I-on ClassCode

Licenciatura em Engenharia Informática e de Computadores

Relatório do projeto realizado no âmbito de Projeto e Seminário

André David dos Santos
A48309@alunos.isel.pt

Ricardo de Freitas Henriques
A48322@alunos.isel.pt

João Diogo da Anunciação Magalhães
A48348@alunos.isel.pt

Orientador:
Pedro Miguel Henriques Santos Félix
pedro.felix@isel.pt

10 de Julho de 2023

Resumo

O projeto **i-on ClassCode** tem como objetivo o desenvolvimento de um sistema para a criação e gestão de repositórios **GitHub** em contexto académico. No sistema existem duas aplicações, **Web** e **Mobile**, que interagem com um **Serviço Backend** que fornece uma **API HTTP**. As aplicações implementam funcionalidades como a criação de repositórios **GitHub** para diferentes turmas, a sua associação a organizações **GitHub** representantes de uma unidade curricular e a criação de grupos de trabalho, entregas e tarefas. Os principais objetivos do sistema são: a facilitação dos processos de criação e configuração de novos repositórios para grupos de alunos, a identificação institucional de alunos e o processo de *cloning* associado à avaliação desses alunos.

O projeto desenvolvido aborda temas relevantes como a autenticação de utilizadores através da *framework OAuth 2.0* e do **GitHub**, a preocupação do armazenamento de *tokens* de acesso capazes de ações destrutivas fora de componentes pertencentes ao sistema e sincronização entre duas fontes de informação, **GitHub** e o **Serviço Backend** do **i-on ClassCode**. Ainda se acrescentam aspetos como a definição de um modelo de dados, a garantia da integridade do sistema como um todo, o cumprimento de inúmeras restrições associadas à lógica de negócio e os conhecimentos necessários ao desenvolvimento de *Single Page Applications* em execução no ambiente *browser* e aplicações para ambiente *Android*.

Como maiores dificuldades é possível referir a complexidade associada ao modo como as aplicações **Web** e **Mobile** interpretam e reagem a respostas HTTP com diferentes códigos de resposta, o processo de autenticação a partir de serviços externos e a garantia da coerência e da correção no uso das tecnologias em todo o projeto.

Palavras-Chave: i-on ClassCode, GitHub, Serviço Backend, API HTTP, Web, Mobile, Tokens de acesso, Browser, Android

Abstract

The **i-on ClassCode** project aims to develop a system for creating and managing **GitHub** repositories in an academic context, providing a set of tools for both students and teachers to increase productivity and convenience. The system consists of two applications, **Web** and **Mobile**, which interact with a **Backend Service** that provides an **HTTP API**. The applications implement functionalities such as: creating **GitHub** repositories for different classes, associating them with **GitHub** organizations representing a course, and creating workgroups, assignments, and tasks. The main objectives of the system are to facilitate the process of creating and configuring new repositories for student groups, institutional identification of students, and the cloning process associated with evaluating these students.

The developed project addresses relevant topics such as user authentication through the OAuth 2.0 framework and **GitHub**, the concern of storing access tokens capable of destructive actions outside of system components, and synchronization between two sources of information, **GitHub** and the **i-on ClassCode Backend Service**. Additionally, aspects such as the definition of an data model, ensuring system integrity as a whole, compliance with numerous business logic restrictions, and the necessary knowledge for developing Single Page Applications running in the browser environment and applications for the Android environment are included.

The greatest difficulties can be attributed to the complexity associated with how the **Web** and **Mobile** applications interpret and react to HTTP responses with different response codes, the authentication process through external services, and ensuring coherence and correctness in the use of technologies throughout the project.

Keywords: i-on ClassCode, GitHub, Backend Service, HTTP API, Web , Mobile, Access Tokens, Browser, Android

Agradecimentos

Primeiramente agradeço aos meus colegas, Ricardo Henriques e João Magalhães, toda a colaboração e compreensão na elaboração do projeto. Agradeço também aos meus pais e irmão que me incentivaram nos momentos difíceis e compreenderam a minha ausência enquanto me dedicava à realização deste trabalho. Quero agradecer ao professor e orientador do projeto, Pedro Félix, pela proposta do tema do projeto e pela constante disponibilidade e ajuda durante o seu desenvolvimento.

Lisboa, Julho de 2022 André Santos

Em primeiro lugar, quero agradecer aos meus colegas e amigos, André Santos e João Magalhães, pelo apoio ao longo destes 3 anos de faculdade mas principalmente durante este projeto. Quer nos debates de ideias conjuntas, à entreajuda e o apoio que só foi possível com estes 2 colegas.

Depois agradecer aos professores do ISEL que fizeram parte deste caminho, com um apreço em particular ao professor Pedro Félix, que desde início se mostrou disponível e deu a ideia deste projeto que nos conquistou logo. Que durante este projeto foi uma peça crucial no meu crescimento e conhecimento nesta reta final.

Em último lugar, quero deixar uma nota à minha família e namorada, por reconhecerem o tempo investido, pela coragem dada e reconhecimento das minhas capacidades para tal, foram uma das peças cruciais.

Lisboa, Julho de 2022 Ricardo Henriques

Este trabalho só foi conseguido devido ao trabalho de equipa e camaradagem. Foi um projeto que todos nó tiveram a sua contribuição, cuidado e esforço. Por isso, um grande obrigado ao Ricardo Henriques e ao André Santos, por terem demonstrado o poder de ter uma boa equipa.

Foi no ISEL onde aprendemos todo o conhecimento aplicado neste trabalho. E todo esse conhecimento foi transmitido por professores a quem agradeço bastante. Um agradecimento também a todos os amigos que fiz neste percurso, que ajudaram, falaram e principalmente enriqueceram esta experiência académica. Um agradecimento especial ao professor Pedro Félix devido ao seu tempo investido, trabalho e preocupação que teve com o nosso grupo ao longo deste projeto.

Agradecimentos especiais à minha família, namorada e amigos próximos que ao longo destes anos, por mais que não me pudessem ajudar com a parte técnica, davam-me confiança e um conforto nos momentos mais difíceis.

Lisboa, Julho de 2022 João Magalhães

Lista de Figuras

Índice

1 Introdução

O projeto **i-on ClassCode**[1] tem como objetivo o desenvolvimento de um sistema para a criação e gestão de repositórios **GitHub**[2], para uso em contexto académico, de forma a melhorar a experiência de utilização e o aumento de produtividade para professores e alunos.

O projeto **i-on ClassCode** aplica a experiência de dois projetos anteriores, **i-on CodeGarten**[3] e **Teams**[4], através da melhoria de lacunas da **Aplicação Web**, bem como o desenvolvimento de uma **Aplicação Mobile**, como ferramenta pessoal do professor. O projeto tem como foco ainda a melhoria da segurança no uso dos dados sensíveis dos utilizadores.

1.1 Contexto

O **GitHub** é uma plataforma *web* usada para controle de versões e colaboração em projetos de desenvolvimento de *software*. A ferramenta **GitHub Classroom**[5], criada pelo **GitHub**, promove a criação e gestão de grupos de tarefas, que tem como objetivo facilitar e automatizar o processo de criação de repositórios. É usada em contexto académico para a criação de repositórios associados a trabalhos individuais ou a grupos de alunos.

Em ambiente académico, há professores que usam os repositórios **GitHub** como espaço para os alunos entregarem e guardarem as suas cópias locais dos trabalhos que tem ao seu dispor um mecanismo de controlo de versões. Os repositórios fornecem ainda a possibilidade de visualizar o progresso, as contribuições de cada aluno e entregas através do sistema de criação de *tags*.

O **GitHub Classroom** é uma ferramenta com a função de estabelecer os grupos de alunos através da criação de *teams* e associar esses grupos a um repositório de trabalho. É também usada como forma de identificar alunos e fornece capacidade de *feedback* entre o professor e os grupos.

1.2 Problema

O **GitHub Classroom** apesar de ser útil tem as suas limitações. O processo de criação de novos repositórios e equipas para os vários grupos de alunos é, além de demorado, um processo repetitivo no início de um novo semestre letivo. Outros dos problemas do **GitHub Classroom** proveem da identificação institucional do aluno através da conta **GitHub** e do processo de *cloning* individual e lento dos repositórios para a máquina local do professor durante o processo de avaliação. A relação entre os alunos e os professores não é explorada da melhor forma, sendo bastante reduzida a existência de *feedback* entre o professor e os grupos de trabalho.

1.3 Solução

O **i-on ClassCode** fornece um conjunto de funcionalidades através das aplicações **Web** e **Mobile** que visam solucionar os problemas presentes no uso académico da ferramenta **GitHub Classroom**. A **Aplicação Web** é usada tanto por professores como por alunos e por isso as funcionalidades diferem entre os dois tipos de utilizador. As funcionalidades para os alunos são:

- Entrar num curso.
- Entrar numa turma.
- Verificar se tem trabalhos, *assignments*, pendentes.
- Fechar a equipa quando o grupo não tem o número máximo de elementos, mas já está completo.
- Sair de uma equipa.
- Sair de uma turma.
- Sair de um curso.

As funcionalidades para os professores são:

- Criar um curso.
- Criar uma turma.
- Criar *assignments*.
- Criar entregas, *deliveries*, onde indica o *template* da *tag* do **GitHub** e a data limite de entrega.
- Observar facilmente quais foram as equipas que entregaram ou não.
- Dar feedback às equipas.
- Fazer uma cópia local de todos os repositórios da turma, através de um bash script.
- Arquivar uma turma.
- Pedir para sincronizar a informação do **i-on ClassCode** e o **GitHub**.
- Arquivar um curso.
- Aceitar, rejeitar ou ignorar a entrada de novos professores na aplicação.

A **Aplicação Mobile** permite ao utilizador, neste caso um professor, realizar operações que envolvem escritas na API do **GitHub**. Estas funcionalidades são:

- Aceitar ou rejeitar um pedido de criar equipa.
- Aceitar ou rejeitar um pedido para um aluno entrar numa equipa.
- Aceitar ou rejeitar um pedido para um aluno sair de uma equipa. Se este for o último a equipa é apagada.
- Quando um aluno pede para sair do curso, este pedido é aceite logo quando o professor entra na aba do curso.
- Os pedidos para arquivar um repositório são logo aceites, quando se entra num curso.

Todas as funcionalidades indicadas, possuem um manual de instruções, onde se encontra em pormenor como são realizadas, o manual pode ser encontrado no repositório do projeto **GitHub** (`https://github.com/i-on-project/repohouse`).

O **i-on ClassCode** apresenta um **Serviço Backend** que expõe uma **API HTTP** que será usada por duas aplicações, sendo uma a **Aplicação Web** e a outra a **Aplicação Mobile**, de modo a fornecer suporte para os problemas existentes. O sistema é responsável por aceder e manipular uma **Base de Dados** com finalidade de gerir e guardar os dados do sistema. De forma a aceder a dados e operações externas relacionadas com o **GitHub**, o serviço acede à **GitHub API**[6]. A existência de duas aplicações distintas é devida ao facto da **Aplicação Mobile** ser de uso restrito aos professores, onde os mesmo poderão realizar ações que resultem em escritas na **GitHub API** envolvendo o seu *token* pessoal, garantido assim segurança do mesmo.

1.4 Organização do Documento

A arquitetura do sistema, descrição e função dos seus componentes bem como as tecnologias utilizadas encontram-se no capítulo 2. Os detalhes de implementação de cada um dos componentes e o processo de autenticação completo em ambas as aplicações **Web** e **Mobile** estão presentes no capítulo 3. O capítulo 4 destina-se à explicação do processo de disponibilização do sistema.

2 Arquitetura e Organização

Este capítulo tem como objetivo fornecer uma visão geral da estrutura do sistema e como as diferentes partes estão interligadas. Embora não sejam abordados detalhes específicos de implementação, este capítulo fornece uma base sólida para entender o funcionamento geral do projeto.

