



UNIVERSIDADE DO MINHO

MESTRADO EM ENGENHARIA INFORMÁTICA

Projeto de Informática
CPI Governance Automation
E2 - Accenture

Cláudia Ribeiro (PG49998) Daniel Azevedo (PG50311)
Joaquim Roque (PG50502) Nuno Mata (PG44420)
Pedro Araújo (PG50684) Rodrigo Rodrigues (PG50726)
Rui Guilherme Monteiro (PG50739)

Ano Letivo 2023/2024



Índice

1	Introdução	1
1.1	Contextualização	1
1.2	Motivação	1
1.3	Objetivos principais	1
2	Metodologia e gestão do projeto	2
2.1	Comunicação com a <i>Accenture</i>	2
2.2	<i>Roadmap</i>	2
2.3	Cronograma de desenvolvimento	3
2.4	Repositório de suporte ao desenvolvimento	3
3	Levantamento de requisitos	4
3.1	Método de levantamento de requisitos	4
3.2	Requisitos funcionais	4
3.3	Requisitos não funcionais	4
3.4	<i>Backlog</i> do produto	5
3.5	Alterações à proposta inicial	5
4	Diagrama de casos de uso	5
5	Catálogo das melhores práticas	6
6	Arquitetura da solução	6
6.1	<i>Overview</i> da configuração Docker	7
6.2	Tecnologias utilizadas no sistema	8
6.2.1	<i>Backend</i>	8
6.2.2	<i>Frontend</i>	9
6.2.3	Base de dados	11
6.2.4	<i>Jenkins</i>	11
7	Segunda componente	12
7.1	Objetivos	12
7.2	Desenvolvimento de <i>Mockups</i>	13
7.3	Interfaces finais	13
8	Revisão automática de código <i>Groovy</i> e <i>Java</i>	17
9	Guia de utilização da solução	19
10	Trabalho futuro	19
11	Conclusão	20

1 Introdução

1.1 Contextualização

No âmbito deste projeto, propõe-se a conceção e implementação de uma aplicação denominada ‘*CPI Governance Automation*’. Esta aplicação visa automatizar a revisão das interfaces de integração desenvolvidas na plataforma *SAP Cloud Platform Integration*¹ (*CPI*). A *CPI* desempenha um papel crucial ao proporcionar uma ponte entre aplicações em nuvem e locais com produtos *SAP* e não *SAP* de terceiros, permitindo a troca de dados em tempo real.

Atualmente, a verificação manual das práticas recomendadas para a criação de artefactos no *CPI* aumenta significativamente os custos e o tempo do processo de revisão. O projeto propõe a expansão e aprimoramento da ferramenta existente, *CPILint*², uma ferramenta de linha de comandos para *SAP Cloud Integration*. Através da mesma, um conjunto de regras podem ser especificadas, às quais o artefacto em questão deve obedecer para completar o processo de *review* com sucesso.

1.2 Motivação

A revisão automática de código e interfaces deste género não é novidade, existindo algumas ferramentas no mercado para a realização da revisão das interfaces desenvolvidas no *SAP CPI*, como por exemplo, *Figaf SAP Integration DevOps*. No entanto, essas mesmas soluções apresentam alguns problemas na sua utilização, nomeadamente, por vezes a solução ser demasiado individual ou pouco extensível para outros casos de uso, além de serem acompanhadas por um grande custo monetário. Por isso, é necessário a utilização de novas abordagens no processo de *code review* que sejam menos dispendiosas e mais facilmente adaptáveis a diferentes necessidades nas organizações atuais.

Adicionalmente, embora o *CPILint* seja uma ferramenta valiosa, apresenta limitações na cobertura de regras das práticas recomendadas, e não inclui a revisão de código nas linguagens *Groovy* e *Javascript*.

1.3 Objetivos principais

A solução pretendida deveria seguir e satisfazer os seguintes objetivos para a implementação da componente principal:

- Catalogar e documentar as melhores práticas para o desenvolvimento de fluxos de integração de *SAP CPI*;
- Catalogar e documentar as melhores práticas para o desenvolvimento de código na linguagem *Groovy*;
- Verificar as regras cobertas atualmente pelo *CPILint*;
- Definir e descrever regras novas para o *CPILint*;
- Criar uma aplicação para rever automaticamente a integração de interfaces desenvolvidas na plataforma *SAP CPI*;
- Estender as funcionalidades do sistema de automação da governança do *CPILint*, desenvolvendo um módulo para revisão automática de código *Groovy* e de arquivos *JAR*;

¹<https://help.sap.com/docs/cloud-integration/sap-cloud-integration/what-is-sap-cloud-integration?locale=en-US>

²<https://github.com/mwittrock/cpilint>

2 Metodologia e gestão do projeto

A metodologia escolhida para o desenvolvimento deste projeto foi SCRUM, visto ser uma estrutura ágil de colaboração em grupo utilizada muito comumente, e também devido à grande flexibilidade que esta oferece. De modo a atingir os objetivos, foi realizada a divisão da equipa por *roles* de responsabilidade:

- Nuno - Documentação
- Daniel e Pedro - *Frontend*
- Cláudia e Joaquim - *Backend*
- Rodrigo - *Scrum Master* e *Full Stack*
- Rui - *Product Owner* e documentação

Apesar disso, ao longo do desenvolvimento do projeto, foi notório que estes papéis eram dinâmicos e, existindo necessidade, qualquer membro realizou todo o tipo de tarefas, desde gestão do projeto, até implementação propriamente dita de funcionalidades, entre outras.

2.1 Comunicação com a *Accenture*

Desde cedo foi decidido que as reuniões decorreriam em regime semanal, incidido tanto no planeamento dos *sprints*, como também em *sprint review* e *sprint retrospective*. Decorreram às quintas-feiras de forma presencial nos escritórios da Accenture de Braga quando possível e de forma remota através de uma comunidade criada no *Microsoft Teams* para o efeito. Para mais, de forma a discutir dúvidas mais urgentes e expor ideias de forma célere houve contacto constante através de um grupo de *Whatsapp* e através de e-mail com os orientadores empresariais.

Os *sprints* possuíam idealmente a duração de uma semana, de forma a conseguir eficazmente comunicar os desenvolvimentos todas as semanas na reunião com a empresa. Durante os *sprints* houve lugar a inúmeras reuniões entre os membros do grupo de forma *online* para cumprir as tarefas propostas pelos *stakeholders* do projeto.

2.2 *Roadmap*

Numa fase inicial foi-nos pedido, pelo orientador empresarial, que elaborássemos um *Roadmap*³ para ajudar à gestão das tarefas principais do projeto. Um *Roadmap* é, essencialmente, um diagrama de alto nível que permite ter uma visão das tarefas delineadas para o desenvolvimento de *software* ao longo do tempo.

³<https://go.support.sap.com/roadmapviewer/>

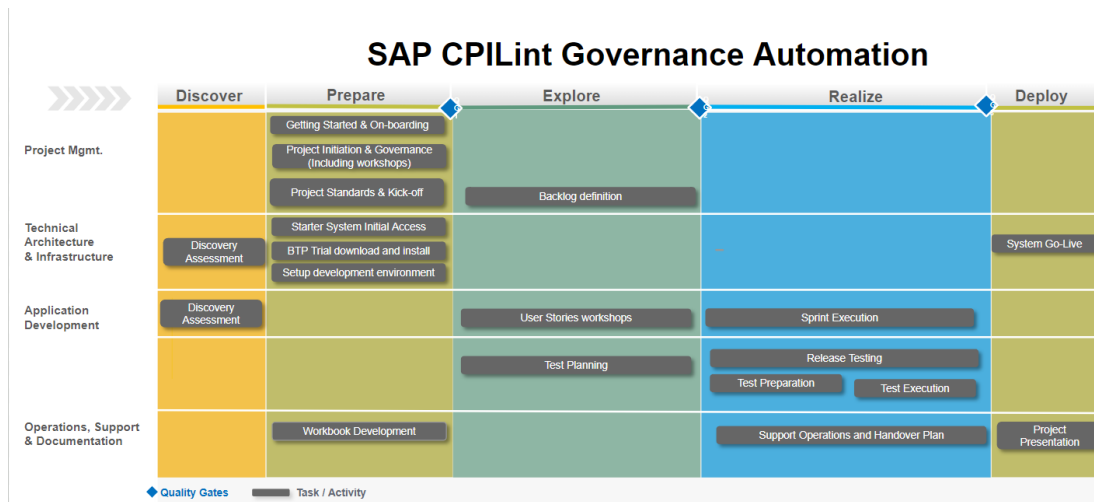


Figura 2: *Roadmap* que sumaria as tarefas previstas para o projeto

Neste caso, o *roadmap* em apreço está dividido em cinco fases principais sendo elas as seguintes:

