UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Gustavo Trentin

Desenvolvimento de um módulo no aplicativo DynaPredict para realização de Espectral Simultânea

Florianópolis
2023

Gustavo Trentin

Desenvolvimento de um módulo no aplicativo DynaPredict para realização de Espectral Simultânea

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Rodrigo Castelan Carlson, Dr.
Supervisor: João Pedro dos Reis, Eng.

Florianópolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Trentin, Gustavo
Desenvolvimento de um módulo no aplicativo DynaPredict para realização de Espectral Simultânea / Gustavo Trentin ; orientador, Rodrigo Castelan Carlson, coorientador, João Pedro dos Reis, 2023.
84 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Manutenção Preditiva. 3. Monitoramento Simultâneo. 4. Desenvolvimento de Aplicativo para iOS. I. Carlson, Rodrigo Castelan. II. Reis, João Pedro dos. III. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. IV. Título.

Gustavo Trentin

Desenvolvimento de um módulo no aplicativo DynaPredict para realização de Espectral Simultânea

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 29 de Junho de 2023.

Prof. Hector Bessa Silveira , Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Rodrigo Castelan Carlson, Dr.
Orientador
UFSC/CTC/DAS

João Pedro dos Reis, Eng.
Supervisor
Empresa Dynamox

Profa. Sheila Regina Oro, Dra.
Avaliadora
Instituição UTFPR

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

Dedicatória: Este trabalho é dedicado a mim, que me esforcei para concluí-lo.

AGRADECIMENTOS

Agradeço a todos que contribuíram para a realização deste projeto, pois sem o apoio e a colaboração de cada um de vocês, essa conquista não seria possível.

Primeiramente, agradeço à minha família, meus pais Ronaldo e Rosane, meu irmão Otávio, que sempre estiveram ao meu lado, apoiando-me em todas as etapas dessa jornada acadêmica.

Agradeço também a minha namorada Bruna Bagnara, pelo auxílio durante a produção textual, apoio emocional e compreensão durante esse longo processo.

Ao meu orientador, Rodrigo Castelan Carlson, sou imensamente grato por sua orientação, paciência e dedicação ao longo deste processo.

Agradeço a instituição Dynamox por ceder o espaço para desenvolvimento deste trabalho, bem como, aos meus colegas de equipe pelo auxílio nos desafios técnicos enfrentados. Em especial ao meu supervisor João Pedro pela compreensão durante o desenvolvimento e correção do documento.

Agradeço também as instituições que contribuíram para minha formação durante essa jornada acadêmica, como a UFSC e todos os docentes com quem tive contato, a Autojun e a equipe a qual fiz parte, e ao LVA e a equipe que participei.

Por fim, agradeço aos amigos e colegas de turma, por toda a parceria durante a etapa acadêmica, que com certeza contribuíram para esse processo de formação ser o mais leve e agradável.

A todos os meus mais sinceros agradecimentos.

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 16 de Julho de 2023.

Na condição de representante da Dynamox na qual o presente trabalho foi realizado, declaro não haver ressalvas quanto ao aspecto de sigilo ou propriedade intelectual sobre as informações contidas neste documento, que impeçam a sua publicação por parte da Universidade Federal de Santa Catarina (UFSC) para acesso pelo público em geral, incluindo a sua disponibilização *online* no Repositório Institucional da Biblioteca Universitária da UFSC. Além disso, declaro ciência de que o autor, na condição de estudante da UFSC, é obrigado a depositar este documento, por se tratar de um Trabalho de Conclusão de Curso, no referido Repositório Institucional, em atendimento à Resolução Normativa n° 126/2019/CUn.

Por estar de acordo com esses termos, subscrevo-me abaixo.

Documento assinado digitalmente
JOAO PEDRO DOS REIS
Data: 17/07/2023 12:24:54-0300
CPF: ***.175.329-**
Verifique as assinaturas em https://v.ufsc.br

