



UNIVERSIDADE
LUSÓFONA

Drop Project para Android (Back-end)

Trabalho Final de curso

Relatório Final

Nome do Aluno: Diogo Araújo

Nome do Orientador: Miguel Tavares

Trabalho Final de Curso | LEI | 09/09/2022

www.ulusofona.pt

Direitos de cópia

(Drop Project para Android), Copyright de *(Diogo Araújo)*, ULHT.

A Escola de Comunicação, Arquitectura, Artes e Tecnologias da Informação (ECATI) e a Universidade Lusófona de Humanidades e Tecnologias (ULHT) têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objectivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

Resumo

O Drop Project (DP) é um Automated Assessment Tool (AAT) para as linguagens de programação Java e Kotlin que foi desenvolvido na universidade lusófona. Um AAT é uma ferramenta ou aplicação de software que ajuda a testar/avaliar software. Desse mesmo modo, o DP permite aos alunos testarem os seus projetos ou exercícios contra uma série de testes unitários.

No entanto, o DP está apenas capacitado para executar projetos Maven que o impede de ser utilizado por outras disciplinas, como é o caso de Computação Móvel onde o projeto principal é desenvolvido em Android com recurso à linguagem de programação Kotlin. De momento, os alunos enviam os trabalhos para o professor através do Moodle, onde este os irá instalar um a um num emulador de modo a proceder com a avaliação. Seguidamente, estes trabalhos terão de ser desinstalados novamente um a um, o que torna este processo muito pouco eficiente.

Este trabalho tem como objetivo alcançar uma solução alternativa à utilizada atualmente que, por se tratar de uma tarefa inteiramente manual, acaba por não ser eficiente. Este projeto retrata uma das duas partes deste sistema (back-end) sendo que a outra (front-end) está a ser desenvolvida noutro trabalho por outro aluno.

Abstract

Drop Project (DP) is an Automated Assessment Tool (AAT) for the programming languages Java and Kotlin that was developed in “Universidade Lusófona”. An AAT is a tool or software application that helps to test Web, Desktop or mobile applications. That being said, DP allows students to test their projects and exercises against a set of unitary tests.

However, DP is only able to execute Maven projects which impedes it from being used by other subjects, as is the case for “Computação Móvel” where the main project is developed on Android using the programming language Kotlin. Currently, students deliver their work to the teacher using Moodle, where he will install them one by one on an emulator so that they may proceed with the evaluation. After that, the works will have to be uninstalled again one by one, which turns it into a not very efficient process.

This work has the objective of reaching an alternative solution to the one being currently used, which by being an entirely manual task ends up not being very efficient. This project portrays one of two parts of this system (back-end), it being that the other (front-end) is being developed in another work by another student.

Índice

Resumo	iii
Abstract	iv
Índice	v
Lista de Figuras	vi
Lista de Tabelas	vii
1 Identificação do Problema	1
2 Levantamento e Análise de Requisitos	3
3 Viabilidade e Pertinência	12
4 Solução Desenvolvida	13
5 Benchmarking	23
6 Método e Planeamento	26
7 Resultados	30
8 Conclusão e Trabalhos Futuros	34
Bibliografia	35
Anexos	36
Glossário	37

Lista de Figuras

Figura 1 – Arquitetura do Sistema	13
Figura 2 – Diagrama da Atividade de Gerar um APK	14
Figura 3 - Imagem sobre Estrutura de Projeto	17
Figura 4 - Variáveis sobre Trabalho no Código	17
Figura 5 - Menu dropdown para receber <i>Engine</i>	18
Figura 6 - Código para importar tooling API	18
Figura 7 - Código para condição que escolhe <i>Invoker</i>	19
Figura 8 - Código para validação de trabalhos <i>Gradle</i>	20
Figura 9 – Código para compilação através de <i>Gradle</i>	21
Figura 10 – Código para compilação através de <i>Android</i>	22
Figura 11 – Erro obtido para compilação em <i>Android</i>	22
Figura 12 – Código para criação de submissão	23
Figura 13 – Código para leitura de testes unitários	24
Figura 14 – Variáveis ambiente para <i>Maven</i> e <i>SDK Android</i>	25
Figura 15 – Utilização do <i>SDK Android</i>	25
Figura 16 – Utilização do <i>Appium</i> em várias Plataformas	27
Figura 17 – Benefícios de <i>Robotium</i>	28
Figura 18 – Calendário da Solução no formato <i>Gantt</i>	30
Figura 19 – Ecrã de Login	34
Figura 20 – Ecrã Principal	35
Figura 21 – Ecrã de um Trabalho	35
Figura 22 – Ecrã de criação de um Trabalho	36
Figura 23 – Ecrã de gerir Trabalhos	37
Figura 24 – Visualizar detalhes de Trabalho	37
Figura 25 – Lista inicial de Trabalhos	38
Figura 26 – Criação do motor e da linguagem	38
Figura 27 – Ecrã principal com trabalho adicionado	39
Figura 28 – Resultado de testes passados	39
Figura 29 – Resultado de testes falhados	40
Figura 30 – Submissão a um trabalho <i>Android</i>	40
Figura 31 – Calendário maior da Solução no formato <i>Gantt</i>	44

Lista de Tabelas

Tabela 1 – Requisitos e o seu estado	11
Tabela 2 – Quantidade de requisitos para cada estado	11
Tabela 3 – Benchmarking entre Soluções	29
Tabela 4 – Tabela associada ao Calendário da Solução	32

1 Identificação do Problema

A entrega e verificação de projetos é um processo relativamente complexo, sendo por vezes necessário instalar e desinstalar cada trabalho manualmente. Este problema evidencia-se na unidade curricular de Computação Móvel onde o pretendemos mitigar com recurso ao Drop Project (DP).

O DP é um sistema de avaliação automática desenvolvido na universidade lusófona estando a ser utilizado em algumas unidades curriculares como Fundamentos de Programação e Linguagens de Programação 2. A avaliação é efetuada através de testes unitários realizados em Java ou Kotlin com recurso ao Maven, onde após a sua execução o aluno obtém um relatório sobre o estado do seu trabalho. No entanto, a disciplina de Computação Móvel não consegue tirar partido desta ferramenta uma vez que o projeto principal é desenvolvido em Android que, para além de estar suportado noutro gestor de dependências que não o Maven (Gradle), também não possui nenhuma compatibilidade com o SDK do Android.

Num caso ideal, a solução atual que o DP oferece seria mantida. No entanto, existem várias limitações que impedem a sua utilização. Os testes android de instrumentalização da [framework Android](#) necessitam de um emulador para serem executados e, acontece que, num universo de vários alunos esta solução torna-se inviável pela limitação de recursos (memória, processador, entre outros) que o processo causa. Isto devido ao facto de que serão vários alunos, em paralelo, a executar vários emuladores que por sua vez estarão a testar os seus trabalhos. Desse modo, o formato dos testes android não poderão ser utilizados pelo DP sendo só possível a utilização dos testes unitários que já são utilizados para testar os projetos em Java e Kotlin.

Esta solução foi criada a partir do DP e tem como âmbito servir como uma ponte entre os trabalhos dos alunos, que foram entregues no website, e o professor, que os iria verificar. Tem como objetivo receber trabalhos Android, através do gestor de dependências Gradle, e enviá-los para um Professor para avaliação. Estes trabalhos, estando no formato Application Package (APK), são entregues e avaliados das mesmas formas que as outras linguagens com a inclusão de também poderem ser compiladas.

A solução final demonstra uma concretização deste objetivo apresentando dois formatos de compilação adicionais, o Gradle e o Android, ao que é utilizado no momento pelo DP, o Maven. Estes são utilizados como motores (*engine*) pelo DP e são o método de compilação de submissões, que poderão utilizar a linguagem Java ou Kotlin, e que serão testadas através de testes unitários ou manualmente. Isto vai de acordo com o âmbito do projeto cujo objetivo era a introdução do Gradle, com o apoio auxiliar do SDK Android, para a compilação de trabalhos Android e, desse modo, a facilitação da entrega desses trabalhos.

Nota Prévia:

Ao longo do relatório serão utilizados os termos “trabalho” e “submissão” para referir a alguns conteúdos do DP. O trabalho, quando utilizado, refere-se aos enunciados disponibilizados pelos Professores. Estes podem ser criados através de um formulário disponibilizado quando a conta tem permissões de Professor. As submissões são os projetos entregues pelos alunos a um trabalho. Estes poderão ser avaliados automaticamente, através de testes, ou manualmente.

2 Levantamento e Análise de Requisitos

O levantamento de requisitos foi efetuado com base nos objetivos previamente detalhados para o DP e com base nas funcionalidades planeadas.

Estes podem ser divididos em duas seções, sendo elas os requisitos funcionais e os requisitos não funcionais. Os requisitos funcionais são os que definem os serviços que o sistema deve fornecer, sendo que isso pode ser o seu comportamento a certas situações ou a sua reação a certos inputs. Os requisitos não funcionais são os que estão relacionados a restrições nos serviços ou a propriedades do sistema. Podem ser o tempo de resposta de um serviço ou até a especificação da tecnologia a ser utilizada.

A implementação dos requisitos será efetuada no código do DP e terá como foco os trabalhos e as submissões efetuadas como também a sua compilação. Poderá ser testada através do processo de criação de trabalhos e a entrega subsequente a estes mesmos. A criação de um trabalho irá utilizar as novas variáveis, sendo essas o motor, que poderá ser o Maven, o Gradle ou o Android, e a linguagem utilizada, que poderá ser o Java ou o Kotlin, que poderão ser selecionadas durante a criação. As submissões irão ser compiladas através do gestor escolhido e, caso o Professor o tenha selecionado, deverão ser efetuados testes à submissão.

Requisitos Funcionais

RF1 - Gestor de Dependências no Sistema

Pré-Condição:

N/A

Requisito:

O formulário da criação de um trabalho permite escolher o motor associado, que neste caso poderá ser Maven. Deverá ser adicionado a estas opções o gestor Gradle, para que se possa efetuar a compilação através dele, como também o Android, de modo a se conseguir utilizar o SDK Android.

