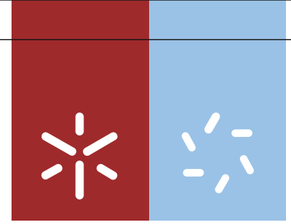


Universidade do Minho
Escola de Ciências

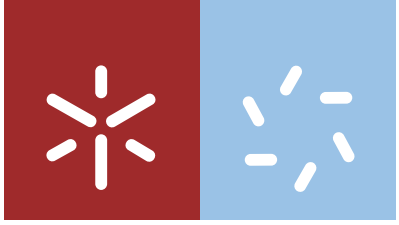
Gabriel Polide da Silva

Nova Plataforma Fiscal

outubro de 2017

Gabriel Polide da Silva **Nova Plataforma Fiscal**

UMinho|2017



Universidade do Minho
Escola de Ciências

Gabriel Polide da Silva

Nova Plataforma Fiscal

Relatório de Estágio
Mestrado em Matemática e Computação
Área de Especialização em Matemática e Ciências da Computação

Trabalho realizado sob orientação do
Professor Doutor Rui Ralha (Universidade do Minho)
e do
Eng. Miguel Marques (Primavera BSS)

outubro de 2017

DECLARAÇÃO

Nome: Gabriel Polide da Silva

Endereço electrónico: gabriel.polide.silva@hotmail.com

Telefone: 933428741

Número do Bilhete de Identidade: 13815997

Título do Relatório de Estágio: Nova Plataforma Fiscal

Orientadores: Professor Doutor Rui Ralha (Universidade do Minho) e Eng. Miguel Marques (Primavera BSS)

Ano de conclusão: 2017

Designação do Mestrado: Mestrado em Matemática e Computação, especialização em Matemática e Ciências da Computação

Universidade do Minho, 31 de Outubro de 2017

Assinatura:

AGRADECIMENTOS

O percurso académico de um estudante é marcado por muitos fatores, entre eles as festas, as horas de estudo, o stress dos testes, as noites deixadas por dormir para conseguir entregar os trabalhos em cima da hora ou até mesmo depois da hora, confesso, mas um dos pontos mais significativos da vida de um estudante é as pessoas que conhece e interage durante o seu percurso. Nesta secção vou agradecer de uma forma direta e breve a quem diretamente e indiretamente me acompanhou e ajudou no meu percurso como estudante.

A todos os professores que até hoje se cruzaram no meu percurso, desde a escola primária até à última aula do curso de mestrado. A eles devo uma grande parte do meu conhecimento e capacidades. Alguns deles acabaram por mudar a minha vida por completo, pois por causa deles eu decidi seguir o caminho em que me encontro hoje e deram-me luzes do que eu poderia ser, em momentos em que eu estava perdido e sem noção de direção. A eles agradeço todo o esforço que fizeram para me transmitir conhecimento.Eu costumo dizer que os professores são como degraus de escada, onde cada um deles nos vai ajudar a subir no edifício a que podemos chamar de "conhecimento".

Ao Eng. Miguel Marques, o meu coordenador por parte da Primavera BSS, com quem tive o previlégio e orgulho de colaborar, agradeço os desafios propostos e a constante motivação durante toda a realização do projeto. Agradeço também a amizade e a boa disposição, presentes em todos os momentos.

Ao Professor Dr. Rui Ralha, meu orientador por parte da Universidade do Minho, agradeço o tempo disponível que teve para me orientar, pelo apoio e pelas ideias que me transmitiu. Agradeço também pela confiança que depositou em mim e pelo sentido de responsabilidade que me incutiu ao longo do projeto.

À Primavera BSS, por me ter dado a oportunidade de realizar este projeto, de me oferecer um posto nas suas instalações e de me ter dado a oportunidade de interagir com o seus profissionais.

A todos os colegas e amigos que tive o prazer de conhecer no ambiente empresarial, a quem agradeço as ajudas e as dicas que me ajudaram a progredir no projeto e a obter novos conhecimentos. Mas para além disso, tenho de lhes agradecer a companhia e os bons momentos passados.

Aos amigos, que sempre me proporcionam bons momentos fora das tarefas de estudante/trabalhador, algo que eu julgo ser indispensável. Agradeço em particular ao José Simão, que estando no mesmo "barco"que eu (realizar uma tese num contexto empresarial na mesma empresa) acabamos por trabalhar lado a lado, onde tive a oportunidade de fazer esta boa amizade.

À família, especialmente aos meus pais e irmão, a quem agradeço todo o apoio incondicional e todos os seus esforços e educação que me ofereceram ao longo da minha vida e ao longo deste projeto. A eles dedico este trabalho.

Resumo

O objetivo principal deste projeto é a migração de uma plataforma desktop (Primavera Fiscal Reporting) para a web, reaproveitando todas as funcionalidades atuais da mesma, tendo como foco maior a produtividade, escalabilidade e disponibilidade. O desafio é conseguir reutilizar o mais possível a lógica de negócio [1] já existente continuando a oferecer aos clientes, via web, uma plataforma eficiente para a entrega das suas declarações fiscais. A ideia base é ter uma só lógica de negócio capaz de servir todos os clientes em simultâneo. A implementação desta nova plataforma consiste no desenvolvimento de duas componentes: a interface (uma aplicação web) e a camada de serviços, que será a ponte entre a interface e a lógica de negócio. Uma parte importante do processo de migração consistiu no desenvolvimento de uma ferramenta informática que gera páginas web a partir de ficheiros xml relativos aos modelos de declarações fiscais.

[1] Em engenharia de software, em particular em análise e desenho orientado a objetos, o termo lógica de negócio (em inglês: business logic) são rotinas que realizam entradas de dados, consultas aos dados, geração de relatórios e mais especificamente todo o processamento que se realiza por trás da aplicação visível para o utilizador (Backoffice). (Origem: Wikipédia, a enciclopédia livre)

Abstract

The main objective of this project consist of a migration of a desktop platform (Primavera Fiscal Reporting) to the web, reusing all the current functionality, focusing more on productivity, scalability and availability. The challenge is to be able to reuse the current business logic as much as possible [2] continuing to offer customers via web an efficient platform for the delivery of their tax returns. The main idea is to have a single business logic that can serve all customers simultaneously. The implementation of this new platform consists of the development of two components: the interface (a web application) and the web services, which will be the bridge between the interface and business logic. An important part of the migration process consisted in the development of a computer tool that generates web pages from xml files related to tax declaration models.