João Pedro dos Reis
Dynamox

RESUMO

Com os avanços da indústria 4.0, o uso de sensores em máquinas e ferramentas (ativos) industriais permitiu que uma nova política de manutenção pudesse ser implementada. A manutenção preditiva (PdM) utiliza os dados de monitoramento de ativos industriais para programar manutenções, sendo mais eficiente e, a longo prazo, mais econômica que as demais políticas de manutenção. Em algumas aplicações, é necessário que os dados de monitoramento para PdM sejam coletados em vários pontos do ativo industrial simultaneamente, para que sejam posteriormente correlacionados. Esses monitoramentos podem ser realizados com sensores acoplados a registradores de dados (*data loggers*), que são fixados em pontos de interesse dos ativos. A configuração e coleta dos *data loggers* pode ser realizada por dispositivos móveis com aplicativos específicos. Nesse contexto, optou-se pela construção de um módulo de um aplicativo para dispositivo móvel com sistema operacional iOS, tendo foco na configuração e coleta de monitoramentos realizados simultaneamente em vários pontos de um ativo industrial. Para isso, o módulo deve conter ferramentas que sejam capazes de conectar-se com os *data loggers*, agendando o monitoramento para que seja realizado de forma simultânea, coletando os dados do monitoramento e apresentando-os graficamente. Por fim, os dados devem ser enviados para um servidor, que será acessado por uma plataforma web para análise profunda dos resultados. Toda a etapa de desenvolvimento dessas funcionalidades foi pautada na metodologia Scrum. Além disso, foram utilizados conceitos de arquitetura de *software* e princípios SOLID. O módulo desenvolvido pode ser dividido em etapas, sendo elas: 1) o tour para os usuários; 2) o agendamento do monitoramento; 3) as notificações sobre o estado do monitoramento; 4) a coleta e visualização gráfica dos dados; e 5) a gestão do envio dos dados para o servidor. As atividades foram desenvolvidas gradualmente, e a cada funcionalidade desenvolvida uma nova versão do código era gerada. Todas as funcionalidades foram validadas a partir de testes automatizados. Além disso, os códigos foram revisados por outros desenvolvedores e as funcionalidades testadas quanto à usabilidade e cumprimento dos requisitos por um analista de qualidade, gerando retrabalho em alguns casos. Por fim, foi obtido um módulo capaz de cumprir com os requisitos e objetivos propostos, proporcionando aos usuários novas ferramentas capazes de auxiliar na previsão de falhas que necessitem de monitoramento simultâneo, contribuindo para ampliar a gama de falhas previstas e assim melhorar a eficiência do processo de manutenção.

Palavras-chave: Manutenção Preditiva. Monitoramento Simultâneo. Desenvolvimento de Aplicativo para iOS.

ABSTRACT

With the advances of Industry 4.0, the use of sensors in machines and tools (assets) allowed a new maintenance policy to be implemented. Predictive maintenance (PdM) uses industrial asset monitoring data to schedule maintenance, being more efficient and, in the long run, more economical than other maintenance policies. In some applications, it is necessary that the monitoring data for PdM be collected at various points of the industrial asset simultaneously, so that they can be correlated later. This monitoring can be performed with sensors coupled to data loggers, which are fixed at points of interest on the assets. The configuration and collection of the data loggers can be performed by mobile devices with specific applications. In this context, it was decided to build an application module for a mobile device with iOS operating system, focusing on the configuration and collection of monitoring performed simultaneously in several points of an industrial asset. For this, the module must contain tools that are able to connect with the data loggers, scheduling the monitoring to be performed simultaneously, collecting the monitoring data and presenting them graphically. Finally, the data should be sent to a server, which will be accessed by a web platform for in-depth analysis of the results. The entire development stage of these functionalities was based on the Scrum methodology. In addition, concepts of software architecture and SOLID principles were used. The developed module can be divided in stages, being them: 1) the tour for the users; 2) the monitoring scheduling; 3) the notifications about the monitoring status; 4) the data collection and graphical visualization; and 5) the management of the data sending to the server. The activities were developed gradually, and for each functionality developed a new version of the code was generated. All functionalities were validated through automated tests. In addition, the codes were reviewed by other developers and the functionalities tested for usability and compliance with the requirements by a quality analyst, generating rework in some cases. Finally, a module capable of fulfilling the proposed requirements and objectives was obtained, providing users with new tools capable of helping in the prediction of failures that need simultaneous monitoring, contributing to expand the range of predicted failures and thus improving the efficiency of the maintenance process.

Keywords: Predictive Maintenance. Simultaneous Monitoring. Application Development for iOS

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A Indústria 4.0 é uma revolução tecnológica que está transformando a forma como as empresas produzem e gerenciam seus negócios. Ela se baseia na integração de tecnologias digitais e físicas (MACÊDO, 2020). Nesse contexto, o monitoramento dos ativos (máquinas e equipamentos) industriais em tempo real é um dos princípios para a implementação da indústria 4.0, e a manutenção com base em dados provenientes desses monitoramentos é a forma mais adequada para acompanhar essa revolução tecnológica (SANTOS NETO; LEITE; NASCIMENTO, 2018).

A manutenção preditiva (PdM), é uma política de manutenção que utiliza tecnologias avançadas para monitorar ativos industriais, e com base nos dados do monitoramento, antecipar e localizar problemas nesses ativos (SUSTO; BEGHI; DE LUCA, 2012). Um sistema de manutenção preditiva pode ser desenvolvido com o uso dos mais diversos tipos de sensores. Esses são escolhidos com base na aplicação para monitorar vibração, temperatura, concentração de gás, umidade... Os sensores podem ser acoplados a registradores de dados (*data loggers*), que armazenam os dados coletados durante o monitoramento. Para a configuração do monitoramento e coleta dos dados podem ser utilizados dispositivos externos capazes de se conectar com os *data loggers*, como *gateways* ou então dispositivos móveis com aplicativos específicos instalados. Após a coleta, os dados podem ser analisados por especialistas com o uso de gráficos e indicadores específicos.

