ISEL - Instituto Superior de Engenharia de Lisboa

i-on Codegarten

Diogo Sousa
Tiago David
João Moura

Orientador: Pedro Félix

Relatório realizado no âmbito de Projecto e Seminário,
do curso de Licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2020/2021

Julho de 2021

Instituto Superior de Engenharia de Lisboa

i-on CodeGarten

46008 Diogo Vasconcelos de Sousa

46078 Tiago Manuel Morgado David

46107 João Pedro Marques Moura

Orientador: Pedro Miguel Henriques Santos Félix

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

Julho de 2021

Resumo

Atualmente, existem unidades curriculares no Instituto Superior de Engenharia de Lisboa (ISEL) que utilizam a plataforma GitHub para a gestão e entrega de código fonte realizado no âmbito de projetos e trabalhos práticos. Algumas das vantagens desta abordagem são a colaboração entre alunos, a introdução dos mesmos em ambientes utilizados em engenharia de *software* e o incentivo do uso de *software* de controlo de versões para a gestão de código desenvolvido.

Para tornar esta abordagem viável, é necessário gerir o acesso aos repositórios de cada grupo/aluno, assim como verificar o estado atual do trabalho e as suas entregas. Uma das formas existentes para esse efeito é a utilização do sistema GitHub Classroom, sistema esse que é utilizado em algumas unidades curriculares. No entanto, este sistema apresenta desvantagens na sua utilização, nomeadamente a inexistência de automatização para a verificação de entregas de trabalhos, a dificuldade em relacionar utilizadores GitHub com alunos, bem como a dificuldade em criar/atualizar cópias locais dos repositórios dos alunos.

Este relatório documenta o sistema desenvolvido que pretende solucionar os problemas acima apresentados, fornecendo funcionalidades como a gestão de entregas de trabalhos, a criação automática de repositórios GitHub para alunos, bem como a relação de utilizadores GitHub a uma identificação específica ao domínio do sistema. Este sistema é composto por uma aplicação servidora que expõe uma API e que implementa a lógica do sistema, e por uma aplicação *web* que funciona como um cliente desta API. Esta aplicação *web* expõe funcionalidades semelhantes ao sistema referenciado acima (GitHub Classroom), mas resolvendo algumas das limitações nela presentes. Ambas estas aplicações encontram-se hospedadas *online* na plataforma Heroku.

Palavras Chave: i-on, CodeGarten, Académico, Alunos, Docentes, Repositórios, API, GitHub, Autenticação, Spring, Kotlin, PostgreSQL, JavaScript, Multi-Page Application, OAuth

Abstract

Currently, there are courses at the Instituto Superior de Engenharia de Lisboa (ISEL) that use the GitHub platform for the management and delivery of source code within the context of projects and assignments. Some of the advantages of this approach are the collaboration between students, their introduction to environments used in software engineering and the encouragement of using a version control system to manage developed code.

To make this approach viable, it is necessary to manage access to the repositories of each group/student, as well as check the current status of the assignment and its deliveries. One of the available ways to achieve this goal is to use the GitHub Classroom system, which is used in some courses. However, this system has some disadvantages, namely the lack of automation for the verification of assignment deliveries, the difficulty in relating GitHub users with students, as well as the difficulty in creating/updating local copies of student repositories.

This report documents the developed system that aims to solve the problems presented above, providing functionalities such as the management of assignment deliveries, the automatic creation of GitHub repositories for students, as well as relating GitHub users to a domain-specific identification present in the system. This system is composed of a server application that exposes an API and implements the system logic, as well as a *web* application that acts as a client of the API. This *web* application provides features similar to the system referenced above (GitHub Classroom), while solving some of its limitations. Both of these applications are hosted *online* on the Heroku platform.

Keywords: i-on, CodeGarten, Academic, Students, Teachers, Repositories, API, GitHub, Authentication, Spring, Kotlin, PostgreSQL, JavaScript, Multi-Page Application, OAuth

Agradecimentos

Primeiramente, gostaria de agradecer ao professor Pedro Félix por todo o apoio e disponibilidade que teve com o nosso grupo. Creio que o conhecimento que partilhou connosco é muito valioso e foi uma honra trabalhar com quem considero um dos melhores professores do ISEL. Agradeço também a todos os membros da iniciativa i-on pelo seu acolhimento, com destaque ao Ricardo Margalhau, pois foi graças a ele que tivemos a oportunidade de realizar este projeto.

Deixo o meu agradecimento ao meu excelente grupo, pelo companheirismo, trabalho em equipa, disponibilidade e conhecimento que partilharam comigo ao longo de todos estes semestres.

Agradeço a todos os colegas e professores que tive ao longo do meu percurso académico neste instituto pela cooperação e espírito de entre-ajuda. As experiências que tive tanto com colegas como com docentes ao longo destes três anos ficarão para sempre na minha memória.

Por fim, agradeço profundamente à minha família por todo o seu apoio e por me fornecerem as condições para que pudesse realizar esta licenciatura.

Junho 2021, Diogo Sousa

Começo por agradecer ao meu colega Ricardo Margalhau por ter convidado o nosso grupo para a iniciativa i-on. Sem ele este projeto não seria possível. Agradeço também ao nosso orientador, Pedro Félix, por propor a ideia para o projeto e por nos suportar ao longo do desenvolvimento do mesmo. A sua disponibilidade e apoio tornou a realização do projeto mais agradável e intrigante. Deixo o meu agradecimento a todos os membros do i-on, docentes e alunos, por serem extremamente acolhedores desde o momento que entrámos na iniciativa.

Agradeço a ambos os meus colegas de grupo, por se mostrarem sempre disponíveis a cooperar e por partilharem a experiência deste curso comigo. Os meus agradecimentos também a todos os colegas e professores com que interagi ao longo deste percurso académico. Agradeço a todos pelo espírito de equipa e de colaboração que demonstraram. Obrigado por todas as amizades e todas as recordações que me deram ao longo destes três anos.

Concluo agradecendo à minha família por sempre me apoiarem e por me darem a oportunidade de realizar licenciatura no Instituto Superior de Engenharia de Lisboa.

Junho 2021, Tiago David

Em primeiro lugar, gostaria de agradecer ao professor Pedro Félix, que nos orientou neste projeto. A sua disponibilidade e o seu apoio foram imprescindíveis para a conclusão do mesmo. Agradeço também ao Ricardo Margalhau por nos ter convidado para a iniciativa i-on, assim como a todos os membros desta por nos terem recebido da melhor forma possível.

Gostaria também de agradecer aos meus colegas de grupo pelo apoio e pela capacidade de colaboração que demonstraram no decorrer deste semestre. A estes e aos restantes colegas, obrigado pelas amizades e memórias que foram feitas. Aos professores que tive o prazer de conhecer, muito obrigado pelo apoio e por todo o conhecimento que partilharam comigo.

Quero também agradecer em especial à minha família, que sempre me apoiou em todas as ocasiões. Fico imensamente grato por me terem oferecido a oportunidade de realizar esta licenciatura.

A todos aqueles que foram mencionados, um grande obrigado por fazerem destes três anos um período que nunca me irei esquecer.

Junho 2021, João Moura

Índice

Lista de Figuras

Capítulo 1

Introdução

O presente capítulo introduz a iniciativa i-on, assim como a motivação, objetivos e especificações do projeto a desenvolver. É também introduzida a estrutura deste relatório.

1.1 Iniciativa i-on

A iniciativa i-on [1] visa desenvolver aplicações informáticas para apoio a atividades académicas, bem como disponibilizar informações relativas a estas de forma computacionalmente conveniente. As aplicações desta iniciativa são desenvolvidas por alunos do Instituto Superior de Engenharia de Lisboa (ISEL) e encontram-se disponíveis em formato *open-source*.