[2] In software engineering, in particular in object-oriented analysis and design, the term business logic are routines that perform data inputs, data queries, report generation and more specifically all processing work behind the user application (Backoffice). (Source: Wikipedia, the free encyclopedia)

Conteúdo

1 Introdução

Nos dias de hoje a Internet já não é apenas uma plataforma de comunicação e de acesso à informação: ela é cada vez mais usada também para armazenar e processar dados (*cloud computing*).
Através do uso de *cloud computing*, na Internet surgiram aplicações onde é possível armazenar os nossos dados, que podem ser obtidos em qualquer lugar desde que haja ligação à Internet. Também surgiram formas de podermos pedir a uma entidade com mais capacidade computacional para nos resolver problemas que nas nossas máquinas demoraria muito mais tempo a ser solucionados.

Essa possibilidade de podermos processar dados na *cloud* veio mudar a visão de como uma aplicação pode ser desenvolvida. Com a capacidade de podermos processar os dados na *cloud*, o trabalho mais intenso de uma aplicação (que é o de processar e armazenar os dados) passa a ser uma função da infraestrutura distribuída que é a *cloud* e não dos recursos computacionais de que o cliente dispõe localmente.

Perante essas possibilidades agora disponíveis na Internet, as empresas veem oportunidades novas de negócio, fornecendo aos seus serviços via web, a que os os utilizadores podem aceder em qualquer lugar em que haja ligação à Internet.

Como acontece como qualquer sistema de gestão empresarial (Enterprise Resource Planning), o ERP Primavera é o sistema responsável por cuidar de todas as operações informáticas da empresa. O **PFR (Primavera Fiscal Reporting)** é uma aplicação desktop desenvolvida pela **Primavera BSS** e é um produto independente do **ERP PRIMAVERA** que tem como objetivo flexibilizar e facilitar as entregas fiscais a que as empresas estão legalmente obrigadas. Trata-se de um produto que permite gerir os vários modelos/declarações de acordo com o mercado de localização [3] das empresas e a periodicidade de entrega dessas mesmas declarações fiscais.

Para ultrapassar algumas limitações desta aplicação desktop, a Primavera BSS pretende desenvolver uma aplicação web capaz de usar os recursos da *cloud* para disponibilizar aos seus clientes todas as funcionalidades do PFR. Para além de estar mais disponível, também é objetivo fazer com que essa nova plataforma ofereça tudo o que a atual aplicação oferece com a adição de novas funcionalidades.

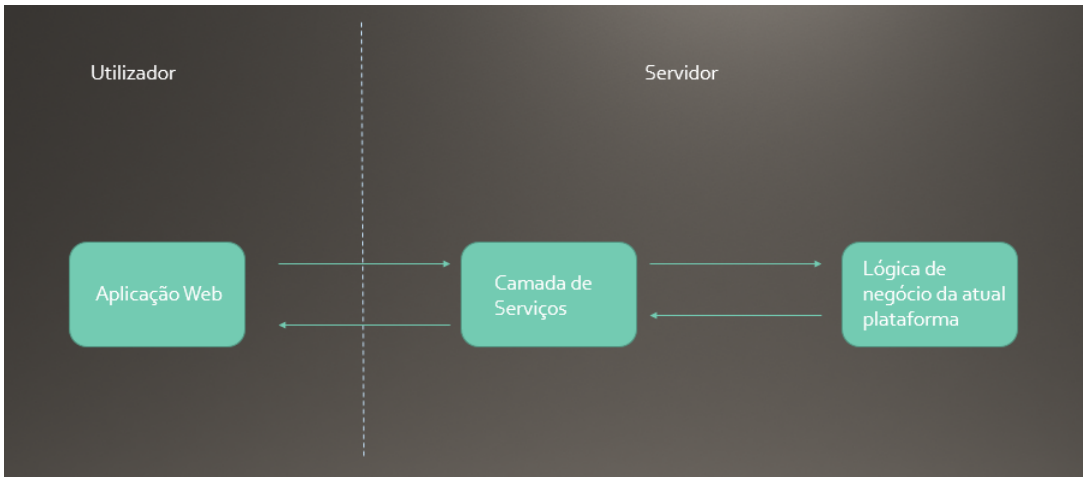
A ideia principal deste projeto foi o desenvolvimento de uma nova *interface* que consiga reaproveitar a lógica de negócio já existente. Para haver ligação entre a nova aplicação e as funcionalidades da lógica de negócio, foi desenvolvida uma camada de serviços intermédia.

A primeira tentativa neste sentido foi recorrer a uma plataforma de desenvolvimento de software criada na própria Primavera BSS designada por **Framework Elevation (versão 3.0)**. O trabalho realizado neste contexto levou-nos a concluir que o uso desta framework apresentava limitações várias para o desenvolvimento da aplicação pretendida.

O desenvolvimento da nova plataforma tem como base a framework Ionic que é uma plataforma de domínio público usada para o desenvolvimento de aplicações híbridas móveis.

[3] Faz-se notar que no Sistema Fiscal Português as taxas de tributação dependem da localização das empresas: Açores, Madeira, Continente.

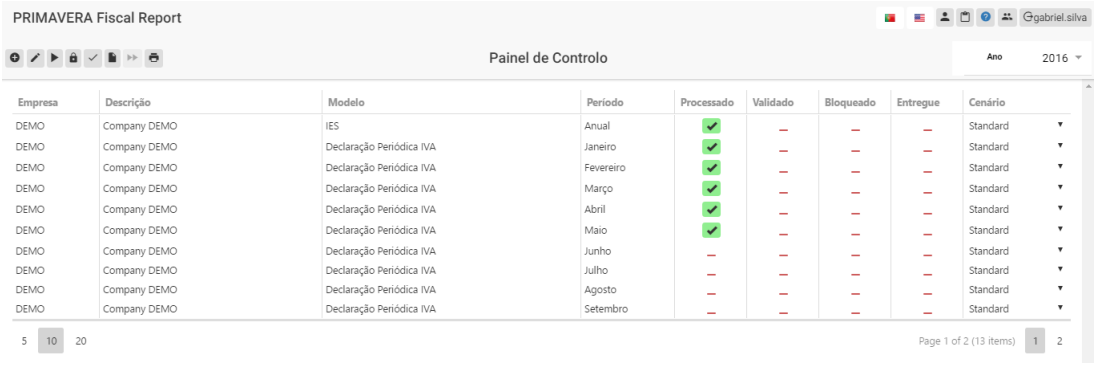
A camada de serviços foi desenvolvida em C#, na qual a aplicação web do lado do cliente se pode conectar e fazer pedidos, por seu lado, a camada de serviços faz a ponte entre a a aplicação web e a lógica de negócio já existente.



A migração para o ambiente web das funcionalidades existentes na atual aplicação desktop obrigou a alterações várias no código da lógica de negócio.
Um dos pontos mais importantes é o da nova plataforma fiscal ter duas versões, a versão que pode ser usada num PC, via navegador, e a versão Android, onde pode ser utilizada em *smartphone* ou tablet com software Android.