No modelo supracitado, o fluxo da manutenção preditiva pode ser expresso de maneira simplificada por cinco etapas: 1) configuração do monitoramento a ser realizado; 2) realização do monitoramento do ativo; 3) coleta dos dados obtidos durante o monitoramento; 4) análise dos dados; e 5) manutenção do equipamento, caso necessário.

Nesse contexto, este documento descreve a problemática, desenvolvimento e validação do módulo de um aplicativo para dispositivo móvel que configura e coleta dados de monitoramento em *data loggers* de um sistema de manutenção preditiva.

Os *data loggers* utilizados neste projeto são acoplados com sensores capazes de monitorar a vibração dos equipamentos. Dessa forma, é possível analisar os dados de velocidade e aceleração obtidos, tanto no domínio do tempo, como no domínio da frequência, podendo ser realizada uma análise do espectro do sinal adquirido no monitoramento. Ainda, o módulo tem como foco a realização de monitoramentos que ocorram simultaneamente em diversos pontos de um ativo industrial. Esses dois conceitos apresentados são capazes de definir o nome do módulo desenvolvido como Espectral Simultânea.

1.1 JUSTIFICATIVA

A PdM é uma inovação que tem se mostrado relevante no setor industrial. Diferentemente da manutenção *run-to-failure* (R2F) [1] e da manutenção preventiva (PvM) [2], a PdM utiliza ferramentas para prever as falhas, sendo essa predição o fator determinante para a realização da manutenção. O uso da PdM evita que um grande número de produtos defeituosos sejam produzidos em decorrência de falha, como pode ocorrer na R2F. Ainda, evita que manutenções desnecessárias sejam realizadas, como ocorre na PvM. Dessa forma, a PdM torna o processo de manutenção mais eficiente e a longo prazo mais econômico (SUSTO; BEGHI; DE LUCA, 2012).

Em um sistema de PdM diversas falhas podem ser identificadas ao realizar o monitoramento em apenas um ponto do ativo, ou em vários pontos em período de tempo distinto. Entretanto, muitas vezes é necessário o uso de correlação entre os resultados das medições que compõem o monitoramento. Para isso, é necessário que essas medições sejam realizadas em um mesmo instante de tempo e em diferentes pontos de um ativo industrial.

Um caso de exemplo de falha identificada a partir da correlação dos dados de monitoramento simultâneo pode ser visto em peneiras vibratórias utilizadas na área de mineração (Figura 1). Nessas peneiras, a eficiência e capacidade do processo é diretamente proporcional a boa distribuição do material de peneiramento (LIU *et al.*, 2012). Isso é atingido através do perfeito balanceamento entre o movimento relativo dos quatro conjuntos de molas da peneira. Tendo isso em vista, as peneiras vibratórias podem ser avaliadas quanto a sua eficiência através da comparação entre os dados de vibração coletados em cada conjunto de mola, sendo necessário o monitoramento simultâneo nestes quatro conjuntos.

1.2 OBJETIVO GERAL

Desenvolver uma solução para aplicativo de dispositivo móvel em sistema operacional iOS que possibilita a realização de monitoramento de dados de maneira simultânea em diversos pontos de um ativo industrial.

1.3 OBJETIVOS ESPECÍFICOS

- Planejar e desenvolver fluxos lógicos para a implementação do módulo;
- Desenvolver as interfaces gráficas;
- Implementar fluxos para o agendamento do monitoramento e coleta dos dados;

[1] Na política de manutenção R2F o processo de manutenção é realizado após a ocorrência da falha.
[2] Na PvM a manutenção é realizada periodicamente, mesmo sem indícios de falhas no equipamento.

Figura 1 – Peneira vibratória utilizada na área de mineração.

Fonte: (DIRECT INDUSTRY, 2023).

- Apresentar graficamente os dados do monitoramento;
- Salvar dados do monitoramento;
- Enviar dados para a plataforma web da empresa;
- Implementar testes automatizados.

1.4 ESTRUTURA DO DOCUMENTO

No Capítulo 2 é feita uma descrição da empresa, tendo enfoque nos produtos e ferramentas oferecidos por ela e nos processos da equipe responsável pelo desenvolvimento do aplicativo.

No Capítulo 3 é apresentada a fundamentação teórica, sendo a base conceitual e metodológica que sustenta o projeto. Nela são apresentados os conceitos e fundamentos necessários para o planejamento e desenvolvimento do projeto.

No Capítulo 4 é apresentada toda a parte de pré-desenvolvimento do sistema, como: a visão geral do projeto e a descrição das tarefas planejadas.

No Capítulo 5 são descritas as etapas e processos utilizados para o desenvolvimento da solução. Para isso, são apresentados alguns diagramas referentes a um projeto de *software* e são retomados alguns conceitos apresentados em capítulos anteriores.

No Capítulo 6 o resultado obtido é apresentado, sendo abordados aspectos referentes aos fluxos e funcionalidades desenvolvidos, bem como, os resultados das validações realizadas sobre do módulo.

