UNIVERSIDADE
LUSÓFONA

App4SHM

Trabalho Final de curso

Relatório Intercalar 2º Semestre

Nome do Aluno: Jorge Lobão

Nome do Orientador: Pedro Alves

Trabalho Final de Curso | LEI | 20/01/2021

Direitos de cópia

Resumo

Este ano vou continuar o desenvolvimento e melhoramento da aplicação elaborada pelos meus colegas do ano passado. A aplicação App4SHM tem como objetivo a assistir na análise e monitorização da integridade de estruturas, pois estes fatores podem apresentar riscos de segurança para a humanidade.

A aplicação recorre ao acelerómetro integrado de um smartphone para medir a integridade estrutural através das vibrações e estes dados depois são processados por algoritmos de Machine Learning no servidor. Um engenheiro civil poderia usar a aplicação para obter dados de estruturas em investigação e indicar se a estrutura se encontra estável, se necessita de reparações ou se está à beira de colapso.

A aplicação tem revelado falhas como problemas de desempenho no servidor, o facto de não poder gravar os dados obtidos no telemóvel quando não estiver ligado à rede e principalmente, a aplicação só está disponível para smartphones com Android.

O TFC apresentado neste relatório pretende resolver estes problemas.

Abstract

This year I will continue developing and improving the application developed by my colleagues last year. The application App4SHM aims to assist in the monitorization and analysis of structural integrity, as these factors can pose security risks to humanity.

The application uses a smartphone's integrated accelerometer to measure structural integrity through vibrations, the data collected is then processed by Machine Learning algorithms on the server. A civil engineer can then use this application to gather data from structures under study and conclude if the structure is stable, needs repairs or is on the verge of collapse.

The application still has its faults like slow server response time, the server breaking with excess information, the fact that you can't save the data when not connected to the network, and mainly the application is only available on Android devices.

The project presented in this report intends to solve these issues.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Extratos de Código Fonte

1 Identificação do Problema

A plataforma App4SHM (Application for Structural Health Monitoring) que vou melhorar vai permitir uma fácil monitorização de estruturas sem a necessidade de ferramentas extra, reduzindo assim custos de operação.

Apesar desta aplicação ser revolucionária no âmbito de Engenharia Civil, a mesma tem vários problemas que foram identificados durante o seu uso. Um exemplo é um professor de Engenharia Civil na ULHT, que estava a fazer um estudo de uma ponte no Brasil, necessitar de fazer várias análises com a aplicação mas como a resposta do servidor estava lento e a rede na zona onde estava era fraca, limitou-se a apontar os dados num papel. Caso o servidor estivesse com uma resposta rápida ou o professor pudesse gravar os dados no móvel, poderia ter feito mais análises à ponte.

Outro problema identificado foi o facto de a aplicação não estar disponível para smartphones iOS. De acordo com o StatCounter [STAT21], existe desde 2020, um bilhão de pessoas no mundo a usar dispositivos iOS, isto corresponde a 28.21% da população mundial. Há alunos e professores de cursos de Engenharia Civil que gostariam de utilizar a aplicação para investigações mas estão limitados porque usam dispositivos iOS.

O crescimento exponencial de utilizadores de iPhone está demonstrada na Figura 1.

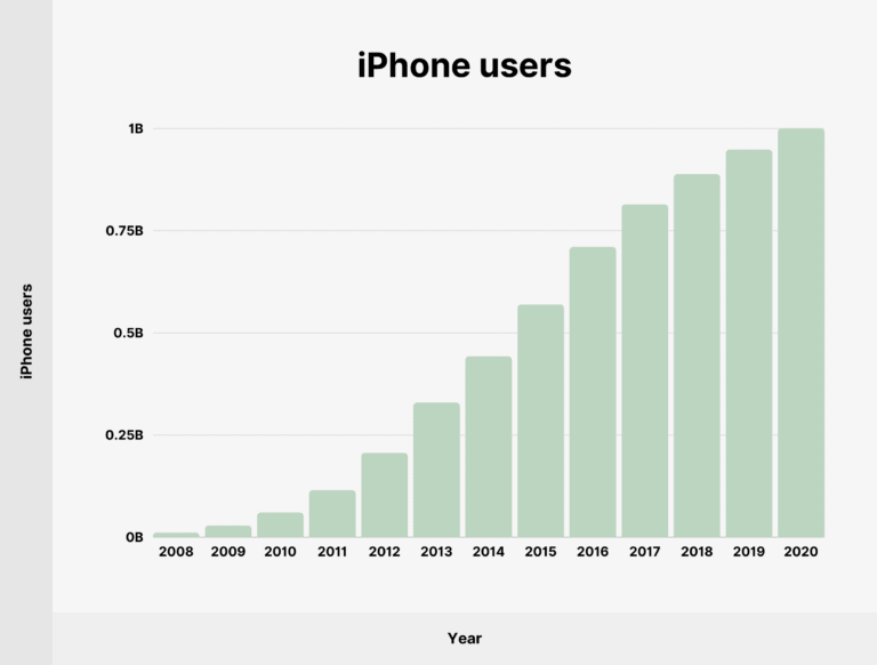

Figura 1 - Número de utilizadores de iPhone desde 2008 até 2020

Neste momento, a aplicação App4SHM já se encontra numa versão 1.1 onde os problemas de desempenho do servidor já foram resolvidos com o novo servidor em Django.

2 Viabilidade e Pertinência

2.1 Viabilidade

A aplicação em questão avançou para a versão 1.1 com os melhoramentos necessários. A aplicação está a ser usada pelo Departamento de Engenharia Civil para elaborar investigações científicas e têm tido muito sucesso em fazer comparações entre um acelerômetro usado profissionalmente e um acelerómetro do telemóvel. Podemos ver que a percentagem de erro entre ambos os aparelhos é semelhante, o que significa que a aplicação tem futuro.

O servidor novo em Django melhorou o tempo de resposta do envio e receção dos dados tal como esperado.

Para a aplicação iOS é necessário a implementação das funcionalidades existentes e melhoramentos da aplicação Android. A aplicação sofrerá mudanças durante o TFC.

Na versão 1.1, a aplicação tem todas as funcionalidades que apresenta na versão 1.0 com alguns melhoramentos tal como o zoom no passo 2.

2.1.1 Custos

As ferramentas que vamos utilizar para fazer este projeto serão todas gratuitas:

1 (IDE) Android Studio [ANST21]

2 (IDE) Xcode [XCOD21]

3 (IDE) PyCharm [PYCH21]

Os dados trabalhados vão ser armazenados no servidor que neste momento está em uso, ou seja, o servidor localizado na faculdade.

Com a adição de uma versão iOS, o maior custo vai ser a manutenção da aplicação e do servidor.

Se a aplicação é publicada nas lojas dos dispositivos mobile (Google Play e App Store), é necessário comprar uma developer license (25 euros para Android e 99 euros para iOS).

A aplicação está disponível na Google Play Store em modo de teste onde as pessoas com link podem fazer download.

2.2 Pertinência

Este trabalho será realizado no âmbito de um estudo a ser realizado pelo departamento de Engenharia Civil da Faculdade de Engenharia da ULHT em específico pelos Prof. Elói Figueiredo e Prof. Ionut Moldovan com o objetivo de produzir uma solução de fácil acesso para auxiliar na monitorização do SHM.

A Figura 2 mostra uma experiência efetuada pelo departamento de Engenharia Civil onde estão a comparar as leituras feitas pela aplicação App4SHM e um acelerómetro profissional.

Figura 2 – Experiencia feita no laboratório do departamento de Engenharia Civil

Estes trabalhos estão enquadrados no âmbito do Civil Research Group (http://civilresearchgroup.ulusofona.pt).

Este projeto candidatou-se ao programa Fazer+, mas não foi adjudicado financiamento pela ULHT. O financiamento é para investir em acelerómetros profissionais (que são caros) e outros equipamentos necessários para o estudo para poder ver a percentagem de erro entre a leitura do acelerómetro do telemóvel e a leitura de um acelerómetro profissional.

Este projeto captou o interesse de Stanford onde os professores iam apresentar a aplicação num seminário na universidade de Stanford. A aplicação também ia ser apresentada num segmento científico na televisão chamada “60 segundos”.