2 Uma análise resumida da aplicação web e das suas funcionalidades

De entre todas as funcionalidades que o PFR oferece, as mais relevantes para o utilizador são a listagem, processamento e edição dos modelos. O foco deste projeto é a desenvolvimento de uma aplicação web que consiga fornecer essas funcionalidades aos utilizadores via *cloud*.

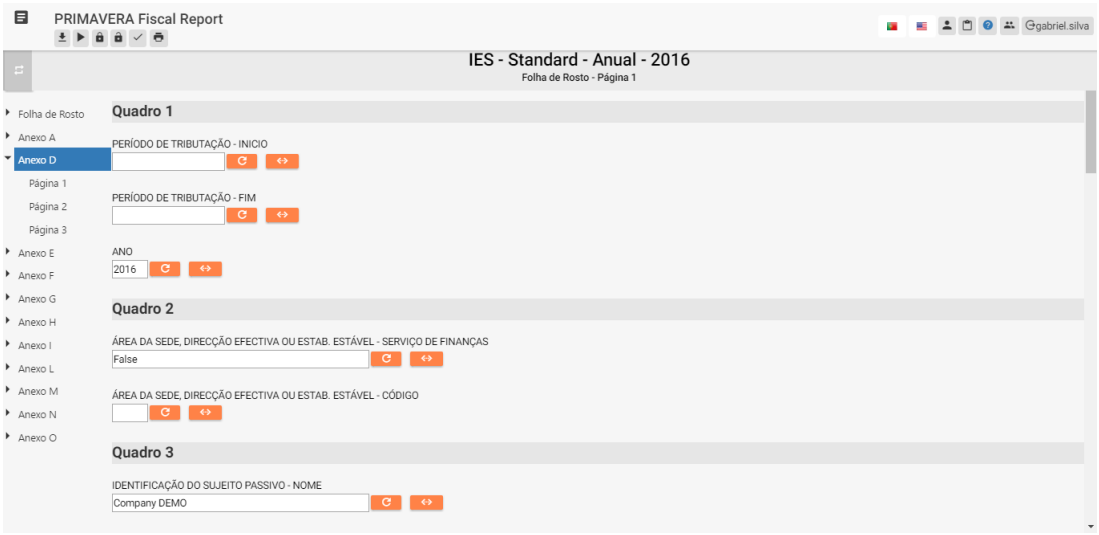


Na imagem acima podemos ver a página que a aplicação web apresenta ao utilizador depois de fazer o login.

Na versão desktop do PFR cada utilizador pode trazer para o seu Painel de Controlo novos modelos de declaração fiscal a partir de uma lista disponível para todos os clientes da empresa. Por falta de tempo, esta e outras funcionalidades presentes na atual plataforma desktop não foram incluídas na aplicação web desenvolvidas no âmbito deste projeto.

Observe-se que um mesmo utilizador pode preencher declarações respeitantes a uma ou mais empresas (identificadas na primeira coluna). Cada linha inclui toda a informação relevante da declaração (o modelo, o período, etc.).

A imagem em baixo corresponde à página que é apresentada quando no Painel de Controlo anterior se seleciona a primeira linha.



Um modelo tem várias páginas, sendo que a primeira a ser mostrada é a "Folha de Rosto - Página 1". Cada página pertence a um determinado grupo de páginas, tais como: "Anexo

A", "Anexo D", etc.

Cada campo de uma página possui uma fórmula, podendo essa fórmula conter funções e cálculos que podem ou não depender de valores de outros campos do modelo. Quando o utilizador processa um modelo, todos os campos de todas as páginas desse modelo serão processados, isto é, cada fórmula será calculada e irá retornar o valor, que por sua vez é mostrado no respetivo campo.
Depois de processar um modelo, esses valores são guardados automaticamente, isso significa que quando o utilizador fizer login no futuro e abrir esse mesmo modelo, esses valores estão já processados e nos seus respetivos campos.

3 Novos conceitos adquiridos

Durante a realização do estágio fui adquirindo novos conceitos técnicos, uns mais a nível prático e outros mais a nível teórico.
Uma das grandes vantagens de desenvolver uma tese num ambiente empresarial é a de poder obter novos conhecimentos e novas formas de analisar e resolver problemas com a ajuda e experiência de profissionais que lidam com essas situações diariamente.

Em seguida estarão listados os conceitos mais relevantes a nível técnico que adquiri durante o estágio, alguns deles não me eram de todo desconhecidos mas nunca os tinha posto em prática. Foi-me proposto o estudo de alguns conceitos antes de partir para o desenvolvimento da nova plataforma, pois seriam bases para poder entender com clareza como a plataforma atual é constituída e de como a nova deve ser construida.

Para o estudo destes conceitos foi necessário gastar uma fatia do tempo disponível para a realização do estágio, digamos por alto que essa fatia representa 5% do tempo. Nela foram feitas muitas pesquisas, tanto em português como em inglês, foram feitos exemplos e também a análise de resultados dos mesmos.

3.1 Framework Ionic

É com o uso da plataforma Ionic onde todas as funcionalidades e interface do lado do cliente são implementadas com o auxílio das tecnologias: Angular 2, TypeScript, HTML, CSS, DevExtreme, Bootstrap. Com a plataforma Ionic é possível que com um pequeno número de passos podermos construir uma aplicação base, que pode já vir com um menu lateral, com *tabs*, ou então em branco, sem nenhum componente de raiz. O Ionic fará todo o trabalho de construir toda uma família de pastas e ficheiros que serão a nossa aplicação, depois disso só precisamos de construir as nossas funcionalidades sobre essa aplicação base criada.

3.2 Padrão MVVM

É um padrão que nos ajuda a fazer uma separação de responsabilidades dentro de uma aplicação e com isso também vem uma melhor organização e manutenção do próprio código. [1]
Ele está dividido em 3 componentes, são elas: **Model**,**ViewModel** e a **View**.

View: É a interface que o utilizador vê e interage
Model: Contém a lógica de negócio e os dados.
ViewModel: Podemos ver a ViewModel como a ligação entre a **View** e o **Model**, já que estes dois últimos não têm conhecimento um do outro.

A **View** e a **ViewModel** comunicam através de mecanismos de *Data Binding*.

3.2.1 Data Binding

Através do *Data Binding* podemos obter dados que estão na ViewModel apartir da View e vice-versa.

Existem quatro maneiras de fazer binding de dados: *One-Way Data Binding*, *Two-Way Data Binding*, *Interpolation* e *Event Binding*. [2]

Nota: os exemplos apresentados a seguir são exemplos em Angular 2.

One-Way Data Binding

Como o nome indica, está forma de fazer binding é só num sentido, a View vai buscar dados à ViewModel, qualquer mudança de valores do lado da View não irá afetar o valor do lado da ViewModel, mas se o dado for alterado no lado da ViewModel ele irá ser atualizado no lado da View.

