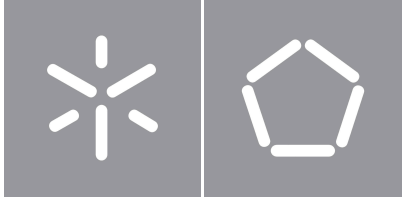
Universidade do Minho
Escola de Engenharia

José Pedro Fernandes Veloso

Geração automática de código Oracle Retail: Uma solução baseada em templates e Django

outubro 2023



Universidade do Minho
Escola de Engenharia

José Pedro Fernandes Veloso

Geração automática de código Oracle Retail: Uma solução baseada em templates e Django

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação de
Professor João Miguel Lobo Fernandes

outubro 2023

Direitos de Autor e Condições de Utilização do Trabalho por Terceiros

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho:

CC BY

https://creativecommons.org/licenses/by/4.0/

Agradecimentos

Para a construção da presente dissertação, não poderia deixar de dirigir um agradecimento a todos aqueles que, de alguma forma, me acompanharam e me apoiaram durante a elaboração da mesma. Sendo assim, em primeiro lugar, gostaria de agradecer ao meu orientador universitário, Prof. João Miguel Lobo Fernandes, pela sua orientação, pela disponibilidade e pelo feedback.

Em segundo lugar, dirijo um agradecimento meritório à Retail Consult, que representa o contexto prático na qual a tese se desenvolveu e concretizou e a todos os colaboradores com a qual tive oportunidade de me cruzar. Sem dúvida, desenvolver a dissertação nesta empresa foi uma experiência transformadora e enriquecedora aos mais variados níveis, quer pelos conhecimentos adquiridos, mas também pelos recursos ao qual tive acesso e pela produtiva partilha de ideias, as quais, foram fundamentais para melhorar a qualidade e a relevância investigação.

Em terceiro lugar, agradeço à estimada Universidade do Minho, que durante estes cinco anos representou a minha segunda casa e graças a esta consegui adquirir os princípios basilares para a construção da presente dissertação, bem como crescer pessoalmente e profissionalmente.

Por fim, mas não menos importante, agradeço à minha família e aos meus amigos, por me fornecerem o alento nos momentos que precisei, pelo incentivo e encorajamento constante, por serem o meu alicerce em todos os momentos desta jornada e pelas memórias criadas em conjunto.

A todos,

O meu mais sincero e humilde agradecimento.

Declaração de Integridade

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Universidade do Minho, Braga, outubro 2023

José Pedro Fernandes Veloso

Resumo

O Oracle Retail oferece um conjunto de aplicações de software que ajuda os retalhistas a gerir os seus negócios, incluindo ponto de venda, gestão de inventário, gestão de relações com clientes e gestão da cadeia de fornecimento.

No entanto há necessidade de adaptar a Solução Oracle Retail à realidade do cliente, isto é, fazer algumas personalizações à Solução. Estas necessidades dos clientes são várias vezes idênticas, sendo o volume de alterações por vezes elevado e repetitivo, mudando pequenas especificidades de cliente para cliente. Isto, por sua vez, resulta numa perda de tempo valioso que poderia ser alocado a tarefas mais produtivas.

Combinando esta realidade de trabalhos semelhantes com a pressão do mercado onde a disponibilidade e continuidade de recursos é cada vez mais complexa, as organizações devem procurar formas de automatizar estes processos e beneficiar da redução do tempo gasto tanto no desenvolvimento como na resolução de problemas.

À luz destes desafios, o desenvolvimento de uma aplicação destinada a gerar código Oracle Retail personalizado é uma solução promissora para responder às necessidades dos clientes na adaptação da solução Oracle Retail aos seus requisitos.

Palavras-chave Oracle Retail, Django, Gerador de Código, Templates, Gerador de Código com Templates, Django

Abstract

Oracle Retail offers a suite of software applications that help retailers manage their businesses, including point of sale, inventory management, customer relationship management and supply chain management.

However, there is a need to adapt the Oracle Retail Solution to the customer's reality, i.e. to make some customizations to the Solution. These customer needs are often identical, and the volume of changes is sometimes high and repetitive, changing small specifics from customer to customer. This, in turn, results in a waste of valuable time that could be allocated to more productive tasks.

However, there is a need to adapt the Oracle Retail Solution to the customer's reality, i.e. to make some customisations to the Solution. These customer needs are often identical, and the volume of similar changes, configurations and/or solution extensions that are applied is very high. This, in turn, results in a waste of valuable time that could be allocated to more productive tasks.

Combining this reality of similar work with market pressure where resource availability and continuity is increasingly complex, organisations must look for ways to automate these processes and benefit from reduced time spent on both development and troubleshooting.

In the light of these challenges, the development of an application designed to generate personalised Oracle Retail code is a promising solution for responding to the similar needs of customers in adapting the Oracle Retail solution to their requirements.

Keywords Oracle Retail, Django, Code Generator, Templates, Code Generator with Templates, Django

Conteúdo

Lista de Figuras

Lista de Tabelas

Acrónimos

E2E End-to-End Testing.

MVT Model-View-Template.

ORM Object Relational Mapper.

ReIM Retail Invoice Matching.

RMS Retail Merchandising System.

RWMS Retail Warehouse Management System.

VTL Velocity Template Language.

Parte I

Material Introdutório

Capítulo 1

Introdução

1.1 Contexto

De empresas tão grandes como o Walmart ou Sonae a empresas locais como a pequena loja perto das nossas casas, é provável que quase todos os dias confie nestes retalhistas para a compra dos bens e serviços necessários. Sendo uma indústria em constante crescimento, as vendas a retalho cresceram em média 3,7% anualmente de 2010 a 2019 e após a pandemia o retalho está a crescer a níveis nunca vistos em mais de 20 anos. As vendas a retalho cresceram 7% em 2020 e mais de 14% em 2021 [2].

A indústria do retalho é uma parte vital da economia global, com milhões de empresas em todo o mundo a oferecer uma vasta gama de produtos e serviços aos consumidores. De acordo com dados da National Retail Federation (NRF), o total de vendas a retalho nos Estados Unidos atingiu 4,06 triliões de dólares em 2020 [3].

O retalho é um campo em constante evolução, com as empresas a enfrentarem uma pressão crescente para se manterem à frente da concorrência e satisfazerem as necessidades dos seus clientes. Uma forma de os retalhistas enfrentarem estes desafios é implementando soluções tecnológicas para racionalizar as operações e melhorar a experiência de compra.

Uma das soluções é o Oracle Retail que fornece soluções tecnológicas para esta indústria e oferece uma gama de serviços concebidos para ajudar as empresas a otimizar as suas operações e a melhorar a experiência dos seus clientes.

O Oracle Retail Suite subdivide-se em três domínios: Planning and Optimizations Product Domain Oracle, Retail Merchandising Product Domain e o Oracle Retail Store/Warehouse Product Domain. Cada um destes domínios é composto por um conjunto de aplicações que se articulam com o Retail Merchandising System (RMS). O ultimo é a aplicação central do Oracle Retail Suite e a sua principal função é executar

atividades centrais de merchandising, incluindo gestão de produtos, reposição de stocks, compras, gestão de fornecedores e acompanhamento financeiro.

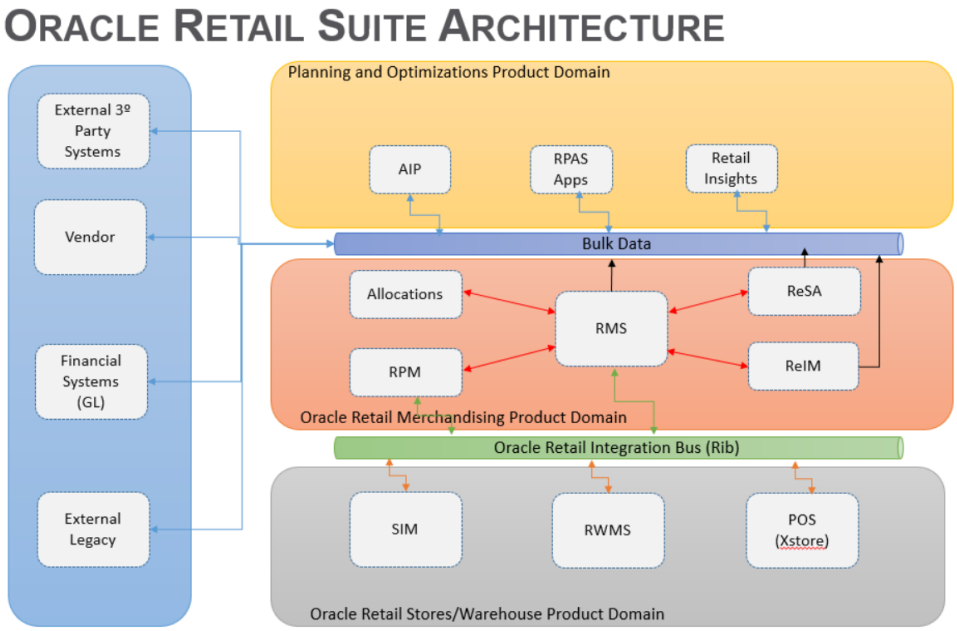


Figura 1: Arquitetura da Oracle Retail Suite retirado de documentação da Retail Consult

Algumas das aplicações que fazem parte dos domínios anteriormente referidos na figura 1 são:

- Retail Merchandising System (RMS)
- Retail Warehouse Management System (RWMS)
- Point of Sale (POS)
- Retail Invoice Matching (ReIM)
- Price Management (RPM)
- Retail Predictive Application Server (RPAS)

De uma forma geral, estas aplicações comunicam umas com as outras, por exemplo, o **RMS**, o **RWMS** e o **ReIM**, trabalham em conjunto para garantir uma gestão integrada e eficiente de mercadorias, logística e finanças no sector do retalho, permitindo um ciclo completo que vai desde o planeamento e compra de mercadorias, passando pela gestão de stocks e centros de distribuição, até à reconciliação de faturas e pagamentos, garantindo uma operação de retalho coesa e otimizada.

Globalmente, é uma solução que aumenta a eficiência, reduz custos e melhora a experiência do cliente. Como tal, o Oracle Retail é uma solução valiosa para os retalhistas que procuram manter-se competitivos e satisfazer as necessidades dos seus clientes.

1.2 Motivação

No decorrer de uma implementação de Oracle Retail, há necessidade de adaptar a Solução à realidade do cliente, isto é, fazer algumas customizações à Solução. De uma forma geral, caso um retalhista tenha alguma especificidade no que diz respeito ao cálculo de preço médio, deverá ser customizado o processo base do Oracle Retail para que de uma forma escalável consiga dar resposta ao processo do Cliente.

Esta solução é usada em 96 países, como afirmado pelo website da Oracle [4], sendo feitas todos os anos novas implementações para retalhistas em todo o mundo por diversas equipas.

Como visto anteriormente, com a pressão do mercado onde a disponibilidade e continuidade de recursos é cada vez mais complexa, as organizações devem procurar formas de automatizar estes processos e beneficiar da redução do tempo tanto no desenvolvimento como na resolução de problemas.

Usando para o efeito, um dos maiores retalhistas Nacionais, e tendo sido feita uma análise aos diversos projetos desenvolvidos em conjunto no decorrer dos anos que esta parceria existe, foi sendo verificado que na maioria dos projetos, os erros/falhas bem como o desenvolvimento em si eram de facto muito semelhantes. Tendo equipas muito rotativas, é difícil obter uma uniformização nos métodos de desenvolvimento e respetivos entregáveis, como tal, obter uma ferramenta responsável por criar *scripts* com base em *templates* vem colmatar estas dificuldades. Falhas como, falta de sinónimo público ou falta de instrução de "DROP TABLE"depois da primeira criação são alguns dos casos que não sendo uma falha critica muitas vezes trazem impactos na implementação do mesmo e retiram qualidade à entrega. Estes dois pontos, por exemplo, são facilmente ultrapassados com uma script sempre gerado com base no template.

A geração de código pode ser uma ferramenta valiosa para os programadores de software, uma vez que pode poupar tempo e esforço ao automatizar a criação de código repetitivo [5]. Além disto, também ajuda a impor normas de codificação e melhores práticas, uma vez que o código gerado aderirá à estrutura e à formatação definidas, tendo como resultado código com qualidade.

1.3 Principais Objetivos

O objetivo principal desta dissertação é desenvolver um gerador de código eficiente, projetado para automatizar a criação de código Oracle Retail repetitivo utilizando *templates* de código e a *framework* de desenvolvimento web Django. O resultado deste trabalho produzirá uma aplicação web fácil de utilizar e compatível com vários navegadores, como Chrome, Edge e Firefox.

Com a finalidade de aumentar a produtividade das empresas, esta aplicação oferecerá uma interface intuitiva, que permitirá que os utilizadores realizem as suas tarefas de forma eficaz. Além disso, ela tem o objetivo de promover melhorias na qualidade do código em projetos de implementação Oracle Retail.

1.4 Estrutura do Documento

O presente documento encontra-se dividido em sete capítulos, os quais, Introdução, Estado da Arte, Análise de Requisitos, Arquitetura, Implementação, Testes e Conclusão. No primeiro capitulo apresenta-se uma breve contextualização teórica acerca do retalho e do Oracle Retail, a motivação para a criação do gerador de código e os principais objetivos. O segundo capitulo comporta uma descrição dos principais conceitos que compõem este trabalho, nomeadamente, as tecnologias, os geradores de código, os geradores de código com *templates*, expondo-se as vantagens/desvantagens e a aplicabilidade de cada um. O terceiro capitulo, especifica os requisitos de software, bem como os *use cases* que descrevem as interações do utilizador, o tipo de código que vai ser gerado e o modelo de dados da aplicação. O quarto capitulo descreve os principais componentes do sistema. O quinto capitulo reúne como foi implementada a aplicação desenvolvida. O sexto capitulo descreve os testes que foram realizados para verificar a funcionalidade e a qualidade da aplicação. Por fim, no último capitulo é feita uma conclusão que resume o que foi realizado e sugere possíveis melhorias ou extensões para trabalhos futuros.

Capítulo 2

Estado da arte

Este capitulo subdivide-se em três partes, na primeira aborda-se a tecnologia utilizada para o desenvolvimento da aplicação. Na segunda parte, tendo em conta que o âmbito deste projeto é a criação de um gerador de código, torna-se relevante a contextualização acerca dos geradores de códigos pelo que será abordado a sua função, vantagens e desvantagens e alguns exemplos da sua aplicação. Na terceira parte são descritas de uma forma detalhada as ferramentas que se assemelham ao gerador de código que se pretende criar com este projeto.

2.1 Tecnologias

2.1.1 Python

O Python [1] é uma linguagem de programação de alto nível que é amplamente utilizada para o desenvolvimento web, análise de dados e inteligência artificial. É uma linguagem *open-source* criada por Guido Van Rossum durante as férias de Natal de 1989 para resolver um problema do seu trabalho e como teve um feedback positivo dos seus colegas decidiu adicionar novas características e lançou pela primeira vez a linguagem em 1991. [6]

É conhecida pela sua sintaxe clara e legível, tornando fácil a aprendizagem para principiantes e para programadores experientes a leitura e manutenção do código. A língua tem um número relativamente pequeno de palavras-chave e uma estrutura simples. Também usa indentação para indicar blocos de código, em vez dos símbolos "{"e "}", facilitando a leitura do código.

Esta linguagem suporta múltiplos paradigmas de programação, incluindo programação orientada a objetos, orientada a procedimentos e funcional. O Python suporta várias plataforma, tais como o Windows e variantes de Unix incluindo Linux e macOS.

[1] https://www.python.org/

Tem uma comunidade ativa, que é responsável pela criação de várias bibliotecas e *frameworks*, tais como o Django para desenvolvimento web, o NumPy para computação numérica, e o TensorFlow para machine learning. Além disto, o Python também suporta a utilização de ambientes virtuais e gestores de pacotes como Pip [2], o que facilita a instalação e gestão de bibliotecas e módulos externos.

2.1.2 Django

O Django [3] é uma *framework open-source* escrita em Python, mantida pela Fundação Django Software e é amplamente utilizada para construir aplicações web em vários domínios. Atualmente algumas das aplicações que utilizam o Django são o Instagram, Pinterest, National Geographic e o Mozilla.

Foi inicialmente desenvolvido entre 2003 e 2005 por uma equipa de web development que trabalhava no desenvolvimento de websites para jornais. Ao longo do tempo, foram desenvolvidos vários websites idênticos e a equipa começou a reutilizar vários fragmentos de códigos e padrões de design comuns, permitindo que o código comum evolui-se para uma *framework* de desenvolvimento web, sendo *open-sourced* como o projeto "Django"em Julho de 2005.

Esta *framework* é considerada pelos desenvolvedores como **MVT**.

- **Modelo**: Define o esquema da base de dados e interage com a mesma para realizar operações CRUD (criar, ler, atualizar e apagar). Em Django, o modelo é definido usando classes Python, que são depois automaticamente traduzidas em tabelas de base de dados.

- **View**: Define a lógica que lida com pedidos e respostas HTTP. Em Django, as views são definidas como funções, que recebem um objeto *HttpRequest* como argumento e devolvem um objeto *HttpResponse*, além disso também podem ser definidos funções de view auxiliares. As *views* podem ser definidas para lidar com diferentes tipos de pedidos tais como GET, POST, PUT e DELETE.

- **Template**: Define a estrutura do HTML que é enviado ao cliente. No Django, os *templates* são definidos como ficheiros de texto que contém uma combinação de HTML simples e tags especiais que permitem conteúdo dinâmico [7]. Além disto, existe herança, permitindo aos programadores criar um *template* base que define a estrutura global e posteriormente herdar em outros *templates*. Isto permite que o projeto se mantenha consistente em certas partes, ao mesmo tempo que permite que as páginas individuais tenham elementos únicos.

[2] https://pypi.org/project/pip/
[3] https://www.djangoproject.com/

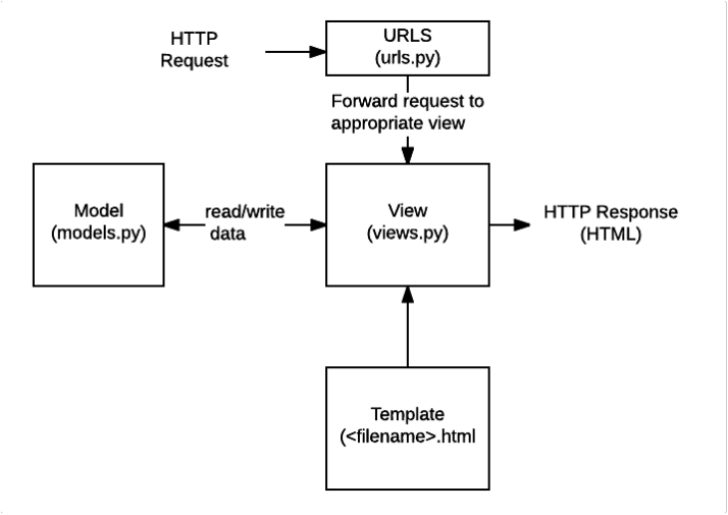


Figura 2: Padrão da arquitetura do Django retirado de [1]

O Django funciona da forma descrita na figura 2. Quando um utilizador faz um pedido HTTP a uma aplicação web Django, o pedido é tratado pelos URLS e é encaminhado para a view apropriada com base no padrão de URL. Quando esta executa, pode consultar a base de dados ou processar dados de formulários, e devolve uma resposta HTTP. Esta resposta pode ser uma página HTML, um objecto JSON, ou qualquer outro tipo de conteúdo que possa ser enviado através de HTTP.

Esta *framework* fornece uma variedade de outros componentes e características integradas que podem ser utilizados para acrescentar funcionalidades adicionais a um projeto, tais como:

- Uma interface de administração que permite a fácil gestão dos dados na base de dados.
- Autenticação e características de autorização que podem ser utilizadas para controlar o acesso a certas partes da aplicação.
- Tratamento de formulários, que podem ser utilizados para processar facilmente os dados submetidos pelos utilizadores
- Apoio ao envio e tratamento de correio eletrónico
- Vários *packages* que podem ser utilizadas para acrescentar funcionalidades adicionais à aplicação [8].