- **Descoberta** - Fase inicial de levantamento de requisitos e definição do escopo do projeto;
- **Preparação** - Fase em que é feito o contacto inicial com as plataformas e tecnologias, por exemplo *SAP CPI* que serão essenciais para o desenvolvimento do projeto. Aqui foi ainda desenvolvido o *Workbook* que será abordado na secção 5.
- **Exploração** - Nesta fase foram realizados *workshops* presenciais na *Accenture* de forma a alcançarmos um consenso relativamente aos objetivos finais desejados para a solução de forma a podermos passar para a próxima fase. Nesta altura foi ainda elaborado o documento do *Product Backlog*, com o propósito de desenvolver uma lista de tarefas, derivadas do *Roadmap*, e as suas respetivas prioridades, de forma a auxiliar a compreensão de como seriam alcançados os objetivos os propostos.
- **Realização** - Esta fase representa o período em que a equipa de desenvolvimento de *software* está ativamente a trabalhar na implementação e construção do *software* com base nos requisitos e objetivos estabelecidos nas fases anteriores.
- **Implantação** - No nosso caso esta é a etapa que representa a conclusão dos documentos auxiliares ao projeto, como também o desenvolvimento dos diapositivos para complementar a apresentação da solução desenvolvida, tanto no âmbito da unidade curricular universitária como na apresentação realizada na empresa *Accenture* em Braga.

2.3 Cronograma de desenvolvimento

Foi decidido também que havia a necessidade da criação de um cronograma expectável do desenvolvimento de todo o projeto, que se encontra em anexo (e disponível de forma *online* neste [link](#)) e reflete a ideia inicial da divisão temporal do trabalho de acordo com o prazo de entrega final. Nele estão descritas as tarefas previstas inicialmente e o seu período de execução, englobando algumas reuniões iniciais e os *sprints* de desenvolvimento planeados quando foi começado o projeto.

2.4 Repositório de suporte ao desenvolvimento

O *Git* foi o sistema de controlo de versões utilizado ao longo do desenvolvimento deste projeto, aliado ao *Github*, enquanto repositório de código fonte. O repositório do grupo pode ser acedido [aqui](#). Ademais, foram utilizados ‘*Issues*’, de forma a agilizar a divisão de tarefas do projeto no *Github* como forma de discussão e planeamento de tarefas relevantes.

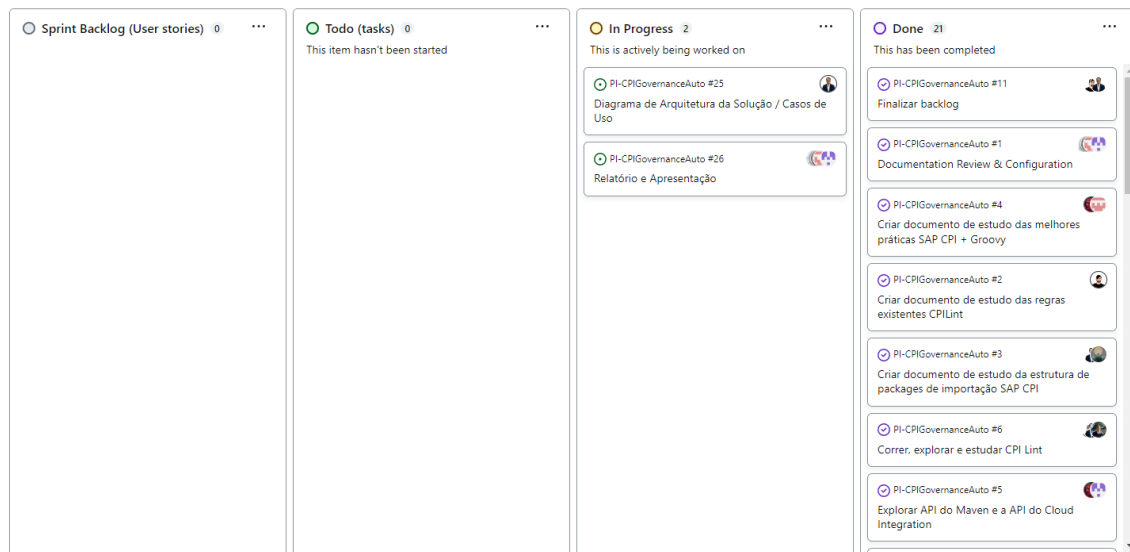


Figura 3: Task board do projecto no Github

3 Levantamento de requisitos

3.1 Método de levantamento de requisitos

Através de *workshops* presenciais nos escritórios de Braga da *Accenture*, foram amplamente discutidos os requisitos principais do sistema pretendido entre todo o grupo (*Product Owner* incluído) e os orientadores empresariais presentes.

3.2 Requisitos funcionais

Em seguida serão apresentados os requisitos funcionais levantados pela equipa:

- Requisitos de configuração: Englobam o armazenamento das credenciais necessárias, dos ficheiros de regras de *CPILint* e *CodeNarc*, bem como, a configuração do repositório a ser utilizado.
- Requisitos da revisão automática: Contêm o controlo da execução do *pipeline* de revisão de código no *Jenkins*.
- Requisitos dos fluxos: Compreende o *download*, através da API do SAP, dos fluxos de integração e o *upload* dos fluxos e dos *reports* gerados para o repositório configurado.
- Requisitos da interface web: Permitir o *upload* de ficheiros de regras através da mesma, a seleção dos ficheiros necessários para a execução, relevar o *report* de execução englobando cores diferentes para severidades diferentes.

3.3 Requisitos não funcionais

Durante os *workshops* de obtenção de requisitos, foram sugeridos pela *Accenture* vários que estão referidos de seguida, bem como algumas sugestões por parte do grupo:

- Requisitos da revisão automática: Dizem respeito à integração da revisão de código *Groovy* e ficheiros JAR, com a revisão de fluxos de integração SAP CPI.

- Requisitos tecnológicos: A necessidade do uso da ferramenta *CodeNarc*⁴, do *Dependency-Check*⁵ e do *CPILint*, em conjunto com o servidor de automatização de código *Jenkins*⁶, aliado a uma arquitetura baseada em *containers Docker*.
- Requisitos de usabilidade: Plataforma *web* apropriada para desenvolvedores de fluxos de integração SAP CPI.
- Requisitos de acessibilidade: Capacidade de suportar diferentes idiomas de forma a assegurar uma experiência (*UX*) mais inclusiva para os utilizadores. No caso da nossa plataforma *web*, é possível alternar a interface entre o idioma Português e Inglês.

3.4 Backlog do produto

O *backlog* é geralmente priorizado, no desenvolvimento de um projeto, para garantir que as funcionalidades mais importantes e valiosas sejam abordadas primeiro. O *backlog* do produto inclui todas as funcionalidades e tarefas que a equipa planeia abordar ao longo do tempo. O seu objetivo passa por garantir que todos os envolvidos compreendam os objetivos e prioridades do projeto. Cada item no *backlog* é frequentemente representado como uma tarefa que descreve uma funcionalidade específica ou requisito do ponto de vista do utilizador final. Isso ajuda a equipa de desenvolvimento a entender o que precisa de ser feito. O *backlog* desenvolvido encontra-se em anexo, mas também se encontra disponível [online](#). De modo a haver um melhor entendimento das tarefas necessárias, o seu desenvolvimento inicial foi dividido em duas partes, uma relativa à componente principal do projeto e a outra relativa à segunda componente do projeto, em que estão representadas as *user stories* identificadas inicialmente, de cada componente, as suas prioridades de implementação e, por último, o seu estado atual.

3.5 Alterações à proposta inicial

Durante o desenvolvimento do projeto ‘CPI Governance Automation’, uma revisão estratégica foi conduzida pela *Accenture*, resultando em ajustes no escopo inicialmente proposto. Inicialmente, planeávamos a criação de novas regras para o *CPILint*, visando ampliar a cobertura do processo de revisão de integração no SAP CPI. Contudo, após uma análise criteriosa das necessidades da empresa, foi optada por uma abordagem mais específica.

O projeto concentra-se na identificação e descrição minuciosa das regras existentes no *CPILint*. Com a alteração à proposta inicial, catalogamos as melhores práticas para o desenvolvimento de interfaces de integração SAP CPI, sem a necessidade de criar novas regras de forma proativa apenas descrevendo-as e retirando a necessidade nesta fase de permitir a alteração das regras através de um formulário *web*, permitindo apenas a utilização de ficheiros de regras pré-configurados por parte do utilizador. Apesar disso, a complexidade do projeto aumentou com a utilização não prevista de uma ferramenta como o *Jenkins*, para realizar a integração do *CPILint* com o *CodeNarc* e o *DependencyCheck*, em detrimento da utilização de uma *REST API*, como estava previsto na segunda componente.

