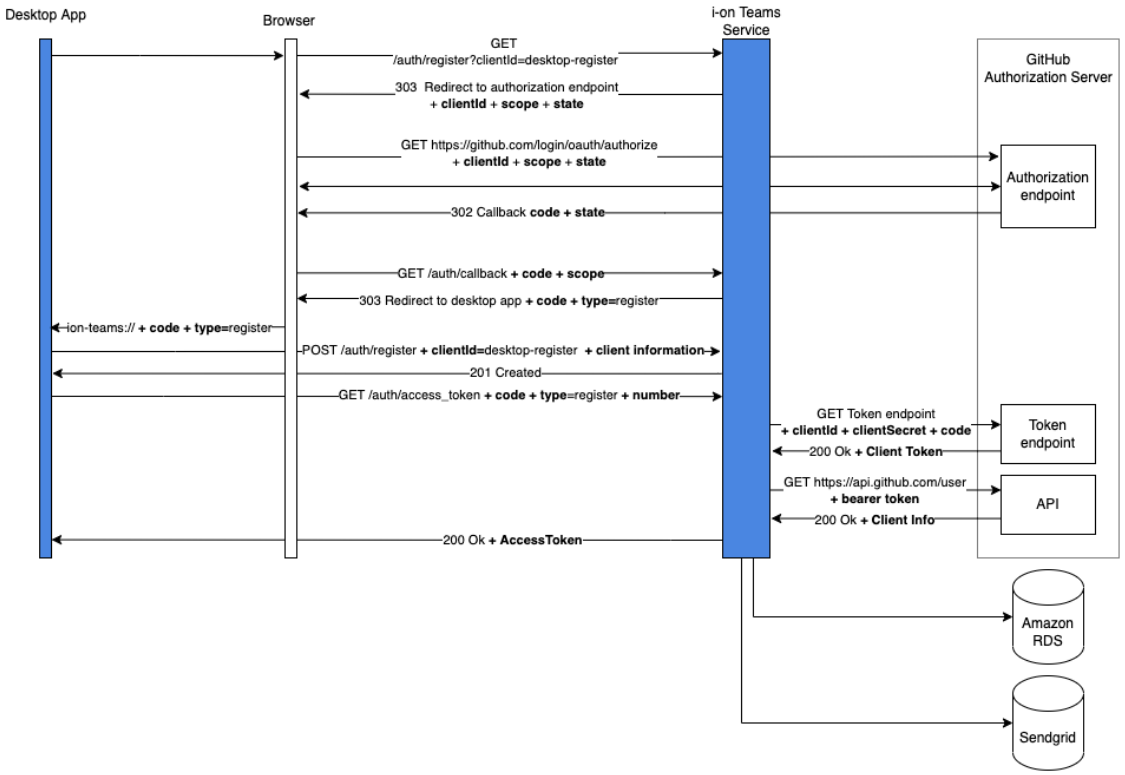

Figura 2.4: Arquitetura Autenticação da Aplicação Desktop - *Register*

2.3.2 Aplicação Web

Nas figuras 2.6 e 2.7 estão apresentados os esquemas de autenticação da aplicação web, respetivamente, *register* e *login*.

Na autenticação da aplicação *web*, o aspeto mais importante a salientar é o facto do serviço não poder enviar uma resposta de redirecionamento direto. Isto deve-se ao facto de que se fosse enviada uma resposta de redirecionamento, quem iria processar o redirecionamento seria a **web app** diretamente, por via da função *fetch*, o que resultaria no bloqueio do pedido por violar as políticas de CORS [24] do *browser*. Para contornar esta situação foi necessário o serviço responder com status 200 ou 201 e enviar no *payload* da resposta o *endpoint* para onde deve ser feito o pedido de autenticação. Ao receber essa resposta a aplicação utiliza o url recebido e pede ao browser para fazer esse pedido diretamente, cumprindo assim todos os requisitos da política de CORS.

2.4 Organização do Projeto

A base de código de todo o projeto, assim como toda a sua documentação encontra-se num repositório [25] público do GitHub, fazendo jus a uma das máximas da iniciativa i-on, sobre a disponibilização *open-source*, ou seja, pública e acessível por todos.

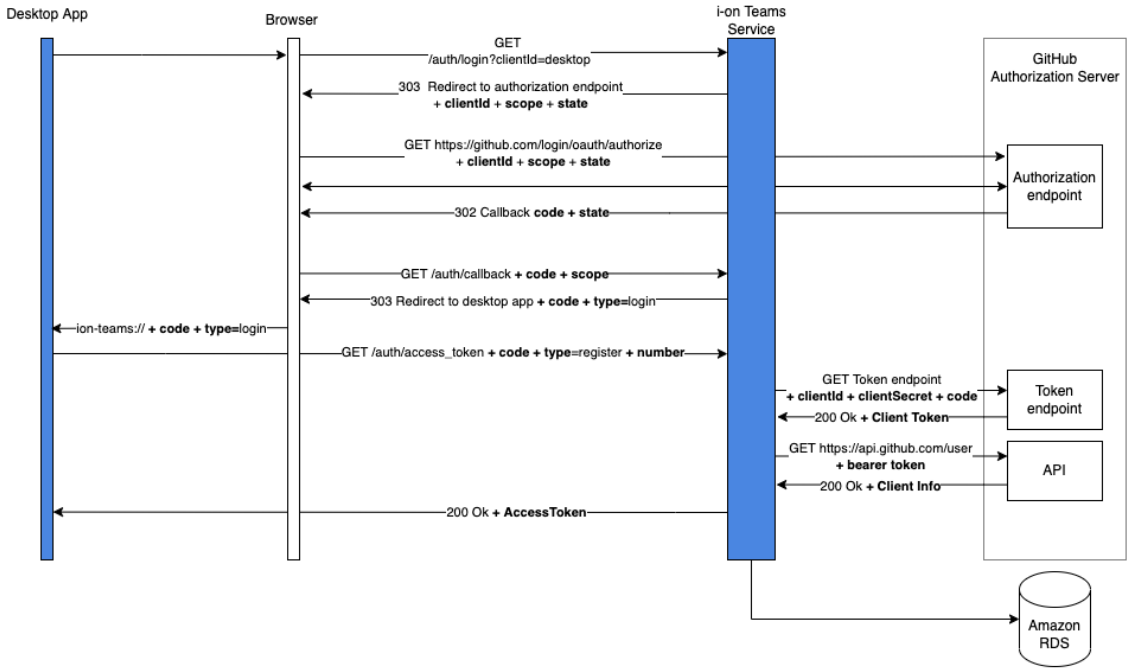

Figura 2.5: Arquitetura Autenticação da Aplicação Desktop - *Login*

Todo o código encontra-se na pasta **code** e toda a documentação produzida para o mesmo na pasta **docs** do referido repositório, para que exista uma divisão clara entre o código e a documentação.

O repositório encontra-se organizado da seguinte forma:

- Diretoria **docs**
  - Pasta **2022** - Documentação produzida nas diversas entregas de Projeto e Seminário.
- Diretoria **code**
  - Pasta **js** - Código para ambas as aplicações web e desktop;
  - Pasta **jvm** - Código referente à Web API exposta pelo serviço;
  - Pasta **sql** - Código SQL referente à base de dados do projeto.