Vantagens e Desvantagens

Tal como todas as ferramentas, esta também possui as suas limitações e potencialidades as quais serão descritas abaixo:

Vantagens:

- Rápido desenvolvimento: O Django possui um amplo conjunto de características integradas que permitem o desenvolvimento célere de aplicações web. Uma das principais vantagens é o facto de oferecer um **ORM** que permite ao programador concentrar-se na interface do utilizador e menos no código *backend* que está instalado em *packages* [9].

- Segurança: O Django apresenta um sistema de segurança robusto que protege contra ataques de cross-site scripting (XSS), cross-site request forgery (CSRF), e SQL Injection.

- Escalabilidade: O Django foi concebido para dar resposta a uma elevada quantidade de dados e de tráfego, tornando-se adequado para aplicações de âmbito empresarial . Este *framework* alicerça-se na arquitetura "shared-nothing", permitindo que seja possível adicionar hardware para servers de base de dados, servers de caching ou servers de aplicação sendo facilmente escalável a nível de hardware [8].

- Documentação extensiva e comunidade ativa: O Django contém uma extensa documentação e possui uma comunidade ativa e muito extensa, tornando mais acessível a resolução de problemas e obtenção de recursos, sempre que é necessário.

Desvantagens:

- Curva de aprendizagem: O Django é composto por uma multiplicidade de características que poderão tornar complexa a aprendizagem, sobretudo para programadores iniciantes ou que tenham escassa experiência em Python e/ou desenvolvimento web.

2.2 Geradores de Código

2.2.1 Definição de Geradores de Código

Os geradores de código são ferramentas que geram automaticamente código para uma linguagem de programação específica com base no input fornecido pelo utilizador. São amplamente utilizados no desenvolvimento de software como web e *mobile development* para reduzir o tempo e esforço necessários para escrever código repetitivo e critico, bem como para assegurar que o código é consistente e segue as melhores práticas.

Estes geradores, como atesta Syriani [10], estão a ser adotados pelas organizações uma vez que reduzem o tempo do processo de desenvolvimento e aumentam a produtividade.

Atualmente, existe a possibilidade de uma extensa heterogeneidade de equipas intra e internacionais poderem trabalhar no mesmo projeto. Consequentemente, é expectável que existam diferenças na escrita de código que poderão ser ultrapassadas se forem utilizados geradores de código que tenham uma base igual para todos os membros da equipa.

A geração de código pode ser utilizada em várias fases do ciclo de vida do desenvolvimento de software, desde a conceção, implementação e testes. Com a crescente complexidade dos sistemas, a utilização de geradores de código tornou-se uma necessidade no desenvolvimento de software moderno. Desta forma, está a tornar-se um elemento chave para o desenvolvimento de software e é amplamente utilizado tanto na indústria como no meio académico.

Vantagens e Desvantagens

De seguida são descritas as vantagens e desvantagens identificadas nos geradores de código. Após analisar, é possível verificar que existem muitas mais vantagens que desvantagens, o que mostra que este tipo de software é muito benéfico para quem o usa.

Vantagens

- Produtividade: Com a geração de código escreve-se o gerador uma vez e este pode ser reutilizado tantas vezes quantas forem necessárias. Portanto, dar valores como input para o gerador e invocá-lo é significativamente mais rápido do que escrever o código manualmente várias vezes [5].
- Qualidade: Com o código gerado os erros são significativamente reduzidos [5].
- Consistência: O código gerado é concebido de acordo com os mesmos princípios e melhores praticas. O código gerado vai funcionar sempre da forma que se espera e que foi inicialmente programado.

Desvantagens

- Código customizado: Quando são utilizados geradores de código para código altamente customizado, se for efetuada uma geração de código sobreponível, pode-se incorrer no risco de danificar o código previamente existente. Portanto, o custo de construir *templates* customizados pode ser mais elevado do que escrever código manualmente.

2.2.2 Aplicabilidade Contemporânea

Um exemplo bastante conhecido e muito utilizado pelos desenvolvedores são as ferramentas de *scaffolding*, que geram a estrutura inicial de um projeto de software com base em parâmetros especificados pelo utilizador. Isto pode incluir a criação de hierarquias de ficheiros e diretorias, bem como a geração de código critico para tarefas comuns, tais como acesso a bases de dados e autenticação de utilizadores.

As ferramentas de *scaffolding* mais conhecidas são o *Express-generator* que é uma ferramenta para criar aplicações Node.js, conseguindo criar a estrutura básica das diretorias e os ficheiros de configuração. Além dista também existe a ferramenta React-create-app que cria aplicações React.js com uma simples configuração e um ambiente de desenvolvimento *frontend*.

Outro exemplo de gerador de código é o Swagger [4] que gera automaticamente uma documentação da API através da estrutura da API descrita na aplicação. Além disso também é possível adicionar testes automáticos às varias *routes* da API.

Existe também aplicações de gerar código para interface de utilizadores. Por exemplo, Glade [5] é uma ferramenta que permite aos utilizadores conceber interfaces gráficas de utilizador usando uma interface de arrastar e largar, e depois gera o código necessário para a interface de utilizador numa variedade de linguagens de programação.

No entanto, existe outro tipo de gerador de código que é baseado em *Templates* que será descrito no subcapítulo seguinte.

2.3 Gerador de código com templates

Este tipo de gerador de código baseia-se na utilização de *templates* que são similares a formulários. Os últimos referidos consistem em texto estático que incorpora algum texto dinâmico. No que se refere à parte estática, o texto é exatamente igual no ficheiro de output, em oposição, na parte dinâmica, o texto é alterado de acordo com o input do utilizador e esta geralmente é identificada por caracteres específicos, entre os quais, %, # ou $ [11].

Além destas características é importante referir que os *templates* supra-mencionados podem conter qualquer tipo de linguagem, tais como Java, Python ou SQL.

[4] https://swagger.io/
[5] https://glade.gnome.org/

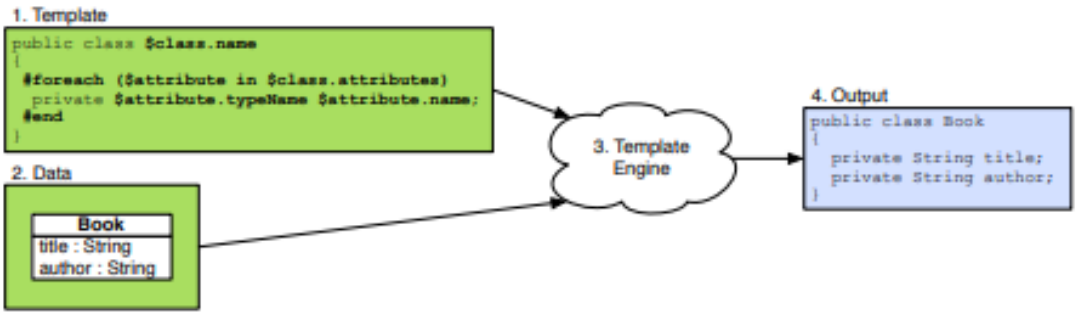


Figura 3: Funcionamento do *Template Engine*

De acordo com a Figura 3, verifica-se que o *Template Engine* recebe, em simultâneo, o input do *template* a ser utilizado e os dados que serão as variáveis que irão substituir o texto dinâmico. Finalmente, para que se possa gerar o ficheiro de output o *Template Engine* processa o input do utilizador e substitui as partes dinâmicas do *template* por texto estático como indica UYANIK [5].

Atualmente, já existem vários *Templates Engines*, nomeadamente, Jinja2, FreeMarker, Velocity, Xpand e Mustache. De salientar que, cada um tem a sua própria linguagem, ou seja, o freemarker utiliza o símbolo $ para distinguir as partes dinâmicas, o Mustache utiliza o # e o velocity utiliza ambos os símbolos.

```
<html>
  <body>
  #set( $foo = "Velocity" )
  Hello $foo World!
  </body>
</html>
```

Figura 4: Exemplo de um *template* Velocity

Na figura 4 apresenta-se um *template* que utiliza a **VTL**, sendo que o #set é usado para dar o valor Velocity à variável foo. À posteriori, essa variável é referenciada utilizando o símbolo $.

O resultado do processamento consiste numa página página WEB com "Hello Velocity World!".

No que se refere à aplicabilidade dos *Templates Engines*, hoje em dia, existem várias ferramentas que utilizam estes Engines como base, mais especificamente, o AndroMDA que utiliza Velocity e FreeMarker, o Fujaba que utiliza Velocity e antigas versões do Xtext que utiliza Xpand, tal como atesta [11] e ainda, o CodeSmith utiliza uma linguagem proprietária.

2.4 Aplicações Atuais

Existe no mercado soluções para geração de código a partir de *templates*. Os programas em analise são o Telosys e CodeSmith.

2.4.1 Telosys

O Telosys é uma ferramenta que utiliza *templates* para gerar código em qualquer linguagem de programação. Os *templates* são escritos numa sintaxe simples e fácil de aprender, e definem a estrutura e o conteúdo do código gerado. Vem com um conjunto de *templates* predefinidos para exemplos de código comuns, tais como classes, interfaces e camadas de acesso a bases de dados, mas o utilizador também pode criar os seus próprios *templates* para gerar código personalizado.

O Telosys utiliza o Velocity como *Template Engine* sendo possível criar *templates* em **VTL**.

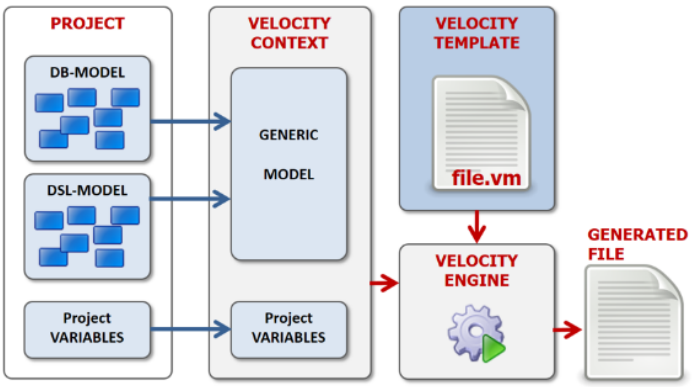


Figura 5: Arquitetura do Telosys

Como é possível ver na figura 5, o Telosys permite aos programadores definir um modelo, sendo este uma representação da estrutura de dados, utilizando uma DSL (Domain Specific Language) simples e fácil de utilizar ou ligando-se a uma base de dados. Este modelo é utilizado como fonte para a geração do código. Além disso também é possível usar variáveis para gerar o código.

Para além da sua principal funcionalidade, o Telosys inclui também uma série de outras características que facilitam a sua utilização e manutenção. Tem uma interface simples e de fácil utilização que lhe permite selecionar os *templates* que o utilizador pretende utilizar. Além disto utiliza o github para armazenar os *templates*, portanto tem também um sistema de controlo de versões que analisa as alterações aos *templates*, e permite retroceder para versões anteriores se necessário.

Utilização

Para utilizar o Telosys [6], o utilizador terá primeiro de o instalar no seu sistema. Isto pode ser feito normalmente ao descarregar o instalador apropriado a partir do website do Telosys e executa-lo.

Uma vez instalado, é necessário criar um projeto. Após isto é preciso iniciar o telosys no projeto e fazer as devidas configurações caso necessário.

Uma vez configurado o projeto, pode-se definir os *templates* que definem a estrutura e o conteúdo do código que se pretende gerar. Estes *templates* podem ser escritos numa linguagem de *template* que é específica da Telosys. Podem se usar os *templates* já criados previamente pelo software ou como já foi referenciado criar um customizado.

Como referenciado anteriormente, esta ferramenta permite ao utilizador criar Modelos e entidades que darão origem ao código. Por exemplo, é criado uma entidade "Car", com os atributos ID e Nome e através disso quando são selecionados os *templates* são criados todos os ficheiros para esta entidade e para os atributos definidos.

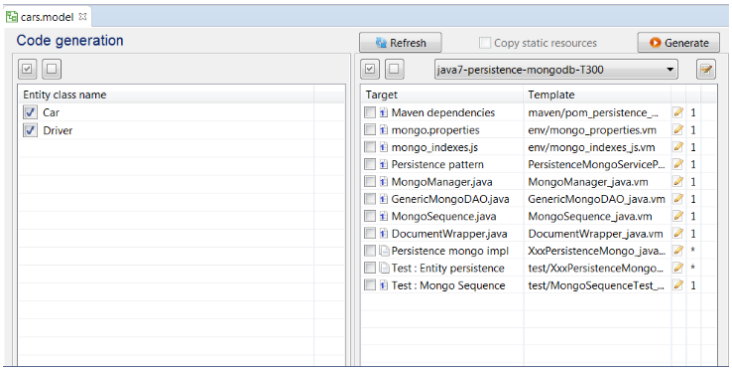


Figura 6: Ecrã para gerar código da ferramenta Telosys

Para gerar o código, como está na figura 6, terá de se clicar em *generate* e então o gerador irá então criar os ficheiros, que podem ser incorporados na aplicação.

2.4.2 CodeSmith

O CodeSmith [7] é uma ferramenta de geração de código comercial que utiliza uma linguagem de *templates* proprietária para definir a estrutura e o conteúdo do código gerado. Inclui uma série de *templates* pré-criados para tarefas comuns. Tal como o Telosys é possível criar manualmente *templates* com a linguagem do gerador de forma a ter código customizado. Também é possível editar os *templates* já existentes.

[6] https://www.telosys.org/
[7] https://www.codesmithtools.com/

Uma vez criado um ou mais *templates*, o utilizador pode utilizar o gerador para gerar código baseado nesses *templates*. Para tal, terá de fornecer dados de entrada aos modelos, quer especificando uma base de dados ou introduzindo manualmente os dados. O CodeSmith Generator processará então os *templates* e gerará o código, que pode ser guardado num ficheiro ou copiado para a área de transferência como apresentado na figura 7.

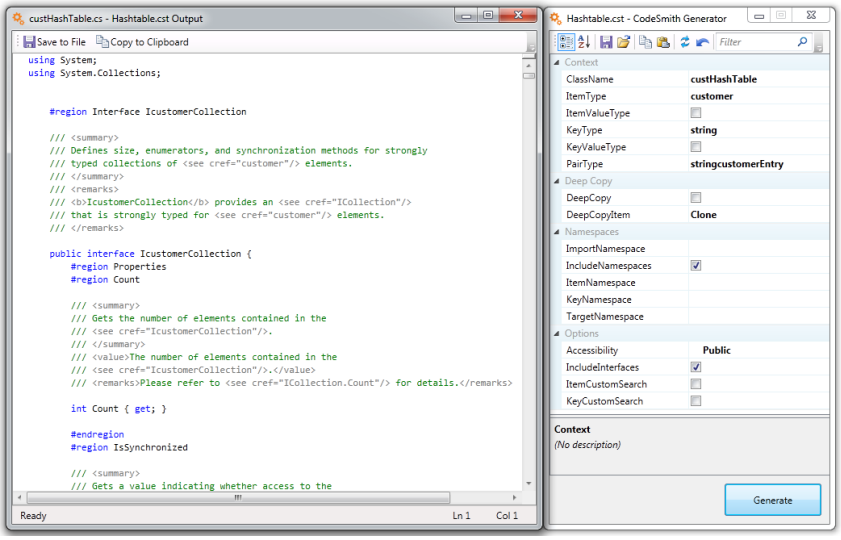


Figura 7: Ecrã com output do CodeSmith gerado

Utilização

Para utilizar o CodeSmith, é necessário primeiro instalá-lo no computador. Posteriormente, é necessário adicionar *templates*, tendo a opção de criar ou importar. Após ter os *templates* definidos é necessário configurar as propriedades do *template*, podendo introduzir uma conexão da base dados para gerar o código através da base de dados e/ou o nome das classes e métodos que o utilizador quer que seja gerado.

Após este processo, como é possível ver na Figura 7, o utilizador têm as propriedades definidas e quando é acionado o botão Generate é aberto o ecrã com o ficheiro de output gerado.

2.4.3 Comparação

O Telosys e o CodeSmith são ambas ferramentas de geração de código que são usadas para gerar código em múltiplas linguagens, tais como Java, C# e PHP, no entanto têm algumas diferenças fundamentais.

Enquanto que o Telosys é uma ferramenta open-source, com a capacidade de ser integrado no eclipse, o CodeSmith é uma ferramenta comercial que pode ser integrado com o Visual Studio. Além disso, devido

ao facto de o Telosys ser open-source, existe uma comunidade que contribui para o projeto ao contrário do CodeSmith. Por fim, o CodeSmith contem um editor de *templates* muito avançado com a funcionalidade de completar código durante a escrita enquanto que o Telosys é muito mais simples.

Parte II

Núcleo da Dissertação

Capítulo 3

Análise de Requisitos

A engenharia de requisitos é o processo de determinar o que um sistema deve fazer para satisfazer as necessidades do utilizador, concentrando-se na funcionalidade e não nos detalhes da implementação. Envolve o estudo do domínio do problema e o trabalho com as partes interessadas para identificar as funções que são pretendidas para o sistema [12].

No domínio do desenvolvimento de software, a análise de requisitos constitui uma fase fundamental que influencia significativamente o sucesso de um projeto, uma vez que é nela que são definidos os objetivos, as funcionalidades e as restrições do sistema de software são delineadas, segundo [13]. Em termos estruturais, este capítulo subdivide-se em seis subcapítulos: O que vai ser gerado pela aplicação, os Requisitos Funcionais, Requisitos Não-Funcionais, o modelo de dados e por fim a definição dos Templates de código. De salientar que, o capítulo Requisitos Funcionais encontra-se ainda subdividido em Requisitos do Utilizador e Requisitos do Sistema.

3.1 O que vai ser gerado?

De uma forma sucinta, este subcapítulo destina-se a descrever como e o que foi selecionado para ser gerado pelo gerador de código. Portanto, antes de enumerarmos os objetos que vão incorporar a implicação é importante explicar como é que foi o procedimento para selecionarmos os objetos mais importantes para um developer que trabalha com Oracle Retail.

Em primeiro lugar foi efetuada uma reflexão pessoal, identificando os objetos essenciais, tomando como ponto de partida, exemplos práticos inerentes à atividade em Oracle Retail. No entanto, como os recursos ainda eram parcos, foi efetuada uma pesquisa e recolhida informação através de entrevistas informais a *developers* experientes, os quais orientaram os métodos que deveriam ser aplicados, o que deveria ser incluído na aplicação e os atributos que cada objeto deveria ter, pelo que se depreende que os *developers* supracitados contribuíram significativamente para a construção da presente aplicação.

Os objetos escolhidos e os respetivos atributos encontram-se descritos na seguinte tabela 1.

Objeto	Atributos
Retek Errors	• Key • Description PT • Description EN • Description ES
System Parameters	• Parameter Id • Number Value • String Value • Comments
Code Head/Detail	• Name • Description
Table	• Name • Comments • Folder Description • Action
Package	• Name
Report	• Name • Description • Mod Id • Npo1 • Npo2 • Npo3
Form	• Name • Folder • Folder Description • Folder Parent • New Folder • Nullable
Wif	• Doc Type • Name
Proc	• Name • Description • Threads Number • Commit Max Ctrl • Has Prepost • Restart Table • Restart Columns • Package • Function Pre • Function Thread • Post • Process

Tabela 1: Objetos e respetivos atributos

3.1.1 Retek errors

Os retek errors são dados introduzidos na tabela retek errors, tendo como principal finalidade a atribuição de uma mensagem de erro a um determinado código, permitindo que essa mesma informação fique codificada e armazenada e que o código fonte nunca seja alterado. Este tipo de objetos são comummente utilizados tendo em conta que simplificam o código e, mais ainda, permitem alterar uma mensagem de erro que está repetida, através de um simples update dos dados da tabela para o respetivo código do erro.

