UNIVERSIDADE
LUSÓFONA

App4SHM 3.0

Trabalho Final de curso

Relatório Final

Gabriel Pacheco, a22206007, LEI

Diogo Fonseca, a22204579, LEI

Orientador: Pedro Alves

Entidade Externa: Departamento de Eng. Civil da Universidade Lusófona

Departamento de Engenharia Informática da Universidade Lusófona

Centro Universitário de Lisboa

27/06/2025

Direitos de cópia

Agradecimentos

Agradecemos aos professores que nos acompanharam ao longo do curso, em especial ao Professor Pedro Alves por nos ter escolhido para este projeto e por todo o apoio e disponibilidade.

Resumo

A App4SHM tem como objetivo apoiar a análise e monitorização da integridade de estruturas, no sentido de permitir um melhor controlo sobe as mesmas de forma a garantir a segurança de quem utiliza as estruturas em monitoração.

No presente ano letivo, prosseguimos com o desenvolvimento e melhoramento da aplicação elaborada por colegas de anos anteriores.

A aplicação utiliza o acelerómetro embutido nos smartphones para captar vibrações que ajudam a avaliar a integridade estrutural de edifícios e infraestruturas. Esses dados são enviados para um servidor, onde algoritmos de Machine Learning os analisam. Com base nos resultados, um engenheiro civil pode usar a aplicação para avaliar o estado da estrutura, determinando se está estável, se requer reparos ou se corre o risco de colapsar.

A aplicação já desenvolvida revelou alguns erros que pretendemos corrigir, procurando também aprimorar as suas funcionalidades, nomeadamente conseguir-se medir a força de tirantes de uma ponte e efetuar medições, em simultâneo, através da comunicação entre diferentes dispositivos.

O Trabalho Final de Curso (TFC) apresentado neste relatório descreve o desenvolvimento das tarefas efetuadas para a resolução dos erros detetados, bem como a implementação de novas funcionalidades.

Abstract

App4SHM aims to support the analysis and monitoring of structural integrity to allow better control over these structures, ensuring the safety of those using the monitored infrastructures.

In the current academic year, we will continue the development and improvement of the application created by colleagues from previous years.

The application uses the accelerometer embedded in smartphones to capture vibrations that help assess the structural integrity of buildings and infrastructures. This data is sent to a server, where Machine Learning algorithms analyze it. Based on the results, a civil engineer can use the application to evaluate the structure's condition, determining whether it is stable, requires repairs, or is at risk of collapse.

The already developed application has shown some bugs that we aim to fix, while also seeking to enhance its functionalities, notably enabling the measurement of the force of bridge cables and performing simultaneous measurements through communication between different devices.

The final course project presented in this report describes the development of the tasks carried out to resolve the detected errors, as well as the implementation of new functionalities.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas

App4SHM - Application for Structural Health Monitoring

API - Application Programming Interface

REST - Representational State Transfer

1 Introdução

A App4SHM (*Application for Structural Health Monitoring*), ao longo do tempo de utilização, foi revelando diversos erros, embora a aplicação cumpra os objetivos, considerou-se que está apta para se proceder à melhoria dos erros. Para além disso, foi-nos proposta a implementação de novas funcionalidades que levarão ao lançamento da **App4SHM 3.0**.

Uma dessas funcionalidades será a opção de medir a força de tirantes de um a ponte, onde o utilizador coloca o telemóvel horizontalmente sobre o tabuleiro da estrutura e dessa forma ser-lhe-á dada a força desse tirante.

A outra funcionalidade será a possibilidade de realizar leitura de frequências por vários dispositivos em simultâneo, de modo a obter resultados mais precisos sobre os danos da estrutura medida.

1.1 Enquadramento

Face à importância de agir de forma preventiva, evitando acidentes como quedas de pontes devido ao comprometimento da sua integridade estrutural ao longo do tempo, lançando, diariamente, desafios a quem trabalha nesta área, as tecnologias são, cada vez mais, fortes aliadas na procura de soluções práticas e acessíveis.

Partindo desta realidade, a APP4SHM surgiu como uma solução inovadora para facilitar a análise de danos em estruturas, auxiliando o trabalho de monitorização dos responsáveis, nomeadamente engenheiros civis.

O trabalho em equipa com o departamento de engenharia civil da Universidade Lusófona tem possibilitado a atualização da APP4SHM na procura de melhorias nas suas funções, promovendo uma análise e monotorização das estruturas, gradualmente, mais eficaz e inovadora.

1.2 Motivação e Identificação do Problema

Os desafios enfrentados na monitorização de estruturas residem tanto na complexidade técnica quanto na acessibilidade das ferramentas tradicionais. A motivação para este projeto advém da necessidade de:

1. Corrigir erros existentes (bugs) que comprometem a estabilidade e utilização da aplicação.

2. Expandir a funcionalidade da App4SHM, permitindo medir forças específicas, como a tensão em tirantes de pontes, dando resposta às necessidades solicitadas por engenheiros civis.
3. Habilitar a conexão de múltiplos dispositivos em medições simultâneas, aumentando a precisão e o alcance das análises.
4. Garantir a evolução e eficiência da aplicação.

A importância deste trabalho reflete-se na sua capacidade de proporcionar soluções inovadoras, acessíveis e eficazes, democratizando o acesso a ferramentas avançadas de monitorização com custos mais reduzidos.

1.3 Objetivos

Os objetivos deste projeto estão interligados com a evolução da App4SHM e podem ser divididos em:

- **Geral**: Ampliar as capacidades da aplicação, resolvendo problemas técnicos e implementando funcionalidades inovadoras.
- **Específicos**:
    - Corrigir os bugs identificados nas versões anteriores.
    - Implementar a funcionalidade de medição de força em tirantes de pontes.
    - Permitir a conexão e sincronização de múltiplos dispositivos para medições simultâneas.

1.4 Estrutura do Documento

Este relatório está estruturado da seguinte forma:

- Secção 1: Introdução, incluindo enquadramento, motivação e objetivos do projeto.
- Secção 2: Pertinência e viabilidade do projeto, analisando o impacto e a sustentabilidade das soluções propostas.
- Secção 3: Especificação e modelação, detalhando os requisitos, casos de uso e protótipos.
- Secção 4: Solução desenvolvida, com descrição funcional, arquitetura e ferramentas utilizadas.
- Secção 5: Testes e validação, detalha como foram feitos os testes unitários e funcionais

- Secção 6: Métodos e planeamento, do projeto, como funcionalidades e metodologia do usada no trabalho
- Secção 7: Resultados dos testes em utilizadores, e análise do cumprimento de requisitos.
- Secção 8: Conclusão e propostas para trabalhos futuros.

2 Pertinência e Viabilidade

Esta secção apresenta uma análise detalhada da relevância do projeto App4SHM no contexto atual da monitorização estrutural, bem como uma avaliação da sua viabilidade técnica, económica, social e ambiental.

2.1 Pertinência

Dado o sucesso da utilização da App4SHM, desenvolvida em anos anteriores, por parte de engenheiros civis na monitorização de estruturas, o departamento de engenharia civil da Universidade Lusófona solicitou atualizações nesta aplicação. Desta forma, o presente projeto assenta nos feedbacks dados, em relação a alguns erros apresentados no decorrer da sua utilização, bem como na implementação de novas funcionalidades na App4SHM relacionadas com a capacidade de resolver problemas práticos enfrentados pelos seus utilizadores.

Os principais aspetos que destacam a relevância do projeto incluem:

Segurança pública: Falhas estruturais, como colapsos de pontes e edifícios, podem causar perdas humanas e materiais significativas. O App4SHM fornece uma solução acessível para monitorizar a integridade de estruturas e prever problemas antes que ocorram.

Redução de custos: Ao utilizar acelerómetros embutidos em smartphones, a aplicação elimina a necessidade de equipamentos caros e especializados, tornando a monitorização viável para empresas e organizações com orçamentos limitados.

Ampliação de funcionalidades: As novas funcionalidades propostas, como a medição de forças em tirantes e medições simultâneas, ampliam o leque de funções da App4SHM, posicionando-a como uma ferramenta versátil e essencial no setor.

Impacto social e ambiental: Através da prevenção de desastres estruturais, o projeto contribui para a proteção de vidas humanas e minimiza os impactos ambientais associados à reconstrução.

2.2 Viabilidade

A implementação das melhorias na App4SHM é viável com base em critérios técnicos, económicos e sociais, conforme detalhado a seguir.

Viabilidade técnica

- **Ferramentas disponíveis**: As tecnologias já utilizadas no projeto (Flutter, Django, Python) são adequadas para as novas funcionalidades. A introdução de funcionalidades como medições em múltiplos dispositivos será suportada por APIs (*Application*

Programming Interface) REST (*Representational State Transfer*) e sincronização em tempo real.

- **Capacitação da equipa**: A equipa de desenvolvimento possui experiência nas ferramentas tecnológicas necessárias, garantindo um progresso eficiente.
- **Protótipos anteriores**: As funcionalidades básicas da aplicação já foram testadas, e o projeto baseia-se em soluções comprovadas, como algoritmos de análise de frequência.

Viabilidade económica

- **Custos reduzidos**: As ferramentas usadas no projeto são de código aberto (Flutter, Python, Django), minimizando os custos de desenvolvimento.
- **Sustentabilidade financeira**: Com a publicação da aplicação em lojas como Google Play e App Store, pode-se explorar um modelo de negócios baseado em subscrições ou parcerias com instituições de engenharia.
- **Manutenção**: A aplicação já conta com uma infraestrutura de servidor implementada, reduzindo custos adicionais.

Viabilidade social

- **Aceitação pelo público-alvo**: Professores e engenheiros civis já demonstraram interesse na aplicação, indicando uma potencial adoção generalizada.
- **Usabilidade e acessibilidade**: O desenvolvimento da aplicação assenta em interfaces intuitivas, permitindo que seja utilizada por profissionais com diferentes níveis de especialização técnica.

Viabilidade ambiental

- **Minimização de impacto**: Ao utilizar smartphones como sensores, evita-se a produção de outros dispositivos, contribuindo para a redução do desperdício de recursos e emissões associadas.

2.3 Análise Comparativa com Soluções Existentes

2.3.1 Soluções existentes

No mercado atual, há soluções especializadas para monitorização estrutural, geralmente baseadas em dispositivos específicos, como acelerómetros profissionais. Exemplos incluem:

Geokon: Dispositivos avançados de monitorização, mas de alto custo.

Trimble 4D Control: Sistema de monitorização geotécnica que exige equipamentos caros e configurações complexas.

Autodesk Force Effect: Sistema de engenharia móvel para simular conceitos de design no campo, no escritório ou na sala de aula.

Essas soluções, embora precisas, não são acessíveis para instituições ou profissionais com restrições orçamentais, deixando um nicho significativo para a App4SHM.

2.3.2 Análise de benchmarking

Tabela 1- Comparação entre soluções existentes

	Geokon	Trimble	Autodesk Force Effect	App4SHM
Tipo de Medição	Inclinação / deformação	Geodésica / deformação estrutural	Simulação 2D de forças	Frequência vibracional
Cálculo de Força de Tirantes	✕	✕	✓	✓
Deteção de danos estruturais	✕	Parcial	✕	✓
Utiliza acelerómetro interno do smartphone	✕	✕	✕	✓
Necessita hardware externo	✓	✓	✕	✕
Funciona remotamente (sem ligação física direta)	✓	✓	✕	✓
Capacidade de análise em tempo real do dispositivo	✕	Parcial	✓	✓
Correção contínua de bugs e atualizações	✓	✓	✓	✓
Facilidade de uso em campo	Média	Média	Alta	Alta
Custo de entrada (licenças + equipamento)	Elevado	Muito elevado	Gratuito	Gratuito

Adequado para monitorização de pontes ou tirantes	✕	✓	✕	✓
Permite colaboração entre múltiplos dispositivos	✕	✕	✕	✓

Proposta de inovação e mais-valias

A inovação do App4SHM reside em:

- Tornar a monitorização de estruturas mais acessível e prática.
- Incorporar funcionalidades específicas, como a medição de forças em tirantes, ausentes em muitas soluções concorrentes.
- Permitir medições colaborativas em múltiplos dispositivos, algo inovador no mercado.

2.4 Identificação de oportunidade de negócio

O projeto apresenta potencial de comercialização, com aplicações tanto no setor público quanto privado:

- **Mercado-alvo:** Empresas de manutenção de infraestruturas, instituições educacionais e governos locais.
- **Modelo de negócio:** Licenças corporativas para grandes instituições, bem como subscrições para engenheiros individuais.
- **Escalabilidade:** A aplicação pode evoluir para incluir novos módulos, como monitorização de temperatura e corrosão.

3 Especificação e Modelação

Nesta secção, são detalhadas as características da solução a produzir, com base nos novos objetivos definidos para a **App4SHM 3.0**, bem como os requisitos funcionais e técnicos necessários para sua implementação. A modelação inclui os principais casos de uso, diagramas e protótipos que auxiliam na definição das interações da aplicação.

3.1 Análise de Requisitos

3.1.1 Enumeração de Requisitos

Tabela 2 - Enumeração de Requisitos

ID	Descrição	Prioridade	Tipo	Critérios de Aceitação
R1	Corrigir erros e bugs existentes na aplicação.	Alta	Técnico	Aplicação estável, sem falhas recorrentes.
R2	Implementar medição da força de tirantes.	Alta	Funcional	Cálculos precisos e consistentes.
R3	Suportar medições colaborativas de múltiplos dispositivos.	Alta	Funcional	Sincronização em tempo real com o menor atraso possível
R4	Criar interface gráfica para sessões colaborativas.	Média	Técnico	Utilizador deve ser capaz de configurar sessões sem dificuldades.

3.1.2 Descrição detalhada dos requisitos principais

Requisito R1 – Erros Detetados

O requisito R1 encontra-se dividido em duas categorias: erros do servidor e erros na aplicação mobile.

Erros no Servidor

Erro a calcular a gaussiana das cable forces.

Descrição:

Erro ocorrido em produção:

Após POST /api/cable-force com os seguintes dados:


```
{
"frequencies": [
5.7109557109557105,
17.83216783216783,
22.377622377622377
],
"reading": 5794,
"structure": 1,
"training": true
}
```


Rebentou nesta linha: kde = gaussian_kde(mean_forces)

com este erro: LinAlgError: Singular Matrix

(erro detetado pelo sentry)

Análise e causa:

Apesar de o erro ter sido registado no servidor, a causa residia numa chamada incorreta feita pela aplicação mobile. Esta estava a invocar o *endpoint* era chamado com estruturas do tipo Structure fora do modo de *training*. Estas estruturas não possuem uma lista de mean_forces, o que resultava numa tentativa de construção de uma matriz inválida, provocando o *crash* na linha referida.

Resolução:

Procedemos à validação do tipo de estrutura e à verificação da existência de mean_forces antes de prosseguir com o cálculo da Gaussiana, prevenindo assim este tipo de falha.

Inconsistência nos formatos de input e output de frequências

Descrição:

Neste momento, os formatos dos *input* & *output* de frequências são bastante díspares, o que complica muito tanto o tratamento dos resultados como o carregamento de novas frequências.

O formato de output é este:

structure__name,reading__date,training,frequencies

baloiço,2024-06-13 10:02:48,0,"[12.407254740313272, 17.60098928276999, 22.135201978565 54]"

baloiço,2024-06-13 09:58:25,1,"[10, 16.58823529411765, 19.764705882352946]"

Reparem que este formato contém frequências separadas por vírgula e não dá para abrir diretamente no Excel (por exemplo), por causa das aspas e dos parênteses retos.

O formato de input é este:

2022-10-21 20:01:46;8.333333333333334; 16.666666666666668; 20.0

2022-10-21 20:01:15;6.862745098039215; 9.80392156862745; 20.588235294117645

2022-10-21 19:57:49;6.578947368421053; 9.86842105263158; 13.815789473684212

Este formato contém frequências separadas por ponto e vírgula e dá para abrir diretamente no Excel.

Idealmente, um ficheiro exportado devia poder ser diretamente importando.

Resolução:

A correção foi simples e passou por alterar a forma como os dados são exportados no ficheiro admin.py, garantindo que o formato de *output* corresponde ao formato de *input*, isto é, frequências separadas por ponto e vírgula, sem aspas nem parênteses.

Desta forma, assegurámos a compatibilidade total entre ficheiros exportados e importados, permitindo que um ficheiro exportado possa ser utilizado diretamente para uma nova importação, sem necessidade de edição manual.