O projeto i-on CodeGarten insere-se na iniciativa i-on pois pretende fornecer, a alunos e a docentes, um sistema para apoio à gestão de repositórios GitHub em contexto académico.

1.2 Motivação

Atualmente, existem unidades curriculares no ISEL que utilizam a plataforma GitHub para a gestão e entrega de código fonte realizado no âmbito de projetos e trabalhos práticos. Algumas das vantagens desta abordagem são:

- Promover a colaboração entre alunos;
- Introduzir os alunos a ambientes utilizados em engenharia de *software*;
- Incentivar o uso de *software* de controlo de versões para gerir o código desenvolvido.

Uma das principais razões que levaram ao início deste projeto foram as dificuldades manifestadas pelos docentes na utilização de ferramentas para gestão de trabalhos académicos, mais concretamente o sistema GitHub Classroom [2]. Também existiu a necessidade de melhorar a experiência dos alunos ao consultar as unidades curriculares que utilizam a plataforma GitHub para a realização dos seus trabalhos, assim como melhorar a forma de entrega dos mesmos, pois atualmente os alunos têm de verificar no enunciado quais os passos necessários para efetuar a mesma. Existem também várias desvantagens em utilizar o sistema, tais como:

- Falta de automatização da verificação de entregas dos trabalhos;
- Dificuldade em relacionar utilizadores GitHub com os alunos;
- Dificuldade em criar/atualizar cópias locais dos repositórios dos alunos.

1.3 Objetivo

O objetivo deste projeto é o desenvolvimento de um sistema que pretende solucionar os problemas apresentados na secção 1.2, fornecendo funcionalidades como a gestão de entregas de trabalhos, a criação automática de repositórios GitHub para alunos, bem como a relação de utilizadores GitHub a uma identificação específica ao domínio do sistema.

1.4 Especificação do Projecto

Para que o sistema seja capaz de alcançar os objetivos pretendidos, foram definidos os seguintes requisitos mínimos:

- Autenticação de utilizadores através da sua conta GitHub;
- Listagem das várias organizações onde o utilizador se encontra;
- Listagem e criação de turmas numa dada organização;
- Listagem e gestão de alunos e grupos numa turma;
- Listagem e gestão de trabalhos numa turma;
- Criação automática de repositórios GitHub aquando a entrada de um aluno ou grupo num trabalho;
- Listagem e planeamento de entregas de trabalhos;
- Verificação das entregas realizadas por alunos por parte dos docentes;
- Capacidade de convidar alunos para uma turma ou para um trabalho em específico.

1.5 Método de Trabalho

Todo o projeto foi desenvolvido tirando partido do sistema de controlo de versões Git e o serviço GitHub, utilizando um repositório público [3] para o efeito. Utilizou-se o mecanismo de *issues*, *pull requests* e *projects*.

No contexto do projeto, uma *issue* é uma tarefa a realizar, que pode ser uma nova funcionalidade ou um problema a resolver. As alterações relativas a uma *issue* são colocadas num *branch* separado do principal onde podem ter várias iterações (*commits*). Caso este *branch* satisfaça os requisitos apresentados na *issue*, é realizado um *pull request* onde o código fica disponível para revisão por parte do orientador e dos restantes elementos do grupo. Caso seja aprovado, é realizado um *merge* para o *branch* principal.

Esta abordagem permite que haja desenvolvimento em paralelo para o projeto bem como troca de opiniões entre os vários elementos do grupo.

1.6 Estrutura do Relatório

No capítulo 2 são identificados os aspetos mais relevantes da arquitetura do sistema i-on CodeGarten. No capítulo 3 são apresentados aspetos de implementação do sistema, assim como as razões das decisões tomadas. No capítulo 4 são resumidas as soluções propostas, bem como os desafios que foram encontrados com o desenvolvimento do projeto.

Capítulo 2

Arquitetura

Neste capítulo é realizada uma descrição da arquitetura do sistema desenvolvido, estando os seus componentes ilustrados na figura 2.1.

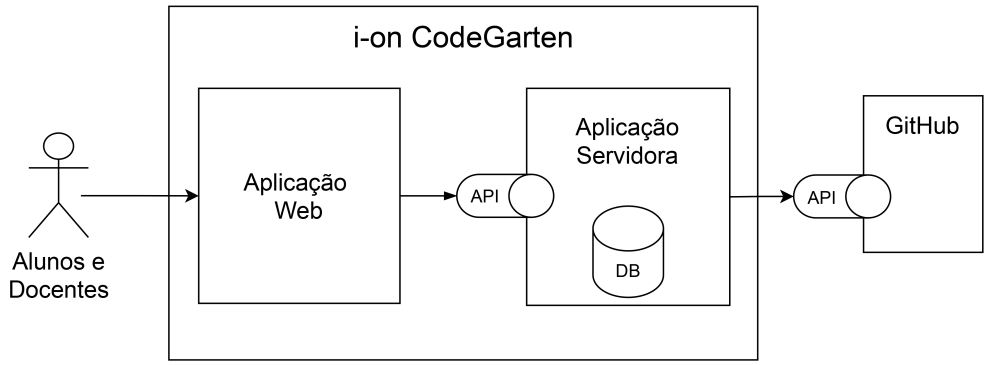

Figura 2.1: Arquitetura do sistema

O sistema divide-se em duas aplicações. A principal é a aplicação servidora, responsável por implementar a lógica do sistema, armazenar informação, gerir a interação com o GitHub e expor uma API HTTP que disponibiliza um conjunto de funcionalidades para possíveis clientes. A outra aplicação do sistema é a aplicação *web* que age como um cliente da aplicação servidora, permitindo gerir interações com utilizadores através duma interface gráfica.

Foi escolhida esta abordagem de modo a suportar a possível existência de múltiplas formas de interação com os utilizadores, quer seja na forma de aplicação *web*, móvel ou *desktop*. Deste modo, os clientes não necessitam de armazenar informação relativa ao sistema, implementar a lógica do mesmo, nem de implementar um mecanismo de interação com o GitHub, pois estes elementos encontram-se presentes na aplicação servidora.

Para o desenvolvimento de uma aplicação cliente, foi escolhida uma aplicação *web* devido a esta ser compatível com uma maior gama de dispositivos em comparação com outras alternativas, tais como aplicação *desktop* ou móvel.

2.1 Aplicação Servidora

A aplicação servidora do i-on CodeGarten é uma aplicação que expõe uma *web* API. Esta API é o ponto de comunicação comum entre todos os clientes, ou seja, todos os clientes (*web*, *mobile*...) comunicam com a mesma aplicação servidora. É aqui também onde é implementada toda a lógica do sistema, sendo que os clientes apenas têm de efetuar chamadas aos vários *endpoints* para aceder, modificar ou criar os recursos pretendidos. Esta aplicação é também responsável por efetuar toda a comunicação com a *Web* API disponibilizada pelo GitHub [4].

Para armazenar os dados desta aplicação, foi utilizada uma base de dados relacional PostgreSQL [5]. Foi escolhido este Sistema de Gestão de Base de Dados (SGBD) devido à experiência adquirida anteriormente com o mesmo e à facilidade da hospedagem na plataforma Heroku [6].

Esta aplicação foi desenvolvida utilizando a linguagem de programação Kotlin [7] juntamente com o módulo Spring MVC da *framework* Spring [8].