Um exemplo deste objeto seria atribuir uma mensagem a uma key tais como "Erro de formatação da data"à key "DATA_ERROR"e utilizar esta key em vez da mensagem completa onde for necessário.

3.1.2 System Parameters

Os System Parameters funcionam da mesma forma que o objeto anterior e são dados que são introduzidos na tabela nb_system_parameters. A tabela supramencionada é utilizada para atribuir valores a chaves e/ou alterar os valores em vez de alterar o código fonte.

No domínio prático, poderíamos utilizar esta tabela para atribuir um email a uma chave "EMAIL"e, dessa forma, alterar apenas os dados da tabela para alterar o respetivo email.

3.1.3 Code Head/Detail

O Code Head/Detail são dados introduzidos em duas tabelas, a code_head e a code_detail. A code_head é responsável por conter o código e uma descrição enquanto que a code_detail é responsável por conter um ou mais valores referentes a esse código.

3.1.4 ProC

O ProC é um pré-compilador para a linguagem de programação C. Este é frequentemente utilizado em conjunto com sistemas de bases de dados Oracle e permite aos programadores incorporar instruções Structured Query Language (SQL) diretamente em código C. Esta componente é normalmente utilizada quando existe a necessidade de invocar um processo *schedule*, como por exemplo execução de um *batch* noturno.

3.1.5 Table

As tabelas são objetos que agrupam todos os dados de uma base de dados, permitindo a estratificação e organização dos mesmo no formato de linhas e colunas semelhante a uma folha de cálculo. Cada linha representa um registo único e cada coluna representa um campo no registo.

3.1.6 Package

Em PL/SQL, um package é um aglomerado de procedimentos, funções e variáveis, ou seja, representa uma forma de organizar e modularizar o código nas bases de dados Oracle. Cada package é dividido em duas partes principais, a interface/cabeçalho e o corpo. A primeira define os procedimentos, funções e variáveis que podem ser acedidos de fora do package e, na segunda, são implementados os procedimentos e funções definidos previamente.

3.1.7 Wif

Um WIF ou Web Import File é utilizado pela Oracle Retail para fazer carregamentos massivos de dados para a aplicação. Estes carregamentos são efetuados com base num ficheiro CSV, com uma estrutura de colunas definida e sendo efetuado o upload para a Base de Dados através do Oracle Retail. Estes dados, respeitando a estrutura definida, são lidos e carregados para tabelas de *staging* de forma a serem processados com a lógica definida pelo retalhista.

3.1.8 Report

Os relatórios no Oracle Retail são ferramentas essenciais para os retalhistas analisarem dados, monitorizarem o desempenho empresarial e tomarem decisões informadas. Existem vários tipos de relatórios, gerados através de ferramentas de reporting e podem ser personalizados para se adaptarem a necessidades dos retalhistas.

Estes reports são construídos utilizando a ferramenta Oracle Reports Builder, mas é necessário parametrizar o report na base de dados de forma a este ser gerado, como por exemplo gerir as permissões. Esta parametrização será o que vai ser gerado com esta aplicação.

3.1.9 Form

Os Forms são construídos utilizando o Oracle Forms Builder, sendo possível desenvolver interfaces para input e manipulação de dados. À semelhança dos Reports, também é necessário parametrizar o form na base de dados, sendo esta o que vai ser gerado na aplicação.

3.2 Requisitos Funcionais

3.2.1 Requisitos do utilizador

Os requisitos do utilizador constituem a base de qualquer projeto de desenvolvimento de software, uma vez que são derivados da compreensão das necessidades, expectativas e aspirações de um utilizador ou de outro tipo de parte interessada. Como indica, um bom requisito do utilizador provém de um utilizador ou de outro tipo de parte interessada e exprime uma propriedade do domínio ou do processo comercial que a introdução de um novo sistema irá proporcionar.

Requisito	**Descrição**
1	O ator deve ser capaz de criar, remover e alterar múltiplos projetos. Sendo cada projeto um agrupamento de diferentes objetos.
2	O ator deve ser capaz de adicionar, remover, alterar os diferentes tipos de objetos dentro de cada projeto.
3	O ator deve ser capaz de aceder a cada projeto e seus objetos através de um menu lateral
4	O ator deve ser capaz de gerar os ficheiros correspondentes a cada objeto.
5	O ator deve ser capaz de ver erros a quando do preenchimento do formulário.

Tabela 2: Requisitos do Utilizador

3.2.2 Requisitos do sistema

Findada a definição dos requisitos do utilizador, o próximo passo reside na definição das especificações técnicas e funcionais que o sistema de software deve cumprir. Os requisitos do sistema englobam os aspetos de arquitetura, conceção e integração da solução, abordando a forma como o software irá funcionar e interagir com os vários componentes do ambiente.

Requisito	Descrição
1	O Sistema deve armazenar múltiplos projetos para um utilizador.
2	O Sistema deve armazenar múltiplos objetos para cada projeto.
3	O Sistema deve gerar os objetos para uma pasta.
4	O Sistema deve apresentar os projetos e objetos no Menu.

Tabela 3: Requisitos do Sistema

3.3 Requisitos Não Funcionais

Para além das necessidades funcionais do software, existe outra dimensão que determina o desempenho e a qualidade, nomeadamente, os requisitos não funcionais. Estes requisitos são mais abrangentes e centram-se em atributos como o desempenho, a segurança, a escalabilidade, a fiabilidade e a facilidade de utilização, ou seja, não se centram somente em funcionalidades específicas.

Requisito	Descrição
1	A aplicação deve ter tempos de resposta céleres e os atrasos no carregamento da página deverão ser mínimos, mesmo com um volumoso número de utilizadores.
2	O processo de geração de código deve ser otimizado para garantir uma execução eficiente e minimizar o tempo de processamento.
3	A interface do utilizador deve ser intuitiva e acessível em termos de utilização, com instruções e mensagens de erro claras e inteligíveis.
4	A modularidade e a reutilização do código devem facilitar a adição de novas funcionalidades.

Tabela 4: Requisitos Não Funcionais

3.4 Use Cases

Segundo (Wiegers e Beatty, 2013) [13] um "Use Case"descreve uma sequência de interações entre um sistema e um interveniente externo que resulta na possibilidade de o ator alcançar um resultado. Um ator é uma pessoa, outro sistema de software ou, até mesmo, um dispositivo de hardware que interage com o sistema para executar um caso de utilização.

De seguida, são descritos os uses cases que constituem a aplicação.

ID	UC01
Use Case	Criar um novo projeto
Atores	Utilizador
Descrição	Ação para um Utilizador criar um novo projeto.
Pré-condições	- O utilizador tem de ter iniciado sessão para criar um novo projeto.
Fluxo Básico	1. O User seleciona a opção "Criar um Novo Projeto". 2. O sistema apresente um formulário para capturar os detalhes do novo projeto. 3. O User preenche o formulário com os atributos do Projeto. 4. O sistema adiciona o novo projeto e atualiza a lista de projetos.
Pós-condições	- Um novo projeto é criado com sucesso no software. - O novo projeto é adicionado à lista de projetos.

Tabela 5: Use Case para criação de projeto

ID	UC02
Use Case	Remover um projeto
Atores	Utilizador
Descrição	Ação para um Utilizador remover um novo projeto.
Pré-condições	- O utilizador tem de ter iniciado sessão para remover um novo projeto.
Fluxo Básico	1. O User seleciona a opção de "Remover"no menu lateral. 2. O Sistema apresenta um ecrã de confirmação. 3. O User seleciona a opção de remover. 4. O Sistema remove o projeto, todos os objetos associados e atualiza a lista de projetos.
Pós-condições	- O projeto é removido com sucesso no software. - O projeto é removido da lista de projetos.

Tabela 6: Use Case para remover um projeto

ID	UC03
Use Case	Alterar um projeto
Atores	Utilizador
Descrição	Ações para um Utilizador remover um projeto.
Pré-condições	- O utilizador tem de ter iniciado sessão para remover um novo projeto.
Fluxo Básico	1. O User seleciona a opção de "Alterar"no menu lateral. 2. O Sistema apresente um formulário com os dados do projetos preenchidos. 3. O User faz as alterações que pretende e submete. 4. O Sistema altera o projeto.
Pós-condições	- O projeto é alterado com sucesso no software.

Tabela 7: Use Case para alterar um projeto

ID	UC04
Use Case	Adicionar um novo Objeto a um Projeto
Atores	Utilizador
Descrição	Ações para um User adiciona um novo objeto a um projeto existente.
Pré-condições	- O utilizador tem de ter iniciado sessão como Utilizador
Fluxo Básico	1. O utilizador seleciona o Projeto onde pretende adicionar o objeto. 2. O utilizador seleciona a opção "Adicionar"associada ao objeto 3. O sistema apresente um formulário para capturar os detalhes do novo objeto 4. O utilizador preenche e submete o formulário 5. O sistema adiciona o novo objeto e atualiza a lista de objetos.
Pós-condições	- O novo objeto é criado com sucesso. - O novo objeto é adicionado à lista de objetos do projeto selecionado.

Tabela 8: Use Case para dicionar um novo Objeto a um Projeto

ID	UC05
Use Case	Remover um novo Objeto de um Projeto
Atores	Utilizador
Descrição	Ações para um User remover um novo objeto de um projeto existente.
Pré-condições	- O utilizador tem de ter iniciado sessão como Utilizador
Fluxo Básico	1. O utilizador seleciona o Projeto onde pretende remover o objeto. 2. O utilizador seleciona a opção "Remover"associada ao objeto. 3. O sistema apresenta um ecrã de confirmação. 4. O utilizador seleciona a opção de Remover. 5. O sistema remove o objeto e atualiza a lista de objetos.
Pós-condições	- O objeto é removido com sucesso. - O objeto é removido da lista de objetos do projeto selecionado.

Tabela 9: Use Case para remover um novo Objeto a um Projeto

ID	UC06
Use Case	Alterar um Objeto de um Projeto
Atores	Utilizador
Descrição	Ações para um User alterar um objeto de um projeto existente.
Pré-condições	- O utilizador tem de ter iniciado sessão como Utilizador
Fluxo Básico	1. O utilizador seleciona o Projeto onde pretende remover o objeto. 2. O utilizador seleciona a opção "Alterar"associada ao objeto. 3. O Sistema apresente um formulário com os dados do projetos preenchidos. 4. O Utilizador faz a edição e submete. 5. O sistema altera o objeto.
Pós-condições	- O objeto é alterado com sucesso.

Tabela 10: Use Case para alterar um Objeto de um Projeto

ID	UC07
Use Case	Alterar um Objeto de um Projeto
Atores	Utilizador
Descrição	Ações para um User alterar um objeto de um projeto existente.
Pré-condições	- O utilizador tem de ter iniciado sessão como Utilizador
Fluxo Básico	1. O utilizador seleciona o Projeto onde pretende remover o objeto. 2. O utilizador seleciona a opção "Alterar"associada ao objeto. 3. O Sistema apresente um formulário com os dados do projetos preenchidos. 4. O Utilizador faz a edição e submete. 5. O sistema altera o objeto.
Pós-condições	- O objeto é alterado com sucesso.

Tabela 11: Use Case para alterar um Objeto de um Projeto

ID	UC08
Use Case	Gerar Ficheiros correspondentes a um Objeto
Atores	Utilizador
Descrição	Descreve as etapas envolvidas quando um membro do projeto gera arquivos ou documentos correspondentes a um objeto específico.
Pré-condições	- O utilizador tem de ter iniciado sessão como Utilizador.
Fluxo Básico	1. O utilizador seleciona o Projeto e o objeto que pretende. 2. O utilizador seleciona a opção "Gerar"associada ao objeto 3. O sistema gera os ficheiros numa localização determinada. 4. O sistema apresenta uma mensagem de confirmação.
Pós-condições	- Os ficheiros gerados são guardados numa determinada localização.

Tabela 12: Use Case para gerar Ficheiros correspondentes a um Objeto

3.5 Modelo de Dados

Neste subcapítulo é abordado o modo como o modelo de dados foi desenvolvido. Este encontra-se subdividido em três subdomínios, os quais, Utilizador, Objetos e Auxiliares, por forma a facilitar o entendimento do modelo.

No que respeita ao utilizador, o Django dispõe de um sistema de autenticação integrado, que é automaticamente gerido por uma *framework*, sendo as várias tabelas criadas no momento da primeira migração.

No respeitante aos objetos, cada objeto existe numa tabela com os respetivos atributos definidos anteriormente, sendo estes associados a um projeto através de um *Foreign Key*. De ressalvar que, cada projeto pode ter vários objetos associados representando uma ligação de 1 para N e que estes podem ser acedidos pelos utilizadores, numa escala de 1 utilizador para N projetos. Em oposição, cada objeto só pode ser correlacionado a um único projeto indissociavelmente.

Para terminar, foram criadas tabelas auxiliares com o intuito de complementar a informação dos objetos. Cada objeto *Table* é composto por uma tabela com os atributos principais e por uma tabela denominada *Column* com uma ou várias colunas, sendo que a informação presente em cada uma destas componentes se interrelacionam através de uma *Foreign Key*, tendo assim uma *Table*, N *Column*.

Foi desenvolvido na imagem 8 uma simplificação do diagrama de Entidade-Relação com todas as entidades e as respetivas relações, exceto as tabelas de autenticação criadas pelo *Django*.

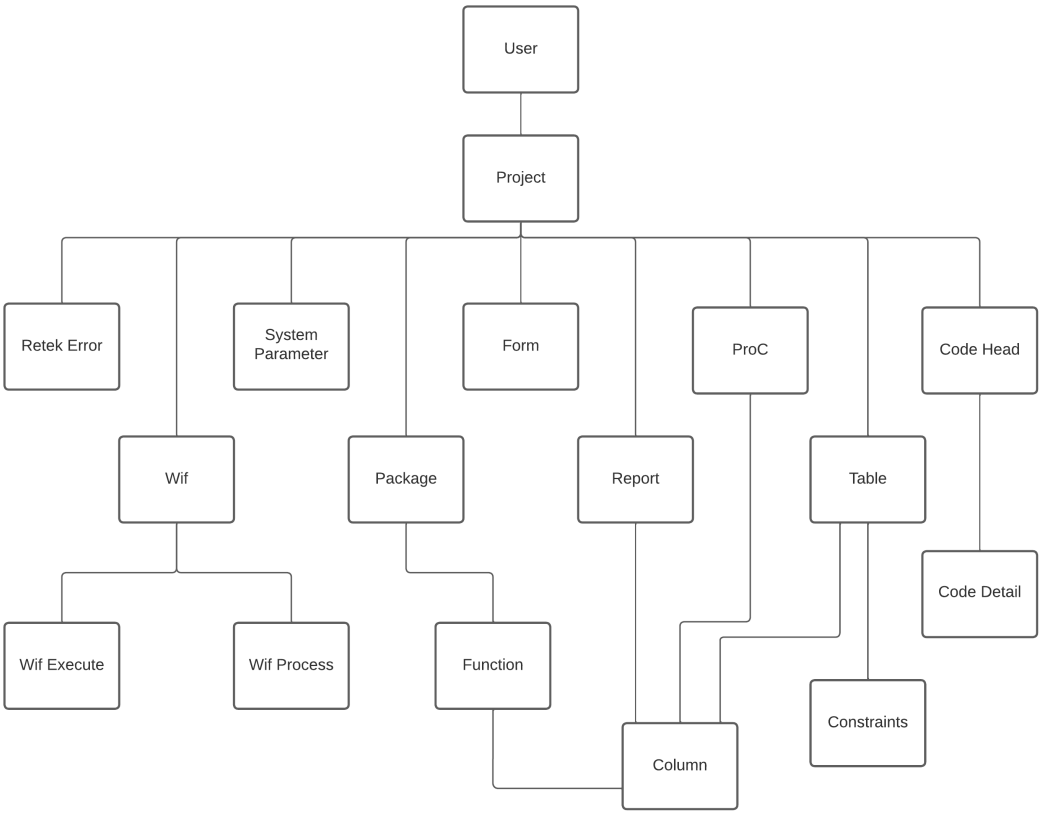


Figura 8: Diagrama de Entidade-Relação

3.6 Definição dos Templates

Após o estabelecimento de requisitos e modelos de dados, torna-se imperativo delinear os *templates* que serão utilizados para cada objeto. Para defini-los, foi efetuado um levantamento das diversas metodologias para a sua construção, sendo o código desenvolvido baseado nas melhores práticas. O capítulo 5 enumera os vários *templates*, acompanhados do código gerado para cada um deles.

3.6.1 Regras para Templates

De seguida, são enunciadas as melhores práticas de código que foram tidas em conta nos templates criados que se refletem posteriormente nos ficheiros gerados para cada Objeto.

Padrões de código

Uma padrão de código é um conjunto de regras e diretrizes que definem como escrever código de forma consistente. A utilização de uma norma de código pode ajudar a melhorar a legibilidade, a capacidade

de manutenção e a qualidade do código como atesta Feuerstein [14].

- Usar indentação e espaço em branco consistentes;
- Adicionar comentários para explicar código complexo;
- Evitar o uso de nomes enigmáticos de variáveis e funções;
- Seguir convenções de nomenclatura consistentes para diferentes tipos de variáveis.

Modularidade e encapsulamento

A modularidade refere-se à prática de dividir tarefas complexas em módulos mais pequenos e reutilizáveis. O encapsulamento refere-se à prática de esconder os pormenores de implementação de um módulo de outros módulos [15].

Alguns pontos tidos em conta nos templates, segundo [14], incluem:

- Usar pacotes para agrupar procedimentos e funções relacionados
- Usar variáveis e procedimentos privados para esconder detalhes de implementação
- Exportar apenas os procedimentos e funções públicas necessários de um pacote

Tratamento de erros

Como afirma [14], um tratamento de erros robusto e consistente é um elemento importante para dois públicos muito diferentes: o utilizador e o programador. Se o utilizador receber informações fáceis de compreender e bem formatadas quando ocorre um erro, poderá comunicar esse erro de forma mais eficaz à equipa de apoio. Além disto, se a aplicação tratar e registar os erros da mesma forma em toda a aplicação, o suporte poderá corrigir o problema mais eficazmente.

Para isto foram incluídos nos templates os seguintes pontos:

- Usar o bloco EXCEPTION para tratar erros;
- Registo de erros numa tabela da base de dados;
- Retornar mensagens de erro informativas para o utilizador.

Capítulo 4

Arquitetura

Um dos principais desafios que se apresenta é desenvolver algo intuitivo, de caráter simples e eficaz, na qual os utilizadores se sintam confortáveis, daí ter optado pela versão mais recente do Django 4.2.3 e pela utilização do painel de administração, **ORM** e respetiva autenticação, o que tornou possível não só dar resposta às necessidades supramencionadas, bem como rentabilizar o tempo no desenvolvimento de outras funcionalidades primordiais da aplicação.

Além disto, outras tónicas na qual a dissertação se debruçou foram o código a ser gerado e a forma como o ecrã seria apresentado para uma melhor usabilidade. Dito isto, é possível depreender que o objetivo da aplicação é algo mais orientado para a funcionalidade e aplicabilidade e não tanto com um frontend interativo, algo que pode ser atingido usando a framework React.

Outra das opções metodológicas foi, inicialmente, a utilização da base de dados SQLITE para uma implementação mais rápida visto que naquele momento não seria necessário outro tipo de base de dados para o desenvolvimento. No entanto, próximo da finalização do projeto foi decidido fazer uma migração para uma base de dados POSTGRES.

Quanto à arquitetura da aplicação, foi utilizada uma arquitetura cliente-servidor com recurso ao Django.

A arquitetura cliente-servidor caracteriza-se por uma clara separação de responsabilidades entre duas entidades principais: o cliente e o servidor. O cliente é a interface através da qual os utilizadores interagem com a aplicação, enquanto o servidor é responsável pelo processamento dos pedidos, pela gestão dos dados e pela orquestração das funcionalidades principais da aplicação, de uma forma geral, como atesta Oluwatosin [16] a arquitetura cliente-servidor pode ser definido como uma arquitetura de software composta por um cliente e um servidor, em que os clientes enviam sempre pedidos enquanto o servidor responde a estes pedidos enviados.