A forma de fazer *One-Way Data Binding* em Angular 2 é usar "[ ]"sobre o atributo do elemento em que queremos aplicar,por exemplo:

```
<h1 [innerText]="tituloPagina"></h1>
```

Onde "tituloPagina"é uma variável que está presente na ViewModel.

Interpolation

Com o *Interpolation* conseguimos obter os mesmos resultados como o *One-Way Data Binding*, mas a forma de invocar é diferente, para usarmos o *Interpolation* precisamos de colocar "{{ }}"sobre o nome da variável na qual queremos obter o valor.

```
<h1>{{tituloPagina}}</h1>
```

Neste caso, tal como no caso anterior, irá ser mostrado na View o valor da variável "tituloPagina"no elemento <h1> do HTML.

Two-Way Data Binding

Com *Two-Way Data* podemos fazer binding dos dados nos dois sentidos, isto é, qualquer mudança no valor da variável no lado da ViewModel ela é atualizada no lado da View e vice-versa. Para indicar que queremos fazer um *Two-Way Data Binding*, precisamos de utilizar "[(ngModel)]".

```
<input [(ngModel)]="userName"/>
```

Quando o utilizador digitar valores neste campo "input"da *View*, esses valores estão a ser atualizados na variável "userName"no lado da ViewModel.

Event Data Binding

Esta é a forma de fazer binding quando algum evento ocorre.

```
<button (click)="enviarMensagem()">Enviar</button>
```

Neste exemplo, quando o utilizador clicar no botão a função "enviarMensagem()"vai ser invocada.

3.3 Caliburn.micro

É uma framework para o desenvolvimento de aplicações em plataformas XAML [4] e é um suporte para o desenvolvimento dos padrões MV* [5].

3.4 In-Memory Caching

É o acto de armazenar em memória informação que costuma ser obtida com chamadas à base de dados ou ao servidor, assim conseguimos reduzir o número de chamadas às base de dados/servidores, obtendo a informação diretamente da memória.

3.5 AppDomain no C#

Podemos ver uma *AppDomain* como uma área isolada dentro de um processo. Tudo o que se costuma pensar ser "por programa"(como variáveis estáticas), na verdade elas são "por *AppDomain*", assim sendo, podemos ter por exemplo uma variável estática num processo que por sua vez é estática individualmente em cada uma das *AppDomain* localizadas dentro desse processo.
Um processo pode ter várias *AppDomains* desde que o programador as crie e as corra dentro do processo, pois por defeito, um processo tem uma *AppDomain*.

3.6 Arquitetura de software

Antes do desenvolvimento de um software, onde ainda os seus componentes se encontram separados e sem uma ligação lógica e estruturada, a função da arquitetura de software é fornecer um esboço de como tudo se vai relacionar e interligar, ajudar a quem vai desenvolver o software a entender o comportamento de todo o sistema, sendo assim um apoio para as decisões a tomar sobre o desenvolvimento.
Uma arquitetura de software serve também, quando um projeto já está concluído, mostrar como tudo se relaciona dentro do sistema.

3.7 Servidores Statefull & Servidores Stateless

Eles são o oposto um do outro, vamos imaginar um cenário de um servidor web.
Com **Statefull** nós somos capazes de guardar no lado do servidor alguma informação sobre a sessão e assim identificar o utilizado nas suas várias chamadas ao servidor. Já com o **Stateless** nós não guardamos qualquer tipo de informação sobre a sessão, então é necessário que a cada chamada ao servidor vá alguma informação que ajude a identificar o utilizador.

3.8 Web services

Através dos *web services* é possível a comunicação entre aplicações distintas e facilita a interação entre novas aplicações e as aplicações já existentes. São mecanismos que permitem com que as aplicações possam enviar e receber dados como também compartilhar serviços. Essas aplicações podem ter sido desenvolvidas num contexto computacional diferente (linguagens de programação distintas), onde a comunicação irá dar uso a um formato intermédio tais como: **XML**, **Json**, **CSV**.

[4] é a linguagem usada pela Microsoft para a criação de interfaces de utilizadores
[5] MVVM, MVC, MVP

3.9 Representational State Transfer (REST)

É um estilo de arquitetura que se aplica a *web services* para induzir propriedades desejáveis tais como desempenho e escalabilidade, fazendo com que os serviços funcionem melhor na Web. Na arquitetura REST, que podemos ver como uma arquitetura cliente/servidor e tem como foco o uso de um protocolo de comunicação sem estado , tipicamente HTTP, tanto os dados como as funcionalidades são vistos como recursos e são acedidos usando Uniform Resource Identifiers (URIs), que são basicamente links na Web para identificar recursos. [3]

3.10 Service-Oriented Architecture (SOA)

É uma arquitetura de software que tem como ideia principal de que as funcionalidades das aplicações devem ser disponibilizadas na forma de serviços via *web services*, podendo ser reutilizados e compartilhados por várias aplicações.

3.11 Managed Extensibility Framework (MEF)

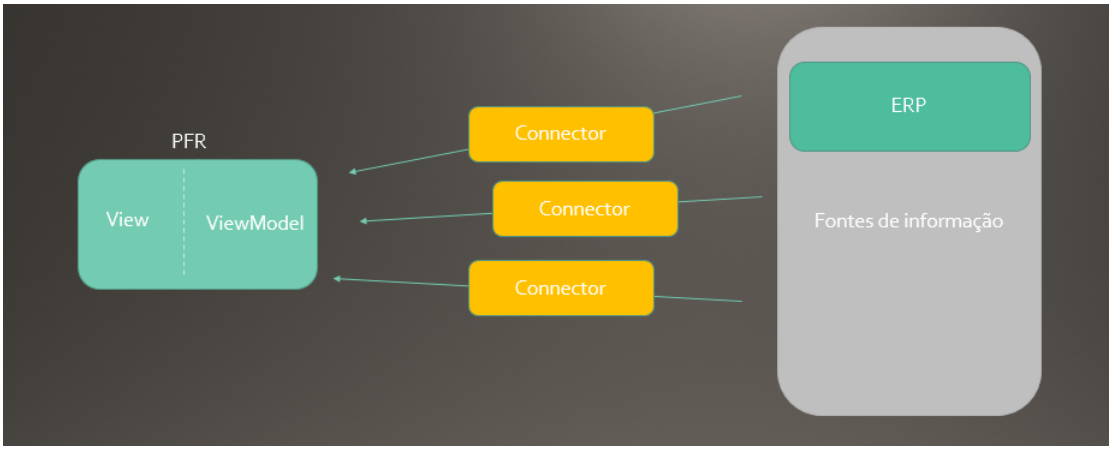
É uma biblioteca para criar aplicações leves e com possibilidade de ser extendidas, facilitando o uso de extensões exernas dentro da aplicação em desenvolvimento.