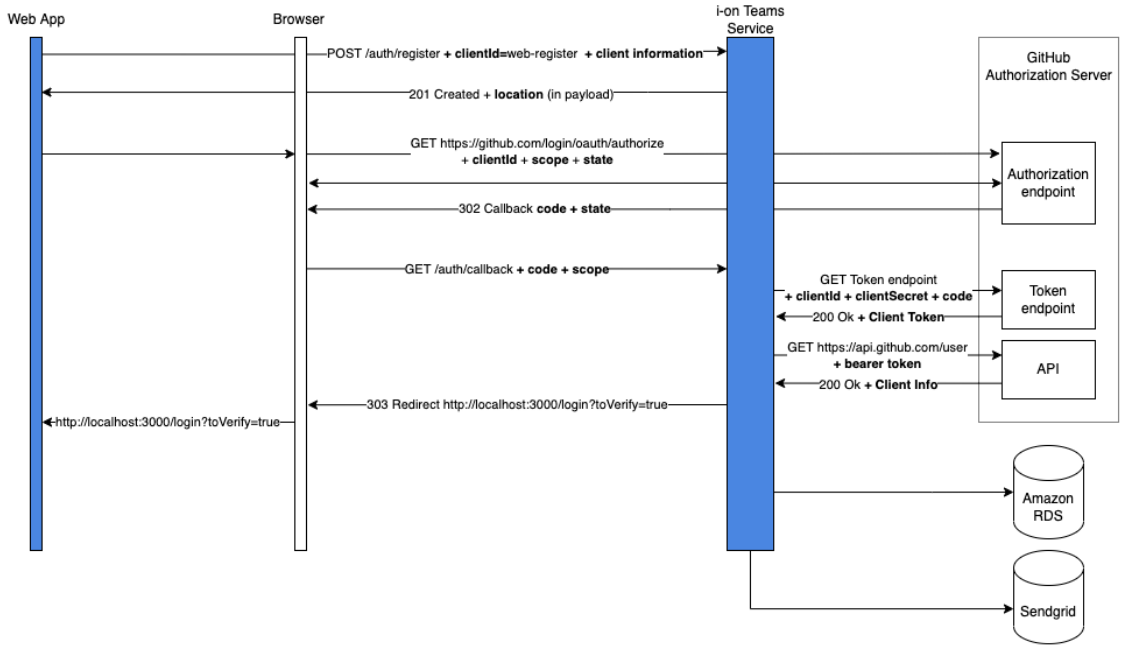

Figura 2.6: Arquitetura Autenticação da Aplicação Web - *Register*

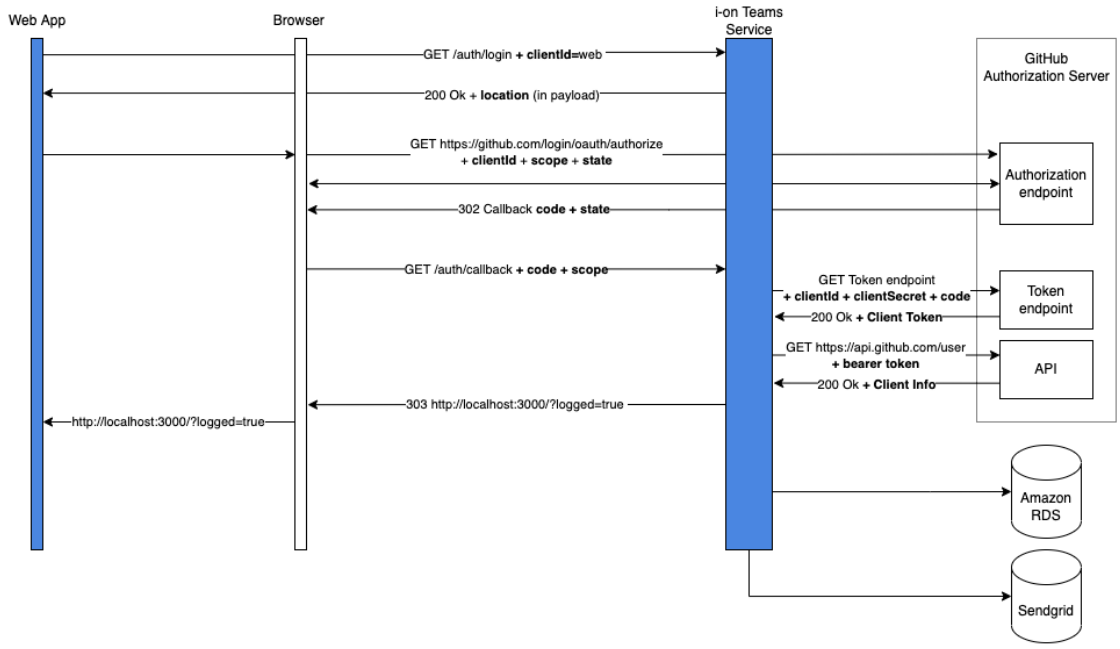

Figura 2.7: Arquitetura Autenticação da Aplicação Web - *Login*

Capítulo 3

Aspetos de Implementação do i-on Teams Service

O serviço i-on Teams Service é responsável pelo armazenamento de dados e consequente disponibilização dos mesmos aos diversos clientes do i-on Teams. O serviço serve também como ponto intermédio para a autenticação de utilizadores entre os clientes do i-on Teams e o GitHub.

Tendo em conta os traços gerais dos principais requisitos do nosso serviço, identificados anteriormente, e com particular destaque para o armazenamento e disponibilização de dados, foi decidido que os dados tratados pelo mesmo seriam armazenados com recurso a uma base de dados relacional e que estes seriam expostos aos clientes através de uma API Web.

Assim sendo, este capítulo trata em mais detalhe toda a implementação realizada no âmbito da disponibilização do i-on Teams Service, com destaque para os tópicos referidos anteriormente, mas não se cingindo apenas a estes e abordando também outros tópicos relevantes como a sua estrutura de implementação ou os testes realizados para validar esta mesma implementação.

3.1 Modelo de Dados

Nesta secção será apresentada a solução encontrada para a gestão e armazenamento dos dados necessários do i-on Teams Service. Visto que o nosso serviço depende largamente das relações entre dados armazenados foi tomada a decisão de usar uma base de dados relacional.

O ponto de partida para a implementação de um modelo de dados relacional organizado, eficiente e consistente é a definição das suas entidades e, por consequência, também das relações entre as mesmas. Na figura 3.1 apresenta-se o diagrama entidade-associação utilizado na implementação do modelo de dados em questão.

Neste modelo serão armazenadas todas as informações necessárias ao funcionamento do sistema, algumas entidades estão disponíveis para ambos os tipos de utilizadores, docentes e alunos, como turmas (*classrooms*), grupos (*teams*), trabalhos (*assignments*), entregas (*deliveries*) e repositórios (*repos*), enquanto que outras entidades apenas estão disponíveis