O Django serve como um conjunto de ferramentas para construir os componentes do cliente e do servidor desta aplicação.

No que se refere ao cliente, o motor de criação de modelos e as capacidades de front-end do Django permitem a criação de uma interface intuitiva e fácil de utilizar. Esta interface funciona como porta de entrada para os utilizadores introduzirem os seus pedidos que, posteriormente, são transmitidos ao servidor para processamento.

No que respeita ao servidor da aplicação, este é responsável pelo tratamento dos pedidos recebidos, o que envolve várias tarefas tais como, a autenticação de utilizadores e a recolha ou atualização de dados na base de dados. Em termos práticos, as robustas funcionalidades de back-end do Django facilitam estas operações, garantindo que a aplicação responde prontamente e com precisão às ações do utilizador.

Na figura 9 apresenta-se uma figura ilustrativa da arquitetura da aplicação descrita anteriormente.

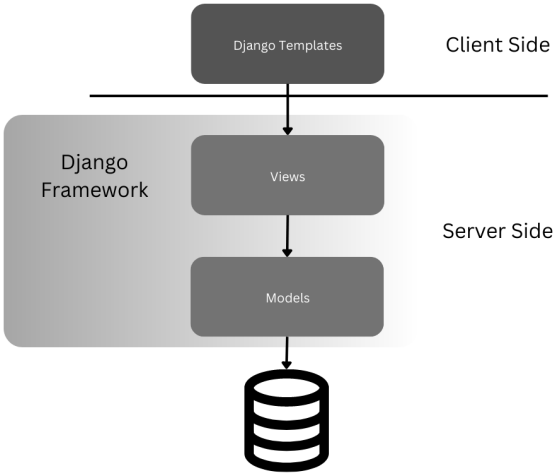


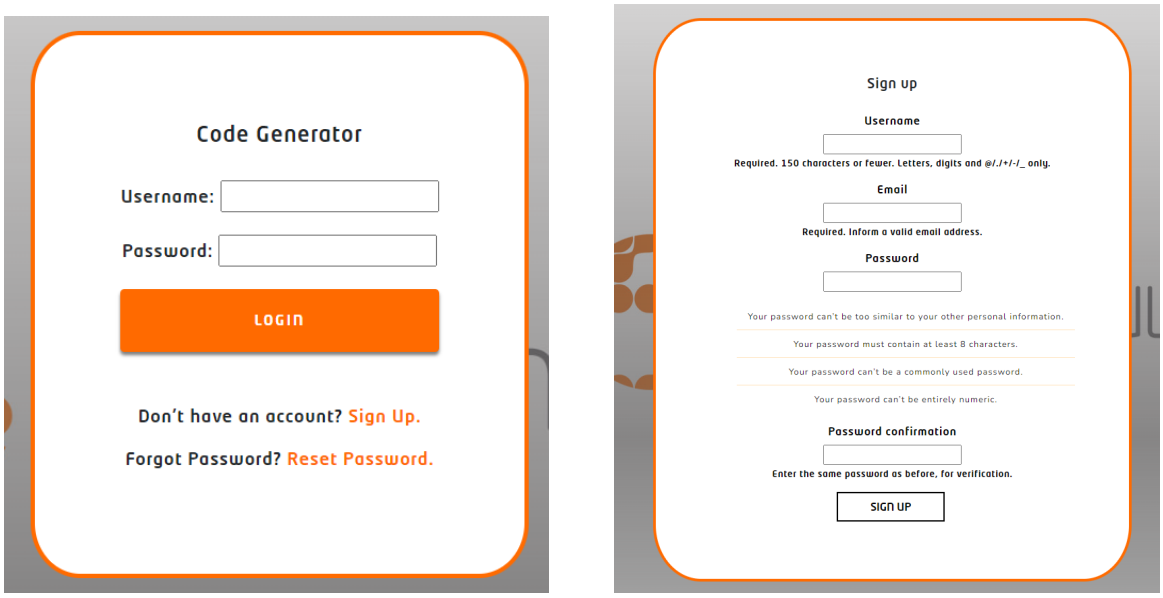
Figura 9: Arquitetura

Capítulo 5

Implementação

5.1 Autenticação

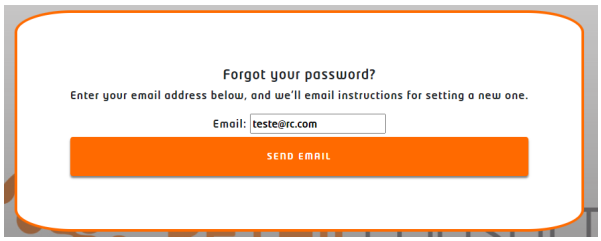
Como dito anteriormente, a parte de autenticação é da responsabilidade do Django, no entanto foi necessário desenvolver as templates com código HTML e CSS para o Login e Registo ficarem com o aspeto igual ao da aplicação. Portanto, foram desenvolvidos os templates - Login, SignUp - como demonstrado na imagem 10.



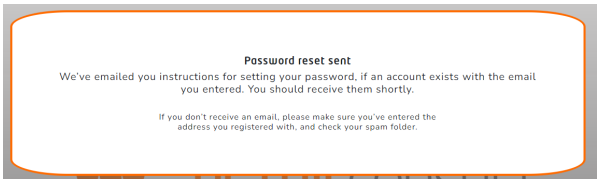
(a) Login (b) Registo

Figura 10: Autenticação

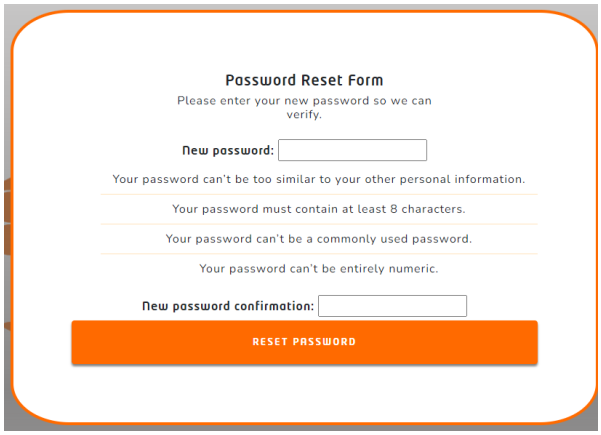
Além disto foi desenvolvido um fluxo de recuperar a password. Nas imagens 11 é possível ver como este funciona. Em primeiro lugar é inserido o email da conta em que pretende recuperar a password, de seguida é enviado um email com um link que dá acesso ao Formulário para inserir a nova password. Este link utiliza um Token único que não permite fazer múltiplas alterações da Password.



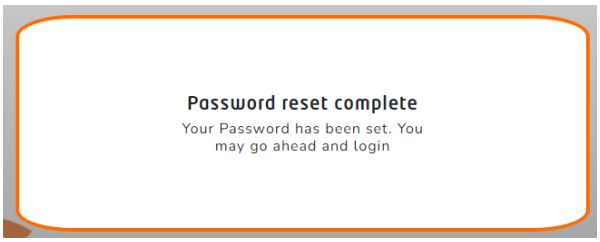
(a) Formulário de recuperação da Password



(b) Aviso de Email de recuperação enviado



(c) Formulário para preencher com a nova Password



(d) Aviso do sucesso da recuperação de Password

Figura 11: Fluxo de recuperação de Password

5.2 Menu

O "Side Tree Menu"foi concebido para fornecer uma representação hierárquica dos projetos do utilizador e dos objetos associados. Este é apresentado numa estrutura em forma de árvore, com os projetos a servirem de nós principais e os objetos a servirem de nós secundários. Este design promove uma experiência de utilizador intuitiva, permitindo que os utilizadores localizem e naveguem rapidamente pelos seus dados.

Os utilizadores podem expandir ou recolher projetos, revelando ou ocultando os objetos associados, tendo assim a possibilidade de clicar num objeto para efetuar as operações que pretende.

Para implementação deste menu, foi utilizado "inclusion tags"do Django. Estas são uma forma de criar componentes reutilizáveis que podem ser incluídos e renderizados dentro de templates. Para implementação do menu de árvore lateral, foi usado uma "inclusion tag"para criar um componente reutilizável que gera a estrutura HTML do menu enquanto obtém os dados mais atualizados da base de dados.

O uso de "inclusion tags"na implementação do menu da árvore lateral exemplifica a flexibilidade e eficiência do Django na construção de componentes reutilizáveis. Ao encapsular a recolha de dados da base de dados e a lógica de renderização dentro da tag, foi criada uma abordagem modular para integrar o menu em vários templates.

A figura 12 mostra o Menu quando está preenchido com os vários objetos.

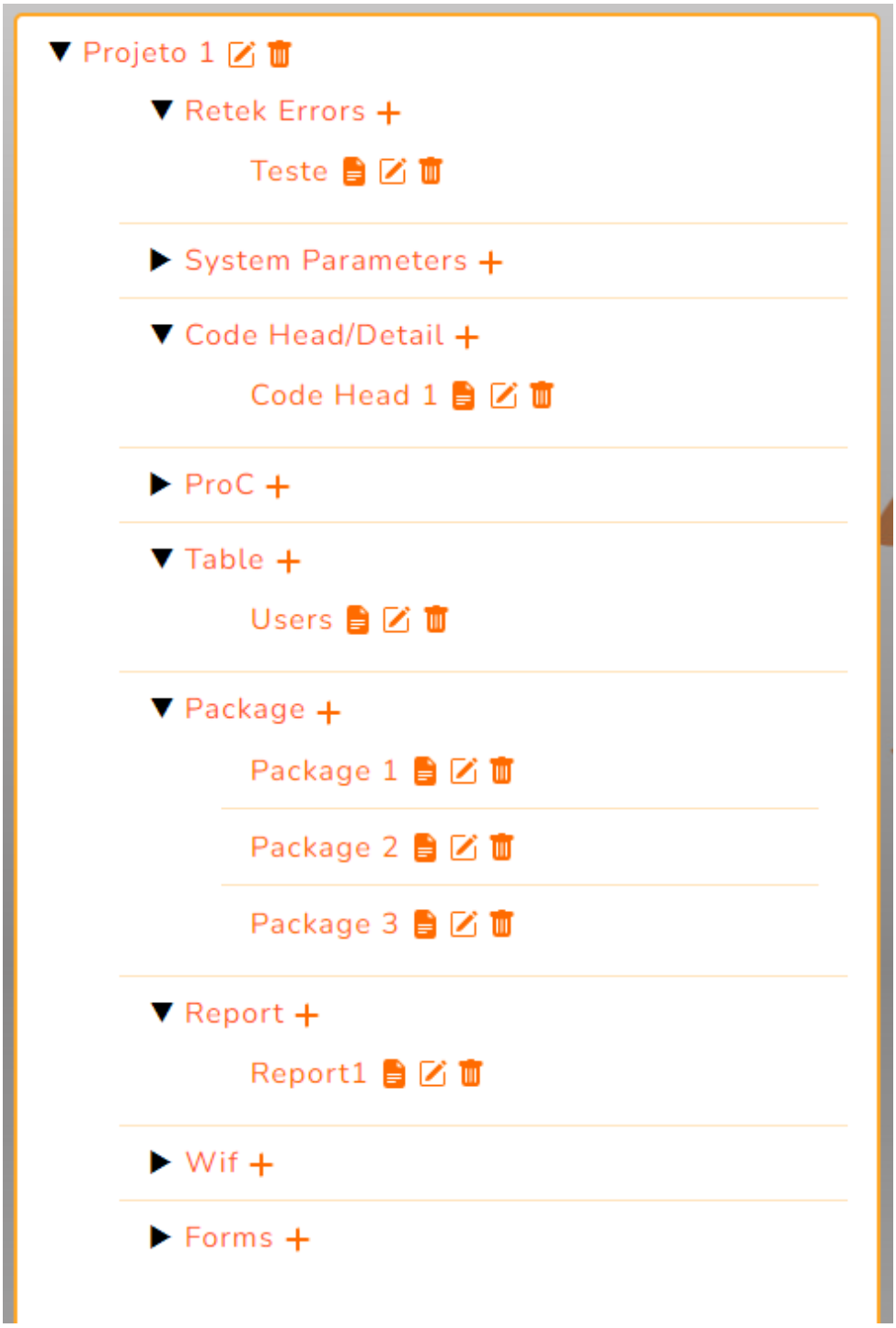


Figura 12: Menu em árvore

No entanto, também existe a possibilidade de não existir projetos associados ao utilizador. Portanto também foi desenvolvido no Menu um estado vazio como está na figura 13.

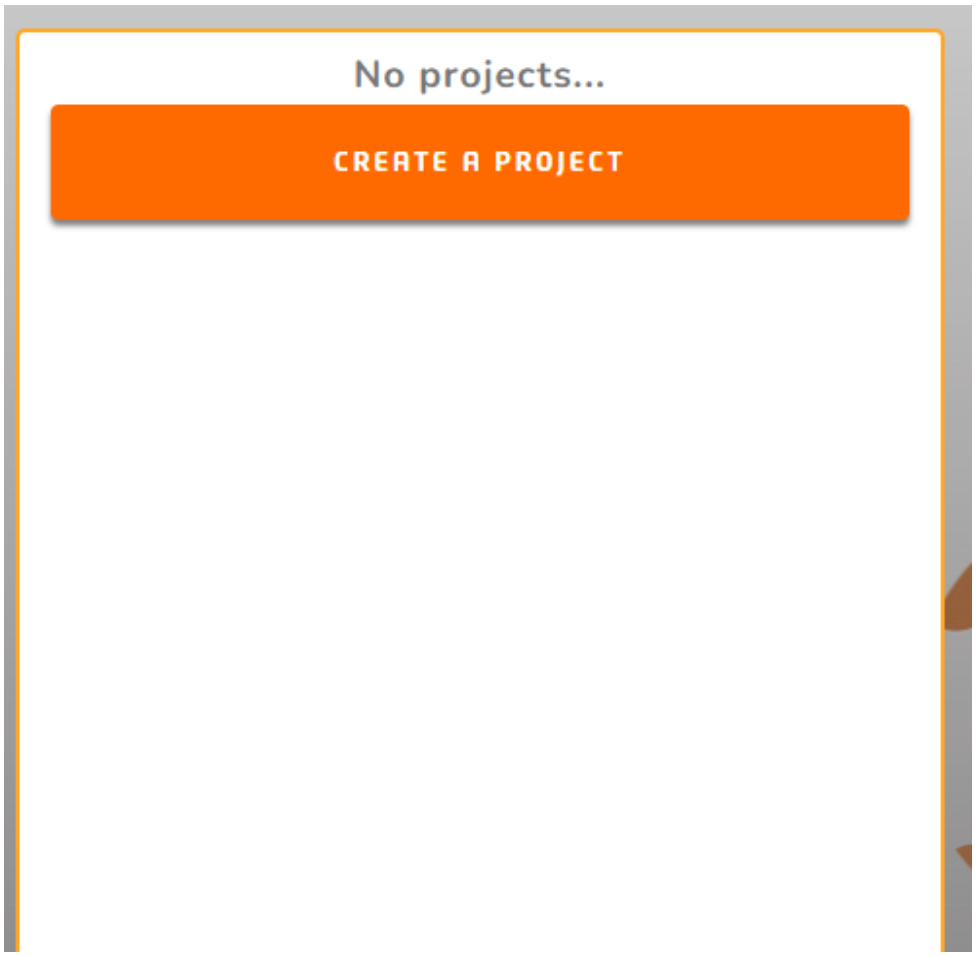


Figura 13: Menu em árvore com o estado Vazio

5.3 Implementação da aplicação

Esta secção tem o intuito de explicar como a aplicação foi desenvolvida.

Portanto, esta aplicação foi subdividida em duas aplicações secundárias: codegenerator e code_generator. A aplicação codegenerator cuida da autenticação, enquanto que a aplicação code_generator cuida de tudo o resto depois do utilizador efetuar a autenticação.

Aplicação de Autenticação:

Nesta aplicação, é tratada a autenticação do utilizador. Portanto, é onde o registo do utilizador, o início de sessão e a reposição da palavra-passe é efetuado. Como tal, o principal objetivo é garantir que os utilizadores podem iniciar sessão de forma segura para que posteriormente possam aceder às partes protegidas da aplicação.

Aplicação com Lógica da aplicação:

Esta aplicação é dedicada à implementação da funcionalidade principal quando os utilizadores já

estão autenticados. Aqui são definidas as Views, os Models e os Templates com a interface, ou seja, a lógica é toda implementada para ser feita a criação, atualização e remoção dos objetos, como também é feita a apresentação da interface e é onde é feito o tratamento das interações dos utilizadores.

Com isto é possível manter o código mais organizado dividindo a lógica da aplicação e a autenticação em partes separadas. Esta técnica mantém duas funcionalidades importantes separadas o que ajuda a criar uma aplicação Django mais escalável e modular.

A aplicação foi montada para gerar código para oito objetos distintos, cada um dos quais é representado por um ficheiro de view dedicado, sendo cada um composto por quatro funções essenciais: criar, atualizar, remover e gerar código.

Desta forma, o código fica modular devido a cada objeto estar no seu próprio ficheiro de visualização, o que torna o código mais fácil de compreender e manter. Além disto, facilita a adição de novos objetos à aplicação sem afetar o código existente e também a adição de novas funcionalidades.

- Criar: Esta função é responsável pela criação do objeto. Capta os dados introduzidos pelo utilizador e armazena-os na base de dados.
- Atualizar: A função de atualização permite aos utilizadores modificar os objeto. Recupera os dados do objeto, apresenta-os para edição e atualiza a base de dados após as alterações do utilizador.
- Remover: Esta função é dedicada à remoção do objeto. Fornece um mecanismo para os utilizadores eliminarem, garantindo a integridade e a gestão dos dados.
- Geração de código: Esta função é a principal caraterística da aplicação, sendo responsável por gerar o código relacionado com o objeto específico. O código é produzido com base nos atributos do objeto sendo estes fornecidos pelo utilizador.

Esta aplicação inclui templates HTML que servem de interface de utilizador para os ecrãs "Criar", "Atualizar"e "Eliminar". Tal como as views, cada objeto tem o seu próprio template de criação e atualização com o código HTML, o que facilita a personalização do aspeto da aplicação para cada objeto, visto que cada um é diferente. No entanto, o template de remoção é partilhado por todos os objetos de forma a promover a modularidade.

5.4 Formulários

Nesta secção, é discutida a implementação detalhada dos formulários utilizados para cada objeto. Esta funcionalidade foi feita recorrendo aos formulários do Django que são utilizados para recolher dados do utilizador. Eles são definidos usando classes do Python que herdam da classe forms.ModelForm, ou seja são utilizados os modelos definidos com os atributos do objeto para definir se os campos são de texto, números ou datas [8].

De forma a simplificar esta secção, os formulários foram divididos em duas categorias - Formulários Simples e Formulários com detalhe.

5.4.1 Formulário Simples

Este tipo de Formulário são denominados Simples devido a terem apenas os inputs para o modelo de dados previamente construido. Isto é, apenas contem a *label* e respetivo input.

Para a implementação foi necessário criar formulários de acordo com o modelo de base de dados. Por exemplo, para os Retek Erros, foi criado com os campos "Key", "Description_PT", "Description_EN", "Description_ES"e "Project". De seguida, este foi inicializado na view correspondente e renderizado pelo Template. Portanto, o resultado final foi ecrã apresentado na figura 14.