3 Levantamento e Análise dos requisitos

Como referido anteriormente, este projeto tem como objetivo melhorar a aplicação App4SHM. Assim, os professores de Engenharia Civil apresentaram uma lista de melhorias que são necessárias para que a aplicação atinja um nível de usabilidade alto para os seus utilizadores.

3.1 Versão 1.1 da aplicação App4SHM (Versão Android)

Para a versão 1.1, os seguintes updates têm de ser implementados:

- 1.1 - Geração de ficheiros de dados locais (séries temporais e espetro de resposta)
- 1.2 - Criação de um novo servidor Django para a aplicação
- 1.3 – Associar, no servidor, um timestamp (data e hora) às observações realizadas
- 1.4 - Compressão dos dados no telemóvel e descompressão dos dados no servidor
- 1.5 - Melhoramento da acessibilidade do servidor para manutenção da base de dados - O utilizador pode apagar estruturas existentes, apagar ou acrescentar frequências
- 1.6 - Correção dos saltos nos timestamps
- 1.7 - Alteração do tempo na abcissa (milissegundos para segundos)
- 1.8 - Respostas temporais apenas do eixo z
- 1.9 - Permitir o zoom do eixo das abcissas sem alterar o eixo das ordenadas
- 1.10 - Representar o eixo vertical em escala logarítmica
- 1.11 - Alterar o label do eixo das abcissas
- 1.12 - Apresentar um protótipo funcional

De seguida, vou explicar cada requisito da versão 1.1 da aplicação.

Para o requisito 1.1, o utilizador quer descarregar um ficheiro local para observar os dados num ficheiro de texto ou Excel.

Para o requisito 1.2, tem que se criar um novo servidor em Django com todas as funcionalidade do servidor original e as adicionais e estabelecer a ligação à aplicação.

Para o requisito 1.3, para cada observação realizada, tem que se associar um timestamp para marcar que dia e hora é que se realizou a observação. Com este requisito, é mais fácil encontrar uma observação em estudo específico para o utilizador.

Para o requisito 1.4, a aplicação tem que comprimir os dados que vai enviar ao servidor e o servidor tem que descomprimir os dados recebidos. Com este requisito, esperamos que o tempo de resposta do servidor seja mais rápido, resolvendo um dos problemas indicado na identificação do problema.

Para o requisito 1.5, no servidor, o utilizador pode apagar estruturas (leituras) existentes e apagar ou acrescentar frequências em cada estrutura. Com o servidor Django, o utilizador que tem autorização pode manipular os dados presente na base de dados do servidor, ou seja, pode apagar ou acrescentar estruturas através do painel de Django admin.

Para o requisito 1.6, a correção dos saltos dos timestamps tem que ser feita na aplicação. Isto é, o problema que existe neste momento é que para cada frequência numa leitura, o tempo não é constante.

Para o requisito 1.7, a alteração de milissegundos para segundos é feita no gráfico da aplicação. Esta funcionalidade facilita os cálculos matemáticos que o utilizador terá de fazer para o estudo de uma estrutura.

Para o requisito 1.8, o eixo z é as vibrações detectadas pelo acelerómetro com várias frequências, ou seja, o eixo x e y, de acordo com os professores de Engenharia Civil, não são necessárias para o estudo da estabilidade de uma estrutura.

Para o requisito 1.9, o zoom do eixo das abcissas facilita a leitura das frequências presentes no gráfico e ver as correspondências entre o eixo das abcissas e das ordenadas.

Para o requisito 1.10, a escala logarítmica para o estudo de deteção de danos numa estrutura é muito útil de acordo com os professores de Engenharia Civil. A implementação deste requisito facilitava muito o estudo de estruturas.

Para o requisito 1.11, o label do eixo das abcissas necessita ser alterado. A alteração é feita nos gráficos da aplicação.

Para o requisito 1.12, é necessário um protótipo funcional para mostrar as funcionalidades da aplicação e como seria uma boa alternativa a acelerômetros profissionais numa apresentação da aplicação em Stanford.

3.2 Versão 2.0 da aplicação App4SHM (Versão iOS)

Para a versão 2.0, vai-se implementar os seguintes requisitos:

- 2.1 - Desenvolvimento da aplicação iOS
- 2.2 - Permitir que o utilizador possa escolher mais do que três frequências para estudar
- 2.3 - Permitir a inserção de NaN para frequências que não se conseguem reconhecer no espectro de Welch
- 2.4 - Upload de observações diretamente no servidor
- 2.5 – Descarregamento de forma simples dos dados no servidor
- 2.6 – Adição da temperatura ambiente como observação opcional
- 2.7 - Aplicar os algoritmos diretamente na aplicação do telemóvel

Finalmente, vou explicar os requisitos da versão 2.0 da aplicação.

Para o requisito 2.1, como estipulado na Identificação do Problema, é necessário ter uma versão iOS da aplicação. Este requisito é o requisito que demora mais a implementar pois é necessário fazer uma aplicação de raiz.

Para o requisito 2.2, a escolha de mais de três frequências permite que o estudo das estruturas seja mais preciso pois existem mais dados para estudo.

Para o requisito 2.3, a utilização do NaN (Not a Number) para frequências que não sejam reconhecidas no espectro de Welch ajuda a facilitar a leitura das frequências pois se uma frequência é NaN significa que não se aplica ao estudo.

Para o requisito 2.4, o upload de observações já feitas para o servidor para o processamento dos dados permite a diminuição da perda de informação (por exemplo, se o servidor for limpo de observações) pois o utilizador pode simplesmente enviar os ficheiros perdidos para o servidor se os tiver descarregue (interligado com o requisito 2.5 e 1.1).

Para o requisito 2.5, o utilizador pode descarregar os dados de uma observação para o seu dispositivo em ficheiro de texto ou Excel. Esta funcionalidade ajuda diminuir a perda de dados caso aconteça algum erro no servidor.

Para o requisito 2.6, a temperatura ambiente como observação opcional ajuda a aprofundar o estudo de uma estrutura pois a temperatura afeta a saúde de uma estrutura.

Para o requisito 2.7, no caso que os dados não podem ser enviados para o servidor ou não há ligação com o servidor, pode-se aplicar os algoritmos matemáticos diretamente na aplicação do telemóvel.

4 Solução Desenvolvida

Esta aplicação móvel tem o objetivo de analisar e investigar a integridade das estruturas. As melhorias já estão implementadas no lado da aplicação Android e no lado do servidor em Django. A aplicação iOS está parcialmente concluida.

4.1 Arquitetura de Software

A aplicação App4SHM é composta por duas componentes: a aplicação mobile e o servidor. A aplicação mobile tem quatro páginas correspondente aos quatro passos para a deteção de danos numa estrutura:

- Passo 1 – Identificação de Estruturas – Nesta página, o utilizador pode nomear na aplicação a estrutura em estudo e começar a sua análise.
- Passo 2 – Aquisição de Dados – Nesta página, o utilizador pode medir as vibrações de uma estrutura com uma série de aceleração(m/s^2)/tempo(ms) durante um período controlado pelo utilizador. As vibrações são representadas graficamente.
- Passo 3 – Captura de Frequências – O utilizador pode escolher três frequências de uma série de densidade espectral do acelerómetro (método de Welch) e guarda os dados selecionados num vetor de recursos para enviar para o servidor.
- Passo 4 – Deteção de Danos – Para cada observação, calcula-se um indicador de danos baseado na distância de Mahalanobis. O ponto do gráfico pode ser verde ou vermelho para indicar se a estrutura não está danificada ou está danificada, respetivamente.

A aplicação está ligado a um servidor em Flask através de serviços web REST para obter estruturas existentes em estudo, fazer cálculos e guardar as frequências numa base de dados MYSQL. Existe também uma aplicação web onde os utilizadores podem fazer upload de observações feitas para o servidor ou descarregar dados já analisados pelo servidor [ASHM22].

A Figura 3 mostra a arquitetura do software.