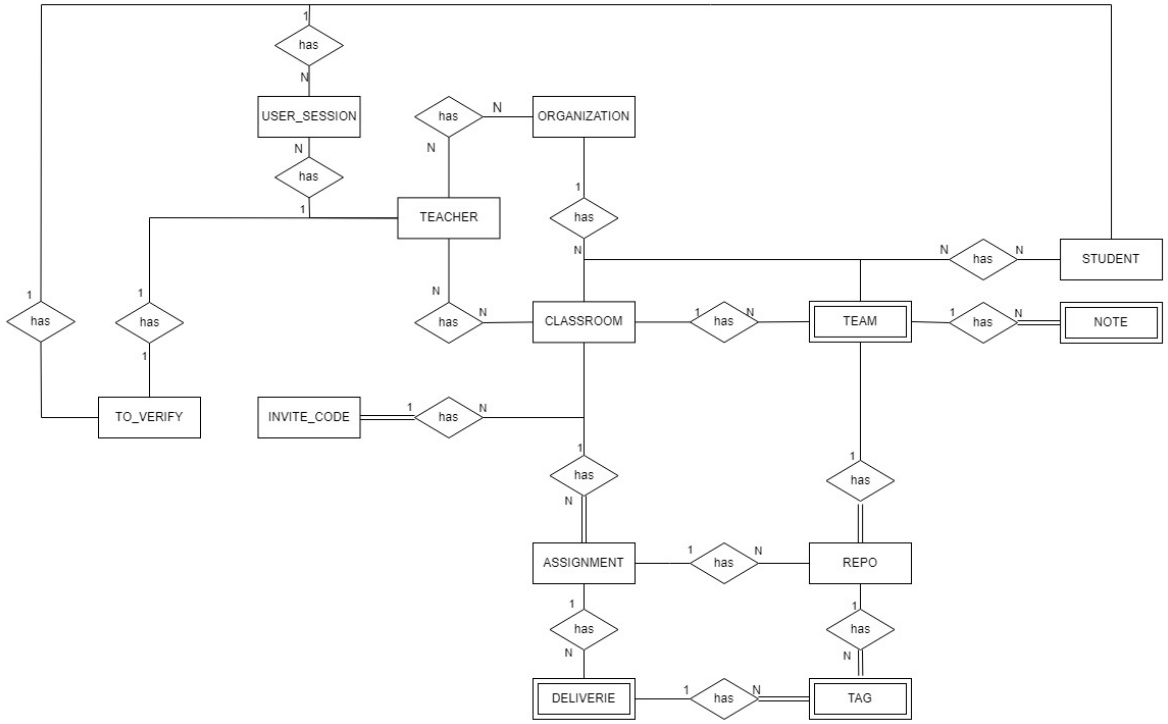

Figura 3.1: Diagrama Entidade-Associação do Modelo de Dados

para consulta e manipulação por parte do docente, como alunos (*students*), notas (*notes*), organizações (*organizations*) e *invite codes*.

Por fim, no modelo de dados foi implementado *soft deletes* que ao invés de apagar permanentemente determinado recurso, marca-o como apagado mantendo-o na base de dados até que seja conveniente a sua remoção total. O modelo de dados utiliza *triggers* associados a funções que permitem alguma lógica de negócio, como por exemplo, o limite de elementos por equipa e o limite de equipas por *classroom*, e também permitem um efeito *cascade delete* aplicado no contexto dos *soft deletes*. As vistas também foram utilizadas para ajudar a reunir informação de múltiplas tabelas e informação que não esteja eliminada.

3.2 Modelação da API

A *Web* API garante que os clientes conseguem aceder, adicionar ou alterar informação de um recurso, efetuando pedidos ao *endpoint* pretendido. Na figura 3.2 esta representada a arquitetura do serviço e, por sua vez, os elementos da *Web* API.

O acesso a dados é feito através da API declarativa da biblioteca **JDBI** [26]. Assim sendo, o serviço está dividido em três principais componentes, os *Controllers* que expõem os *endpoints* da API, os *Services* que acedem a dados e implementam a lógica de negócio dos recursos, e os *Data Access Objects* ou *DAOs* que contratualizam as funções de acesso a dados utilizadas pelos *Services*.

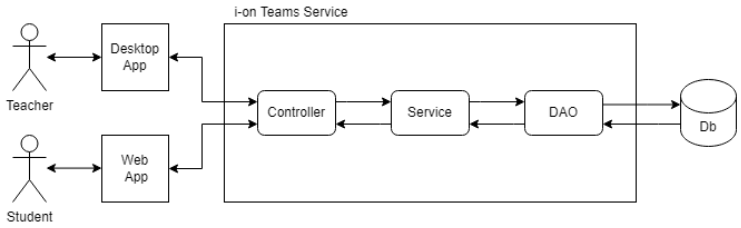

Figura 3.2: Arquitetura do i-on Teams Service

Respostas da API

As respostas de sucesso da API do **i-on Teams Service** recorrem a *hypermedia*. O tipo de resposta escolhido para estas foi *application/siren+json* visto que foi considerado o mais adequado por, de entre outras vantagens, permitir que as respostas levassem um conjunto de *actions* representativas de novas ações que podem ser realizadas sobre o recurso em questão, como novas inserções e/ou atualizações de recursos já existentes.

Para que pudéssemos converter os recursos, do modelo usado internamente pelo serviço, para *siren* foi necessária a implementação de uma *domain specific language (DSL)* que define este tipo de resposta. De seguida, foi construido um conversor específico para cada recurso, que a partir dos dados obtidos, através da base de dados, constrói o corpo de resposta inteiro em *siren*. Segue-se um exemplo de uma resposta *siren*.

Listing 3.1: Exemplo *Siren Collection* correspondente a organizações.

```
{
    "class": [
        "collection",
        "organization"
    ],
    "properties": {
        "pageIndex": 0,
        "pageSize": 0
    },
    "entities": [],
    "actions": [
        {
            "name": "create-organization",
            "title": "Create Organization",
            "method": "POST",
            "href": "/api/orgs",
            "type": "application/json",
            "fields": [
                {
                    "name": "name",
                    "type": "text"
                },
                {
```

```
                    "name": "description",
                    "type": "text"
                }
            ]
        }
    ],
    "links": [
        {
            "rel": "self",
            "href": "/api/orgs"
        },
        {
            "rel": "home",
            "href": "/api/home"
        },
        {
            "rel": "logout",
            "href": "/auth/logout"
        }
    ]
}
```

Não foram encontrados benefícios suficientes para que respostas a pedidos *POST* e *PUT* recorressem também a *hypermedia*, sendo assim as respostas vindas da criação ou atualização de recursos é to tipo *application/json*.

A produção das respostas de erro tem duas implementações. Os erros que necessitam de informação adicional, além da oferecida pela norma, são enviados com um corpo de resposta que usa o *mediatype application/problem+json* que permite complementar o *status code* de erro. Já os erros cuja informação definida na norma é suficiente para a descrição do mesmo, não sofrem qualquer tipo de alteração.

Listing 3.2: Corpo de resposta de uma mensagem de erro em *application/problem+json*.

```
{
    "type": "https://github.com/i-on-project/teams/blob/main/
    docs/api/problems/unauthorized.md",
    "title": "Unauthorized",
    "status": 403,
    "detail": "The user is not authenticated."
}
```

Para que, a API pudesse enviar estes erros mais completos foi necessária a definição de novas exceções específicas ao nosso serviço que, quando lançadas, são tratadas com recurso a um *Controller Advice* [27] do *Spring* que formula a resposta de erro a partir da informação disponível na exceção.

3.3 Autenticação

O i-on Teams Service é o componente do sistema com mais responsabilidade no processo de autenticação. Este é responsável por gerir todo o processo de autenticação, incluindo o começo e término da autenticação do utilizador no GitHub, envio do email de verificação para novos utilizadores e a criação e remoção de sessões de utilizadores armazenando-as através de *cookies*.

3.3.1 Recolha de informação do utilizador

O utilizador, independentemente da aplicação cliente, terá de inserir dados pessoais antes de desencadear o processo de autenticação. O armazenamento de informação é feito nas tabelas *teacher* e *student*, respetivamente docente e aluno, estas estão evidenciadas na figura 3.3. A única informação que não é inserida pelos os utilizador, em ambos os tipos de utilizador, antes de iniciar o processo de autenticação é o *githubUsername*. O nome do utilizador do Github é obtido ao fazer um pedido ao endpoint de informação do utilizador da API do Github com o *access token* do utilizador obtido na autenticação com o Github.