2.2 Aplicação Web

A aplicação *web* do i-on CodeGarten fornece uma interface gráfica para interagir com o sistema. Esta é um cliente da aplicação servidora, e tem como objetivo ser utilizada pelos utilizadores finais (alunos e docentes). Para esta aplicação ser reconhecida como um cliente da aplicação servidora, terá de se encontrar registada como tal na base de dados da mesma.

A aplicação *web* é uma *Multi-Page Application* (MPA) que contém algumas páginas dinâmicas, ou seja, a maioria das páginas da aplicação necessitam de ser carregadas completamente, enquanto algumas podem ser alteradas em tempo real através do uso de código JavaScript no navegador *web* do utilizador. Esta aplicação foi desenvolvida utilizando a linguagem de programação TypeScript [9] com o ambiente de execução Node.JS [10] e a *framework web* Express [11]. Para a criação de vistas, foi utilizado o sistema de *templating* Handlebars [12], juntamente com a *framework* CSS Bootstrap [13]. Para as páginas dinâmicas da aplicação, foi utilizado código JavaScript juntamente com o uso da biblioteca JQuery [14].

A figura 2.2 ilustra a arquitetura da aplicação *web* assim como a divisão dos seus componentes.

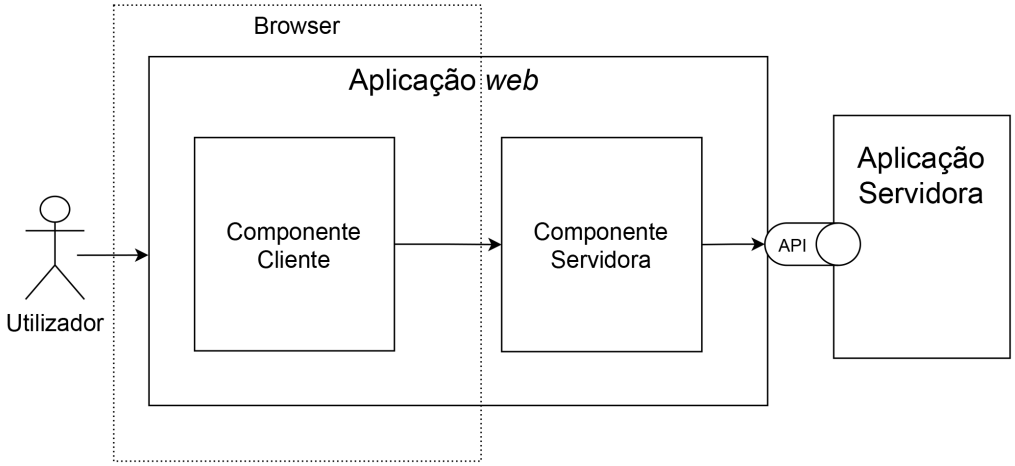

Figura 2.2: Arquitetura da aplicação *web*

A aplicação *web* é composta por duas componentes. Uma componente servidora e uma componente cliente.

A componente cliente da aplicação *web* é responsável por alterar o aspeto da página atualmente carregada no *browser* do utilizador. Esta componente é também responsável por reagir a algumas das ações que o utilizador efetua nas páginas, através de comunicação com a componente servidora da aplicação, nunca comunicando diretamente com a aplicação servidora do sistema.

A componente servidora da aplicação *web* é responsável por efetuar toda a comunicação com a aplicação servidora do sistema i-on CodeGarten. Também é responsável por efetuar a renderização das páginas a enviar ao cliente, assim como o armazenamento das sessões dos utilizadores.

Foi tomada a abordagem de construir uma aplicação *web* com esta estrutura devido à familiaridade com este método de desenvolvimento, o que aumentou a produtividade tendo em conta o tempo disponível para desenvolver a aplicação. Embora fosse possível construir esta aplicação de maneira a ser uma *Single-Page Application*, tornando a aplicação mais fluída e agradável para o utilizador, a falta de conhecimento nesta área não permitiria que a mesma fosse construída com o aspeto e funcionalidades pretendidas no espaço de tempo disponível. Também seria possível utilizar outras tecnologias como Kotlin e Spring MVC de maneira a reutilizar componentes desenvolvidos para a aplicação servidora, mas optou-se por utilizar tecnologias aprofundadas no desenvolvimento de outras aplicações deste tipo de maneira a agilizar a realização desta aplicação.

2.3 Modelo da API

A API disponibilizada pela aplicação servidora do sistema i-on CodeGarten responde aos seus clientes enviando recursos que que seguem o formato *hypermedia* Siren [15]. Esta decisão foi feita de maneira aos utilizadores da API conhecerem tanto as ações possíveis de realizar num determinado recurso, assim como as ligações que existem entre os mesmos. Também foram definidas classes Siren para os diversos tipos de recursos, sendo que cada classe contém o seu conjunto de propriedades.

A figura 2.3 mostra um exemplo de uma representação de um recurso da API definido no formato *hypermedia* Siren.

Figura 2.3: Exemplo de uma representação de um recurso no formato *hypermedia* Siren

A propriedade `class` define o tipo do recurso atual. No contexto da API do sistema i-on CodeGarten, existem classes para os diversos tipos de recursos, como uma *classroom*, um *assignment*, uma entrega ou um utilizador. Estes recursos serão descritos mais em detalhe na secção 3.1.1. A propriedade `properties` contém as propriedades relativas ao recurso. `Actions` é onde se encontram as ações que são possíveis de realizar sobre o recurso, assim como informações relativas às mesmas, que especificam a forma de estas serem executadas. A propriedade `links` enumera as relações presentes no recurso para outros recursos.

2.4 GitHub

A plataforma GitHub fornece um serviço de gestão de repositórios Git, que armazenam código de um determinado projeto. Esta plataforma também fornece o que é denominado de organizações GitHub, que juntam vários utilizadores num local com repositórios partilhados na organização, assim como privados a certos utilizadores. Estes utilizadores poderão ser membros da organização, assim como administradores da mesma.

Em contexto académico, para cada unidade curricular (UC) pode existir uma organização, pois estas agregam vários utilizadores, sendo que os seus repositórios podem representar um trabalho prático dessa mesma UC. Este é o modelo utilizado atualmente em algumas UC do ISEL, onde os docentes tipicamente são administradores das organizações.

Para possibilitar o acesso aos seus dados, a plataforma GitHub também disponibiliza uma API HTTP. Esta API pode ser acedida sem efetuar qualquer autenticação, sendo que algumas

ações, como a criação de repositórios ou visualização de informações privadas, necessita de autenticação por parte de quem efetua o acesso. As respostas desta API estão definidas seguindo o formato JSON.

De maneira a possibilitar certas interações do sistema i-on CodeGarten com a plataforma GitHub, é necessário registar este sistema na forma de uma aplicação no domínio do GitHub. Esta aplicação poderá ser uma GitHub App ou uma OAuth App, e permite comunicar com a API HTTP disponibilizada pelo serviço. É importante referir que esta App não é uma aplicação *desktop* que necessita de instalação, mas sim uma forma de associar uma aplicação a algo no domínio do GitHub.

Uma OAuth App realiza ações no GitHub em nome de um dado utilizador. Isto implica que uma OAuth App esteja limitada às ações que um utilizador é capaz de realizar, ou seja, terá as mesmas permissões. Por exemplo, caso o utilizador não seja administrador de uma organização GitHub, a OAuth App não consegue realizar ações que necessitem de permissões elevadas (por exemplo criação de repositórios ou criação de convites).