4 Diagrama de casos de uso

O diagrama de casos de uso pretende representar todos os atores presentes no sistema e as suas tarefas. Tendo isso em conta, foram definidos como atores o próprio utilizador como único ator para os casos de uso do utilizador e, para os casos de uso do sistema, foram considerados como atores o *Frontend*, *Backend* e o *Jenkins* na sua função de aplicação de integração.

⁴<https://codenarc.org/>

⁵<https://owasp.org/www-project-dependency-check/>

⁶<https://www.jenkins.io/>

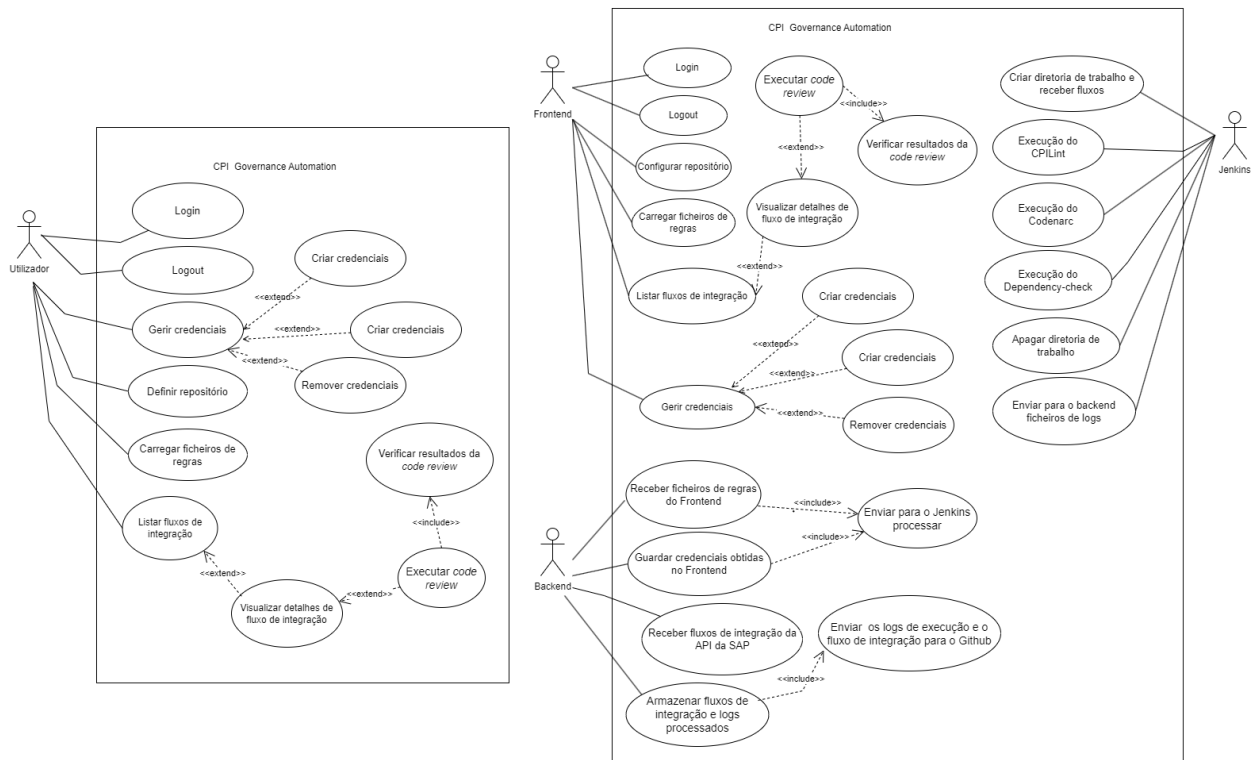


Figura 4: Diagramas de casos de uso do utilizador e do sistema, respetivamente

5 Catalogação das melhores práticas

Nos objetivos da componente principal do projeto, estavam previstas a catalogação das melhores práticas para desenvolvimento de fluxos de integração de *SAP CPI*, de código *Groovy* e das regras cobertas atualmente pelo *CPILint*. Nesse âmbito foi desenvolvido um documento intitulado ‘*CPI Governance Automation: Estudo do tema*’ que foi entregue ao orientador na fase inicial de preparação do projeto.

Esse documento encontra-se em anexo e contém satisfatoriamente, para os orientadores empresariais, a catalogação de melhores práticas de *SAP CPI* e *Groovy*. Além disso, possui uma catalogação das regras atualmente cobertas pelo *CPILint* e um estudo das potenciais regras a serem adicionadas futuramente. No entanto, o âmbito do projeto sofreu alterações ao longo do desenvolvimento inicial para não incluir a adição de novas regras no *CPILint*, mas sim para fornecer uma integração e entrega contínuas do *CPILint* em conjunto com outras ferramentas de *code review*, abordada nas próximas secções. A título de exemplo, algumas regras foram catalogadas para adição ao *CPILint* foram precisamente as funcionalidades acrescentadas através da utilização de *CodeNarc* e *DependencyCheck*, ou seja, deteção de vulnerabilidades dos ficheiros JAR presentes nos fluxos de integração e a deteção de más práticas de desenvolvimento em *scripts Groovy*, respetivamente, cumprindo assim esse requisito.

6 Arquitetura da solução

Neste secção, discute-se de forma geral a arquitetura seguida de modo a atingir os objetivos estabelecidos para o projeto.

6.1 Overview da configuração Docker

De acordo com o processo de discussão do âmbito do projeto por parte da empresa e das funcionalidades que seriam fulcrais para o sucesso do sistema, foi idealizada uma arquitetura baseada na utilização de *containers Docker* (Figura 5).

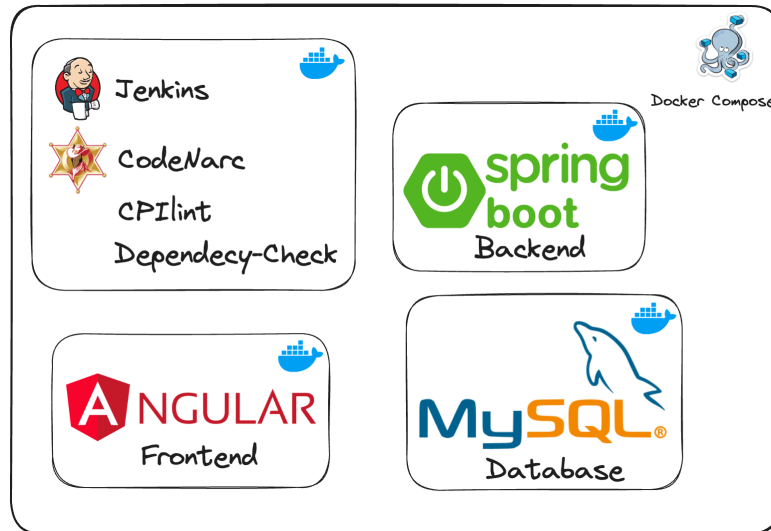


Figura 5: Arquitetura do sistema com a vista dos seus *Docker Containers*

Esta arquitetura surge com base na divisão de responsabilidades e modularidade entre os vários componentes. Dessa forma, de acordo com as funcionalidades necessárias para o projeto, utiliza-se um *container* para a implementação do *backend* do sistema, outro *container* para o *frontend*, um *container* para a base de dados e, por último, um *container* com que encapsula o serviço *Jenkins* e os três programas orientados às verificações dos fluxos de integração — *CPILint*, *CodeNarc* e *DependencyCheck*. Por forma a facilitar o *deployment* dos vários componentes, tornou-se evidente que era necessário recorrer à ferramenta de orquestração *docker-compose*, sendo a arquitetura especificada segundo um ficheiro de configuração *YAML*.

Para a base de dados, optou-se por recorrer à versão mais recente da imagem do MySQL, com o *container* a ser adequadamente configurado com utilizador, *password* e nome da base de dados com a qual o *backend* irá interagir.

Por sua vez, no *backend* recorre-se a uma imagem local, baseada na imagem do *Maven* (*OpenJDK 17*), de modo a produzir o JAR executável que contém o servidor *REST* baseado no *framework* de *Java Spring*. A configuração do servidor (e.g. *connection string* para a base de dados, caminhos para ficheiros necessários, portas, etc.) é fornecida através de variáveis de ambiente na secção apropriada do ficheiro de orquestração, sendo a API posteriormente exposta externamente na porta 9001.