Pós-Condição:

O sistema recebe e guarda o valor do motor através de uma variável.

Estado: Implementado

RF2 - Linguagem de Programação no Sistema

Pré-Condição:

N/A

Requisito:

O formulário da criação de um trabalho permite escolher a linguagem de programação associada, que neste caso poderá ser Java ou Kotlin. Esta opção irá determinar a linguagem que será utilizada nos trabalhos e submissões.

Pós-Condição:

O sistema recebe e guarda o valor da linguagem de programação associada ao trabalho.

Estado: Implementado

RF3 - Criação de Trabalhos Gradle

Pré-Condição:

N/A

Requisito:

O sistema deve permitir a criação de um trabalho utilizando o gestor de dependências Gradle da mesma forma que um em Maven. O processo deve validar o trabalho entregue pelo utilizador de modo a certificar que pode ser criado.

Pós-Condição:

A submissão deve ser testada através do gestor de dependências com os testes associados ao trabalho.

Estado: Implementado

RF4 - Testes no Gestor de Dependências Gradle

Pré-Condição:

O sistema deve ter um trabalho criado com o gestor de dependências Gradle e que utilize testes unitários na avaliação.

Requisito:

As submissões efetuadas a um trabalho que utiliza as linguagens Java ou Kotlin, mesmo utilizando o gestor de dependências Gradle, devem ser avaliadas através dos testes.

Pós-Condição:

As submissões feitas pelos alunos, num trabalho com o gestor Gradle, devem poder ser avaliadas através de testes unitários.

Estado: Implementado

RF5 - Integração de SDK Android

Pré-Condição:

N/A

Requisito:

O SDK Android deverá ser integrado na plataforma de modo a fazer com que seja possível efetuar testes unitários em submissões para trabalhos Android e gerar os APKs das submissões.

Pós-Condição:

Um trabalho que escolha o Android irá utilizar as tarefas relacionadas com o SDK Android.

Estado: Implementado

RF6 - Utilização de Gradle para trabalhos Android

Pré-Condição:

N/A

Requisito:

Qualquer trabalho que utilize o Android só poderá ser compilado através do gestor de dependências Gradle. Este irá utilizar as tarefas do Gradle da mesma forma que os outros trabalhos Gradle.

Pós-Condição:

Um trabalho que escolha o Android só poderá utilizar o Gradle como gestor de dependências.

Estado: Implementado

RF7 - Compilação da Submissão para APK

Pré-Condição:

O sistema deve ter um trabalho criado e uma submissão deve ter sido feita a esse projeto. O trabalho deve ser Android.

Requisito:

Depois de uma submissão ter sido efetuada a um trabalho Android, e ter sido compilada com sucesso, o sistema deve conseguir gerar o APK resultante do processo e associá-lo ao trabalho.

Pós-Condição:

O APK relativo à submissão deve ser gerado e associado ao trabalho.

Estado: Parcialmente Implementado

RF8 - Devolver APK de uma Submissão

Pré-Condição:

O sistema deve ter um trabalho criado e uma submissão deve ter sido feita a esse projeto. O trabalho deve ser Android.

Requisito:

Uma submissão que tenha sido efetuada a um trabalho Android, e que compilado com sucesso, terá um APK resultante. Este deve poder ser devolvido ao selecionar a submissão.

Pós-Condição:

O APK relativo à submissão deve ser devolvido ao professor.

Estado: Não Implementado

RF9 - Listar Submissões de um Trabalho - Webservice

Pré-Condição:

O sistema deve ter um trabalho criado e uma submissão deve ter sido feita a esse projeto.

Requisito:

O utilizador, ao seleccionar um trabalho, deve começar por visualizar a sua página de detalhes. A partir desta, deve ter uma opção onde deve conseguir visualizar todas as submissões efetuadas a este trabalho. Isto deve ser feito através do webservice.

Pós-Condição:

Submissões feitas ao trabalho devem poder ser visualizadas nos detalhes.

Estado: Não Implementado

RF10 - Listar Trabalhos de um Utilizador - Webservice

Pré-Condição:

O sistema deve ter um trabalho criado.

Requisito:

Todos os trabalhos associados a um utilizador devem ser listados depois de ele efetuar o login. Isto deve ser feito através do webservice.

Pós-Condição:

Trabalhos associados a um utilizador devem ser listados ao entrar na plataforma.

Estado: Não Implementado

RF11 - Apresentar detalhes de uma Submissão - Webservice

Pré-Condição:

O sistema deve ter um trabalho criado e uma submissão deve ter sido feita a esse projeto.

Requisito:

Uma submissão, ao ser selecionada, deve mostrar os seus detalhes. Estes podem ser os alunos que entregaram a submissão, a data e o tempo em que foi entregue e o resultado dos testes unitários efetuados. Isto deve ser feito através do webservice.

Pós-Condição:

Detalhes de uma submissão visíveis após selecioná-la na listagem.

Estado: Não Implementado

Requisitos Não Funcionais

RNF1 - Webservices com Spring MVC

Os webservices desenvolvidos devem ser integrados no projeto sobre a tecnologia Spring MVC, no qual o DP se baseia.

RNF2 - Webservices do tipo REST

Os webservices desenvolvidos devem ser do tipo REST seguindo a nomenclatura Javascript Object Notation (JSON).

RNF3 - Pedidos associados a Webservices

Os webservices desenvolvidos só responderão a pedidos devidamente autenticados pelo DP.

RNF4 - Desenvolvimento de Webservices

Os webservices serão desenvolvidos na linguagem de programação Kotlin.

RNF5 - Restrição de Testing a projetos Android

Os trabalhos, cujo tipo de projeto seja Android, não poderão ser associados testes devido à falta de recursos no servidor para os emuladores. Desse modo, só os trabalhos que sejam do tipo Kotlin ou Java podem ter testes associados.

Devido à quantidade de requisitos, foram criadas duas tabelas adicionais de modo a resumir a sua implementação. Nestas é possível visualizar o id associado a um requisito, o seu título e o seu estado atual como também o número de requisitos para cada estado.

Tabela 1 - Requisitos e o seu estado.

ID	Título	Estado
RF1	Gestor de Dependências no Sistema	✓
RF2	Linguagem de Programação no Sistema	✓
RF3	Criação de Trabalhos Gradle	✓
RF4	Testes no Gestor de Dependências Gradle	✓
RF5	Integração de SDK Android	✓
RF6	Utilização de Gradle para trabalhos Android	✓
RF7	Compilação da Submissão para APK	X / ✓
RF8	Devolver APK de uma Submissão	X
RF9	Listar Submissões de um Trabalho - Webservice	X
RF10	Listar Trabalhos de um Utilizador - Webservice	X
RF11	Apresentar detalhes de uma Submissão - Webservice	X

Tabela 2 - Quantidade de requisitos para cada estado.

Estado	Quantidade
Implementado	6
Parcialmente Implementado	1
Não Implementado	4

3 Viabilidade e Pertinência

Com a conclusão do projeto, a solução final do Drop Project consegue apoiar a entrega de trabalhos feitos para Android que, tal como foi explicado anteriormente, era o objetivo principal do trabalho.

Idealmente, esta solução poderia ser utilizada para o próximo ano na disciplina de Computação Móvel. No momento, com a utilização do Moodle, é necessário ao Professor efetuar o download dos trabalhos de cada grupo. Após este download, será necessário enviá-los para um emulador, instalar cada um, efetuar a sua avaliação e depois desinstalá-los. Este processo, como pode ser visto, é cansativo e desnecessariamente excessivo. Com a substituição do moodle pela solução, seria possível automatizar quase todo o processo, permitindo uma avaliação dos trabalhos mais eficiente pelo Professor e um aumento da produtividade. Isto seria devido às funcionalidades adicionadas à solução final que lhe permitem efetuar a compilação de trabalhos Gradle e Android como também guardar as submissões enviadas pelos alunos para poderem ser instalados depois.

No entanto, a implementação do projeto não necessita de acabar no seu estado atual. Com a adição do gestor de dependências Gradle foi adicionada a possibilidade de serem adicionados ainda mais gestores de dependências que por si irão permitir mais tipos de projetos. Ao implementarmos estes, só necessitamos de nos certificar que a plataforma está preparada para os receber e, caso seja possível, passá-los por testes unitários. Caso a implementação dos testes não seja possível, então pode funcionar como um serviço de recepção e instalação para esse tipo de projeto.

Assim, o Drop Project pode continuar a apoiar docentes após a sua implementação em Computação Móvel, sendo que se pode melhorar e incluir mais tecnologias / linguagens ao longo das necessidades das disciplinas.

4 Solução Desenvolvida

A solução está dividida em dois componentes. O *front-end*, que é uma aplicação Android a ser desenvolvida por um colega de outro TFC em paralelo, e o *back-end*. Para a conclusão desta foi utilizada matéria relevante à disciplina de Linguagens de Programação (Kotlin) sendo que o objetivo principal está na área do *back-end* (Maven e Gradle).

A aplicação móvel (DP) tem como objetivo automatizar as tarefas que no momento o professor das práticas de Computação Móvel trabalha com. Estas incluem instalar o trabalho de um aluno / grupo, consultar os detalhes de uma entrega e desinstalar ou executar um determinado trabalho num dispositivo Android. Existem outras duas funcionalidades que são complementares, sendo que uma delas permite a instalação de todos os trabalhos entregues e a outra a desinstalação de todos. Note-se que, a funcionalidade que permite a desinstalação geral dos trabalhos é particularmente útil pois de outra forma o professor teria de desinstalar os trabalhos um a um. Na solução desenvolvida estas tarefas não estão presentes, sendo que a solução serve só como uma base para a possível inclusão destas no futuro.

Arquitetura do Sistema

O *back-end* é o servidor que irá tratar de receber os trabalhos dos alunos, associá-los ao projeto e fornecer as operações de instalação e desinstalação. Terá igualmente o papel de guardar os trabalhos. O *front-end* será a interface que o utilizador irá visualizar e onde poderá iniciar as operações previamente ditas.