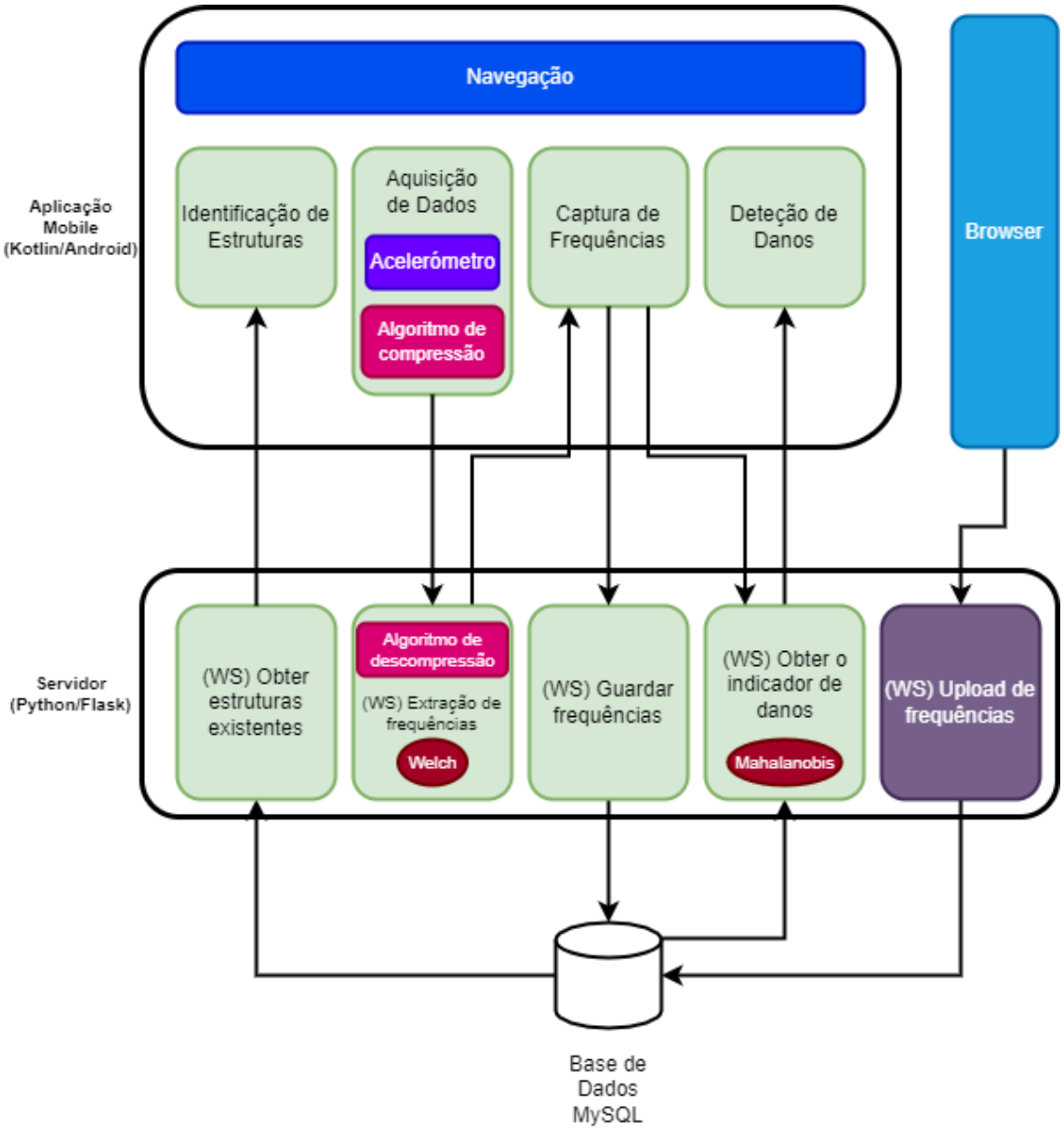

Figura 3 - Esquema da arquitetura do software

4.2 Servidor em Django

A aplicação móvel desenvolvida tem um servidor escrito com o Python framework Flask e a aplicação móvel em si foi feita no Android Studio. A tecnologia usada vai ser alterada consoante a necessidade no decorrer do TFC.

Flask é um framework de Python que é usado no desenvolvimento de web apps básicos. De acordo com o hackr.io [HACK21], existem muitas vantagens para substituir Flask por Django, uma delas é o facto de Django ser um full-stack web framework enquanto que Flask é um light stack framework.

Django é um web framework de Python com muitas funcionalidades:

- É versátil e pode ser utilizado com websites com conteúdo em diversos formatos (por exemplo, JSON);
- É seguro e gere automaticamente recursos de segurança como a gestão das contas dos utilizadores;
- É escalável e sustentável. Django segue padrões e princípios de reutilizar e sustentar código. Segue o princípio de não se repetir código para evitar duplicações. Cada camada da aplicação web é independente, logo fazer com que a aplicação tenha escalabilidade.

Nesta Tabela 1 está demonstrada as principais diferenças entre o Django e o Flask de acordo com o blog AppVenturez [APPV21]

Tabela 1 - Principais diferenças entre Django e Flask

Django	Flask
Repleto de recursos, com uma interface para base de dados, estrutura de diretórios da aplicação, ORM, painel de administração	É leve, flexível e simples
A framework está presente no mercado com uma maior comunidade ativa	Suporta uma estrutura personalizável
Economiza tempo dado que fornece um mecanismo de templates integado	Indicado para pequenos projetos
Não precisa de bibliotecas ou ferramentas externas	O processo de aprendizagem é simples
Está bem segura	Oferece margem aos developers de efetuarem experiências
Indicada para projetos de grande e média escala	

Decidimos fazer esta mudança pois o Django é uma framework mais apto para este projeto devido à uma maior garantia de estabilidade e desempenho para projetos de grande ou médio porte enquanto que o Flask é mais apropriado para protótipos.

Com esta alteração, vou também adicionar a funcionalidade do upload do histórico de leitura. Com isto, o utilizador pode fazer upload no servidor um histórico de leituras, desde que respeite o formato de Excel definido, que pode, por exemplo, utilizar para analisar uma investigação de uma ponte que está danificada e as leituras foram feitas por um acelerómetro profissional. A aplicação vai ter esta diversificação de funcionalidades.

Para estabelecer a ligação entre a aplicação e o servidor, decidimos utilizar REST APIs para estabelecer as ligações às páginas. Um REST (Representational State Transfer) é uma arquitetura que facilita a comunicação entre sistemas na web. O cliente (App4SHM) faz um pedido ao servidor (Servidor em Django) para receber ou alterar dados [REST22]. Em Computação Distribuída, aprendemos a estabelecer ligações REST com Web Services (RESTful Web Services)

através de REST APIs. APIs (Application Programming Interface) são protocolos para construir e integrar software de uma aplicação, ou seja, se quisermos interagir com um computador ou sistema para obter informação ou fazer uma função, um API ajuda-nos a comunicar com o sistema para o sistema perceber o nosso pedido [APIS20].

Os APIs implementados no servidor Django ajudam a interpretar o pedido das páginas da aplicação App4SHM para realizar uma certa tarefa, por exemplo, receber os dados de uma leitura e armazena-las.

Com a implementação do servidor em Django, houve várias melhorias perante a flexibilidade comparada ao servidor antigo. Com o painel de admin do Django, o administrador do servidor consegue adicionar ou remover utilizadores consoante necessário que podem manipular os dados do servidor. Neste caso, os utilizadores principais são os engenheiros de Engenharia Civil e a nossa equipa técnica de Engenharia Informática.

Os dados recebidos estão separados por 4 componentes: As estruturas criadas para estudo, as séries temporais com todos os dados gravados dentro de um espaço de tempo, as frequências naturais para estudo da estrutura e os espectros de energia também usados para estudo da estabilidade da estrutura.