Por sua vez, a imagem do *frontend* é baseada na imagem do *Nginx*, sendo, por isso, um servidor *web* que irá consumir a API implementada no *backend*. O servidor *web* é exposto externamente na porta 4200.

Finalmente, para o serviço do *Jenkins*, parte-se da imagem base desse mesmo serviço, instalando adicionalmente as ferramentas para os processos de *review* supra-mencionadas. Neste caso em particular, o serviço necessita de acesso a ficheiros externos para configuração das ferramentas de *review*, bem como aos respetivos fluxos de integração. Além disso, estas ferramentas produzem ficheiros de *log* que serão usados pelo *backend*, que são criados na diretoria dos dados internos ao *Jenkins*. Como tal, é necessário recorrer a dois volumes (i.e. diretorias locais mapeadas no *container*), um para os dados internos do *Jenkins* e outro para os ficheiros externos, de modo a que estes fiquem acessíveis ao *backend*. O serviço fica exposto externamente na porta 8080.

Adicionalmente, de modo a garantir a comunicação entre os vários *containers*, é definida uma rede interna *default* (*project-network*), ou seja, sem qualquer tipo de configuração específica.

6.2 Tecnologias utilizadas no sistema

Em seguida é feita uma descrição das tecnologias escolhidas para desenvolver a aplicação proposta, de forma a explicar o âmbito que cada componente terá no funcionamento global da solução e a motivação que levou a essa escolha. Após maior discussão nas semanas iniciais entre o *Product Owner* e a *Accenture*, foi decidido que teríamos a total liberdade de escolha das tecnologias a utilizar para desenvolver o projeto, de maneira a facilitar a implementação em tempo útil do mesmo.

6.2.1 Backend

Optamos por utilizar Java e Spring Boot no desenvolvimento do backend, oferecendo suporte tanto à interface web quanto às funcionalidades do *CPILint*. Essa decisão foi tomada de forma consensual e considerada a mais apropriada para o grupo.

O backend é responsável por definir os *endpoints*, configurando uma API *Rest* em Java, que serão invocados pelo *frontend* para executar operações solicitadas pelo utilizador. Isso inclui interações com o *CPILint* para aplicar revisões de integrações de acordo com as regras carregadas na base de dados pelo utilizador.

Em outras palavras, a API definida permite acionar o *CPILint* para aceder remotamente o *CPI Tenant* e realizar o processo de revisão. Ao final desse processo, o *backend* alimenta a página web com o conteúdo dos erros gerados pelo *CPILint* que será exibido na página *web*.

O backend é composto por várias *packages*, cada uma contendo *controllers* responsáveis por diferentes funcionalidades. Normalmente, cada *controller* possui um *Service* e um *Repository* associado. Abaixo encontram-se as *packages* principais:

- **Auth e User:** Estas *packages* são importantes no que toca à implementação de autenticação em *Spring Boot*. O serviço de autenticação dispõe de métodos para registar novos utilizador, validando a exclusividade do e-mail, criando um novo utilizador e guardando as respetivas informações no repositório, bem como autenticar utilizadores existentes.

A autenticação faz uso do *AuthenticationManager* para validar as credenciais, gerando um token *JWT* para o utilizador autenticado. Para a gestão e acesso às informações dos utilizadores no sistema, é feito uso do *UserService*.

- **Credentials:** A *Credentials* no projeto está relacionada à gestão de credenciais. Cada *subpackage* dentro dela lida com um tipo específico de credencial (github, jenkins ou SAP CPI), com *endpoints* para guardar, atualizar, listar e eliminar credenciais.
- **Repository:** Responsável pela gestão dos repositórios GitHub dentro do sistema.
- **Packages:** Esta *package* contém classes relacionadas à gestão de pacotes e fluxos de integração.

PackagesController: Este é um controlador que lida com operações relacionadas a pacotes e fluxos de integração. Alguns dos principais métodos incluem:

- *enableJenkins*: Inicia e executa um pipeline no Jenkins com base em parâmetros fornecidos.
- *getPackages*: Obtém informações sobre todos os pacotes disponíveis.
- *getPackage*: Obtém informações sobre um pacote específico.
- *getPackageFlows*: Obtém informações sobre os fluxos associados a um pacote.
- *getFlow*: Obtém informações sobre um fluxo específico.
- *downloadFlow*: Faz o download do conteúdo de um fluxo.

- `uploadFlowZip`: Realiza o upload de um ficheiro `.zip` contendo um fluxo.
- `enableGithub`: Faz o download de um fluxo, converte para o formato `.zip` e envia para o GitHub, associado a um *branch* específico.

Consideremos o exemplo do *endpoint* associado à obtenção de um fluxo, a fim de demonstrar a utilização de DTOs (Data Objects), ou seja, estruturas de dados que facilitam a transferência de informações entre *backend* e *frontend*.

- `FlowResponseDTO`: A classe `FlowResponseDTO` funciona como uma estrutura de transferência de dados e contém uma instância da classe `IntegrationFlow`. Essa classe é anotada com `@JsonDeserialize` para personalizar o processo de desserialização.
- `FlowResponseDTODeserializer`: O `FlowResponseDTODeserializer` é um desserializador personalizado projetado para a classe `FlowResponseDTO` que desempenha um papel crucial ao extrair informações e converte essas informações numa instância da classe `IntegrationFlow`. Finalmente, essa instância é configurada em um objeto `FlowResponseDTO`.
- `IntegrationFlow`: A classe `IntegrationFlow` é responsável por representar detalhes específicos de um fluxo de integração. Ela abrange informações essenciais, como ID, versão, Package ID, nome, descrição, criado por, data de criação, modificado por e data de modificação. Em resumo, essa classe é fundamental para encapsular dados associados a um fluxo de integração específico.

Ao utilizar este conjunto de classes, o sistema é capaz de transferir informações sobre fluxos de integração de maneira eficiente e estruturada. A anotação `@JsonDeserialize` e o desserializador personalizado fornecem um controle preciso sobre como os dados são convertidos entre a representação em JSON e as instâncias de classe no código Java. Este mecanismo é crucial para garantir uma comunicação fluida e consistente entre diferentes partes do sistema que lidam com dados de fluxos de integração.

- **Jenkins**: Oferece funcionalidades relacionadas ao *Jenkins*, como criação de *jobs*, execução de *pipelines*, verificação do estado do *build* e obtenção de relatórios. Inclui métodos para criar, executar, atualizar ficheiros Jenkins, além de gerenciar a execução de pipelines.
- **Configuration Files**: Oferece funcionalidades relacionadas aos ficheiros de configuração do Codenarc (ficheiros `.groovy`) e ficheiros de regras do CPILint (ficheiros `.xml`).

6.2.2 Frontend

Inicialmente, estava prevista a utilização de *SAP UI5* para o desenvolvimento do *Frontend*, mas acabou por ser escolhido o *Angular* como solução para o desenvolvimento do *Frontend*, visto haver uma maior familiarização por parte dos membros do grupo e uma menor curva de aprendizagem.

Nesta secção, abordamos a implementação do *frontend* do projeto, desenvolvido com recurso à *framework Angular* e à linguagem de programação *TypeScript*.

Para ilustrar o funcionamento das páginas da nossa interface *web*, foquemo-nos no componente principal *PackageDetailComponent* e sua integração com serviços e módulos do Angular, que servirá como um bom exemplo de como foi desenvolvido o *frontend*.

1. **Componentes Angular**: No núcleo de cada página encontra-se um componente do *Angular*. Ora, neste exemplo concreto, o *PackageDetailComponent* é a peça central para exibir detalhes de pacotes e fluxos ao utilizador na interface *web*. Estes componentes normalmente são definidos em *typescript* num ficheiro próprio onde se encontram propriedades e métodos específicos do componente em questão, e são acompanhados de um ficheiro *html* e um ficheiro de estilos *css*.