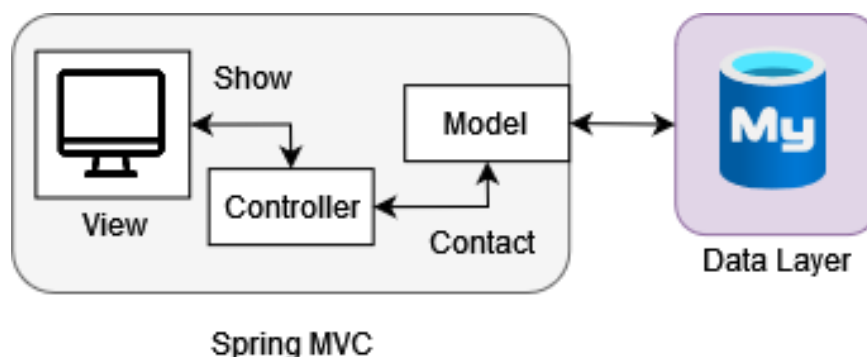


Figura 1 - Arquitetura do Sistema.

A entrega dos trabalhos será iniciada no *front-end* onde o professor irá criar um projeto para que eles possam ser entregues. Durante a criação, poderá selecionar o formato de entrega que irá estar à espera de receber, através do tipo de projeto, o gestor de dependências que irá utilizar e adicionar mais informações relativas ao projeto. Poderá verificar todas as submissões através de uma lista onde pode selecionar cada uma individualmente e visualizar informação sobre quem a entregou, a data e a hora de quando foi entregue e duas opções para a sua instalação e/ou desinstalação. Ao selecionar uma delas, irá iniciar uma ligação com o *back-end* para encontrar o trabalho na base de dados e efetuar a opção escolhida. Se o tentar instalar, então irá receber o trabalho no formato APK, onde o poderá executar num emulador. Um APK (Android Package) é um arquivo utilizado que pode ser comparado com os arquivos do Windows, tal como o “.exe”, só que para Android. A avaliação irá ocorrer no *front-end* (Mobile) onde o professor irá efetuar a verificação do trabalho. Após esta verificação, poderá desinstalar o trabalho através da outra opção.

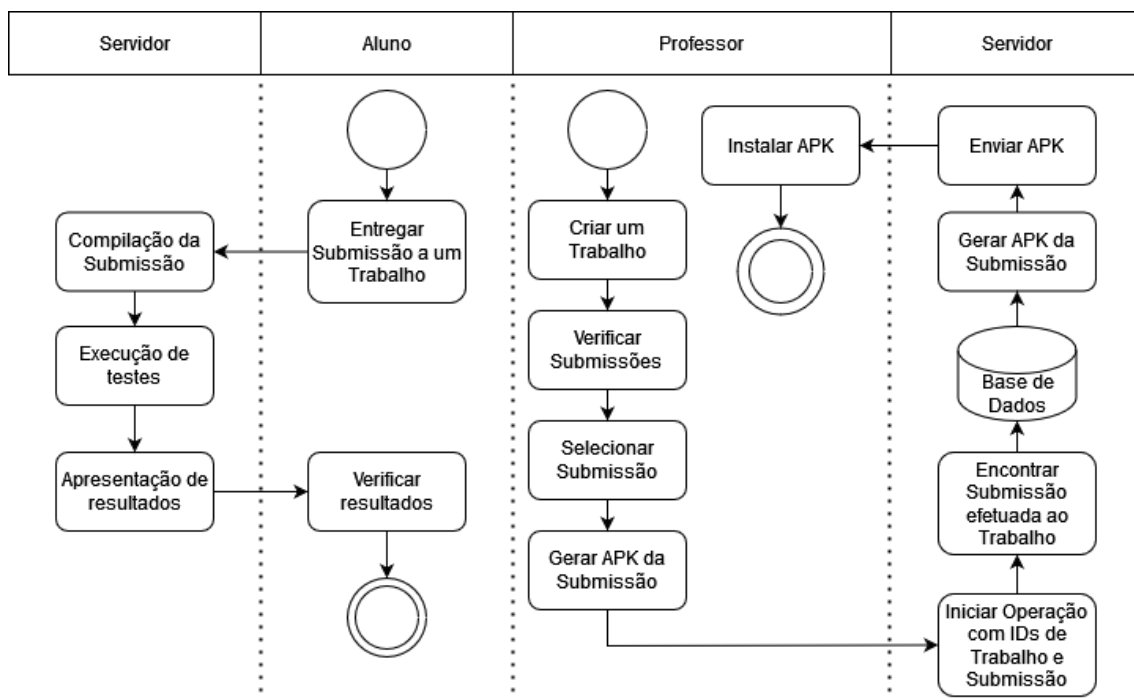


Figura 2 - Diagrama da Atividade de Gerar um APK.

O *back-end* está desenvolvido em Kotlin e tem como função receber os projetos dos alunos e os encaminhar para o professor. Foi utilizado em conjunto com o gestor de dependências Gradle, que fornece métodos de *build* e *deployment*, mas não é o único que será utilizado para os projetos. Desse modo, o DP teve de conseguir acomodar outras opções. Foi criada uma variável do tipo *enum* para guardar o tipo de gestor que seria utilizado, sendo que poderia ser ou Maven ou Gradle, que são as mesmas opções dadas ao professor durante a criação de um trabalho. Um *enum* é um tipo especial de dados que só permite à variável ser um dos valores pré-definidos. Este tipo é utilizado de novo para guardar a linguagem utilizada no projeto, sendo essa outra escolha que o professor efetua na criação de um trabalho e que pode ser os valores Java, Kotlin ou Android. A opção para ser Android só está disponível caso o gestor de dependências escolhido seja o Gradle. Qualquer submissão efetuada a este trabalho será compilada através do gestor de dependências escolhido e deverá ser do tipo de projeto associado. Caso não seja, o utilizador receberá uma mensagem de erro.

Tecnologias Utilizadas

1. Kotlin

O Kotlin é uma linguagem multiplataforma sem um propósito específico, embora seja a preferida para o desenvolvimento de aplicações Android. Foi desenvolvida para trabalhar com Java e as suas bibliotecas. Foi previamente aprendida na disciplina de Fundamentos de Programação e, neste projeto, foi utilizada para escrever o *back-end* onde iremos receber os projetos dos alunos e encaminhá-los para o professor.

2. API REST

O *Representational State Transfer*, melhor conhecido como REST, vai ser a conexão entre o *back-end* e o *front-end*. É um estilo de arquitetura de software que define um conjunto de restrições a serem usadas durante a criação de *web services*. Um *web service* é uma solução utilizada na comunicação entre aplicações e sistemas diferentes. Os *web services RESTful*, sendo esses os que seguem o estilo arquitetural REST, permitem que os sistemas incluídos no processo acessem e manipulem representações textuais de recursos usando um conjunto uniforme e predefinido de operações.

3. Maven

O Apache Maven, ou Maven, é uma ferramenta de automatização de compilação. Utiliza-se em dois aspectos da construção de software: como o software é construído e as suas dependências. O Maven utiliza um ficheiro XML para descrever o projeto/software que está a ser construído, as suas dependências e componentes, a ordem de construção, os seus diretórios e os *plugins* necessários. Também é inicializado com processos previamente definidos, tal como a compilação do código e o seu packaging. Caso seja pedido, o Maven efetua o download dinâmico de bibliotecas Java e de plugins a partir de um repositório, podendo um desses ser o Maven 2 repositório central.

4. Gradle

O Gradle é uma ferramenta de automatização de compilação para várias linguagens no desenvolvimento de *software*. Controla o processo através de tarefas de compilação, *packaging*, *testing* e publicação que podem ser corridas ao mesmo tempo. Também recolhe dados estatísticos sobre a utilização de bibliotecas de software. O Gradle constrói nos conceitos do Maven, introduzindo uma linguagem específica ao domínio baseada no Kotlin e no Groovy em contraste com o XML utilizado na configuração de projetos no Maven.

Resultado Final

A solução final foi criada com base no código disponibilizado pelo Professor Pedro Alves do Drop Project. Este código pode ser encontrado no [GitHub](#) e é open-source, completo com um *README* que descreve o seu processo de utilização. A partir deste repositório foi criado outro [GitHub](#) que contém o código da solução final.

O primeiro passo, após realizar um fork do código do DP, foi de entender a forma como funcionava. Deste modo foi utilizado o *README* como guia para entender como o DP funcionava e uma imagem feita pelo Professor que explica o que cada *folder* contém. Após perceber as funcionalidades do DP foi necessário percorrer os diretórios à procura de ficheiros que pudessem ser importantes para a continuação do trabalho. Este processo envolveu mais trabalho da minha parte por ser a primeira vez que trabalhei num projeto desta escala e não ter estado envolvido na criação inicial.

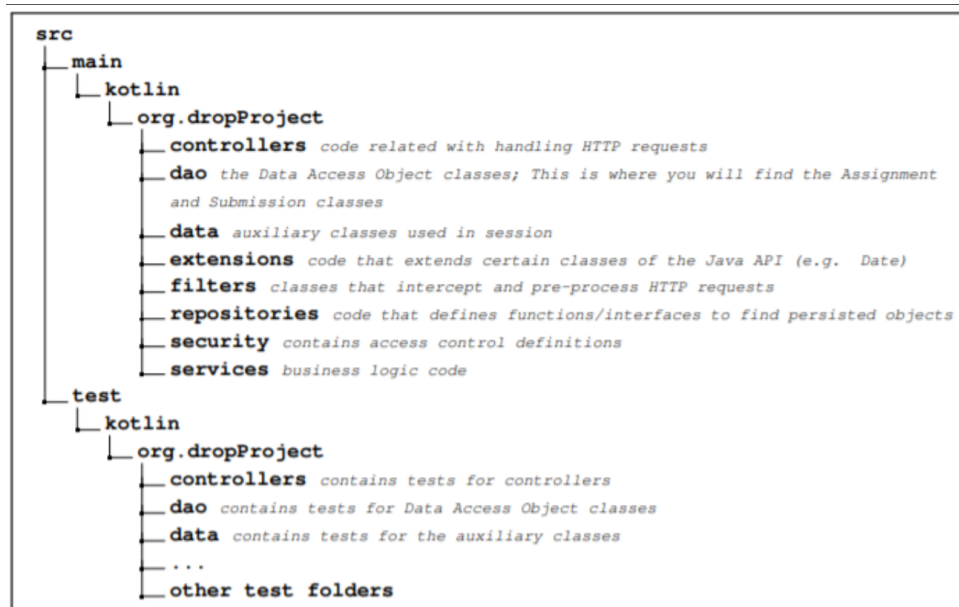


Figura 3 - Imagem sobre Estrutura de Projeto.