Power spectrum files should have the csv extension

In django admin, if I go to Power Spectrums:

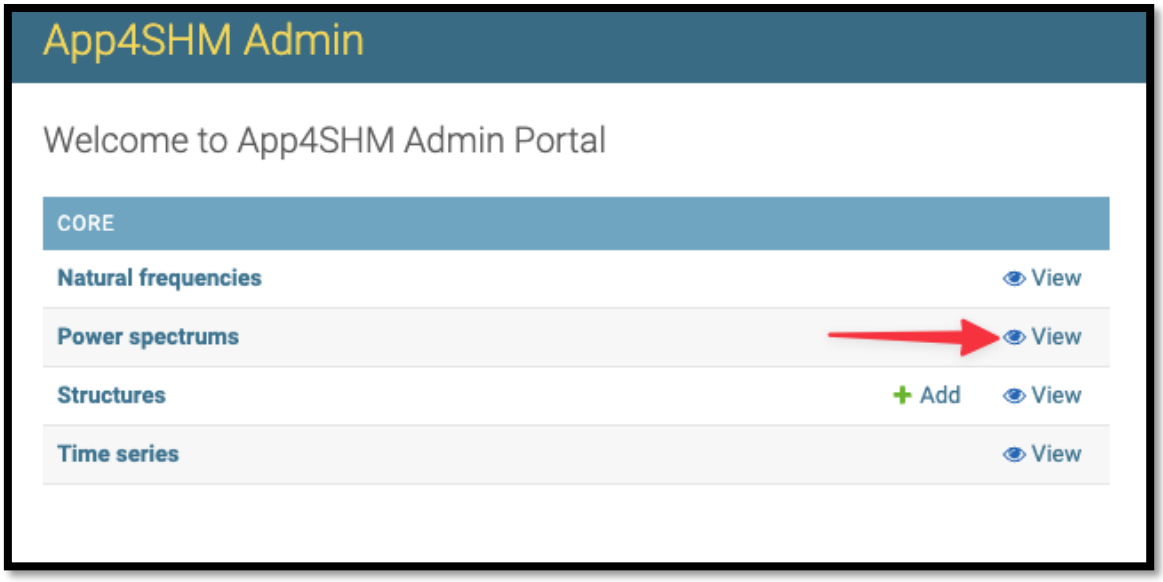


Figura 1 - Admin Django

Then click in one of the readings I get this:

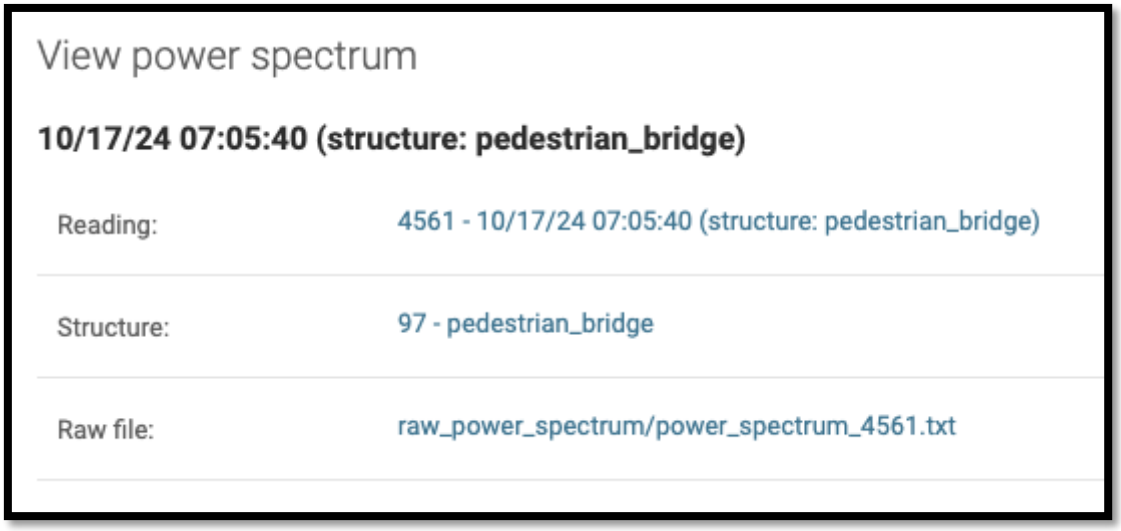


Figura 2 - Power Spectrum Django

The Raw File should be a csv file, not a txt file. Actually the file is already in csv format, it's only the extension that is wrong.

Resolução:

A resolução foi bastante simples. Bastou alterar a extensão do ficheiro exportado de .txt para .csv, garantindo que o conteúdo segue o formato esperado pelo Excel (valores separados por ponto e vírgula, sem aspas ou parênteses desnecessários). Com esta pequena alteração,

conseguimos assegurar a compatibilidade entre os ficheiros de input e output, permitindo que um ficheiro exportado possa ser reutilizado diretamente como input, sem necessidade de transformação manual.

Erros na Aplicação Mobile

After going back from the Time Series to the Structures screen, "next" stops working

Steps to reproduce:

- Login as guest
- In the Structures screen hit the "next" button
- In the Time Series screen, go back, tapping the back arrow in the top left corner
- The "next" button no longer works

Causa:

O problema ocorria porque, após a navegação para o ecrã de Time Series, o fluxo não notificava que a operação tinha sido concluída ao regressar. Como consequência, qualquer lógica associada ao evento de "conclusão" (como a ativação do botão “next”) não era executada.

Resolução:

A resolução consistiu em modificar a navegação para garantir que a função onFinished fosse chamada após o retorno do ecrã Time Series. Isto foi feito adicionando um then() à chamada pushNamed.

UI enhancements in multiple device readings

- In the first screen, change "Create a room" to "Multiple devices reading"
- This screen:

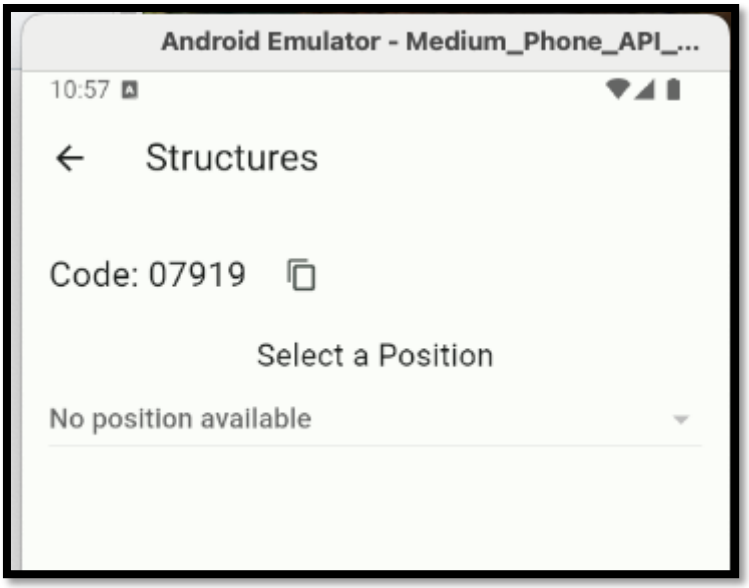


Figura 3 - Structures Multiple Devices

Should have a title "Multiple devices configuration" instead of "Structures"

Should show the selected strucuture id and name

Should show a text "A devices' network has been created with the following code. Please select the position of this device"

After selecting a position, no need to show snack bar "Connected to positon ..." since it it already clear by the green information panel on top.

Not sure about this bottom bar since it doesn't fit in the overall appearance of the app:

Figura 4 - Bottom Bar

For now, let's keep it but with labels (a good UI principle)

Resolução:

Melhorámos a página conforme solicitado, ficando como ilustrado na figura abaixo.

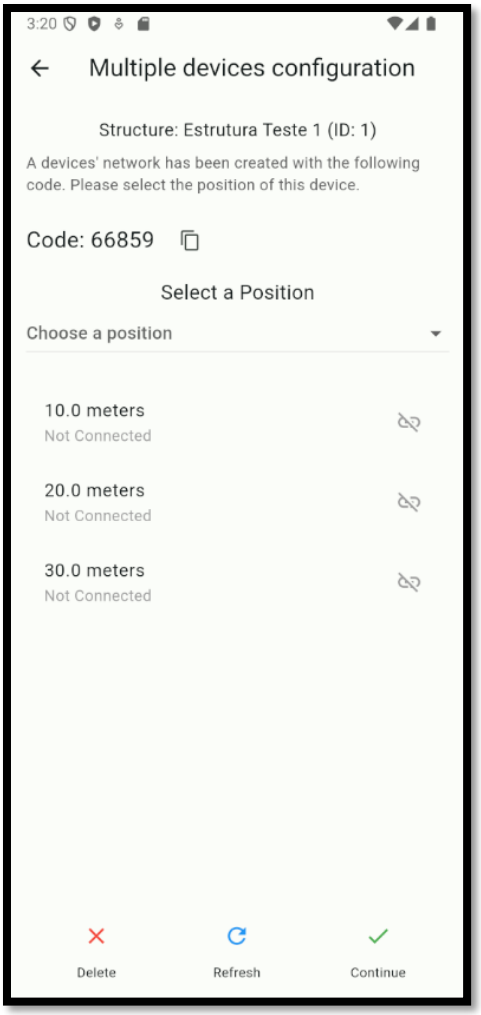


Figura 5 - Structures Multiple Devices

“Cable Force” screen should fully display Gaussian curve

In the “Cable Force” screen, the graph currently does not fully display the entire Gaussian-shaped curve. The curve appears cut off, and the plotted point is not well-centered within the graph area. The graph should automatically adjust its scaling or margins so that the entire curve is visible and properly framed, as expected in a Gaussian distribution. Please refer to the image below for the current issue.

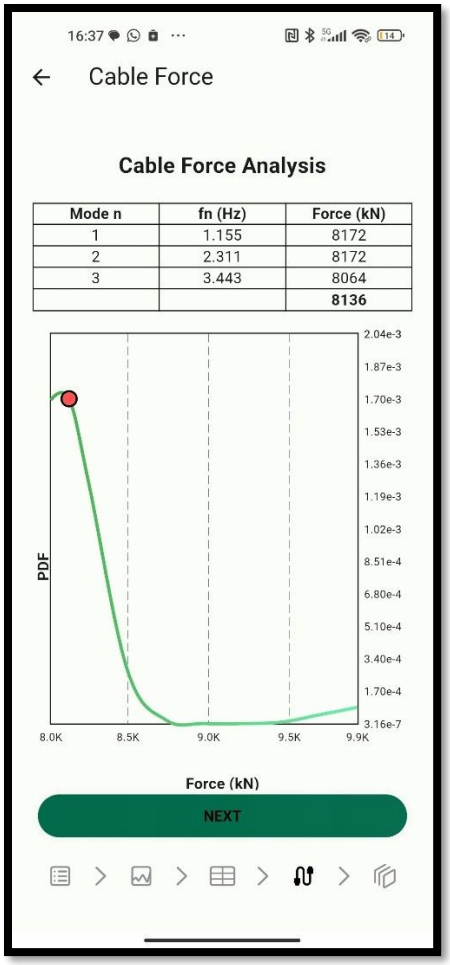


Figura 6 - Ecrã Cable Force

Resolução:

Adicionámos margens aos eixos X e Y, ajustadas dinamicamente consoante o tamanho do gráfico, de forma a garantir que toda a curva Gaussiana fique visível e corretamente enquadrada. Esta abordagem assegura que o ponto representado se encontra centrado e que a forma completa da distribuição é apresentada como esperado.

Os requisitos principais (R2, R3 e R4) são descritos em detalhe a seguir, com dependências, objetivos e critérios de aceitação para orientar a sua implementação e validação.

R2: Implementar a medição da força de tirantes

- **Objetivo:** Permitir que os utilizadores obtenham a força em tirantes de pontes a partir de dados de vibração coletados pelos acelerómetros dos dispositivos móveis.
- **Dependências:**

    - Desenvolvimento de algoritmos específicos para calcular forças com base em dados de aceleração.
    - Integração de algoritmos fornecidos por professores de Engenharia Civil.
- **Critérios de aceitação:**
    - Resultados devem ser precisos e consistentes, com uma margem de erro aceitável (até ±5%).
    - Os cálculos devem ser apresentados em tempo real e exibidos na interface gráfica da aplicação.
    - Os valores devem ser armazenados no servidor para referência futura.

R3: Suportar medições colaborativas de múltiplos dispositivos

- **Objetivo:** Habilitar que múltiplos dispositivos trabalhem juntos, recolhendo dados simultaneamente e sincronizando-os em tempo real para melhorar a precisão das medições.
- **Dependências:**
    - APIs REST para troca de dados entre dispositivos e servidor.
    - Implementação de timestamps para sincronização precisa dos dispositivos.
- **Critérios de aceitação:**
    - O sistema deve sincronizar os dispositivos com o menor atraso possível, idealmente abaixo de 20 ms.
    - As medições consolidadas devem ser armazenadas no servidor e exibidas de forma unificada.
    - Deve ser possível visualizar o estado de conexão de cada dispositivo na sessão.

R4: Criar interface gráfica para sessões colaborativas

- **Objetivo:** Proporcionar uma interface intuitiva que permita aos utilizadores configurar e monitorizar sessões colaborativas de medição.
- **Dependências:**
    - Feedback de utilizadores (engenheiros civis) para garantir usabilidade.
    - Design responsivo e adequado tanto para smartphones quanto para tablets.
- **Critérios de aceitação:**
    - O utilizador deve conseguir configurar uma sessão em menos de 1 minuto.

- A interface deve apresentar informações claras sobre dispositivos conectados, progresso das medições e resultados.
- Deve incluir mensagens de erro e status no caso de falhas na conexão.

3.2 Modelação

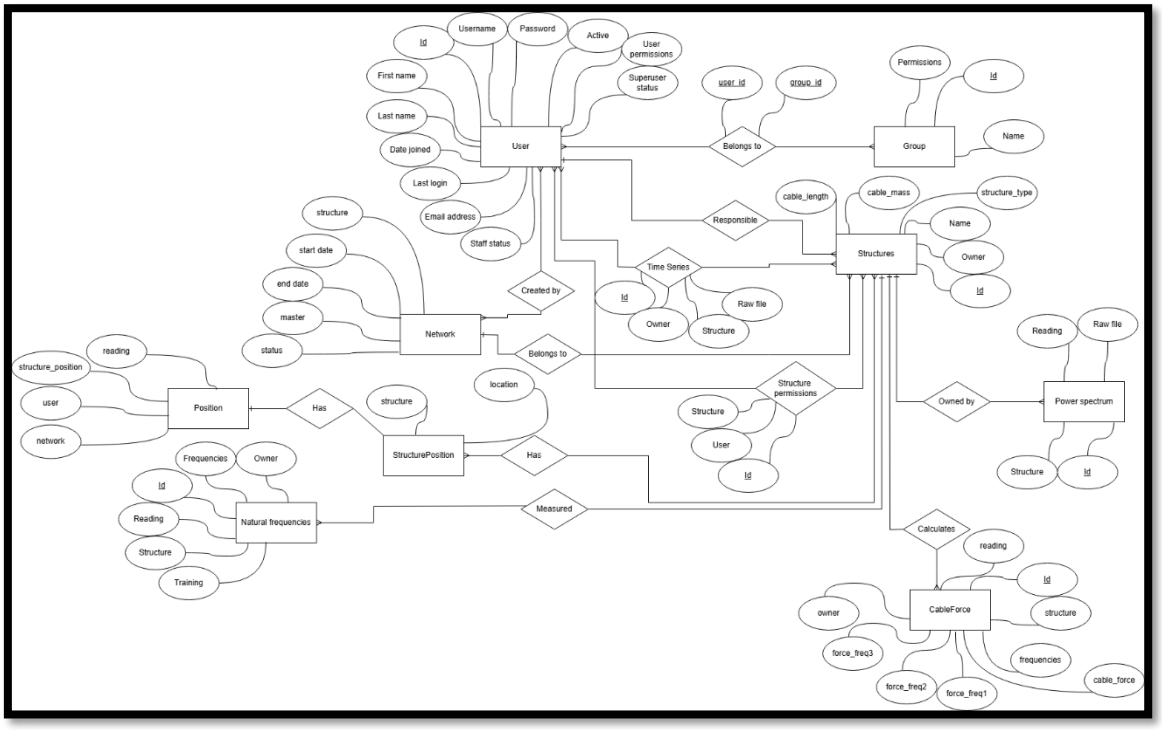


Figura 7 - Diagrama ER

O sistema de gestão de estruturas é composto por diversas entidades interligadas, centradas na tabela *Structure*, que representa estruturas físicas ou cabos.

As medições temporais são armazenadas em *TimeSeries* e enriquecidas por entidades associadas como *PowerSpectrum*, *NaturalFrequencies* e *CableForce*. A localização espacial das estruturas é gerida por *StructurePosition* e integrada em redes de medição (Network) através da entidade *Position*. O controlo de acesso é feito através da tabela *StructurePermission*.

A modelação favorece uma estrutura normalizada e extensível, adequada para sistemas de monitorização estrutural com suporte a múltiplos utilizadores e análises em série temporal.

3.2.1 Structure

Entidade central que representa uma estrutura física ou cabo.

- *name*: nome único da estrutura.
- *owner*: utilizador responsável.
- *structure_type*: tipo da estrutura (*structure* ou *cable*).
- *cable_length* & *cable_mass*: apenas aplicável para estruturas do tipo cable.
- Validação personalizada em save() para garantir preenchimento correto para cabos.

Relações:

- *TimeSeries*, *PowerSpectrum*, *NaturalFrequencies*, *CableForce*, *StructurePosition*, *Network*, *StructurePermission*.

3.2.2 StructurePermission

Controla permissões de acesso a estruturas por utilizadores específicos.

- *structure*: estrutura à qual o acesso é concedido.
- *user*: utilizador com permissão.

3.2.3 TimeSeries

Representa leituras temporais de sensores numa estrutura.

- *date*: data de criação automática.
- *structure*: estrutura à qual se refere a leitura.
- *raw_file*: ficheiro bruto da leitura.
- *owner*: utilizador que carregou a leitura.

Relações:

- *PowerSpectrum*, *NaturalFrequencies*, *CableForce*, *Position*.

3.2.4 PowerSpectrum

Contém o espectro de potência associado a uma leitura temporal.

- *reading*: referência à leitura (*TimeSeries*).
- *structure*: redundância para facilitar filtragens.
- *raw_file*: ficheiro bruto com os dados do espectro.

3.2.5 NaturalFrequencies

Guarda as frequências naturais extraídas de uma leitura.

- *reading*: TimeSeries base.
- *structure*: estrutura associada.
- *frequencies*: JSON com as frequências.
- *training*: se os dados são usados para treino
- *owner*: utilizador que processou os dados.

3.2.6 CableForce

Calcula ou armazena a força em cabos com base em frequências.

- *reading*: TimeSeries base.
- *structure*: estrutura (cabo) associada.
- *frequencies*: JSON com frequências usadas para o cálculo.
- *cable*_force: força total.
- *force_freq1*, *force_freq2*, *force_freq3*: forças específicas por frequência.
- *owner*: utilizador responsável.

3.2.7 StructurePosition

Define posições específicas ao longo da estrutura.

- *structure*: estrutura associada.
- *location*: localização (em metros).

3.2.8 Network

Representa uma rede de recolha de dados ou análise para uma estrutura.

- *start_date* / *end_date*: datas de operação.
- *master*: utilizador responsável.
- *structure*: estrutura associada.
- *status*: estado da rede (*waiting*, *reading*, *completed*).

3.2.9 Position

Guarda dados posicionais dentro de uma rede.

- *reading*: leitura base (opcional).
- *structure_position*: referência a *StructurePosition*.
- *user*: utilizador que fez a medição.
- *network*: rede à qual a posição pertence.

3.3 Protótipos de Interface

3.3.1 *Mockups* Pré Reunião

O mapa aplicacional teve uma ligeira alteração, pois para as implementações das novas funcionalidades não será necessário a criação de muitos ecrãs. Adicionámos apenas um novo ecrã chamado *Join Structure Measurement* (Figura 5).