Uma GitHub App realiza ações no GitHub em seu próprio nome. Ao contrário de uma OAuth App, uma GitHub App não está limitada às permissões de um utilizador e tem as suas próprias permissões. Para possibilitar a realização de ações numa organização, é necessário efetuar uma instalação na organização correspondente [16]. Esta instalação é apenas o ato de autorizar o acesso da GitHub App a informações da organização, assim como permitir que a GitHub App efetue ações no domínio da organização alvo. É também necessário registar os identificadores das instalações da GitHub App. Este registo terá de ser feito no sistema i-on CodeGarten, o que implica a existência de um *endpoint* que receberá estes identificadores. Além disto, será necessário que o modelo de dados suporte a existência desta informação.

Para o sistema i-on CodeGarten, optou-se pelo uso de uma GitHub App em relação a uma OAuth App. Esta escolha foi tomada devido a uma GitHub App permitir que as ações sejam feitas em nome da aplicação e não do utilizador autenticado, assim como realizar ações numa organização, mesmo que o utilizador não seja dono desta. O uso de uma GitHub App também é recomendado pelo GitHub [17].

Capítulo 3

Aspetos de Implementação

Neste capítulo serão mencionados os aspetos de implementação relevantes das aplicações desenvolvidas neste projeto.

3.1 Aplicação Servidora

3.1.1 Modelo de Dados

Para armazenar os dados utilizados pela aplicação servidora de maneira eficiente e organizada, foi necessário definir um modelo de dados para utilizar na base de dados relacional. Para tal, foram definidas entidades que irão corresponder aos vários tipos de dados a armazenar. Na figura 3.1 encontra-se um diagrama que ilustra o modelo de dados implementado.

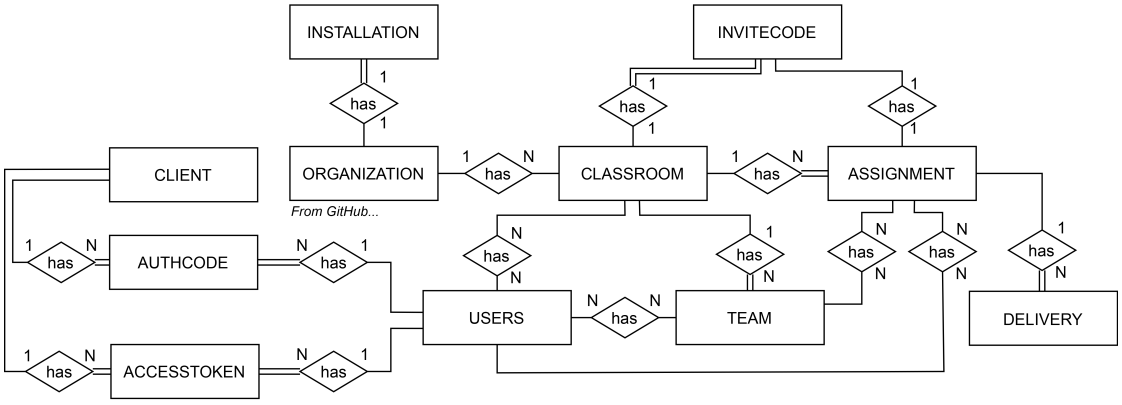

Figura 3.1: Diagrama do modelo de dados da aplicação servidora

A figura 3.2 complementa a figura 3.1, relacionando os dados armazenados no GitHub com os dados armazenados no i-on CodeGarten.

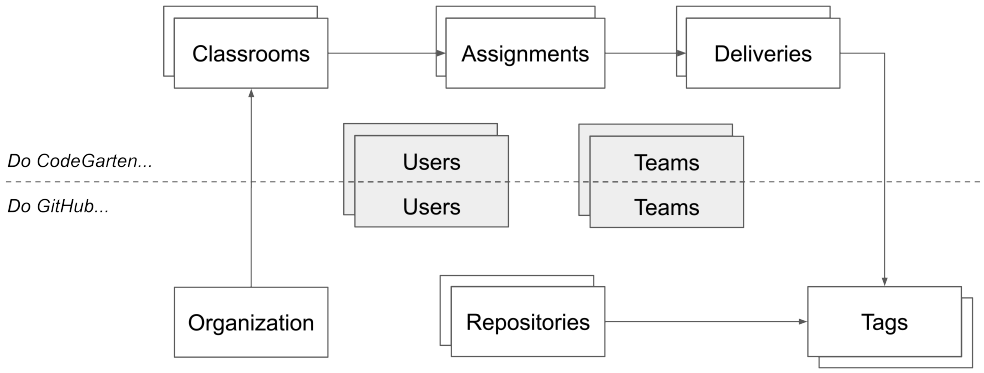

Figura 3.2: Diagrama da relação entre os dados do GitHub e do i-on CodeGarten

Foram primeiramente definidas as entidades relativas aos dados principais da aplicação, ou seja, entidades que representam os utilizadores, turmas (*classrooms*), trabalhos (*assignments*), entregas (*deliveries*) e equipas (*teams*). Foi necessário definir as relações entre entidades como a presença de utilizadores nas *classrooms*, *assignments*, *teams*, assim como a presença destas últimas nos vários *assignments*.

Foram também definidas entidades para informações de autenticação para a API, como dados de clientes da API, códigos de autorização e *tokens* de acesso, estando estes dois últimos associados a um utilizador e a um cliente. Também foram armazenadas informações relativamente a instalações da GitHub App em organizações (mais informações na secção 3.1.3), assim como os códigos de convite para *classrooms* e *assignments*.

Com este modelo estabelecido, foram definidas as tabelas SQL que refletem o mesmo. Para além disso foram utilizadas outras ferramentas da linguagem SQL como *triggers* e *functions* para assegurar as regras de negócio.

3.1.2 Modelação da API

Autenticação

A API desenvolvida necessitou de autenticação por parte do utilizador de maneira a identificar quem executa as ações que a mesma disponibiliza. Um dos principais desafios deste projeto foi desenvolver um método simples de utilizar e viável em termos de segurança. Com isto em mente, foi utilizado um sistema baseado no protocolo OAuth 2.0 seguindo o esquema *Authorization Code Grant* [18] com *tokens* do tipo *Bearer* [19].

Devido a algumas das rotas presentes no processo de autenticação necessitarem de comunicar com um serviço de autenticação externo (neste caso, do GitHub) através de redirecionamento do utilizador, foi criado um módulo da aplicação servidora ao qual foi designado de Interaction Manager (IM). Este módulo é responsável por conter todos os *endpoints* que resultarão no redirecionamento dos utilizadores dos clientes da API.

Na figura 3.3 encontra-se um diagrama que ilustra o processo de autenticação para possibilitar o acesso à API do i-on CodeGarten.