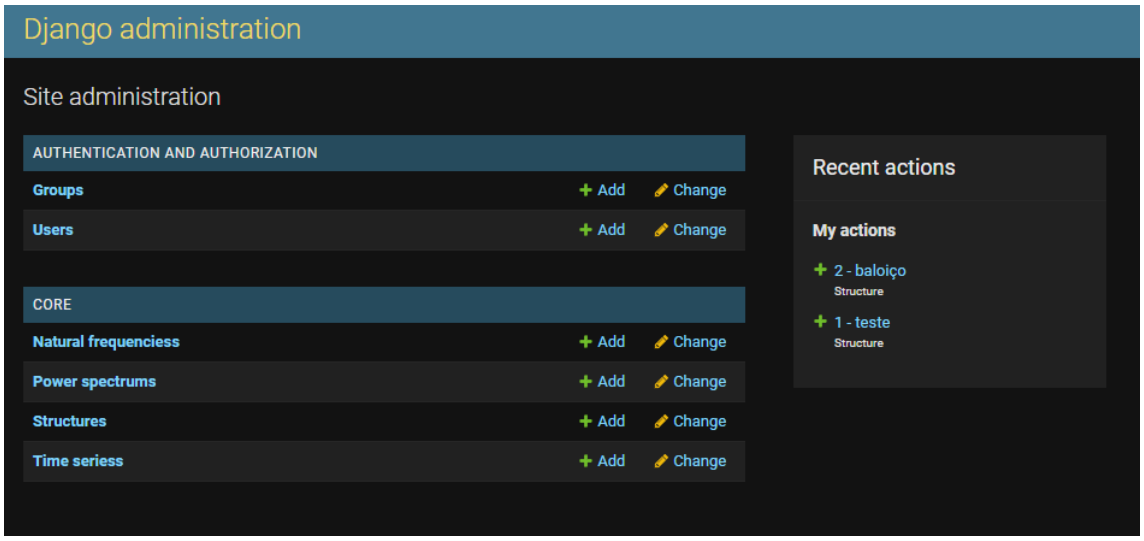

Figura 4 - Painel de admin do servidor em Django

Foram necessárias várias alterações à aplicação principal para conseguir funcionar com o servidor novo. Uma das maiores alterações foi como o ficheiro era enviado para o servidor. Como referido anteriormente no requisito 1.1 , os dados são exportados para um ficheiro csv e para aumentar a velocidade de envio para o servidor, decidimos mandar esse ficheiro diretamente para o servidor em formato JSON.

O ficheiro quando chega ao servidor é desvendado para calcular as frequências de Welch e são enviados para a aplicação de novo. Para ler as frequências de Welch, foi necessário alterar a forma como a aplicação recebia os dados do servidor. Logo, a alteração que fizemos foi alterar a forma de leitura dos dados em que cada objeto JSON correspondia a um array JSON, por

exemplo, as frequências e eram inseridos para dentro de um array novo para ser utilizado nos cálculos.

```
val file = writeToFile( fileName: "readings-${InfoSingleton.structureId}-${date}.csv", readingsText)
Handler(Looper.getMainLooper()).post {
    Toast.makeText(
        App4SHMApplication.context,
        text: "Saved readings-${InfoSingleton.structureId}-${date}.csv",
        Toast.LENGTH_SHORT).show()
}
```

Código 1 - Escrever os dados para um ficheiro CSV

```
val formBody: RequestBody = MultipartBody.Builder()
    .setType(MultipartBody.FORM)
    .addFormDataPart(
        name: "raw_file", filename: "readings-${InfoSingleton.structureId}-${date}",
        RequestBody.create("text/plain".toMediaType(), file)
    )
    .addFormDataPart( name: "structure", InfoSingleton.structureId.toString())
    .build()
```

Código 2 - O ficheiro é convertido para JSON

```
val request: Request =
    Request.Builder().url( url: "${InfoSingleton.IP}api/readings/").post(formBody).build()
```

Código 3 - O pedido para o ficheiro ser enviado para o servidor

```
def calculate_welch_frequencies(lines):

    data_stream = []
    for line in lines:
        if len(line.strip()) > 0:
            parts = str(line.rstrip()).split(";")
            timestamp = int(parts[0])
            x = float(parts[1])
            y = float(parts[2])
            z = float(parts[3])
            reading = ReadingDTO(timestamp, x, y, z)
            data_stream.append(reading)

    interpolated = interpolate_data_stream(data_stream)

    time_array = []
    x_array = []
    y_array = []
    z_array = []
    for i in interpolated:
        time_array.append(i.timestamp)
        x_array.append(i.x)
        y_array.append(i.y)
        z_array.append(i.z)
    welch_x_f, welch_x_pxx = calculate_welch_from_array(time_array, x_array)
    welch_y_f, welch_y_pxx = calculate_welch_from_array(time_array, y_array)
    welch_z_f, welch_z_pxx = calculate_welch_from_array(time_array, z_array)

    return welch_x_f.tolist(), welch_x_pxx.tolist(), welch_y_pxx.tolist(), welch_z_pxx.tolist()
```

Código 4 - Calcular as frequências de Welch

```
val responseObj = JSONObject(response!!.body!!.string())
readingId = responseObj.getInt( name: "id")
val meansArray = responseObj["mean"] as JSONArray
for (i in 0 until meansArray.length()) {
    meanLocal.add(meansArray.getDouble(i))
}
val freqArray = responseObj["frequencies"] as JSONArray
for (i in 0 until freqArray.length()) {
    welch_f.add(freqArray.getDouble(i))
}
val xArray = responseObj["x"] as JSONArray
for (i in 0 until xArray.length()) {
    welch_x.add(xArray.getDouble(i))
}
val yArray = responseObj["y"] as JSONArray
for (i in 0 until yArray.length()) {
    welch_y.add(yArray.getDouble(i))
}
val zArray = responseObj["z"] as JSONArray
for (i in 0 until zArray.length()) {
    welch_z.add(zArray.getDouble(i))
}
```

Código 5 - Importação dos dados do JSON com as frequências de Welch

Houve mais um problema necessário resolver, a importação dos dados do servidor antigo para o novo servidor. Para resolver este problema, foi necessário criar um script que lia o ficheiro csv que era um ficheiro Excel fornecido pelo Departamento de Engenharia Civil e importava os dados para o novo servidor após a leitura do ficheiro.

```
# Baloico
s1 = Structure(name="Baloico")
s1.save()
```

Código 6 - Criação de uma estrutura no servidor

```
with open('./scripts/app4shm_db.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 1
    next(csv_reader)
    next(csv_reader)
```

Código 7 - Leitura do ficheiro em formato CSV

```
for row in csv_reader:

    date_format = datetime.strptime('{} {}'.format(row[1], row[2]), '%Y-%m-%d %H:%M:%S')

    if row[6] == 'Baloico':
        t1 = TimeSeries(structure=s1)
        t1.save()
        t1.date = date_format
        t1.save()
        f1 = NaturalFrequencies(reading=t1, structure=s1, frequencies=[row[3], row[4], row[5]])
        f1.save()
        print(t1.date)
```

Código 8 - Importação dos dados da estrutura em questão para o servidor

4.3 Versão iOS

Outra característica que foi adicionada ao trabalho desenvolvido é a disponibilização do App4SHM para dispositivos com iOS.

De acordo com o MLSDev [MDEV21], existem quatro passos fundamentais a ter em conta quando converter uma aplicação Android para iOS. O primeiro passo é rever os requerimentos e as funcionalidades da aplicação. Neste passo, verifica-se a otimização da aplicação e que funcionalidades são necessárias para que a aplicação seja simples para o utilizador. Aplica-se aqui também a análise da lógica empresarial da aplicação, ou por outras palavras, analisar a procura e exigência da aplicação no mercado. No passo seguinte, aplica-se conhecimento da área de Interação Máquina-Humana, ou seja, a parte de UI e UX da aplicação. No terceiro passo, vem a parte de alteração do código fonte e das estruturas de arquitetura. No passo final, testar ambas as aplicações por erros e verificar se cada aplicação tem todas as funcionalidades implementadas.

No decorrer deste processo, mostro os meus conhecimentos na área de Engenharia de Software, Linguagens de Programação II e Computação Móvel.

A ferramenta que usei para desenvolver a versão iOS desta aplicação é o Flutter. Flutter é um open source framework que é utilizado para criar aplicações multiplataforma de uma forma eficaz a partir de uma base de código. A linguagem utilizada neste framework é Dart desenvolvida pela Google. A ideia era substituir a aplicação desenvolvida em Android nativo para uma aplicação que funciona em iOS e Android.