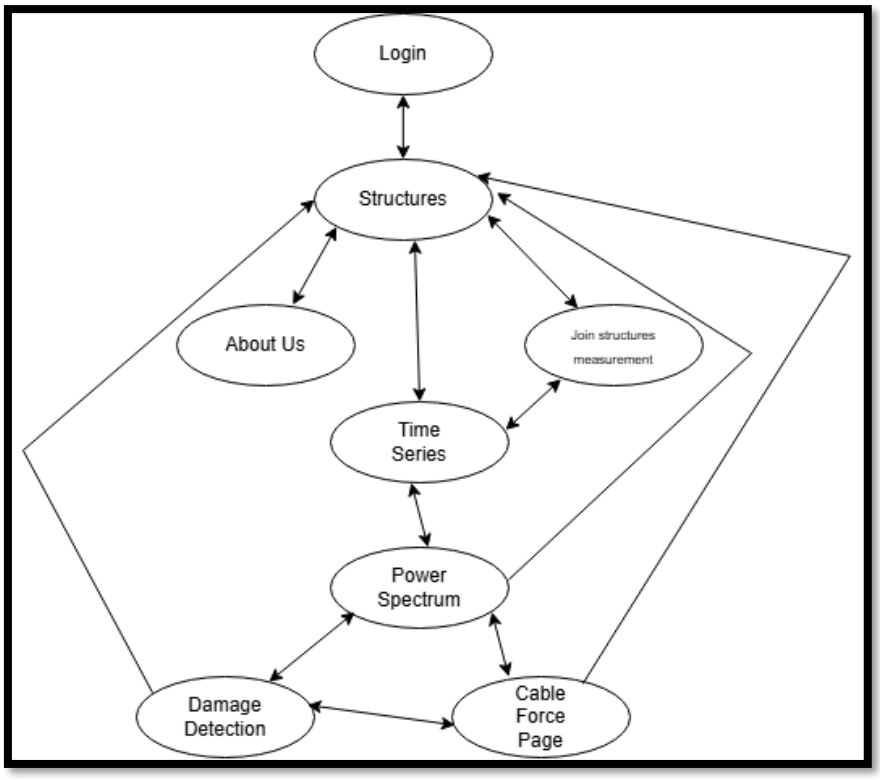


Figura 8 - Mapa Aplicacional

O ecrã *Structures* (Anexo 1) vai ter uma ligeira alteração para que sejam implementadas estas novas funcionalidades.

Alterações:

- Na D*ropDown* de *Select a Sctructure*, vai dizer o tipo de estutura, tirante ou estrutura.

- Para realizar as medições em simultâneo será adicionada uma nova *CheckBox*.

Caso a *CheckBox* seja selecionada, aparecem novas informações levando a que novas mudanças surjam no ecrã (Anexo 2). O dispositivo que selecionar a *CheckBox* será considerado o *host*.

Mudanças:

- Código para entrar na medição em simultâneo.
- Lista de posições, mostrando se está ou não conectado.

Para se conectar a esta medição em simultâneo, é necessário selecionar os 3 pontos que se encontram no canto superior direito do ecrã *Structures*.

Alterações:

- Adicionamos uma opção de Entrar na Medição (Anexo 3).

Ao clicar na opção de Entrar na Medição surge um novo ecrã (Anexo 4), este permite que o utilizador se conecte a uma medição como *slave*.

Neste Ecrã:

- *TextBox* para introduzir o código que é gerado no *host*.
- Após o código da *TextBox* ser validado, a *DropDown* da posição fica com conteúdo relativamente às posições da estrutura selecionada no *host*.
- Ao selecionar a posição, o botão de *Connect* fica ativo.

Por fim, voltando ao *host*, após as medições necessárias, no ecrã de *Demage Detection* é, então, apresentada uma nova secção para mostrar a força dos tirantes, isto caso a estrutura seja apenas desse tipo.

Informação:

- Aparece um pequeno texto com a informação das forças dos tirantes (Anexo 5).

3.3.2 *Mockups* Pós Reunião

Após reunirmos com o Departamento de Engenharia Civil da Universidade Lusófona, foi proposto alterar alguns dos *mokups* para que ficassem mais intuitivos para os clientes. Concluímos que seria melhor refazê-los por completo e detalhar mais os mesmos. Apresentamos então os esboços dos *mokups* finais.

Na página das *Structures* (Anexo 6), mantivemos a ideia inicial com algumas melhorias. **Neste ecrã:**

- Foi adicionada uma *CheckBox* com a opção de realizar medições em simultâneo.
- O nome do botão foi alterado para *Create a Room*, pois o utilizador irá criar uma network associada a uma estrutura.
- Será exibido o ID da *Network*, permitindo o acesso de outros utilizadores (explicado com mais detalhe mais à frente).

No *mockup* do Anexo 7, é apresentada a *DropDown* onde o utilizador escolhe a estrutura.

Neste ecrã:

- A *DropDown* exibe não só o nome da estrutura, mas também o respetivo tipo: S*tructure* ou *Cable*.

Ao clicar no botão *Create a Room*, o utilizador é direcionado para um novo ecrã (Anexo 8), onde pode configurar a sua *Network*.

Neste ecrã:

- É apresentado o ID da *Network* (*code*), que pode ser copiado.
- É possível selecionar a posição atual do utilizador através da *DropDown*.
- São visíveis as posições que já estão conectadas ou não.
- Estão disponíveis três botões:
    - *Refresh*: para atualizar as informações.
    - Confirmar: para validar os dados e avançar para a próxima página.
    - Cancelar: para sair do processo.

No *mockup* do Anexo 9, é possível ver a *DropDown* das posições em funcionamento.

Neste ecrã:

- A *DropDown* apresenta apenas as posições que ainda não estão ocupadas, facilitando a seleção por parte do utilizador.
- Mostra a distância a que o utilizador está de cada uma das posições.

Voltando ao ecrã das *Structures* (Anexo 10), será utilizado o ícone dos três pontos no canto superior direito para permitir que outros utilizadores se juntem a uma *Network* existente.

Neste ecrã:

- Ao clicar nos três pontos, surge a opção *Join Structure*.
- Ao selecionar esta opção, o utilizador é redirecionado para uma nova página.
- Nessa página, deve ser introduzido o ID da *Network*, apresentado no ecrã do utilizador que criou a *Network* (Anexo 9).

No Anexo 11, é mostrado o ecrã para o qual o utilizador é direcionado após clicar em *Join Structure*.

Neste ecrã:

- É possível ver uma *DropDown* com as posições disponíveis.
- Existirá uma *TextBox* onde o utilizador deve introduzir o ID da *Network*.
- Há um botão para confirmar e juntar-se à *Network*.
- Como é mostrado na *mockup*, a *DropDown* estará inicialmente vazia, pois o utilizador ainda não está conectado a nenhuma *Network*, não sendo possível determinar a qual estrutura está associado.

Após o utilizador inserir o ID da *Network* e clicar no botão *Join*, a *DropDown* é automaticamente preenchida com as posições disponíveis, mostra também a distância a que se encontra atualmente de cada posição.

- Esta atualização pode ser visualizada no Anexo 12.

Após o utilizador selecionar uma posição na *DropDown*, o botão *Connect* fica ativo.

- Este comportamento pode ser observado no Anexo 13.

No Anexo 14, é mostrado o ecrã *Join Structure* com o menu de opções aberto no canto superior direito.

Neste ecrã:

- É possível aceder aos três pontos no canto superior direito.
- Ao clicar, surge uma opção que permite voltar ao ecrã *Structure*.

- Esta funcionalidade oferece uma forma rápida de regressar ao ecrã anterior sem concluir a ligação à *Network*.

No Anexo 15, é apresentado o ecrã *Time Series* do *host*.

Neste ecrã:

- É indicada a posição em que se encontra o *host*.
- É mostrada uma tabela com o estado de todas as posições.
- Está incluído o gráfico habitual do *Time Series*, tal como nas páginas normais desta funcionalidade.
- No canto superior direito, existe um botão para iniciar a leitura.

No Anexo 16, é apresentada a animação do gráfico após o início da leitura no *host*.

Neste ecrã:

- É visível a animação do gráfico correspondente à leitura em tempo real.
- A leitura em curso nos dispositivos *slave* também é apresentada na tabela das posições, refletindo os dados recebidos em simultâneo.

No Anexo 17, é apresentado o ecrã após o término da leitura.

Neste ecrã:

- O botão *Connect* volta a ficar ativo.
- A tabela das posições é atualizada, apresentando o estado de cada uma delas após a leitura.

No Anexo 18, é apresentado o ecrã *Time Series* do *slave*.

Neste ecrã:

- A interface é idêntica à do *Time Series* normal.
- A única diferença é a ausência do botão para iniciar a leitura, uma vez que esta é controlada diretamente pelo *host*.

No Anexo 19, é apresentado o ecrã *Cable Force*, utilizado para medir a força dos tirantes.

Neste ecrã:

- É mostrada uma tabela com as forças calculadas para cada uma das três frequências selecionadas no *Power Spectrum*.
- É apresentado um gráfico com o histórico das forças medidas anteriormente, indicando com pormenor a força atual no gráfico.

4 Solução Desenvolvida

Nesta secção, descreve-se a solução proposta para atingir os objetivos definidos, com foco na arquitetura, tecnologias, ferramentas utilizadas e nos componentes principais do sistema.

4.1 Apresentação

A solução proposta baseia-se na ampliação e otimização da App4SHM, mantendo a sua missão de fornecer uma alternativa acessível para monitorização estrutural. O projeto concentra-se em três melhorias principais:

1. **Correção de *bugs*** para garantir a estabilidade da aplicação.
2. **Medição da força de tirantes em pontes** utilizando algoritmos específicos.
3. **Conexão de múltiplos dispositivos para medições colaborativas**, permitindo maior abrangência e precisão.

A solução é projetada para ser multiplataforma (iOS e Android) e utiliza uma arquitetura cliente-servidor para processar e armazenar dados de forma eficiente.

4.2 Arquitetura

A arquitetura do sistema é composta por três camadas principais:

- **Cliente**: Aplicação móvel (desenvolvida em Flutter) instalada nos dispositivos dos utilizadores. Os clientes capturam dados, exibem informações e gerem sessões colaborativas.
- **Servidor**: Um servidor central (desenvolvido em Django) responsável por:
    - Processar dados coletados (cálculo de forças, tensões e sincronização de dispositivos).
    - Armazenar informações em uma base de dados relacional (MySQL).
- **Comunicação**: APIs REST que permitem a troca de dados entre clientes e servidor de forma segura e eficiente.

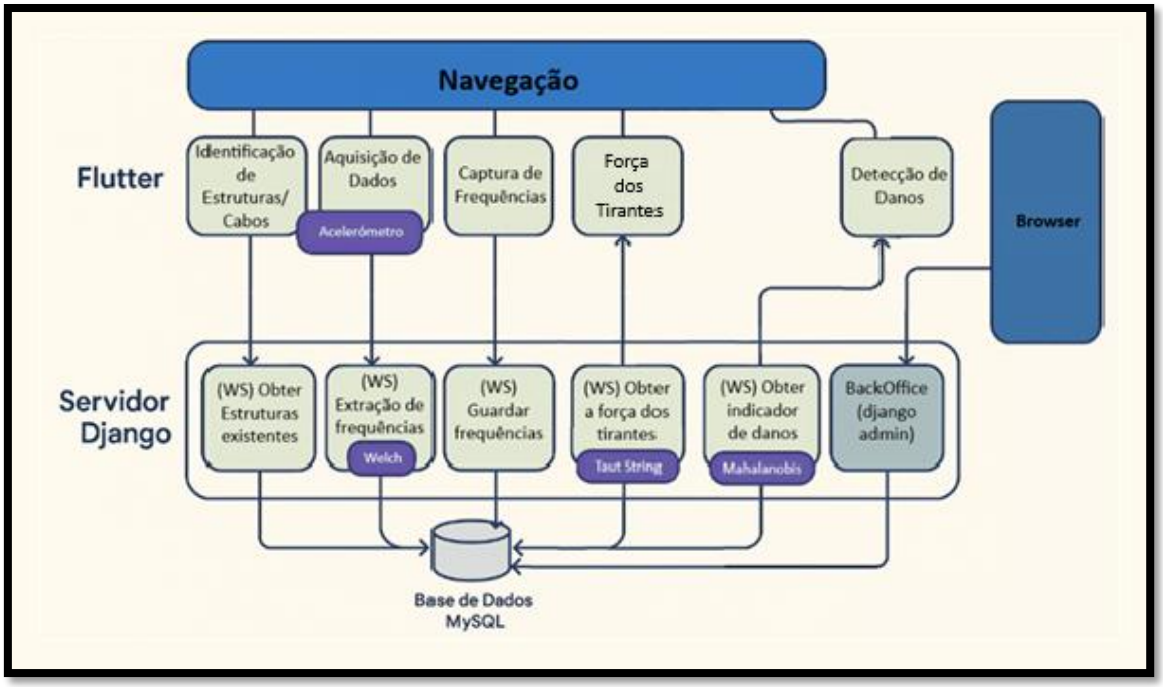


Figura 9 - Desenho da Arquitetura

4.3 *WebServices* – Força de Tirantes

A App4SHM implementa uma funcionalidade avançada de análise da força em tirantes com base no espectro de potência obtido via acelerómetro. Esta abordagem permite estimar a força axial presente em cabos ou elementos tensionados a partir da análise das frequências naturais de vibração.

A comunicação entre o dispositivo móvel e o servidor segue o seguinte fluxo:

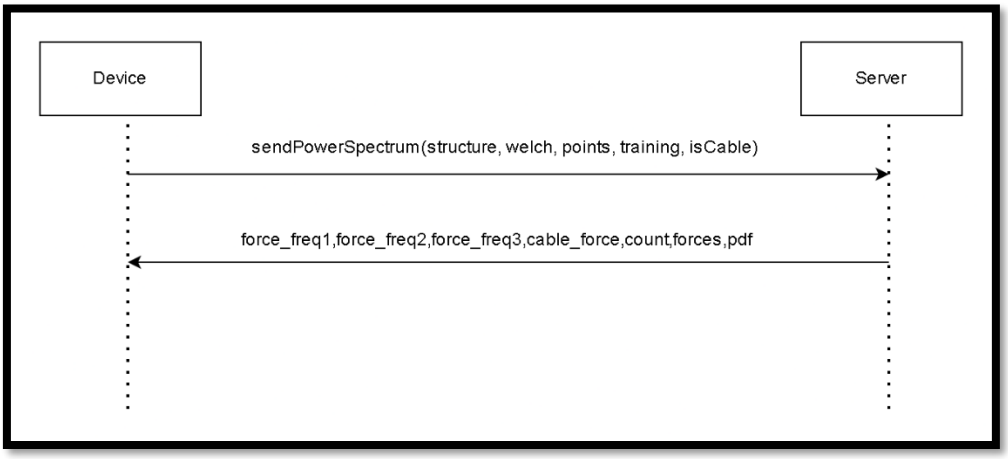


Figura 10 - Diagrama de Sequência (Força de Tirantes)

Legenda:

- ***Device*:** smartphone do utilizador que realiza a medição.
- ***Server*:** *backend* responsável por processar o espectro de potência e devolver os resultados.

No Anexo 20 é possível encontrar uma descrição detalhada do funcionamento do *WebService* utilizado na etapa de *CableForce*, explicando de forma clara os dados envolvidos e o seu processamento

4.4 *WebServices* – Comunicação entre Dispositivos

A comunicação entre os dispositivos e o servidor segue uma arquitetura baseada em redes colaborativas.

O diagrama abaixo ilustra o fluxo de funcionamento:

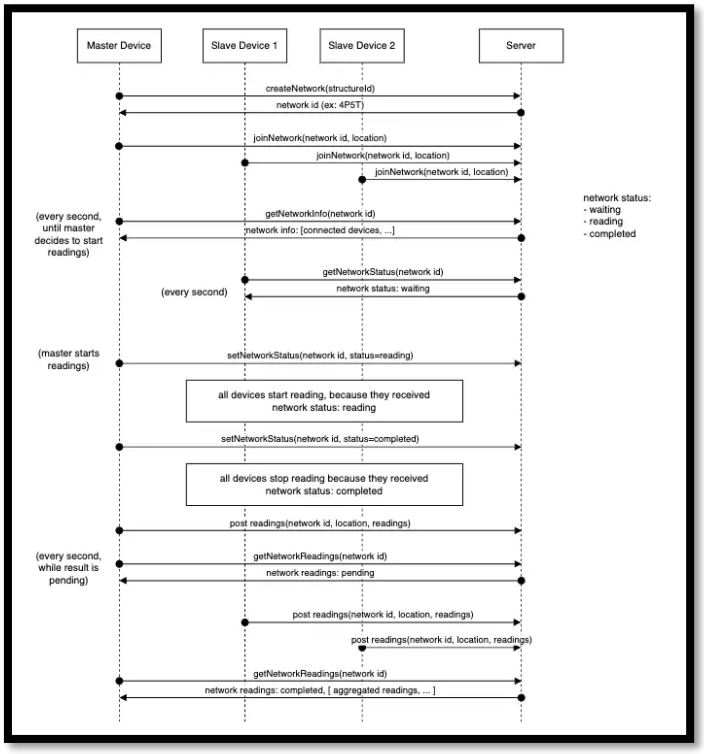


Figura 11 - Diagrama de Sequência (Comunicação entre Dispositivos)

Legenda:

- ***Master Device*:** dispositivo que cria e controla a sessão de leitura.
- ***Slave Device*:** dispositivos que participam na leitura em rede.
- ***Server*:** *backend* responsável pela sincronização, persistência e agregação de dados.

Estados da rede:

- ***waiting*:** antes de iniciar a leitura.
- ***reading*:** durante a leitura.
- ***completed*:** leitura finalizada e resultados disponíveis.

Do Anexo 21 ao Anexo 27, é possível encontrar uma descrição detalhada do funcionamento de cada um dos *WebServices*, permitindo compreender de forma clara os processos envolvidos em cada etapa da comunicação entre os dispositivos.