3.12 T4 text template

T4 text template é um gerador de ficheiros de texto que está presente no Visual Studio, ele utiliza blocos de texto simples misturado com blocos de código [6]. Foi usada esta tecnologia para desenvolver uma aplicação capaz de gerar páginas web a partir dos dados obtidos de um determinado ficheiro XML. Podemos ver na segunda imagem da secção 2 deste documento uma dessas páginas web geradas.

[6] C# ou Visual Basic

4 Arquitetura da atual plataforma



O **Primavera Fiscal Reporting** é uma aplicação independente que pode ter várias fontes de informação, sendo o **ERP** da **Primavera** a fonte de informação mais importante.

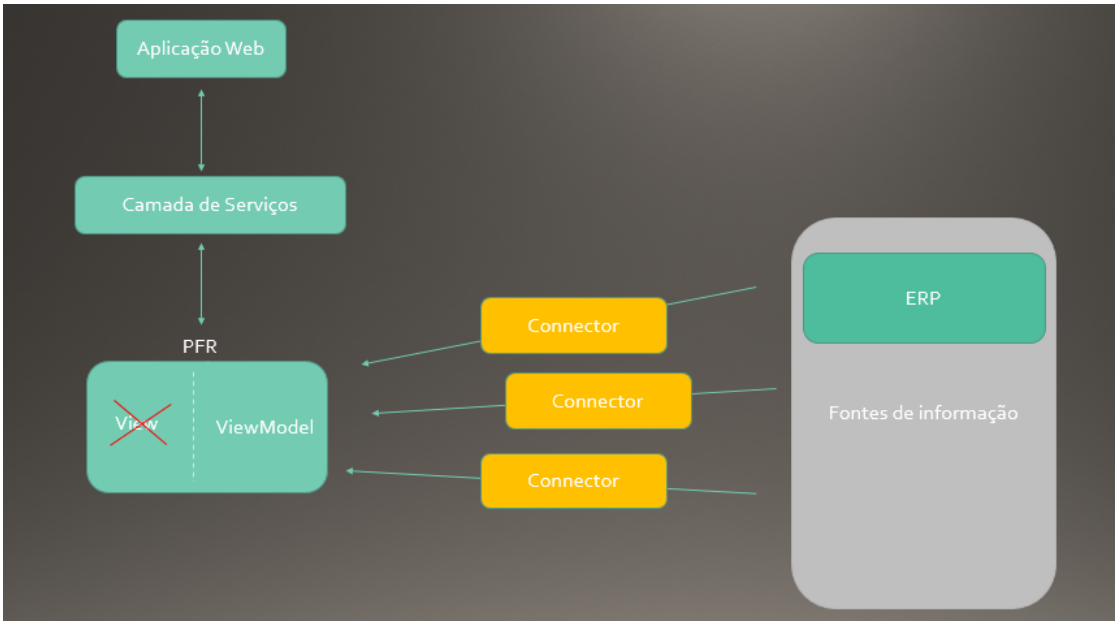
Na arquitetura mostrada na imagem a cima podemos ver que existem duas entidades, o PFR e as fontes de informação, ligadas por um mecanismo chamado de *Connector*. Tanto o PFR como as fontes de informação estão localizados na máquina do utilizador, fazendo assim com que o sistema seja um sistema local, não necessitando de dados externos.
O PFR é uma aplicação desenvolvida sobre o padrão MVVM, onde há então uma separação de responsabilidades, o que é mostrado ao utilizador na $View$ e o que é processado na $ViewModel$.

A $View$, que é a interface da aplicação, neste caso é uma aplicação desktop onde o utilizador pode interagir com a plataforma, ver os modelos associados à sua conta, editar, processar e gravar esses mesmos modelos, entre outras funcionalidades. Cada modelo pode ter vários cenários, isto é, o utilizador pode criar cenários para cada modelo e assim conseguir simular vários tipos de resultados com inputs diferentes para o mesmo modelo.

Por seu lado, a $ViewModel$, **que nesta arquitetura está a representar a lógica de negócio**, é a entidade que tem como função responder aos pedidos feitos pelo utilizador a partir da $View$.

A $ViewModel$ da atual plataforma está desenvolvida para responder aos pedidos de um só cliente por aplicação. Quando um utilizador faz o seu login no Primavera Fiscal Reporting, dentro da lógica de negócio é criado um contexto sobre esse utilizador e todas as funcionalidades a partir desse momento são computadas sobre os dados e informações do mesmo.

5 Arquitetura da nova plataforma



Ao analisar a arquitetura da nova plataforma, podemos já ver algumas diferenças em comparação com a arquitetura anterior. Um dos pontos mais importantes a ter em conta é que as entidades presentes na arquitetura anterior que também estão presentes nesta arquitetura [7], elas deixam de estar no local do utilizador e passam a estar disponíveis na web. A *view* do PFR foi descartada pois ela foi substituída pela aplicação web nesta arquitetura.
Dois novos elementos foram adicionados, a camada de serviços e a aplicação web.

A aplicação web veio substituir a interface da atual plataforma e é o único elemento desta arquitetura que faz parte do lado do cliente. Algumas modificações foram feitas a nível de design para dar uma aparência mais moderna a esta nova interface.

A camada de serviços, que é um *web service* e está desenvolvido sobre a arquitetura Representational State Transfer (REST), tem como função ligar a aplicação web à *viewmodel* (lógica de negócio do PFR) e com isso ser capaz de dar resposta aos pedidos do utilizador via web.

Como vimos na arquitetura anterior, a lógica de negócio está desenvolvida para dar resposta a um só cliente, logo, isto é um problema para o que é pretendido com esta nova plataforma. Relembrando que o que é pretendido é ter um serviço online que seja capaz de dar resposta a vários clientes em simultâneo. Na secção 6.2.2 é mostrado o que se tentou fazer para contornar este problema.

[7] PFR (mais precisamente a sua $ViewModel$) e as fontes de informação

6 Desenvolvimento da plataforma

A nova plataforma fiscal é uma aplicação web desenvolvida sobre a framework Ionic.
Foi também desenvolvida uma camada de serviços em C# onde a aplicação web pode comunicar e obter a resposta aos seus pedidos. Sendo esses pedidos processados pela camada de serviços com o auxílio da lógica de negócio.

O desenvolvimento do projeto foi feito com base na arquitetura mostrada anteriormente a secção 5. O primeiro passo foi criar um design para a página de login juntamente com as suas funcionalidades. Depois dessa etapa, o desenvolvimento da aplicação web e da camada de serviços, juntamente com a análise da lógica de negócio, foram feitas em simultâneo, para que com o desenrolar do projeto não houvesse falhas de execução. Basicamente quando era feito uma funcionalidade para ser usada pelos utilizadores a partir da aplicação web, era de imediato testada para ver se a execução e interligação das 3 componentes [8] não apresentava qualquer tipo de falhas.