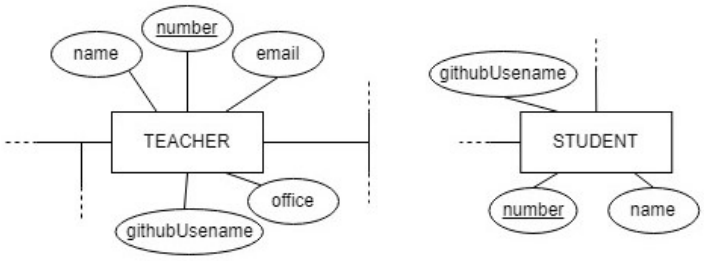

Figura 3.3: Armazenamento da informação do utilizador.

3.3.2 Email de verificação

Quanto à verificação via email, usada no processo de autenticação, é concretizada usando o **Twilio SendGrid Email API** [28]. De forma a, usar esta API é necessário uma *API key* que é obtida registando no site da mesma. O i-on Teams Service utiliza *Twilio SendGrid API wrapper* para fazer o pedido, este consiste num método POST e contém no *body* a *API key*, o remetente, o destinatário e o conteúdo do email. O email do utilizador é inserido no início do processo de autenticação pelo mesmo, no caso do professor, ou no caso do estudante é utilizado o email institucional com o número de aluno introduzido também no início do processo. O i-on Teams Service envia o email em *endpoints* diferente dependendo da aplicação onde originou o pedido de aunteticação. Logo, para a:

- Desktop-app - usa o *endpoint* do *access token*;
- Web-app - usa o *endpoint* do *callback*;

Sempre que o utilizador tenta iniciar a sessão sem verificar o email é enviado um novo email, nos *endpoints* mencionados anteriormente.

3.3.3 Processo de Autenticação

Na autenticação de utilizadores existem duas opções, registo de novos utilizadores e *login* para utilizadores já registados. O processo de autenticação é iniciado quando um utilizador assim o deseje, através de uma das aplicações disponibilizadas pelo sistema. O serviço recebe um pedido para *login* ou registo e responde encaminhando o utilizador para a autenticação do GitHub, através da **GitHub OAuth App** criada em nome do sistema i-on Teams. Ao reencaminhar o utilizador para o GitHub o serviço cria duas *cookies* que representam o seu *clientId* e um *state*. O *clientId* é enviado na *query string* inicial ao serviço e serve para identificar se o pedido corresponde a um *login* ou a um registo, e se é oriundo da aplicação desktop ou web. O *state* por outro lado é utilizado para verificar se existiu algum tipo de ataque no processo de autenticação do GitHub, sendo que o valor é enviado para o GitHub no início do processo e retornado no fim, se o valor retornado for igual ao valor armazenado na *cookie* garantindo que a resposta é iniciado pela aplicação no contexto do *browser* com o *cookie*.

O GitHub, depois de concluir com sucesso o seu processo de autenticação, redireciona o utilizador para o *endpoint* de *callback* do serviço. Ao receber o pedido, o *callback* toma ações e responde consoante o *clientId* recebido:

- *desktop* e *desktop-register*: Em ambos os casos de autenticação na **Desktop-app** o resto do fluxo de autenticação deve ser feito pela mesma, assim sendo, o *callback* limita-se a redirecionar o *browser* para a aplicação, enviando também o *code* recebido do GitHub e um *type* que representa se o fluxo é de *sign up* ou de *login*.

- *web*: No caso de *login* na **Web-app**, o *callback* verifica se o utilizador existe no serviço e se o mesmo já confirmou a sua identidade através do email, caso não tenha sido confirmado, um novo email de confirmação é enviado. Por fim, é criado um *cookie* que representa a informação necessária para a autenticação dos pedidos feitos à API a partir da **Web-app**.

- *web-register*: Se o processo de autenticação for de registo o *callback* é responsável por enviar um email de confirmação ao utilizador, e associar o seu nome de utilizador do GitHub ao utilizador armazenado no serviço.

A autenticação da **Web-app**, sendo mais simples, termina no *callback*. O mesmo não acontece na **Desktop-app** visto que esta ainda necessita de fazer mais um pedido para ver o seu processo de autenticação concluído. Este pedido é realizado para que o serviço possa pedir o *Access Token* do docente e envia-lo de imediato para a aplicação desktop. O *endpoint*

utilizado para estes pedidos é o *getDesktopAccessToken* e visto que representa também o fim da autenticação da **Desktop-app**, executa algumas ações que no caso da **Web-app** tinham sido realizadas no *callback*, nomeadamente a criação de *cookies*, a verificação do utilizador e a associação do nome de utilizador do GitHub ao utilizador do serviço

3.3.4 Cookies

O *cookie* utilizado para autenticação de pedidos realizados à API é chamado de *session*. A sua função é armazenar um UUID gerado no momento da sua criação e armazenado também na base de dados do serviço, associando-o ao utilizador na tabela *user session*, figura 3.4.

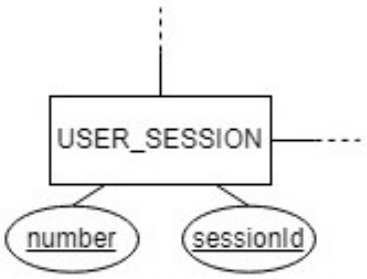

Figura 3.4: Associação do *sessionId* ao utilizador

O *cookie* com informação de sessão criado tem os seguintes atributos:

- httpOnly: true;
- secure: true;
- sameSite: none

A autenticação de pedidos é feita através de um *Interceptor* [29], implementado em *Spring*, ou seja, um pedido será intercetado à chegada antes de ser tratado e é verificado se o *cookie session* existe e se o mesmo representa um utilizador existente na base de dados. Caso a verificação falhe o pedido não é tratado e é rejeitado enviando o código de erro **401 Unauthorized**.

Na aplicação desktop existe também um segundo *cookie* chamado de *accessToken* utilizado para armazenar o *access token* do docente. A sua utilização será explicada com mais detalhe na secção sobre a implementação da **Desktop-app** mais a frente no presente documento. O *cookie* do *access token* tem os seguintes atributos:

- httpOnly: true;
- secure: true;
- sameSite: none;

3.4 Estrutura do Código

A descrição do conteúdo referente ao Serviço de cada pasta e *package* encontra-se de seguida.

- Diretoria **code**
  - Pasta **jvm** - código da Web API (*Spring boot*), que inclui também os testes à mesma. Esta, por sua vez foi organizada da seguinte maneira:
    - Uma pasta por recurso - Cada recurso disponível através da Web API tem a sua pasta onde constam o seu controller, service, DAO, modelos e o seu conversor para siren.
    - Pasta **common** - Esta foi usada para implementar elementos comuns a toda a API como definição de exceções, controladores de erros para tratamento de exceções e definição da *DSL* utilizada para as respostas em siren.
  - Pasta **sql** - código SQL da base de dados, capaz de construir todo o modelo de dados.
- Diretoria **docs**
  - Pasta **API** - Documentação API
    - Pasta **problems** - Todos os erros implementados na API;
    - Pasta **resources** - Documentação para todos os recursos acessíveis através da API, em que para cada um é indicado o vocabulário existente (domain specific ou standard), especificação dos *endpoints* e a descrição do tipo de erros gerados.
  - Pasta **auth** - Documentos referentes à autenticação;
  - Pasta **AWS** - Anotações acerca do funcionamento da hospedagem na AWS;
  - Pasta **data_model** - Documentos referentes ao modelo de dados.