4.5 Tecnologias e Ferramentas Utilizadas

As principais tecnologias utilizadas para o desenvolvimento são:

- **Frontend (Aplicação Móvel)**:
    - **Flutter**: *Framework* multiplataforma para desenvolvimento rápido e eficiente.
    - **Dart**: Linguagem de programação usada no Flutter.
- **Backend (Servidor)**:
    - **Django**: *Framework web* em Python para desenvolvimento robusto e escalável.
    - **MySQL**: Sistema de base de dados relacional para armazenamento persistente.
- **Comunicação**:
    - **APIs REST**: Para troca de dados entre cliente e servidor.
- **Outras ferramentas**:
    - **Postman**: Testes de APIs.
    - **PyCharm** e **Android Studio**: Ambientes de desenvolvimento integrados.

4.5.1 Ambientes de Teste e de Produção

Ambiente de Teste:

- Servidor local configurado para simular operações.
- Dispositivos móveis com versões de desenvolvimento da aplicação.

Ambiente de Produção:

- Servidor Linux nas instalações da Universidade Lusófona.
- Base de dados MySQL.
- Publicação da aplicação nas lojas Google Play e App Store.

4.6 Abrangência

O projeto utiliza conhecimentos de várias áreas do curso, incluindo:

- **Engenharia de Software**: Para análise de requisitos e modelagem do sistema.
- **Computação Móvel**: No desenvolvimento da aplicação multiplataforma.
- **Redes e Computação Distribuída**: Para comunicação via APIs REST.
- **Algoritmia e Estrutura de Dados**: Na implementação de algoritmos de cálculo de forças e tensões.
- **Programação Web:** Desenvolvimento em Django.

4.7 Componentes

Os principais componentes da solução são:

4.5.1 Componente 1: Correção de Bugs

Este componente concentra-se em identificar e corrigir os problemas relatados pelos utilizadores em versões anteriores. O processo inclui:

- Análise de *logs* de erro e relatórios.
- Implementação de correções.
- Testes regressivos para validar a estabilidade.

4.5.2 Componente 2: Medição de Tirantes

Este componente envolve:

- Desenvolvimento de algoritmos para calcular forças e tensões a partir de dados de vibração.
- Integração dos algoritmos com a aplicação móvel e o servidor.
- Criação de uma interface gráfica intuitiva para visualização dos resultados.

4.5.3 Componente 3: Conexão de Múltiplos Dispositivos

Este componente permite que vários dispositivos sejam sincronizados para medições colaborativas. Inclui:

- Implementação de APIs para criação e gestão de sessões colaborativas.
- Desenvolvimento de um painel na aplicação para monitorização das conexões.
- Sincronização de leituras com desvio máximo de 20 ms.

4.8 Interfaces

A aplicação móvel apresenta as seguintes interfaces:

- **Ecrã de Medição de Tirantes**:
    - Exibe os valores calculados de força e tensão em tempo real.
    - Inclui gráficos para acompanhamento visual das medições.
- **Dashboard de Sessões Colaborativas**:
    - Lista dispositivos conectados e seu *status*.
    - Exibe resultados consolidados das medições.
- **Relatórios de Medição**:
    - Permite exportar resultados em formatos CSV ou PDF.

5 Testes e Validação

O principal objetivo dos testes desenvolvidos foi garantir que a solução proposta cumpre os seus objetivos fundamentais:

- Interpolar e alinhar leituras de sensores com precisão temporal.
- Permitir a análise espectral fiável (via *Welch*) de dados acelerométricos.
- Contribuir para a monitorização de integridade estrutural ou comportamental de dispositivos, sensores ou sistemas físicos em ambientes reais.

Para isso, foram desenhados testes funcionais, operacionais e de validação quantitativa, suportados por dados reais e pela simulação de cenários representativos.

5.1 Abordagem de Testes

A estratégia de validação foi dividida em três categorias principais:

Tabela 3 - Tipos de Testes

Tipo de Teste	**Objetivo**
Funcionais	Verificar o correto comportamento das funções desenvolvidas (interpolação, FFT).
Operacionais	Validar alinhamento e integração de múltiplos *streams* de sensores.
Espectrais (FFT)	Avaliar precisão da análise de frequências via método de Welch.

Sempre que possível, foi utilizada uma abordagem comparativa, confrontando os resultados esperados (calculados manualmente ou por referência externa) com os gerados pela solução.

5.2 Plano de Testes

Para garantir uma cobertura das funcionalidades desenvolvidas, foi definido um plano de testes composto por vários testes unitários e funcionais. Abaixo estão descritos, passo a passo, os testes realizados.

5.2.1 NetworkListViewTest

Estes testes fazem parte da camada de *backend* da aplicação, desenvolvida em Python com o Django REST Framework, e validam funcionalidades relacionadas à criação, entrada, consulta, desconexão e eliminação de redes (*Network*), um elemento essencial na estrutura da app.

test_create_network

O que faz: Envia um pedido *POST* para o *endpoint* de criação de rede (create-network), associando-a a uma estrutura existente.

O que valida:

- Se o endpoint cria corretamente uma rede (HTTP_201_CREATED).
- Confirma que os dados mínimos obrigatórios (ID da estrutura) são suficientes para criação.

test_join_network

O que faz: Envia um pedido *POST* para o endpoint join-network, simulando um utilizador a juntar-se a uma rede existente, indicando a localização.

O que valida:

- Se o sistema permite associar corretamente uma posição (*location*) a uma rede.
- Se o código de resposta é HTTP_200_OK, indicando sucesso.
- Se a resposta inclui o ID da posição onde o utilizador foi associado ("join position id"), confirmando persistência dos dados.

test_get_network_info

O que faz: Envia um pedido *GET* ao *endpoint* network-info com o ID codificado da rede.

O que valida:

- Se a API devolve corretamente informações da rede.
- Se os campos esperados estão presentes: "locations", "structure_positions_count", "positions_count".

- Valida a integridade e formato dos dados devolvidos.

test_disconnect_network

O que faz: Envia um pedido *POST* para o *endpoint* disconnect-network, removendo uma posição específica de uma rede.

O que valida:

- Se o sistema remove corretamente a ligação entre o utilizador e a rede.
- Se o código de resposta é HTTP_200_OK.
- Se a resposta indica qual a localização removida ("deleted position location"), o que demonstra clareza e feedback ao utilizador.

test_delete_network

O que faz: Envia um pedido *DELETE* ao *endpoint* delete-network, com o ID codificado da rede a eliminar.

O que valida:

- Se o sistema permite eliminar uma rede.
- Se o código de resposta é HTTP_200_OK, confirmando sucesso.
- Este teste também implica que o sistema trata corretamente as dependências ligadas à rede.

Configuração dos testes (setUpTestData e setUp)

- **setUpTestData**: Cria dados iniciais no início da execução dos testes (utilizador, estrutura, rede e posições), partilhados por todos os testes. Isto garante consistência nos dados.
- **setUp**: Autêntica automaticamente o cliente com o utilizador master para que todos os pedidos sejam autenticados, simulando uma sessão real de utilizador.

5.2.2 ComplexNetworkFlowTest

Estes testes simulam um fluxo complexo e completo de utilização da funcionalidade de redes da aplicação, validando tanto os caminhos normais como situações de erro. Testam cenários com múltiplos utilizadores (dispositivos) interagindo com uma mesma rede em diferentes posições.

test_complex_network_flow

Este teste representa um caso de uso completo com interações encadeadas, simulando o ciclo de vida de uma rede desde a criação até à recolha de leituras.

Etapas testadas:

1. **Criação da rede**
    - *POST* create-network
    - Verifica se a rede é criada corretamente (*HTTP* 201) e se o ID é retornado.
2. **Consulta inicial de estado**
    - *GET* network-info
    - Confirma que a rede foi criada com 0 posições ligadas.
3. **Ligação de três dispositivos a posições distintas**
    - Cada utilizador (*device1*, *device2*, *device3*) junta-se a uma *location* (1, 2 e 3).
    - Confirmações de sucesso (*HTTP* 200 *OK*).
4. **Verificação do número de posições ligadas**
    - Confirma que há agora 3 posições ligadas na rede.
5. **Desligar um dispositivo**
    - device3 desliga-se da *location* 3.
    - A rede passa a ter 2 posições ligadas.
6. **Religação**
    - device3 volta a juntar-se à rede, devolvendo o total para 3.
7. **Alterar o estado da rede para “reading”**
    - Define o início da fase de recolha de dados com uma startDate.
8. **Início da recolha de leituras**
    - Cada dispositivo cria uma *TimeSeries* (série temporal de dados).
    - Isto simula o envio real de dados sensores para a rede.

9. **Fechar a rede para leituras**
    - Muda o estado para *completed*, marcando o fim da recolha.
10. **Inserção de leituras**
    - Cada dispositivo envia leituras (*POST* network-readings) com sucesso (*HTTP* 201).

O que valida:

- Funcionamento de todos os *endpoints* envolvidos no fluxo da rede.
- Correta autenticação e controlo de acessos.
- Estados corretos da rede ao longo do tempo.
- Integração entre entidades: *Network*, *StructurePosition*, *User*, *TimeSeries*.
- Verificação de contagens (positions_count, etc.).

test_network_flow_with_errors

Este teste cobre cenários de erro e exceção, testando a robustez e resposta do sistema a dados inválidos ou incoerentes.

Etapas testadas:

1. **Criação de rede com structureId inválido**
    - Esperado: *HTTP* 404.
2. **Ligação a *location* inexistente**
    - Esperado: *HTTP* 404.
3. **Ligação a uma *location* já ocupada**
    - Esperado: *HTTP* 404.
4. **Desligar *location* inválida ou não conectada**
    - Esperado: *HTTP* 404.
5. **Consulta de rede inexistente**
    - *GET* network-info, *GET* network-status, *POST* network-status com IDs inválidos: *HTTP* 404.
6. **Envio de leitura com *reading* inválido**
    - Esperado: *HTTP* 400 *BAD REQUEST*.

7. **Consulta de leituras de rede inexistente**
    - Esperado: *HTTP* 404.

O que valida:

- Tratamento de erros e mensagens apropriadas.
- Verificação de estados incorretos e respostas consistentes da API.
- Robustez contra dados inválidos ou tentativas de utilização incorreta.

Observações Técnicas

- Todos os testes usam force_authenticate() para simular sessões autenticadas de diferentes utilizadores (master e dispositivos).
- A criação de *TimeSeries* é feita diretamente no modelo, sugerindo que os *endpoints* de leitura foram testados mais pela submissão do que pela criação da série.
- Uso correto de id_to_code e code_to_id, o que demonstra a existência de um sistema interno de codificação para identificação segura de redes.

5.2.3 NetworkDiagramTest

Este teste verifica o fluxo de dados da rede com foco na geração do diagrama lógico da rede, ou seja, como diferentes dispositivos se conectam a posições e enviam leituras.

test_diagram

Este teste valida o comportamento visual/lógico da rede simulando a ligação dos dispositivos a posições fixas de uma estrutura e o envio de leituras ao longo do tempo. Ele cobre:

- Criação de rede
- Ligação dos dispositivos
- Alteração do estado da rede
- Envio de leituras
- Verificação do estado da rede e leitura completa ou não (all_done)

Etapas testadas:

1. **Criação da rede**

- POST create-network
- Confirma se a rede foi criada com sucesso (HTTP 201) e obtém o ID codificado (networkId).

2. **Ligação dos utilizadores às posições**
   - master → location 1
   - device1 → location 2
   - device2 → location 3
   - A cada ligação, confirma-se o incremento da contagem de posições conectadas.
3. **Verificação do estado da rede**
   - GET network-status: deve estar em "waiting" antes de começar a leitura.
4. **Alteração do estado para *reading***
   - Define o estado de leitura com uma data de início.
5. **Criação das leituras (*TimeSeries*) por cada utilizador**
   - Cada dispositivo cria uma leitura associada à estrutura.
6. **Alteração do estado para *completed***
   - Fecha a janela de recolha de dados.
7. **Submissão das leituras por posição**
   - POST network-readings para cada *location*.
8. **Verificação do progresso das leituras**
   - Após cada envio, verifica-se se all_done já foi alcançado.
   - Após as duas primeiras leituras, o sistema ainda indica "pending".

O que valida:

- Correta codificação e decodificação de networkId.
- Ligação ordenada e sincronizada de múltiplos dispositivos às posições da rede.
- Transições de estados da rede: *waiting* → *reading* → *completed*.
- Submissão de leituras em sequência e verificação do progresso.
- Comportamento incremental do estado all_done, útil para visualizações (como diagramas na interface).

5.2.4 NetworkReadingsMultiUserTest

Este teste valida o comportamento da API quando muitos utilizadores (ou dispositivos) submetem ficheiros de leitura associados a diferentes posições de uma rede, garantindo que as leituras são corretamente processadas e agregadas.

test_network_readings_aggregation

Dois dispositivos (*users*) carregam ficheiros .txt com dados estruturais e associam-nos a duas posições distintas de uma rede. No final, o teste avalia se os dados agregados (frequências e eixos) estão corretamente disponíveis.

Etapas testadas:

1. **Criação da rede**
    - Utilizador device1 cria a rede e autentica-se.
2. **Criação de posições na estrutura**
    - São definidas 2 localizações fixas para a estrutura (*location* 1 e *location* 2).
3. **Associação de posições à rede**
    - As posições são associadas explicitamente à rede (via *Position*).
4. **Upload de ficheiros com dados (*readings*)**
    - Dois ficheiros .txt distintos são enviados para o *endpoint* api-readings, simulando medições dos dois dispositivos.
5. **Fecho da rede (*status* = *completed*)**
    - A rede é encerrada para marcação do fim da recolha de dados.
6. **Associação das *readings* às posições**
    - Cada leitura é associada à sua posição com base na *location*.
7. **Verificação da agregação final dos dados**
    - GET network-readings retorna:
        - Lista de localizações.
        - all_done: *completed*
        - Dados agregados: *frequencies*, *x*, *y*, *z*.

O que valida:

- Upload e *parsing* correto dos ficheiros de leitura .txt.

- Mapeamento entre *readings* e *locations* dentro da rede.
- A estrutura dos dados agregados está completa (todas as dimensões presentes).
- O sistema reconhece que todas as leituras foram submetidas, retornando *completed*.

5.2.5 WelchTestCase

test_interpolate_data_stream_aligned_d1

Valida se os dados de leitura do dispositivo 1 são corretamente interpolados para timestamps regulares (intervalo fixo de 20ms).

Etapas testadas:

- Lê o ficheiro sample_readings_md_d1.txt.
- Aplica a função interpolate_data_stream_aligned.
- Compara os dados interpolados com uma sequência esperada manualmente construída (expected_lines).

O que valida:

- Mesmo número de pontos interpolados e esperados.
- Cada ponto é comparado (timestamp, x, y, z) com tolerância de erro mínima (assertAlmostEqual).

test_interpolate_data_stream_aligned_d2

Idêntico ao teste anterior, mas com dados do dispositivo 2.

Etapas testadas:

- Usa sample_readings_md_d2.txt.
- Interpola e compara com expected_lines.

O que valida:

- Confirma interpolação correta mesmo em caso de mudanças abruptas nos dados (verificados nos últimos timestamps).

test_align_multiple_streams

Testa o alinhamento e agregação de múltiplos *streams* interpolados de sensores (d1 e d2), seguido do cálculo da média das densidades espectrais (via Welch).

Etapas testadas:

- Lê os dois ficheiros de leitura bruta (d1 e d2).
- Alinha os *streams* com align_multiple_streams.
- Calcula médias espectrais com calculate_mean_welch_frequencies.

O que valida:

- As frequências médias (avg_freq) e os espectros (x, y, z) são comparados com valores esperados calculados manualmente.
- Usa np.allclose com tolerâncias adequadas para lidar com ruído numérico.

6 Método e Planeamento

Este capítulo apresenta o método de trabalho seguido no desenvolvimento do projeto, focando-se, nesta fase, no planeamento inicial.

6.1 Planeamento Inicial

O planeamento inicial foi elaborado com base no cronograma apresentado (Figura 39) e seguiu uma abordagem estruturada, utilizando um gráfico de Gantt para organizar e acompanhar as tarefas. A gestão do projeto focar-se-á em garantir o cumprimento dos prazos e a distribuição eficiente do esforço para a implementação dos requisitos descritos.

O cronograma organiza o projeto em 8 fases principais, distribuídas ao longo de 7 meses. Cada etapa corresponde a tarefas de desenvolvimento, testes ou documentação, conforme o objetivo de cada componente:

Plano de Trabalho

1. **Correção de Erros e Bugs** (2 semanas, Alta prioridade)
   - Esforço: focados na estabilidade do sistema.
2. **Prototipagem e Implementação de Medições de Tirantes** (8 semanas no total, Alta prioridade)
   - Esforço: 50% para protótipos e 50% para implementação definitiva.
3. **Medições Colaborativas** (10 semanas, Alta prioridade)
   - Prototipagem (2 semanas): Iteração inicial, testes colaborativos.
   - Implementação (8 semanas): Foco em sincronização e testes de carga.
4. **Interface Gráfica Colaborativa** (2 semanas, Média prioridade)
   - Esforço: Design intuitivo, baseado no feedback dos utilizadores.
5. **Fase de Testes** (5 semanas, Alta prioridade)
   - Esforço: Testes abrangentes com documentação de resultados.
6. **Relatório Final** (3 semanas, Média prioridade)
   - Consolidar trabalho, validar requisitos e apresentar conclusões.

Para garantir a robustez do planeamento e minimizar os impactos de eventuais imprevistos, foram identificados os principais riscos associados ao projeto e definidas estratégias de mitigação adequadas:

Tabela 4 - Análise de Riscos e Estratégias de Mitigação

Risco Identificado	Impacto Potencial	Probabilidade	Mitigação
Atrasos na implementação dos protótipos	Efeito cascata nas fases seguintes	Média	Estabelecer marcos intermédios e avaliações semanais de progresso.
Problemas técnicos na integração das medições	Comprometimento da funcionalidade principal	Baixa	Reservar tempo extra para testes e iterações rápidas
Dificuldades de sincronização na colaboração remota	Redução de eficiência nos testes colaborativos	Alta	Utilizar ferramentas colaborativas robustas e promover reuniões regulares de alinhamento.
Falhas na recolha de feedback dos utilizadores	Interface pouco intuitiva ou mal-adaptada	Baixa	Implementar sessões curtas de teste com utilizadores após cada iteração.
Subestimação do tempo necessário para testes finais	Comprometimento da qualidade final do produto	Média	Alocar uma semana adicional de contingência dentro da fase de testes.