4.3.1 Implementação da versão iOS

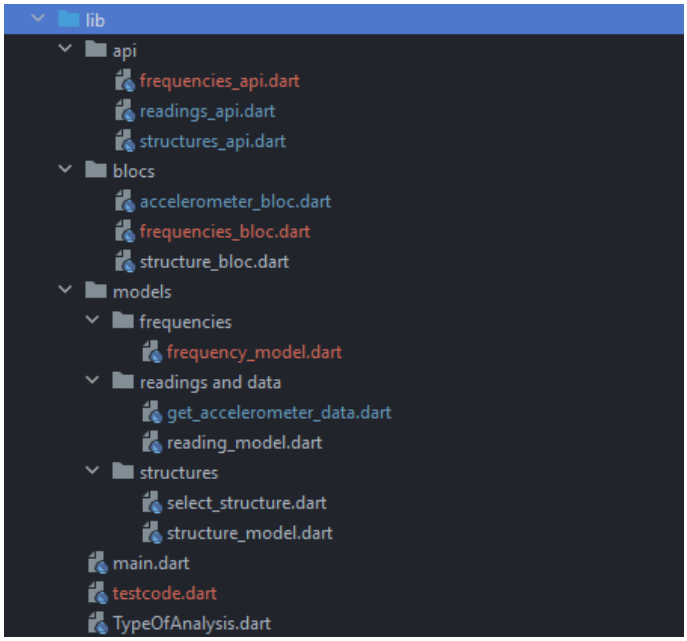

Figura 5 - Estrutura do projeto da aplicação iOS

Para fazer a implementação da versão iOS, decidi como irei estruturar o projeto. Primeiramente, criei os vários diretórios para os ficheiros mais importantes. A pasta *api* contém os ficheiros que importam e exportam os dados do servidor. A pasta do *blocs* contém os elementos que aplicam a lógica do *Business Logic Component* (BLoC) que irei explicar no passo (temp). A última pasta *models* contém os elementos visuais da aplicação, ou seja as componentes de UI.

4.3.2 BLoC

BLoC ou *Business Logic Component* é um padrão onde a apresentação da aplicação e a lógica da aplicação estão separadas para facilitar a reutilização de código e a testabilidade da aplicação.

Para esta aplicação, decidi aplicar o padrão BloC em todas as páginas. Cada bloc cria eventos que permite interações com a interface da aplicação e de seguida vai emitir os dados pedidos com o estado. Vou utilizar a lógica do bloc da primeira página como um exemplo.

O bloc de estruturas tem um evento associado. O evento é composto por vários estados, neste caso será o LoadingStructureState, o LoadedStructureState e o FailedToLoadStructureState. O LoadingStructureState tem como objetivo mostrar o estado da aplicação quando os dados da api ainda não foram lidos.

```dart
abstract class StructureEvent {}

class LoadStructureEvent extends StructureEvent {}

abstract class StructureState {}

class LoadingStructureState extends StructureState {}
```

Código 9 - Evento BLoC da Estrutura

O LoadedStructureState é o estado da aplicação onde as estruturas da api estão lidos e podem ser utilizados. O último estado, FailedToLoadStructureState, mostra o estado quando os dados da api não conseguiram ser lidos ou seja, deu erro.

```dart
class LoadedStructureState extends StructureState {
  List<Structure> structures;
  LoadedStructureState({required this.structures});
}

class FailedToLoadStructureState extends StructureState {
  Error error;
  FailedToLoadStructureState({required this.error});
}
```

Código 10 - Estado carregado e falha ao carregar estado das estruturas

Para que a aplicação mude de estado, é necessário uma função que recebe um evento e um estado. O estado inicial da aplicação na página inicial é o LoadingStructureState onde a função tenta chamar a função getStructureDataService que irei explicar na segmento (temp) para receber os dados da API do servidor da Lusófona. Nesta fase, vai ver se o estado atual é o LoadStructureState o que indica que os dados foram lidos ou se o estado é FailedToLoadStructureState. Após a verificação, a função vai emitir os dados do servidor para a aplicação.

4.3.3 Passo 1 – Identificação de Estruturas

Na criação da primeira página da aplicação, foi necessário ter um menu suspenso que mostrava uma seleção de estruturas para selecionar e dois botões de radio para selecionar o tipo de análise: Dados de Treino ou Deteção de Danos.

Para obter as estruturas que estão no API, a função getStructureDataService foi criada. Esta função faz um pedido ao servidor com um GET ao link da API, recebe um JSON no qual vai decodificar o corpo da mensagem e de seguida mete os elementos da mensagem numa lista com o formato de estrutura (Structure). A função depois retorna a lista das estruturas.

```dart
class GetStructureDataService {
  Future<List<Structure>> fetchStructure() async {
    try {
      final uri = Uri.http(link, '/api/structures/', {'format' : 'json'});
      final response = await http.get(uri);
      final json = jsonDecode(response.body.toString()) as List;
      final structures = json.map((structureJson) => Structure.fromJson(structureJson)).toList();
      globalStructure.firstStructure = structures[0];
      return structures;
    } catch (e) {
      throw e;
    }
  }
}
```

Código 11 - Função para capturar as estruturas do API

Apliquei um BlocProvider para gerar os estados e ver qual será o aspeto visual de cada estado. Quando o estado é LoadingStructureState, a aplicação vai estar num estado de carregamento onde o ecrã irá apresentar um indicador de progresso circular. Quando o estado é FailedToLoadStructureState, o ecrã vai mostrar uma mensagem a indicar o erro.

O estado pretendido, o LoadedStructureState, é onde aparece o UI da página e onde se aplica a sua funcionalidade. Aqui, o menu suspenso foi criado através de um DropdownButton utilizando a biblioteca do MaterialUI. Esta ferramenta neste caso, após o utilizador clicar, mostra uma lista de estruturas que podem ser selecionadas para fazer o estudo. Para conseguir escolher qual das estruturas será selecionada, era necessário estabelecer uma função para alterar o estado utilizando um setState para alterar a estrutura que está selecionada para a estrutura que o utilizador selecionou.

```dart
DropdownButton(
  hint: const Text('Choose a Structure'),
  value: _currentStructure,
  items: state.structures.map(
          (structure) =>
          DropdownMenuItem(value: structure,
              child: Text('${structure.name} (${structure.count} readings)'))).toList(),  // DropdownMenuItem
  onChanged: (structure) => setState(() => _currentStructure = structure as Structure),
)  // DropdownButton
```

Código 12 - Botão de dropdown

Aqui está uma representação do ecrã na Figura 6 do passo um com a listagem de todas as estruturas que estão presentes no servidor da Lusófona.

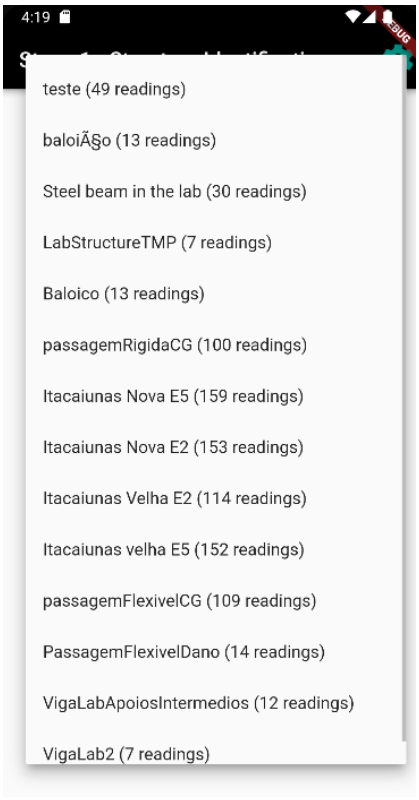

Figura 6 - Lista de estruturas no menu suspenso

4.3.4 Passo 2 – Aquisição de Dados

A segunda página da aplicação tem um gráfico de aceleração e tempo que demonstrava as leituras do acelerómetro no eixo z e dois botões: Um botão para começar a leitura do acelerómetro e um botão para avançar para o próximo passo.