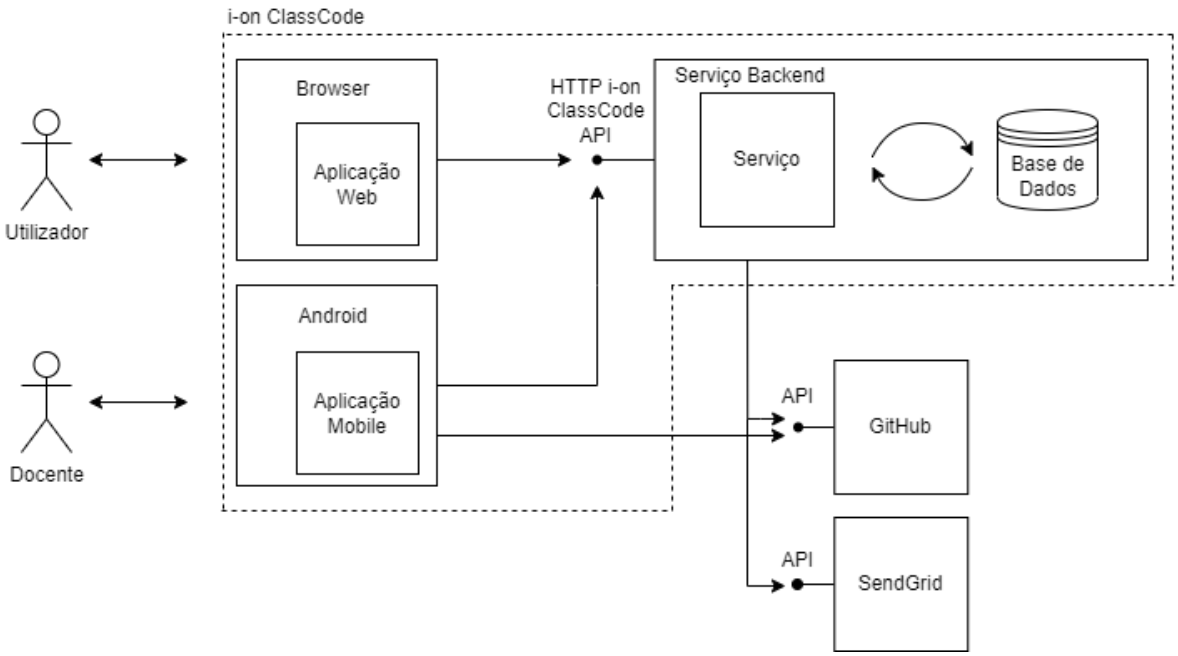

Figura 1: Arquitetura Geral do Sistema

Tal como ilustrado na figura 10, o sistema **i-on ClassCode** está dividido em duas aplicações, a **Aplicação Web** e a **Aplicação Mobile**, e o **Serviço Backend** que expõe uma **API HTTP** e que acede uma base de dados. Ambas as aplicações comunicam diretamente com a **API HTTP**. O sistema comunica ainda com duas API´s externas, a **GitHub API** e a **SendGrid API**[7]. Os componentes do sistema que comunicam diretamente com a **GitHub API** são a **API HTTP** e a **Aplicação Mobile**. Por outro lado, apenas a **API HTTP** comunica com a **SendGrid API**.

O **Serviço Backend** do **i-on ClassCode** é responsável por garantir a integridade do sistema, fazendo cumprir todas as restrições do mesmo.

Para manter a modularidade e a escalabilidade do projeto, a arquitetura foi pensada de forma a separar as responsabilidades de cada parte do sistema.

2.1 Serviço Backend

O **Serviço Backend** usado pelo sistema, foi desenvolvido com recurso à *framework Spring Boot*[8] e à linguagem de programação *Kotlin*[9]. O *Spring Boot* é uma ferramenta de código aberta que facilita a criação de micro-serviços e aplicações *web*. A comunicação com a **GitHub API** foi feita utilizando a biblioteca *OkHttp*[10] e a sua API síncrona. A desserialização das respostas HTTP foi realizada a partir da biblioteca *Jackson*[11], conhecida como a biblioteca JSON[12] para *Java*[13].

A **API HTTP** do **Serviço Backend** desenvolvida, é também uma API *Hypermedia* de forma a tornar o sistema mais flexível e passível a mudanças. Assim, fornece ao mesmo tempo informações sobre como realizar pedidos HTTP, normalmente, relacionados com o recurso em contexto. Com o objetivo de seguir um modelo simples e bem estruturado para a representação de respostas HTTP, seguiu-se a especificação *Hypermedia*, *Siren*[14]. Esta especificação permite a representação de operações da **API HTTP** e ajuda a promover a descoberta e uso de recursos *web* por parte da **Aplicação Web**.

A comunicação do serviço com a **Base de Dados** relacional PostgreSQL[15] do sistema foi concretizada com o uso da biblioteca JDBI[16] que permite através de um conjunto de classes e interfaces *Java*, o envio de instruções SQL para qualquer base de dados relacional que tenha drivers JDBC [17].

Todas as tecnologias usadas no **Serviço Backend** proveem do conhecimento adquirido ao longo do curso, que tem como propósito principal de escolha a capacidade a aprimorar a utilização das tecnologias além do reforço do estudo das mesmas.

O desenho do **Serviço Backend** segue também as boas práticas de implementação como a utilização de URL´s descritivos e códigos de resposta HTTP expressivos.

2.2 Base de Dados

A base de dados do **i-on ClassCode** é uma base de dados relacional PostgreSQL e é responsável por armazenar todos os dados relativos ao sistema e manter o seu estado.

Qualquer acesso feito à **Base de Dados** é sempre realizado a partir do **Serviço Backend** e nunca diretamente. Assim qualquer ação realizada por cada uma das aplicações do sistema que envolva a manipulação da **Base de Dados** é sempre feita via a **API HTTP** do **Serviço Backend**.

Na grande maioria dos casos a peça responsável por manter a integração do sistema é o **Serviço Backend**, porém em alguns casos específicos, a **Base de Dados** foi configurada para que após certas alterações dos dados por parte da **API**, esta automaticamente manipule os seus próprios dados. Desta forma durante o desenvolvimento, foi preciso ter um cuidado específico entre os possíveis acessos realizados à **Base de Dados** por parte da API e os *triggers* criados para manter também a integração do sistema.

As principais informações armazenadas são, por exemplo, os vários utilizadores registados, os pedidos realizados por alunos e todo o domínio de informação relacionado com o **GitHub**. O modelo de dados desenvolvido, representante da estrutura da **Base de Dados**, será apresentado na secção 3.1.4 com maior detalhe.

2.3 Aplicações

As duas aplicações apresentadas, a **Aplicação Web** e a **Aplicação Mobile**, têm como funcionalidade fornecer uma interface gráfica ao utilizador, que serve como ponte de comunicação entre o utilizador e o **Serviço Backend**.

2.3.1 Web

A **Aplicação Web** foi desenvolvida em *TypeScript*[18] com recurso à biblioteca *ReactJS*[19]. Esta é uma biblioteca *JavaScript*[20] com foco em criar interfaces de utilizador para páginas *web*.

Uma tecnologia usada apenas durante a fase de desenvolvimento da aplicação foi o *Webpack*[21]. O *Webpack* é um empacotador de módulos estáticos para aplicações *JavaScript* modernas. Inicialmente é criado internamente um grafo de dependências a partir dos vários módulos e em seguida, são combinados todos os módulos do projeto em um ou mais *bundles* que serão depois utilizados para entregar a aplicação ao browser em produção.

A aplicação é uma *Single-Page Application*[22] e por isso, consiste em um único ficheiro HTML que é carregado para o *browser* e depois realiza a manipulação dos elementos *DOM* para a exibição de diferentes conteúdos. Ao invés da navegação ser realizada por diferentes páginas, a interação é feita por componentes que são exibidos e ocultados dinamicamente na página, prevenindo assim a recarga completa de uma página.

2.3.2 Mobile

A **Aplicação Mobile** foi desenvolvida em *Kotlin* e **Jetpack Compose**[23]. A tecnologia foi escolhida pela familiaridade e por ter sido lecionado em uma unidade curricular anterior. O seu uso é exclusivo para os professores.

A motivação principal para a criação de uma aplicação móvel veio da possibilidade de oferecer segurança adicional ao sistema. Esta aplicação é uma ferramenta apenas para professores onde são realizadas todas operações de escrita sobre o **GitHub** e permite que no **Serviço Backend** nunca exista conhecimento ou armazenamento das chaves de acessos únicas dos professores que possibilitam essas ações.

A **Aplicação Mobile** oferece uma navegação simples com uma interface simples e tem o objetivo de aceitar pedidos realizados por estudantes, através da **Aplicação Web**, que envolvam a realização de operações de escritas no **GitHub**.

Para a serialização e desserialização usou-se a biblioteca *Jackson*, que foi preferida à biblioteca *Gson*[24] uma vez que se encontra em fase de manutenção e sem planos para melhorias, usando a biblioteca *OkHttp* para a realização dos pedidos às API's.

Existem duas fontes de informação, a do **GitHub** e a do **i-on ClassCode**. As ações de escrita precisam de autorização de um professor, o que permite um maior controlo por parte de um professor e uma maior facilidade na utilização das duas fontes de informação. Caso um professor aceite uma destas operações, é feito um pedido à **GitHub API** e consoante a resposta, são realizadas alterações no **Serviço Backend**. Um exemplo onde é possível verificar isto é na criação de uma equipa. Primeiro é criado um pedido através da **Aplicação Web** para criar uma equipa. Este pedido posteriormente surge na **Aplicação Mobile** onde será aceite ou rejeitado. Caso seja aceite, é feito um pedido à **GitHub API** e dependendo do resultado obtido como resposta, a equipa será criada no sistema **i-on ClassCode**.

Como existe o uso de *Uris templates*, optou-se pela utilização da biblioteca *damnHandy*[25], devido à sua implementação seguir a versão mais recente do RFC-6570[26].

2.4 Processo de Autenticação no Sistema

Durante o desenho da arquitetura do sistema **i-on ClassCode** foi tomada a decisão que o processo de autenticação do sistema não faria uso de uma solução que envolvesse o armazenamento de informação sensível dos utilizadores. Como tal, ao invés de ser adotado um desenho de arquitetura simples como o uso de *username* e *password*, optou-se pelo uso da *framework OAuth 2.0*[27] em todo o sistema. A *framework OAuth 2.0* tem como objetivo delegar autorização condicionada. Assim, o dono de recursos autoriza o acesso temporário a um conjunto pré determinado de recursos. No contexto do sistema **i-on ClassCode**, os utilizadores concedem acesso para aceder a recursos protegidos da sua conta **GitHub**, sem terem de partilhar informação sensível com o **i-on ClassCode**. Com isso é possível ter uma camada de segurança no sistema e oferecer controlo por parte do utilizador de quais permissões serão concedidas.

Como a implementação *OAuth* do **GitHub**, para a obtenção de *tokens* de acesso, oferece apenas suporte para *grant flows* do tipo *Authorization Code Grant*, o sistema obrigatoriamente utilizou este tipo de *grant flow*.

A utilização desta *framework* também traz dificuldades como a complexidade do desenho da experiência de utilização, a dependência de serviços externos e a configuração de uma aplicação cliente que envolve a obtenção das credenciais *Client ID* e *Client Secret* e a definição do URL de *callback*. Adicionalmente, os *tokens* de acesso fornecidos pelo *OAuth* podem ter um tempo de expiração o que envolve a sua atualização. A maior dificuldade associada à utilização desta *framework* é a realização de testes adequados à sua implementação porque é necessário garantir que todos os cenários e casos extremos, como falhas de autorização e adulteração dos pedidos do *flow OAuth*, sejam tratados corretamente.

2.5 API´s externas

O **i-on ClassCode** apresenta duas conexões a API´s externas, a **GitHub API** e a **SendGrid API**.

É de notar, aquando da existência de dependências externas, a necessidade de garantir a fiabilidade dos dados e da segurança entre serviços bem como do acrescento da dificuldade da manutenção do sistema devido à possibilidade da alteração do funcionamento dos serviços externos.

2.5.1 GitHub

A **GitHub API** permite aceder e manipular a dados e funcionalidades do **GitHub** de forma programática. A **GitHub API** suporta ainda, também, a autenticação de aplicações que utilizem a *framework OAuth 2.0*. Desta forma é possível obter informações sobre os utilizadores do **GitHub** registados no sistema do **i-on ClassCode** bem como a obtenção, criação e manipulação de organizações, repositórios e equipas. A utilização da **GitHub API** requereu um registo prévio da **API HTTP** do sistema no **GitHub**.