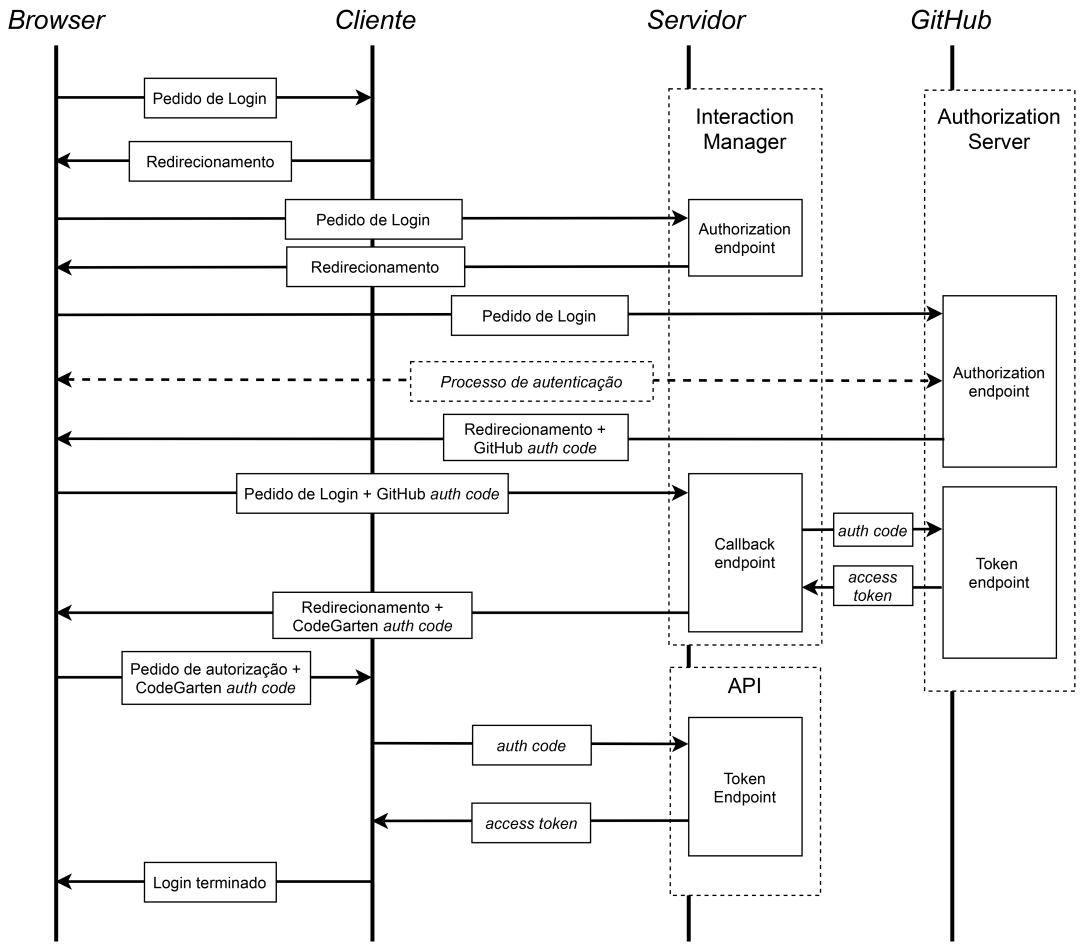

Figura 3.3: Diagrama do processo de autenticação

O processo de autenticação é iniciado através de um pedido a um *endpoint* da aplicação. Para efetuar este pedido, é necessário identificar um cliente válido através da definição de um parâmetro na *query string*. Esta informação é necessária devido a limitar o uso do código de autorização, que será gerado e dado ao cliente que efetuou o pedido de autenticação. De seguida, o utilizador necessitará de se autenticar no serviço de autenticação do GitHub e autorizar a obtenção de dados da conta deste serviço por parte do i-on CodeGarten. Caso aceite, o utilizador será encaminhado de volta ao IM, que irá obter um código de autorização referente à conta GitHub do utilizador. Este código será então utilizado para obter um *token* de acesso (*access token*) referente à conta GitHub. Depois da obtenção deste *token*, o IM responderá ao pedido inicial com um código de autorização da API do i-on CodeGarten. Por fim, este código poderá ser trocado por um *access token* da API, utilizando o *client secret* que se encontra definido para o cliente que efetuou o pedido de autenticação. Este *access token* tem um período de validade durante o qual pode ser utilizado, sendo necessário requisitar um novo aquando o término deste período.

Funcionalidades

Conforme descrito anteriormente, a aplicação servidora implementa a lógica do sistema, logo, a API que esta expõe terá de fornecer funcionalidades que permitam interagir com os recursos que esta possui. Com base nisto, foram definidos vários *endpoints* para a API que caracterizam os vários recursos. Para estes *endpoints*, foram também definidas ações que são

permitidas efetuar. Estas ações são executadas condicionalmente com base nas permissões que o utilizador possui. Estas permissões são dadas com base no papel que este desempenha dentro de uma organização do GitHub, assim como dentro de uma *classroom*, podendo este último ser:

- Aluno (*Student*);
- Professor (*Teacher*).

Os donos de organizações do GitHub são aqueles que são capazes de gerir *classrooms* numa organização. Isto significa que estes podem criar novas *classrooms*, atribuindo-lhes automaticamente o papel de professor nas mesmas.

Os professores são aqueles que podem gerir *assignments*, *deliveries* e utilizadores (alunos ou outros professores) dentro de uma *classroom*. Estes também são capazes de visualizar as entregas dos vários alunos/grupos presentes num determinado *assignment*, verificando se estes já efetuaram cada uma das entregas planeadas para o mesmo. Os alunos são capazes de entrar em *assignments* e de submeter *deliveries* nos mesmos. Não têm permissões de gerir nem a *classroom* nem os *assignments* nela presentes.

Em seguida encontra-se enumerado o conjunto de funcionalidades que a API dispõe. Entenda-se por "gestão" a possibilidade de criação, edição e remoção de um recurso:

- Listagem de organizações GitHub disponíveis ao utilizador no domínio do sistema;
- Edição e remoção do utilizador autenticado;
- Obtenção de dados relativos de um utilizador ou equipa;
- Listagem e gestão de utilizadores presentes numa equipa;
- Listagem e gestão de *classrooms*;
- Listagem e gestão de utilizadores presentes numa *classroom*;
- Listagem e gestão de equipas presentes numa *classroom*;
- Listagem e gestão de *assignments*;
- Listagem e gestão de participantes presentes num *assignment*;
- Listagem e gestão de *deliveries* num *assignment*;
- Obtenção de dados relativos a um convite, mesmo sem o aceitar;
- Aceitação de convites para um *assignment* ou *classroom*;
- Criação automática de repositórios GitHub aquando a aceitação de convites para *assignments*;
- Criação automática de convites para organizações GitHub aquando a aceitação de convites para *classrooms*;

A especificação dos *endpoints* disponibilizados pela API encontra-se descrita no repositório GitHub do projeto [20].

Pedidos e Respostas

Todos os pedidos são feitos à aplicação servidora e que necessitam de corpo terão de o incluir no formato JSON. Foi escolhido este formato devido à conveniência do mesmo e ao suporte nativo que o Spring oferece.

Todas as respostas da aplicação servidora seguem a especificação *hypermedia* Siren. Esta escolha foi feita de maneira a que a API ofereça capacidades *hypermedia* aos clientes, possibilitando que os mesmos naveguem pelos recursos através das ligações presentes na resposta, assim como conhecer as ações que podem ser realizadas num determinado recurso e as suas especificações. Qualquer resposta a pedidos que resultem num erro encontra-se no formato `application/problem+json` [21], que apresenta informações sobre o erro que ocorreu, juntamente com um identificador do mesmo.

3.1.3 Comunicação com o GitHub

A aplicação necessita de comunicar com o GitHub de maneira a efetuar algumas das suas ações, como por exemplo a criação de repositórios e consulta de dados dos utilizadores. Para isto, foi utilizada a API HTTP disponibilizada pelo GitHub.

Características da API

A API HTTP do GitHub permite que uma dada aplicação ou utilizador possa efetuar ações no seu domínio através do chamamento a *endpoints* definidos pela mesma. Embora alguns recursos não necessitarem de autenticação por parte do utilizador, é recomendada a presença da mesma devido aos limites impostos por parte do GitHub a acessos não autenticados a recursos da API [22].