Após a verificação e entendimento do código, o próximo passo era a preparação do DP para os novos objetivos do software, sendo esses a utilização de um novo compilador e a adição de uma nova linguagem. Para efetuar isto alterei o código associado aos trabalhos disponibilizados pelos Professores, que têm o nome de *Assignment* no código, e adicionei duas variáveis. A primeira variável, o *Engine*, irá conter o gestor de dependências escolhido para o trabalho. A segunda variável, a *Language*, irá conter a linguagem de programação que será utilizada nos trabalhos.

```

/**
 * NEW: Enum representing the engine that Drop Project supports.
 */
enum class Engine {
    MAVEN, GRADLE, ANDROID
}

/**
 * NEW: Enum representing the programming languages that Drop Project
 * supports.
 */
enum class Language {
    JAVA, KOTLIN
}
  
```

Figura 4 - Variáveis sobre Trabalho no Código.

Com as variáveis criadas, e servindo como um contentor de dados, era necessário conseguir receber esses dados. Desse modo tivemos que alterar o ficheiro “assignment-form.html”, que representa o *layout* do formulário para criar um trabalho, e adicionar uma nova opção através de um menu dropdown. O valor do menu será recebido na variável *Engine* e só poderá ser os valores disponíveis que foram permitidos, estes sendo o Maven, o Gradle e o Android.

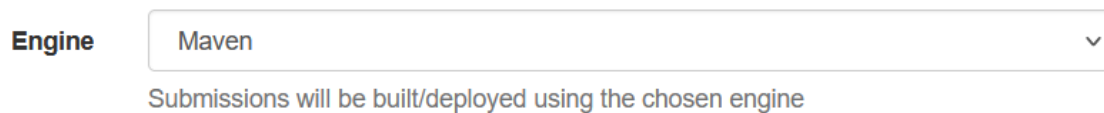


Figura 5 - Menu dropdown para receber *Engine*.

O passo seguinte, após a criação da base do projeto, foi a implementação do Gradle ao DP. Esta parte foi onde a maioria do esforço foi efetuado assim como o maior tempo. Isto deve-se ao facto de ser a primeira vez que estive a trabalhar num projeto que não tenha sido inicialmente criado por mim, especialmente deste tamanho, como também de ser a primeira vez que trabalhei com um gestor de dependências, sendo que estes funcionam de uma forma muito diferente ao que se está habituado no curso especialmente quando tratado num formato *third party*. Este formato necessita da adição de uma ferramenta para conseguir executar os comandos / tarefas fora da linha de comandos. Este é o *tooling* API que nos permite executar tarefas do Gradle através do código e que é importado através do ficheiro pom.xml. A inicialização do Gradle em si, e desse modo do “build.gradle”, só é efetuado no trabalho que o Professor irá enviar, sendo que um exemplo deste trabalho poderá ser encontrado no [GitHub](#) associado ao projeto.

```
<!-- Tooling API to run tasks in 3rd party -->
<!--https://mvnrepository.com/artifact/org.gradle/gradle-tooling-api -->
<dependency>
  <groupId>org.gradle</groupId>
  <artifactId>gradle-tooling-api</artifactId>
  <version>7.4.2</version>
</dependency>
```

Figura 6 - Código para importar *tooling* API.

A inclusão do *tooling API* permite a utilização do Gradle para a compilação de projetos, sendo este um passo necessário para finalizar a solução. No entanto, primeiro teve de se encontrar onde a compilação era efetuada. Ao verificar os *logs*, durante a execução do DP, podemos verificar que a compilação é feita na classe **BuildWorker**. Dentro desta encontra-se uma ligação à classe **MavenInvoker** que, através da função *run*, efetua a compilação dos projetos Maven. Desse modo, foi necessário criar uma nova classe pelo nome de **GradleInvoker** que efetuasse a mesma função só que para o Gradle. A compilação dos trabalhos Android também é feita através do Gradle, mas foi igualmente criada uma classe **AndroidInvoker** de modo a acomodar as diferenças entre as tarefas que eram necessárias executar entre os trabalhos Gradle e Android. Também foi adicionada uma condição no **BuildWorker**, dentro da função *checkSubmission*, para escolher qual dos invocadores é que será utilizado na compilação das submissões.

```
//NEW: Added condition to check if either Maven, Gradle or Android are used
var result: Result = Result()
if (assignment.engine == Engine.MAVEN) {
    //Run invoker of engine (clean, compile, test)
    result = mavenInvoker.run(...)

    //Create build report for Maven
    buildMaven(...)
} else if (assignment.engine == Engine.GRADLE) { //Assignment engine is Gradle
    //Run invoker of engine (clean, compile, test)
    result = gradleInvoker.run(...)

    //Create build report for Gradle
    buildGradle(...)
} else if (assignment.engine == Engine.ANDROID) { //Assignment engine is Android
    //Run invoker of engine (clean, compile, test)
    result = androidInvoker.run(...)

    //Create build report for Android
    buildAndroid(...)
}
```

Figura 7 - Código para condição que escolhe *Invoker*.

Como pode ser visto no parágrafo anterior tiveram de ser criadas novas classes para efetuar a compilação através de cada um dos motores. Estas classes foram baseadas no **MavenInvoker** e usaram-se como base para a criação dos seus métodos. Isto foi feito através da herança, sendo que isto pode ser visto noutros pontos do projeto, e teve de ser feito devido ao DP inicialmente só utilizar o Maven. Desse modo, não tinha sido previsto pelo criador a utilização de outras soluções.

No entanto, antes de se passar para a compilação dos projetos, foi necessário criar uma validação nova para os trabalhos. Ou seja, ao ser enviado um trabalho, o **AssignmentValidator** repara que este não tem um ficheiro “pom.xml” e manda uma mensagem de erro. Isto devido a achar que o trabalho ainda utiliza o gestor Maven e, desse modo, necessita desse ficheiro para as dependências. Assim foram criadas três novas classes, sendo essas o **AssignmentValidatorMaven**, o **AssignmentValidatorGradle** e o **AssignmentValidatorAndroid**, de modo a servirem o propósito de validar os trabalhos entregues para cada um dos motores. A sua classe base, sendo essa o **AssignmentValidator**, foi tornada numa classe abstrata que contém as funções necessárias para ambas as validações. Na solução final, a validação do Gradle e do Android encontra-se incompleta pois só consegue verificar se os ficheiros necessários, tal como o “build.gradle” e o “gradle.properties”, estão no projeto. Não consegue verificar se as dependências estão implementadas corretamente, dentro do “build.gradle”, da mesma forma que o Maven faz para o “pom.xml” através do *MavenXpp3Reader*.

```
val gradleFileKotlin = File(assignmentFolder, "build.gradle.kts")
val gradleFileGroovy = File(assignmentFolder, "build.gradle")

//Check if build.gradle exists (check for one)
if (!gradleFileGroovy.exists() && !gradleFileKotlin.exists()) {
    report.add(Info(InfoType.ERROR, "..."))
    return
} else {
    report.add(Info(InfoType.INFO, "Assignment has a build.gradle"))
}
```

Figura 8- Código para validação de trabalhos Gradle.

Com a validação efetuada, teve de ser adicionado um método para efetuar a compilação das submissões. No **MavenInvoker** este método é o “run” que tem como propósito compilar os projetos e efetuar os testes relativos caso seja necessário. Desse modo, foi criado um método semelhante no **GradleInvoker** com o mesmo nome e que serve o mesmo propósito. A criação deste método envolveu a utilização do *tooling* API para executar as tarefas do Gradle equivalentes às que eram corridas no Maven, sendo que estas tarefas é que irão efetuar a compilação e o *testing* da submissão. O código de utilização do *tooling* API foi dado pelo orientador do TFC e tem como propósito efetuar as tarefas de compilação e testagem.

```
//Setup variables for output
var exitLines = ArrayList<String>() //Output lines from invoker
var exitCode = 0 //0 is good, everything else is bad (1)
try {
    val connection =
        GradleConnector.newConnector().forProjectDirectory(projectFolder).connect()
    val build: BuildLauncher = connection.newBuild()

    //Select tasks to run
    build.forTasks("clean", "build", "test")

    //if you want to listen to the progress events:
    build.addProgressListener(ProgressListener {
        LOG.info("progress ${it.description}")
        exitLines.add(it.description)
    })

    //kick the build off
    build.run()
} catch (ex: Exception) {
    ex.printStackTrace()
    LOG.error(ex.localizedMessage)
    exitCode = 1
}
```

Figura 9 - Código para compilação através de Gradle. (dado pelo orientador)

O plano inicial para o Android, e desse modo o **AndroidInvoker**, foi com que ele se conseguisse efetuar o *testing* da submissão. No entanto, alguns erros encontrados durante os testes unitários com o DP impediram o cumprimento deste objetivo. Desse modo, só pode ser utilizada a tarefa “clean” nas submissões Android sendo que esta só limpa os ficheiros do projeto de modo a retirar compilações e testes efetuados previamente. Isto é para que depois o projeto possa ser instalado pelo Professor que criou o trabalho desse modo servindo como um serviço de instalação de projetos Android.