No sistema do **i-on ClassCode** existem três instâncias de comunicação com a **GitHub API**:

- No **Serviço Backend**, no processo de autenticação de um utilizador.
- Na **Serviço Backend**, onde são obtidas informações dos utilizadores **GitHub** (apenas leituras), de organizações.
- Na **Aplicação Mobile**, para serem realizadas operações de escrita como o caso de criação de um repositório.

O detalhe sobre o qual é retratado cada uma das instâncias de comunicação será abordado posteriormente na secção 3.

2.5.2 SendGrid

A **SendGrid API** é uma API fornecida pela **SendGrid** que é uma fornecedora de serviços email em nuvem. A **SendGrid API** permite o envio de e-mails de forma transacional através do serviço.

A **SendGrid API** oferece ainda uma ampla gama de recursos, incluindo o envio de emails personalizados e suporte para linguagens como *Java*, *Python*[28] e *Node.js*[29]. Os serviços do **Serviço Backend** utiliza a **SendGrid API** como uma melhoria de eficácia do fluxo de autenticação de um utilizador e também como uma peça adicional de segurança no processo de verificação.

2.6 Organização

Tal como todos os projetos que estão sobre alçada da iniciativa **i-on**[30], o projeto **i-on ClassCode** também é *open-source*. Logo, todo o código, do projeto **i-on ClassCode** pode ser visualizado e modificado por qualquer pessoa, o que traz a oportunidade de no futuro existirem contribuições para o desenvolvimento e melhoria do mesmo.

Todo o código fonte do projeto, como a sua organização podem ser encontrados no repositório **GitHub** (`https://github.com/i-on-project/repohouse`), onde existe um ficheiro README responsável por guiar qualquer pessoa pelas diretorias do projeto.

Na raíz do projeto, além do ficheiro de README, é possível encontrar duas pastas. Uma relacionada com a documentação (docs) e a outra com todo o código do projeto (code) onde dentro da mesma é feita a divisão em quatro partes da seguinte forma:

- *android* - onde pode ser encontrado todo o código desenvolvido referente à **Aplicação Mobile**.
- *js* - onde pode ser encontrado todo o código desenvolvido referente à **Aplicação Web**.
- *jvm* - onde pode ser encontrado todo o código desenvolvido referente ao **Serviço Backend**.
- *sql* - onde pode ser encontrada toda a esquemática da **Base de Dados**.

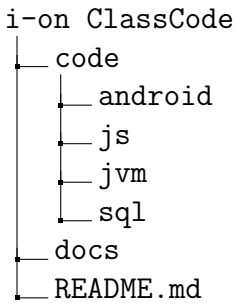

Figura 2: Árvore de pastas do projeto

3 Aspetos de Implementação

Nesta secção serão abordados todos os aspetos de implementação do projeto para todos os componentes do sistema. Serão aprofundados os temas de autenticação, acesso a API´s externas, testes, **Base de Dados**, **Serviço Backend**, **Aplicação Web** e **Aplicação Mobile**.

3.1 Serviço Backend

O **Serviço Backend** do **i-on ClassCode** é o serviço responsável por disponibilizar a **API HTTP** para a **Aplicação Web** e a **Aplicação Mobile**, realiza o tratamento da lógica do domínio e faz a comunicação à **Base de Dados**. A **API** é a interface de comunicação entre o serviço e as aplicações e é responsável por receber os pedidos, processá-los e produzir as devidas respostas.

É possível dividir o serviço em três partes principais: os **Handlers**, os **Services** e o **Repository**. O esquema da arquitetura do serviço pode ser visto na figura 3.

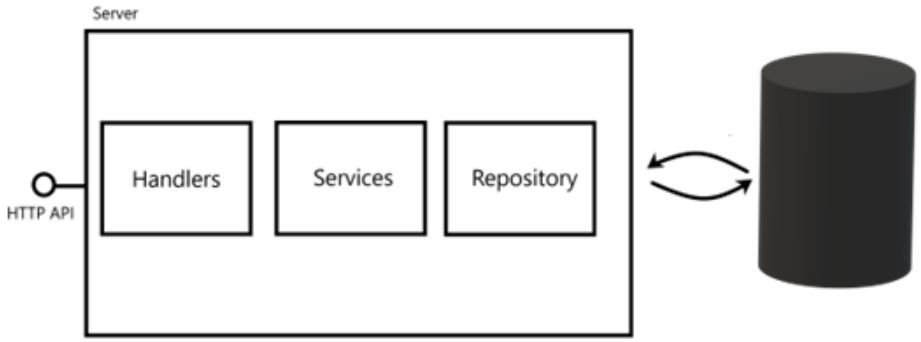

Figura 3: Arquitetura do serviço i-on ClassCode

3.1.1 Handlers

Um **Handler** é uma interface que facilita o tratamento de pedidos HTTP de forma bastante flexível no *Spring MVC* [31]. É usado em conjunto com anotações *Handler-Mapping*, que mapeiam um método para um *endpoint* específico da **API HTTP**. Além disso, os **Handlers** definem o código de *status* da resposta de acordo com o processamento do pedido. O *DispatcherServlet* é responsável pela gestão e distribuição dos pedidos HTTP, processando-os e reencaminhando-os para os **Controllers**. [ver 4].

Os **Handlers** encontram-se agrupados em **Controllers**. São também a única peça da **API HTTP** que conhece e usa HTTP.

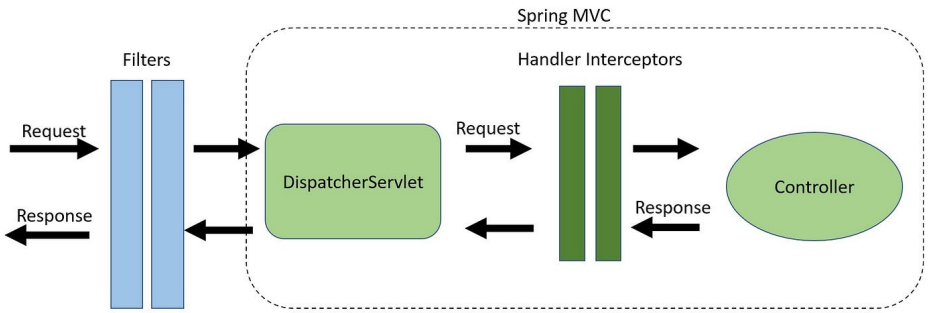

Figura 4: Exemplo do fluxo do Spring MVC

3.1.1.1 Respostas HTTP e Hypermedia

Seguindo o protocolo HTTP, as respostas contêm um código de *status*, um cabeçalho e um corpo.

A utilização de *Hypermedia* é uma das características do estilo da arquitetura REST. É também uma das características que a diferencia de outros estilos de arquitetura porque oferece uma série de vantagens, como a possibilidade de evolução da API sem quebrar clientes antigos, resiliência a mudanças, possibilidade de navegação autónoma por parte dos clientes e criação de um sistema mais robusto.

O formato das respostas HTTP da **API HTTP** geradas é baseado na especificação *Siren*, um padrão que contém várias propriedades que descrevem o recurso, os *links* e *actions* relacionados com o mesmo. O *media-type* das repostas da **API HTTP** é:

- para pedidos que sejam bem sucedidos - *application/siren+json*
- para pedidos que não sejam bem sucedidos - *application/problem+json* (independente da especificação *Siren*)

A representação *Siren* inclui:

- **class** : Um array que especifica a classe ou o tipo de recurso. Ajuda a providenciar contexto ao cliente sobre o mesmo. Podem ser defenidas classes para representar tipos de recursos.
- **properties** : Contém os atributos e valores do recurso. Representa o estado atual do recurso e dados associados a ele.
- **entities**: É um array com objetos de entidades aninhados. Cada entidade representa um recurso relacionado ao recurso atual. Os recursos relacionados podem ser usados para incluir dados ou providenciar uma estrutura hierárquica.
- **links** : Definem hiperligações associados ao recurso atual. Podem representar relações entre recursos e ser usados como navegação entre recursos. Cada *link*

contém informação da relação (*rel*) e o *href* representante do recurso, além de outras informações opcionais.

- **actions**: Descrevem ações ou operações que podem ser realizadas sobre o recurso atual. Uma ação especifica o método do pedido HTTP, um *endpoint* (*href*), um campo *fields* com os *body parameters* necessários para executar a ação e um *title* descritivo da ação.

A representação dos erros nas respostas HTTP segue a especificação do *media-type application/problem+json*:

- **type** : Um URL único identificador do tipo de erro para a documentação do mesmo.
- **title** : Um título que indica o problema que causou o erro.
- **description**: Uma descrição que fornece informação específica sobre o problema em questão.

Durante a implementação da geração das respostas da **API HTTP** definiu-se que todas as rotas retornadas (URI´s) devido ao uso de *Hypermedia*, não seriam específicas para o recurso em questão, mas sim *templates*. Por exemplo, se o recurso em questão for um *Course* que possui o identificador 1 no sistema, então para a obtenção da representação do recurso a resposta retornará o URI *template*, "/courses/{id}"ao invés do URI específico, "/courses/1".

Esta decisão foi justificada por algumas dificuldades enfrentadas durante o desenvolvimento da **Aplicação Web**, relacionadas com o armazenamento das rotas aprendidas por *Hypermedia*. Essas dificuldades consistiam na perca de estado da **Aplicação Web** sempre que ocorresse um redirecionamento ou fosse inserido um *deep-link* da aplicação *React* no *browser*. Para superar estas dificuldades seguiu-se a especificação RFC 6906 - Home Documents for HTTP API´s [32], mais conhecida como JSON *Home*. Esta especificação forneceu uma forma padronizada de descrever recursos e respetivos URI´s através do uso de URI *templates* e inspirou a criação de um *endpoint home* da **API HTTP** que como resposta, retorna todos os URI *templates* que correspondem a *endpoints* da **API**. A criação deste *endpoint* permitiu que durante qualquer carregamento da **Aplicação Web** no *browser* fosse feito um pedido HTTP para o mesmo, de forma a obter todos os URI *templates* necessários ao consumo da API.

3.1.1.2 Pipeline

A *pipeline* na *framework Spring* é o mecanismo de processamento de pedidos HTTP. O processamento é feito por um conjunto de componentes entre eles:

- **HandlerMapping**: mapeia um pedido para um *handler* e decide como é invocado.
- **DispatcherServlet**: serve como *entry point* para o processamento do pedido.
- **HandlerInterceptor**: interceta os objetos de pedido e de resposta antes e após ser processado pelo *handler*.
- **HandlerExceptionResolver**: resolve as excepções lançadas durante o processamento do pedido.

Durante o desenvolvimento da **API HTTP** criaram-se e adicionaram-se os seguintes componentes à *pipeline*:

- **LoggerFilter**: intercepta o processamento normal de um pedido no início e no final da *pipeline* e apresenta no *standard output* o *endpoint* para o qual foi feito o pedido e o código de resposta resultante do seu processamento.
- **AuthenticationInterceptor**: intercepta o processamento normal de um pedido no início da *pipeline* e verifica se um utilizador encontra-se autenticado quando necessário através da leitura de *cookies* de sessão.
- **UserArgumentResolver**: trabalha juntamente com o **AuthenticationInterceptor** para adicionar atributos ao pedido com informação do utilizador autenticado. Essa informação é depois também usada durante o processamento dos pedidos nos *handlers*.

3.1.2 Services

Os **Services** são a ponte entre os **Handlers** e o **Repository** e garantem a integridade da informação do sistema. São responsáveis pela verificação e validação dos *request bodies* e dos parâmetros do *request*, pela garantia das restrições do modelo de dados, controlo transacional de forma atómica e pela verificação da autenticidade dos utilizadores. Os **Services** têm como dependência um controlador de transações que permite a criação de transações e garante que estas são *commited*, quando não existe nenhuma violação da integridade do sistema, ou sofrem *rollback* quando tal não se verifica ou quando ocorre um erro de implementação interno do sistema. O controlador de transações fornece aos **Services** um objeto *transaction* que por sua vez permite acesso ao **Repository** responsável pela manipulação da **Base de Dados**.