Foi necessário também fazer uma chamada do API para saber o número do id da leitura. Para o gráfico, foi utilizado uma biblioteca chamada flutter_charts da Syncfusion, esta biblioteca permite ter gráficos em tempo real lineares. Foi criado uma ferramenta chamada _accelerometerGraph onde devolve um gráfico cartesiano que tem um controlador que controla o movimento da linha e recebe uma fonte de dados que neste caso são as leituras do acelerómetro.

```
Widget _accelerometerGraph() {
  return SfCartesianChart(
    series: <LineSeries<LiveData, int>>[
      LineSeries<LiveData, int>(
        onRendererCreated: (ChartSeriesController controller) {
        _chartSeriesController = controller;
        },
        dataSource: chartData,
        color: const Color.fromRGBO(192, 108, 132, 1),
        xValueMapper: (LiveData data, _) => data.time,
        yValueMapper: (LiveData data, _) => data.z,
      )  // LineSeries
    ],  // <LineSeries<LiveData, int>>[]
    primaryXAxis: NumericAxis(
      majorGridLines: const MajorGridLines(width: 0),
      edgeLabelPlacement: EdgeLabelPlacement.shift,
      interval: 5,
      title: AxisTitle(text: 'Time (seconds)')
    ),  // NumericAxis
    primaryYAxis: NumericAxis(
      axisLine: const AxisLine(width: 0),
      majorTickLines: const MajorTickLines(size: 0),
      edgeLabelPlacement: EdgeLabelPlacement.shift,
      title: AxisTitle(text: 'Acceleration (m/s²)')
    )  // NumericAxis
  );  // SfCartesianChart
}
```

Código 13 - Gráfico que mostra os dados do acelerómetro no eixo z

Foi necessário ter um temporizador para controlar quando os dados alteravam no ecrã de 50 em 50 milisegundos. O temporizador usufrui de uma função chamada updateDataSource onde adiciona as leituras do acelerómetro para uma lista e pede ao controlador para alterar os dados que estão visíveis no gráfico, criando um gráfico em tempo real.

```dart
int count = 0;
void updateDataSource(Timer timer) {
  chartData.add(LiveData(count, x, y, z));
  fullDataChart.add(LiveData(DateTime.now().millisecondsSinceEpoch, x, y, z));
  if (chartData.length == 20) {
    chartData.removeAt(0);
    _chartSeriesController.updateDataSource(
      addedDataIndexes: <int>[chartData.length - 1],
      removedDataIndexes: <int>[0],
    );
  } else {
    _chartSeriesController.updateDataSource(
      addedDataIndexes: <int>[chartData.length - 1],
    );
  }
  count++;
}
```

Código 14 - Atualização dos dados apresentados no gráfico

Para finalizar, existe um botão que controla quando é que o sensor do acelerómetro começa a funcionar de modo a mostrar no gráfico. Se o botão for clicado outra vez, a leitura do acelerómetro é pausado. Aqui está uma apresentação visual do passo 2 em funcionamento na Figura 7.

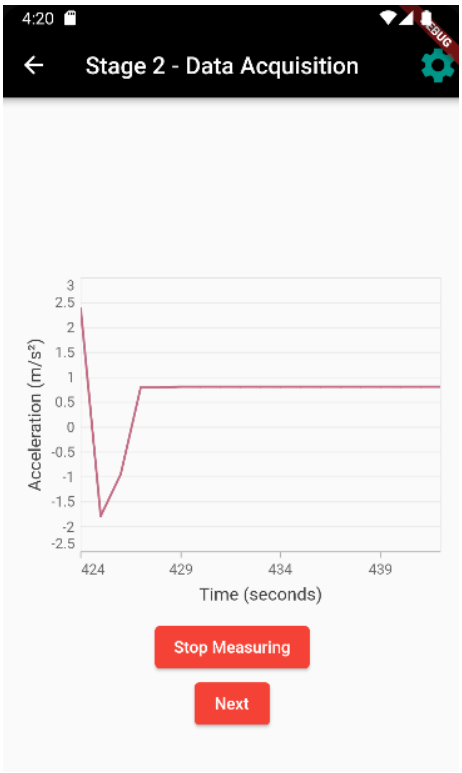

Figura 7 - Acelerómetro em funcionamento

4.4 Algoritmos de Compressão e Descompressão

Para resolver os problemas de desempenho no servidor, a implementação de um algoritmo de compressão e descompressão seria ideal. Um algoritmo de compressão é um algoritmo que reduz a quantidade de bytes necessários para ler dados e também reduz a quantidade de memória que é armazenada. A estratégia de compressão que vou implementar vai ser compressão baseada em dicionários.

A compressão baseada em dicionários não usa modelos estatísticos, foca na memória de strings. A memória pode ser um dicionário que pode ser estendido infinitamente ou pode ser um dicionário limitado implícito como janelas deslizantes. Cada string é armazenado num dicionário com um índice. As índices de todos os strings são usadas como palavras-código. O algoritmo de compressão e descompressão mantém individualmente cada um o seu dicionário mas os dicionários são idênticos. De acordo com ScienceDirect [SCDI21], existem muitas variações baseadas nas três famílias, nomeadamente LZ77, LZ78 e LZW.

Para implementar o algoritmo de compressão na aplicação móvel, decidimos usufruir da biblioteca Gzip. Esta biblioteca permite reduzir a memória dos dados através de compressão.

Para fazermos a compressão na aplicação, tive que criar uma nova função compress que recebia texto e comprimia o texto com gzip. Esta compressão devolvia uma lista de bytes do texto. Após a compressão, convertia a lista de bytes para texto com uma encriptação Base64. A função devolve o texto encriptado.

Depois da compressão ser feita, converti o ficheiro em JSON e enviei para o servidor.

No servidor é feita a descompressão e decodificação do ficheiro JSON enviado. Usando a biblioteca gzip, criei uma função decompress no servidor onde apliquei o decompress no “body” do ficheiro JSON e descodifiquei Base64 para UTF-8. Este último passo é para que os dados da leituras possam ser lidos e processados pelas restantes funções no servidor.

Após vários testes de desempenho, chegamos à conclusão que o problema da lentidão do envio de dados não era por causa do tamanho do ficheiro mas era pela ligação à base de dados. Decidimos manter a compressão porque vimos que a velocidade de envio dos dados era mais rápido do que anteriormente no localhost.

4.5 Outras funcionalidades necessárias

Outras funcionalidades que vou implementar para melhorar o desempenho da aplicação vão ser remover o eixo x e y (ao pedido dos professores de Engenharia Civil), corrigir o temporizador que não está sincronizada com o tempo atual, ou seja, depois de vários testes elaborados na aplicação, chegamos à conclusão que a contagem do tempo não estava sincronizada com o tempo real e ainda vou disponibilizar a opção de escolher mais do que três amostras. Todas estas melhorias são no âmbito de melhorar a experiência do utilizador e o desempenho da aplicação.

5 Benchmarking

Ao analisar o mercado, verifiquei que não existe nenhuma aplicação que tenha em mente o objetivo que a App4SHM quer concretizar. Como o foco deste projeto é a versão iOS, encontrei uma aplicação que tem funcionalidades semelhantes na App Store chamada vibration analysis desenvolvida pelo Dmitriy Kharutskiy. Uma amostra desta aplicação e algumas capturas de ecrã estão demonstradas na Figura 8.

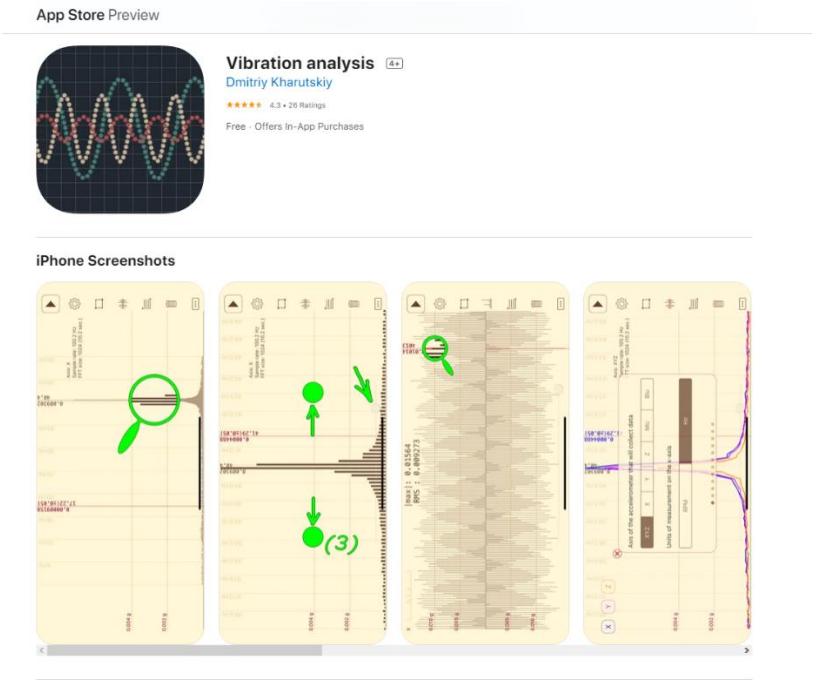

Figura 8 – App Store preview da aplicação Vibration analysis

Existem diferenças entre esta aplicação e a App4SHM dado a que não são aplicadas algoritmos de compressão e descompressão, a aplicação não tem um servidor dedicado onde se pode guardar os dados estudados, o export do ficheiro do concorrente é em .txt e Wav enquanto que a App4SHM exporta para Excel e lê ficheiros csv e a diferença fundamental é que apesar de serem aplicadas as mesmas técnicas de regressão linear e logarítmica, a minha aplicação envia os dados para o servidor para-se aplicar algoritmos de Machine Learning para prever os futuros comportamentos da estrutura em estudo e indica a possibilidade de danificações nos próximos tempos.