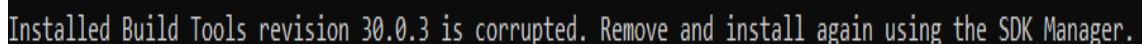
```
val connection =
    GradleConnector.newConnector().forProjectDirectory().connect()
val build: BuildLauncher = connection.newBuild()

//Select tasks to run
build.forTasks("clean")

//kick the build off
LOG.info("Before android clean...")
build.run()
LOG.info("After android clean!")
```

Figura 10 - Código para compilação através de Android.

O erro que foi obtido na compilação das submissões Android foi em relação ao SDK. Este não conseguia efetuar a compilação nem o *testing* das submissões devido a um erro em relação às *build tools* do SDK instalado. No entanto, instalar o SDK de novo e até alterá-lo por outras versões não corrigiu o problema, levando a um impasse em relação à compilação dos trabalhos que impediu que pudessem ser utilizados os testes e desse modo fosse só utilizada a limpeza.

The image shows a screenshot of a terminal window with a black background and white text. The text reads: "Installed Build Tools revision 30.0.3 is corrupted. Remove and install again using the SDK Manager." This is an error message from the Android Studio environment.

```
Installed Build Tools revision 30.0.3 is corrupted. Remove and install again using the SDK Manager.
```

Figura 11 - Erro obtido para compilação em Android

As tarefas que vão ser executadas não podem ser utilizadas só nos ficheiros de submissão que são dados pelos estudantes. Estes só contêm o código para se conseguir executar o seu projeto sendo que faltam as propriedades necessárias para se conseguirem executar e os testes unitários para estes poderem ser comparados com o código da submissão. A forma que o Drop Project lida com isto é que utiliza o código do Professor, que contém as propriedades e os testes unitários necessários, e junta com o código da submissão, que contém o código do estudante para o seu projeto. Isto é feito na função “standardizeFolder”, na classe **UploadController**, e são criadas mais três funções para cada um dos motores de modo a certificar que as propriedades corretas são adicionadas.

```
//Delete submission folder recursively
newProjectFolder.deleteRecursively()

// first copy the project files submitted by the students (main)
...

// first copy the project files submitted by the students (student tests)
...

FileUtils.copyFile(File(projectFolder, "AUTHORS.txt"),File(newProjectFolder,
"AUTHORS.txt"))
if (submission.gitSubmissionId == null && deleteOriginalProjectFolder) {
    FileUtils.deleteDirectory(projectFolder)
}

// next, copy the project files submitted by the teachers
assignmentTeacherFiles.copyTeacherFilesTo(assignment, newProjectFolder)
```

Figura 12 - Código para criação de submissão.

Com a compilação concluída, como também a criação da nova submissão, foi preciso concluir a leitura dos testes para estes serem demonstrados ao utilizador. Para isto, foram utilizadas as classes **BuildReportBuilder**, a **BuildReport** e a **JUnitResultsParser**. Estas foram divididas em várias classes, sendo uma para cada um dos motores utilizados, e foram depois alteradas para funcionar com o que era necessário. As classes **BuildReport** e **BuildReportBuilder** servem como a base para a criação do relatório mas não foram completamente implementadas para o Gradle e o Android, sendo isto um ponto de foco para melhorias futuras. A classe **JUnitResultsParser** foi finalizada e cumpre o seu objetivo demonstrar os resultados dos testes unitários através da função *parseXml*. Estes resultados estão na pasta “build/test-results” da localização criada para a submissão nova, num ficheiro xml, e irão detalhar os resultados dos testes unitários do Professor para a submissão do estudante. Caso um teste tenha passado, então o relatório só necessita de indicar que mais um teste passou. No entanto, caso um teste tenha falhado é necessário explicar a razão. Para isso, é mostrada uma versão resumida do erro que deu no ficheiro xml ao utilizador.

```
val parser = TestSuiteXmlParser(PrintStreamLogger(System.out))
val byteArrayIs = ByteArrayInputStream(content.toByteArray())
val parseResults: List<ReportTestSuite> =
    parser.parse(InputStreamReader(byteArrayIs, Charsets.UTF_8))

val parseResult = parseResults[0]
val testCases : List<ReportTestCase> = parseResult.testCases

val junitMethodResults = testCases.map { JUnitMethodResult(...) }.toList()

return JUnitResults(...)
```

Figura 13 - Código para leitura de testes unitários.

A utilização do Android requer a instalação e utilização do seu SDK. Para isto, foi detalhado um guia sobre o seu setup na seção *Quick Start* do README. De modo a se conseguir utilizar o SDK, foi adicionada uma variável ambiente ao ficheiro “drop-project.properties” da mesma forma que havia para a utilização do maven. Caso não se queira utilizar o Android não será necessário alterar o valor da variável, mas caso tente ser utilizado sem a alteração da variável irá receber uma mensagem de erro. O Android só pode ser utilizado localmente, sendo que a sua utilização através do servidor não é possível pois este não foi implementado nele.

```
# maven configuration (Server)
dropProject.maven.home=${DP_M2_HOME}
dropProject.maven.repository=${DP_MVN_REPO}

# Android SDK configuration (Server)
dropProject.android.home=placeholder
```

Figura 14 - Variáveis ambiente para Maven e SDK Android.

Com a localização do SDK definida só falta a sua utilização. Esta vai ser utilizada na criação de um trabalho e nas submissões, sendo adicionada ao ficheiro “local.properties”. Este ficheiro é, nos trabalhos Android, utilizado para definir a localização do SDK Android de modo a este ser utilizado para a sua compilação. Assim, só é necessário utilizar a variável ambiente previamente definida e adicionar o seu valor para o ficheiro. Esta alteração é feita na classe **UploadController**.

```
//Change local.properties file SDK location in mavenized project
val properties = "local.properties"
val sdk = "sdk.dir=${androidHome}"
LOG.info("SDK DIR changed to -> ${sdk}")

//Add sdk.dir line
File(newProjectFolder, properties).writeText("sdk.dir=${androidHome}")
```

Figura 15 - Utilização do SDK Android.

A solução final entra no objetivo definido para o projeto, esse sendo a inclusão do Android e do gestor de dependências Gradle, mas ainda pode ser melhorado mais no que vai de acordo com a validação dos trabalhos e dos testes unitários em Android.

5 Benchmarking

O Drop Project, de momento, não é utilizável na disciplina de Computação Móvel pelas razões ditas na identificação do problema. No entanto, existem outras alternativas já existentes no mercado que poderiam ser utilizadas antes da nossa solução estar completa, sendo essas o “Robotium” e o “Appium”.

Estas soluções são, tal como o Drop Project, Automated Assessment Tools (AATs) que são utilizadas com o Android, entre outras linguagens, tal como a nossa solução também será.

Appium

O “Appium” é uma framework de automatização de testes open-source para aplicações nativas, híbridas e móveis na web. Aplicações nativas são escritas no IOS, Android ou SDKs Windows. Aplicações híbridas facilitam a interação com o conteúdo da internet e aplicações móveis na web são aquelas que podem ser acedidas através dela. Quando se indica a utilização da web, refere-se ao suporte que a solução tem em relação à utilização do Safari ou do Google Chrome, no iOS, e do browser nativo do Android. Também é *cross-platform*, permitindo a escrita de testes que podem ser utilizados em várias plataformas (iOS, Android, Windows) utilizando o mesmo API, permitindo a reutilização de código. [\[1\]](#)

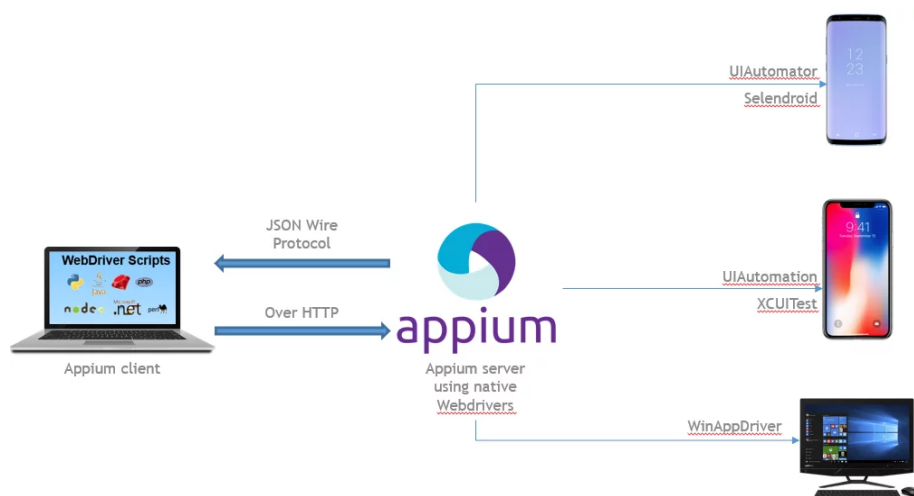


Figura 16 - Utilização do Appium em várias Plataformas.

Robotium

O “Robotium” é uma framework de automatização de testes focado no Android com suporte para aplicações nativas e híbridas. Consegue aguentar várias atividades automaticamente e executar testes de uma forma contínua, devido a conseguir integrar-se com o Maven, o Gradle ou o ant. É fácil de utilizar, dado que não é necessário saber muito sobre a aplicação a ser testada, e também é muito eficiente, podendo escrever testes fortes num tempo mínimo que irão ser executados rapidamente. [\[2\]](#)



Figura 17 - Benefícios de Robotium.

Drop Project

A nossa solução procura, para as linguagens Java e Kotlin, servir o mesmo propósito que as alternativas procurando efetuar os testes através do Maven, do Gradle e do Android. No entanto, irá distinguir-se em relação ao Android onde irá divergir o foco para fora dos testes android, sendo estes os testes que têm de ser realizados num emulador, para os testes unitários. Estas soluções mais focadas nos testes android não seriam possíveis de serem utilizadas devido à falta de recursos no servidor para os emuladores, como foi dito no primeiro capítulo, logo o Drop Project serve melhor para o nosso propósito ao desviar-se deles. A entrega e remoção automática será adicionada numa versão futura, onde na implementação atual só é possível efetuar o download manual de um trabalho enviado.