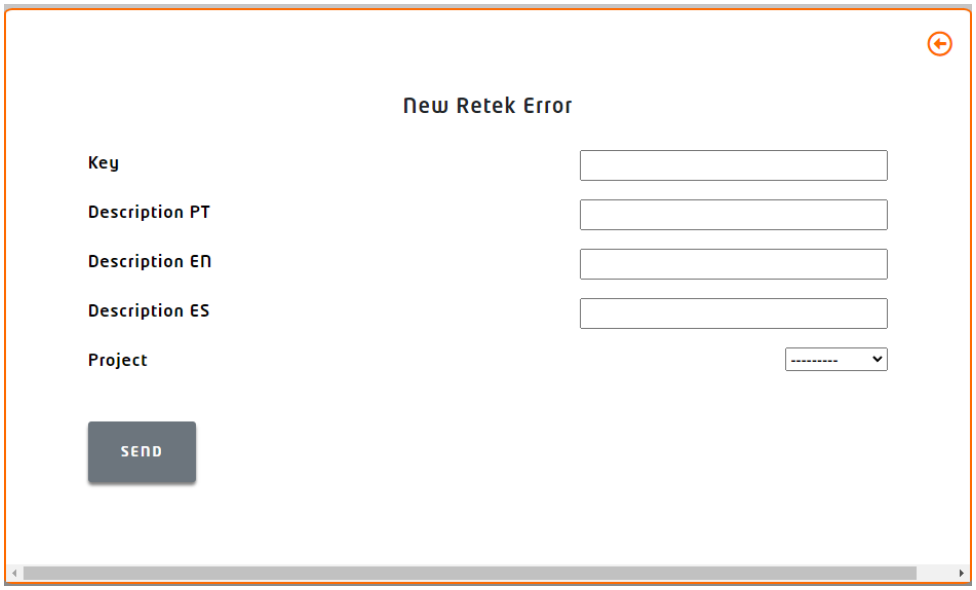


Figura 14: Formulário para o Objeto Retek Errors

Este processo foi realizado para o "System Parameters"e "Form"como é possível ver na figura 15 e figura 16, respetivamente.

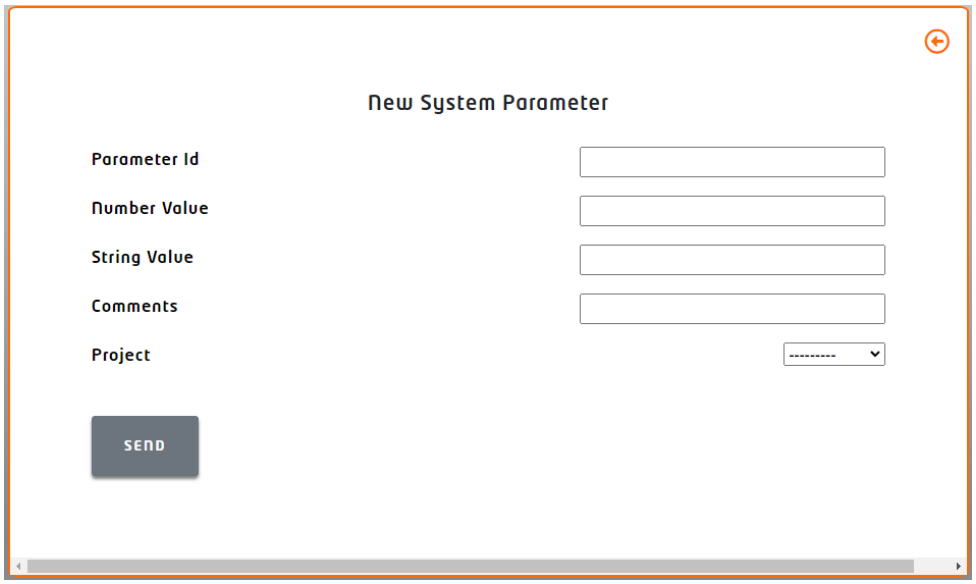


Figura 15: Formulário para o Objeto System Parameters

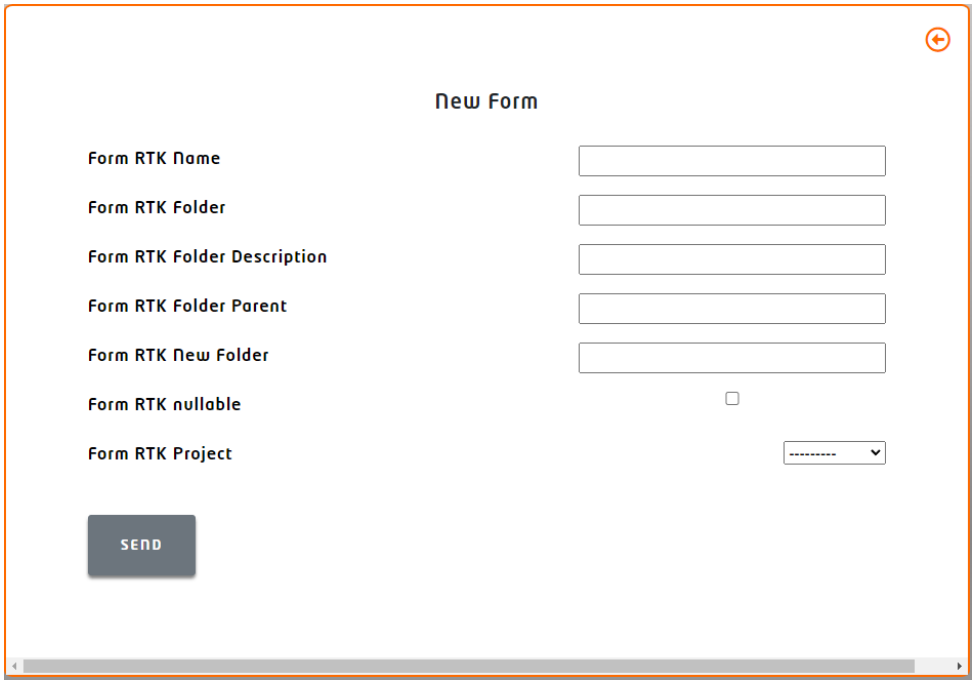


Figura 16: Formulário para o Objeto Form

5.4.2 Formulário com Detalhe

Por outro lado existem os formulários com Detalhe. Estes são constituídos pelos Simples e por uma nova funcionalidade que permite o utilizador inserir um número de linhas que necessite. Este Formulário funciona da seguinte forma: Para adicionar uma nova linha é selecionado o botão de + e para remover a

mesma é clicado no botão de "Remove Last".

Este tipo de formulários foi aplicado para os objetos "Table", "Code Head", "ProC", "Package", "Report"e "Wif".

Table

Para a Table foi utilizado um formset com Colunas para definição do número de colunas que a Table irá conter. Na imagem 17 foi utilizado um simples exemplo de uma tabela "Users", sendo definida com uma Coluna "Nome"e outra coluna "Número". Estas colunas podem ser definidas consoante o que o utilizador precisa, por exemplo, pode definir a Coluna Número com o tipo de dados "VARCHAR2".

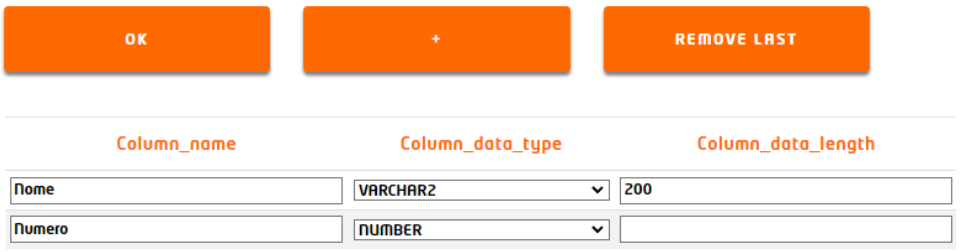


Figura 17: Formulário dinâmico com as colunas da tabela

Além disso também foi adicionado um formset Constraint para definir as constraints que a table vai ter. Por exemplo, foi utilizado o exemplo anterior e foi adicionado uma constraint Primary Key para a coluna "Nome"como está na figura 18. Desta forma o ficheiro gerado vai colocar automaticamente a coluna como Primary Key.

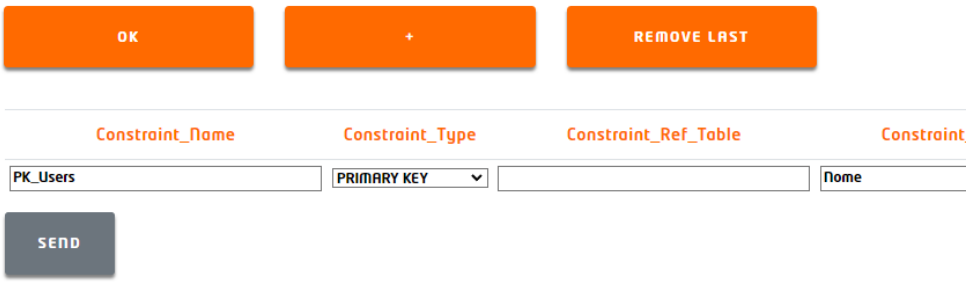


Figura 18: Formulário dinâmico com as Constraints da Tabela

Portanto, para efetuar a criação de uma Table é necessário esta componente dinâmica para o utilizador introduzir o número de Colunas e Constraints necessárias.

A figura 19 apresenta o formulário com todas as componentes descritas anteriormente.

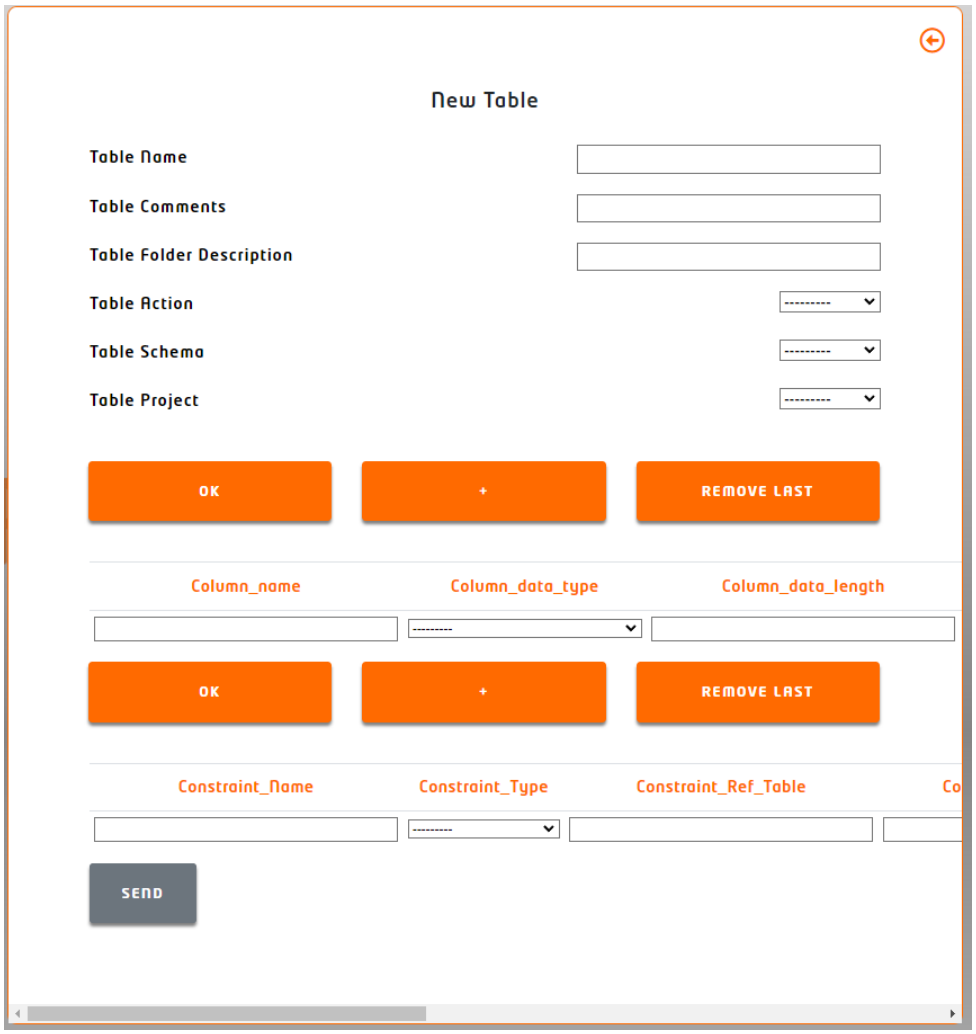


Figura 19: Formulário completo do Objeto Table

Code Head

Para este objeto foi utilizado um formset com os detalhes, sendo estes constituídos por um atributo "Value"e um atributo "Descrição". Estes valores podem ser adicionados consoante o desejo do utilizador, por exemplo, foi adicionado os valores "1","2","3","4"com respetivas descrição, como exemplificado na figura 20.

Figura 20: Formulário para o Objeto Code Head/Detail

Portanto, a Code Head é representado pelos atributos iniciais e a Code Detail corresponde aos valores do detalhe que podem ser 1 ou N.

Package

O Package é constituído por um conjunto de funções, e como tal é necessário definir estas funções que farão parte dele e por consequência é necessário saber os argumentos de cada função. Portanto, foi feito um sistema para adicionar um número qualquer de funções e associar a essas funções um número qualquer de argumentos.

Para isto ser possível, como demonstrado na figura 21, foi utilizado o modelo "Functions"para definir os dados das funções que o utilizador pretende adicionar e foi, também, utilizado o modelo "Columns"para definir os dados que cada argumento de que cada função vai ter.

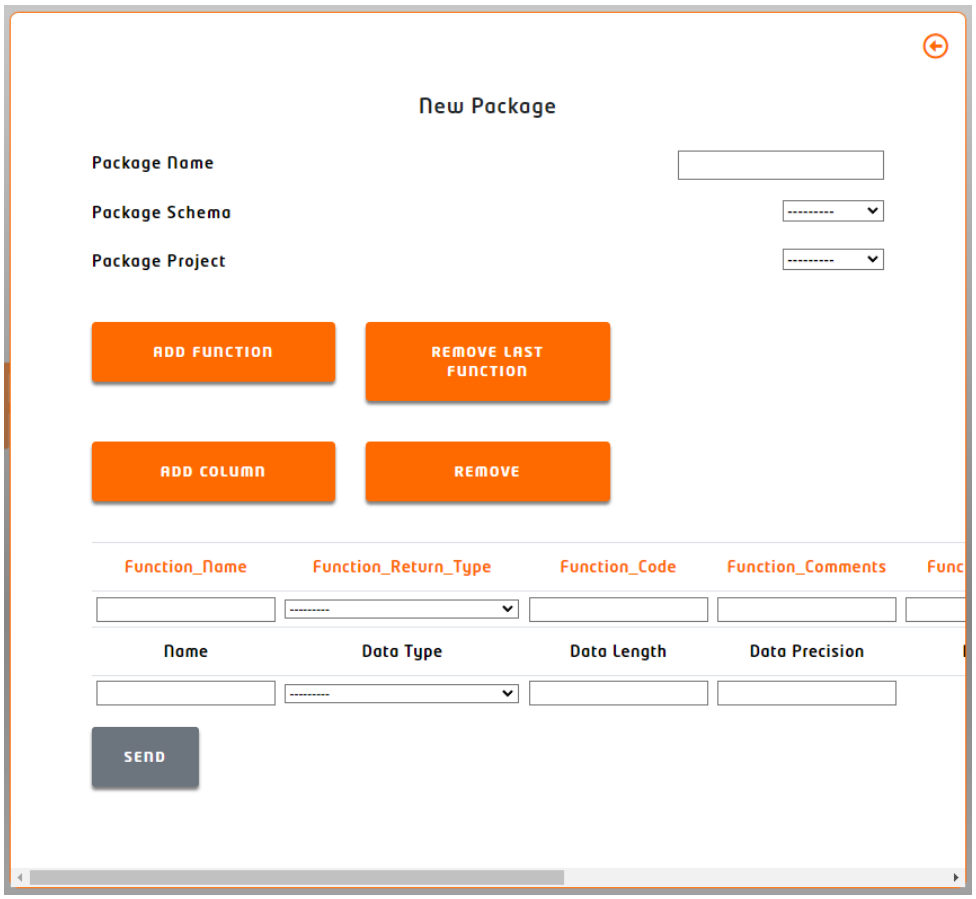


Figura 21: Formulário para o Objeto Package

Como o número de funções e respetivos argumentos não têm limite, o Utilizador pode adicionar qualquer número de funções ao selecionar a opção "ADD FUNCTION"e remover a ultima função criada ao selecionar "REMOVE LAST FUNCTION", sendo também possível selecionar o butão "+"numa qualquer função para um novo argumento e -"para remover. É de notar que o número mínimo de funções e respetivos argumentos é sempre um, ou seja, o Package vai sempre conter no mínimo uma função. Esta funcionalidade está visível na figura 22 com duas funções criadas e múltiplos argumentos para cada.

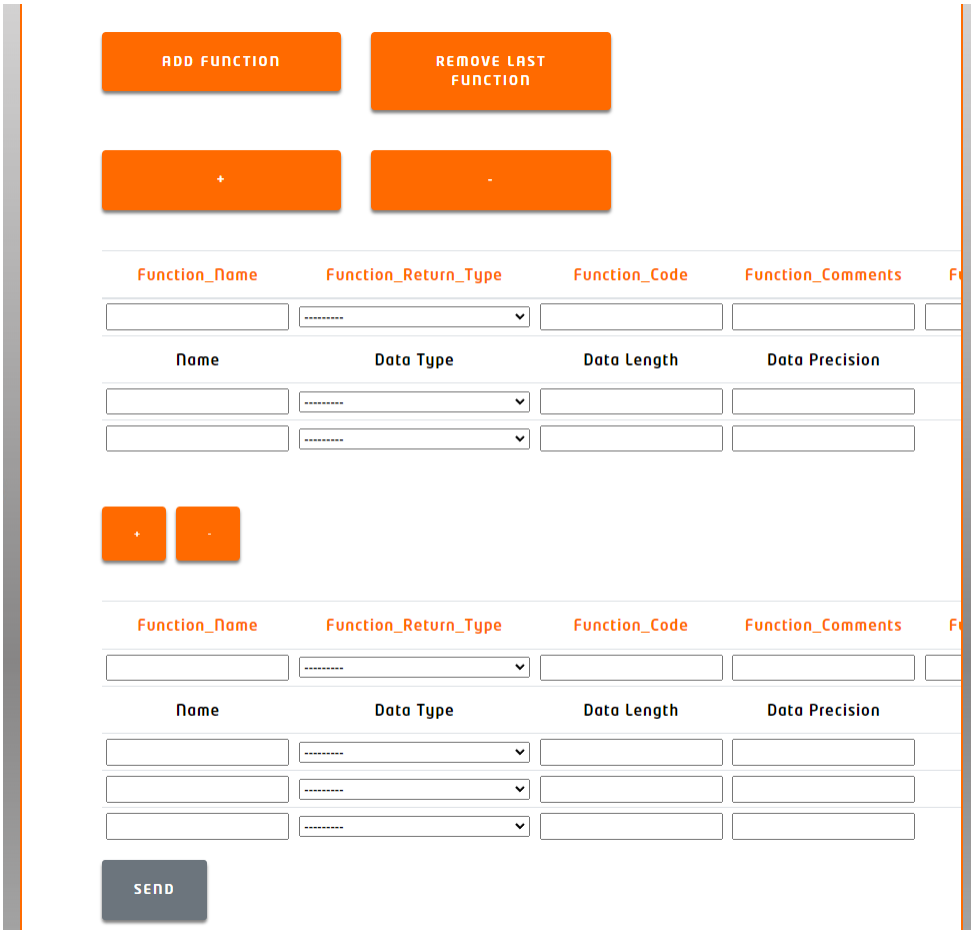


Figura 22: Formulário para o Objeto Package

ProC

Os *ProC's* são dos objetos mais complexos, sendo necessário valores para a criação de um Package, um ficheiro ProC e também uma tabela com as respetivos inserções. Para isso é necessário preencher os vários atributos presentes no formulário como o nome do ProC, o nome da tabela e do Package como também o número de colunas que a tabela irá conter, dai a utilização do formset "Column".