3.5 Testes

Para que possa ser assegurado o correto funcionamento de todo o sistema quando este já estiver em produção, foram realizados testes para confirmar a correta implementação do mesmo.

De forma a que todos os recursos e funcionalidades expostas pela API fossem testadas, foram realizados testes de integração à mesma. Estes permitem testar o funcionamento conjunto de vários componentes, ao contrário de testes unitários que se caracterização por ser testes a funções ou componentes isoladamente. Assim foi-nos possível testar o funcionamento da API antes mesmo de começar a implementar os clientes que viriam a usufruir dela.

Estes testes são realizados através de um contexto Spring diferente do utilizado em produção denominado de MockMVC [30], que permite a realização de pedidos à API que são injetados diretamente na pipeline do Spring, sem recurso a pedidos HTTP. Sobre as respostas aos pedidos de teste são realizadas as asserções necessárias para a sua validação.

Para que os testes possam também ser totalmente independentes da versão final de produção, foi utilizada uma base de dados exclusiva para teste, completa com *mock data* para que os pedidos não só viessem completos com um corpo de resposta adequado, como assim seria possível prever o seu conteúdo e realizar asserções sobre o mesmo.

Os testes realizados caracterizam-se por testar individualmente todos os *mappers* dos controladores de cada um dos recursos acessíveis através da API, através da elaboração de pedidos pelo **MockMVC** e da asserção da coerência da resposta relativamente aos dados falsos presentes na base de dados de testes.

Na 3.5 pode verificar-se uma comparação entre o contexto do **Spring** em testes e em produção.

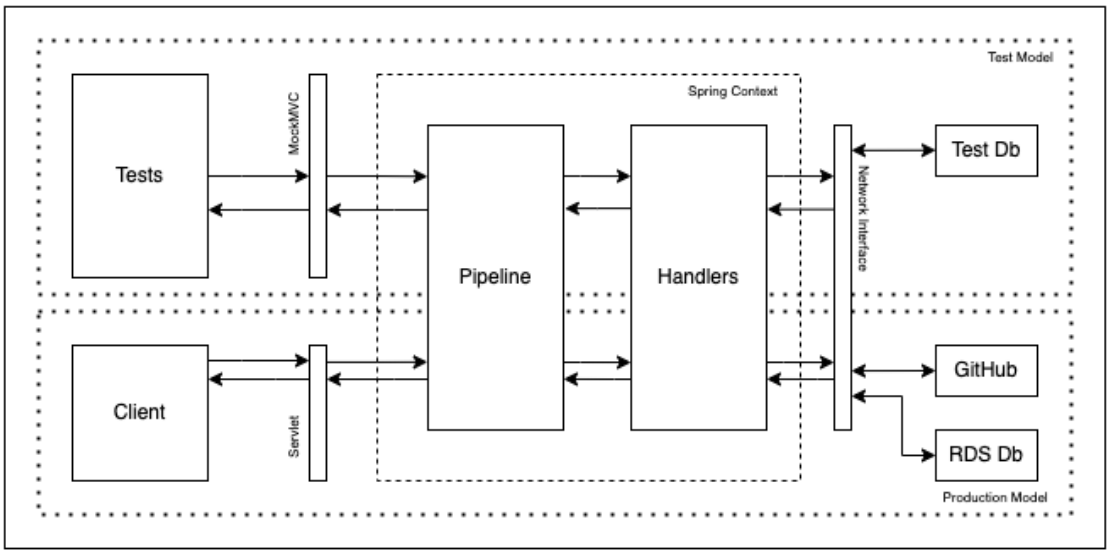

Figura 3.5: Comparação do contexto Spring em testes e em produção.

Capítulo 4

Aspetos de Implementação das Aplicações

De forma a obter um sistema completo é necessário o **i-on Teams Service**, retratado no capítulo anterior, e uma componente aplicacional, através da implementação de aplicações clientes do serviço.

Tendo em conta o objetivo do projeto, foi decidida a implementação dois clientes distintos. Um cliente destinado ao docente, que se trata de uma aplicação desktop que permitirá a total integração do serviço e das funcionalidades previstas prestar ao docente. Por outro lado o aglomerado de informação que este projeto armazena também pode ser muito útil para os estudantes, e desta forma foi também criado um segundo cliente, uma aplicação web, para ser utilizada exclusivamente pelos estudantes para consulta de informações académicas consideradas relevantes.

Este capítulo irá aprofundar a implementação de ambos os clientes, descrevendo as decisões tomadas no decorrer da sua implementação e das tecnologias utilizadas.

4.1 Aplicação Desktop

4.1.1 Acesso ao i-on Teams Service

A aplicação desktop é um cliente do serviço e grande parte dos pedidos feitos à API que este expõe devem ir devidamente autenticados. A única exceção a esta regra são os pedidos para os endpoints de autenticação, visto que é a partir destes que a aplicação consegue obter a autorização para fazer os restantes.

A autorização referida anteriormente é fornecida pelo serviço por meio de um *cookie* de sessão, que é atribuído aquando do *login* de um utilizador, e revogado no respetivo *logout*.

Findo o processo de autenticação, caso este seja concluindo com sucesso, toda a restante comunicação com o serviço é efetuada através da Web API exposta por ele, permitindo a consulta e criação dos recursos necessários ao correto funcionamento da aplicação.

Os pedidos para obtenção de recursos são feitos através de URIs previamente conhecidos

pela mesma. Apesar desta não ser a melhor abordagem, pois visto que as respostas têm a informação necessária para a navegação, tornando o processo dinâmico e evitando *breaking changes* em possíveis futuras versões da Web API que necessitem de alterar as localizações dos recursos, esta foi aplicada dados os constrangimentos temporais que foram sentidos e que impediram esta implementação. No entanto, todos os pedidos de criação, atualização e remoção (POST, PUT e DELETE) de recursos, são feitos através de informação obtida dinamicamente através das respostas dos pedidos de obtenção de recursos (GET), isto inclui a localização, método HTTP a ser utilizado, tipo do corpo do pedido e os campos que devem existir no corpo do pedido, completo com o seu nome e tipo.

Visto de a aplicação desktop é uma SPA, todos os pedidos efetuados ao serviço são realizados na própria SPA, a ser executada no *renderer process*, e não pelo *main process* da aplicação

4.1.2 Integração com o Ambiente de Execução

O Electron possui um modelo de processos baseado no do *browser open-source* **Chromium**. Isto resulta num nível de isolamento entre os conteúdos apresentados ao utilizador, executados num processo chamado de *renderer process*, e as funções necessárias para a execução da aplicação, da responsabilidade do *main process*.

Sendo esta aplicação uma aplicação desktop, a mesma tem de ser capaz de comunicar com as diferentes APIs fornecidas pelo sistema operativo. Este tipo de integração é fornecida pelo Electron através da API que expõe, no entanto, as comunicações com o sistema operativo e com outras aplicações em execução no mesmo apenas deve ser realizado no *main process* da aplicação. Dado que o *main process* tem acesso a toda esta informação, o livre acesso ao mesmo e às APIs de integração com o sistema operativo podem tornar-se facilmente em vulnerabilidades de um ponto de vista da segurança.

Para colmatar este problema foi criada uma barreira entre o *main process* e os *renderer processes* para que os mesmos não tenham livre acesso às funcionalidades expostas no processo principal. Assim sendo, toda a comunicação deve ser previamente definida. Este tipo de comunicação é denominado como *Inter-Process Comunication* e será explicado em maior detalhe mais a frente neste relatório.