Por fim, no Capítulo 7 é feita uma conclusão, retomando os assuntos anteriormente abordados e trazendo um fechamento. Ainda, são feitas análises críticas sobre o projeto, abordando suas limitações e futuras melhorias.

2 CONTEXTUALIZAÇÃO

A Dynamox é uma empresa que tem o propósito de auxiliar na manutenção preditiva de ativos industriais. A solução oferecida pela empresa é composta por *data loggers* com sensores acoplados, aplicativos para dispositivo móvel, *gateways* e uma plataforma web.

O projeto em questão foi desenvolvido inteiramente para o aplicativo em dispositivo móvel para sistema operacional iOS, fazendo o uso dos *data loggers* da empresa para a realização de alguns testes e enviando os dados coletados para a plataforma web da empresa.

Portanto, no presente capítulo serão apresentadas as ferramentas que compõem a solução oferecida pela empresa. Ainda, serão abordadas questões referentes ao funcionamento da empresa em geral, bem como, o funcionamento da equipe responsável pelo desenvolvimento dos aplicativos da empresa.

2.1 SOLUÇÃO

As interações entre as ferramentas que compõem a solução oferecida pela empresa podem ser compreendidas através do fluxo de dados dos resultados do monitoramento. A base operacional da solução são os *data loggers*, responsáveis pelo monitoramento e armazenamento de dados de temperatura e vibração. Os *gateways* da empresa conectam-se aos *data loggers*, coletando os dados obtidos durante o monitoramento e enviando-os para um servidor que é acessado pela plataforma web. O aplicativo para dispositivo móvel também é capaz de realizar a coleta, entretanto, deve ser manuseado por um inspetor para isso. Em contrapartida, o aplicativo proporciona algumas funcionalidades a mais que os *gateways*. Por fim, a plataforma web é utilizada para análise e prognóstico dos dados coletados.

Para entender melhor o funcionamento de cada ferramenta disponibilizada pela empresa, elas serão detalhadas separadamente a seguir:

2.1.1 Data logger

O *data logger* da Dynamox recebe o nome de DynaLogger (Figura 2) e tem como objetivos principais realizar a coleta e armazenar os dados de vibrações e temperatura de ativos industriais. Para que o monitoramento possa ser realizado, um ou mais DynaLoggers devem ser fixados em pontos específicos do ativo, sendo que a fixação é realizada com o uso de cola, parafuso ou ímã.

Os DynaLoggers atuam com dois tipos diferentes de monitoramento:

- **Monitoramento contínuo/telemetria:** consiste em medições contínuas, compassadas de acordo com as configurações do usuário. Cada medição é caracterizada

Figura 2 – *Data logger* da empresa Dynamox

Fonte: (DYNAMOX, 2023b).

pela realização de um monitoramento em curto período de tempo com frequência máxima específica. O resultado de cada medição são valores únicos, calculados através de operações realizadas sobre o sinal obtido (como RMS do sinal, por exemplo). Dessa maneira as métricas obtidas em cada medição não são embasadas em apenas um ponto isolado no tempo.

Com esse tipo de coleta, diferentes métricas podem ser construídas, como: aceleração, velocidade e deslocamento em RMS, pico, pico-pico e fator de crista, skewness, curtose e temperatura de contato (DYNAMOX, 2023a);

- **Monitoramento para análise espectral/análise de forma de onda:** o monitoramento para análise espectral também consiste na coleta de dados por um período de tempo e frequência pré determinados. Entretanto, esse tipo de monitoramento não é realizado ciclicamente ao longo do tempo e retorna todos os pontos medidos. Vale ressaltar que, o monitoramento para análise espectral suporta maior variação de configurações se comparado ao contínuo (maior gama de frequências e durações).

  Com esse tipo de monitoramento é possível obter algumas métricas, como: espectro, forma de onda (linear, circular e orbital), filtros de frequência, cepstro, envelope espectral (demodulação), autocorrelação e multimétricas (DYNAMOX, 2023a).

Por fim, a empresa disponibiliza diferentes modelos de DynaLoggers, sendo

que a diferença entre os modelos se dá pela velocidade de envio de dados, memória, frequência máxima de coleta e suporte a diferentes configurações de coleta. Tendo isso em vista, existem modelos de DynaLogger mais indicados de acordo com o ativo que se deseja monitorar.

2.1.2 Gateway

O *gateway* da empresa é denominado DynaGateway (Figura 3) e atua paralelamente ao aplicativo para dispositivo móvel, sendo capaz de coletar os dados dos DynaLoggers e enviá-los para um servidor que a plataforma web da empresa utiliza. Ressalta-se aqui a importância da coleta de dados do DynaLoggers devido à limitação de armazenamentos que eles oferecem (assim como outros *hardwares* embarcados).

Figura 3 – *Gateway* da empresa Dynamox

Fonte: (DYNAMOX, 2023b).