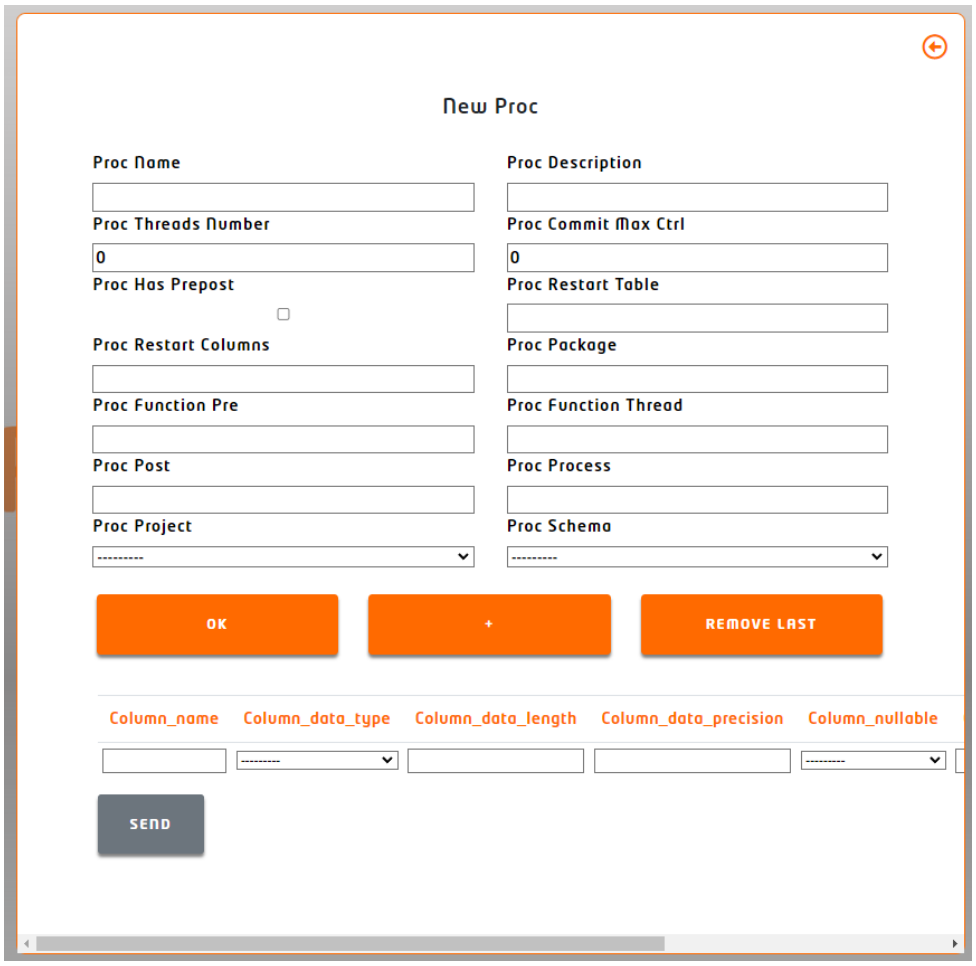


Figura 23: Formulário para o Objeto ProC

Como visto na figura 23, é necessário também dar um nome às funções do Package que são a de Pré, Thread, Post e Process que são abordadas no subcapítulo seguinte. Além disto, também é necessário definir o número de *threads* caso seja pretendido fazer um processamento *multithreading*.

Report

O objeto Report para serem gerados necessita de obter o nome e uns dados específicos relacionados com permissões dos utilizador, sendo estes os parâmetros NPO.

Além disto também é necessário parametrizar o Report com o tipo de parâmetros de input que este vai receber no momento que é gerado, como por exemplo, uma data. Para tal foi desenvolvido o sistema para permitir ao utilizador introduzir o número de parâmetros que necessite. Foi novamente utilizado o modelo "Columns"para obter estes dados.

A figura 24 apresenta a interface deste formulário como descrito no parágrafo anterior.

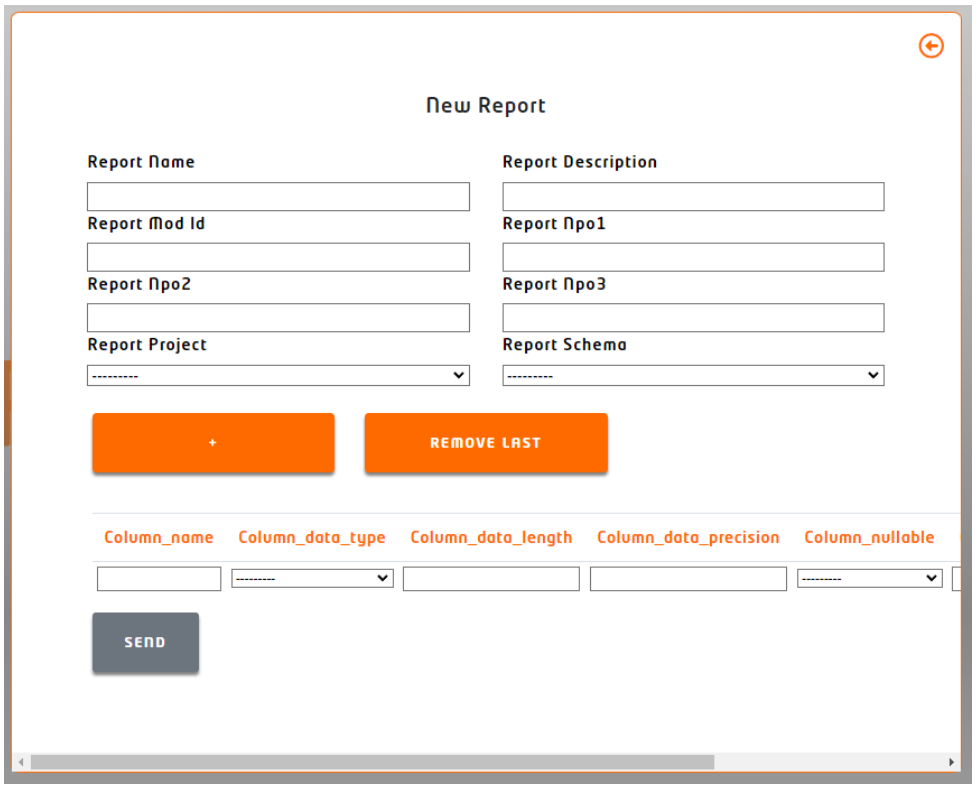


Figura 24: Formulário para o Objeto Report

Wif

Os Wifs precisam de obter o tipo de documento de onde onde dados vão ser carregados para além de receberem o seu nome. Para complementar o Wif, é necessário receber um ou mais argumentos que correspondem a função de Purga e um ou mais argumento para a função de Execute, sendo estas duas funções explicadas no seguinte subcapítulo. Estas duas componentes estão apresentadas na figura 25.

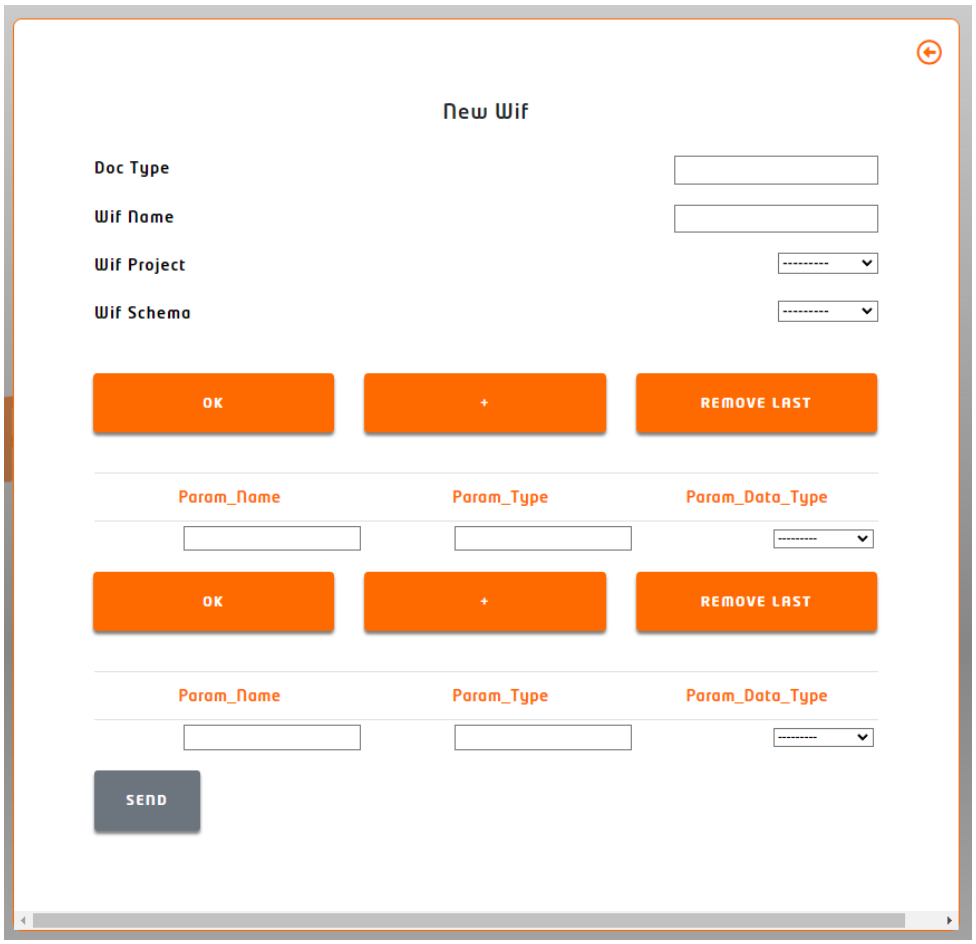


Figura 25: Formulário para o Objeto Wif

5.5 Geração de código

Nesta secção vai ser abordado como foi realizada a geração de código e o que foi gerado.

Em primeiro lugar, durante a fase de requisitos foram definidas *templates* para cada objeto, sendo estes *templates* implementados no código da aplicação. Contudo, durante o desenvolvimento foi pensado em definir ficheiros externos com os *templates*, sendo este posteriormente lidos e processados pela aplicação, dando assim vantagens a nível da manutenção dos *templates*. No entanto, apesar da vantagem, a primeira opção foi utilizada devido a ser necessário utilizar lógica dentro dos mesmos, como por exemplo *loops* e *if statments*.

É de sublinhar que todos os *templates* foram construídos sobre as regras e melhores práticas para garantir a máxima qualidade de código. Por exemplo, é muito comum que um programador se esqueça de identificar a alteração que efetuou num código com milhares de linhas, sendo isto prejudicial para a resolução de problemas caso exista um erro futuro devido a não ser possível identificar quem realizou a

ação. Como tal, em todos os *templates* este texto está incorporado como apresentado na figura 26.

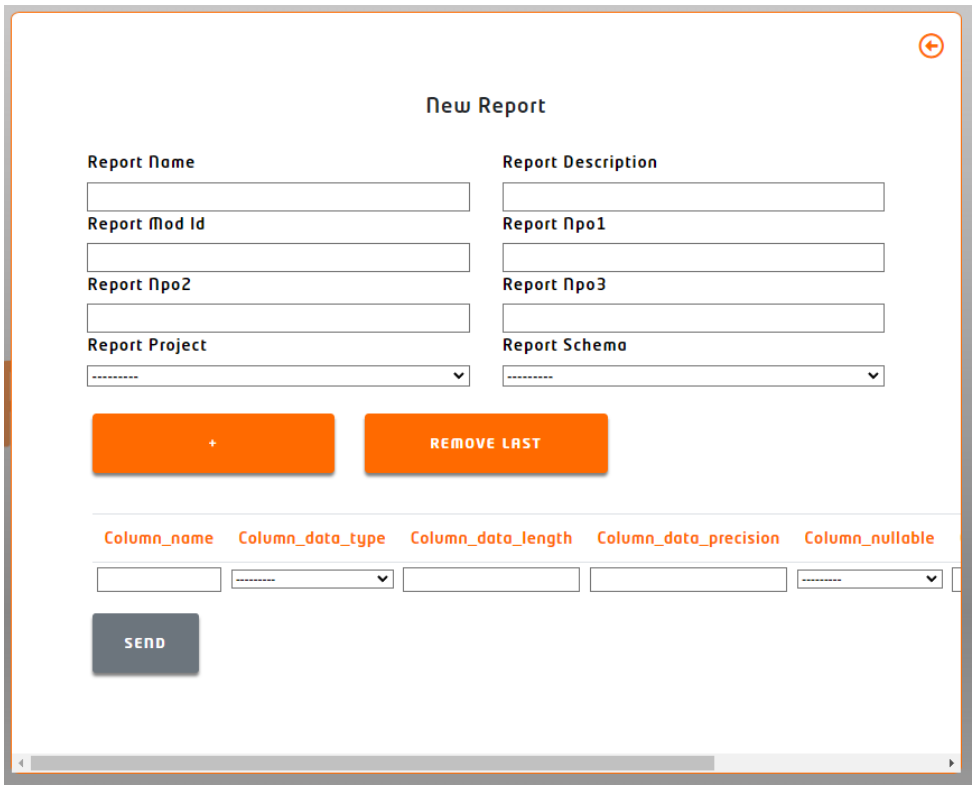


Figura 26: Cabeçalho com informação da alteração

5.5.1 Retek Errors

Este tipo de objeto é bastante simples, sendo apenas necessário gerar um ficheiro .sql. Este ficheiro contêm três inserts à tabela RTK_ERRORS, sendo cada insert referente a uma das três linguagens (PT, EN, ES). Além disso também contem uma instrução de DELETE de forma a evitar erros na compilação do código gerado.

O código na figura 27 corresponde a um dos inserts do script, sendo este referente à linguagem PT, devido à coluna rtk_lang estar a ser preenchida com o valor 1.

```
INSERT INTO rtk_errors
   (rtk_key
   ,rtk_lang
   ,rtk_text
   ,rtk_user
   ,rtk_approved)
VALUES
   (l_rtk_key
   ,1
   ,l_rtk_text_pt
   ,l_rtk_user
   ,'Y');
```

Figura 27: Instrução de inserção.

Como é possível ver na figura 28, as colunas rtk_key, rtk_text e rtk_user estão atribuídas a variáveis. Esta atribuição é realizada no inicio do script como está no seguinte código.

```
l_rtk_key      rtk_errors.rtk_key%TYPE  := 'Teste';
l_rtk_user     rtk_errors.rtk_user%TYPE := 'RETAILCONSULT';
l_rtk_text_pt  rtk_errors.rtk_text%TYPE := 'TESTE PT';
l_rtk_text_eng rtk_errors.rtk_text%TYPE := 'TEST EN';
l_rtk_text_es  rtk_errors.rtk_text%TYPE := 'TEST ES';
```

Figura 28: Declaração de variáveis.

Portanto, o template foi construido desta forma para o utilizador poder fazer alterações mais rapidamente e para manter um formato modelo para todos estes inserts.

5.5.2 System Parameters

À semelhança do objeto anterior, os Retek Errors, este objeto comporta-se da mesma forma. Portanto, apenas é criado um ficheiro .sql com um *insert* relativo ao system Parameter como está na figura29.

```
INSERT INTO nb_system_parameters
   (id
   ,valuen
   ,valuea
   ,comments)
VALUES
   (l_id
   ,l_valuen
   ,l_valuea
   ,l_comments);
```

Figura 29: Instrução de inserção.

Novamente os valores estão atribuídos no declare como demonstrado no bloco de código da figura 30:

```
l_id       nb_system_parameters.id%TYPE       := 'Parameter1';
l_comments nb_system_parameters.comments%TYPE := 'Novo system parameter';
l_valuen   nb_system_parameters.valuen%TYPE   := '1';
l_valuea   nb_system_parameters.valuea%TYPE   := 'Teste';
```

Figura 30: Declaração de variáveis.

5.5.3 Code Head/Detail

Para este objeto é novamente gerado apenas um ficheiro ".sql", no entanto, este divide-se em duas partes. A primeira parte consiste em fazer um INSERT na tabela code_head, sendo esta a tabela principal. De seguida, a segunda parte faz vários inserts à tabela code detail com referencia á primeira tabela.

Como é um procedimento base, em primeiro lugar as variáveis são declaradas como apresentado na figura 31, sendo posteriormente utilizadas no script gerado.

```
TYPE t_codes IS TABLE OF code_detail.code%TYPE;
TYPE t_codes_desc IS TABLE OF code_detail.code_desc%TYPE;

l_new_code_type  VARCHAR2(1)                  := 'Y';
l_max_seq        NUMBER                       := 0;
l_code_type      code_head.code_type%TYPE     := 'Teste';
l_code_type_desc code_head.code_type_desc%TYPE := 'Teste';
tcodes           t_codes                      := t_codes( '1','2','3','4');
tcodes_desc      t_codes_desc                 := t_codes_desc( 'UM','DOIS','TRES','QUATRO');
```

Figura 31: Declaração de variáveis e tipos.

São utilizados dois types, sendo o primeiro t_codes responsável por guardar o código de cada detalhe e o segundo por guardar a descrição.

A instrução da figura 32 simplesmente insere o novo code_head na tabela.

```
INSERT INTO code_head
    (code_type
    ,code_type_desc)
VALUES
    (l_code_type
    ,l_code_type_desc);
```

Figura 32: Inserção dos dados relativos à code head.

Por ultimo, na figura 33 é feito um loop que itera pelos types criados e insere na tabela code_detail.

```
INSERT INTO code_detail
    (code_type
    ,code
    ,code_desc
    ,required_ind
    ,code_seq)
VALUES
    (l_code_type
    ,tcodes(i)
    ,tcodes_desc(i)
    ,'Y'
    ,l_max_seq);
```

Figura 33: Inserção dos dados relativos à code detail.

5.5.4 ProC

Nesta componente, e sempre que necessário efetuar este desenvolvimento, este parte do mesmo principio que é a necessidade de invocar um processo *schedule*. Para tal e como definido na maioria dos clientes isto pressupõe a existência de uma tabela de controlo, um processo Pré, um processo de balanceamento de carga (Thread), o processo com a lógica de negócio, e um processo Post, não sendo todos obrigatórios. Este código em C, muitas vezes difícil de codificar para quem se encontra mais familiarizado com PL/SQL usualmente é o mesmo, sendo que o que muda é o cursor driver sobre a tabela de controlo bem como o package invocado para processamento, como indicado na figura 34. Com base nisto foi criado um template dinâmico que com base no preenchimento das respetivas funções, deteta a necessidade de ter, por exemplo, a fase de POST ou não.

A figura 34 representa uma pequena parte do código do ficheiro ProC que foi gerado pela aplicação. Este excerto de código é responsável por fazer a chamada à base de dados através da função de processamento, cujo nome e package que a encapsula é definido pelo utilizador.

```
int process_ProcessFunction()
{
   /* local variables */
   char*  function= 'process_ProcessFunction';

   /* Get info */
   EXEC SQL EXECUTE
      DECLARE
         O_ERROR_MESSAGE VARCHAR2(255);
      BEGIN
         :pi_plsql_error := 0;

         if NOT ProcPackage.ProcessFunction(O_ERROR_MESSAGE => O_ERROR_MESSAGE
                                            I_C1 => :ps_restart_C1)
         then
            :pi_plsql_error   := 1;
            :ps_error_message := 'ERROR IN FUNCTION ProcPackage.ProcessFunction ' || O_ERROR_MESSAGE;
         end if;

      EXCEPTION
         WHEN OTHERS THEN
            :pi_plsql_error   := 1;
            :ps_error_message := 'ERROR IN FUNCTION ProcPackage.ProcessFunction ' || sqlerrm;
      END;
   END-EXEC;

   if SQL_ERROR_FOUND
   {
      sprintf(err_data, 'Error on ProcPackage.ProcessFunction');
      WRITE_ERROR(SQLCODE,function,"",err_data);
      return(FATAL);
   }

   if (pi_plsql_error == 1)
   {
      sprintf(err_data, "Error: %s", ps_error_message);
      WRITE_ERROR(SQLCODE,function,"",err_data);
      return(FATAL);
   }
   /* --------------- */

   return(OK);

} /* end function process_ProcessFunction*/
```

Figura 34: Chamada da Função de Processamento da Base de dados através do ProC.