Esta aplicação desenvolvida substitui os acelerômetros profissionais para reduzir o custo de investigações na área de Engenharia Civil e ser acessível a pessoas interessadas no estudo de integridade estrutural.

6 Método e Planeamento

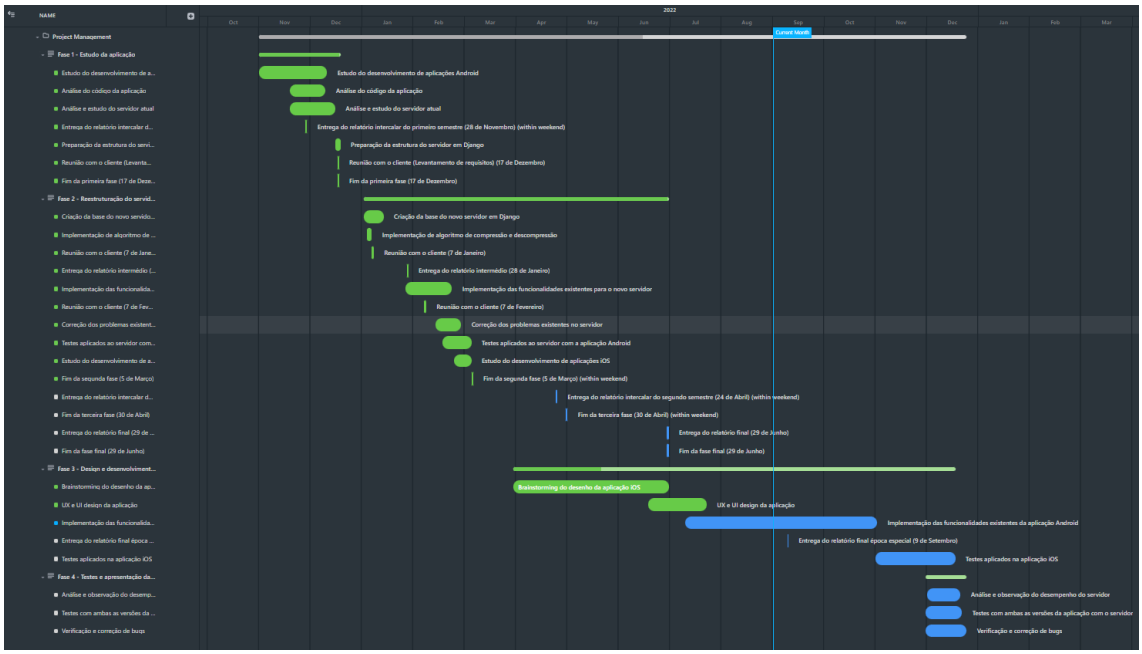

Figura 9 - Calendário de Gantt atualizado

Na Figura 5, está apresentada o calendário de planeamento da aplicação App4SHM. Na primeira fase, temos a parte do estudo da aplicação onde ocorre as análises necessárias para compreender o funcionamento da aplicação e do servidor. Esta fase é fundamental para perceber que alterações e correções são necessárias para o futuro da aplicação. Esta fase está completa com o levantamento de requisitos e entrega do relatório intercalar do primeiro semestre.

Na segunda fase é feito a reestruturação do servidor, ou seja, como referido na Solução , é nesta fase onde ocorre a mudança de Flask para Django. Esta fase é critica porque se o servidor não estiver a funcionar, a aplicação não pode usufruir das funcionalidades do servidor. Esta fase está concluida com o novo servidor de Django já estabelecido e a compressão e descompressão implementados. Esta fase foi completada dentro do prazo pois o cliente necessitava de um protótipo funcional para um dos requisitos mencionados 1.12 .

O *design* e desenvolvimento da aplicação iOS é executado na terceira fase. Nesta fase ocorre o desenho dos padrões para a aplicação em iOS, implementar as funcionalidades existentes na versão da aplicação Android e fazer testes para ver se a aplicação funciona como previsto. Esta fase é onde o projeto se encontra neste momento com uma aplicação iOS parcial devido a muitos atrasos por minha conta própria e com as dificuldades que foram encontradas.

A fase final é a fase de testes. Os testes vão ser elaborados pelo Departamento de Engenharia Civil para ver se a aplicação está a funcionar de acordo com as expectativas. Ocorre também a verificação e correção de *bugs* nesta fase. Esta fase não vai ser executado neste TFC devido ao cumprimento parcial da aplicação iOS.

Com a comparação dos dois calendários, podemos observar que existem muitos atrasos que ocorreram durante o decorrer deste projeto. A fase 3 ainda está a decorrer devido à continuação do desenvolvimento da aplicação iOS.

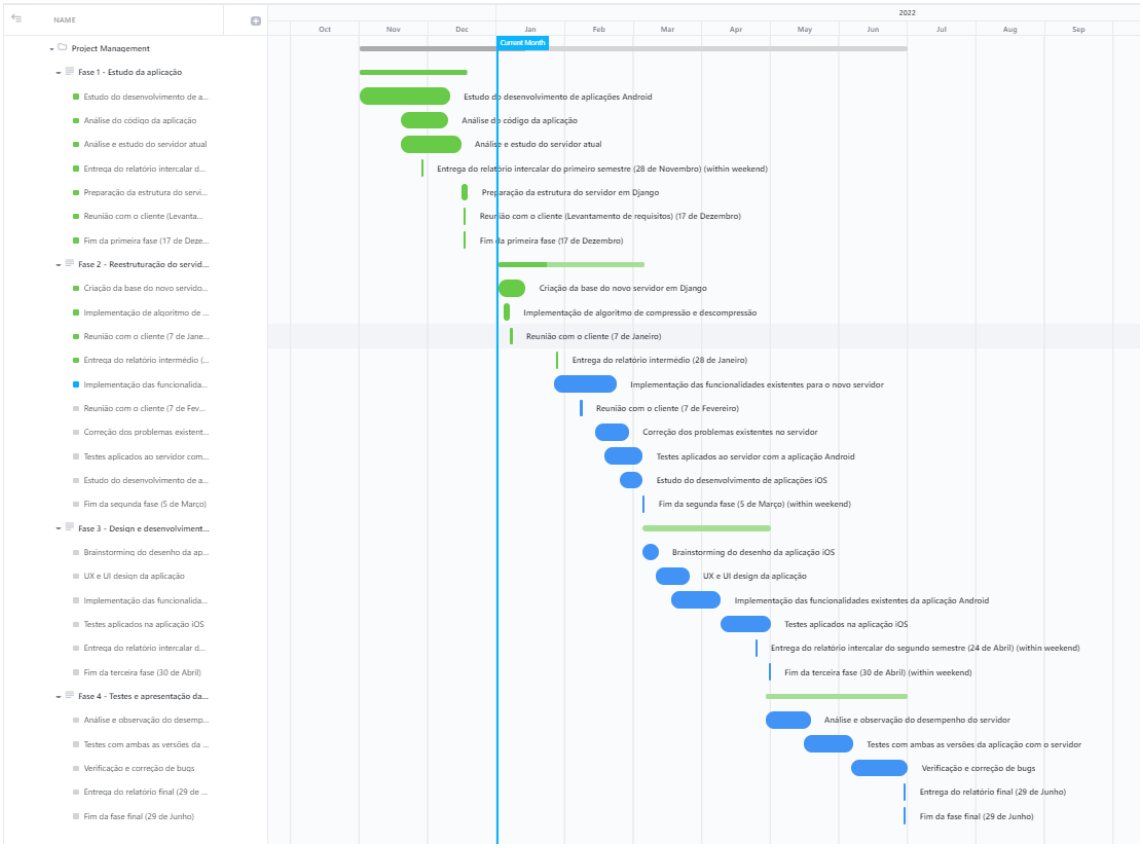

Figura 10 - Primeiro calendário de Gantt

7 Resultados

Este trabalho levou ao melhoramento de uma aplicação Android já desenvolvida pelos colegas do ano passado, à criação de um novo servidor numa nova framework Django e à origem de uma aplicação iOS com algumas funcionalidade pretendidas que poderá ser continuado após o termino deste trabalho final de curso.