No caso de acessos à API do GitHub por parte de aplicações, é necessário a existência de uma App no domínio do GitHub. Pode-se optar por duas variedades de Apps, sendo estas uma GitHub App ou uma OAuth App. No caso do sistema desenvolvido, optou-se pela criação de uma GitHub App devido a esta agir em seu nome, sendo que uma OAuth App age em nome do utilizador autenticado. O uso de uma GitHub App também é recomendado pelo GitHub, e permite executar ações numa organização mesmo quando o utilizador não é dono da mesma.

Autenticação

A aplicação servidora utiliza contas GitHub para autenticar os seus utilizadores. A GitHub App acima mencionada é utilizada para este efeito, sendo possível obter *tokens* de acesso relacionados com as contas GitHub através da mesma, desde que os utilizadores estejam de acordo com as permissões requisitadas.

Obtenção de Dados

Para a obtenção de dados provenientes do GitHub, foi utilizada a API HTTP disponibilizada pelo mesmo.

De maneira a possibilitar a obtenção de dados relativos a uma organização GitHub, é necessário que a GitHub App do sistema i-on CodeGarten se encontre instalada na mesma. Com isto, é possível obter um *token* de acesso da instalação, que será utilizado em acessos a recursos que necessitam do mesmo, como por exemplo a criação de repositórios, equipas e convites para a organização. A instalação também permite que o *token* de acesso do

utilizador que é obtido pela GitHub App possa ser utilizado no contexto da organização a que se pretende aceder, desde que este utilizador se encontre presente dentro da mesma.

Para a obtenção de dados relativos ao utilizador em si, é utilizado o seu *token* de acesso que é obtido pela GitHub App no processo de autenticação. É utilizado o *token* do utilizador na maior parte dos casos devido aos limites impostos pelo GitHub em termos de acessos na sua API. Como estas limitações são impostas ao nível do *token* utilizado e não ao nível da fonte de onde os pedidos são provenientes, a utilização do *token* do utilizador aumenta o número de pedidos que a aplicação pode realizar ao GitHub por hora.

A aplicação servidora conta ainda com a presença de uma *cache* para reduzir o número de pedidos repetidos que poderão eventualmente ser realizados ao GitHub. O uso de uma *cache* também resulta numa diminuição dos tempos de resposta da API, o que é importante devido a estes pedidos serem os mais demorados no contexto do sistema.

3.1.4 Estrutura do Código

De maneira a facilitar a leitura do código desenvolvido, é importante que este esteja organizado e consistente. Para a aplicação servidora, este foi dividido em várias *packages*, cada uma com o seu propósito.

- Raiz do projeto: Ficheiros relacionados com a totalidade do sistema. É aqui onde se encontra o *entrypoint* da aplicação, o agendamento de tarefas e as rotas da API;
- *Package* `auth`: Ficheiros relacionados com a autenticação de utilizadores. Encontra-se aqui o processo de validação do *header* que contém o *token* de acesso da API;
- *Package* `controllers`: Ficheiros relacionados com o tratamento dos pedidos realizados à aplicação servidora. Aqui encontram-se definidos os processos a realizar, assim como a modelação dos corpos das respostas e dos pedidos recebidos;
- *Package* `database`: Ficheiros relacionados com o acesso à base de dados relacional. Nesta *package* estão presentes os ficheiros que contém as *strings* das *queries* a realizar, assim como todo o código que efetua o mapeamento das respostas da base de dados para objetos do domínio de aplicação, também conhecidos por *Data Transfer Objects* (DTO);
- *Package* `exceptions`: Ficheiros que definem exceções específicas ao domínio do sistema. Todas as exceções que foram definidas durante o desenvolvimento da aplicação encontram-se aqui;
- *Package* `pipeline`: Ficheiros relacionados com a *pipeline* dos pedidos no contexto do Spring. Encontram-se definidos aqui os processos relacionados com interceção de pedidos para fins de autenticação, *logging* de pedidos, tratamento de exceções e resolução de argumentos;
- *Package* `remote`: Ficheiros relativos a acessos exteriores ao sistema. Encontram-se aqui definidos os processos para efetuar pedidos à API do GitHub, assim como o sistema de *cache* implementado para os mesmos;
- *Package* `responses`: Ficheiros relativos a respostas da API. Aqui definem-se os formatos de resposta que a API utiliza para comunicar com os seus clientes;
- *Package* `utils`: Ficheiros auxiliares. Definem-se aqui utilitários para auxiliar o sistema em vários contextos, como por exemplo no contexto de encriptação de dados.

Para auxiliar os pedidos efetuados à base de dados relacional, foi utilizada a biblioteca JDBI [23] que simplifica a definição dos mesmos, assim como o mapeamento das respostas devido ao seu suporte a *data classes* do Kotlin.

Para a realização de pedidos HTTP, foi utilizada a biblioteca OkHttp [24] devido à sua facilidade de utilização e a não ser necessário o uso de clientes HTTP assíncronos.

3.1.5 Gestão de Dados Secretos

Para possibilitar o acesso à API do GitHub, é necessário que a aplicação servidora conheça os dados relativos à GitHub App que esta irá utilizar. É necessário fornecer os seguintes dados:

- Nome da GitHub App;
- Identificador (ID) da GitHub App;
- Identificador de Cliente (*Client* ID) da GitHub App;
- *Client Secret* da GitHub App;
- Chave privada da GitHub App.

De maneira a facilitar a definição destes dados, foi decidido que estes serão fornecidos à aplicação servidora através de variáveis de ambiente. Esta decisão foi tomada de maneira a facilitar a hospedagem da mesma através de contentores Docker [25], mais especificamente na plataforma Heroku, visto que esta não suporta volumes Docker [26].

Os quatro primeiros dados referidos acima são definidos em formato JSON, sendo que este será posteriormente codificado em Base64 na variável de ambiente. A chave privada da GitHub App também será fornecida em Base64, sendo o seu conteúdo o conteúdo do ficheiro PEM obtido através da aplicação *web* do GitHub.

A aplicação servidora também necessita de conhecer os *tokens* de acesso dos utilizadores e de instalações para aceder aos seus dados. Devido a estes serem dados sensíveis, é necessário que estejam protegidos de alguma forma. A opção tomada para proteger estes dados foi de encriptá-los. Utilizou-se a cifra *Advanced Encryption Standard* (AES) [27], sendo que a chave utilizada para a mesma é fornecida num ficheiro de texto, de forma a facilitar a sua alteração.

Os *tokens* de acesso à API do sistema assim como os *client secrets* dos clientes da mesma, devido a serem informações sensíveis, não podem ser armazenados em claro na base de dados. Devido a isto, foi decidido utilizar a função de *hash* SHA-256 para os armazenar. Estes dados não são encriptados pois não é necessário obter a sua forma original, bastando apenas comparar o *hash* dos dados recebidos com aqueles que se encontram armazenados.

3.1.6 Testes da Aplicação

De maneira a assegurar o correto funcionamento do código desenvolvido para a aplicação servidora, foram desenvolvidos testes que verificam as funcionalidades implementadas.

Para testar as funcionalidades expostas pela API, foram desenvolvidos testes de integração. Foi tomada esta decisão devido à forma que o código encontra-se desenvolvido. Para além disto, os testes de integração permitem testar um cenário que se aproxima da realidade, em que um cliente interage com a API. Para os componentes independentes do sistema, foram desenvolvidos testes unitários.

A figura 3.4 descreve o modelo dos testes realizados para a aplicação servidora.