A figura 35 mostra a lista todos os ficheiros criados para um ProC. São gerados DDL's, DML's, ficheiros ProC, e um Package com as funções.

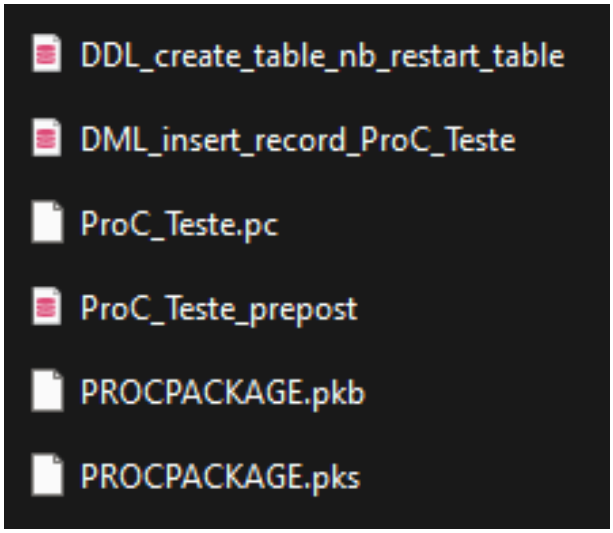


Figura 35: Ficheiros criados.

5.5.5 Table

A Table permite executar três operações - New, Modify e Drop. Portanto, caso o utilizador pretenda criar uma Table nova seleciona a opção New, caso pretenda alterar uma table existente seleciona o Modify e

caso pretenda apagar a Table seleciona a opção Drop.

Como tal, é necessário gerar três tipos diferentes de ficheiros. Para a opção New, a table é criada de raiz, ou seja, é feito uma operação "CREATE TABLE"como demonstrado na figura 36.

```
-- Drop Table
DROP TABLE Users;
-- Drop table synonym
DROP PUBLIC SYNONYM Users;

-- Create Table
CREATE TABLE Users
(
  Nome VARCHAR2(200)  NOT NULL
 ,Numero NUMBER NOT NULL
);

-- Add comments to the table
COMMENT ON TABLE Users IS 'Table de teste'
-- Add comments to the columns
COMMENT ON COLUMN Users.Nome IS 'Nome do Utilizador';
COMMENT ON COLUMN Users.Numero IS 'Número';
```

Figura 36: DDL gerado para criação de uma tabela.

Para a opção Modify, é feita uma alteração à table ou seja é utilizado uma operação "ALTER TABLE". No entanto, dentro desta operação existe um funcionalidade no que diz respeito as colunas, isto é, é possível adicionar, alterar, ou remover colunas. Portanto, quando uma coluna tem o atributo New, é feito um "ALTER TABLE .. ADD COLUMN"para adicionar uma nova coluna, quando é feito Alter, é feito um "ALTER TABLE ... MODIFY"para alterar características da coluna e por fim quando é feito Drop é feito "ALTER TABLE .. DROP COLUMN"para eliminar a coluna como mostrado na figura 37

```
ALTER TABLE Users
ADD Nome VARCHAR2(200) NOT NULL ;
COMMENT ON COLUMN Users.Nome IS 'Nome do Utilizador';

ALTER TABLE Users
MODIFY Numero NUMBER NOT NULL;

ALTER TABLE Users
DROP COLUMN Morada;
```

Figura 37: DDL gerado para alteração de uma tabela.

Para o opção de Drop é feita uma simples operação "DROP TABLE"que elimina definitivamente a tabela, como na figura 38.

```
-- Drop Table
DROP TABLE Users;
-- Drop table synonym
DROP PUBLIC SYNONYM Users;
```

Figura 38: DDL gerado para remoção de uma tabela.

Além disto, ainda é gerado código referente às Constraints, sendo estas as seguintes:

- Primary Key
- Foregin Key
- Unique Index
- Index
- Check Constraint

Estas opções têm um código especifico, como é possível ver na figura 39.

```
ALTER TABLE Users
DROP CONSTRAINT Users_PK;

ALTER TABLE Users
ADD CONSTRAINT Users_PK PRIMARY KEY(Nome);
```

Figura 39: Instrução utilizada para adicionar, remover ou alterar *Constraints*.

5.5.6 Package

O Package vai criar dois novos ficheiros, sendo um deles o Package Spec e o outro o Package Body. O Spec vai constituir as assinaturas das funções de forma a estas serem chamadas pelo exterior, como é possível ver na figura 40.

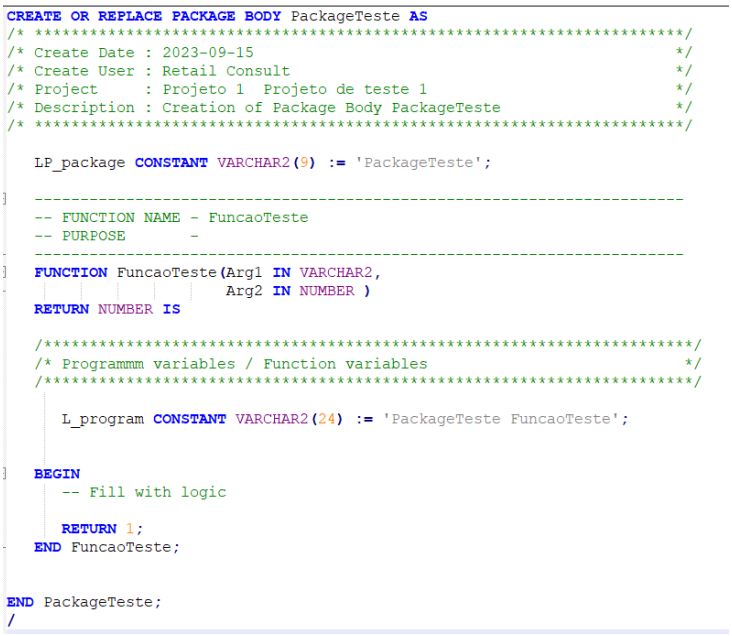
```
CREATE OR REPLACE PACKAGE PackageTeste AS
/* ********************************************************/
/* Create Date : 2023-09-15                               */
/* Create User : Retail Consult                           */
/* Project     : Projeto 1  Projeto de teste 1            */
/* Description : Creation of Package Spec PackageTeste    */
/* ********************************************************/

   ------------------------------------------------------------
   -- FUNCTION NAME - FuncaoTeste
   -- PURPOSE       -
   ------------------------------------------------------------
   FUNCTION FuncaoTeste(Arg1 IN VARCHAR2,
                        Arg2 IN NUMBER)
   RETURN NUMBER;


END PackageTeste;
/
```

Figura 40: Especificação do *Package*.

No entanto, o Body vai ter o mesmo do Spec, mas também vai conter um espaço para a função ser construida manualmente pelo programador, como na figura 41.



```
CREATE OR REPLACE PACKAGE BODY PackageTeste AS
/* ************************************************************************/
/* Create Date : 2023-09-15                                               */
/* Create User : Retail Consult                                           */
/* Project     : Projeto 1  Projeto de teste 1                            */
/* Description : Creation of Package Body PackageTeste                    */
/* ************************************************************************/

   LP_package CONSTANT VARCHAR2(9) := 'PackageTeste';

   ---------------------------------------------------------------------
   -- FUNCTION NAME - FuncaoTeste
   -- PURPOSE       -
   ---------------------------------------------------------------------
   FUNCTION FuncaoTeste(Arg1 IN VARCHAR2,
                        Arg2 IN NUMBER )
   RETURN NUMBER IS

   /************************************************************************/
   /* Programmm variables / Function variables                             */
   /************************************************************************/

      L_program CONSTANT VARCHAR2(24) := 'PackageTeste FuncaoTeste';


   BEGIN
      -- Fill with logic

      RETURN 1;
   END FuncaoTeste;


END PackageTeste;
/
```

Figura 41: Corpo do *Package*.

5.5.7 Wif

Para os Wifs é inicialmente gerado um ficheiro .sql com a sua parametrização.

Além disto também é gerado um package, constituído por um ficheiro para o spec e outro para o body, como na figura 42.

O package spec é constituído por duas funções: PURGE_TRANS e EXECUTE_TRANS. Como o nome indica a primeira tem como função fazer a purga dos dados recolhidos pelo wif e a segunda têm a função de ler os dados do ficheiro. Para completar, o body contém um template inicial destas duas funções com um código genérico, mas também contem duas funções essenciais, sendo uma principal denominada LOAD_RECORDS responsável por fazer a leitura e respetivas validações. Esta função utiliza a segunda função para para fazer o split das linhas do ficheiro.

```
CREATE OR REPLACE PACKAGE NB_INTF_WEB_ITRT_SQL IS
/* ************************************************ */
/* Create Date : 2023-09-16                         */
/* Create User : Retail Consult                     */
/* Project     : Projeto 1  Projeto de teste 1      */
/* Description : Creation of Wif PksTeste           */
/* ************************************************ */

FUNCTION EXECUTE_TRANS(teste IN VARCHAR2)
RETURN BOOLEAN;

FUNCTION PURGE_TRANS(teste IN VARCHAR2)
RETURN BOOLEAN;

END NB_INTF_WEB_ITRT_SQL;
/
```

Figura 42: Especificação do Package gerado para o WIF.

5.5.8 Report

Neste caso é apenas gerado um ficheiro .sql que efetua as operações necessárias para fazer setup do Report. Este tipo de objeto é construido através do Oracle Reports Builder, portanto este Script apenas é necessário para fazer a parametrização das tabelas de forma ao Report ser utilizado e também é utilizado para definir as permissões através de um sistema de perfis.

Para os perfis são realizados um insert na tabela nb_profiles para adicionar o novo perfil associado ao report e na tabela nb_profile_privs para associar este mesmo perfil a um outro perfil do nível acima, figura 43.

```
DELETE nb_profiles
 WHERE nb_profiles.profile = l_rep_profile;

INSERT INTO nb_profiles
   (profile
   ,profile_desc)
VALUES
   (l_rep_profile_npo3
   ,'Profile for Report ' || l_report);

DELETE nb_profile_privs
 WHERE nb_profile_privs.granted_profile = l_rep_profile_npo3;

INSERT INTO nb_profile_privs
   (profile
   ,granted_profile
   ,profile_type)
VALUES
   (l_rep_profile_npo2
   ,l_rep_profile
   ,'P');
```

Figura 43: Inserção de permissões para aceder ao Report.

Quando à parametrização do form, em primeiro lugar é feito um insert na tabela nb_rep_report, cuja função é introduzir o report na base de dados e associar o perfil. De seguida são feitos vários inserts

na talela nb_rep_report_param consoante o número de parâmetros do report, ou seja, caso exista dois parâmetros são feitos dois inserts. Por fim é feito um insert na tabela nb_report_label_shadow.

5.5.9 Form

Este objeto é semelhante ao anterior, dado o facto que é necessário o Oracle Forms Builder para fazer a criação dos forms. Portanto, este script gerado é utilizado da mesma forma que o script do Report.

Em primeiro lugar, como apresenta a figura 44 são feitos os inserts nas tabelas de permissões como feito anteriormente.

```
INSERT INTO nb_profiles
   (profile
   ,profile_desc)
VALUES
   ('NPO3_' || l_form || '_' || tmodes(i)
   ,NULL);

INSERT INTO nb_profile_privs
   (profile
   ,granted_profile
   ,profile_type)
VALUES
   ('NPO2_SUPPORT_ALL'
   ,'NPO3_' || l_form || '_' || tmodes(i)
   ,'P');
```

Figura 44: Inserção de permissões para aceder ao Form.

De seguida, é necessário que o form esteja disponível no menu da aplicação, para isso é feito um insert na tabela nav_element_mode e na nb_nav_element_mode_role para associar o perfil ao form.

Capítulo 6

Testes

Neste capitulo são descritos os testes realizados após e durante os desenvolvimentos. Este tipo de testes são vitais, pois ajudam a encontrar e corrigir erros, garantido que a qualidade é assegurada e que a aplicação funciona corretamente, como atesta Sneha [17]. Principalmente o foco dos testes realizados são para que o utilizador tenha uma boa experiência de utilização.

Durante os desenvolvimentos foram testados os seguintes pontos:

- Autenticação
- Menu
- Preenchimento dos Formulários
- Gerar ficheiros

Inicialmente, foram feitos testes à Autenticação mas como esta funcionalidade é implementada através do Django, o único que foi testado foi a interface do utilizador. Ao longo do desenvolvimento esta interface foi alterada de forma a que o utilizador consiga fazer *sign up* e *login* rapidamente. Além disto, numa fase inicial não foi implementado forma de recuperar password, mas após testar foi identificado que seria essencial e por isso foi inserido na aplicação.

Após esta fase ter sido concluída, foram iniciados os desenvolvimentos do Menu e dos Formulários, sendo inicialmente desenvolvido o menu da imagem 45 que posteriormente, quando se selecionava uma das opções apresentadas era iniciado o menu da figura 46 que apresentava todos os dados do objeto escolhido. No entanto, foi identificado que não seria a melhor opção para o utilizador devido ao processo de iniciar um novo objeto ser bastante lento e com passos excessivos, portanto foi decidido avançar com os desenvolvimentos do menu em formato de árvore que simplifica a interface.

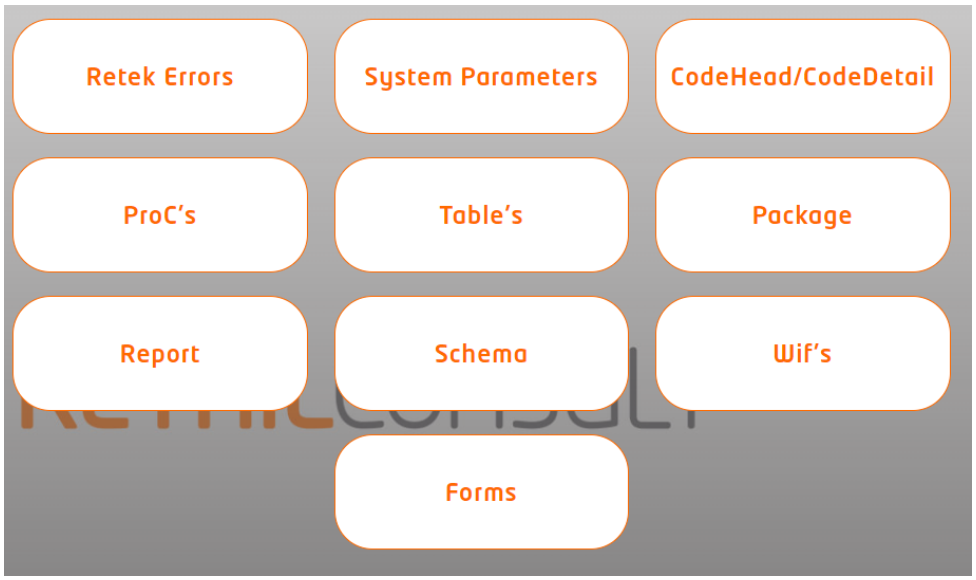


Figura 45: Menu da aplicação antes do testes

Figura 46: Tabela com todos os dados de um objeto.

Quando à parte de gerar ficheiros foi essencial fazer imensos testes com diferentes tipos de input, devido a serem encontrados imensos erros nos *templates*, como por exemplo, virgulas, parênteses e aspas em falta ou erros a gerar o ficheiro. Além disto, foi identificado que existia uma falta de *feedback* da aplicação quando um utilizador gera os ficheiros, portanto foi introduzido notificações quando dá erro ou quando dá sucesso.

6.1 End-to-end Testing

6.1.1 Estratégia

O teste End-to-end (**E2E**) é uma estratégia utilizada para verificar se a aplicação funciona como esperado em todas as camadas do software [18].

Estes testes como o nome indica, consistiram de verificar todo o fluxo que o utilizador tem de percorrer para gerar um ficheiro. Para isso foram efetuados os seguintes passos:

1. Registo de um novo utilizador

2. Login
3. Criação de um Projeto
4. Criação de um objeto
5. Manutenção do objeto
6. Gerado o ficheiro do objeto criado
7. Remoção do objeto

Para não deixar objetos sem testar foi repetido o processo desde o número 4 ao número 7 para todos os objetos. De forma a completar os testes, após cada Inserção, alteração ou remoção de um objeto foi analisado na base de dados se os dados estavam corretos para não existir informação em falta ou vice-versa.

6.1.2 Teste prático

Este teste exemplo vai mostrar todos os passos necessários para o utilizador gerar um objeto. Neste caso será um Retek Error, contudo o fluxo é igual para todos.

Em primeiro lugar é criado um projeto como demonstra a figura 47 e 48. Para criar é apenas necessário pressionar o botão central, preencher o nome e descrição e por fim submeter.

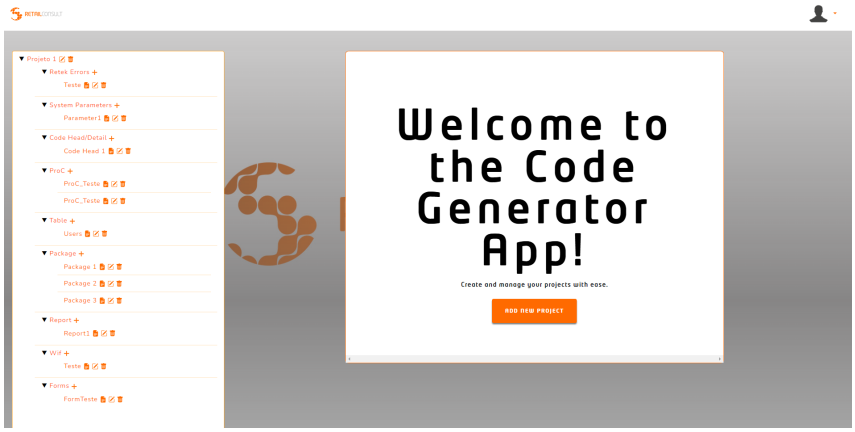


Figura 47: Página principal.

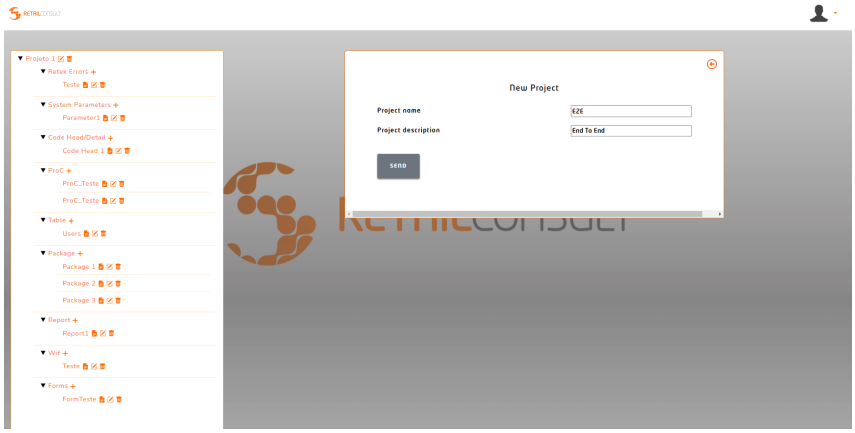


Figura 48: Formulário de criação de um novo projeto.

Esta criação resultou num novo Projeto no menu lateral onde dá a possibilidade de criar todos os objetos como mostra a figura 49.