Tabela 3 - Benchmarking entre Soluções.

Feature	Appium	Robotium	Minha Proposta
Automatização de Testes	✓	✓	✓
Suporte a Android	✓	✓	✓
Automatização de Entregas	X	X	✓
Integração com Maven e Gradle	X	✓	✓
Utilização na Web	✓	X	✓

Logo, é melhor adequada para a nossa situação, enquanto as outras soluções não seriam possíveis de ser utilizadas corretamente devido ao seu foco nos testes.

6 Método e Planeamento

O desenvolvimento do projeto foi dividido em 4 fases para a facilitação de leitura, sendo que cada uma das fases é dividida pela entrega de um dos relatórios e seguem o calendário que foi criado na primeira entrega. Este desenvolvimento foi devidamente seguido até as suas fases finais onde devido a certas complicações, que serão desenvolvidas mais neste capítulo, atrasaram a conclusão da solução.

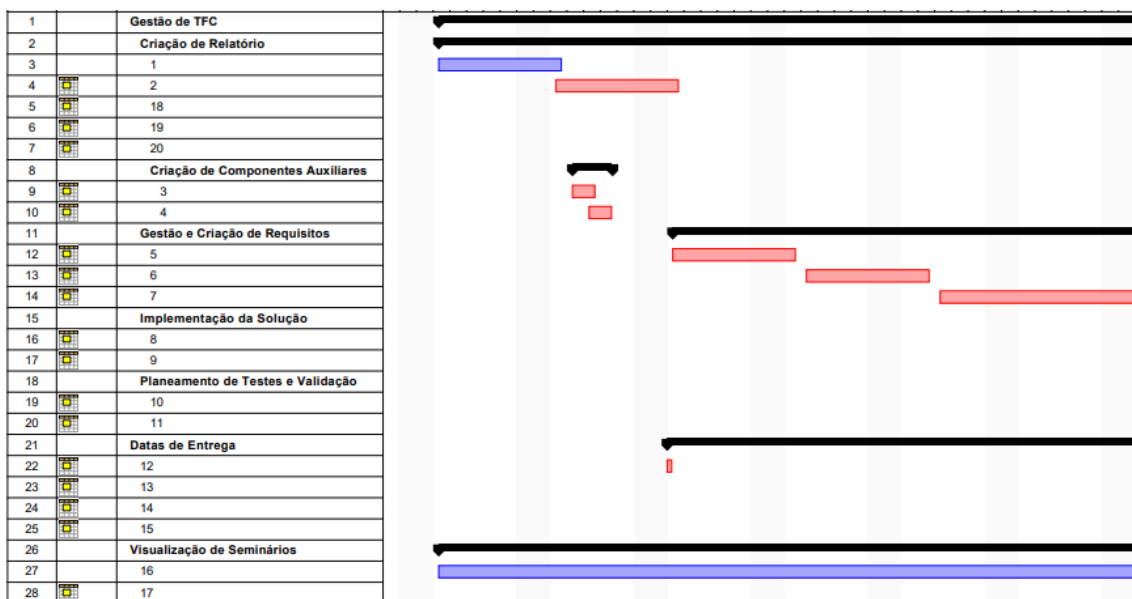


Figura 18 - Calendário da Solução no formato Gantt.

A primeira fase de desenvolvimento envolveu o estudo do Drop Project como também um estudo do problema que a solução iria tratar. Esta fase é marcada pelos sprints 1, 2, 3, 4 e 12 do calendário e foi concluída com a entrega do relatório intercalar do 1º semestre. Antes de se iniciar a fase teve que ser feito o setup do ambiente que seria utilizado. Isto envolveu o download do DP, no [github](https://github.com), e o download do ambiente em si. Com a recomendação do orientador foi utilizado o Linux, sendo que no meu caso foi instalado e utilizado o WSL (*Windows Subsystem for Linux*) para Ubuntu de modo a não necessitar de utilizar uma máquina virtual ou de alterar para um sistema operativo que não estava habituado.

O estudo do código do DP foi um processo menos demorado que inicialmente esperado devido ao esforço do docente Pedro Alves para criar código compreensível, isto sendo para um programador não só exterior do projeto como também menos experiente, e desse modo fazendo o estudo mais fácil. Todo o código estava explicado, as classes estavam divididas em diferentes diretorias e tinha sido criado um README muito compreensivo que explicava o seu funcionamento. Este descreve as suas limitações, como o DP funciona e como o inicializar localmente caso efetuasse o download. Também tinha um link para a sua execução no Heroku sendo que este foi o método utilizado para a sua execução e testes inicialmente. Isto demonstrou-se ser ineficiente e uma das razões pelo qual os primeiros testes efetuados, depois de ser adicionado a primeira versão do Gradle, demonstraram-se insuficientes. Nesta fase foram igualmente estudados os gestores de dependências Maven e Gradle.

A segunda fase de desenvolvimento foi marcada pela resolução dos requisitos, sendo estes os sprints 18, 5, 6, 7 e 13 do calendário, e foi concluída com a entrega do relatório intermédio. Estes foram criados com base no objetivo do projeto, que foi estudado outra vez, como também foram revistos várias vezes pelo orientador de modo a conseguir melhor definir os objetivos que a solução deveria cumprir. Inicialmente houve alguns problemas na diferenciação dos requisitos funcionais com os não funcionais, mas estes foram revistos e tratados várias vezes antes da entrega do relatório.

A terceira fase de desenvolvimento foi marcada pela criação do protótipo inicial e da sua testagem, sendo estes os sprints 19, 8, 10 e 14 do calendário, e acabou com a entrega do relatório intercalar do 2º semestre. A criação do protótipo envolveu alterações ao código do DP sendo que estas podiam envolver a alteração de pequenas partes do código que só funcionassem com o Maven, tal como a adição de condicionais em certas partes para não efetuar código específico para só um dos gestores, como também a adição e alteração de classes inteiras para adequar as operações já criadas a ambos os gestores. Os testes nesta fase foram efetuados através do Heroku mas este acabou por demonstrar-se incapaz de efetuar o necessário devido a retornar erros de memória na criação de um trabalho e na entrega de submissões. Estes mesmos erros foram o que causaram o inicial atraso ao projeto e que levou a outros problemas não serem identificados mais cedo.

A quarta e última fase de desenvolvimento, a atual, é marcada pela criação da solução final e da sua testagem. A fase é marcada pelos sprints 20, 9, 11 e 15 do calendário e será concluída com a entrega do relatório final. Esta fase era suposto ser iniciada com a criação da solução final com base no protótipo criado. No entanto, foram encontrados vários erros pelo docente que não iam de encontro com o planeado para o projeto e, desse modo, faziam com que as alterações efetuadas não funcionassem corretamente. Assim, o nosso foco alterou para uma revisão do protótipo, que depois levou a uma alteração do código inicialmente escrito. Depois de ter sido efetuada a revisão foi possível passar para a criação da solução final mas, devido aos problemas encontrados inicialmente, não foi possível efetuar os objetivos todos. Os testes nesta fase foram também alterados para serem feitos localmente em vez de no Heroku devido a ser mais rápido de executar e funcionar melhor para a verificação dos resultados.

Tabela 4 - Tabela associada ao Calendário da Solução.

# Sprint	Funcionalidade	Data Início - Data Fim
16	Visualização de Seminários (1 Semestre)	15/11/21 - 31/01/22
1	Início e Primeira Versão do Relatório	15/11/21 - 22/11/21
2	Várias Revisões do Relatório	20/11/21 - 29/11/21
3	Criação de Diagramas Auxiliares	23/11/21 - 24/11/21
4	Criação de Calendário Auxiliar e Tabela	24/11/21 - 25/11/21
12	Entrega do Relatório Intercalar do 1º Semestre	29/11/2021
18	Criação do Relatório Intermédio	29/11/21 - 28/01/22
5	Criação Inicial de Requisitos Não Funcionais	30/11/21 - 06/12/21
6	Criação Inicial de Requisitos Funcionais	07/12/21 - 14/12/21
7	Revisão e Modificação de Requisitos	15/12/21 - 28/12/21
13	Entrega do Relatório Intermédio	28/01/2022
19	Criação do Relatório Intercalar do 2º Semestre	30/01/22 - 24/04/22
8	Criação do Protótipo Inicial	31/01/22 - 31/03/22
10	Testing do Protótipo Inicial	09/02/22 - 24/04/22

17	Visualização de Seminários (2 Semestre)	14/02/22 - 11/06/22
14	Entrega do Relatório Intercalar do 2º Semestre	24/04/2022
20	Criação do Relatório Final	25/04/22 - 18/07/22
9	Criação da Solução Final	25/04/22 - 20/05/22
11	Testing da Solução Final	19/05/22 - 03/06/22
15	Entrega do Relatório Final	09/09/2022

(As tarefas assinaladas a verde são as que foram efetuadas e eram necessárias para a entrega dos relatórios associados)

Os problemas que foram detectados no início da quarta e última fase de desenvolvimento, que acabaram por causar um atraso na entrega da solução, foram encontrados demasiado tarde devido a uma falta de comunicação da minha parte e do que eu penso agora ter sido um mau entendimento de como a minha implementação do código funcionava no DP. Olhando para trás, consigo reparar que a minha teimosia de tentar efetuar o projeto sem pedir ajuda ao orientador levou a uma falta de transparência no que eu estava a fazer, fazendo com que os meus erros continuassem a propagar-se ao longo do projeto e desse modo causando um atraso na entrega final após serem encontrados.