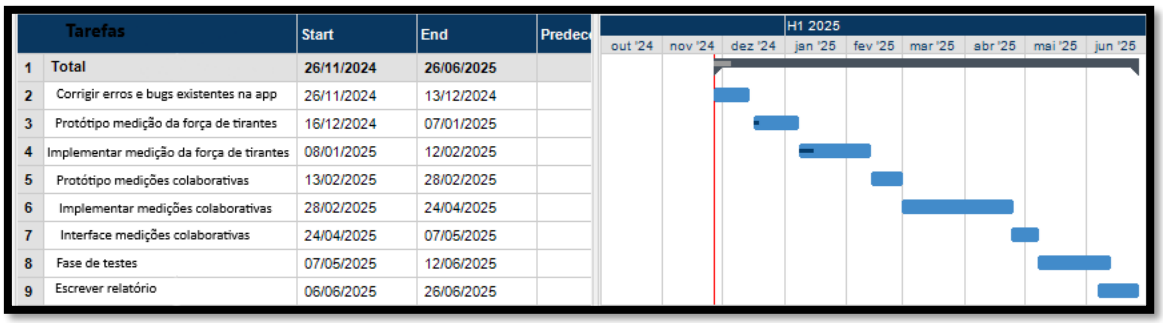


Figura 12 - Calendário de Gantt

6.2 Funcionalidades Implementadas e Grau de Complexidade

Tabela 5 - Requisitos e Descrição

ID	Descrição do Requisito	Subcomponentes / Tarefas	Estado	Dificuldade
R1	Correção de Erros Detetados	Erro gaussiana sem *mean_forces* (server); Formato inconsist. freq. input/output; Extensão .csv em vez de .txt; Botão "next" inativo; Melhorias UI múltiplos dispositivos; Gráfico da curva Gaussiana cortado	Implementado	Média
R2	Medição da força dos tirantes	Cálculo da força com base nas frequências naturais; Algoritmo validado por docentes; Armazenamento e visualização na app	Implementado	Alta
R3	Suporte a medições colaborativas com múltiplos dispositivos	Criação de "network" de dispositivos; Sincronização com timestamps; Armazenamento das medições; Interface e ligação entre dispositivos	Implementado	Muito Alta
R4	Interface gráfica para sessões colaborativas	- Criação/entrada em sala; Seleção de posição; Estado das ligações; *DropDowns* dinâmicas; Informações da rede e dos dispositivos conectados	Implementado	Média

Correção de Bugs

Esta foi a primeira tarefa que realizámos no projeto, e revelou-se um dos maiores desafios iniciais, dado que se tratava do nosso primeiro contacto com a aplicação. Corrigimos bugs deixados por alunos de anos anteriores, o que exigiu um esforço significativo para compreender a estrutura do código existente e o funcionamento geral da aplicação.

Apesar das dificuldades, esta fase foi fundamental para consolidar a nossa compreensão sobre o fluxo da app, o seu funcionamento interno e a forma como os diferentes módulos interagem.

Complexidade: média, com elevada importância formativa para o resto do projeto.

Medição da Força dos Tirantes

A medição da força dos tirantes reutiliza parte da lógica de leitura de frequências usada noutras funcionalidades da aplicação. No entanto, esta tarefa teve uma complexidade superior por envolver fórmulas e conceitos físicos que não dominávamos completamente.

Além disso, foi a primeira vez que desenvolvemos uma nova funcionalidade de raiz, incluindo o design da interface, a construção dos gráficos e a ligação com a API. Esta fase foi particularmente exigente por termos poucos conhecimentos prévios de Flutter na altura — dependíamos principalmente de cursos online (como NetNinja e vídeos do professor Pedro Alves). A dificuldade prendeu-se não só com a implementação da lógica, mas também com a interpretação dos dados e validação dos resultados.

Complexidade: alta, especialmente devido ao desconhecimento do domínio físico e à inexperiência com Flutter na fase inicial.

Suporte a Medições Colaborativas com Múltiplos Dispositivos (Python)

Esta foi a fase mais longa e tecnicamente exigente de todo o projeto. Implementámos suporte para sessões de medição colaborativa entre vários dispositivos, focando inicialmente apenas no *backend* em Python, como recomendado pelo nosso orientador.

A implementação incluiu múltiplas *views* e *endpoints*, bem como a gestão de estados entre diferentes dispositivos (*master* e *slaves*). Realizámos testes unitários exaustivos que abrangeram vários cenários e casos de erro, garantindo a robustez da solução antes de iniciar a integração com a interface.

Com a ajuda do professor Pedro Alves, que realizou o diagrama de sequência, o que deu um grande impulso ao nosso projeto. Não começámos às cegas.

Complexidade: muito alta, devido à necessidade de gerir estados concorrentes, lógica distribuída e múltiplas camadas de comunicação.

Interface Gráfica para Sessões Colaborativas (Flutter)

Após o *backend* estar completamente funcional, passámos para a implementação da interface gráfica no Flutter. Nesta fase, já tínhamos uma base sólida em Flutter, fruto da disciplina de Computação Móvel e da experiência acumulada ao longo do projeto.

A maior dificuldade residiu em conceptualizar e implementar uma interface que suportasse a interação simultânea de múltiplos utilizadores, distinguindo claramente os papéis (*master* vs *slaves*).

Além disso, tivemos de otimizar a performance da aplicação, dado que certas páginas — como a de séries temporais — fazem chamadas ao servidor a cada segundo para atualizar o estado das medições em tempo real.

Complexidade: média, sobretudo pela necessidade de coordenar múltiplos utilizadores e otimizar o desempenho da aplicação.

6.3 Uso de *Pull Requests* e *Issues* no GitHub

Durante o desenvolvimento do projeto final de curso, adotámos uma metodologia de trabalho colaborativa baseada nas funcionalidades de controlo de versões oferecidas pelo GitHub. Para facilitar a organização e integração das contribuições, cada membro da equipa trabalhou em *forks* (cópias independentes) dos repositórios principais, tanto do servidor como da aplicação.

As alterações realizadas foram regularmente *commits* e *pushes* para os respetivos *forks*, permitindo manter um histórico claro e separado das contribuições individuais. Sempre que uma funcionalidade era concluída ou uma correção estava pronta, submetíamos um *Pull Request* (PR) para o repositório principal.

Este processo permitiu:

- Revisão de código: Antes de ser integrado no repositório principal, cada PR podia ser revisto e discutido, promovendo boas práticas de programação e deteção precoce de erros.
- Integração controlada: Apenas os PRs aprovados eram incorporados no projeto principal, garantindo maior estabilidade e qualidade do código final.

Além disso, fizemos uso ativo das *Issues* no GitHub, que serviram para registar bugs, sugestões de melhorias e tarefas pendentes. Algumas *Issues* foram abertas pelos utilizadores do projeto, enquanto outras foram criadas pelo professor orientador. À medida que resolvíamos os problemas identificados, referenciávamos as *Issues* nos *commits* e *PRs* correspondentes, o que facilitou o acompanhamento do progresso e a rastreabilidade das soluções implementadas.

Esta abordagem, centrada no uso de *Pull Requests* e *Issues*, permitiu uma colaboração eficiente, documentada e transparente, alinhada com as práticas modernas de desenvolvimento de *software open source*.

7 Resultado

7.1 Resultado dos Testes

Feedback do Professor após Teste em Campo da Aplicação App4SHM.

No dia 27/03/2025, o professor Inout realizou um teste prático da funcionalidade de cabos na Ponte Edgar Cardoso. De acordo com o seu relato, a aplicação teve um desempenho satisfatório durante os ensaios, apresentando-se funcional e eficiente na recolha e análise dos dados de força nos cabos.

O professor destacou positivamente a organização do back-office, referindo que a estrutura da informação está bem concebida e faz sentido no contexto do trabalho desenvolvido.

No entanto, identificou dois pontos de melhoria:

- **Exportação em PDF**: Foi detetado um problema com o enquadramento das figuras nos relatórios em PDF. Em alguns casos, os gráficos não são devidamente ajustados ao layout, comprometendo a clareza visual dos resultados, problema esse que foi resolvido posteriormente.
- **Importação de Base de Dados**: O sistema ainda não possui uma funcionalidade para importar uma base de dados externa, o que poderá limitar a escalabilidade e integração com outros sistemas de gestão ou análise.

Apesar destas observações, o professor considerou a aplicação aprovada para os fins a que se destina.

7.2 Cumprimento de Requisitos

7.2.1 *Cable Force*

Após algumas semanas e com a ajuda do nosso orientador, conseguimos perceber com maior clareza o que deveria ser implementado. Iniciámos o desenvolvimento da página *Cable Force*, tanto no *frontend* como no *backend*.

No *backend*, o foco foi principalmente relacionar esta nova entidade, *Cable Force*, com as outras entidades existentes. Criámos também *WebServices* para possibilitar a sua utilização no *frontend*.

O resultado no *frontend* é apresentado na Figura 13, com um pequeno pormenor: o gráfico das forças ainda não é exibido, pois não há leituras suficientes. Isto é indicado pela mensagem em vermelho que aparece no centro do ecrã.

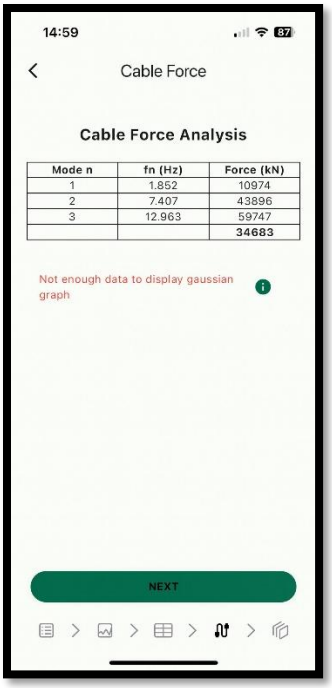


Figura 13 - Ecrã Cable Force, sem Gráfico

Na Figura 14, é possível ver quantas leituras faltam para que o gráfico seja exibido. Esta informação pode ser acedida ao clicar no símbolo de informação no ecrã *Cable Force*.

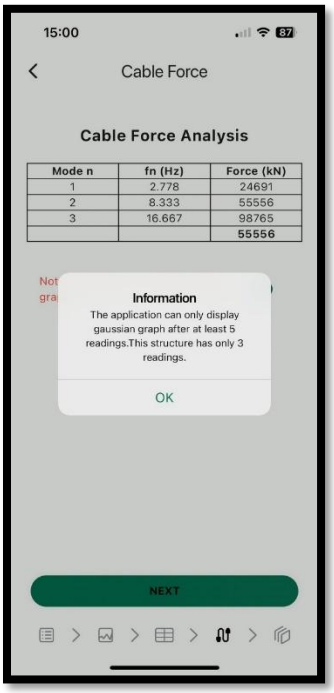


Figura 14 - Mensagem de Informação

Por fim, quando todas as leituras estiverem concluídas, o gráfico estará disponível para visualização (Figura 15). No gráfico, é possível identificar a leitura que acabamos de realizar, pois ela aparece com maior destaque.

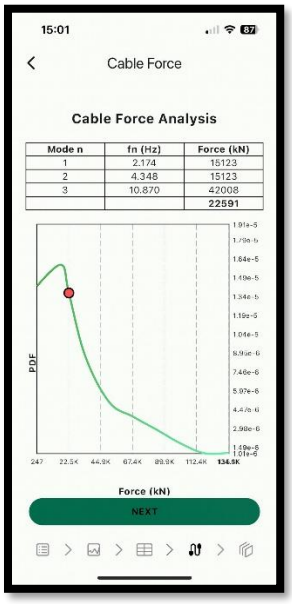


Figura 15 - Ecrã Cable Force, com Gráfico

Além disso, podemos navegar pelo gráfico para obter mais informações sobre as leituras, conforme mostrado na Figura 16.

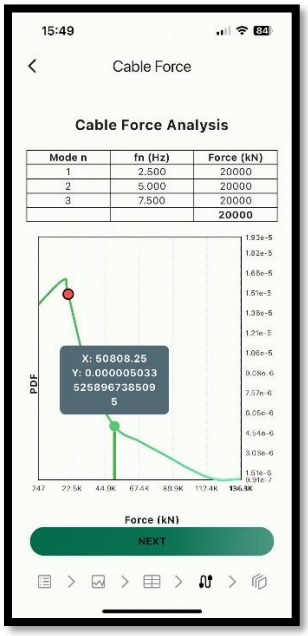


Figura 16 - Detalhes do Gráfico

7.2.2 Multiple Devices

Com o decorrer do projeto e o acompanhamento do nosso orientador, foi possível clarificar os requisitos a implementar. A partir desse momento, iniciámos o desenvolvimento das páginas relativas ao módulo *Multiple Devices*, abrangendo tanto a vertente de *frontend* como a de *backend*.

No *backend*, o principal foco consistiu na criação e associação das novas entidades, Network, *Structure Position* e *Position*. Adicionalmente, foram desenvolvidos *WebServices* que permitem a sua integração e utilização no *frontend*.

O resultado apresentado no *frontend* pode ser observado na Figura 17, onde é visível uma *CheckBox* acompanhada do texto "Multiple devices reading".

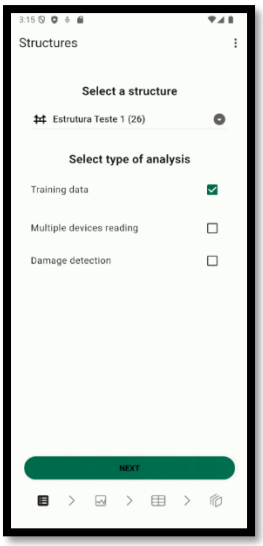


Figura 17 - Ecrã Structures com Multiple Devices

No canto superior direito do ecrã *Structures*, onde se encontram os três pontos, ao clicar nessa área surge uma nova opção com o nome "Join Structure", como é possível observar na Figura 18.

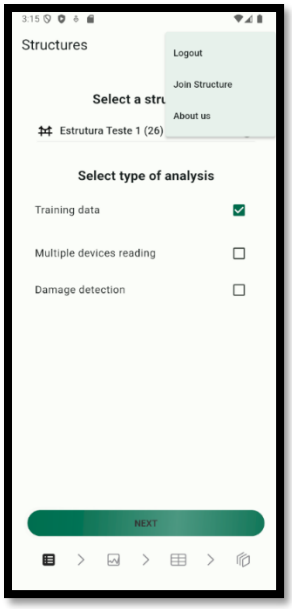


Figura 18 - Tês Pontos no Ecrã Structures

Ao selecionar a *CheckBox* com o texto "Multiple devices reading" e, de seguida, clicar no botão NEXT, somos encaminhados para uma nova página *Multiple devices configuration*, ilustrada na Figura 19. Nesta página, é apresentado o nome da estrutura, o identificador da *Network*, o respetivo código associado à *Network*, bem como uma *DropDown* que permite selecionar as diferentes posições disponíveis da estrutura. É possível ver também tês ícones, *Delete*, *Refresh* e *Continue*.

Nota que, o dispositivo que é direcionado para esta página assume automaticamente a função de dispositivo *Master*.

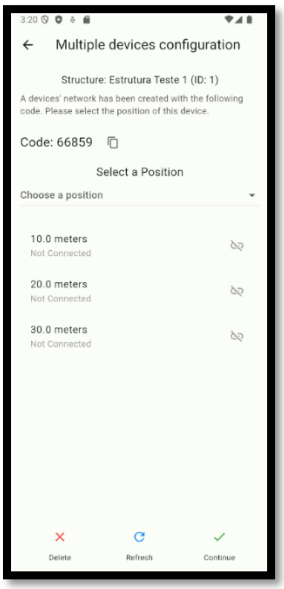


Figura 19 - Ecrã Master Multiple Devices

Ao clicar nos três pontos localizados no canto superior direito e, de seguida, selecionar a opção "Join Structure" (Figura 20), somos encaminhados para a página com o mesmo nome. Nesta nova página é apresentada uma *TextBox* que pede a introdução do código da *Network*, assim como um botão identificado com a palavra "Join".

Nota que, o dispositivo que é direcionado para esta página assume automaticamente a função de dispositivo *Slave*.

Figura 20 - Ecrã Join Structure

Após inserir o código de uma *Network* existente e clicar no botão *Join*, surge uma *DropDown* acompanhada da legenda "Select a Position", bem como um botão identificado como *Connect* (Figura 21).

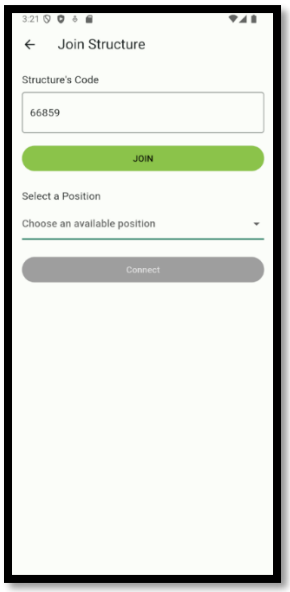


Figura 21 - Ecrã Join Structure, depois de Join

O conteúdo da lista suspensa, que corresponde às localizações disponíveis, é exibido ao clicar na mesma, como ilustrado na Figura 22.

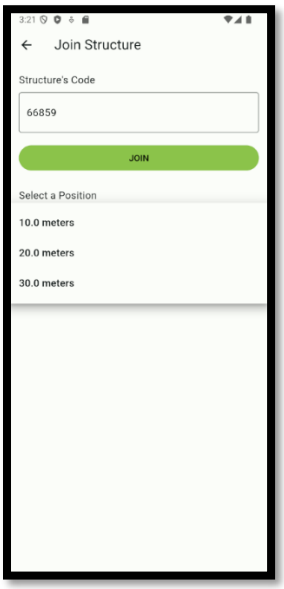


Figura 22 - Select a Position, Ecrã Join Structure

Após selecionar a posição, ocorre a ligação à *Network*, sendo exibida uma notificação em caso de sucesso, acompanhada de uma mensagem que surge no centro do ecrã, conforme ilustrado na Figura 23.