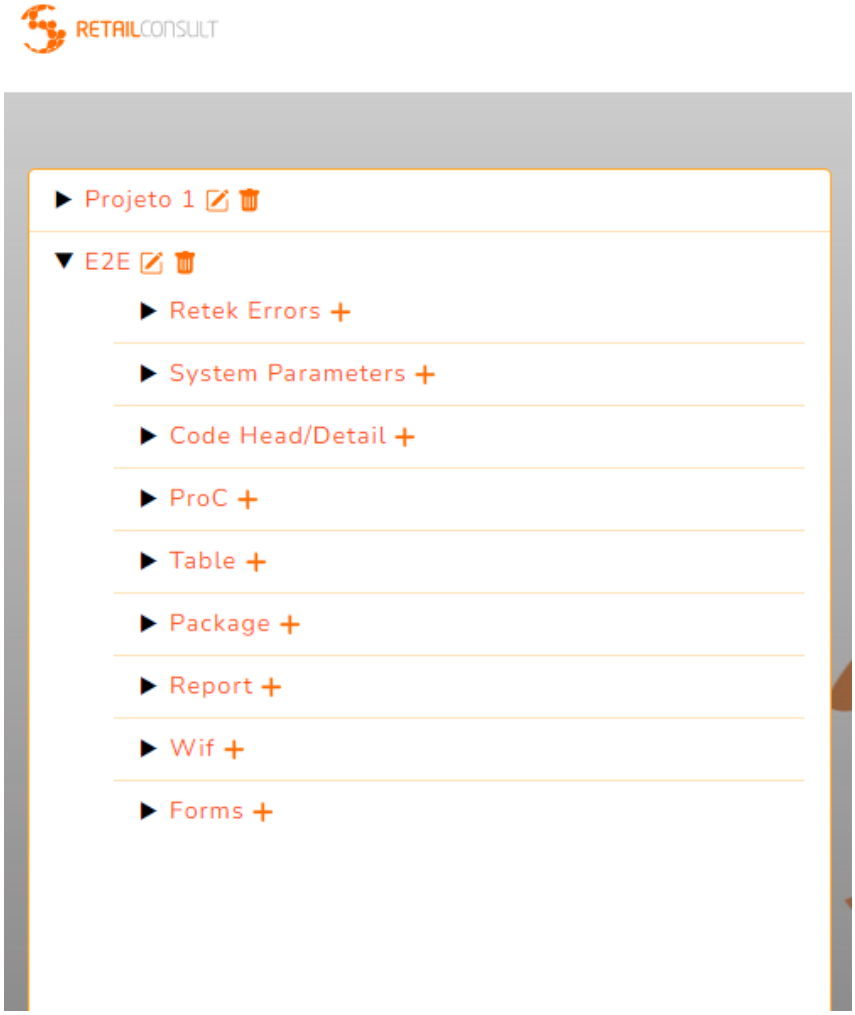


Figura 49: Menu com o novo projeto.

Com isto, é selecionado o botão "+"para adicionar um novo objeto. Com esta ação é aberto um formulário, como apresentado na figura 50, que é necessário preencher e submeter para o objeto ficar disponível para ser gerado. Esta criação resulta em uma nova entrada na lista de Retek Errors, como na figura 51, dando as opções de ser alterada, removida ou gerar código.

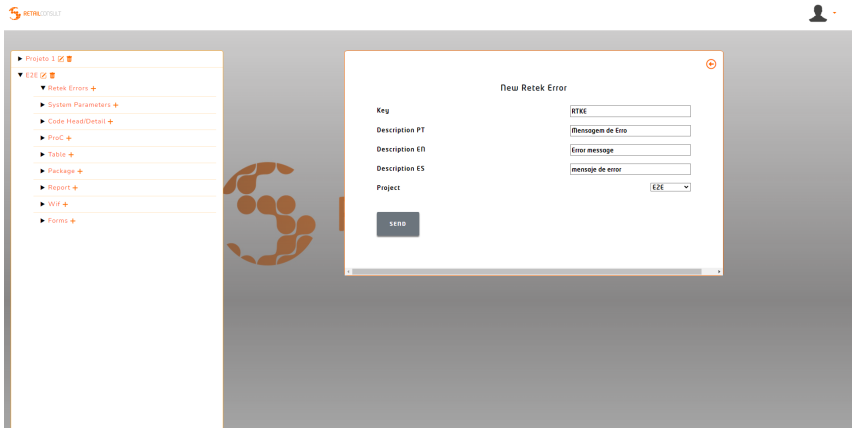


Figura 50: Formulário dos Retek Errors.

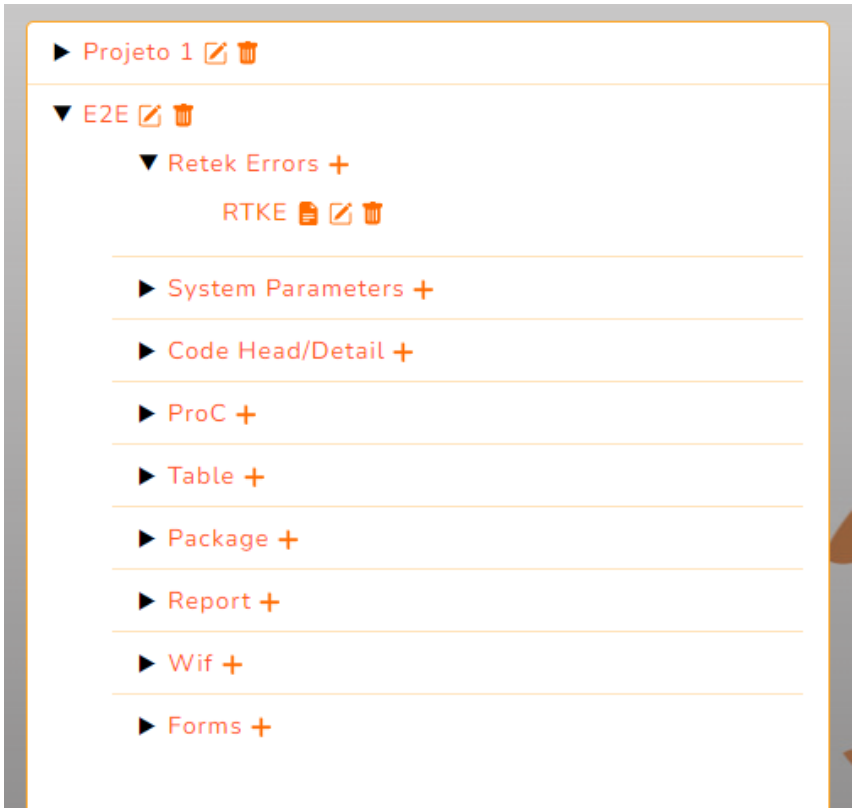


Figura 51: Menu com o objeto criado.

Por fim, para gerar o objeto é selecionado o primeiro botão que apresenta uma mensagem de sucesso,

como na figura 52, para notificar o utilizador que o ficheiro foi gerado. Com isto, o ficheiro gerado na figura 53 encontra-se pronto para ser utilizado.

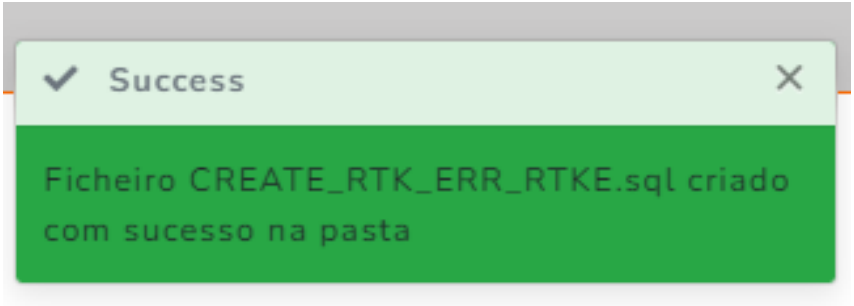


Figura 52: Mensagem de sucesso.

CREATE_RTK_ERR_RTKE.sql

```
/* ********************************************************************************************************** */
/* Create Date : 2023-10-05                                                                                   */
/* Create User : Retail Consult                                                                               */
/* Project     : E2E_End To End                                                                               */
/* Description : Creation of Retek Error RTKE                                                                 */
/* ********************************************************************************************************** */

DECLARE
   l_rtk_key      rtk_errors.rtk_key%TYPE  := 'RTKE';
   l_rtk_user     rtk_errors.rtk_user%TYPE := 'RETAILCONSULT';
   l_rtk_text_pt  rtk_errors.rtk_text%TYPE := 'Mensagem de Erro';
   l_rtk_text_eng rtk_errors.rtk_text%TYPE := 'Error message';
   l_rtk_text_es  rtk_errors.rtk_text%TYPE := 'mensaje de error';

BEGIN

   DELETE rtk_errors
    WHERE rtk_errors.rtk_key = l_rtk_key;

   INSERT INTO rtk_errors
      (rtk_key
      ,rtk_lang
      ,rtk_text
      ,rtk_user
      ,rtk_approved)
   VALUES
      (l_rtk_key
      ,1
      ,l_rtk_text_pt
      ,l_rtk_user
      ,'Y');

   INSERT INTO rtk_errors
      (rtk_key
      ,rtk_lang
      ,rtk_text
      ,rtk_user
      ,rtk_approved)
   VALUES
      (l_rtk_key
      ,3
      ,l_rtk_text_eng
      ,l_rtk_user
      ,'Y');

   INSERT INTO rtk_errors
      (rtk_key
      ,rtk_lang
      ,rtk_text
      ,rtk_user
      ,rtk_approved)
   VALUES
      (l_rtk_key
      ,4
      ,l_rtk_text_es
      ,l_rtk_user
      ,'Y');

EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK;
END;
/
```

Figura 53: Ficheiro gerado.

6.2 Testes de compatibilidade

Os testes de compatibilidade do browser, são necessários devido às aplicações Web darem terem um forte carregamento do lado do cliente com tecnologias como o Ajax, o Flash e o HTML dinâmico, que diferem entre navegadores. Além disto, o vasto número de browsers, quase 100, juntamente com vários sistemas operativos, resultam em centenas de diferentes ambientes do lado do cliente para aplicações Web, como atesta Mesbah [19]. Portanto, estes testes vão verificar se o website que foi desenvolvido está a funcionar como esperado num determinado navegador ou não [20].

Estes testes foram subdivididos nas seguintes duas partes:

- Compatibilidade entre browsers.
- Compatibilidade entre dispositivos.

Nos testes dos navegadores, o Google Chrome, Microsoft Edge e Brave foram avaliados, onde as funcionalidades foram validadas e o design permaneceu conforme o esperado. Para o segundo caso de teste, embora o design para dispositivos móveis não tenha sido considerado, foram realizados testes em dispositivos com diferentes resoluções para verificar se estava com aparência aceitável.

6.3 Resultados

Em relação aos testes **E2E**, apesar de terem sido feitos vários testes unitários ao longo do desenvolvimento e corrigido vários bugs, foram ainda encontrados outros problemas para casos específicos tanto na geração dos ficheiros como no preenchimento de formulários. Além disto foram feitas pequenas alterações a nível de UI, como por exemplo, alinhamento de campos de input e alteração de nomes de atributos para ser ainda mais intuitivo.

Como exemplificado anteriormente, foram feitos testes para cada objeto, e como é possível concluir através do teste exemplo, é possível criar um objeto válido rapidamente em pouco passos.

Em relação à compatibilidade, o resultado foi positivo para o teste entre navegadores, mas como esperado, para os diferentes dispositivos, devido a não ser um foco do projeto, a aplicação encontra-se com problemas para dispositivos móveis, sendo estes problemas a nível de UI, visto que as componentes não se encontravam totalmente responsivos.

De uma forma geral, os testes **E2E** foram essenciais para encontrar bugs que não foram detetados durante o desenvolvimento. Além disso, eles foram amplamente utilizados para a execução de testes finais, a fim de verificar se o fluxo do utilizador estava conforme o esperado.

Quanto á qualidade de código foi utilizada a ferramenta SonarQube [1] para efetuar os testes. Utilizando novamente um projeto real foi verificado que existem code smells e falhas de comentários no código como por exemplo a identificação de quem efetuou o desenvolvimento.

Tomando como exemplo o ficheiro apresentado na figura 54 que representa uma alteração a uma code_detail. Este contém um code smell e não apresenta um cabeçalho com a informação do desenvolvedor, data, projeto e motivo da alteração, provavelmente por esquecimento de quem o criou.

	Lines of Code	Bugs	Vulnerabilities	Code Smells	Security Hotspots	Coverage	Duplications
RMS.DML	90,745	6	0	4,913	0	0.0%	58.0%
DML_code_detail.sql	24	0	0	1	0	0.0%	0.0%

Figura 54: Ficheiro analisado com o SonarQube

Com o gerador de código este tipo de problemas já não acontecem devido a ser tudo automaticamente criado em segundos de acordo com as regras. Portanto, o gerador de código reduz os erros e aumenta a qualidade do código, além de diminuir a quantidade de código repetitivo que os programadores têm de escrever [21].

Para concluir, como demonstrado no estudo [5], a produtividade do programador é significativamente melhorada pela utilização de geradores de código como evidenciado pelos resultados experimentais em várias métricas.

Em primeiro lugar, o tempo médio de desenvolvimento foi reduzido de 397,18 minutos durante a implementação manual para apenas 4,17 minutos quando se utilizou o gerador de código. Isto corresponde a uma melhoria de 98,95% no tempo médio de desenvolvimento [5].

Outra métrica importante é o número de execuções de teste. Com o gerador de código, foram necessárias apenas duas execuções de teste. Em contrapartida, a implementação manual exigiu numerosas execuções de teste devido ao desenvolvimento iterativo e à correção de erros. Isto produziu uma notável melhoria de 93,97% na contagem média de execuções de teste [5].

A contagem de erros foi a terceira métrica examinada. O gerador de código apresentou uma ausência de erros. Por outro lado, a implementação manual implicou a descoberta e a resolução de vários erros [5].

Além disso, os dados revelam que os ficheiros HTML que foram gerados no estudo são mais concisos em comparação com os ficheiros criados manualmente, com uma melhoria observada de 49,37% no tamanho médio do código, sendo esta vantagem atribuída à abordagem baseada em *templates* [5].

[1] https://www.sonarsource.com/products/sonarqube/

Em resumo, estes resultados validam as vantagens e benefícios dos geradores de código discutidos nesta dissertação.

Capítulo 7

Conclusões e trabalho futuro

7.1 Conclusões

Em conclusão, esta dissertação de mestrado apresenta uma exploração abrangente do desenvolvimento e implementação de um gerador de código para *Oracle Retail* utilizando a *framework Django*. O objetivo deste trabalho foi desenvolver um processo de geração de código para aplicações *Oracle Retail*, com um foco específico no aumento da eficiência e qualidade do código.

Foram alcançados vários resultados importantes. Foi desenvolvido um gerador de código Oracle Retail totalmente funcional usando o *Django*. Esta aplicação automatiza a criação de vários componentes necessários para as aplicações Oracle Retail, reduzindo significativamente o tempo e o esforço necessários para a codificação manual, o que leva a uma maior produtividade do programador. Portanto, esta implementação levou a um aumento substancial da eficiência durante a fase de desenvolvimento. Os programadores podem agora gerar objetos rapidamente, permitindo-lhes concentrar-se em tarefas de nível superior e reduzindo o risco de erro humano.

O código gerado pelo sistema demonstra consistentemente alta qualidade, devido à utilização de *templates* que aderem as melhores práticas. Isto contribui para que as implementações de projetos sejam mais fiáveis, além de reduzir a possibilidade de erros futuros.

Após todos estes objetivos alcançados, é importante referir os vários desafios que surgiram durante o decorrer do projeto. O desenvolvimento de uma aplicação Web tornou-se bastante desafiante, mas foi algo ultrapassado devido aos requisitos estarem bem definidos desde o início apesar de ter existido alterações a meio do projeto, como é normal suceder em qualquer projeto de média ou elevada dimensão.

7.2 Perspetiva de trabalho futuro

Fazendo a reflexão sobre o progresso e o potencial deste projeto de dissertação, que se centra num gerador de código para *Oracle Retail*, seria interessante que os utilizadores partilhassem projetos com outros utilizadores, permitindo que colaborassem em projetos dando a possibilidade de uma melhor organização das equipas responsáveis por implementações *Oracle Retail*.

Além disso, a manutenção dos *templates* de código pode ser melhorada permitindo que estes *templates* sejam atualizados manualmente pelos utilizadores, tomando como partida a ideia referida anteriormente de definir ficheiros externos com os *templates*. Para completar, seria interessante adicionar a funcionalidade de permitir aos utilizadores adicionar os seus próprios *templates* permitindo que a aplicação seja escalável.

Outro ponto importante é a segurança, que deve continuar a ser uma prioridade máxima. Implementar medidas de segurança robustas é crucial para salvaguardar o código e os dados gerados pelo utilizador, incutindo confiança na integridade do sistema.

Por fim, esta aplicação está a ser pensada de forma a se efetuar o *rollout* para as restantes equipas dentro da Retail Consult.

Referências bibliográficas

[1] Django introduction - Learn web development | MDN. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction. [acedido a Outubro de 2023].

[2] 21 compelling retail statistics [2023]: How many retailers are in the us? https://nrf.com/topics/economy/state-retail. [acedido a Outubro de 2023].

[3] State of retail. https://nrf.com/topics/economy/state-retail. acedido a 16 Dezembro de 2022.

[4] Oracle. Oracle retail. https://www.oracle.com/pt/retail/, 2023. [acedido a Outubro de 2023].

[5] Burak Uyanik and Veysel Harun Şahin. A template-based code generator for web applications. *Turk. J. Of Electr. Emg. Comput. Sci.*, 28(3):1747–1762, May 2020. doi: 10.3906/elk-1910-44.

[6] General python faq. https://docs.python.org/3/faq/general.html#what-is-python. [acedido a Outubro de 2023].

[7] Dayo J Osunrinde. *The design and implementation of a web application for visualizing chemical structures and information using test driven development*. PhD thesis, Memorial University of Newfoundland, 2017.

[8] Django. https://docs.djangoproject.com/en/4.1/, 2013. [acedido a Outubro de 2023].

[9] Jonathan Moore. Building a reusable application with django: Building a reusable application with django. 2016.

[10] Eugene Syriani, Lechanceux Luhunu, and Houari Sahraoui. Systematic mapping study of template-based code generation. *Computer Languages, Systems Structures*, 52:43–62, 2018. doi: 10.1016/j.cl.2017.11.003.

[11] Sven Jorges. *Construction and evolution of code generators: A model-driven and service-oriented approach*, volume 7747. Springer, 2013.

[12] Joao M Fernandes and Ricardo J Machado. *Requirements in Engineering Projects*. Lecture Notes in Management and Industrial Engineering. Springer International Publishing, Cham, Switzerland, 1 edition, July 2015. doi: 10.1007/978-3-319-18597-2.

[13] Karl Wiegers and Joy Beatty. *Software Requirements*. Microsoft Press, Redmond, WA, 3 edition, August 2013.

[14] Steven Feuerstein. *Oracle PL/SQL Best Practices*. O'Reilly Media, Sebastopol, CA, 2 edition, November 2007.

[15] Raymond Turner. *Modularity*, pages 141–149. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018. doi: 10.1007/978-3-662-55565-1_17.

[16] Haroon Shakirat Oluwatosin. Client-server model. *IOSR Journal of Computer Engineering*, 16(1):67–71, 2014.

[17] Karuturi Sneha and Gowda M Malle. Research on software testing techniques and software automation testing tools. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pages 77–81, 2017. doi: 10.1109/ICECDS.2017.8389562.

[18] End-to-end testing. https://docs.gitlab.com/ee/development/testing_guide/end_to_end/index.html. [acedido a Outubro de 2023].

[19] Ali Mesbah and Mukul R. Prasad. Automated cross-browser compatibility testing. ICSE '11, page 561–570, New York, NY, USA, 2011. Association for Computing Machinery. doi: 10.1145/1985793.1985870.

[20] LAMBDATEST. Browser compatibilty testing: A must for web apps and websites. Technical report, Lambda Computing. [acedido a Outubro de 2023].

[21] Akhan Akbulut and Sezer Toprak. Code generator framework for smart TV platforms. *IET Softw.*, 13(4):268–279, August 2019.