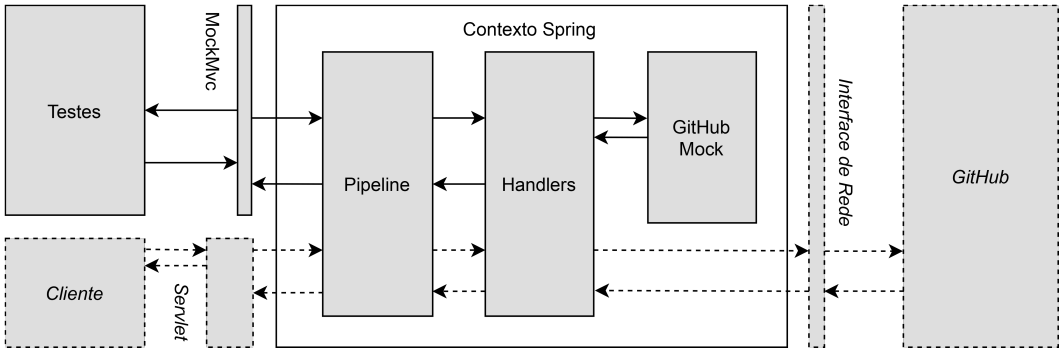

Figura 3.4: Diagrama do modelo de dados da aplicação servidora

Os testes de integração são realizados através do uso de um contexto Spring diferente do utilizado em produção, com configurações específicas para os testes, nomeadamente o uso de um simulador de respostas do GitHub. Este contexto será reutilizado entre testes. Para efetuar os pedidos à API, é utilizado uma instância de MockMvc [28] de maneira a estes não serem realizados na interface de rede. Desta forma, os testes realizados não efetuam pedidos HTTP à aplicação servidora. Estes pedidos utilizam as classes dos modelos das respostas e pedidos como DSL, e é sobre estas que são feitas as asserções necessárias.

Com o objetivo de evitar que os testes interfiram com a base de dados utilizada pela aplicação, foi utilizada uma base de dados exclusiva aos testes, isolando assim toda a interação dos testes com a base de dados.

De maneira a tornar os testes realizáveis num ambiente *offline*, foi necessário simular as respostas provenientes do GitHub. Para isto, foi realizada uma nova implementação da *interface* responsável pela comunicação com o GitHub, simulando as respostas necessárias, ou seja, não efetuando qualquer comunicação com o exterior.

3.2 Aplicação Web

3.2.1 Acesso à API

A aplicação *web* age como um cliente da API do sistema i-on CodeGarten. De maneira a tal ser possível, a aplicação servidora terá de reconhecer esta aplicação como um cliente, ou seja, a aplicação *web* terá de possuir um identificador de cliente para comunicar com a API. Este identificador será utilizado nos pedidos de autenticação de utilizadores.

A aplicação conhece previamente quais as localizações dos vários recursos que a API dispõe. Embora não seja a melhor abordagem, tendo em conta que seria possível utilizar as localizações disponibilizadas nas respostas da API, foi tomada esta decisão devido a constrangimentos que impediram o melhoramento do projeto, nomeadamente o tempo disponível para o mesmo.

Todos os pedidos realizados à API são feitos na componente servidora da aplicação *web* e não na componente cliente. Ou seja, o código da aplicação presente no *browser* do utilizador não realiza pedidos diretamente à API do sistema.

3.2.2 Interação com Utilizadores

Um utilizador, para aceder às funcionalidades da aplicação, terá de se autenticar na API com a sua conta GitHub. Caso não o faça, apenas lhe é apresentada uma página inicial com informação sobre a aplicação.

A aplicação conta com a presença de uma barra de navegação que permite ao utilizador voltar à página inicial (listagem de organizações), assim como aceder à informação da sua conta e efetuar *log-out* da mesma. Na página relativa à sua conta, este pode alterar o nome que é utilizado na aplicação, mantendo o nome da sua conta GitHub. Isto permite que, em contexto académico, o utilizador tenha, por exemplo, o seu identificador de aluno no i-on CodeGarten, mantendo o seu nome original no GitHub.

Relativamente a uma *classroom*, o utilizador pode, na mesma página, visualizar os *assignments*, os utilizadores e *teams* pertencentes à mesma. Caso o utilizador seja um professor, este pode efetuar mudanças à *classroom*, ou seja, alterar o seu nome e descrição, assim como apagá-la da organização. Ainda na mesma página, é possível gerir os utilizadores da *classroom*, alterando os seus papéis (aluno ou professor) ou removendo-os da mesma. É ainda possível efetuar a gestão de equipas, ou seja, efetuar a criação e remoção destas, assim como a criação de *assignments*.

A página que representa um *assignment* é semelhante à que descreve uma *classroom*. Esta permite visualizar e gerir os participantes do mesmo, assim como efetuar o planeamento de entregas caso seja professor (um aluno apenas pode visualizar as suas próprias entregas). Um professor também pode visualizar as entregas dos participantes do *assignment*, verificar o estado das mesmas e aceder ao repositório GitHub correspondente a esse participante. À semelhança da página de um *classroom*, um professor pode alterar o nome e descrição do *assignment* e apagá-lo da *classroom*.

A maioria das páginas da aplicação são alteradas de acordo com as ações do utilizador sem efetuar o carregamento de uma nova página. Isto proporciona uma experiência mais fluída para o utilizador, tornando a navegação mais rápida entre certos conteúdos da aplicação.

Devido ao uso da *framework* CSS Bootstrap, a aplicação *web* é responsiva, ou seja, o aspeto da aplicação é ajustável à dimensão do ecrã do utilizador. Isto permite que a aplica-

ção seja utilizada não só em computadores pessoais, como também em dispositivos móveis, apresentando uma interface gráfica com funções e aspeto adequado a esse ecossistema.

3.2.3 Gestão de Sessões

De maneira a evitar o armazenamento de sessões do utilizador na componente servidora da aplicação *web*, as informações relevantes à sessão do mesmo serão armazenadas nas *cookies* do browser. Informações sensíveis tais como o *token* de acesso à API do sistema i-on Code-Garten encontram-se encriptadas utilizando uma cifra AES. A chave utilizada neste processo é fornecida através de uma variável de ambiente, de forma a facilitar a sua alteração.

A componente servidora da aplicação *web* não armazena qualquer tipo de dados relativos a sessões de utilizadores de maneira a possibilitar a execução várias instâncias desta aplicação em paralelo, ou seja, possibilitar a escalabilidade do sistema. Desta forma, um utilizador pode comunicar com qualquer instância da aplicação *web* e manter os dados relativos à sua sessão entre elas.

Durante o processamento de um pedido à componente servidor da aplicação *web*, esta verifica a validade do *token* de acesso à API, caso este exista. Esta verificação é feita tendo em conta uma margem de tempo adequada de maneira a evitar o uso de um *token* expirado em páginas dinâmicas da aplicação. Caso esta verificação conclua que o *token* não é válido, é automaticamente feito um pedido de *log-in*, de maneira a obter um novo *token* para aceder à API.

3.2.4 Estrutura do Código

À semelhança da aplicação servidora, o código desenvolvido para a aplicação *web* encontra-se organizado em diretorias que especificam o seu propósito, de maneira a facilitar a sua compreensão. Também existem dois tipos de código desenvolvido: código da componente servidora (a executar no servidor) e código da componente cliente (a executar no *browser* do utilizador).

O código correspondente à componente servidora da aplicação encontra-se localizado na diretoria `lib`, enquanto o código correspondente à componente cliente encontra-se na diretoria `public`, que contém os recursos estáticos disponibilizados pela componente servidora.

Para além do código desenvolvido, foram também definidas as vistas a serem renderizadas pela componente servidora. Estas vistas estão definidas em forma de *templates* Handlebars, e encontram-se na diretoria `views`. Estas vistas apresentam elementos que estão relacionados com as respostas da API, ou seja, as vistas dependem das ações que o utilizador pode executar, sendo a API responsável pela definição das mesmas.