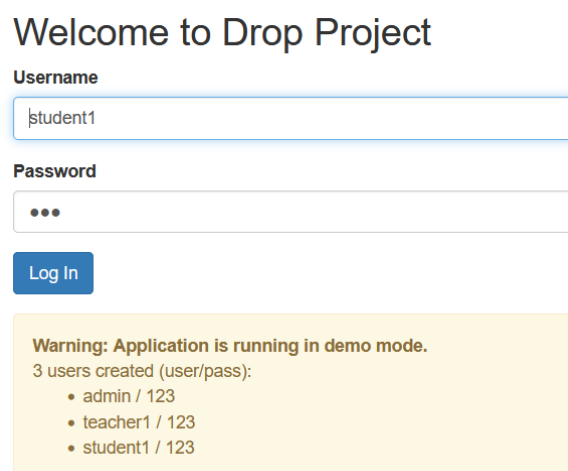
7 Resultados

Os resultados que poderão ser visualizados neste capítulo irão ser mostrados com base na utilização da solução final. Todos os resultados, outputs e *outcomes* que são vistos são baseados numa versão testada da solução e só irão demonstrar o que foi completo. Estes testes foram efetuados localmente, embora seja possível visualizar estes mesmos resultados no servidor.

Demonstração de Funcionalidades

A solução final apresentada demonstra várias funcionalidades presentes não só na versão base do Drop Project como também que foram adicionadas na solução final. Estas funcionalidades irão ser demonstradas nesta seção de modo a familiarizar o leitor com a sua execução para permitir uma melhor utilização da plataforma. Caso seja necessário, foi criado um [vídeo](#) para demonstrar o setup do DP.

Após a inicialização do DP, o utilizador consegue visualizar um ecrã de login. Este está disponível na versão inicial da solução e permite o utilizador entrar em três contas diferentes, estas sendo a do estudante, a do professor e a do administrador. Dependendo da conta em que se entrar, podem ser utilizadas diferentes funcionalidades da solução. O estudante só consegue efetuar submissões a trabalhos, o professor consegue efetuar submissões e consegue criar trabalhos e o administrador consegue efetuar tudo isso com a adição de visualizar informação extra sobre o DP.



Welcome to Drop Project

Username

Password

Log In

Warning: Application is running in demo mode.
3 users created (user/pass):

- admin / 123
- teacher1 / 123
- student1 / 123

Figura 19 - Ecrã de Login.

Com a conta inicializada o utilizador irá entrar para a página principal do DP. Nesta ele poderá visualizar todos os trabalhos que está associado a e em que pode enviar uma submissão. Para o propósito de exemplificar esta funcionalidade, foram criados dois novos trabalhos exemplo que são criados imediatamente na inicialização da solução. Cada um destes serve para exemplificar partes diferentes da solução tendo dois trabalhos Maven no Java e no Kotlin, um trabalho em Gradle e outro trabalho em Android.

My Assignments







Assignment ID	Name	Actions
sampleMavenJavaProject	Sample Maven Java Assignment	
sampleMavenKotlinProject	Sample Maven Kotlin Assignment	
sampleGradleKotlinProject	Sample Gradle Kotlin Assignment	 
sampleAndroidKotlinProject	Sample Android Kotlin Assignment	 

Figura 20 - Ecrã Principal.

Caso o utilizador selecione um deles, através do olho azul, então será levado para a página do trabalho onde poderá ver as instruções desse trabalho como também enviar a sua submissão.

Sample Gradle Kotlin Assignment

Choose or drop here the zip file containing the project

Sample Kotlin Assignment with Gradle

This is just a very simple Kotlin assignment just to experiment with Gradle assignments in Drop Project

The source of this assignment is available on <https://github.com/Diogo-a21905661/test-kotlin-gradle-assignment>

Figura 21 - Ecrã de um Trabalho.

Se quiser criar um trabalho então pode utilizar a opção “Manage Assignments” presente no canto superior esquerdo do ecrã. Esta só está disponível caso o utilizador esteja na conta do Professor ou do administrador e permite ao utilizador aceder a mais funcionalidades em relação aos trabalhos, tais como visualizar os trabalhos criados, os trabalhos guardados, criar um trabalho e importar um trabalho.

Esta opção, que tem o nome de “Create Assignment” no menu, leva o utilizador para um novo ecrã. Neste pode seleccionar vários valores num menu que permite definir o seu id, o seu nome, o motor que será utilizado, a linguagem de programação que será utilizada e também o formato de entrega das submissões. Também é necessário preencher a opção “Git Repository URL” com o link ao repositório Git do trabalho. Isto é de modo a conseguir receber o código do trabalho.

Create a new assignment

The screenshot shows a web form titled "Create a new assignment". It contains two input fields. The first field is labeled "ID" and contains the text "cs1Project2018". Below this field is a hint: "An unique short ID, with no spaces (e.g., cs1Project2018)". The second field is labeled "Name" and contains the text "CS1 Main Project for 2018". Below this field is a hint: "A longer human-friendly name (e.g., CS1 Main Project for 2018)".

Figura 22 - Ecrã de criação de um Trabalho.

O utilizador pode visualizar em mais detalhe os trabalhos que criou através da opção “Current Assignments”. Esta leva o utilizador para um ecrã parecido com o principal, só que com novos detalhes associados a cada trabalho. Neste novo ecrã é possível visualizar quando a última submissão foi efetuada, se o trabalho está ativo, se está privado e o número de submissões como também os seus resultados se este for selecionado. Também são apresentadas mais ações para cada trabalho, sendo que o utilizador pode apagá-lo, arquivá-lo ou visualizar mais detalhes.

Manage Assignments

Assignment ID	Tags	Last submission	Active?	Private?	Submissions	Actions
sampleMavenJavaProject	sample maven java	18/Dec 14:37	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	9	
sampleMavenKotlinProject	sample maven kotlin	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	
sampleGradleKotlinProject	sample kotlin gradle	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	
sampleAndroidKotlinProject	sample kotlin android	-	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	0	

[Create Assignment](#)
[Import Assignment](#)

Figura 23 - Ecrã de gerir Trabalhos.

Caso o utilizador selecione a opção para visualizar mais detalhes então é levado para um novo ecrã. Neste, irá conseguir visualizar todos os detalhes que preencheu na criação do trabalho. Também consegue visualizar o relatório de validação do trabalho, criado durante a sua inicialização, e os testes unitários associados ao trabalho. Esta última opção não consegue detectar os testes na versão atual da solução, sendo um ponto de foco para implementações futuras.

Assignment sampleGradleKotlinProject

[Overview](#)
[Validation Report](#)
[Tests](#)

[Reconnect with Git](#)
[Export](#)

Name: Sample Gradle Kotlin Assignment (created by teacher1)

Figura 24 - Visualizar detalhes de Trabalho.

Demonstração de Resultados

Através da execução do Drop Project poderão ser verificadas as funcionalidades implementadas assim como os resultados que podem ser obtidos através da sua execução. Os resultados demonstrados nesta seção podem ser visualizados no [vídeo](#) de demonstração.

A criação de um trabalho é uma das funcionalidades mais importantes para o funcionamento do DP. De modo a testar os seus resultados foi necessário verificar se, após a sua criação, era possível verificar a sua adição no ecrã principal. Para isto, teve de se verificar o número de trabalhos logo após a inicialização do DP, sendo estes quatro.





Assignment ID	Name	Actions
sampleMavenJavaProject	Sample Maven Java Assignment	
sampleMavenKotlinProject	Sample Maven Kotlin Assignment	
sampleGradleKotlinProject	Sample Gradle Kotlin Assignment	
sampleAndroidKotlinProject	Sample Android Kotlin Assignment	

Figura 25 - Lista inicial de Trabalhos.

Durante a criação de um trabalho o utilizador terá a hipótese de escolher o motor que poderá utilizar e a linguagem de programação. Embora a opção de alterar a linguagem estivesse na solução base, a opção para o motor foi adicionada de modo a diferenciar os novos tipos de trabalhos que podiam ser enviados. Dependendo do valor escolhido o utilizador irá precisar de entregar um trabalho com um formato diferente, sendo que este pode tomar o formato do Maven, do Gradle ou do Android. Isto vai de encontro com os requisitos funcionais um, dois e três.

Engine

Gradle

Submissions will be built/deployed using the chosen engine

Programming Language

Kotlin

Submissions must be implemented using this language

Figura 26 - Criação do motor e da linguagem.

Após a criação do trabalho o utilizador será levado para sua página de detalhes onde pode visualizar toda a informação que preencheu no formulário. Se voltar para a página principal, ao seleccionar o “Drop Project”, irá também reparar que o trabalho criado foi juntado à lista de trabalhos.






Assignment ID	Name	Actions
sampleMavenJavaProject	Sample Maven Java Assignment	
sampleMavenKotlinProject	Sample Maven Kotlin Assignment	
sampleGradleKotlinProject	Sample Gradle Kotlin Assignment	
sampleAndroidKotlinProject	Sample Android Kotlin Assignment	
testDropProject	Test Drop Project	

Figura 27 - Ecrã principal com trabalho adicionado.

Com um trabalho criado, será possível efetuar submissões a este. Da mesma forma que o Maven, o Gradle consegue realizar a verificação das submissões pelos testes unitários criados pelo Professor. A estes, vai retornar um resultado que depois irá demonstrar num novo ecrã, sendo que isto vai de encontro com o requisito funcional quatro. A submissão que será utilizada neste caso será um [GitHub](#) exemplo criado para submissões em Kotlin que sejam feitas para os trabalhos exemplo. Caso seja entregue uma submissão que passe os testes todos, o utilizador poderá visualizar um ecrã de sucesso.

Results summary




Project Structure	
Compilation	
Code Quality (Checkstyle)	
Teacher Unit Tests	3 / 3

Figura 28 - Resultado de testes passados.

Caso seja entregue uma submissão com testes falhados, o utilizador irá visualizar um ecrã com resultados diferentes. Este irá informar o utilizador do número de testes que acertou, com base no número total, como também os testes que errou e a razão pela qual os errou.