Existem também outros dois tipos de serviços que são apenas responsáveis pela comunicação com as duas API's externas, a **GitHub API**, para a obtenção de informação sobre os utilizadores e a **SendGrid API** para o envio de emails utilizados no processo de verificação do registo de novos estudantes.

3.1.2.1 GitHub

A **GitHub API**, no contexto dos serviços, é utilizada para obter informação sobre os utilizadores, nomeadamente o seu nome, informação sobre o seu identificador e o seu email.

Caso o utilizador seja um professor, ainda é obtido o seu *token* de acesso, com o *scope 'read:org'*, *'user:email'* e *'repo'*, permitindo assim aceder a informação sobre as organizações, os emails e os repositórios do utilizador.

Através do *token* de acesso, é possível que o serviço faça pedidos à **GitHub API**, em nome do utilizador, permitindo assim aceder à informação que seja necessária, contudo estes dados só são passíveis de serem obtidos caso o utilizador tenha dado o seu consentimento, através da autenticação com o **GitHub**. [ver 3.1.5]

Um dos casos de utilização do token de acesso é quando um professor realiza a sincronização da turma ou do trabalho. Essa sincronização consiste em obter toda a informação dos repositórios e as respetivas *tags*, se existirem, das equipas de trabalho, verificando também os alunos que estão inseridos nas mesmas. Após obtenção de toda a informação, é comparada com a informação na **Base de Dados**, fazendo assim as alterações às entidades correspondentes, para ambas as verdades se encontrarem em sincronização.

3.1.2.2 SendGrid

O **Serviço Backend**, faz utilização da *Twilio* ***SendGrid*** *Email* ***API***, para o envio de emails. Após configuração da conta e obtenção da chave da API, é possível fazer pedidos à API, para o envio de emails.

Os emails são enviados no fluxo do registo de um estudante. Após a inserção do seu identificador escolar, é feito o envio de um email para o endereço de email do estudante, com um código de verificação, que deverá ser inserido no formulário de registo, para que o mesmo seja validado.

O pedido é feito através de um POST, contendo no *body*, além da API *Key*, o remetente, o destinatário e o conteúdo do email, sendo neste caso uma pequena mensagem com um OTP *(One Time Password)*.

Este processo foi implementado, seguindo o *Outbox Pattern*[33], onde o envio de emails é feito de forma assíncrona, guardando a informação dos emails enviados e por enviar na **Base de Dados**, sendo que o pedido é feito ao serviço de envio de emails, e retorna uma resposta de sucesso ou erro, sem que o utilizador tenha de esperar pelo envio do mesmo. Com a utilização do *scheduler*[34] do *Spring*, é possível que o serviço verifique num intervalo de tempo se existem emails por enviar, e caso existam, envia-os. Para a verificação, foi criado também a possibilidade do reenvio de emails para o utilizador. Preveniu-se também o reenvio de inúmeros emails através da

adição do conceito de *cooldown* ao modelo de dados, evitando assim uma sobrecarga na **SendGrid API**. Além disso, o OTP tem um número de tentativas máximo associado, que após ser ultrapassado, torna o OTP inválido.

3.1.3 Repository

O repositório providencia uma camada de abstração sobre a **Base de Dados** que permite a manipulação de objetos de domínio. O repositório é responsável por gerir a persistência dos entidades de domínio, ou seja, garante que são armazenados e recuperados objetos da **Base de Dados**. As interfaces pertencentes ao repositório, devido à abstração que fornecem, permitem que as camadas superiores não tenham conhecimento da **Base de Dados**, ou seja, as camadas superiores não precisam de saber como é que os objetos são armazenados, apenas precisam de saber que o repositório fornece métodos para guardar e recuperar os dados, fazendo assim uso do princípio de inversão de dependências, DIP[35].

3.1.4 Base de Dados

A **Base de Dados** é construída sobre PostgreSQL, com o auxílio da ferramenta *pgAdmin4*. É um sistema de gestão de dados relacional que providencia um armazenamento seguro, escalável e com várias capacidades de manipulação.

O modelo de dados pode ser encontrado no anexo A, e nele estão representadas todas as entidades usadas para a definição da **Base de Dados** que sustenta o sistema. É possível fazer uma divisão das entidades em grupos, como:

- **Base** : Todas as entidades que representam as peças domínio principais do sistema (eg. Users, Teacher, Student, Course, Classroom, Assignment, Team, Delivery, Repo, Tags e Feedbacks).
- **Fracas** : É o grupo das entidades que ajudam a fornecer informação entre entidades pertencentes ao grupo base (eg. Teacher_Course, Student_Classroom e Student_Team).
- **Pedidos** : Grupo ao qual pertencem as representações de todos os pedidos que auxiliam a **Aplicação Móvel** a realizar os pedidos à **GitHub API** (eg. Request, Composite, ArchiveRepo, CreateRepo, LeaveCourse, LeaveClassroom,CreateTeam, JoinTeam e LeaveTeam).
- **Outbox** : Três entidades que são responsáveis pelo processamento e envio de emails para verificação das contas no sistema(eg. Outbox, OTP e Cooldown).

- **Registo** : Antes de confirmar o registo de um cliente é guardada uma versão pendente do mesmo, isto serve para guardar o estado do pedido de registo (eg. PendingTeacher e PendingStudent).

- **Autenticação Móvel** : Para os pedidos de autenticação móvel, é feito o registo do *challenge* associado ao *state* do pedido realizado, para o mesmo ser depois verificado (eg. ChallengeInfo) [ver 3.3.4].

A entidade *Composite* representa um conjunto de *Request* e é usada para agrupar pedidos que devem ser realizados em conjunto. Um exemplo é o pedido de criação de uma *Team* no **GitHub** que é associado ao pedido de criação do repositório e o pedido de criação da *Team*.

3.1.4.1 Triggers

Os *triggers* são utilizados para garantir a integridade dos dados e são configurados para que quando ocorra uma determinada alteração na **Base de Dados**, sejam ativados e executem uma função específica. A **Base de Dados** tem configurados os seguintes *triggers*:

- **UpdateApplyRequests**: após a alteração de estado de um pedido de um registo de professor, se este for aceite, são feitas as inserções necessárias para que o utilizador passe a ser utilizador e professor no sistema.

- **DeleteTeachers**: um *trigger* que, quando ativado, está encarregado da eliminação de todos os registos que estejam associados ao professor eliminado.

- **DeleteTeam**: antes da eliminação de uma equipa, o *trigger* garante que tudo o que está referenciado pela equipa é apagado.

3.1.5 Autenticação

A autenticação em todo o sistema é realizada seguindo a *framework OAuth2.0*. O servidor de autorização usado foi o **GitHub**. Durante o processo de autenticação, a aplicação cliente faz um pedido ao **GitHub**. A resposta é uma página **HTML**, que será carregada e é pedido ao utilizador que autorize a aplicação a aceder aos seus dados através da sua conta do **GitHub**.

O protocolo de autorização pelo **GitHub** necessita do registo inicial da aplicação para ser colocado em uso:

1. **Registo**: O registo da aplicação é feito através do *GitHub Developer Settings*[36], onde é preciso indicar o nome da aplicação, a descrição da aplicação, o URL da aplicação e o URL de *callback*.

- ***Callback***: O URL de *callback* é o URL para onde o utilizador é redirecionado após a autenticação no **GitHub**.

2. **Credenciais**: Após o registo da aplicação, são geradas as credenciais da aplicação, sendo estas:

   - ***Client ID***: É um identificador único da aplicação.
   - ***Client Secret***: É uma chave secreta usada para autenticar os pedidos feitos pela aplicação.

A **API HTTP** apresenta vários *endpoints* que pertencem ao processo de autenticação. Existem *endpoints* de autenticação próprios para ambas as aplicações do sistema. Na **Aplicação Mobile** apenas é possível fazer *login* sendo necessário o registo prévio na **Aplicação Web**. A imagem 5 reflete o processo de autenticação no **Serviço Backend**.

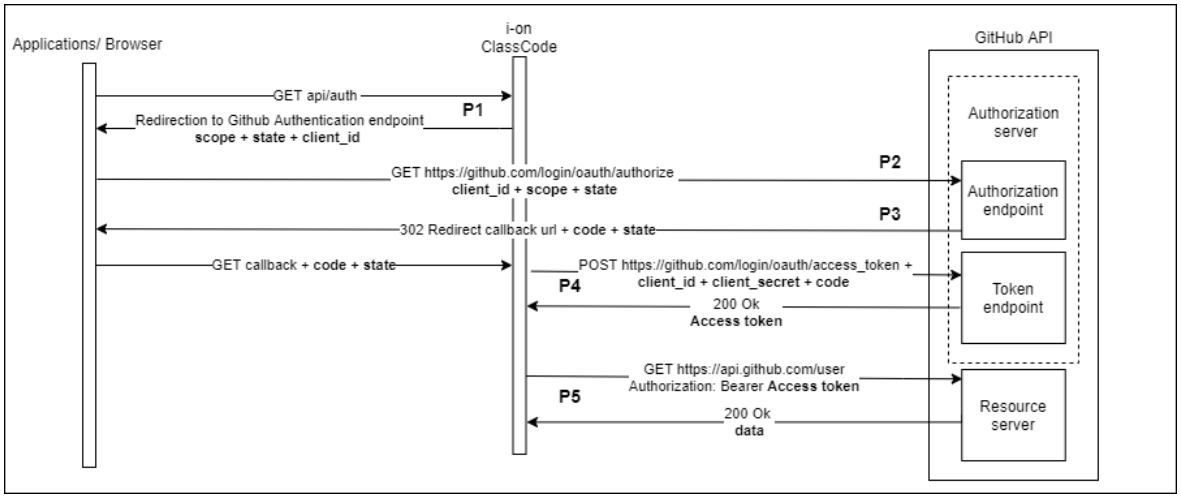

Figura 5: Autenticação no Serviço Backend

Para iniciar o processo de autenticação para os diferentes tipos de utilizador existem *endpoints* distintos. No caso da **Aplicação Web**, um pedido *GET* para estes *endpoints* deverá será feito pela própria aplicação e origina sempre como resposta o URL para a página de autorização do **GitHub**. Este URL possui sempre o *Client ID*, o URL de *callback* e o *scope* correspondente ao tipo de utilizador (**P1**). No caso da **Aplicação Mobile**, um pedido *GET* para o *endpoint* de início do processo de autorização deverá ser feito pelo *browser* do dispositivo móvel e a resposta provoca um redirecionamento para a página de autenticação do **GitHub**. O URL de redirecionamento possui também o *Client ID*, o URL de *callback* e o *scope* correspondente aos professores.

Em qualquer um dos casos, o URL para a página de autorização do **GitHub** possui sempre um valor aleatório denominado *state*. O *state* é um valor aleatório gerado pelo **Serviço Backend** e é usado para prevenir ataques CSRF [37]. Quando é feito o pedido

de retorno para o URL de redirecionamento é verificado se o *state* recebido é igual ao *state* enviado e caso não seja, é produzida uma resposta com um erro.

Após a autenticação no **GitHub** (**P2**), se bem sucedida, o **GitHub** realiza o pedido ao *endpoint* de *callback* da **API HTTP** (**P3**). Após a verificação do *state* por parte do **Serviço Backend**, é feito um pedido à **GitHub API** para obter o *token* de acesso do utilizador em troca do código fornecido após a autenticação no **GitHub** (**P4**). O URL que corresponde ao *endpoint* para o pedido de obtenção do *token* de acesso deve apresentar na *query string*, o *Client ID*, *Client Secret* e o código.

O *scope* é usado para definir o nível de acesso que o **Serviço Backend** tem aos dados do utilizador.

No caso dos professores, o *scope* tem além da informação do utilizador, a informação das organizações que possuí, armazenando no servidor o *token* de acesso cifrado com apenas permissões de leitura. Com a autenticação na aplicação móvel, o *scope* dos professores usado é o de mais alto nível (*admin:org*), sendo que não é guardada informação sobre o *token*.

A partir do *token* de acesso é obtida a informação do utilizador (**P5**) que será armazenada durante o primeiro registo. Com a obtenção da informação do utilizador, o registo de um utilizador envolve ainda, no caso de:

- **Professores**: A confirmação da informação de utilizador obtida a partir do **GitHub** para a criação de um pedido de registo que deverá ser ainda aceite por outros professores no sistema.