O trabalho conclui uma grande parte dos requisitos pedidos, dando origem a uma aplicação funcional onde os professores e alunos do Departamento de Engenharia Civil podem fazer os seseus estudos sobre a danificação de estruturas. Os problemas encontrados foram maioritariamente resolvidos a não ser a aplicação iOS.

Para determinar que requisitos foram cumpridos, foi feito uma Tabela 2 para mostrar os resultados.

Tabela 2 - Tabela de requisitos e o seu cumprimento

Número do requisito	Descrição	Implementado?	Versão em que foi implementado	Em produção?
1.1	Geração de ficheiros de dados locais	Sim	1.1	Sim
1.2	Criação de um novo servidor Django para a aplicação	Sim	1.1	Sim
1.3	Associar, no servidor, um timestamp às observações realizadas	Sim	1.1	Sim
1.4	Compressão dos dados no telemóvel e descompressão dos dados no servidor	Sim	1.1	Não usada
1.5	Melhoramento da acessibilidade do servidor para a manutenção da base de dados	Sim	1.1	Sim
1.6	Correção dos saltos nos timestamps	Sim	1.1	Sim
1.7	Alteração do tempo na abcissa (de milissegundos para segundos)	Sim	1.1	Sim
1.8	Respostas temporais apenas no eixo z	Sim	1.1	Sim

1.9	Permitir zoom no eixo das abcissas sem alterar o eixo das coordenadas	Sim	1.1	Sim
1.10	Representar o eixo vertical em escala logarítmica	Não	1.1	Não
1.11	Alterar o label do eixo das abcissas	Sim	1.1	Sim
1.12	Apresentar um protótipo funcional	Sim	1.1	Sim
2.1	Desenvolvimento da aplicação iOS	Parcialmente	2.0	Não
2.2	Permitir que o utilizador possa escolher mais do que três frequências para estudar	Não	2.0	Não
2.3	Permitir a inserção de NaN para frequências que não se conseguem reconhecer no espetro de Welch	Não	2.0	Não
2.4	Upload de observações diretamente no servidor	Não	2.0	Não
2.5	Descarregamento de forma simples dos dados no servidor	Não	2.0	Não
2.6	Adição de temperatura ambiente como observação opcional	Não	2.0	Não
2.7	Aplicar os algoritmos diretamente na aplicação telemóvel	Não	2.0	Não

A aplicação iOS tem os primeiros dois das quatro etapas que devia ter o que é uma resolução parcial do problema da aplicação iOS.

Aqui estão as páginas e a explicação de como cada um funciona:

- **Passo 1 – Identificação de Estruturas**: Aqui, o utilizador pode selecionar qual a estrutura que quer estudar e escolher o tipo de análise que quer realizar.

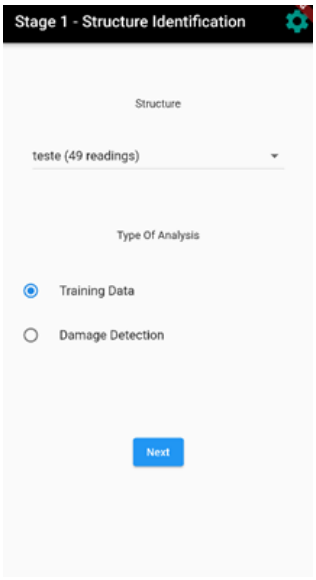

Figura 11 - Página de Identificação de Estruturas

- **Passo 2 – Aquisição de Dados**: Neste passo, o utilizador pode visualizar a captura de dados do acelerómetro no eixo z que vão ser usadas para o estudo da estrutura.

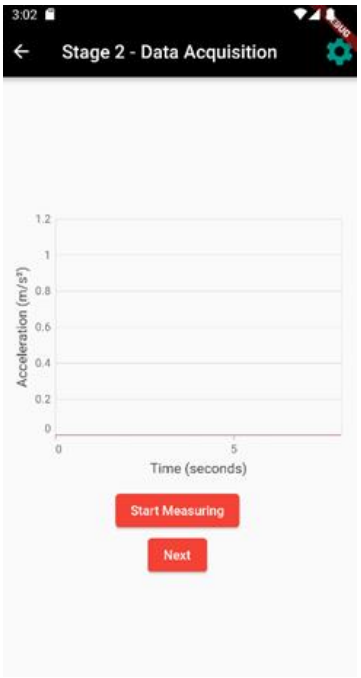

Figura 12 - Página de Aquisição de Dados

8 Conclusão e Trabalhos Futuros

Este trabalho teve muitas dificuldades nomeadamente no desenvolvimento da aplicação iOS. A aplicação desenvolvida pelos colegas do ano passado foi melhorado ao ponto de ser utilizável para fazer estudos da ponte localizado no rio Itacaiúnas durante o verão. O servidor Django comparado ao servidor de Flask do ano passado é muito mais prático e eficaz para o envio e visualização dos dados obtidos.

Aperfeiçoar o meu conhecimento de Kotlin, Python e Flutter foi muito necessário para o desenvolvimento das aplicações. Aprender a aplicar conceitos já conhecidos na disciplina de Computação Móvel como o BLoC e desenvolver ainda mais esse conhecimento foi essencial para a parte do desenvolvimento da aplicação iOS. Admito que foi necessário muito estudo para conseguir construir uma aplicação multiplataforma.

A aplicação iOS ainda tem de ser concluída pois só as primeiras duas etapas foram desenvolvidas. Gostarei de continuar a trabalhar nesta aplicação e terminar os requisitos que ainda são necessários fazer após o termino do TFC.

Em termos de melhoramentos necessários, diria que é necessário ter um design mais apelativo para os utilizadores da aplicação e implementar um sistema de geolocalização para poder associar as estruturas estudadas ás suas localizações, indicando no mapa se a estrutura está estável ou não.

Bibliografia

[STAT21] Mobile Operating System Market Share Worldwide, https://gs.statcounter.com/os-market-share/mobile/worldwide/, Oct 2021

[HACK21] Flask vs Django in 2021: Which Framework to Choose?, https://hackr.io/blog/flask-vs-django/, May 2021

[XCOD21] Xcode 13 Preview, https://developer.apple.com/xcode/, Nov 2021

[APSW21] Swift, https://www.apple.com/swift/, Nov 2021

[MDEV21] How to Convert an Android App to iOS or Vice Versa: 4-Step Process, https://mlsdev.com/blog/how-to-convert-android-app-to-ios/, Nov 2021

[SCDI21] Compression Algorithm, https://www.sciencedirect.com/topics/computer-science/compression-algorithm/, Nov 2021

[ANST21] Android Studio, https://developer.android.com/studio/, Nov 2021

[PYCH21] PyCharm, https://www.jetbrains.com/pycharm/, Nov 2021

[APPV21] Flask vs Django – Which Python framework to choose?, https://www.appventurez.com/blog/flask-vs-django/, Jan 2021

[REST22] What is REST?, https://www.codecademy.com/article/what-is-rest, Jan 2022

[APIS20] What is a REST API?, https://www.redhat.com/en/topics/api/what-is-a-rest-api, May 2020

[ASHM22] App4SHM – Smartphone application for structural health monitoring, Aguarda Publicação, Jan 2022

Glossário

LEI	Licenciatura em Engenharia Informática
LIG	Licenciatura em Informática de Gestão
TFC	Trabalho Final de Curso