6.1 Aplicação Web

6.1.1 Correr a aplicação num dispositivo android

Para correr a aplicação num dispositivo android, o primeiro passo é ligar o dispositivo android ao computador via cabo USB e certificar de que o modo "Depuração USB"está ligado. Depois temos de adicionar uma plataforma ao projeto, isto é, indicar ao Ionic que queremos correr a aplicação numa plataforma android. Para isso precisamos de executar os seguintes comandos no terminal, que já se encontra posicionado na pasta raiz da aplicação: **cordova platform add android**. Para finalmente obtermos a aplicação dentro sistema android que se encontra ligado via USB, basta executar o comando **ionic run android**.

Nota: Existem algumas dependências para que seja possível correr uma aplicação num dispositivo android a partir de um computador. As dependências são: ter **JDK** e **Android SDK** instalados no computador, ter a **variável de sistema "ANDROID_HOME"** com o valor do caminho onde está instalado o **Android SDK**, e por fim, adicionar a **variável de sistema "ANDROID_HOME"** à **variável de sistema "PATH"**.

6.1.2 Erros encontrados & Solução dos erros

- Não é possível encontrar o "gradle"
Depois de instalar o **Android Studio** (que já vinha com o **Android SDK**), ao tentar correr os comandos para executar a aplicação num smartphone Android, apareceu um erro a dizer que não era possível encontrar o "gradle"dentro da pasta do **Android SDK**. A solução foi fazer o download da pasta "tools"do **Android SDK** individualmente do link `https://developer.android.com/studio/index.html` e atualizar a pasta "tools"do **Android SDK** já instalado no computador.

- Application Error - The connection to the server was unsuccessful

[8] aplicação web, camada de serviços e lógica de negócio

A primeira vez que que se correu a aplicação numa máquina Android, a aplicação acabava por se desligar no momento e que ela arrancava, e o erro "Application Error - The connection to the server was unsuccessful."aparecia. Depois de alguma pesquisa acabei por encontrar a solução no link `http://stackoverflow.com/questions/12319809/application-error-the-connection-to-the-server-was-unsuccessful-file-andr`, tudo o que precisamos de fazer para resolver este problema foi adicionar a linha de código

```
<preference name="loadUrlTimeoutValue" value="180000" />
```

no nosso ficheiro "config.xml", essa linha de código o que vai fazer é aumentar o tempo de espera máximo (neste caso para 180000 ms, 3 minutos) que a aplicação tem para que a rede estabeleça uma conexão. Caso nenhuma modificação foi feita relativamente ao tempo de espera, a rede não iria conseguir estabelecer uma conexão dentro do tempo limite e então esse erro era executado pelo sistema e a aplicação era obrigada a fazer *shutdown*.

- FATAL ERROR: CALL_AND_RETRY_LAST Allocation failed - JavaScript heap out of memory
Este erro aparece no momento da compilação quando estamos a tentar processar ficheiros que necessitam de mais memória do que aquela que está a ser fornecida.
Neste caso, o Node Js, por defeito, vem com um limite de memória de 512 mb, como neste projeto estamos a lidar com um grande número de ficheiros, este erro acabou por aparecer. A solução deste erro foi aumentar então o limite de memória para o nosso projeto seguindo os passos que podem ser vistos na referência[4].

6.2 Camada de serviços (Web Service)

É a ponte entre a aplicação web e a lógica de negócio. O objetivo é garantir que a camada de serviços esteja preparada para receber e responder a pedidos de vários clientes em simultâneo.
Foi no desenvolvimento da camada de serviços onde apareceram vários problemas a resolver, estando esses problemas ligados à reutilização da lógica de negócio para os fins pretendidos. A lógica de negócio da atual plataforma, onde a camada de serviços vai buscar resposta ao pedido dos clientes, funciona em grande parte por estruturas estáticas, estando assim desenvolvida para dar resposta a pedidos de um só cliente. Ora, isso é o oposto do que se espera obter. Para que a nossa camada de serviços possa estar funcional no contexto desejável, foi necessário analisar o código da lógica de negócio, apresentar ideias e possíveis soluções para esta situação.

6.2.1 Erros encontrados & Solução dos erros

- Could not load file or assembly
Este erro ocorria quando estávamos a reaproveitar a lógica de negócio e tentar aceder a ela a partir da Camada de Serviços. Este erro está relacionado com a **Strong Name Validation**.
Uma solução para este problema pode ser encontrada em ambas as referências [5] [6].

6.2.2 Reutilização da lógica de negócio da atual plataforma

Alterações feitas na lógica de negócio
Foram descartadas todas as funcionalidades presentes na lógica de negócio que eram dependentes do comportamento da aplicação desktop [9]. Se essas funcionalidades não fossem

[9] tais como: progress bars, clique do rato, etc

descartadas, ocorreriam erros de execução quando o fluxo de computação da lógica de negócio, ao processar uma resposta para o cliente que fez um pedido via aplicação web, verificava que as tais funcionalidades não poderiam ser executadas, pelo facto de o pedido ter sido feito por uma nova plataforma e não pela plataforma desktop, da qual essas funcionalidades dependem.

As classes da lógica de negócio são e estão constituídas por elementos estáticos, se nenhuma alteração fosse feita a este nível, o que aconteceria num contexto real onde há mais do que um utilizador a fazer pedidos, é que todos os utilizadores da nova plataforma iriam listar e editar as informações referentes ao último cliente que efetuou o login na aplicação. Pois, sendo a lógica de negócio constituída por elementos *static*, ela irá criar um contexto para todas as suas funcionalidades relativamente ao último login recebido.
Depois do apoio de alguns colegas e de várias pesquisas, cheguei a conclusão de que a única solução possível para este problema era recorrer às *APPDomain* do *C#*, isto é, criar uma zona isolada dentro do processo em execução para cada cliente, onde cada zona isolada é uma lógica de negócio independente das outras.