- **Estudantes**: A inserção do seu número de estudante, a confirmação da informação de utilizador obtida a partir do **GitHub**, o envio de um OTP (One Time Password) para o seu email académico e a confirmação da conta após a verificação desse OTP.

Após a conclusão do processo de autenticação, quer o utilizador faça *login* ou se registe, se bem sucedido, são mantidos *cookies* de sessão que permitem ao utilizador não ter de se autenticar novamente, caso o *token* de acesso não tenha expirado.

3.1.5.1 Cookies

Os *cookies* são pequenos blocos de dados que os *websites* ou aplicações guardam num dispositivo do utilizador. São usados para manter informação sobre o utilizador, tais como, preferências, estado de autenticação e outros tipos de informação.

Aos *cookies* podem ser atribuídos atributos que modificam a forma como são usados. No sistema do **i-on ClassCode** os atributos relevantes são:

- **Path** : indica um URL que deve existir no URL solicitado para o envio do cabeçalho do *Cookie*.
- **HttpOnly** : quando verdadeira, esta *flag* restringe o acesso ao *cookie* prevenindo assim que *scripts* do lado cliente tenham acesso e também mitigando possíveis ataques XSS [38].
- **Secure** : define se os *cookies* sejam só enviados através de conexões seguras (HTTPS).
- **SameSite** : especifica se os *cookies* são enviados em pedidos *cross-site*.
- **MaxAge** : define o tempo de validade do *cookie*, permitindo a sua eliminação se este for 0.

Os *cookies* presentes no sistema são:

- **UserState** : incluí o valor de *state*, para o processo de autenticação do utilizador.
- **Session** : é o valor do *token* de sessão do utilizador no sistema.
- **Position** : indicação sobre o tipo de utilizador, professor ou estudante.
- **UserGithubId** : valor do identificador **GitHub** do utilizador.

Todos estes *cookies* apresentam o valor *true* nos atributos *HttpOnly* e *Secure*. Porém, os *cookies UserState* e *Position* apresentam o valor *None* no atributo *SameSite* porque pretende-se que sejam mantidos durante e após o processo de autenticação na página do **GitHub**. Por outro lado, os *cookies Session* e *UserGithubId* apresentam o valor *Strict* no atributo *SameSite* para evitar ataques às credenciais dos utilizadores.

Quer para o caso do *cookie Session*, quer para o caso do *cookie UserGithubId*, os seus valores são encriptados usando uma combinação de algoritmos criptográficos e esquemas de *padding*. O sistema usa o esquema criptográfico AES/CBC/PKCS5PADDING, sendo que :

- **AES** : é um algoritmo de cifra simétrica que fornece uma maneira segura e eficiente de encriptar informação em blocos fixos de tamanho, 128, 192 ou 246 *bits*.
- **CBC** : é um modo de operação de cifra em bloco que usa a operação *XOR* em relação ao bloco anterior cifrado, permitindo assim segurança e prevenir que blocos sejam facilmente descobertos.
- **PKCS5PADDING** : é um esquema que adiciona *padding* aos blocos antes de os mesmos serem cifrados.

Para o algoritmo AES é necessária uma chave secreta para cifrar e decifrar informação e essa chave é definida através da variável de ambiente `CLASSCODE_ENCRYPTION_KEY`.

3.1.6 Armazenamento de Informação Sensível

Durante o desenvolvimento do **Serviço Backend**, foram consideradas duas soluções para a autenticação de pedidos HTTP. Autenticação através do uso de *cookies* e através de *Bearer Tokens* presentes nos cabeçalhos de autenticação do pedido. O processo de autenticação pelo qual se optou é baseado em *cookies*, uma vez que o uso dos cabeçalhos HTTP estava suscetível a ataques XSS e os *cookies* oferecem proteção contra este tipo de ataques através do uso do atributo *HttpOnly*.

O armazenamento dos *tokens* de sessão bem como dos *tokens* de acesso ao **GitHub**, apenas com permissões de leitura, é feito na **Base de Dados**. Como tal, foi necessário implementar medidas de proteção para que não seja possível de forma simples a consulta destes dois tipos de informação sensível. Antes de serem armazenados, é criada uma marca sobre os *tokens* de sessão através da aplicação da função de *Hash SHA256*, para que exista uma camada de proteção caso a base de dados seja comprometida.

O armazenamento desta marca impossibilita que a partir da consulta da **Base de Dados** seja possível obter um *token* de sessão, uma vez que valores *hash* são impossíveis de ser decifrados devido a não corresponderem a uma forma de encriptação. Desta

forma a autenticação de um utilizador é feita produzindo o *hash* do valor do *cookie* de sessão e comparando-o com as marcas armazenadas.

Os *tokens* de acesso **GitHub** por sua vez não necessitam de tanta proteção visto que não conseguem ser usados para a realização de ataques que envolvam escritas. Estes *tokens* são encriptados através do esquema criptográfico AES/CBC/PKCS5PADDING antes de serem armazenados na **Base de Dados**.

3.1.7 Testes Automáticos

Para o **Serviço Backend** foram criados testes unitários para os componentes **Services** e **Repository**. Existem ainda alguns testes de integração para os componentes **Controller**.

Os testes unitários são testes com foco na individualidade dos componentes e servem para garantir que o código desenvolvido apresenta a funcionalidade pretendida e que não sofre alterações entre iterações no desenvolvimento.

A *framework* usada foi a *JUnit*[39], uma *framework* Java que facilita a criação e execução dos testes unitários. Através de um conjunto de anotações e asserções permite criar casos de testes e a automatização do processo de testes.

Para testar todos os componentes do sistema individualmente foi criada uma adaptação do controlo transacional que evita que quaisquer alterações realizadas durante a execução dos testes sejam *commited* e que no final dos testes sofram *rollback*. Todos os dados usados para os testes pertencem ao ficheiro de inserção SQL disponível na pasta code/sql com o nome *insert.sql*.

3.2 Aplicação Web

A **Aplicação Web** é o principal meio de interação entre o cliente e o sistema. É nela em que estão presentes as principais funcionalidades do sistema:

1. Associar um repositório do **GitHub** a um curso
2. Criar turmas associadas a cursos e organizações
3. Enviar códigos de convite aos alunos para se juntarem a uma turma
4. Criar tarefas
5. Criar grupos de trabalho
6. Definir várias entregas para uma tarefa com controle de *tags* e prazo
7. Verificar, para cada tarefa, os grupos de trabalho que entregaram e que não entregaram
8. Publicar *feedbacks* para as equipas
9. Aceitar que outros utilizadores do **GitHub** se juntem como professores

Decidiu-se que a **Aplicação Web** seria uma *Single-Page Application* para evitar a necessidade de realizar um pedido cada vez que fosse necessário uma mudança da página a mostrar. Isto foi conseguido através de duas tecnologias, o *ReactJS* e o *React Router*[40]. O *React Router* é uma biblioteca que permite a navegação entre componentes sem que seja necessário recarregar a página. Para além dessas duas tecnologias foi usada uma extensão da linguagem *JavaScript*, *JSX*, que permite escrever código *JavaScript* com elementos semelhantes a HTML.

A **Aplicação Web** está dividida em 3 partes, os componentes *React*, que definem a interface de utilização,os serviços, que definem como são feitos os vários pedidos para a **API HTTP** do **Serviço Backend**, e o repositório de navegação, responsável pelo armazenamentos de todos os *links* e *actions* obtidos por *Hypermedia*.

3.2.1 Componentes React

Os componentes *React* foram criados com o intuito de serem reutilizáveis e de serem facilmente compreendidos. Eles englobam tanto a lógica da aplicação como uma parte da interface de utilização especifica e são compostos por outros componentes *React*, ou por elementos HTML.

De forma a manter informação sobre o estado de autenticação de um utilizador ao longo de toda a aplicação fez-se uso de uma das funcionalidades fornecidas pelo

React, chamada *Context*. Um *Context* permite que a informação seja passada entre componentes, sem que seja necessário passá-la explicitamente como propriedade ao longo de toda a árvore de componentes. Desta forma é possível manter um estado global da aplicação passível de ser acedido por qualquer componente da aplicação.

A implementação de um *Context* envolve para além da sua criação e definição das suas propriedades (estados da aplicação e os *setters* para a sua alteração), a criação de um componente de topo (*Context Provider*) e a definição de funções que retornem os estados da aplicação e os respetivos *setters*.

No contexto da aplicação foi criado um *Context* que mantém informação sobre o estado de autenticação da aplicação. Essa informação é caracterizada por três estados diferentes:

- **loggedin**: indicação do tipo de utilizador autenticado (estudante, professor ou nenhum);
- **userId**: valor que pode ser *null* ou representa o identificador do utilizador no sistema do **i-on ClassCode**;
- **githubId**: valor que pode ser *null* ou representa o identificador do utilizador no **GitHub**.

3.2.2 Repositório de Navegação

O repositório de navegação é responsável pelo armazenamentos de todos os *links* e *actions* obtidos por *Hypermedia*. O armazenamento é feito com base em valores par chave-valor. Neste contexto as chaves são propriedades dos objetos JSON das respostas da **API HTTP**. Estas chaves correspondem ao valores da propriedades *rel*, no caso de *links*, e *title*, no caso de *actions* [ver 3.1.1.1].

3.2.3 Serviços

Os serviços na **Aplicação Web** são responsáveis pela comunicação com o **Serviço Backend**. São responsáveis ainda pela preparação dos *request bodies* e transformação dos URI's *templates* em URL's específicos. A obtenção do URL necessário à realização do pedido é feito a partir de chaves armazenadas no repositório de navegação.

A utilização dos serviços por parte dos componentes requereu a criação de um *hook* customizado da aplicação uma vez que no *React* não é permitida a presença de *side-effects* nos componentes *React*. Assim a chamada a qualquer método assíncrono dos serviços deve ser feita usando o *hook* customizado - *useAsync*. Este *hook* internamente utiliza o *hook useEffect* que na biblioteca *React* fornece suporte para a realização de *side-effects* nas funções componente.

3.2.4 Autenticação

No processo de autenticação de um utilizador a partir da **Aplicação Web** é necessário ter em consideração a existência de pedidos HTTP realizados pelo *browser* e pela própria aplicação a partir da *Fetch API*.

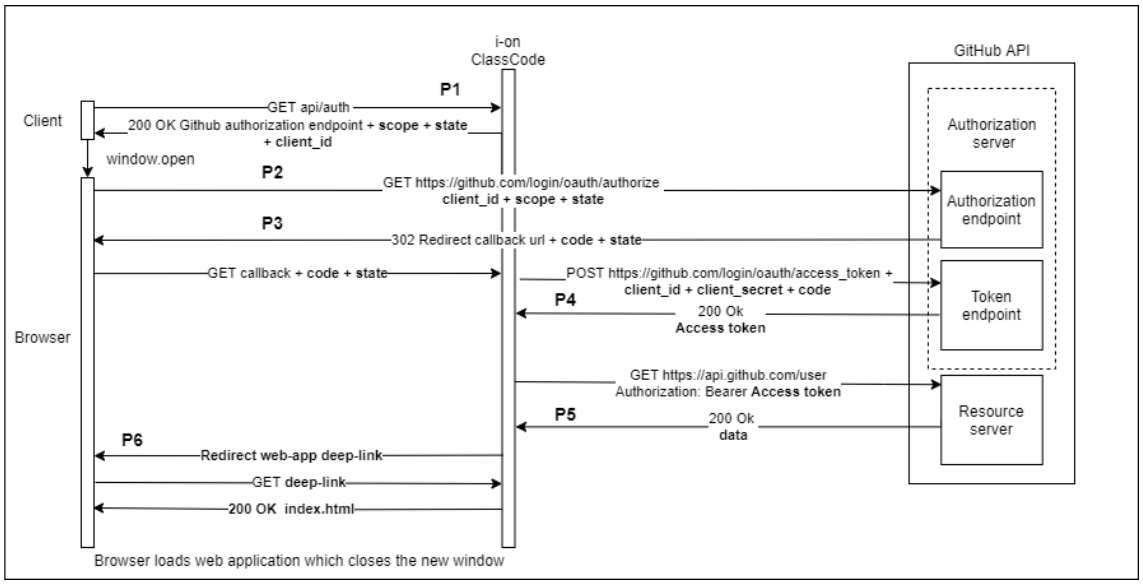

Figura 6: Autenticação na Aplicação Web

Tal como referido na secção 3.1.5, para iniciar o processo de autenticação é necessário a **Aplicação Web** realizar um pedido para um dos *endpoints* de autenticação do **Serviço Backend**, de acordo com o tipo de utilizador. A resposta a esse pedido fornece à **Aplicação Web** o URL para com o pedido de autorização (**P1**).