É também importante que outras aplicações tenham a capacidade de comunicar com a desktop app, assim é possível que o fluxo de autenticação seja concluído com uma mensagem enviada do browser para a aplicação desktop em execução. Para isso foi utilizado *Deep-Links* [31] através de um *Custom Protocol* definido para a nossa aplicação. Este processo também será explicado em mais detalhe de seguida.

Todas as configurações específicas à **desktop app** necessárias ao cumprimento dos requisitos mencionados anteriormente, tais como *listeners* de *inter-process comunication*, configuração do *custom protocol* para *deep-links* e até a configuração da janela de apresentação,

foram realizadas no ficheiro **main.js**.

Inter-Process Comunication

Como foi referido anteriormente *Inter-Process Comunication* é o nome dado ao mecanismo utilizado nas aplicações Electron para comunicação entre o *main* e os *renderer processes*. Esta comunicação é conseguida através do estabelecimento de APIs que serão expostas em tempo de execução no *renderer process*.

Existem três padrões de comunicação entre processos disponíveis nas aplicações Electron:

- Do *renderer* para o *main process*. Este padrão permite a comunicação unidirecional entre o *renderer process* e o *main process*, sendo possível o envio de objetos que depois serão recebidos no *main process*;

- Do *main* para o *renderer process*. Este padrão, também unidirecional, permite a comunicação entre o *main process* e o *renderer process* através de um *listener* implementado no *renderer process* que fica à escuta de mensagens num canal específico de comunicação.

- Bidirecional, do *renderer process* para o *main process* com possibilidade de obter uma resposta. Este padrão de comunicação permite ao *renderer process* fazer pedidos e/ou enviar informação para o *main process* e obter de seguida uma resposta sobre o mesmo. As respostas são assíncronas e não bloqueantes, o que significa que o *renderer process* recebe uma *promise* da resposta.

Para cada um destes padrões são estabelecidos canais de comunicação consoante o comportamento esperado para cada tipo de pedido, ou seja, podem ser estabelecidos diferentes canais de comunicação consoante a mensagem que é desejada passar e o comportamento que deve existir sobre essa mensagem.

Os padrões estabelecidos são disponibilizados ao *renderer process* e, consequentemente, à *SPA* através de APIs definidas no ficheiro **preload.js**.

No âmbito da aplicação desktop foram usados apenas os padrões de desenho unidirecionais com os seguintes propósitos:

- *Renderer process* para *main process*:
  - Acesso ao *clipboard* para copiar um *invite-code* ao carregar num botão.
  - Abertura de um browser externo para dar início ao fluxo de autenticação.

- *Main process* para *renderer process*:
  - Envio do url recebido no *main process* aquando da chamada da aplicação pelo browser pelo seu *custom protocol*.

Deep-Links

No contexto da aplicação desktop do sistema i-on Teams é fundamental a existência de um canal de comunicação entre a mesma e outras aplicações em execução no sistema operativo, nomeadamente o browser. O browser é utilizado como ponte entre o serviço, o GitHub e a aplicação para efeitos de autenticação de utilizadores. O Electron possui um mecanismo que permite que este tipo se integrações sejam conseguidas pelas aplicações através dele desenvolvidas. Este mecanismo chama-se *Deep-Links*, que consiste no uso de um *hyperlink* que navega especificamente para um recurso ou conteúdo específico, em vez da pagina *home* de uma aplicação.

Através da criação de um *custom protocol* foi possível o broswer chamar, trazer a foco e enviar informação para a aplicação desktop. Este *custom protocol* é definido no ficheiro *main.js* e, como tal, é gerido pelo *main process* da aplicação. Depois da sua definição a aplicação, através do seu *main process* fica à escuta por chamadas a esse protocolo.

O *custom protocol* definido no ambito da aplicação desktop em análise foi: "ion-teams://". Todas as chamadas a este protocolo serão interpretadas pelo *main process*.

Como foi referido anteriormente este mecanismo é extremamente importante na autenticação de utilizadores na aplicação pois parte do fluxo de autenticação tem de ser realizado diretamente pela aplicação e o browser tem de ter a capacidade de passar a informação necessária para que a aplicação consiga fazê-lo.

O mecanismo de *Deep-Links* tem como uma das suas principais características o facto da sua implementação ter de ser diferente consoante o sistema operativo em que a aplicação está em execução. Esta particularidade faz com que na sua implementação tenham de ser previstas a realização de chamadas tanto em ambientes Windows como em ambientes MacOS ou Linux. Isto deve-se a duas principais razões:

- O facto de por omissão uma chamada a um custom protocol no Windows lançar uma nova instância da aplicação. Resolvido ao analisar a criação de novas instâncias e, caso se verifique que a aplicação já se encontrava em execução, parar essa nova instância e chamar a foco a atual;
- O facto de para MacOS e Linux estar disponível um evento, que pode ser escutado, e que disponibiliza o url passado na chamada do *custom protocol*. Este evento não está presente em Windows, pelo que a obtenção do url passado deve ser feita através dos parâmetros passados no *array argv*.

4.1.3 Autenticação

A aplicação desktop possui também a particularidade de servir de ponte entre o sistema i-on Teams e o GitHub. Visto que existe um paralelismo entre os recursos presentes no serviço e os recursos existentes no GitHub é necessário haver comunicação com a API disponibilizada

pelo mesmo. Para que haja esse nível de comunicação, que envolve escrita e leitura de recursos do GitHub, é necessário um *Access Token* com esse tipo de permissões. Visto que esse *Access Token* está associado ao docente e é altamente sensível, armazena-lo no serviço tornava-se numa vulnerabilidade do sistema. Assim sendo foi tomada a decisão de armazená-lo localmente na máquina do docente através da aplicação desktop. Foi então adicionada uma responsabilidade à aplicação desktop de ser o ponto do sistema que comunica com a API do GitHub.

O *Access Token* referido é obtido pela aplicação no concluir da autenticação do utilizador e enviado pelo serviço para a aplicação desktop de duas maneiras:

- A primeira é através do corpo da resposta de sucesso a um endpoint do serviço específico para o efeito. Isto faz com que a aplicação possua o *Access Token* durante o seu tempo de vida. Em execução o *Access Token* é armazenado num *context* acessível em qualquer componente da aplicação.

- A segunda é através de um *cookie*. Este é armazenado na aplicação, mas a aplicação não consegue aceder ao mesmo, tendo de requisitar ao i-on Teams Service o seu valor, sendo que o *cookie* é enviado para o serviço. O objetivo de armazenar o *Access Token* também num *cookie* é o de manter o mesmo armazenado para além da instância da aplicação em que foi criado. Sempre que a aplicação entra em execução faz um pedido ao serviço para o mesmo *endpoint* utilizado para obter o *Access Token*, e o serviço verifica que o pedido possui o *cookie* com o *Access Token* e se possuir envia-o de volta para a aplicação para que a mesma possa tê-lo presente na atual instância. Ao receber uma resposta com sucesso, que contem o *Access Token* a aplicação inicia sessão automaticamente, não sendo necessário o utilizador iniciar sessão novamente durante o tempo de vida do *cookie*.

4.1.4 Comunicação com o GitHub

Devido a limitações temporais não foi possível o desenvolvimento da integração entre a aplicação desktop e o GitHub através da sua API.

Esta comunicação serviria para criar e gerir repositórios GitHub de turmas e equipas de um determinado docente, assim como confirmar o estado das entregas dos trabalhos e o download de cópias dos repositórios para que os trabalhos pudessem ser analisados pelo docente.