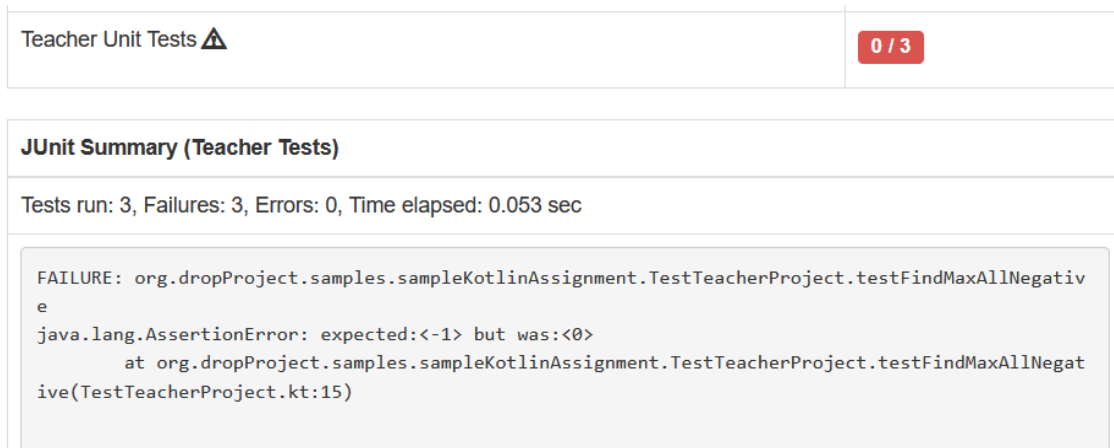


Figura 29 - Resultado de testes falhados.

No entanto, as submissões Android passam por um tratamento diferente. Estas, na solução desejada, eram suposto conseguir efetuar os testes unitários passados pelo Professor, sendo que na versão atual estes testes unitários não são possíveis executar devido a erros com o SDK logo estes não são verificados nem demonstrados. Isto vai de encontro com os requisitos funcionais cinco, seis e sete.

Results summary

Project Structure	
Compilation	
Code Quality (Checkstyle)	

Figura 30 - Submissão a um trabalho Android.

Com esta submissão dá-se por concluído a demonstração da solução final e das funcionalidades adicionadas. O resultado obtido na entrega de trabalhos Android pode ser melhorado com a adição dos testes unitários.

8 Conclusão e Trabalhos Futuros

O projeto, no seu estado atual, demonstra-se quase completo com base no objetivo definido no início deste trabalho como também nos requisitos definidos. No entanto, ainda requer alterações do código para melhor cumprir o seu objetivo, como também pode servir como uma base para trabalhos futuros.

O principal objetivo deste trabalho foi, desde o início, o apoio ao professor de Computação Móvel no processo de entrega e avaliação de trabalhos. A solução final tenta cumprir este objetivo com a adição da entrega de submissões Android, a trabalhos criados por um Professor, com base na adição do Gradle. Mesmo assim, a adição do Android não necessita de ser o passo final para o projeto. A implementação do Gradle, como um gestor de dependências, adicionou um novo mundo de possibilidades às diferentes tecnologias que podem ser adicionadas. Pode ser dado como exemplo o NPM (Node Package Manager). [3] Este é um gestor de dependências com um foco em projetos Javascript que poderia ser utilizado para apoiar, caso seja necessário, os docentes de Programação Web poderiam utilizar o DP não só para visualizar o projeto dos alunos como também o seu código numa só plataforma. Isto para exemplificar que o projeto pode evoluir com base na evolução das cadeiras da faculdade, podendo apoiar todas as disciplinas que o achem necessário.

Bibliografia

- [DEISI21] DEISI, Regulamento de Trabalho Final de Curso, Set. 2021.
- [ULHT21] Universidade Lusófona de Humanidades e Tecnologia, www.ulusofona.pt, acedido em Out. 2021.
- [APPIUM] Software Appium, <https://appium.io/docs/en/about-appium/intro/>, acedido em Nov. 2021.
- [ROBOTIUM] Software Robotium, <https://github.com/RobotiumTech/robotium>, acedido em Nov. 2021.
- [NPM] Software NPM, <https://www.npmjs.com/>, acedido em Julho 2022.
- [DP EDU] Drop Project EDU, <https://github.com/drop-project-edu/drop-project>, acedido em Nov. 2021.
- [DP TFC] Drop Project TFC, <https://github.com/Diogo-a21905661/drop-project-tfc>, acedido em Nov. 2021.
- [DP ASSI GRA K] Drop Project create Gradle Assignment using this GitHub (Kotlin Assignment), <https://github.com/Diogo-a21905661/test-kotlin-gradle-assignment>, acedido em Julho 2022.
- [DP ASSI GRA J] Drop Project create Gradle Assignment using this GitHub (Java Assignment), <https://github.com/Diogo-a21905661/test-java-gradle-assignment>, acedido em Julho 2022.
- [DP ASSI MAV] Drop Project create Maven Assignment using this GitHub (Kotlin Assignment), <https://github.com/Diogo-a21905661/test-kotlin-maven-assignment>, acedido em Julho 2022.
- [DP ASSI AND] Drop Project create Android Assignment using this GitHub (Kotlin Assignment), <https://github.com/Diogo-a21905661/test-kotlin-android-assignment>, acedido em Agosto 2022.
- [DP SUB KOT] Drop Project create Submission (Gradle and Maven) using this GitHub (Kotlin Submission), <https://github.com/Diogo-a21905661/test-kotlin-submission>, acedido Julho 2022.
- [VIDEO SET] Vídeo de Setup, <https://youtu.be/-anDD6zGHR0>, acedido em Julho 2022.
- [VIDEO DES] Vídeo de Demonstração, <https://youtu.be/YPrNa21vFEw>, acedido em Julho 2022.

[SERVER] Servidor que contém solução final, <https://drop-project.duckdns.org>, acessado em Agosto 2022.

[ANDROID TEST] Website com explicação para fundamentos de testing em Android, <https://developer.android.com/training/testing/fundamentals>, acessado em Setembro 2022.

Anexos

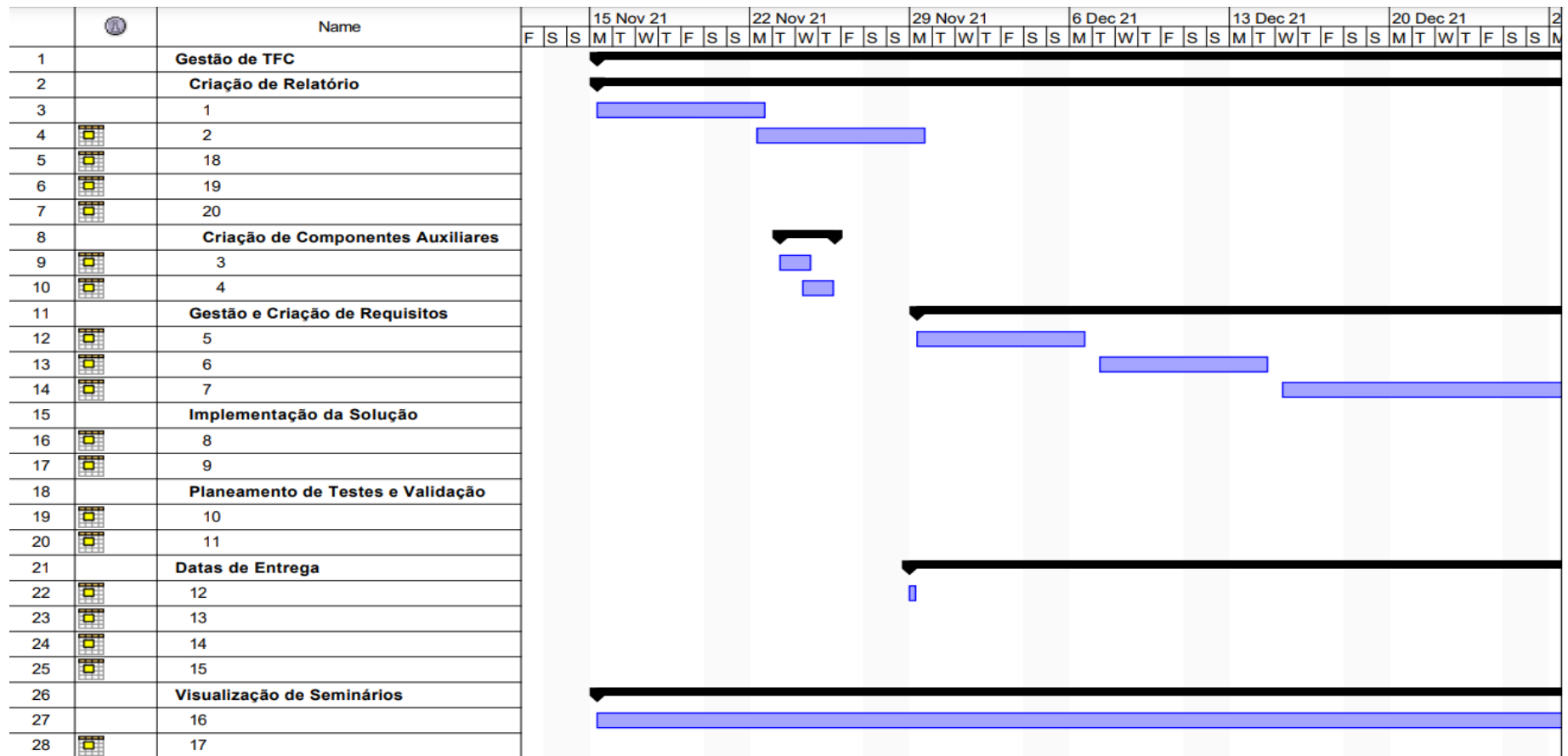


Figura 31 - Calendário maior da Solução no formato Gantt.

Glossário

LEI	Licenciatura em Engenharia Informática
TFC	Trabalho Final de Curso
DP	Drop Project
AAT	Automated Assessment Tool
APK	Android Package
NPM	Node Package Manager
WSL	Windows Subsystem for Linux
REST	Representational State Transfer
SDK	Software Development Kit