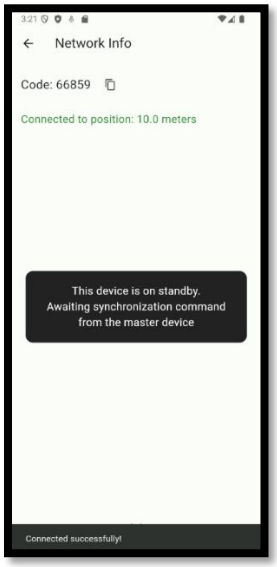


Figura 23 - Ecrã Join Structure, depois de Connect

No ecrã "Multiple devices configuration" se atualizar a página através do ícone de *refresh*, é possível verificar que outro dispositivo se conectou na posição designada como “10 meters”, conforme demonstrado na Figura 24.

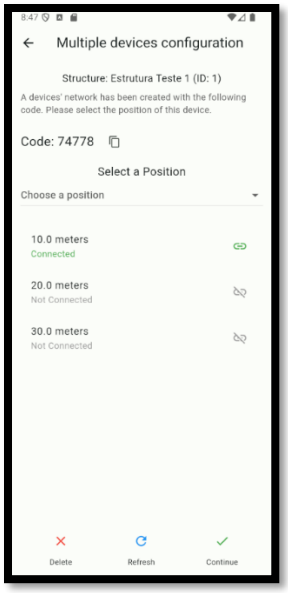


Figura 24 – Ecrã Multiple Devices Connected Positions

A *DropDown* no ecrã "Multiple devices configuration" é então atualizada, passando a apresentar apenas as posições que ainda não foram conectadas (Figura 25).

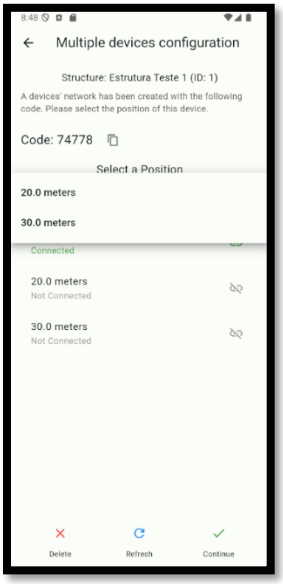


Figura 25 - Select a Position Ecrã Multiple Devices

Ao selecionar uma posição, na tabela das posições é indicado a que posição estamos ligados através da legenda "(Your device)", surgindo também um novo ícone que permite efetuar a desconexão, identificado como *Disconnect* (Figura 26).

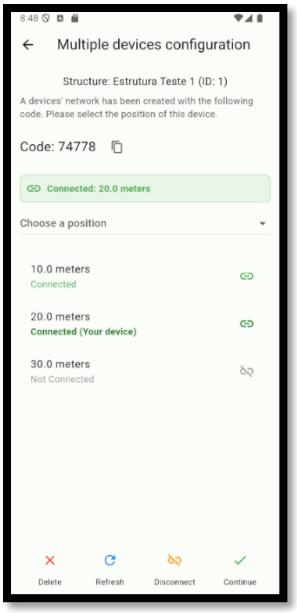


Figura 26 - Ecrã Multiple Devices, Connected Position

Ao clicar no ícone *Continue*, somos encaminhados para uma nova página denominada "Network Time Series", onde é possível visualizar a posição à qual estamos conectados, o estado das restantes posições que também se encontram ligadas, assim como o gráfico habitual da *Time Series*, como podemos ver na Figura 27.

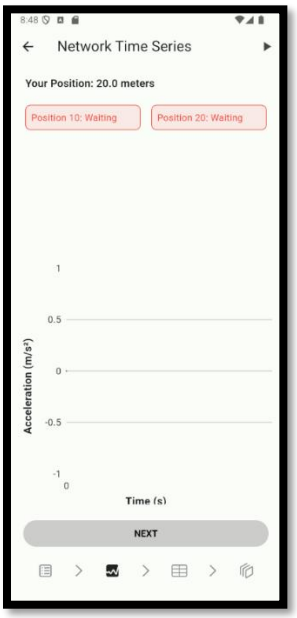


Figura 27 - Ecrã Network Time Series

Ao iniciar a leitura, ação que apenas pode ser realizada a partir do dispositivo *Master*, é possível observar alterações no gráfico em tempo real, refletindo as vibrações detetadas pelo acelerómetro do telemóvel, como ilustrado na Figura 28.

Nesta figura, é apresentado o ecrã tanto do dispositivo *Master* como do dispositivo *Slave*, evidenciando visualmente as diferenças entre ambos. O ecrã do *Slave* não apresenta a tabela com as posições nem os botões para iniciar ou parar a leitura, funcionalidades que estão exclusivamente disponíveis no dispositivo *Master*.

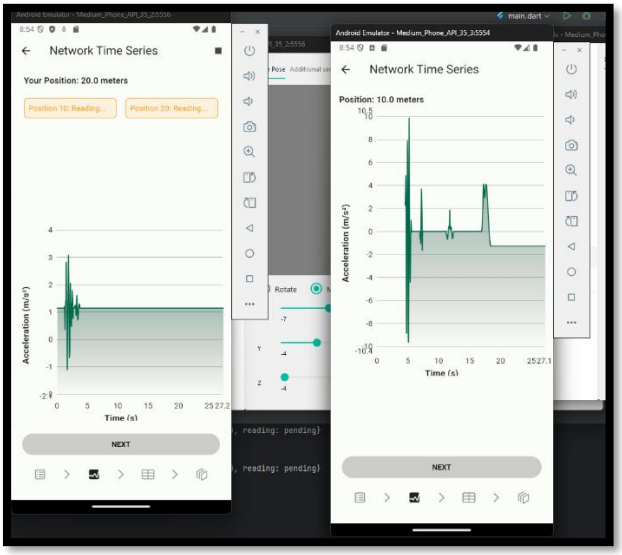


Figura 28 - Ecrã Network Time Series, Master and Slave

No final da leitura, ao clicar no botão *NEXT*, somos encaminhados para a página "Power Spectrum Network". Esta página apenas é exibida na totalidade após a receção dos cálculos realizados pelo servidor, apresentando também o estado de cada uma das posições conectadas, conforme ilustrado na Figura 29.

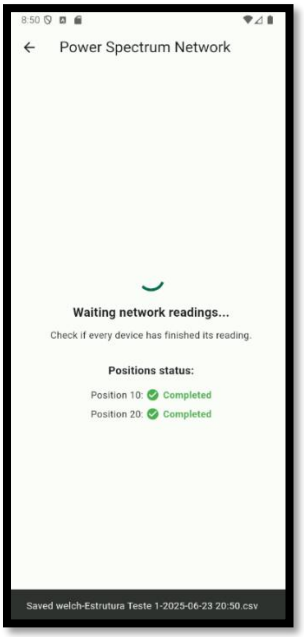


Figura 29 - Ecrã Power Spectrum Network, antes de Carregar Readings

Após a receção dos cálculos, é apresentado o gráfico correspondente à curva Gaussiana, tal como demonstrado na Figura 30.

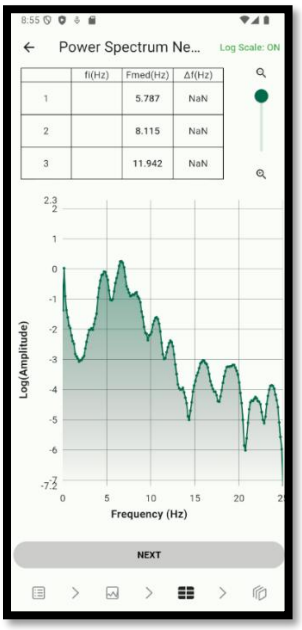


Figura 30 - Ecrã Power Spectrum Network, depois de Carregar Readings

8 Conclusão

8.1 Conclusão

No geral, ficámos bastante satisfeitos por termos conseguido cumprir todos os requisitos definidos no início deste projeto. Não foi um desafio fácil, por diversos motivos já mencionados anteriormente: trabalhámos com uma *Framework* completamente nova para nós, desenvolvemos serviços que seriam utilizados simultaneamente por vários utilizadores, e lidamos com cálculos e conceitos pouco familiares, tudo isto representou um conjunto de experiências que nunca tínhamos enfrentado antes.

Sentimo-nos também bastante orgulhosos da nossa autonomia ao longo do projeto. Embora o nosso professor orientador tenha demonstrado grande disponibilidade, algo de que usufruímos pontualmente, por exemplo, na modelação do servidor, procurámos sempre resolver os nossos problemas de forma independente. Acreditamos que esta abordagem reflete a realidade do mercado de trabalho, onde a proatividade e a capacidade de resolver problemas por conta própria são fundamentais. Só recorremos ao professor quando considerámos que a continuidade do projeto estava em risco.

Consideramos que tivemos muita sorte com o tema do projeto e com as pessoas envolvidas. Para além do nosso orientador, tivemos o privilégio de colaborar com a equipa de Engenharia Civil e com os professores Elói Figueiredo e Ionuț, que se mostraram sempre profissionais e disponíveis para esclarecer dúvidas relativas aos requisitos da sua área.

É relevante destacar que utilizamos ferramentas amplamente adotadas no ambiente profissional, como o GitHub, para a gestão e colaboração do projeto. Fizemos uso de *commits*, *pull requests* e *issues* para organizar o desenvolvimento, garantir a rastreabilidade do código e facilitar a revisão entre os membros da equipa, práticas essenciais em qualquer ambiente de trabalho moderno.

Temos plena consciência de que este projeto nos preparou melhor para o contexto profissional. Trabalhámos sobre uma base de código criada por alunos de anos anteriores, o que nos obrigou a compreender e adaptar código escrito por terceiros, uma experiência até então inédita para nós. Interagimos com “clientes” que não eram programadores, o que nos obrigou a melhorar a nossa capacidade de comunicação, compreendendo com clareza os requisitos discutidos em reunião. Além disso, desenvolvemos uma aplicação que será utilizada por dezenas de pessoas, o que exigiu um cuidado redobrado com a usabilidade e a experiência do utilizador.

8.2 Trabalhos Futuros

Atualmente, a aplicação não utiliza geolocalização durante a funcionalidade de medição em grupo. A integração desse recurso representaria uma melhoria significativa, pois permitiria aos participantes localizarem-se com maior precisão nas posições atribuídas, garantindo uma execução mais eficiente e coordenada da medição.

Outra limitação a ser considerada é a ausência de deteção automática de falhas nos dispositivos durante a medição em grupo.

Caso ocorra algum problema técnico (como perda de ligação ou falha no envio de dados), a aplicação atualmente não alerta o master nem tenta recuperar ou adaptar a medição.

A implementação de um sistema de monitorização e deteção de falhas, com notificações ou tentativas de reconexão automática, seria uma adição valiosa para tornar o processo mais robusto e fiável.

Bibliografia

DEISI, Regulamento de Trabalho Final de Curso, Out. 2024.

DEISI, www.deisi.ulusofona.pt, Out. 2024.

Tanenbaum, A. e Wetherall,D., *Computer Networks*, 6ª Edição, Prentice Hall, 2020.

Universidade Lusófona de Humanidades e Tecnologia, www.ulusofona.pt, acedido em Out. 2024.

Relatório Trabalho Final de Curso, http://informatica.ulusofona.pt/wp-content/uploads/sites/57/2022/10/TFC_21_DEISI135.pdf, acedido em Out. 2024.

Relatório Trabalho Final de Curso, http://informatica.ulusofona.pt/wp-content/uploads/sites/57/2022/10/TFC_22_DEISI219.pdf, acedido em Out. 2024.

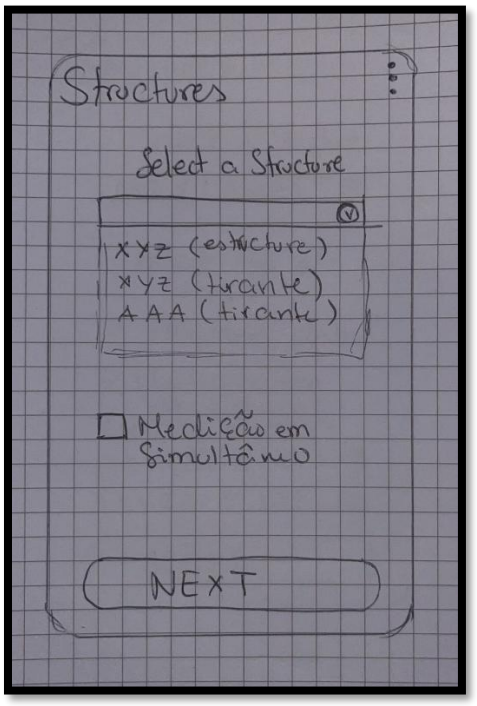
Linkedin autodes-force-effect, https://www.linkedin.com/pulse/autodesk-force-effect-calcule-vigas-e-tirantes-seu-fabricio/, acedido em Out. 2024.

Linkedin geokon, https://www.linkedin.com/company/geokon-usa, acedido em Out. 2024.

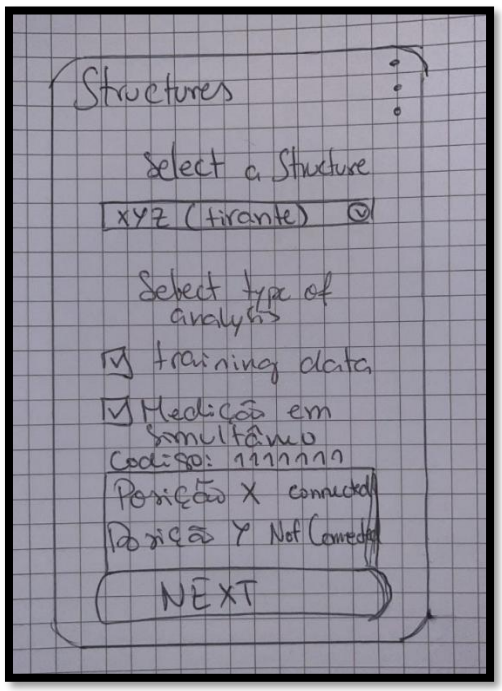
Linkedin trimble-geopatial, https://www.linkedin.com/showcase/trimble-geospatial/, acedido em Out. 2024.

Anexos

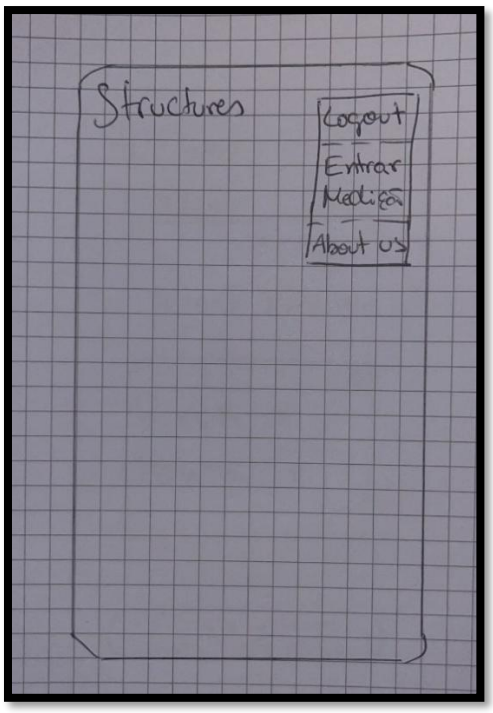
1. Ecrã *Structures* (*mockup*)



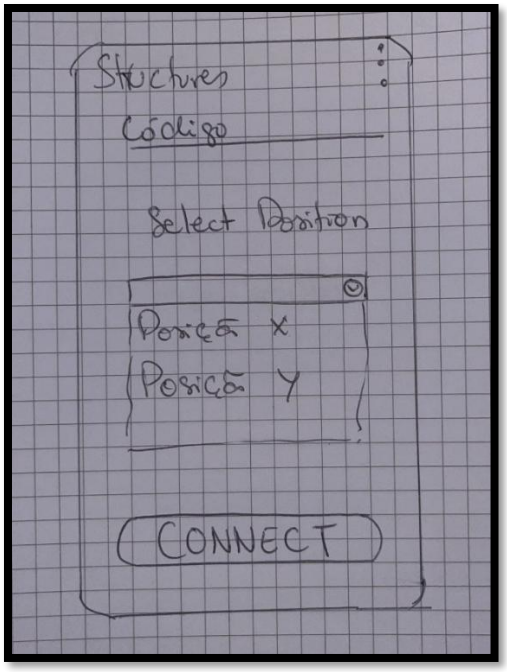
2. Medição em Simultâneo (*mockup*)



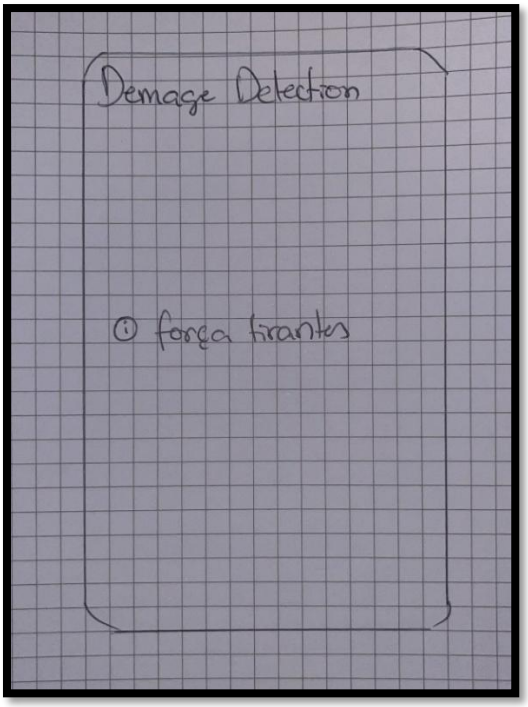
3. Opção para Entrar na Medição (*mockup*)