Após obter o URL para a página de autorização, a aplicação abre uma nova janela do *browser* e provoca um pedido realizado pelo *browser*, para a página de autenticação do **GitHub**. A abertura de uma nova janela é feita através do método *window.open* (**P2**) e é feita para que não haja perca do estado da janela da aplicação. Neste ponto o utilizador insere as credencias da sua conta **GitHub** e após a autenticação, o **GitHub** realiza um pedido para o *endpoint* de *callback* da **API HTTP** do **Serviço Backend** (**P3**).

Nesta fase, o **Serviço Backend** obtém o *token* de acesso do utilizador a partir do código retornado na *query string* (**P4**) e a informação de utilizador (**P5**). Como o pedido para o *endpoint* de *callback* foi realizado pelo *browser*, a resposta a esse pedido não deve ser uma representação JSON mas sim uma resposta com o código de *status* 303 de forma a provocar um redirecionamento para um *deep-link* da **Aplicação Web**.

O pedido para o *deep-link* da **Aplicação Web** também é realizado pelo *browser* porém, configurou-se o servidor do **i-on ClassCode**, sempre que for feito um pedido para um *endpoint* desconhecido da **API HTTP**, para enviar como resposta o ficheiro *index.html* que contém a **Aplicação Web**. Assim, o *browser* carrega nova-

mente na nova janela a **Aplicação Web** que é capaz de resolver o *deep-link* (**P6**). A estes *deep-links* correspondem componentes da *User-Interface* da **Aplicação Web** que são responsáveis pelo fecho da nova janela e pela atualização da *User-Interface* na janela principal. A nova janela realiza o seu próprio fecho através do método *window.close* após o envio de uma mensagem para a janela principal através do método *window.opener.postMessage*. Essa mensagem contém o novo *deep-link* que será depois resolvido pela **Aplicação Web** em execução na janela principal do *browser* para a continuação do uso da aplicação.

3.2.5 Testes de Interface de Utilizador

Para a realização de testes automáticos do lado do cliente fez-se uso da biblioteca *open-source* do *Node.js*, *Playwright*[41]. Esta biblioteca é capaz de automatizar testes da interface de utilização em execução no próprio *browser* cliente e fornece suporte visual durante a execução dos testes. Permite ainda visualizar as várias iterações das páginas *web* resultantes das várias ações definidas nos testes de forma automática em diversos *browsers*.

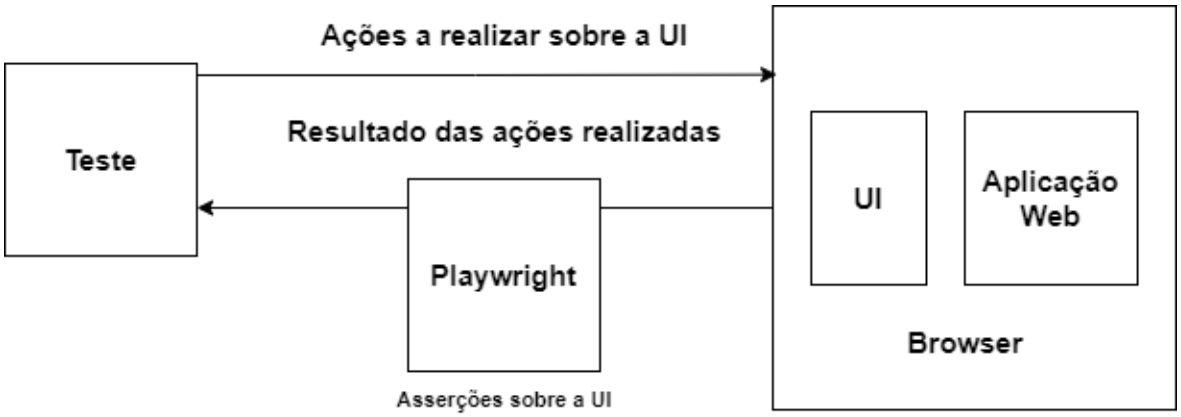

Figura 7: Diagrama da execução dos testes de interface de utilizador

Durante a execução dos testes de interface de utilizador a **Aplicação Web** encontra-se em execução no *browser*. Os testes representam as ações que devem ser realizadas sobre a interface de utilizador, como por exemplo, a espera da autenticação de um utilizador no **GitHub** e a inserção de texto em elementos HTML de *input*. Após a realização de cada ação, é produzido um novo estado da interface de utilizador e a biblioteca *Playwright* realiza asserções sobre o novo estado para a verificação dos comportamentos esperados.

Para a execução dos testes é necessário inicialmente que sejam criados quatro utilizadores na **Base de Dados**. Estes utilizadores são um professor que se encontra

registado, um professor em processo de validação, um estudante registado e um estudante ainda pendente que se encontra na fase de inserção do seu identificador escolar. Para tal, deve ser executado um *script* SQL que se encontra na pasta code/js/tests.

Os testes criados permitem garantir que o fluxo e experiência de utilização da aplicação vão de encontro ao desejado e garantem melhor conhecimento do comportamento da aplicação e a prevenção de erros de implementação.

3.3 Aplicação Mobile

A **Aplicação Mobile** foi desenvolvida com o intuito de resolver o problema associado à existência de *tokens* de acesso do **GitHub** com permissões de escrita, na **Base de Dados** do **Serviço Backend**. Assim é possível evitar um ataque interno capaz de expor os *tokens* de acesso. Estes *tokens* são informação sensível porque permitem fazer escritas na conta **GitHub** de um utilizador do **i-on ClassCode**. A principal função da aplicação é fornecer suporte para serem aceites vários tipos de pedidos diferentes, como criar uma equipa ou entrar numa organização.

É importante realçar a existência de duas fontes de informação: o **i-on ClassCode** e o **GitHub**. Por exemplo, se no sistema do **i-on ClassCode** existirem *classrooms*, que representam turmas, haverão estados em que essas *classrooms* não existem no **GitHub**.

Quando um professor abre a aba de pedidos é possível ver quais pedidos vão ser feitos ao **GitHub**. Também é possível ver o histórico de todos os pedidos que já foram aceites para uma dada equipa.

Na **Aplicação Mobile** existem três aspetos de implementação principais:

- **Segurança** - como é que o *token* de acesso ao **GitHub** e o *cookie* de sessão para o consumo do **Serviço Backend** do **i-on ClassCode** foram guardados e utilizados.
- **Serviços** - oferecem suporte para a realização de pedidos HTTP às duas API´s e transformam as respetivas respostas em informação compreensível e necessária para as camadas superiores.
- ***User Interface*** (**UI**) - desenvolvida com *Jetpack Compose* e descreve o que o utilizador vai observar.

3.3.1 Segurança

Para que a utilização de um aplicação móvel fosse uma opção viável para aumentar a segurança na autenticação perante as duas API's, usou-se um *Android Key Store*[42]. O *Android Key Store* dificulta a extração das chaves criptográficas e também restringe

a utilização das mesmas. A chave presente no *keystore* é usada para a encriptação e desencriptação do token de acesso à **GitHub API** e a *cookie* de sessão do **i-on ClassCode**.

A cifra e decifra utiliza o esquema criptográfico AES/CBC/PKCS7Padding, referido na secção 3.1.5.1 . Caso exista já uma chave no *keyStore* é usada essa mesma chave, caso contrário é gerada uma. O formato *Base64* é usado depois no resultado da encriptação para evitar problemas ao converter um array de bytes para string e vice-versa.

O resultado da encriptação e o *Initialization vector* (IV) são guardados usando um *Data Store Preferences*[43]. No curso foi lecionado o uso das *Shared Preferences*, mas acabou por se fazer uso de *Data Store Preferences* devido a duas vantagens:

- Garantir *type safety*, permitindo encontrar erros em tempo de compilação. Algo que as *Shared Preferences* só realizam em *runtime*.
- Faz uso de operações assíncronas e não bloqueantes de *Input/Output*, que permite uma utilização de operações que operam no *data storage* sem bloquear a *main thread*.

Para o utilizador se autenticar perante a **Aplicação Mobile**, necessita de usar a forma de autenticação predefinida que tem no telemóvel. Caso o utilizador tenha biometria como a sua principal, a mesma precisa de um *pin* no caso de haver algum problema com o sensor ou se ultrapasse o número máximo de tentativas. Na **Aplicação Mobile** se não se fizer uso de biometria, é possível usar outra forma de autenticação que o utilizador tenha no seu dispositivo. Depois de se autenticar, o *KeyStore* permite o uso da chave criptográfica tornando possível desencriptar a informação sensível necessária para os pedidos às API´s.

3.3.2 Serviços

Os serviços são o intercomunicador entre a camada de apresentação e a camada de acesso a dados. Os serviços, quando é necessária autenticação para executar um pedido, têm de pedir à camada de acesso a dados a informação indispensável para realizar o pedido.

Também são responsáveis por desserializar o *payload* das respostas HTTP para as classes que representam as várias entidades, como *Classroom* ou *Assignment*. É de referir que para os serviços terem acesso ao vários *links* e *actions* das respostas *Siren* omitidas pelo **Serviço Backend**, foi criada uma classe que permite a gestão destes objetos. A sua implementação é bastante semelhante à mesma que foi desenvolvida para a **Aplicação Web** [ver 3.2.2].

Após processar as respostas, os serviços enviam os dados para a camada de apresentação. Caso haja um pedido onde a resposta seja de um erro, os serviços são capazes

de enviar a informação desse erro à camada superior para que a mesma tome as devidas ações.

3.3.3 User Interface

A UI foi desenvolvida com *Jetpack Compose* com a presença de 3 componentes fundamentais, *Activity*, *View Model* e *Ecrãs/Views*.

A *Activity* representa um ecrã que está constantemente visível. É responsável pela lógica de navegação entre as várias *Activities* que existam (assumindo que existem várias *Activities* e que exista uma *Activity* por um *Ecrã/View*). Também é a *Activity* que tem a responsabilidade de conjugar o *ViewModel* e o *Ecrã/View*.

Um *ViewModel* armazena o estado especifico da *Activity* a que está associado. Gere operações assíncronas e gere o cancelamento das mesmas. Tem uma enorme utilidade devido a não ser destruído e perder o estado caso haja uma reavaliação de estado na *Activity*. Tem também a função de chamar os métodos dos serviços e guardar e alterar o estado que daí provém para a recomposição da *Activity*.

Os *Ecrãs* são uma junção de uma ou várias funções *composable* que descrevem a UI. Os *Ecrãs* mantêm estado de apresentação e não de aplicação porque caso exista uma reavaliação o estado é perdido. As *Views* são funções *composable* que poderão ser reutilizadas por vários *Ecrãs*.

Durante o desenvolvimento da **Aplicação Mobile** adotou-se o esquema em que para uma *Activity* existe um *ViewModel* e um *Ecrã*. Com isto, garantiu-se uma maior modularidade e também uma maior facilidade na navegação e compreensão do código.

3.3.4 Autenticação

Para a autenticação na **Aplicação Mobile** fez-se uso da *framework OAuth2.0* e seguiu-se o RFC 7636: Proof Key for Code Exchange (PKCE)[44]. O PKCE adiciona uma camada de proteção quando um atacante interceta a mensagem que contém o código a ser trocado pelo *token* de acesso. No PKCE, a **Aplicação Mobile** gera um segredo e envia o seu *hash* ao servidor. Quando enviar o código para ser trocado pelo *token* de acesso, também será enviado o segredo que o servidor irá validar [ver 8].

Para usar o browser, foi utilizado *Custom Browser Tabs*[45] para seguir o RFC 8252: OAuth 2.0 for Native Apps[46]. Mais especificamente, o subcapítulo 'B.2. Android Implementation Details'[47], que é para o caso específico duma aplicação *Android*. Importante referenciar que não foi usado um esquema HTTPS.