A mesma seria autenticada através do *Access Token* do docente, descrito na secção anterior, que daria permissões de escrita e leitura para a criação, atualização e consulta de repositórios nos quais o docente se inseria.

4.1.5 Interação com Utilizadores

Considerando que se trata de uma aplicação desktop esta necessita de uma instalação prévia à sua utilização, para tal o utilizador terá de aceder à aplicação web para proceder ao download.

Depois de instalar a aplicação e abrir, o docente terá de se registar fornecendo a sua informação e todos os passos consequentes de forma a concluir o processo de registo, incluindo a verificação do email. Findo o registo o utilizador pode iniciar a sua sessão, autenticando-se pelo GitHub através do botão designado para o efeito.

Ao iniciar sessão é apresentado ao utilizador um ecrã principal que contem uma lista das organizações em que se insere, completa com as turmas às quais do docente pertence. Toda a informação apresentada nessas listas é interativa pelo que se o docente carregar numa determinada turma ou organização será levado para a página específica desse recurso.

Durante toda a utilização da aplicação, a partir do início de sessão, uma barra lateral de navegação é também apresentada. A mesma é dinâmica e mostra conteúdo relevante consoante o recurso a ser visualizado. O utilizador pode navegar através desta barra lateral, terminar a sua sessão ou regressar à página principal da aplicação.

Em cada página de recurso são também apresentadas as ações que o utilizador pode tomar sobre esse recurso, tais como: criar um novo recurso; atualizar um recurso existente; eliminar um recurso. As ações permitidas variam entre recurso e resultam da interpretação das respostas aos pedidos feitos à API sobre o recurso pretendido.

As páginas que apresentem listas contêm também paginação, desde que o recurso pedido seja uma lista com mais de dez elementos. A aplicação interpreta a resposta e identifica a existencia de paginação apresentado os botões *next* e *previous* no caso de haver a possibilidade de navegar para uma página seguinte ou anterior.

4.1.6 Estrutura do Código

O código da aplicação web encontra-se na diretoria **code**, dentro da pasta **js**, como já tinha sido referido, havendo depois a pasta **desktop** que contém todo o código referente à aplicação desktop. Por cada página existirá uma pasta, na qual o nome identifica a página a que se refere, também existirá com o componente **Page** e, caso se aplique, uma pasta **components** com os componentes utilizados para apresentar o recurso correspondente à mesma.

- Diretoria **code/js/web**
  - Pasta **public** - Além dos ficheiros descritos abaixo contém, também, as imagens (*svg*) utilizadas na aplicação.
    - Ficheiro **icon.ico** - *icon* que o electron usa no sistema operativo windows;
    - Ficheiro **icon.icns** - *icon* que o electron usa no sistema operativo macOS.

- Pasta **src** - Contém o código fonte. Onde estão localizadas as pastas por recurso, entre outros referidos a seguir.
  - Pasta **common** - Aqui encontram-se todos os componentes e tipos comuns, e também, o ficheiro **Uris** com os links utilizados.
  - Ficheiro **App.tsx**
  - Ficheiro **index.tsx**
  - Pasta **Router** - contém o componente **Router**
- Ficheiro **index.html** - Responsável por definir o *page title* e a tag *script*;
- Ficheiro **main.js**
- Ficheiro **preload.js**

4.2 Aplicação Web

4.2.1 Acesso ao Serviço

A aplicação comunica com o i-on teams service como cliente, ou seja, o serviço tem de autorizar a comunicação com a mesma. Sempre que forem executados pedidos que não correspondem à autenticação o serviço exige uma sessão que será validada.

A aplicação conhece apenas um link, que corresponde ao recurso que contém a informação apresentada na página inicial. Na resposta obtida são disponibilizados, pelo serviço, uma série links relevantes para requisitar informação de recursos adjacentes. Existe a exceção dos links de autenticação, isto é, *login*, *sign up* e *logout* que são conhecidos.

4.2.2 Interação com Utilizadores

A primeira interação será autenticação de forma a aceder à sua informação pessoal e completar a verificação via email, caso não prossiga com a autenticação ou a verificação apenas terá acesso à pagina inicial e à pagina *About*, que contém uma contextualização do projeto.

Durante toda navegação será apresentada uma barra, no topo da página, que contém a página atual, caso esta não seja de inscrição, as páginas com possível navegação e ações perante o estado da sessão. Na pagina inicial, pós inicio de sessão, é possível inserir um código que representa um convite para formação de *team* na respetiva *classroom* que após um clique no botão levará a página capaz de realizar a ação. Também é possível, visualizar todas as *teams* a que o utilizador (estudante) pertence, mostrando também informação sobre a *organization* e a *classroom* onde a *team* se insere.

Quanto à pagina de inscrição numa *classroom* é possível criar uma equipa nova, atribuindo um nome personalizado à mesma, ou aderir a uma equipa já existente.

Sempre que uma ação é executada, é apresentada conforme a resposta obtida, uma mensagem de sucesso ou erro. Em caso de erro a carregar um página onde é feito um pedido o serviço, é disponibilizado o código do erro obtido e respetiva mensagem.

4.2.3 Estrutura do Código

O código da aplicação web encontra-se na diretoria **code**, dentro da pasta **js**, como já tinha sido referido, havendo depois a pasta **web** que contém todo o código referente à aplicação web. Por cada página existirá uma pasta, na qual o nome identifica a página a que se refere, também existirá com o componente **Page** e, caso se aplique, uma pasta **components** com os componentes utilizados para apresentar o recurso correspondente à mesma.

- Pasta **public** - Além dos ficheiros descritos abaixo contém, também, as imagens utilizadas na aplicação.
  - Ficheiro **index.html** - Responsável por definir o *page title* e a tag *script*,
  - Ficheiro **favicon.ico** - *icon* que o browser usa.
  - Pasta **src** - Contém o código fonte. Onde estão localizadas as pastas por recurso, entre outros referidos a seguir.
    - Pasta **common** - Aqui encontram-se todos os componentes e tipos comuns, e também, o ficheiro **Uris** com os links utilizados.
    - Ficheiro **App.tsx**
    - Ficheiro **index.tsx**
    - Ficheiro **Router.tsx**

4.3 Implementação das SPA

A implementação das **SPA** em **React**, também designada como *user interface*, das aplicações tiveram abordagens semelhantes com exceção da implementação do **React Router** [32] devido à diferença da Framework onde as *single page applications* estão a ser executadas.

Começando pela diferente implementação do **React Router**, no caso da **Desktop-app** foi utilizado **HashRouter** [33] dado que o **Electron** não possui um *history stack*, que possibilita a navegação dentro da aplicação. Enquanto que na **Web-app** foi possível utilizar o **BrowserRouter** [34] como recomendado quando a aplicação está a ser executada no browser, onde existe um *built-in history stack*.