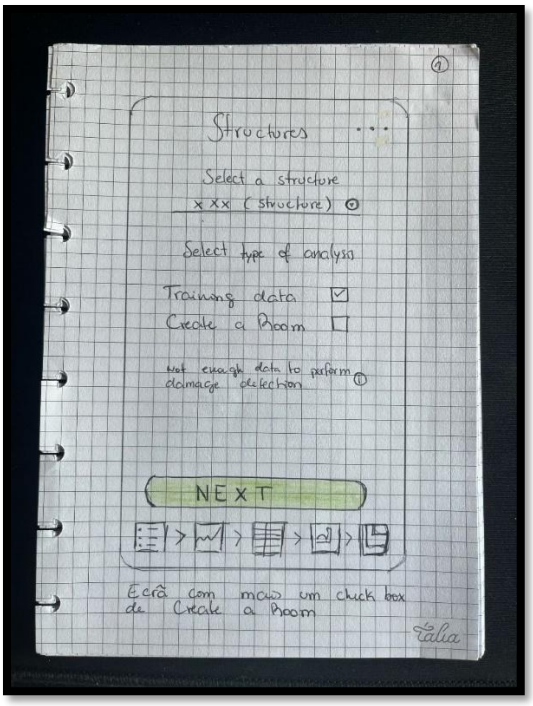
4. Ecrã *Slave* (*mockup*)



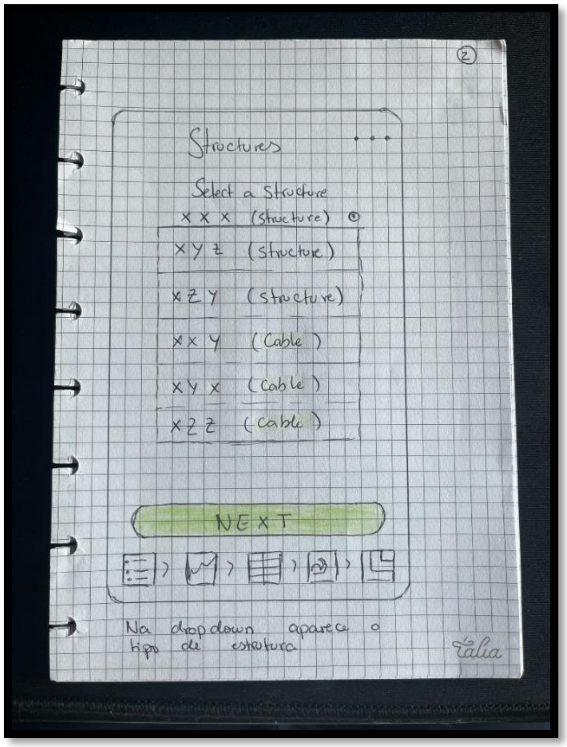
5. Forças dos Tirantes (*mockup*)



6. Ecrã *Structures* (*mockup*)

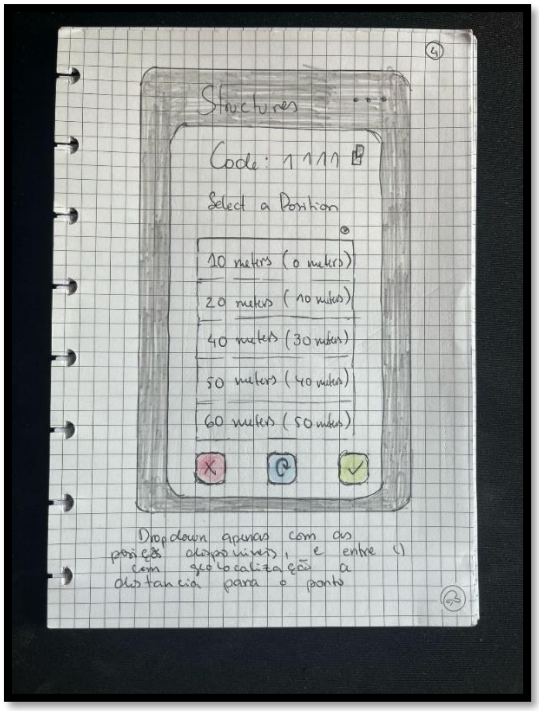


7. Ecrã *Structures*, *DropDown* (*mockup*)

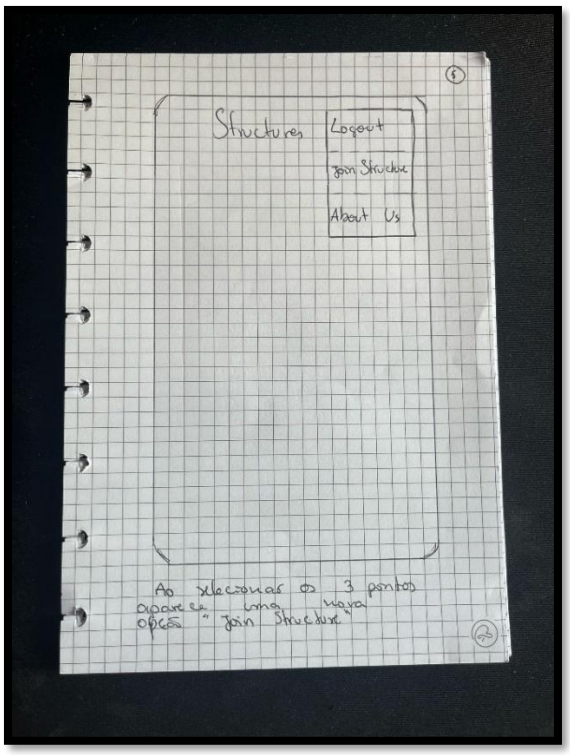


8. Novo Ecrã pós Create a Room (*mockup*)

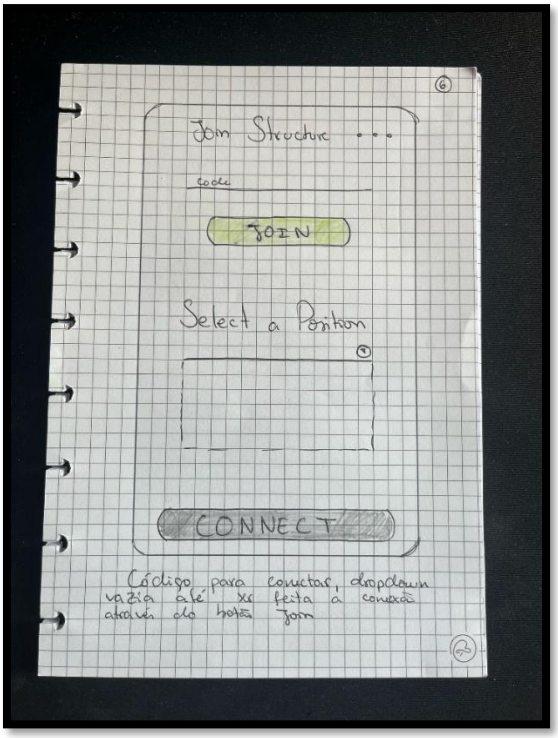
9. *DropDown* das Posições (*mockup*)



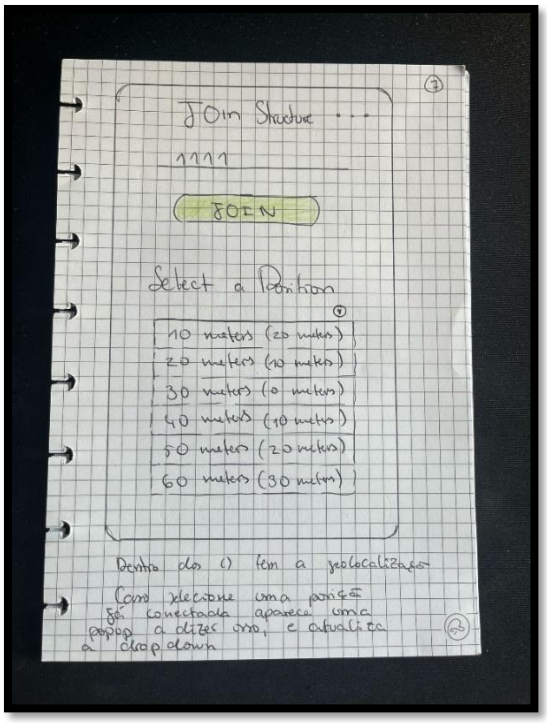
10. Três Pontos no Ecrã *Structures* (*mockup*)



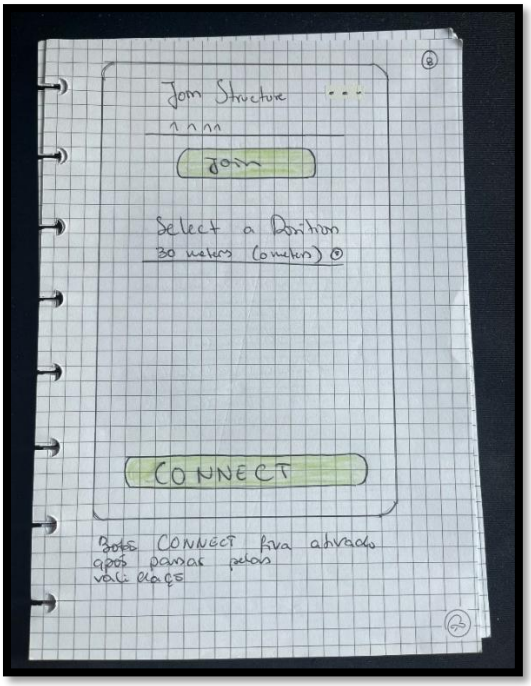
11. Ecrã *Join Structure* (*mockup*)



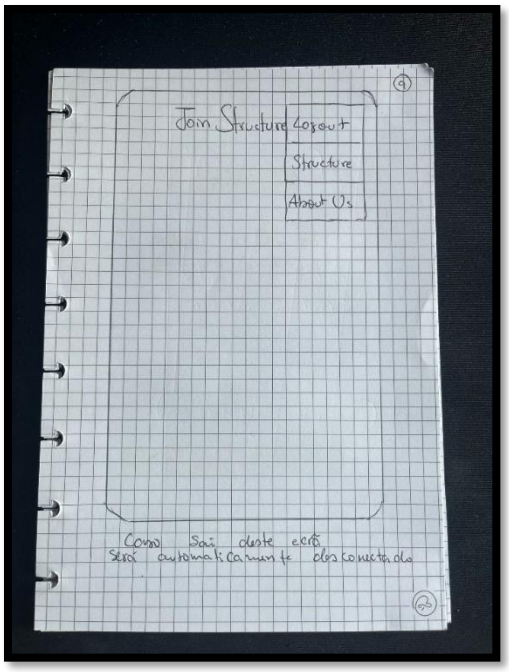
12. *DropDown* atualizada (*mockup*)



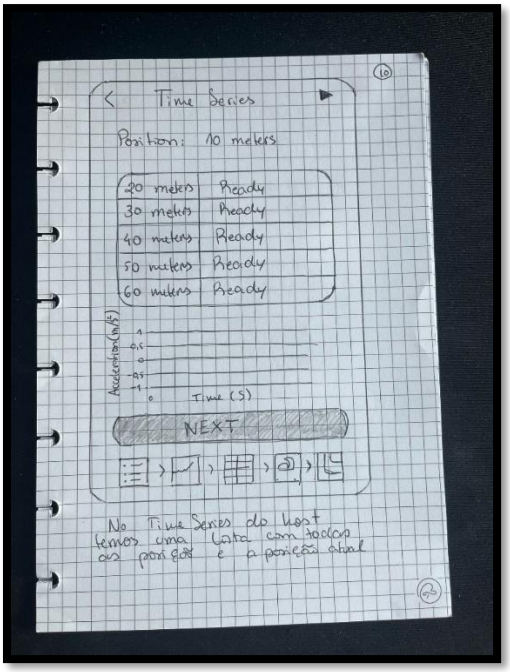
13. Botão *Connect* (*mockup*)



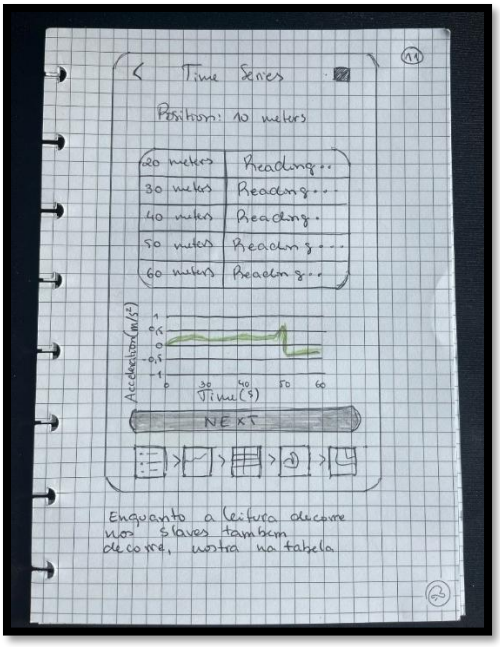
14. Opção para Ecrã *Structures* (*mockup*)



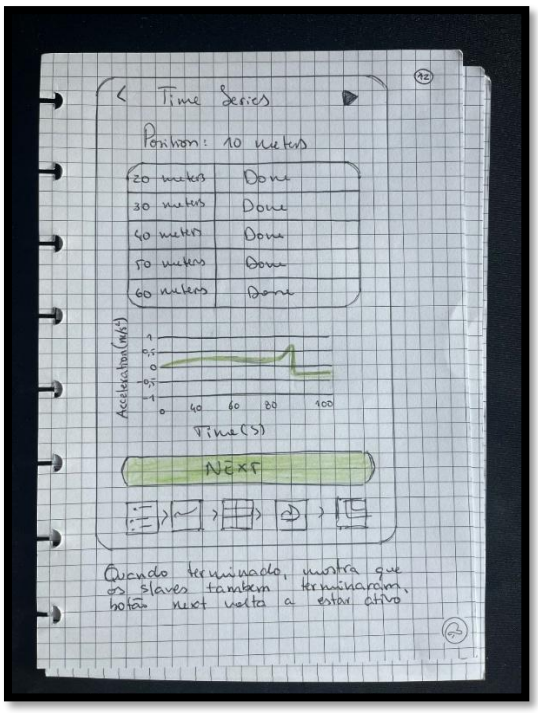
15. *Time Series* do *Host* (*mockup*)



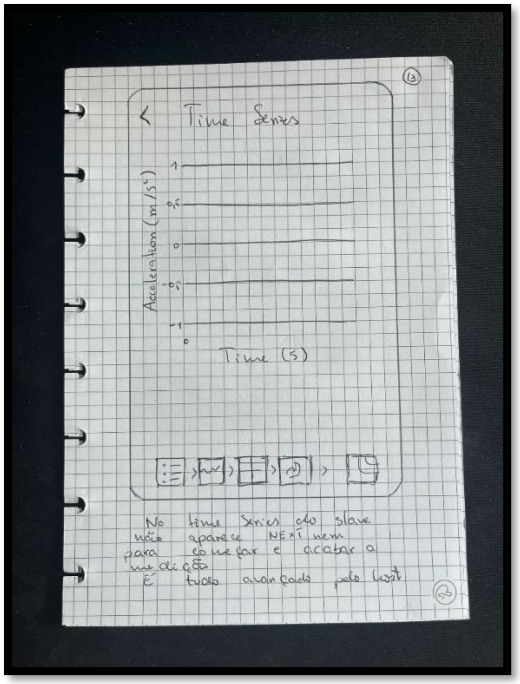
16. Leitura do *Host*, *Time Series* (*mockup*)



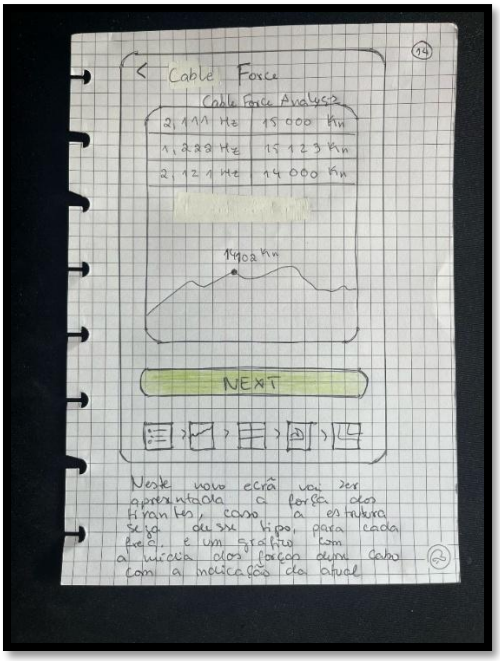
17. Leitura do *Host* terminada (*mockup*)



18. *Time Series* do *Slave* (*mockup*)



19. Ecrã *Cable Force* (*mockup*)



20. *Webservice*: CableForceList

***URL*:** /api/cable-force/

Métodos existentes: *POST*

Nome interno Django: "api-cable_force"

Descrição

O dispositivo envia o espectro de potência processado (com técnica de *Welch*), juntamente com os seguintes parâmetros:

- ***structure***: identificação da estrutura.
- ***welch***: matriz de densidade espectral (PSD).
- ***points***: número de pontos / frequências calculadas.
- ***training***: dados de treino para o modelo (opcional).
- ***isCable***: *flag* booleana que indica se é um cabo ou outro elemento.

O servidor realiza a análise das frequências dominantes, estima os modos vibracionais e calcula as forças correspondentes. O resultado da análise é devolvido ao dispositivo:

force_freq1, *force_freq2*, *force_freq3*: forças associadas às frequências dominantes

- *cable_force*: força total estimada no tirante.

- *count*: número de pontos analisados.
- *forces*: vetor com todas as forças extraídas.
- *pdf*: gráfico PDF gerado com a análise espectral.

A extração das forças é feita com base nos modos naturais de vibração identificados no espectro de potência.

Equações são utilizadas em *background* para converter frequência em força.

Em suma, este *endpoint* permite calcular e guardar uma medição de força num cabo com base em frequências fornecidas, bem como listar todas as medições existentes.

Além disso, se houver medições suficientes, realiza uma estimativa da distribuição estatística das forças usando KDE (Kernel Density Estimation).

Métodos *HTTP* Suportados

- *POST* – Cria uma medição com base nas frequências fornecidas e devolve os resultados.