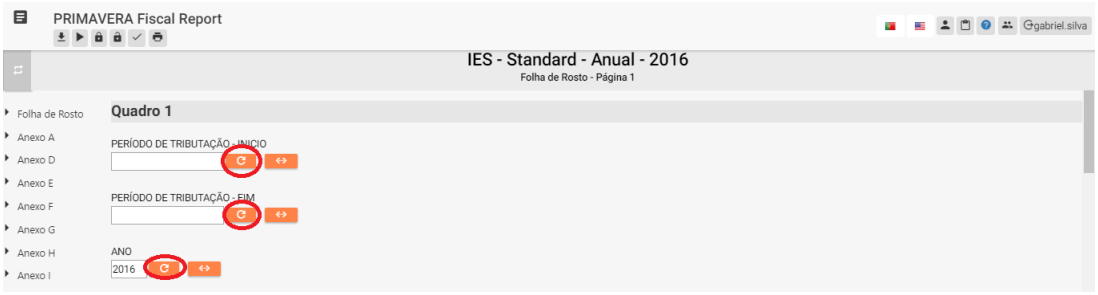
6.3 Novas funcionalidades

6.3.1 Botão para ver e alterar a fórmula do campo

A funcionalidade de alterar a fórmula dos campos em si não é uma nova funcionalidade, em comparação com a atual plataforma, pois a atual plataforma também contém essa funcionalidade, o que foi feito neste projeto foi adicionar um botão para cada campo. Clicando nesse novo botão é possível ver e alterar a fórmula de cada campo do modelo, com isso conseguimos com que o acesso a essas funcionalidades estejam ao dispor dos utilizadores de uma forma mais prática.

6.3.2 Botão para processar individualmente cada campo

Na plataforma atual para processarmos um campo individualmente precisamos de 3 cliques, com a nova plataforma basta um só clique no botão em frente do campo, botão esse que foi adicionado nesta nova plataforma. Nota: esta funcionalidade ainda se depara com algumas falhas por parte da resposta da lógica de negócio.



6.3.3 Terminar sessão por inatividade do cliente

A aplicação está sempre atenta se o utilizador se mantém ativo, caso o utilizador fique inativo durante 20 minutos, a aplicação mostrará um aviso de que a sessão vai terminar, nesse momento o utilizador tem 5 segundos para voltar a estar ativo (mexer o rato, digitar algo no teclado etc) caso contrário a sua sessão será terminada.

6.3.4 Línguas da aplicação

Na nova aplicação podemos escolher em que língua as funcionalidades vão aparecer com apenas um clique no icon da língua desejada (neste momento as línguas disponíveis são: Português-PT e Inglês).

Na atual plataforma precisamos de vários cliques e de reiniciar a plataforma para que as funcionalidades apareçam na língua desejada pelo utilizador.

6.4 Geração de código/Desenho dos modelos de declarações fiscais

A geração de código é feita através de **t4 text templates**, onde temos blocos de código e blocos de texto.
Foi desenvolvido um programa em que ao receber um XML como input ele é capaz de gerar páginas web.

6.4.1 Deserialização

Toda a informação de um modelo de declaração fiscal está guardada num ficheiro XML.
A **deserialização** neste projeto consistiu na conversão da estrutura de dados em XML para uma estrutura de dados em classes C# para que depois, usando o programa desenvolvido que utiliza a tecnologia **t4 text templates**, fosse possível a geração das páginas web referentes a cada modelo. Cada página é representada por um ficheiro ".html", que é o design da página, e um ficheiro ".ts", que contém as funcionalidades da página.

6.5 Resultado final

O objetivo era o estudo da viabilidade da migração para a web da aplicação desktop Primavera Fiscal Reporting reaproveitado a sua lógica de negócio.

Uma parte significativa do trabalho desenvolvido no âmbito deste projeto foi o desenvolvimento de um programa capaz de gerar páginas web correspondentes a diferentes modelos de declarações fiscais existentes no formato XML.

No que toca à interface, verificamos que é totalmente fazível. Com o desenvolvimento da aplicação web verificamos que não há grandes obstáculos para conseguir reproduzir a interface atual para uma interface mais moderna e eficiente. Assim sendo, o objetivo de desenvolver uma aplicação web que substitua a interface atual foi concluído.

A camada de serviços, por seu lado, também se encontra funcional. A ligação entre a aplicação web e a lógica de negócio não se depara com qualquer falha ou problema, com isto podemos afirmar de que é possível migrar a plataforma PFR para a web. Vimos que é possível o desenvolvimento de uma nova interface que se liga via web a um *web service* e de que esse serviço retorna ao cliente uma resposta utilizando as funcionalidades da lógica de negócio.

Mas o resultado final não vai de encontro com as expectativas de que não seria preciso modificar drasticamente a lógica de negócio. Então porquê? Como vimos anteriormente, a lógica de negócio está desenvolvida para dar reposta a um só utilizador (de cada vez), pois quando o utilizador faz o seu login, é criado um contexto em toda a lógica de negócio sobre esse utilizador e a partir daí todas as funcionalidades iram correr sobre dados e informações desse utilizador.
Nesta nova plataforma o que acontecia era o seguinte, temos a lógica de negócio disponível na web em que vários utilizadores podiam aceder em simultâneo, imaginemos que o utilizador "A" faz o seu login, na lógica de negócio é então criado um contexto sobre esse cliente. Perante essa ação, o utilizador pode usufruir de todas as funcionalidades presentes na aplicação web sem qualquer falha ou problema, até aqui tudo perfeito. Quando só há um utilizador a aceder à lógica de negócio concluímos que não existe qualquer tipo de falhas. Os problemas aparecem quando há mais do que um utilizador a aceder. Vamos imaginar o seguinte cenário, temos dois utilizadores, o "A" e o "B", o utilizador "A" faz o login na aplicação, perante esse comando a lógica de negócio cria um contexto sobre esse utilizador, mas nesse mesmo tempo, ou até depois, o utilizador "B" faz o seu login. O que acontece? A lógica de negócio irá construir um contexto sobre o utilizador "B", fazendo com que todas as funcionalidades funcionem sobre os dados e informações do utilizador "B", e assim, o utilizador "A" irá ver e editar as informações do utilizador "B" e não as suas, como seria esperado.

Para tentar resolver esse problema, o que me ocorreu fazer foi criar mecanismos de concorrência em torno das funcionalidades da lógica de negócio, fazendo com que só um cliente de cada vez estivesse dentro das funcionalidades e permitindo só a entrada de novos pedidos vindos de outros utilizadores depois de o pedido atual ter sido respondido. Esta solução teve sucesso para o problema de utilizadores estarem a manipular informações de outros utilizadores, isto é, com mecanismos de concorrência, cada cliente via e editava os seus dados sem interferir com dados de outros. Mas esta solução não tardou em trazer consigo outro problema, quando o número de pedidos crescia, o tempo de espera dos utilizadores para obter as suas respostas podia crescer drasticamente. Como num dia a dia real a nova plataforma iria esperar milhares de pedidos em simultâneo, esta solução mostrou não ser adequada para os fins desejáveis.

Como a solução dos mecanismos de concorrência não era compatível com o desejado, recorri à ajuda de alguns colegas dentro da Primavera BSS que me deram dicas de como uma possível solução podia ser. Fiz algumas pesquisas e deparei-me então com a possível

solução, as *APPDomain* do *C#*.
Como já foi dito na secção 6.2.2, esta foi a última solução a ser testada para resolver o problema de não conseguir dar resposta a vários clientes em simultâneo.