Para a realização da coleta dos dados armazenados nos DynaLoggers, o DynaGateway deve estar constantemente conectado ao Bluetooth, que é a tecnologia utilizada na comunicação entre esses dispositivos. O DynaGateway também deve estar conectado à internet, para o envio de dados ao servidor.

A principal vantagem do DynaGateway em relação ao aplicativo para dispositivo móvel é a realização de coletas de forma recorrente e sem a necessidade de um inspetor. Ao utilizá-lo juntamente com os DynaLoggers, o conjunto é capaz de manter um ativo monitorado constantemente.

2.1.3 Aplicativo para dispositivo móvel

O aplicativo para dispositivos móveis da Dynamox está disponível tanto para sistema operacional Android, quanto iOS, e é denominado DynaPredict. Este aplicativo, na sua versão para iOS, é o foco principal do projeto.

Para melhor compreensão, o aplicativo pode ser dividido em tipos de funcionalidades oferecidas:

- **Funcionalidades preditivas:** permitem ao inspetor gerenciar a árvore de ativos da empresa, possibilitando a associação de DynaLoggers a pontos de monitoramento em máquinas, subconjuntos da máquina e componentes da máquina. Além disso, no momento do cadastro, algumas configurações iniciais condizentes ao monitoramento são realizadas, como o compasso do monitoramento contínuo, a faixa dinâmica, as configurações padrão do monitoramento para análise espectral e alertas baseados em limiares ajustáveis de temperatura ou vibração. A parte preditiva do aplicativo também possibilita a coleta e envio de dados dos DynaLoggers;
- **Funcionalidades sensitivas:** auxiliam os inspetores na realização de rotas para análise de ativos industriais. Uma rota é composta por um ou mais ativos, e é realizada através do preenchimento de *checklists* e realização de monitoramentos para análise espectral predefinidos (definidos durante o cadastro dos pontos de monitoramento). Vale destacar que as *checklists* são preenchidas com base nos sentidos do inspetor (visão, audição, olfato e tato), que também são importantes para avaliar a “saúde” do equipamento (DYNAMOX, 2023b);
- **Demais funcionalidades:** visam auxiliar o inspetor tanto no gerenciamento, quanto no monitoramento dos ativos, contemplando funcionalidades como o “Laudo de Análise Sensitiva”, que possibilita ao inspetor reportar problemas em ativos mesmo que estejam fora de sua rota, os “DynaLoggers Portáteis”, que possibilita ao inspetor realizar o monitoramento com DynaLogger mesmo que esse não tenha sido associado previamente a um ponto de monitoramento, entre outras funcionalidades. Vale destacar que o produto desenvolvido no presente projeto pertencerá a essa área do aplicativo.

Portanto, o DynaPredict oferece aos usuários algumas funcionalidades que auxiliam no monitoramento e gerenciamento dos ativos industriais.

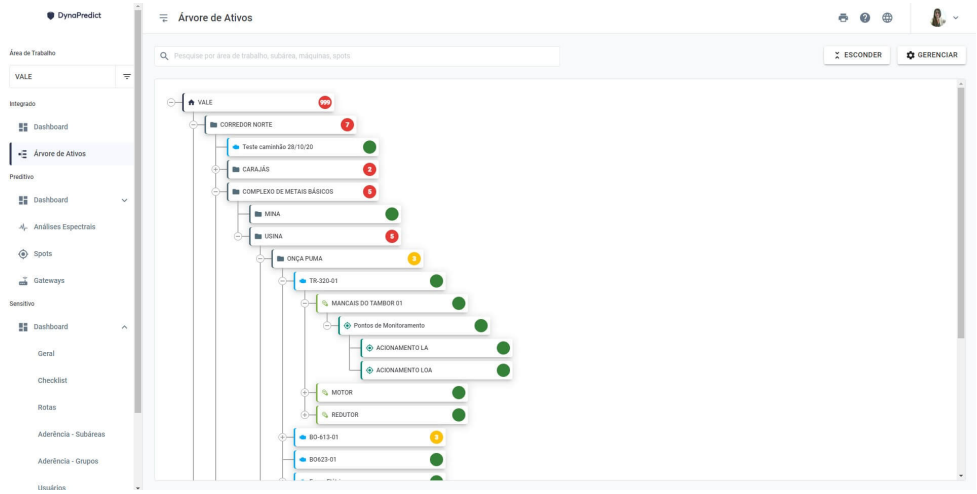
2.1.4 Plataforma web

A plataforma web da empresa recebe o nome de DynaPredict Web e engloba diversas ferramentas que possibilitam a análise aprofundada dos dados coletados,

bem como, o gerenciamento dos ativos em formato de árvore (Figura 4) e *dashboards* (Figura 5) que oferecem uma apresentação mais clara da informação.