No contexto do **i-on ClassCode**, a **Aplicação Mobile** gera o segredo, **(P1)**, e cria um *intent* para ser enviado para uma *Custom Browser Tab* para fazer o pedido ao **Serviço Backend**, **(P2)**. Este pedido contém o *hash* do segredo - *challenge*, bem como o método usado. A aplicação do *i-on ClassCode* irá associar o *state* gerado para

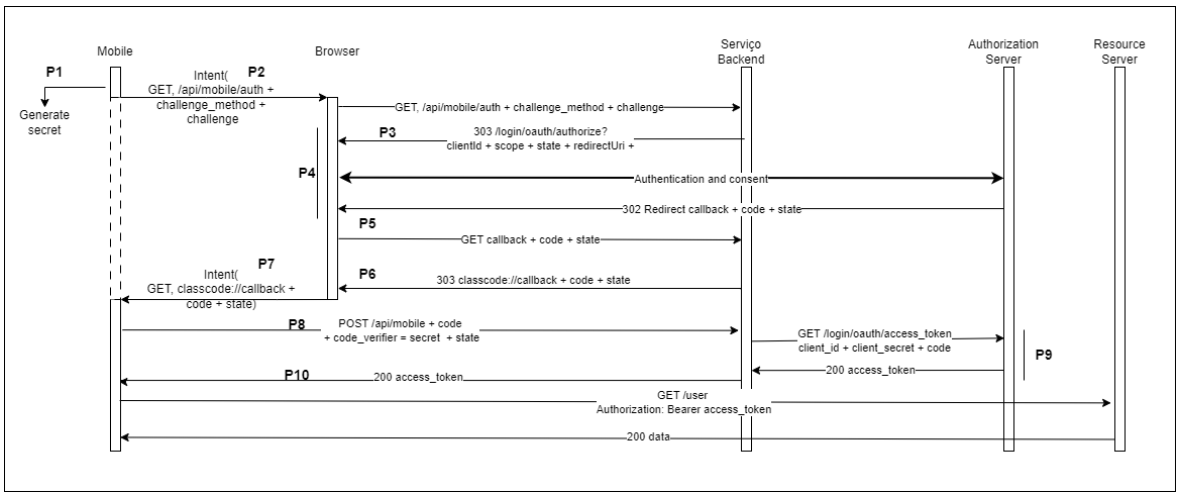

Figura 8: Autenticação na Aplicação Móvel

realizar o *flow OAuth2.0*, com o *challenge* e o método. Esta associação deve-se ao facto de haver necessidade de no futuro, conseguir associar o utilizador ao seu *challenge*. O *state* é usado por ser o único valor que se mantém desde a primeira interação com o **Serviço Backend**.

Após guardar as informações do *challenge*, responderá com um redirecionamento, **(P3)**, e será feito o *flow OAuth2.0* normal, **(P4)**. Quando o *flow* termina, o browser faz um pedido para o *link* de *callback* do **Serviço Backend** que está registado no **GitHub**, **(P5)**. Este pedido contém o *state* a ser verificado no **Serviço Backend** e o código que vai ser trocado pelo *token* de acesso - *code*. A resposta será um redirecionamento para um *deep-link* da **Aplicação Mobile** onde está contido na *query*, o *code* e o *state*, **(P6)**. Com este *deep-link*, a *Custom Browser Tab* cria um *intent* que é enviado a **Aplicação Mobile**, **(P7)**.

A **Aplicação Mobile** faz um pedido POST para **Serviço Backend** com o segredo, *state* e o *code*, **(P8)**. O **Serviço Backend** vai buscar as informações do *challenge* e do método através do *state* e aplica o método ao segredo, comparando o resultado com o *challenge* guardado. Se o resultado for igual, irá enviar um pedido à **GitHub API** para trocar o *code*, **(P9)**.

A resposta ao pedido feito pela **Aplicação Mobile**, retorna o *token* de acesso e um *cookie* de sessão, **(P10)**. Este *cookie* é enviado nos próximos pedidos ao **Serviço Backend**, para autorização de operações que necessitam de um utilizador autenticado.

Com este esquema, aumentou-se a segurança no uso e armazenamento do *token* de acesso porque este nunca será guardado no **Serviço Backend**.

O *code* nunca é guardado, prevenindo que algum ataque interno fosse feito para obter o *token* de acesso. Mesmo que alguém intercete a resposta que contém o *code* não é possível obter o *token* de acesso, porque não tem o segredo e consequentemente não consegue fazer o pedido diretamente ao **GitHub**, porque, mesmo que tenha o

Client ID, não tem acesso ao *Client Secret*.

4 Disponibilização do Serviço Backend e das Aplicações

A disponibilização do **Serviço Backend** bem como da **Aplicação Web** foi feita através do uso do sistema de contentores Docker[48] e da ferramenta Docker Compose[49]. A ferramenta Docker Compose permite definir múltiplas instâncias de serviços docker responsáveis pela execução dos componentes do sistema **i-on ClassCode** em múltiplos contentores. Para disponibilizar tanto o **Serviço Backend** e a **Aplicação Web** criou-se um ficheiro yaml, docker-compose, que define dois serviços docker. O ficheiro docker-compose é responsável por criar:

- uma ou múltiplas instâncias do **Serviço Backend**.
- uma instância do NGINX [50] para servir a **Aplicação Web** e aproveitar as múltiplas instâncias do **Serviço Backend** para balanceamento de carga.

O NGINX é capaz de servir conteúdo estático, balancear a carga entre vários servidores e lidar com comunicação SSL/TLS. O NGINX é amplamente utilizado em *sites*, aplicativos e API´s.

Para a configuração do servidor HTTPS e de forma a tornar todas as ligações a este seguras, atribuíram-se ao servidor um certificado de segurança e a respetiva chave privada. Para a criação dos certificados e respetiva validação foi usada a ferramenta Certbot [51], um software *open-source* que usa de forma automática certificados gerados pelo Let's Encrypt [52].

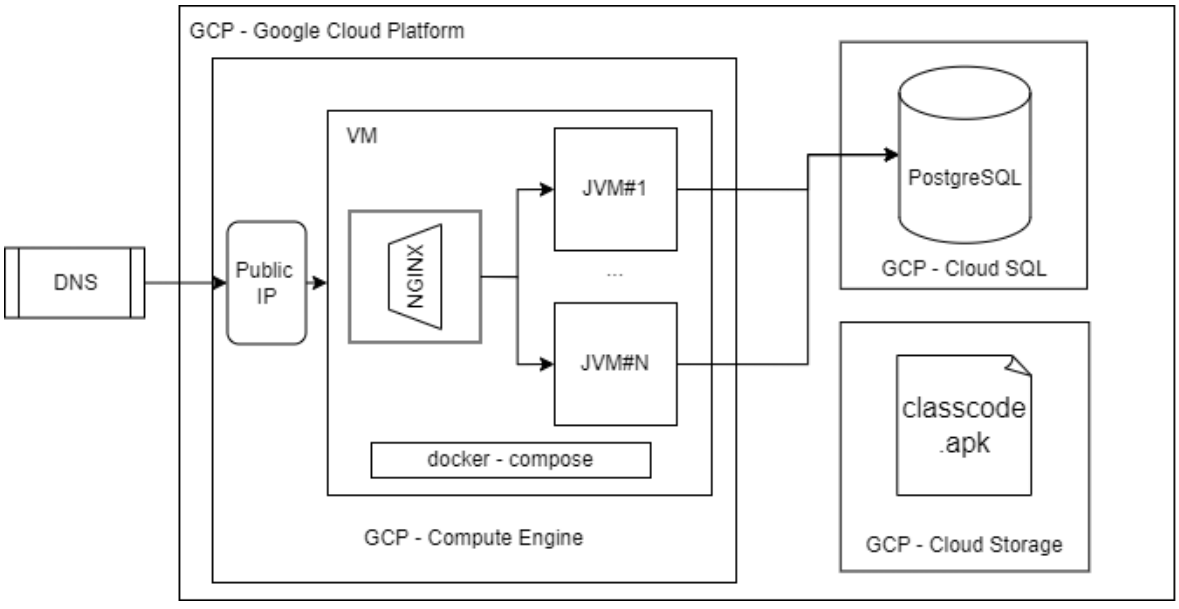

Figura 9: Diagrama da disponibilização dos serviços

A figura 9 representa o diagrama da disponibilização dos serviços que utiliza a *Google Cloud Platform* (GCP[53]). A GCP é uma plataforma de computação em nuvem oferecida pela Google, que oferece um conjunto de serviços e ferramentas que atendem necessidades de computação, armazenamento, desenvolvimento de aplicações e mais. Os três serviços da GCP usados foram:

- *Compute Engine* - Fornece máquinas virtuais (VMs) para executar aplicações e cargas de trabalho.
- *Cloud SQL* - Suporte a bases de dados geridas pelo GCP.
- *Cloud Storage* - Armazenamento de dados de forma flexível na nuvem, desde imagens e vídeos a arquivos.

A **Base de Dados** utiliza o serviço *Cloud* SQL, que possibilita a integração com outros serviços GCP e a gerência de forma automática também pela Google. As várias instâncias do **Serviço Backend** e do servidor NGINX são geridas pelo serviço *Compute Engine*. Este serviço disponibiliza uma máquina virtual onde através do ficheiro docker-compose é possível criar vários contentores para a execução independente de cada um dos componentes do sistema instanciados. Na máquina virtual do *Compute Engine*, além do ficheiro docker-compose estão presentes os certificados criados com a ajuda da ferramenta Certbot.

A criação dos contentores, definição de variáveis de ambiente necessárias ao sistema e a conexão privada entre a **Base De Dados** e as instâncias do **Serviço Backend** são feitas através de um *startup script*, criado na consola do GCP, que permite atribuir à inicialização da máquina virtual um conjunto de instruções. Para a criação dos contentores foi também criada uma *task Gradle* que executa o comando 'docker-compose up'. Desta forma, definindo o comando 'gradlew composeUp' no conjunto de instruções do *startup script* é possível disponibilizar tanto o **Serviço Backend** como a **Aplicação Web**. Caso se pretenda criar múltiplas instâncias do **Serviço Backend** é possível substituir o comando 'gradlew composeUp' pelo comando 'docker-compose up -build -scale spring-service=[número de instâncias]' no *startup script*.

Para associar ao endereço IP público da máquina virtual a um URL (classcode.i-on.live) foi criado um domínio DNS, *i-on.live*, através do *site* Name.com, permitindo assim expor o IP da máquina virtual e a **Aplicação Web**.

Em relação à **Aplicação Mobile**, foi gerado o ficheiro *apk* que foi armazenado no serviço *Cloud Storage*. Foi criado também um *blob* (*Binary Large Object*) do ficheiro *apk* que permite associar um *link* para o *download* do *apk* ao colocar o acesso ao *blob* público. Esse download está disponível na **Aplicação Web** para os professores na página principal.

5 Conclusões

Com a entrega da fase final do projeto foi possível criar um sistema capaz de auxiliar a jornada académica tanto de professores como alunos. Atingiram-se os objetivos pretendidos como a resolução dos problemas associados ao **GitHub Classroom**, ao processo de avaliação dos estudantes e a implementação de um sistema com um grande foco na melhoria da segurança no uso de dados sensíveis dos utilizadores, em relação a projetos anteriores. A arquitetura do sistema desenvolvido permitiu ainda a separação das funções de cada um dos seus componentes e a evolução isolada de cada um deles.

Foram abordados e aplicados vários temas que foram peças cruciais na construção do projeto; a implementação do **Serviço Backend** a partir da *framework Spring* e a criação de uma **API HTTP** *Hypermedia* que utiliza *Siren* e *Problem Json* como *media types* das respostas HTTP. O uso de uma API externa para o envio de emails para verificação de registos no sistema o que permitiu adquirir o conceito de *Outbox Pattern* e adicionar um outro ponto de segurança no sistema.