2. **Propriedades do Componente:** No ficheiro *typescript* principal, são definidas propriedades específicas de cada componente individual. Neste caso concreto, existem propriedades que servem o propósito de armazenar detalhes do pacote (*packageDetails*), ID do pacote (*packageId*), e dados do fluxo (*dataSource*, *displayedColumns*). Estas propriedades são por vezes utilizadas para alimentar o conteúdo do *html*, isto é, podem ser obtidas através de chamadas a *APIs* externas, armazenadas em variáveis e posteriormente referenciadas no *html* respetivo ao componente, possibilitando assim a sua exibição na tela do utilizador.
3. **Integração de Serviços:** Por convenção, e com vista a implementar boas práticas, nos ficheiros *typescript* evita-se realizar lógica de chamadas a *APIs* e *endpoints*. Na verdade, o que se faz é definir um método simples que faz uso de um *Service* que encapsula essas chamadas. O *PackageDetailComponent* realiza uma integração eficaz com o serviço *PackageDetailService*, um serviço Angular responsável por comunicar-se com a camada de *backend* da aplicação. Essa integração de serviços não apenas proporciona uma comunicação eficiente com o *backend*, mas também promove uma arquitetura modular e extensível, permitindo futuras atualizações e inclusões de funcionalidades de forma coesa e escalável. Eis os aspetos-chave desta integração:
 - **Service Layer (PackageDetailService):** O serviço é cuidadosamente projetado para encapsular todas as chamadas de API relacionadas aos detalhes do pacote e aos fluxos. Adota boas práticas, como a definição de constantes para URLs de API (manutenção de código) e a manipulação adequada de cabeçalhos HTTP quando necessário.
 - **Métodos definidos no Componente:** Como já foi mencionado nesta secção, procura-se definir métodos no componente que façam uso de um *Service* que encapsula as chamadas às APIs. A título de exemplo, foquemo-nos no método de obtenção de detalhes de um pacote. O modo de funcionamento é o seguinte: É definido um método no componente que invoca o método *getPackageDetails(packageId: string)* do serviço para obter detalhes específicos do pacote identificado por *packageId*. Os detalhes obtidos são então atribuídos à propriedade *packageDetails* do componente para atualização dinâmica na interface do utilizador.
 - **Tratamento Assíncrono com Observables:** A utilização de *Observables* do *RxJS* é uma prática robusta para lidar com operações assíncronas, garantindo uma programação reativa. Os métodos de serviço que envolvem chamadas HTTP retornam *Observables*, permitindo que o componente reaja dinamicamente às respostas do servidor quando estão prontas.
 - **Tratamento de Erros:** A implementação cuidadosa do serviço inclui tratamento de erros para situações como falha na chamada da API. A utilização de *catchError* assegura que o componente receba uma resposta consistente, mesmo em casos de falha na recuperação de dados.
4. **Integração com Material Design:** Para construir as interfaces gráficas foi feito uso de componentes da biblioteca *Angular Material Design* de modo a implementar uma interface de utilizador consistente e responsiva, incluindo tabelas (*mat-table*), spinners (*mat-spinner*), paginadores (*mat-paginator*), diálogos que "abrem" detalhes de fluxos, entre outros.
5. **Internacionalização (i18n):** Todos os componentes da aplicação adotam uma abordagem proativa para internacionalização (i18n) através da integração do módulo *@ngx-translate/core*. Este módulo proporciona uma solução eficaz para suporte a vários idiomas, permitindo que a aplicação se adapte dinamicamente às preferências linguísticas do utilizador. Para isso, os componentes fazem uso do *TranslateModule*, o qual possui uma configuração de idiomas suportados e ficheiros json definidos pela equipa de desenvolvimento contendo as frases e/ou mensagens traduzidas que deverão ser exibidas nas telas conforme o idioma selecionado. O *TranslateModule* permite a atualização dinâmica do idioma em tempo de execução, logo conforme o utilizador altera suas preferências de idioma, a aplicação ajusta-se imediatamente sem a necessidade de recarregamento da página.

6. **Hooks do Ciclo de Vida:** No âmbito da arquitetura Angular, os *hooks* do ciclo de vida são métodos especiais que são chamados automaticamente em momentos específicos durante a existência de um componente. O `PackageDetailComponent`, por exemplo, emprega um desses *hooks* para realizar ações importantes durante determinadas fases do ciclo de vida do componente. No `PackageDetailComponent`, o `ngOnInit` é implementado para reagir à inicialização do componente, aproveitando este momento para obter detalhes do pacote e fluxos associados.

6.2.3 Base de dados

A escolha da utilização de *MySQL* enquanto base de dados para o sistema desenvolvido prendeu-se pela simplicidade e rigidez dos dados a serem tratados, visto haver lugar ao armazenamento de credenciais da nossa aplicação, credenciais essas necessárias para o processo de revisão automática de fluxos de integração de *SAP CPI* e dos ficheiros de regras e de configuração desse mesmo processo e ferramentas utilizadas no mesmo. Além disso, grande parte do grupo possuía um bom entendimento da tecnologia escolhida, o que facilitou essa decisão. Na Figura 6, está representado o modelo físico da base de dados do sistema. São definidas uma série de tabelas de modo a suportar os dados necessários para o *backend*. A tabela `user` contém a informação fornecida aquando do registo na plataforma por parte do utilizador, para além do *role* que indica se se trata de um administrador ou não. A sua chave primária é uma chave estrangeira nas tabelas referentes às credenciais fornecidas pelo mesmo, nomeadamente `credential_sap_cpi`, `credential_jenkins` e `credential_github`, visto que cada tipo de credenciais fornecidas estão associadas a um utilizador. Estas tabelas contêm dados necessários para as *APIs* de cada um destes serviços, podendo esses ser *tokens* de acesso, *URLs*, entre outros. Por sua vez, a tabela `repository_github` recebe a chave estrangeira da tabela das credenciais do Github, contendo também a informação relativa ao nome do repositório e nome dos *branches*. Finalmente, existem duas tabelas para suportar os ficheiros de regras, tanto do *CodeNarc* como do *CPILint*, guardados sob a forma de *blobs*.

6.2.4 Jenkins

A utilização de um servidor de automação de código como o *Jenkins* foi sugerida por parte da Accenture, após a realização dos *workshops* de discussão do projeto e das funcionalidades do mesmo. Foi evidente a melhoria que traria como um todo ao sistema, a utilização de uma aplicação deste tipo, para facilitar e fornecer uma utilização mais simples ao utilizador final. Tendo em conta a possibilidade de utilizar o sistema de *pipelines Jenkins* para ativar cada ferramenta de *review* utilizada no processo.

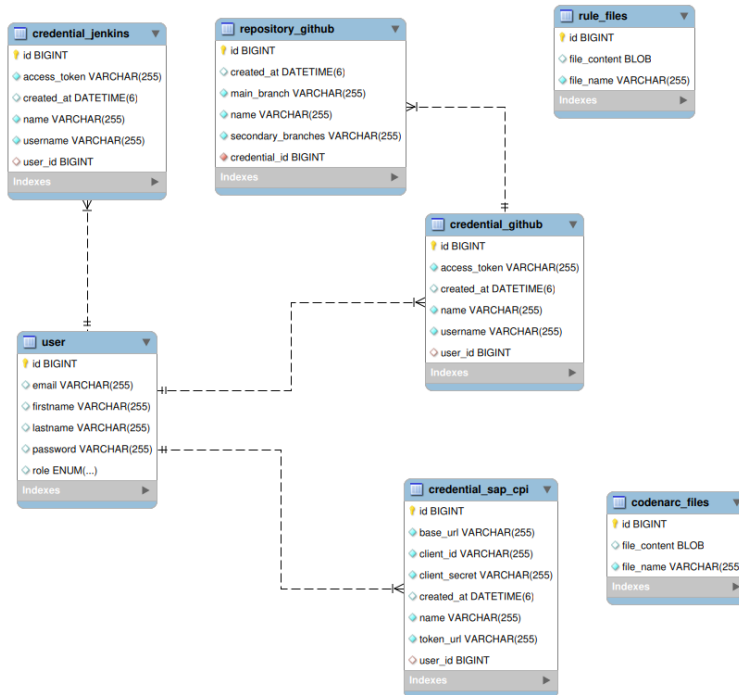


Figura 6: Modelo físico da base de dados

7 Segunda componente

Tendo em conta a modesta interface do *CPILint*, visto tratar-se de uma aplicação de linha de comandos, foi imperativo desenvolver uma versão *web* que facilitasse o seu uso e que pudesse substituir a funcionalidade básica do programa de modo a obtermos uma verdadeira aplicação de governança para o *CPILint*. Aliado ao uso do trabalho desenvolvido para a primeira componente deste projeto, foi possível desenvolver uma interface *web* que fizesse a governança automática do processo de *code review* realizado pelo *CPILint*, aproveitando as melhorias feitas através da utilização de *Jenkins* para a automação de código, para facilitar a integração e entrega contínuas do projeto.