Parâmetros Esperados (*POST*)

```
{
  "structure": 1,
  "reading": 10,
  "frequencies": [2.34, 4.56, 3.21]
}
```

Tabela 6 - Campos *CableForceList POST*

Campo	Tipo	Obrigatório	Descrição
structure	inteiro	Sim	ID da estrutura à qual a medição pertence.
reading	inteiro	Sim	ID da leitura (TimeSeries) associada à medição.
frequencies	array	Sim	Lista de 3 valores de frequência a usar no cálculo das forças.

Se *frequencies* não for fornecido ou estiver vazio, a API devolve erro 400

Resposta de Sucesso (exemplo com < 5 medições anteriores)

HTTP 201 *Created*


```
{
  "force_freq1": 123.45,
  "force_freq2": 130.67,
  "force_freq3": 110.98,
  "cable_force": 121.03,
  "count": 3
}
```


Resposta de Sucesso (exemplo com ≥ 5 medições anteriores e análise KDE)

HTTP 201 *Created*


```
{
  "force_freq1": 123.45,
  "force_freq2": 130.67,
  "force_freq3": 110.98,
  "cable_force": 121.03,
  "count": 8,
  "forces": [118.0, 120.0, 121.03, 122.5, 125.0],
  "pdf": [0.01, 0.06, 0.10, 0.08, 0.02]
}
```


Resposta de Erro

HTTP 400 *Bad Request*


```
{
  "error": "Frequencies not provided or empty."
}
```

Notas Técnicas

- A função tension_forces(...) é responsável por calcular as forças com base na massa e comprimento do cabo.
- Se houver ≥5 medições, a distribuição das forças é estimada com gaussian_kde (Scipy).
- O campo pdf representa a densidade de probabilidade das forças medidas.

Lógica Interna da *View*

1. **Validação Inicial**
    - Verifica se o *array frequencies* existe e tem conteúdo.
    - Vai buscar a estrutura e leitura correspondentes aos IDs fornecidos.
2. **Cálculo das Forças de Tensão**

    Utiliza a função tension_forces(...), que recebe:
    - massa do cabo (kg),
    - comprimento do cabo (m),
    - frequências medidas (Hz)

    E devolve:
    - *force_freq1*, *force_freq2*, *force_freq3*: forças calculadas para cada frequência individual;
    - cable_force: média ponderada das três.
3. **Criação da Medição**

    Cria um registo na base de dados com:
    - Estrutura e leitura associadas;
    - Utilizador autenticado (request.user);
    - Forças calculadas e frequências originais.
4. **Distribuição Estatística (opcional)**
    - Vai buscar todas as medições anteriores dessa estrutura.
    - Se houver 5 ou mais, aplica KDE (estimativa de densidade).
    - Retorna:
        - lista de forças (forces);
        - densidade estimada (pdf), para visualização em gráfico.

21. *Webservice*: CreateNetworkView

***URL*:** /network/create/

Métodos existentes: *POST*

Nome interno Django: "create-network"

Descrição

Este *webservice* permite a criação de novas redes (*Network*) associadas a estruturas físicas (*Structure*) que já existem na base de dados. Cada rede criada será automaticamente associada ao utilizador autenticado que a criou, definido como *master*.

Métodos *HTTP* Suportados

- *POST* – Cria uma *Network* associada a uma *Structure*.

Parâmetros Esperados (*POST*)

```
{
  "structureId": 4
}
```

Tabela 7 - Campos CreateNetworkView *POST*

Campo	Tipo	Obrigatório	Descrição
structureId	inteiro	Sim	ID da estrutura (*Structure*) à qual a nova rede será associada.

Se a estrutura com o structureId não existir, a criação da rede será recusada com erro 404.

Resposta de Sucesso

HTTP 201 *CREATED*

```
{
  "success": "Network created",
  "networkId": 35645
}
```

Resposta de Erro

HTTP 404 *NOT FOUND*

```
{
  "error": "Structure does not exists."
}
```

Lógica Interna da *View*

1. **Validação da Estrutura**
    - Verifica se existe uma *Structure* com o ID fornecido;
    - Se não existir, responde com erro e não prossegue.
2. **Criação da *Network***
    - Se a estrutura for válida, cria um registo Network;
    - Define o utilizador autenticado como master;
    - Liga a estrutura indicada à nova rede.
3. **Resposta de Sucesso**
    - Responde com uma mensagem de confirmação e o networkId gerado;
    - O ID pode ser codificado via id_to_code(...), se o sistema usar codificação obfuscada (por ex., base36, hashids, etc.).

22. Webservice: JoinNetworkView

***URL*:** /network/join/

Métodos existentes: POST

Nome interno Django: “join-network”

Descrição

Este *webservice* permite que um utilizador se junte a uma rede (*Network*) existente, assumindo uma posição livre numa localização específica da estrutura associada à rede. O sistema assegura que cada posição só pode estar ocupada por um utilizador de cada vez.

Métodos *HTTP* Suportados

- *POST* – Junta o utilizador a uma *Network*, criando uma *Position*.

Parâmetros Esperados (*POST*)

```
{
  "networkId": 35645,
  "location": 10
}
```

Tabela 8 - JoinNetworkView Campos *POST*

Campo	Tipo	Obrigatório	Descrição
networkId	inteiro	Sim	Identificador da *Network* (em formato codificado, ex: "35645").
location	inteiro	Sim	Localização desejada dentro da Structure (ex: "10").

Resposta de Sucesso

HTTP 200 *OK*

```
{
  "join position id": 286
}
```

Resposta de Erro

Tabela 9 - JoinNetworkView Erros *POST*

Situação	***HTTP Status***	**Exemplo de Resposta**
Rede não existe (networkId)	404 *NOT FOUND*	{“error": "Network does not exists." }
Localização inválida	404 *NOT FOUND*	{“error": "Position not found." }
Posição já ocupada	404 *NOT FOUND*	{“error": "Position occupied." }

Lógica Interna da *View*

1. **Conversão e Validação da *Network***
    - Decodifica o networkId usando code_to_id;
    - Verifica se a *Network* existe com esse ID;
    - Se não existir, responde com erro 404.
2. **Localização da Estrutura e Posição**
    - Obtém a estrutura (*Structure*) associada à rede;
    - Procura uma posição (*StructurePosition*) com a localização fornecida.
3. **Validação da Posição**
    - Se não existir posição com aquela localização, responde com erro 404;
    - Se a posição já estiver ocupada (Position.exists()), responde com erro 404.
4. **Criação de Posição Válida**
    - Cria um objeto *Position*, ligando o utilizador à *Network* e à *StructurePosition*;
    - Responde com sucesso e o ID da posição criada.

23. *Webservice*: NetworkInfoView

***URL*:** /network/info/

Métodos existentes: *GET*

Nome interno Django: “network-info”

Descrição

Este *webservice* permite consultar o estado atual de uma *Network*, verificando quais as posições previstas na estrutura que já estão ocupadas e quais ainda estão disponíveis. Serve como uma ferramenta de monitorização em tempo real da ocupação da rede.

Métodos *HTTP* Suportados

- *GET* – Obtém o estado da rede: posições disponíveis e ocupadas.

Parâmetros Esperados (*GET*)

/network/info/?networkId=35645

Este parâmetro é passado por query string

Tabela 10 - NetworkInfoView Campos *GET*

Nome	**Tipo**	**Obrigatório**	**Descrição**
networkId	inteiro	Sim	Identificador codificado da Network (ex: "35645").

Resposta de Sucesso

HTTP 200 *OK*

```
{
  "locations": [
    {
      "structure_position": {
        "position_location": 10,
        "status": "connected"
      }
    },
    {
      "structure_position": {
        "position_location": 20,
        "status": "not connected"
      }
    },
    ...
  ],
  "structure_positions_count": 8,
  "positions_count": 3
}
```


Resposta de Erro

Tabela 11 - NetworkInfoView Erros *GET*

Situação	***HTTP Status***	**Exemplo de Resposta**
Rede não existe (networkId inválido)	404 *NOT FOUND*	{“error": "Network does not exists." }

Lógica Interna da *View*

1. **Validação da *Network***
    - Converte networkId com code_to_id();
    - Verifica se a *Network* existe. Se não existir, responde com erro 404.

2. **Obtenção das Estruturas e Posições**
    - Recolhe todos os *StructurePosition* definidos para a *Structure* associada à Network;
    - Recolhe todas as *Position* associadas à *Network*.
3. **Construção do Estado da Rede**
    - Para cada *StructurePosition*, verifica se existe uma *Position* ligada a ela;
    - Marca cada posição como:
        - "connected" → já ocupada
        - "not connected" → ainda livre
4. **Resposta Final**
    - Lista de localizações com estado;
    - Número total de posições estipuladas na estrutura;
    - Número de posições já ocupadas.

24. *Webservice*: NetworkStatusView

***URL*:** /network/set-status/

Métodos existentes: *GET*, *POST*

Nome interno Django: "network-status"

Descrição

Este *webservice* permite consultar e atualizar o estado de uma rede (*Network*). É útil para acompanhar o progresso de uma rede ao longo do tempo — desde o início da recolha de dados até à sua conclusão.

- O método *GET* serve para consultar o estado atual da rede.
- O método *POST* permite atualizar esse estado e, opcionalmente, registar datas importantes (startDate ou endDate), dependendo da fase da rede.

Métodos *HTTP* Suportados

- *GET* – Consulta o estado atual da rede.
- *POST* – Atualiza o estado da rede e regista data de início/fim.

Parâmetros Esperados (*GET*)

```
GET /network/set-status/?networkId=35645
```

Resposta de Sucesso

HTTP 200 *OK*

```
{
  "status": "waiting"
}
```

Resposta de Erro

HTTP 404 *NOT FOUND*

```
{
  "error": "Network does not exists."
}
```

Parâmetros Esperados (*POST*)

```
{
  "networkId": 35645,
  "status": "reading",
  "startDate": "2025-06-22T14:00:00"
}
```

Tabela 12 - NetworkStatusView Campos *POST*

Nome	Tipo	Obrigatório	Descrição
networkId	inteiro	Sim	Identificador da rede codificado.

status	string	Sim	Novo estado da rede: "waiting", "reading", "completed" (por ex.).
startDate	string	Condicional	Obrigatório apenas se o status for "reading". Formato: "YYYY-MM-DDTHH:MM:SS".
endDate	string	Condicional	Obrigatório apenas se o status for "completed". Mesmo formato.

Resposta de Sucesso

```
{
  "status": "reading"
}
```

Resposta de Erro

Tabela 13 - NetworkStatusView Erros *POST*

Situação	***HTTP Status***	**Exemplo de Resposta**
Rede não existe (networkId inválido)	404 *NOT FOUND*	{“error": "Network does not exists." }

Lógica Interna da *View*

Método *GET*:

1. O serviço extrai o parâmetro networkId da *query string*.
2. Converte o valor codificado para o ID real da base de dados com code_to_id.
3. Verifica se a rede com esse ID existe.
    - Se não existir, devolve erro 404 com {"error": "Network does not exists."}.
4. Se a rede existir, obtém o seu status atual.
5. Devolve o estado da rede com um HTTP 200 OK.

Método *POST*:

1. Lê o networkId e o novo *status* do corpo do pedido.

2. Converte o ID codificado para o ID real da rede.
3. Verifica se a rede existe.
    - Se não existir, devolve 404 Not Found.
4. Atualiza o campo status do objeto *Network*.
5. Se o novo estado for:
    - "reading" → guarda a data de início (startDate) no campo start_date.
    - "completed" → guarda a data de fim (endDate) no campo end_date.
6. Grava as alterações com network.save().
7. Devolve o novo estado atualizado como confirmação.

25. *Webservice*: NetworkReadingsView

***URL*:** /network/readings/

Métodos existentes: *GET*, *POST*

Nome no Django: "network-readings"

Descrição

Este serviço permite a gestão e análise das leituras (*TimeSeries*) associadas a uma rede (*Network*):

- Com *GET*, mostra o estado de ocupação e leitura de cada posição da rede.
- Com *POST*, associa uma leitura concreta a uma posição, mas apenas se a rede estiver concluída (status == "completed").

Métodos *HTTP* Suportados

- *GET* – Verifica o estado das leituras por posição numa *Network*.
- *POST* – Associa uma *TimeSeries* a uma *Position*, se permitido.

Parâmetros Esperados (*GET*)

```
GET /network/readings/?networkId=35645
```

Resposta de Sucesso

HTTP 200 *OK*

```
{
  "locations": [
    {
      "structure_position": {
        "position_location": 10,
        "reading": "completed"
      }
    },
    {
      "structure_position": {
        "position_location": 20,
        "reading": "pending"
      }
    }
  ],
  "all_done": "pending"
}
```

Se todas as leituras estiverem completas, os seguintes dados são adicionados:

```
{
  "mean": [12.5, 18.9, 24.3],
  "frequencies": [1.4, 2.6, 3.1],
  "x": [...], "y": [...], "z": [...]
}
```

Resposta de Erro

Tabela 14 - NetworkStatusView Erros *GET*

Situação	***HTTP Status***	**Exemplo de Resposta**
Rede não existe (networkId inválido)	404 *NOT FOUND*	{“error": "Network does not exists." }
Readings ausentes	500 *INTERNAL SERVER ERROR*	{“error": "No valid readings found." }

Parâmetros Esperados (*POST*)

```
{
  "networkId": 35645,
  "location": 10,
  "reading": 128
}
```

Resposta de Sucesso

HTTP 201 *CREATED*

```
{
  "reading": "Reading posted."
}
```

Erros possíveis:

- Rede inexistente
- Estado da rede diferente de "completed"
- Localização inválida
- Leitura inválida

Lógica Interna da *View*

Método *GET*:

1. **Validação do networkId:**

- Converte o ID codificado para ID real com code_to_id.
- Verifica se a *Network* existe.
- Se não existir, retorna 404.

2. **Recolha de posições:**
   - Obtém todas as *Position* associadas à *Network*.
   - Para cada uma:
     - Marca como "pending" se ainda não tiver leitura.
     - Marca como "completed" se já estiver lida.
   - Flag all_done marca se todas estão completas.
3. **Se all_done for *True*:**
   - Lê os ficheiros brutos (raw_file) das leituras associadas.
   - Alinha e analisa os dados com align_multiple_streams() e calculate_mean_welch_frequencies().
   - Calcula média das frequências naturais registadas (*NaturalFrequencies*) para a estrutura.
   - Adiciona os dados de análise (mean, frequencies, x, y, z) à resposta.
4. **Retorno final:**
   - Envia um JSON com todas as posições e (se aplicável) os dados de leitura agregados.

Método *POST*:

1. **Validação da rede:**
   - Converte o networkId.
   - Verifica se a *Network* existe.
   - Se não existir, retorna 404.
2. **Estado da rede:**
   - Se status != "completed", retorna erro 400.
3. **Validação da posição e leitura:**
   - Verifica se a *StructurePosition* para o *location* indicado existe.
   - Verifica se a *Position* correspondente existe.
   - Verifica se a leitura (*TimeSeries*) com o ID existe.

- Qualquer falha retorna 400.

4. **Associação:**
   - Atribui a leitura à *Position*.
   - Grava a alteração com position.save().
5. **Resposta:**
   - Retorna mensagem de sucesso {"reading": "Reading posted."} com HTTP 201.

26. *Webservice*: DisconnectNetworkView

***URL*:** /network/disconnect/

Métodos existentes: *POST*

Nome no Django: "disconnect-network"

Descrição

Este *endpoint* permite remover uma posição (*Position*) de uma rede (*Network*). Ao desconectar uma posição, esta deixa de estar associada à rede e pode ser reutilizada.

Métodos *HTTP* Suportados

- *POST* – Remove (desconecta) uma *Position* da *Network*

Parâmetros Esperados (*POST*)

```
{
  "networkId": 35645,
  "location": 30
}
```

Tabela 15 - DisconnectNetworkView Campos *POST*

Campo	Tipo	Obrigatório	Descrição

networkId	inteiro	Sim	Identificador codificado da rede.
location	inteiro	Sim	Localização da estrutura (ex: "30") que se pretende remover.

Resposta de Sucesso

HTTP 200 *OK*

```
{
  "deleted position location": 30
}
```

Erros possíveis:

- Rede não existe
- Localização não encontrada na estrutura
- Posição já removida ou nunca ocupada

Lógica Interna da *View*

1. **Validação do networkId**
    - Converte o networkId codificado para o ID interno da rede com code_to_id().
    - Verifica se a *Network* existe.
    - Se não existir, retorna 404.
2. **Validação da localização (*location*)**
    - Procura a *StructurePosition* correspondente à estrutura da *Network* e ao campo *location*.
    - Se a localização não existir, retorna 404.
3. **Remoção da posição**
    - Verifica se existe uma *Position* ligada à rede e à localização indicada.
    - Se existir:
        - Guarda o nome da localização.
        - Remove a *Position* com .delete().

- Retorna 200 *OK* com a localização removida.

- Se não existir, retorna 404 com a mensagem "Position not found".

27. *Webservice*: DeleteNetworkView

***URL*:** /network/delete/

Métodos existentes: *DELETE*

Nome no Django: "delete-network"

Descrição

Este *endpoint* permite eliminar permanentemente uma rede (*Network*) do sistema, removendo todos os dados associados à instância da rede. Esta ação é irreversível.

Métodos *HTTP* Suportados

- *DELETE* – Elimina uma instância de Network

Parâmetros Esperados (*DELETE*)

```
{
  "networkId": 35645
}
```

Tabela 16 - DeleteNetworkView Campos *DELETE*

Campo	Tipo	Obrigatório	Descrição
networkId	inteiro	Sim	Identificador codificado da rede.

Resposta de Sucesso

HTTP 200 *OK*

```
{
  "deleted network": 35645
}
```

Resposta de Erro

HTTP 404 *NOT FOUND*

```
{
  "error": "Network does not exist."
}
```

Lógica Interna da *View*

1. **Leitura do networkId:**
    - Obtém o valor enviado no campo networkId do corpo da requisição.
    - Converte-o para o ID interno com code_to_id().
2. **Validação da existência da rede:**
    - Tenta obter o objeto *Network* através do ID.
    - Se não existir, retorna 404 Not Found.
3. **Eliminação da rede:**
    - Chama .delete() no objeto *Network*, eliminando-o permanentemente da base de dados.
4. **Resposta:**
    - Retorna 200 *OK* com o ID da rede eliminada.

Glossário

LEI	Licenciatura em Engenharia Informática
TFC	Trabalho Final de Curso