Para a análise aprofundada dos dados coletados o DynaPredict Web traz ferramentas como: espectros no domínio da frequência e forma de onda, envelope com diversas faixas de filtro, filtros passa-alta, passa-baixa e passa-banda, cursores e harmônicos diversos, e escala logarítmica para destacar frequências em baixa rotação, entre outras. Além disso, muitas dessas ferramentas não estão disponíveis para o DynaPredict de dispositivo móvel, visto que, normalmente o local onde ocorre a coleta dos dados não é apropriado para a realização de analises (chão de fábrica) e o inspetor que realiza a coleta nem sempre é um analista acústico.

Figura 4 – Árvore de ativos apresentada no DynaPredict Web



Fonte: (DYNAMOX, 2023b).

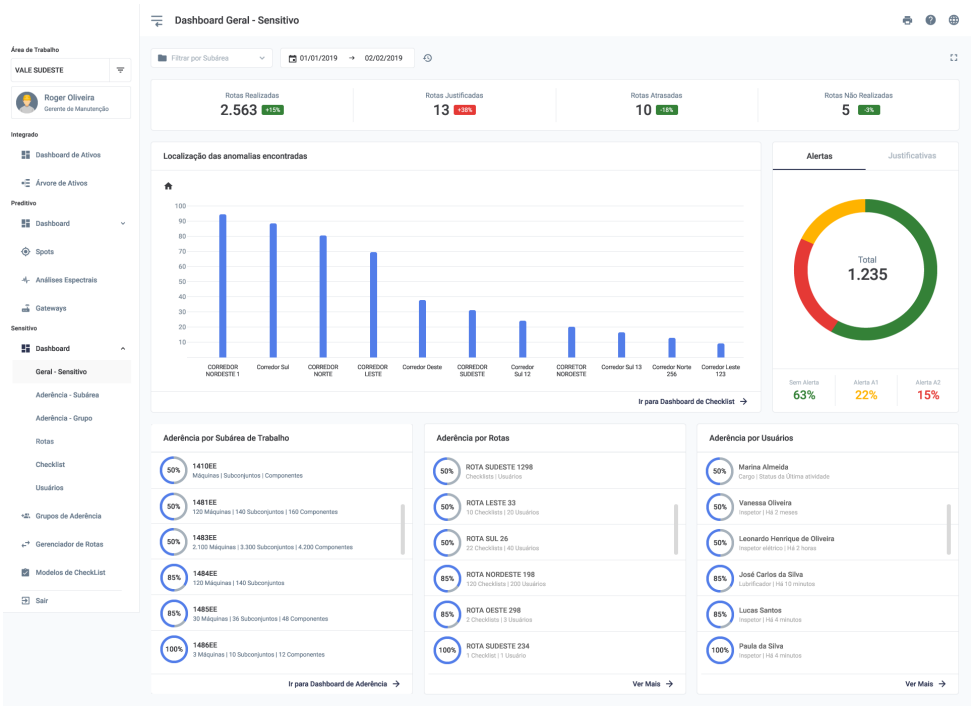
2.2 EQUIPE

A Dynamox é constituída de diversos setores, como: Comercial, Marketing, Financeiro, Recursos Humanos, Desenvolvimento, entre outros. Nesse contexto, a equipe responsável pelo planejamento, desenvolvimento e validação do aplicativo pertence ao setor de Desenvolvimento.

Atualmente a equipe do aplicativo é formada por diversos profissionais de diferentes áreas:

- Um coordenador, que junto ao gerente do setor de Desenvolvimento é responsável por definir metas e objetivos do projeto, gerenciar cronogramas, alocar recursos, assegurar a qualidade do *software* e promover o desenvolvimento profissional da equipe;

Figura 5 – *Dashboards* apresentados no DynaPredict Web



Fonte: (DYNAMOX, 2023b).

- Um gerente de projetos, que têm responsabilidades similares ao coordenador, atuando junto ao mesmo na gerência de cronogramas e objetivos do projeto. Entretanto, a atuação do gerente de projetos está estritamente voltada ao projeto, não sendo responsável pela equipe de desenvolvimento;
- Um analista de qualidade, que tem como responsabilidade garantir a qualidade do produto final. Para isso, suas funções dizem respeito à realização de testes, avaliando a usabilidade, a consistência de fluxos e a presença de erros no aplicativo, garantindo que os requisitos do produto desenvolvido foram atendidos. Além disso, o analista de qualidade é responsável por desenvolver alguns testes automatizados, como (*smoke testing*) e testes de interface de usuário;
- Um designer, que é responsável pelo desenvolvimento gráfico das interfaces de usuário, garantindo que o design do aplicativo atenda às necessidades do usuário e esteja alinhado com os objetivos do projeto. Além disso, comumente as interfaces desenhadas estão dispostas em um fluxo lógico, guiando a solução construída pelos desenvolvedores;
- Doze desenvolvedores, sendo cinco desenvolvedores para sistema operacional iOS, cinco para Android e dois para Windows. Os desenvolvedores têm como objetivos o desenvolvimento de novas funcionalidades e resolução de problemas

no aplicativo. Para isso, são responsáveis pela arquitetura, desenvolvimento e testes a nível de *software*.