O processo de autenticação que envolveu o uso da *framework OAuth 2.0* juntamente com o **GitHub** e que se demonstrou ser um processo complexo deste a criação de um utilizador até ao seu registo e confirmação. O uso desta *framework* também teve impacto no desenvolvimento da **Aplicação Web**, como por exemplo, na definição dos endereços de redirecionamento no *endpoint* de *callback* da **API HTTP** e na abertura de janelas do *browser* para evitar a perca de estado da aplicação durante o processo de autenticação e de registo dos utilizadores. Na autenticação da **Aplicação Mobile** houve a necessidade de usar PKCE. O PKCE requer um verificador adicional que mitiga as vulnerabilidades de acesso indevido ao código a ser trocado pelo *token* de acesso no **GitHub**.

Fez-se também uso de *Custom Browser Tabs* porque permitiram obter transições ininterruptas entre a aplicação e o *web content*. O uso de *Android Key Stores* permitiu garantir a segurança e armazenamento da chave simétrica responsável pela encriptação e desencriptação do *token* de acesso e das *cookies* de sessão no dispositivo móvel dos professores. Adicionalmente o uso dessa chave está restrito ao uso de biometria para que o utilizador se possa autenticar. Para guardar informação sensível como *cookies* e o *token* de acesso encriptado utilizaram-se *Data Store Preferences*. Um marco importante alcançado foi a disponibilização de todo o sistema desenvolvido a partir dos serviços da GCP e do sistema de contentores Docker.

Existem ainda aspetos do projeto desenvolvido passíveis de serem melhorados. Entre eles está a sincronização entre as fontes de informação do **i-on ClassCode** e do **GitHub**, a criação de testes automáticos para a **Aplicação Mobile**, melhorias na verificação de erros e testabilidade da comunicação com o **Serviço Backend** e a **API HTTP** e a implementação de outras funcionalidades que dependerão do gosto pessoal

de cada professor para lecionar as suas cadeiras.

A criação e desenvolvimento do projeto até a esta fase final colocou o nosso conhecimento e capacidade de desenvolvimento de sistemas com uma arquitetura complexa em prova e forneceu-nos a possibilidade de evoluir as nossas competências técnicas como futuros engenheiros, deixando-nos assim satisfeitos com a qualidade do trabalho realizado.

Referências

[1] André Santos, João Magalhães e Ricardo Henriques. *i-on ClassCode*. 2023. URL: https://github.com/i-on-project/repohouse (acedido em 02/06/2023).

[2] GitHub. 2008. URL: https://github.com (acedido em 02/06/2023).

[3] Diogo Sousa, Tiago David e João Moura. *i-on CodeGarten*. 2021. URL: https://github.com/i-on-project/codegarten (acedido em 02/06/2023).

[4] Afonso Machado e Martim Francisco. *i-on Teams*. 2022. URL: https://github.com/i-on-project/teams (acedido em 02/06/2023).

[5] GitHub Classroom. 2015. URL: https://classroom.github.com/ (acedido em 02/06/2023).

[6] GitHub. 2023. URL: https://docs.github.com/en/rest (acedido em 02/06/2023).

[7] SendGrid. 2023. URL: https://sendgrid.com/solutions/email-api/ (acedido em 02/06/2023).

[8] Spring Boot. 2023. URL: https://spring.io/projects/spring-boot (acedido em 02/06/2023).

[9] Kotlin. 2023. URL: https://kotlinlang.org/ (acedido em 02/06/2023).

[10] OkHttp. 2023. URL: https://square.github.io/okhttp/ (acedido em 02/06/2023).

[11] Jackson. 2023. URL: https://github.com/FasterXML/jackson (acedido em 02/06/2023).

[12] JSON. 2023. URL: https://www.json.org/json-en.html (acedido em 02/06/2023).

[13] Java. 2023. URL: https://www.oracle.com/java/ (acedido em 02/06/2023).

[14] Siren. 2023. URL: https://github.com/kevinswiber/siren (acedido em 02/06/2023).

[15] PostgreSQL. 2023. URL: https://www.postgresql.org/ (acedido em 02/06/2023).

[16] JDBI. 2023. URL: https://jdbi.org/ (acedido em 02/06/2023).

[17] JDBC. 2023. URL: https://www.oracle.com/database/technologies/appdev/jdbc.html (acedido em 02/06/2023).

[18] TypeScript. 2015. URL: https://www.typescriptlang.org/ (acedido em 02/06/2023).

[19] ReactJS. 2023. URL: https://legacy.reactjs.org/ (acedido em 02/06/2023).

[20] JavaScript. 2023. URL: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript (acedido em 02/06/2023).

[21] Webpack. 2023. URL: https://webpack.js.org/ (acedido em 02/06/2023).

[22] Single Page Application. 2023. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (acedido em 02/06/2023).

[23] Google. 2019. URL: https://developer.android.com/jetpack/compose (acedido em 01/06/2023).

[24] Google. 2019. URL: https://github.com/google/gson (acedido em 01/06/2023).

[25] Ryan J. McDonough. 2012. URL: https://damnhandy.github.io/Handy-URI-Templates/apidocs/ (acedido em 01/06/2023).

[26] Joe Gregorio. 2012. URL: https://www.rfc-editor.org/rfc/rfc6570 (acedido em 01/06/2023).

[27] OAuth 2.0. 2023. URL: https://oauth.net/2/ (acedido em 02/06/2023).

[28] Python. 2023. URL: https://www.python.org/ (acedido em 02/06/2023).

[29] Node.js. 2023. URL: https://nodejs.org/en (acedido em 02/06/2023).

[30] i-on. 2020. URL: https://github.com/i-on-project (acedido em 02/06/2023).

[31] Spring MVC. 2023. URL: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html (acedido em 02/06/2023).

[32] Google. 2017. URL: https://www.rfc-editor.org/rfc/rfc6906 (acedido em 01/06/2023).

[33] Mehmet Ozkaya. «Outbox Pattern for Microservices Architectures». Em: *Design Microservices Architecture with Patterns Principles* (2021). URL: https://medium.com/design-microservices-architecture-with-patterns/outbox-pattern-for-microservices-architectures-1b8648dfaa27.

[34] Spring Scheduler. 2023. URL: https://spring.io/guides/gs/scheduling-tasks/ (acedido em 04/06/2023).

[35] Ralph Marvin Addo. «Mastering SOLID Principles: Dependency Inversion Principle (DIP)». Em: *The Coding Insider* (2023). URL: https://www.linkedin.com/pulse/mastering-solid-principles-dependency-inversion-dip-ralph-marvin-addo/.

[36] GitHub Developer Settings. 2023. URL: https://docs.github.com/en/apps/oauth-apps (acedido em 01/06/2023).

[37] CSRF. 2023. URL: https://owasp.org/www-community/attacks/csrf (acedido em 01/06/2023).

[38] XSS. 2023. URL: https://owasp.org/www-community/attacks/xss/ (acedido em 01/06/2023).

[39] JUnit. 2023. URL: https://junit.org/ (acedido em 01/06/2023).

[40] React Router. 2023. URL: https://reactrouter.com/en/main (acedido em 01/06/2023).

[41] Playwright. 2023. URL: https://playwright.dev/ (acedido em 04/06/2023).

[42] Google. 2013. URL: https://developer.android.com/training/articles/keystore (acedido em 02/06/2023).

[43] Google. 2020. URL: https://developer.android.com/topic/libraries/architecture/datastore#preferences-datastore (acedido em 02/06/2023).

[44] Google. 2015. URL: https://www.rfc-editor.org/rfc/rfc7636 (acedido em 01/06/2023).

[45] Custom Browser Tabs. 2023. URL: https://developer.chrome.com/docs/android/custom-tabs/ (acedido em 01/06/2023).

[46] Google. 2017. URL: https://www.rfc-editor.org/rfc/rfc8252 (acedido em 01/06/2023).

[47] Google. 2017. URL: https://www.rfc-editor.org/rfc/rfc8252#appendix-B.2 (acedido em 01/06/2023).

[48] Docker. 2023. URL: https://www.docker.com/ (acedido em 04/06/2023).

[49] Docker. 2023. URL: https://docs.docker.com/compose/ (acedido em 04/06/2023).

[50] NGINX. 2023. URL: https://www.nginx.com/ (acedido em 04/06/2023).

[51] Certbot. 2023. URL: https://certbot.eff.org/ (acedido em 08/07/2023).

[52] Let's Encrypt. 2023. URL: https://letsencrypt.org/ (acedido em 08/07/2023).

[53] Google. 2023. URL: https://docs.docker.com/compose/ (acedido em 01/07/2023).

Anexos

A Imagem do Modelo de Dados

A imagem representa o modelo de dados do sistema e pode ser encontrada em anexo na pasta *images*.

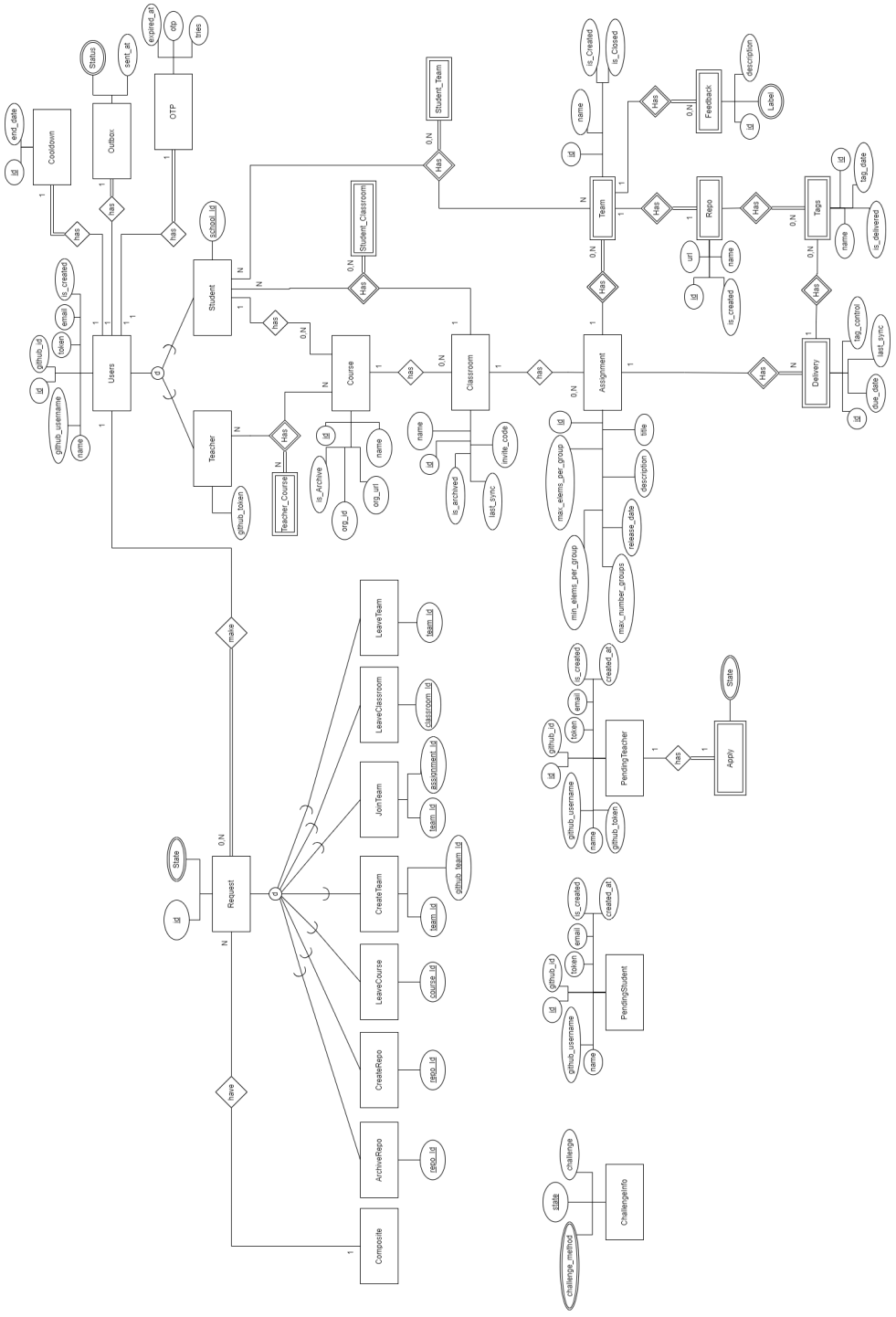

Figura 10: Modelo de Dados