7.1 Objetivos

De modo a atingir o objetivo proposto para a segunda componente por parte da *Accenture* foram definidos, em conjunto com o nosso orientador empresarial os pontos chave a atingir:

- Criação e utilização de um *pipeline CI/CD* para utilizar de forma remota o *CPILint*;
 - Ativar remotamente o processo de revisão de código;
- Criar uma interface *web*, que preveja o seguinte:
 - Possibilitar a inserção das credenciais necessárias ao processo de revisão de código automática (*Github*, *SAP* e *Jenkins*);
 - Possibilitar o *download* de fluxos de integração *SAP CPI* através da *API* existente para o efeito;
 - Possibilitar a inserção dos ficheiros de regras necessários (*CPILint* e *Codenarc*);

- Substituir os comandos de texto na linha de comandos do *CPILint* por botões/formulários;
- Possibilitar o *upload* de ficheiros revistos para repositórios no *Github*;
- Expor o relatório de revisão de código gerado pelo *CPILint* e demais ferramentas de revisão automática empenhadas (*CodeNarc* e *DependencyCheck*) no processo numa página para o efeito.

7.2 Desenvolvimento de *Mockups*

Durante o planeamento das interfaces *web* que seriam utilizadas para satisfazer os objetivos da segunda componente do projeto, foi utilizado o *Figma* para o desenvolvimento das *mockups* principais, que se encontram disponíveis de forma [online](#). De seguida, será apresentada a *mockup* que representa a página de configuração de credenciais ou acessos a que o utilizador teria que preencher, de modo a ser possível utilizar o sistema. Existem três credenciais a preencher, uma seria a conta de *Github* e o repositório que procura utilizar, outra seria relativamente à conta do *BTP Cockpit* da *SAP*, para obter os fluxos de integração necessários, e, por último, as credenciais da conta do *Jenkins* para realizar a criação e execução dos *pipelines*.

Figura 7: *Mockup* de configuração de credenciais

7.3 Interfaces finais

Nesta subsecção serão apresentadas algumas interfaces *web* representativas de páginas importantes para a utilização do sistema, criadas na solução desenvolvida pelo grupo.

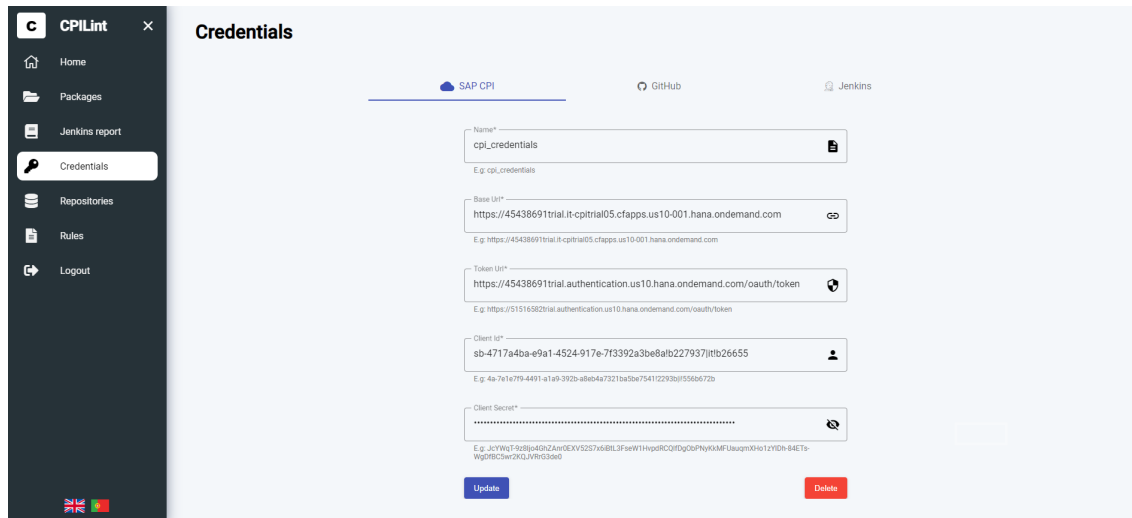


Figura 8: Interface final de configuração de credenciais

Na Figura 8 é apresentada a interface final de configuração de credenciais, que mantém as funcionalidades principais representadas na sua *mockup* original, oferecendo a possibilidade de registar credenciais para o *SAP CPI*, *Github* e *Jenkins*.

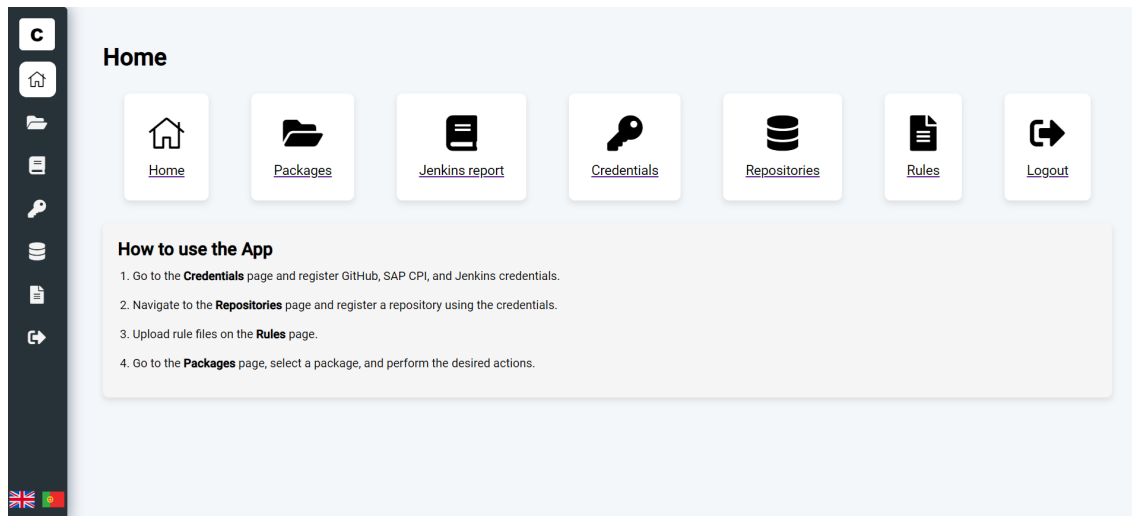


Figura 9: Página inicial

A interface final da página inicial está representada na Figura 9, nesta é possível navegar pelas principais páginas da solução, nomeadamente a configuração de credenciais, de regras e do repositório e também visualizar todos os *packages*.

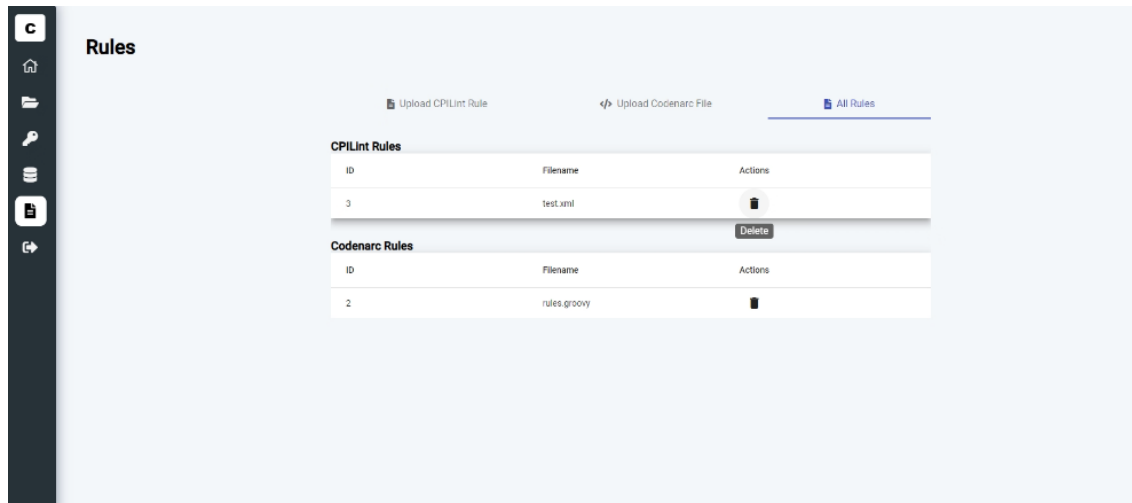


Figura 10: Página de gestão de ficheiros de regras

Na Figura 10, está representada a página de gestão de ficheiros de regras, onde é possível remover ficheiros específicos da base de dados, mas também realizar o carregamento de novos ficheiros de regras, tanto para o *CPILint*, como para o *CodeNarc*.