Componente Servidora

- Diretoria raiz: Ficheiros relacionados com a totalidade do sistema. Aqui encontra-se o *entrypoint* da aplicação, assim como o código relativo à gestão de sessões de utilizadores;
- Diretoria `repo`: Ficheiros relativos ao acesso à API. Estão aqui definidas as rotas da API, assim como os processos de acesso a recursos da API;
- Diretoria `routes`: Ficheiros relacionados com o tratamento de pedidos à aplicação. O código que define as ações a serem executadas aquando a receção de um pedido são definidas aqui;

- Diretoria `types`: Ficheiros relacionados com a definição de tipos do TypeScript.

Componente Cliente

O código da componente cliente da aplicação *web* foi escrito em JavaScript, e encontra-se dividido em vários ficheiros, cada um responsável por ser executado numa determinada página. O código definido é responsável por definir ações a serem executadas em resposta a determinados eventos, como por exemplo da resposta à pressão de um botão. Estas ações poderão resultar na alteração da página que se encontra carregada no *browser* do utilizador, assim como no desencadeamento de um pedido à componente servidora da aplicação.

Capítulo 4

Conclusões e Trabalho Futuro

Este último capítulo apresenta os principais objetivos que foram atingidos assim como as dificuldades que foram encontradas. Também são dadas algumas indicações sobre como o sistema pode ser melhorado no futuro.

4.1 Conclusão

Os docentes das unidades curriculares do ISEL que utilizam o GitHub como meio de alojamento de repositórios juntamente com o sistema GitHub Classroom manifestam dificuldades com o uso desta abordagem. Após o desenvolvimento do sistema i-on CodeGarten, mais concretamente da aplicação servidora e de uma aplicação cliente, algumas destas dificuldades foram solucionadas, como por exemplo a gestão de entregas de trabalhos e a possibilidade de relacionar uma conta GitHub a um outro identificador (nome, número de aluno, entre outros...). Estas funcionalidades foram implementadas na aplicação servidora, sendo que a aplicação cliente desenvolvida apenas apresenta um meio de interação entre o utilizador e o sistema. Esta abordagem permite que, no futuro, possam ser desenvolvidas outras aplicações cliente com o mesmo propósito, mas que fornecem um outro meio de interação com o utilizador.

Os principais desafios durante o desenvolvimento deste sistema foram a aprendizagem da *framework* Spring, sendo que os conhecimentos relativos à mesma foram adquiridos em simultâneo com o seu uso, a modelagem do processo de autenticação, devido à sua complexidade, e a integração com a API do GitHub, devido à tomada de decisão de uma forma de comunicação com a mesma, assim como a existência de inconsistências na sua documentação.

4.2 Trabalho Futuro

Embora o sistema desenvolvido implemente todas as funcionalidades essenciais para o seu funcionamento, assim como algumas que melhoram a sua usabilidade, este pode ainda ser melhorado através da implementação de outras funcionalidades ou do melhoramento de alguns componentes:

- Isolamento da lógica da aplicação servidora;
- Realização de testes unitários e de integração na aplicação servidora;
- Permitir a realização de um *clone* de todos os repositórios de um dado *assignment*;

- Melhorar a sincronização do estado de utilizadores no CodeGarten e no GitHub;
- Implementar um mecanismo de *rate limiting* na aplicação servidora;
- Substituir a saída de participantes de *assignments* por um mecanismo de desistência;
- Melhorar o sistema de convites de maneira a possibilitar a vista de convites pendentes;
- Fornecer a capacidade de efetuar filtros na listagem de *assignments* e participantes;

Esta informação encontra-se presente mais detalhadamente nas *issues* do repositório GitHub relativo ao projeto [29].

Referências

[1] *Iniciativa i-on*. Disponível em https://github.com/i-on-project (acedido em 12/06/2021).

[2] *Github Classroom*. Disponível em https://classroom.github.com (acedido em 12/06/2021).

[3] *i-on CodeGarten GitHub repository*. Disponível em https://github.com/i-on-project/codegarten (acedido em 12/06/2021).

[4] *GitHub REST API*. Disponível em https://docs.github.com/en/rest (acedido em 12/06/2021).

[5] *PostgreSQL*. Disponível em https://www.postgresql.org (acedido em 12/06/2021).

[6] *Heroku*. Disponível em https://www.heroku.com (acedido em 12/06/2021).

[7] *Kotlin*. Disponível em https://kotlinlang.org (acedido em 12/06/2021).

[8] *Spring*. Disponível em https://spring.io (acedido em 12/06/2021).

[9] *TypeScript*. Disponível em https://www.typescriptlang.org (acedido em 12/06/2021).

[10] *Node.js*. Disponível em https://nodejs.org/en (acedido em 12/06/2021).

[11] *Express - Fast, unopinionated, minimalist web framework for Node.js*. Disponível em https://expressjs.com (acedido em 13/06/2021).

[12] *Handlebars*. Disponível em https://handlebarsjs.com (acedido em 12/06/2021).

[13] *Bootstrap*. Disponível em https://getbootstrap.com (acedido em 12/06/2021).

[14] *jQuery*. Disponível em https://jquery.com (acedido em 12/06/2021).

[15] *Siren: a hypermedia specification for representing entities*. Disponível em https://github.com/kevinswiber/siren (acedido em 12/06/2021).

[16] *Installing GitHub Apps*. Disponível em https://docs.github.com/en/developers/apps/managing-github-apps/installing-github-apps (acedido em 12/06/2021).

[17] *Migrating OAuth Apps to GitHub Apps*. Disponível em https://docs.github.com/en/developers/apps/getting-started-with-apps/migrating-oauth-apps-to-github-apps (acedido em 12/06/2021).

[18] *The OAuth 2.0 Authorization Framework*. Disponível em https://datatracker.ietf.org/doc/html/rfc6749 (acedido em 12/06/2021).

[19] *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. Disponível em https://datatracker.ietf.org/doc/html/rfc6750 (acedido em 12/06/2021).

[20] *i-on CodeGarten GitHub repository API documentation*. Disponível em https://github.com/i-on-project/codegarten/tree/main/docs/api (acedido em 12/06/2021).

[21] *Problem Details for HTTP APIs*. Disponível em `https://datatracker.ietf.org/doc/html/rfc7807` (acedido em 12/06/2021).

[22] *Rate limits for GitHub Apps*. Disponível em `https://docs.github.com/en/developers/apps/building-github-apps/rate-limits-for-github-apps` (acedido em 13/06/2021).

[23] *Jdbi 3 Developer Guide*. Disponível em `https://jdbi.org` (acedido em 13/06/2021).

[24] *OkHttp Overview*. Disponível em `https://square.github.io/okhttp` (acedido em 13/06/2021).

[25] *Docker*. Disponível em `https://www.docker.com/` (acedido em 04/07/2021).

[26] *Container Registry & Runtime (Docker Deploys) - Unsupported Dockerfile commands*. Disponível em `https://devcenter.heroku.com/articles/container-registry-and-runtime#unsupported-dockerfile-commands` (acedido em 12/07/2021).

[27] *Specification for the Advanced Encryption Standard (AES)*. Disponível em `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf` (acedido em 13/06/2021).

[28] *MockMvc*. Disponível em `https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/test/web/servlet/MockMvc.html` (acedido em 07/07/2021).

[29] *i-on CodeGarten GitHub repository*. Disponível em `https://github.com/i-on-project/codegarten/issues` (acedido em 10/07/2021).