Vale ressaltar que, com o intuito de padronizar a lógica de negócio entre os aplicativos para Android e iOS, a equipe do aplicativo iniciou recentemente o desenvolvimento de uma biblioteca de *software* interna. Para o desenvolvimento de novas funcionalidades e resoluções de problemas na biblioteca, são recorrentemente alocados alguns desenvolvedores para o sistema operacional iOS e Android da equipe.

Toda metodologia organizacional da equipe é pautada no Scrum, que será apresentado no capítulo que sucede.

3 FUNDAMENTAÇÃO TEÓRICA

No presente capítulo serão apresentados alguns conceitos relevantes para a compreensão do trabalho, tendo o enfoque em metodologias utilizadas para o desenvolvimento de sistemas na área de aplicativo mobile. Serão abordados tanto conceitos organizacionais, referentes à equipe de desenvolvimento como um todo, quanto conceitos técnicos de programação, como: arquitetura, boas práticas, testes automatizados, entre outros.

3.1 SCRUM

O Scrum é uma metodologia ágil de gerenciamento de projetos que foi criada para lidar com a complexidade e a incerteza envolvidas no desenvolvimento de *software*.

Antes do surgimento do Scrum, os projetos de *software* eram frequentemente gerenciados usando uma abordagem em cascata, na qual cada fase do projeto é necessariamente concluída antes de passar para a próxima. No entanto, essa abordagem se mostrou inflexível e difícil de seguir, pois muitas vezes os requisitos mudam durante o desenvolvimento (COCCO *et al.*, 2011).

Ao contrário das abordagens em cascata, o Scrum propõe a realização de pausas regulares durante a execução do projeto, a fim de avaliar se o trabalho está progredindo na direção correta e se os resultados estão alinhados com as expectativas dos envolvidos. Além disso, busca-se melhorar a eficiência e identificar obstáculos que impedem o progresso (SUTHERLAND, 2014). Sendo assim, a metodologia Scrum ocorre de maneira cíclica, introduzindo reuniões que visam analisar o trabalho realizado e elencar as atividades que serão desenvolvidas a cada ciclo de trabalho (Sprint).

Antes de cada Sprint há uma reunião de planejamento, denominada Sprint Planning, onde a equipe determina a quantidade de trabalho e quais as tarefas que poderão ser concluídas no Sprint. As tarefas são escolhidas a partir de uma lista de prioridades denominada Product Backlog, que é constantemente alimentada com novas demandas ou problemas no sistema. Ao final do Sprint, a equipe realiza uma reunião denominada Sprint Retrospective para avaliar o que foi realizado, refletindo se a quantidade de trabalho proposto foi adequada, sendo possível estabelecer um ritmo de trabalho e identificar a velocidade do desenvolvimento (SUTHERLAND, 2014).

Além disso, durante um Sprint é comum a realização de reuniões diárias denominadas Daily Scrum, que têm o objetivo de fortalecer o alinhamento entre os membros da equipe. Nessa reunião, cada membro da equipe deve trazer informações relacionadas às tarefas que executou no dia anterior para ajudar a equipe a concluir o Sprint, as tarefas que pretende desempenhar até a próxima Daily Scrum e os obstáculos que estão sendo enfrentados. Vale ressaltar que idealmente uma Daily Scrum não deve

demorar mais do que quinze minutos (SUTHERLAND, 2014).

Para o desenvolvimento do presente trabalho foram utilizados Sprints com duração de duas semanas, após esse período, eram realizadas as reuniões de Sprint Retrospective para avaliar o que foi desenvolvido e o Sprint Planning para preparar o próximo Sprint. O Daily Scrum também foi realizado. Para auxiliar no gerenciamento do Sprint foi utilizado um sistema do tipo Kanban.

3.2 PROGRAMAÇÃO ORIENTADA A OBJETOS

A programação orientada a objetos (POO) é um paradigma de programação que organiza o código em torno de objetos. Esses objetos possuem características (atributos) e comportamentos (métodos) que definem sua estrutura e funcionalidade (FARINELLI, 2007).

Alguns outros conceitos são necessários para o entendimento da POO, como:

- **Classe:** define um tipo de objeto, especificando os atributos e métodos que o objeto terá (RICARTE, 2001);
- **Herança:** é um mecanismo que permite que uma classe obtenha características (atributos e métodos) de outra classe. A classe que herda é chamada de classe filha ou subclasse, enquanto a classe da qual a herança é realizada é chamada de classe mãe, pai ou superclasse. A herança permite a reutilização de código, evitando a duplicação e promovendo a extensibilidade do *software* (FARINELLI, 2007);
- **Abstração:** é o processo que busca simplificar e restringir o uso dos métodos de uma classe. Em Swift (linguagem de programação utilizada no desenvolvimento do projeto) a abstração é alcançada através do uso de protocolos. Os protocolos podem ser utilizados junto ao conceito de herança, definindo um conjunto de métodos que uma classe deve implementar, sendo que uma classe pode herdar/implementar vários protocolos.