Quanto às semelhanças é possível destacar os seguintes aspetos de implementação:

- Componente **Fetch** capaz de fazer pedidos dividindo um pedido em vários estados, através do **React Reducer** [35], tendo componentes *default* para erros e a opção de os substituir passando novos componentes na sua chamada. O **Fetch** possibilita também o cancelamento de pedidos e evita que sejam feitos pedidos duplicados;
- Componente **DefaultForm**, que receber um *array* de ações em *siren* constrói dinamicamente um formulário para *input* das informações necessárias. Este componente

é também responsável pela elaboração dos pedidos resultates do formulário à API do serviço;

- A construção das tabelas apresentadas ao utilizador também contém um certo nível de abstração. Foi criado um componente genérico **Table** que é reponsável pela criação de uma tabela e é utilizado por componentes específicos para cada recurso que criam e fornecem o conteudo da tabela;

- Foi também criada através de um **React Reducer** uma caixa de mensagens que permite transmitir mensagens de sucesso, informação e erro ao utilizador;

- Foram utilizados **Context** [36] de forma a poder haver acesso a um determinado estado por todos os componentes "filho"do componente onde o *context* foi injetado. Estes foram utilizados para:

  - Estado *Login/Sign up* - disponível para todos os componentes filhos do **App.tsx**;
  - Paginação - disponível apenas nos componentes onde é necessária paginação;
  - Menu - Dois contextos associados, disponíveis em todos os componentes representantes de páginas com o objetivo de passar informação sobre a página atual e as páginas disponíveis para navegação, e outro contexto para o nome do recurso a ser apresentado.

Capítulo 5

Conclusões

5.1 Conclusão

O projeto i-on Teams teve como objetivo criar um sistema para auxiliar os docentes a gerir os trabalhos de grupo realizados no âmbito das suas unidades curriculares. Resolvendo problemas identificados na utilização do Github Classroom, um sistema do próprio GitHub com o objetivo de facilitar o uso de sistemas de controlo de versões em sala de aula. Após o desenvolvimento do sistema, conseguimos solucionar parte das lacunas identificadas. Todas as funcionalidades implementadas têm entre sim um nível de isolamento que permite a evolução de componentes isolados do sistema sem que seja preciso fazer alterações noutros. Visto que o sistema foi desenvolvido com uma componente servidora responsável pelo armazenamento de recursos e outras duas componentes aplicacionais que servem de interface gráfica utilizada pelos utilizadores, é possível que no futuro sejam construidos novos clientes que tomem proveito dessa componente servidora, sem que a mesma necessite de ser substancialmente alterada.

Quanto às dificuldades sentidas no desenvolvimento do sistema, consideramos que a aprendizagem de novas tecnologias pode ser destacada como um das maiores dificuldades, com ênfase para o **Electron** pelo seu modo de funcionamento complexo com múltiplos processos e a sua integração com o sistema operativo e as restantes aplicações que nele estão em execução, entre outras razões. Também com as tecnologias **Spring** e **ReactJS** que apesar de não serem completamente desconhecidas foi necessário evoluir o conhecimento de forma a aplicar a casos específicos ao desenvolvimento do sistema.

Concluindo, sentimos que o nosso conhecimento e capacidade de desenvolver código ou solucionar problemas foi posto à prova e consequentemente foi-nos possível evoluir o nosso conhecimento técnico, ficando satisfeitos com a qualidade do resultado final.

5.2 Trabalho Futuro

O trabalho futuro do sistema está divido em duas secções: funcionalidades que não foram possível serem implementadas; e novas funcionalidades ou ideias que foram surgindo ao longo

do desenvolvimento do sistema.

Começando por funcionalidades propostas e não implementadas:

- Implementação da comunicação com o GitHub, que por não ter sido implementada condiciona as seguintes funcionalidades:
  - Criação, atualização e remoção de repositórios GitHub;
  - Verificação de entregas através de *tags*;
  - Download dos repositórios das diferentes equipas para a máquina do docente.
- Consulta de informação sobre o docente por parte do aluno (ex. email, gabinete, etc.).

Por fim, novas funcionalidades:

- Atualização da página da aplicação de acordo com resposta de um pedido de criação, eliminação ou atualização de um recurso, eliminando a necessidade de fazer um novo pedido para o recurso por inteiro;
- Expiração de *invite codes*, atualmente são criados códigos sem qualquer tipo de possibilidade de os eliminar e sem que os mesmos tenham tempo de vida;
- Convidar um professor para co-gerir uma *classroom* ou *organization*;
- Ao adicionar um *note* a uma *team* guardar informação do docente que o fez;
- Email automático para o(os) docente(es) após a entrega de um *assignment*, informando as *teams* que entregaram e as que não o fizeram;
- Realizar **code signing** [37] para melhorar a segurança da aplicação contra alterações maliciosas e para que a aplicação não dê alertas de segurança nos sistemas operativos em que é executada.
- Atualizações automáticas da aplicação desktop através do serviço disponibilizado pelo Electron para o efeito, **update.electronjs.org** [38].

Referências

[1] i-on. https://github.com/i-on-project.

[2] ISEL. https://www.isel.pt/.

[3] GitHub Classroom. https://classroom.github.com/.

[4] GitHub. https://github.com/.

[5] GitHub OAuth App. https://docs.github.com/en/developers/apps/getting-started-with-apps/differences-between-github-apps-and-oauth-apps.

[6] Spring. https://spring.io/projects/spring-boot.

[7] Kotlin. https://kotlinlang.org/.

[8] Heroku. https://www.heroku.com/.

[9] PostgreSQL. https://www.postgresql.org/.

[10] AmazonRDS. https://aws.amazon.com/rds/postgresql/.

[11] Amazon Web Services (AWS). https://aws.amazon.com/.

[12] Mediatype application/siren+json. https://github.com/kevinswiber/siren.

[13] JSON - JavaScript Object Notation. https://www.rfc-editor.org/rfc/rfc4627.

[14] Mediatype application/problem+json. https://www.rfc-editor.org/rfc/rfc7807.

[15] Semantic UI. https://semantic-ui.com/.

[16] Semantic UI React. https://react.semantic-ui.com/.

[17] Webpack. https://webpack.js.org/.

[18] TypeScript. https://www.typescriptlang.org/.

[19] ReactJS. https://reactjs.org/.

[20] Electron. https://www.electronjs.org/.

[21] ElectronJS's Process Model. https://reactjs.org/.

[22] Chromium. https://www.chromium.org/Home/.

[23] OAuth 2.0. https://oauth.net/2/.

[24] Cross-Origin Resource Sharing. https://developer.mozilla.org/pt-BR/docs/Web/HTTP/CORS.

[25] Teams Repository. https://github.com/i-on-project/teams.

[26] JDBI. https://jdbi.org/.

[27] Controller Advice. https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/ControllerAdvice.html.

[28] Twilio SendGrid Email. https://sendgrid.com/go/email-brand-signup-sales-1?utm_source=google&utm_medium=cpc&utm_term=sendgrid&utm_campaign=SendGrid_G_S_Brand_ROE_Mature&cq_plac=&cq_net=g&cq_pos=&cq_med=&cq_plt=gp&gclid=CjwKCAjw9suYBhBIEiwA7iMhNIZ99j7LSQXFLWb_HDrIkd6dN5Bnx0kjXirdViuSCVH_Vbc367euJBoCzrgQAvD_BwE.

[29] Interceptor. https://www.baeldung.com/spring-mvc-handlerinterceptor.

[30] MockMVC. https://docs.spring.io/spring-framework/docs/current/reference/html/testing.html#spring-mvc-test-framework.

[31] Deep-Links. https://www.electronjs.org/docs/latest/tutorial/launch-app-from-url-in-another-app.

[32] React Router. https://reactrouter.com/en/main.

[33] HashRouter. https://reactrouter.com/en/main/routers/hash-router.

[34] BrowserRouter. https://reactrouter.com/en/main/routers/browser-router.

[35] React - useReducer. https://reactjs.org/docs/hooks-reference.html#usereducer.

[36] Context. https://reactjs.org/docs/context.html#gatsby-focus-wrapper.

[37] Code Signing. https://www.electronjs.org/docs/latest/tutorial/code-signing.

[38] ElectronJS Updater. https://github.com/electron/update.electronjs.org.