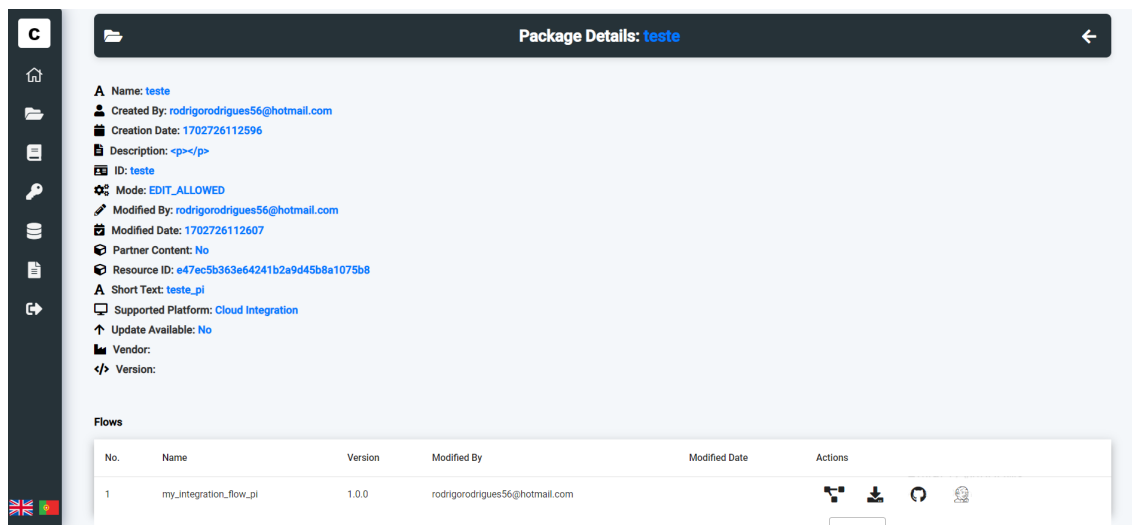


Figura 11: Página de execução do processo de *code review*

Analisando a parte superior da Figura 11, é possível verificar os detalhes relativamente à conta de *SAP Cloud Integration* em utilização, representados na interface de Pacotes. Relativamente aos fluxos de integração, na parte inferior da interface são mostrados os fluxos existentes e, além de informações relativas à versão em que se encontra e sobre o utilizador que o modificou, são visíveis quatro botões, um para cada ação possível no programa.

Jenkins report

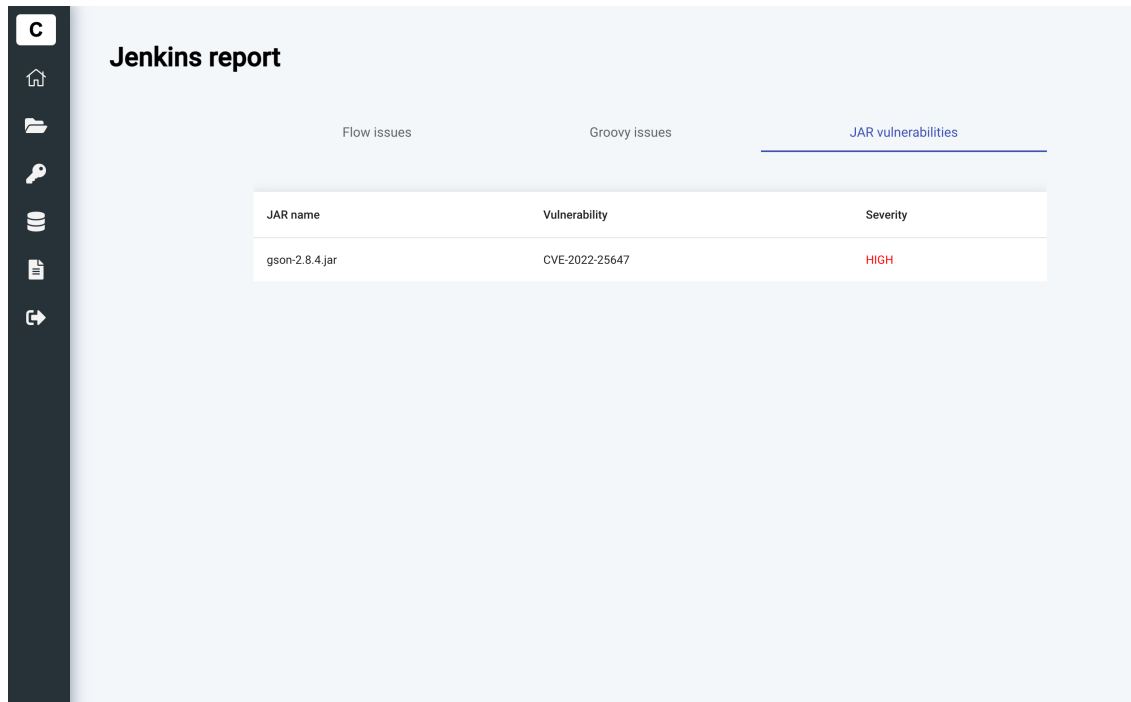
Flow issues **Groovy issues** JAR vulnerabilities

Total files with violations: 2
Total violations: 28

File name	Line	Priority	Violation
script2.groovy	6	3	UnnecessarySemicolon
script2.groovy	7	3	UnnecessarySemicolon
script2.groovy	8	3	UnnecessarySemicolon
script2.groovy	9	3	UnnecessarySemicolon
script2.groovy	13	3	UnnecessarySemicolon
script2.groovy	17	3	UnnecessarySemicolon
script2.groovy	18	3	UnnecessarySemicolon
script2.groovy	19	3	UnnecessarySemicolon
script2.groovy	20	3	UnnecessarySemicolon
script2.groovy	22	3	UnnecessarySemicolon
script2.groovy	23	3	UnnecessarySemicolon
script2.groovy	24	3	UnnecessarySemicolon

Figura 12: Relatório de execução do CodeNarc

Relativamente aos relatórios de execução das três ferramentas utilizadas, na Figura 12 é possível verificar a existência de violações num *script* Groovy que se encontra dentro de um fluxo de integração testado, mostrando o tipo de violação, a prioridade definida no ficheiro de regras, a sua localização e o *script* em específico. Além disso, é possível verificar as outras abas para problemas relativamente aos fluxos detetados pelo *CPILint* e vulnerabilidades detetadas nos arquivos JAR. No caso destas últimas, a cor apresentada é relativa à severidade da vulnerabilidade, de acordo com o seu CVSS ("*Common Vulnerability Scoring System*"), por exemplo, se a severidade for "*High*", a cor apresentada é vermelha, como se vê na Figura 13.



JAR name	Vulnerability	Severity
gson-2.8.4.jar	CVE-2022-25647	HIGH

Figura 13: Relatório de execução do DepedencyCheck

8 Revisão automática de código *Groovy* e *Java*

Um dos objetivos principais do projeto era, tendo em conta que o *CPILint* apenas verificava a correção dos fluxos de integração de *SAP CPI* e não dos *scripts Groovy* que por vezes os acompanham, possibilitar que houvesse a utilização de ferramentas de verificação automática de código acoplados à utilização do *CPILint* que combatessem essa lacuna no programa original. Tendo isso em conta, foi decidida a necessidade da utilização de *CodeNarc*, aplicação de linha de comandos que analisa ficheiros *Groovy*, reportando os seus defeitos — más práticas, inconsistências, problemas de estilo, entre outros. Além disso, de modo a possibilitar a verificação de ‘*Common Vulnerabilities and Exposures*’⁷ da base de dados do *MITRE* nos arquivos JAR, foi implementada também a utilização de *DependencyCheck*, que é uma ferramenta de Análise de Composição de *Software* (*SCA*) para detetar vulnerabilidades publicamente conhecidas dentro das dependências de um projeto.

Para fazer a integração dessas duas ferramentas com o *CPILint*, foi utilizado um esquema de *pipeline* do *Jenkins*, em que o utilizador apenas fornece (além das credenciais necessárias para o acesso ao *Jenkins*) os ficheiros de regras relativos ao *CPILint*, ao *CodeNarc* e o próprio ficheiro de configuração do *pipeline*, o qual é possível utilizar de forma praticamente contínua visto haver uma atualização das diretorias necessárias de forma automática no código desenvolvido pelo grupo. Os ficheiros de regras podem e devem ser alterados pelo utilizador final em função das necessidades de revisão para os fluxos de integração em questão e para os *scripts Groovy* presentes, visto haver um conjunto variado de regras implementadas em ambas as ferramentas para as situações específicas de cada caso.

O *pipeline* do *Jenkins* que faz a execução de todo o processo, consiste em 4 *stages* e uma etapa pós construção do *pipeline* de limpeza do ambiente de trabalho.

Inicialmente, existe uma etapa chamada ‘*Unzip*’ que faz a descompressão do ficheiro que contém o fluxo de integração, obtido anteriormente e coloca os seus conteúdos numa diretoria criada para o efeito.