Foi desenvolvido um programa onde simulávamos um cenário de vários utilizadores a fazerem pedidos ao *web service*, que por sua vez iria responder a esses pedidos recorrendo às capacidades das *APPDomain*. Ao contrário da solução dos mecanismos de concorrência, com as *APPDomain* do *C#* os pedidos dos utilizadores eram processados em paralelo, não sendo necessário que um pedido fique a espera de outro pedido terminar. Isso acontece porque cada pedido está a ser processado por uma *APPDomain* diferente, onde em cada uma delas é criada uma lógica de negócio independente das outras. Os resultados obtidos eram corretos, cada cliente editava as suas próprias informações e dados sem interferir com as dos outros utilizadores. Mas esta solução também trouxe complicações, ao criar uma lógica de negócio para cada *APPDomain*, fazia com que o servidor gastasse muita memória e tempo de processamento, pois uma lógica de negócio do género da que o PFR possui é bastante grande e pesada computacionalmente. Foram feitos vários testes e reparamos que por exemplo para 30 utilizadores em simultâneo, ou seja, 30 *APPDomain* dentro do processo em execução, era necessário de mais ou menos 1 GB de memória e alguns desses 30 utilizadores obtinham resposta ao seu pedido em mais de 2 minutos. Estes dados vão contra o que é desejado pela Primavera BSS, pois não é uma solução eficiente tanto a nível de esforço computacional por parte dos servidores como não é eficiente ao dar resposta aos pedidos dos clientes.

Perante estas informações obtidas com o desenvolvimento e testes da nova aplicação, ficou mostrado de que, sim, é possível migrar o PFR para a web, mas para que o serviço seja eficiente é necessário reformular toda a sua lógica de negócio.

7 Conclusões

No inicio deste ano letivo tive de escolher o tema que mais me agradaria de desenvolver como tese, dentro de todas as hipóteses, o tema de desenvolver uma nova plataforma fiscal da Primavera BSS foi o que me chamou mais à atenção, pois é um tema motivador visto que se trata de fazer algo novo, uma evolução de um projeto que já está desenvolvido e no mercado. Essa evolução trata-se de "transformar"uma aplicação desktop, onde todas as funcionalidades encontram-se em um lugar fixo, numa aplicação web, onde as funcionalidades encontram-se na *cloud*.

A forma como este projeto foi desenvolvido foi bastante compatível com aquilo que eu imaginava que iria ser. A única novidade era de como seria o ambiente dentro da empresa e de como eu me iria adaptar a esse ambiente. Julgo que não poderia ter sido recebido da melhor forma, o apoio sempre foi constante por parte dos colegas, que sempre tinham disponibilidade para me ajudar em algumas dúvidas e obstáculos. Tive também o privilégio de dar dicas, partilhar o meu conhecimento e ajudar colegas em algumas situações.
Sinto que evoluí bastante a nível de trabalho de equipa, pois pude assistir como realmente é um trabalho dentro de uma empresa e de como cada pessoa tem um papel importante no dia a dia de projetos de grande dimensão.
Desde o primeiro dia do estágio senti que o ambiente de trabalho na Primavera BSS seria uma motivação extra para mim, acho que ambientes de trabalho como este que me foi oferecido neste estágio são uma mais valia. Quando as pessoas se sentem bem no seu lugar de trabalho a sua motivação cresce, e quanto maior a motivação, maior são as probabilidades de novas ideias e melhor desempenho.
Com este estágio e com o desenrolar do projeto, adquiri mais capacidades de comunicação, tanto verbal como escrita, como também mais conhecimentos técnicos e teóricos.

A aprendizagem obtida nas unidades curriculares foram sem dúvida muito importantes para o desenvolvimento deste projeto, como a noção de "*multithread*", "*cloud computing*", programação orientada a objetos, entre outras, foram os principais pilares técnicos e teóricos que me ajudaram a desenvolver este projeto.
O meu *background* universitário vem da área das matemáticas e ciências da computação, talvez por alto a matemática pareça descartada de todo o desenvolvimento do projeto, mas na verdade é que não.
O pensamento lógico, aberto e ao mesmo tempo cauteloso obtido nas unidades curriculares na área puramente matemática, foram elas também extremamente importantes.
Um dos objetivos que pode surgir para este projeto no futuro é a analise de segurança da aplicação, mais uma vez a matemática vai aparecer aqui num lugar de destaque, visto que segurança no mundo da informática/computação, ao nível de comunicações entre cliente e servidor, nada mais é do que procedimentos matemáticos, tais como: RSA, El Gamal e NTRU.

Como vimos na secção "Resultado final", conseguimos uma migração do PFR para a web mas essa migração não é eficiente da forma como se pensava que poderia ser. A reformulação da lógica de negócio da atual plataforma é inevitável para que a migração do PFR seja eficiente e vá de encontro com o desejado pela Primavera BSS. Essa tarefa de reformulação vai requerer bastante trabalho e cuidados. Com as análises e testes que fiz com a lógica de negócio, reparei que existem muitas dependências dessas tais estruturas estáticas que incorporam a lógica de negócio, ou seja, ao reformular essas estruturas

estáticas, será necessário estruturar de novo também tudo que depende dessas estruturas ao longo do fluxo computacional. Caso essa reformulação ao código da lógica de negócio seja feita mas a estrutura da atual plataforma continue a mesma, a plataforma desenvolvida neste projeto será 100% compatível com essas mudanças, na minha opinião, só em caso de alterações de nomes de alguns dos métodos dentro da lógica de negócio é que será necessário uma pequena intervenção no código desenvolvido neste projeto.

Perante tudo o que já foi dito neste documento, podemos concluir que os objetivos principais deste estágio foram alcançados. Este projeto deu à Primavera BSS um exemplo de como pode ser a sua "nova plataforma fiscal"e mostrou o que terá de ser mudado para que os seus objetivos de migrar o PFR para a web sejam realizados. Já a mim, este projeto deu-me a oportunidade de me integrar num ambiente profissional excelente e de desenvolver algo do meu interesse profissional e pessoal , onde pude aprender e conhecer novos mundos.

Referências

[1] http://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel mvvm/18411.

[2] https://www.themarketingtechnologist.co/introduction-to-data-binding-in-angular-2-versus-angular 1/.

[3] http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html.

[4] https://www.npmjs.com/package/increase-memory limit.

[5] http://stackoverflow.com/questions/12100006/sgen-error-could-not-load-file-or-assembly-exception-from-hresult 0x80131413009177.

[6] https://brianweet.github.io/2016/01/22/disable-strong signing.html.