⁷<https://nvd.nist.gov/vuln>

```

stage('Unzip') {
    steps {
        echo 'Unzip'
        sh 'mkdir /files/unzip_flow'
        sh 'unzip ${FlowZip} -d /files/unzip_flow'
    }
}

```

Posteriormente, é iniciada a fase de revisão de código em específico com a execução do *CPILint* na sua própria etapa, havendo também a criação de um ficheiro relativo aos ficheiros de *logs* gerados por ele. Para a execução desta etapa, o *CPILint* precisa de receber, além do fluxo de integração, um ficheiro em formato XML de regras a verificar, que é obtido antes do início do *pipeline* e colocado o seu caminho numa variável global '*CPILintRules*'.

```

stage('CPILint') {
    steps {
        script {
            echo 'CPILint'
            def cpilintError = false
            def cpilintLogFile = `${env.WORKSPACE}/cpilint.log'
            catchError(buildResult: 'UNSTABLE', stageResult: 'FAILURE') {
                script {
                    def cpilintOutput = sh(
                        script: "/cp/cpilint-1.0.4/bin/cpilint -rules ${CPILintRules} -files
                                ${FlowZip} -debug > ${cpilintLogFile} 2>&1",
                        returnStatus: true
                    )
                    cpilintError = cpilintOutput != 0
                }
            }
            currentBuild.result = cpilintError ? 'FAILURE' : 'SUCCESS'
        }
    }
}

```

De maneira a possibilitar a revisão do *scripts Groovy* contidos nos fluxos de integração do *SAP CPI*, foi utilizada uma etapa relativa à execução do programa *CodeNarc*, que além dos fluxos de integração em si necessita também de um ficheiro de regras em formato XML, recebendo o caminho relativo para o mesmo através de uma variável global '*CodenarcRules*'.

```

stage('CodeNarc') {
    steps {
        echo 'CodeNarc'
        sh "java -cp /cp/codenarc.jar org.codenarc.CodeNarc
            -report=json:output.json -rulesetfiles=file:${CodenarcRules}
            -basedir=/files/unzip_flow"
    }
}

```

A fase final do *pipeline* consiste na revisão dos ficheiros JAR que se encontram dentro do fluxo de integração como potenciais dependências dos *scripts*. Nesta etapa é utilizada a ferramenta *DependencyCheck* que depois de executada gera um relatório com as vulnerabilidades conhecidas contidas nas dependências do projeto.

```

stage('Dependency check') {

```

```

steps {
  echo 'Dependency check'
  sh '/cp/dependency-check/bin/dependency-check.sh
    --nvdApiKey=f906660d-70f5-4201-ab55-5e04202f34f9
    --format JSON --prettyPrint -scan /files/unzip_flow'
}
}

```

A etapa posterior à construção do *pipeline* apenas remove de forma recursiva a diretoria onde foi colocado inicialmente o fluxo de integração a passar pelo processo de revisão, de forma a garantir consistência entre execuções, ou seja, se houver quebra de alguma fase do pipeline devido a ficheiros de regras incorretos, a diretoria temporária seria sempre removida, garantindo que não haveriam dois fluxos diferentes em análise de forma indevida.

```

post {
  always {
    sh 'rm -rf /files/unzip_flow'
  }
}

```

9 Guia de utilização da solução

Tendo por base o sistema a funcionar em servidor remoto (futuramente) ou através da utilização de uma aplicação como o *Docker Desktop*, encontrando-se na página de *Login*, o utilizador deve seleccionar a opção de se registar no *website*. Finalizado o registo, o utilizador deve proceder ao *login* e se as credenciais estiverem corretas será levado a uma página inicial.

O próximo passo é fazer o registo das credenciais necessárias, ou seja, do *Github*, do *Jenkins* e do *SAP BTP Cockpit*. De seguida, o utilizador deve navegar até à página de repositórios, na qual deve registar um repositório *Github* para possibilitar o armazenamento dos fluxos de integração e dos ficheiros de *report* gerados na execução do programa. Além do mais, é necessário que realize o carregamento dos ficheiros de regras do *CPILint* (em formato XML) e do *CodeNarc* (em formato *Groovy*), na página existente para o efeito.

Por último, na página de artefactos, o utilizador selecciona o artefacto para o qual pretende que se gere o relatório de revisão de código, escolhendo os ficheiros de regras a utilizar para essa mesma revisão e o *branch* do repositório *Github* que pretende que seja utilizado. Através de um botão, executa o *pipeline* do *Jenkins*. Posto isto, será mostrado uma página com os relatórios de execução, um para cada ferramenta utilizada. Esses ficheiros, bem como o fluxo de integração são então carregados no repositório de *Github* previamente configurado para o efeito.

10 Trabalho futuro

Embora tenham sido atingidos os principais objetivos, existe possibilidade de alargar e desenvolver de forma mais aprofundada certas funcionalidades que melhorariam a utilização do sistema em contexto real e pela própria *Accenture*. Por exemplo, de modo a possibilitar a utilização do sistema para vários clientes finais, seria benéfico existir a possibilidade de registar várias credenciais para as tecnologias apresentadas (*SAP BTP Cockpit*, *Github* e *Jenkins*). Além disso, de modo a melhorar o processo de correção de erros e vulnerabilidades identificados nos ficheiros verificados no decorrer do processo de *code review* seria necessário manter no repositório alvo, que recebe os ficheiros de *report* de execução do *CPILint*, *CodeNarc* e *DependencyCheck*, as versões antigas fruto da execução do programa de governança desenvolvido. Neste intuito, também faria sentido permitir que o utilizador tivesse a possibilidade de fazer o descarregamento dos ficheiros de *report* diretamente na interface *web*, possivelmente na página que os apresenta atualmente. Ainda na execução do *pipeline* que realiza o processo de *code review*, futuramente em contexto empresarial

seria uma mais valia a possibilidade de alterar as configurações do *pipeline* através de um formulário *web*, de modo a permitir ao utilizador seleccionar as ferramentas que pretende que façam a revisão, não sendo necessárias utilizar todas as disponíveis, em casos específicos de execução em que não sejam essenciais.

Relativamente à correção das falhas encontradas, poderia ser vantajoso a adoção de um sistema de recomendação automático de correção, por exemplo, das vulnerabilidades encontradas pelo *DependencyCheck* que utilizasse inteligência artificial, como um *LLM*, de modo a facilitar o processo de correção das mesmas ou até da adoção de práticas melhores no desenvolvimento de integrações de *SAP CPI*.

11 Conclusão

A utilização de ferramentas de revisão automática de integração em nuvem *SAP* é uma mais valia para grandes organizações atuais, onde esse *software* é amplamente utilizado para gestão, integração e análise de sistemas. Deste modo, existe a necessidade da criação de uma aplicação que automatizasse o processo de governança para artefactos de *SAP Cloud Platform Integration* que seja de utilização intuitiva e não necessite de grande envolvimento por parte dos seus utilizadores finais, ou seja, facilitando todo o processo de *code review* através da adoção de outras ferramentas já existentes no mercado para estender as funcionalidades de um programa como o *CPILint*, nomeadamente nesta fase, o *CodeNarc* e o *DependencyCheck*.

O *CPILint* é uma ferramenta de importância significativa para desenvolvedores de integrações no *SAP BTP Cockpit*, visto facilitar todo o processo de verificação das mesmas. Este projeto proposto pela *Accenture* procurou facilitar o acesso ao próprio *CPILint* de forma extensível, demonstrando esse ponto ao utilizar *pipelines* de *Jenkins* para promover uma integração e entrega facilitadas das ferramentas nomeadas anteriormente. Embora não seja uma solução inovadora, procura facilitar o acesso a ferramentas existentes e a sua utilização de modo conjunto, além de permitir que o próprio utilizador final futuramente consiga fazer a integração de outras ferramentas de revisão de código.

Tendo em conta os objetivos propostos, a solução obtida pelo grupo passou pela utilização de *Docker* como ferramenta para organização arquitetural, sendo desde início uma grande preocupação para o desenvolvimento deste projeto. Após finalmente obtermos um ambiente funcional, com os *containers* e volumes necessários à utilização das ferramentas em questão, foi necessário um grande esforço para fazer a utilização do *Jenkins* de modo benéfico para o projeto, permitindo a automatização que era pretendida.

De modo geral, foram atingidos os principais objetivos do projeto para ambas as componentes, reconhecendo também, aspetos referidos na secção anterior, que melhorariam o projeto como um todo, mas que não se enquadravam no tempo útil para a realização do projeto, não fazendo parte do escopo inicial do mesmo. Além disso, haverão certamente outros aspetos do sistema desenvolvido que têm potencial para serem melhorados.

Finalmente, agradecemos à *Accenture*, na pessoa do nosso orientador principal Roberto Brasil, e à equipa docente que se mostraram disponíveis para nos guiar durante o desenvolvimento deste projeto.