Com isso, a POO permite a criação de programas modulares, permitindo melhorar a reusabilidade e extensibilidade dos *softwares* (FARINELLI, 2007), sendo um paradigma comumente utilizado no desenvolvimento de aplicativos.

3.3 VERSIONAMENTO DE CÓDIGO

O versionamento de código consiste em acompanhar e controlar as alterações feitas nos arquivos de um *software* ao longo do tempo. Ele possibilita que várias pessoas trabalhem no mesmo projeto, rastreiem e gerenciem alterações, revertam

para versões anteriores, se necessário, e facilitem a colaboração entre os membros da equipe.

Ainda, existem diversos tipos de sistemas de controles de versões, como os sistemas de controle de versão centralizados (SCVCs) e os sistemas de controle de versão distribuídos (SCVDs). Os SCVCs buscam solucionar o problema de colaboração entre desenvolvedores em diferentes máquinas. Nesse modelo, há um servidor único que armazena todos os arquivos versionados, e os clientes verificam esses arquivos a partir desse repositório central. Uma grande desvantagem dos SCVCs é o ponto único de falha, o que aumenta o risco de todo o histórico do projeto ser perdido. Já os SCVDs espelham completamente o repositório. Isso significa que se o servidor falhar, qualquer repositório do cliente pode ser usado como *backup*. Além disso, esses sistemas têm capacidade para lidar de forma eficiente com múltiplos repositórios remotos, permitindo colaboração simultânea com diversos grupos de pessoas em um projeto (CHACON; STRAUB, 2014).

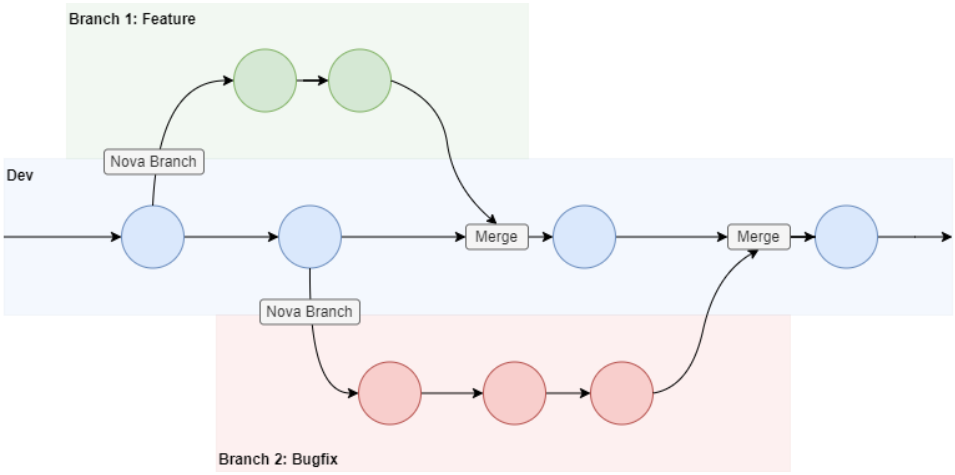
Nesse contexto, um dos mais conhecidos SCVDs é o Git, que utiliza uma estrutura de ramificação para criação de diferentes versões do código. Essas ramificações de código são conhecidas como *branches* e nada mais são do que cópias independentes do código.

O fluxo de desenvolvimento e utilização do Git pode ser entendido com auxílio da Figura 6, e será melhor detalhado abaixo:

- Criação do repositório: Inicialmente, é criado um repositório Git para o projeto;
- Criação da *branch* principal: a *branch* principal é criada para representar o estado estável do código. Todas as alterações nessa *branch* são consideradas prontas para serem implantadas em produção, na Figura 6 ela pode ser identificada pelo nome “Dev”.
- Criação de uma nova *branch*: sempre que há a necessidade de realizar alterações no código (novas funcionalidades ou correção de *bugs*), uma nova *branch* é criada a partir da principal. Na Figura 6 são utilizadas como exemplo uma *branch* de nova funcionalidade (em verde) e uma de correção de *bug* (em vermelho). Vale ressaltar que as novas *branch* podem ser criadas a partir de qualquer versão da *branch* principal;
- Desenvolvimento de *commits*: um *commit* é uma forma de salvar e registrar as alterações feitas, devendo ser pequeno e focado em uma única funcionalidade. Cada *branch* pode ter quantos *commits* forem necessários. Na Figura 6 os *commits* são identificados pelos círculos, sendo que a *branch* representada pela cor verde tem dois *commits*, por exemplo.

- Integração: quando uma *branch* é concluída com as alterações localmente testadas, é hora de integrá-la de volta à *branch* principal. Isso é feito através de um processo chamado *merge*.
- Pull Request (PR): se o desenvolvimento for realizado por uma equipe é comum que haja antes do *merge* a abertura de um PR. O PR é uma solicitação de revisão das alterações realizadas na *branch* que será unida à *branch* principal. Essa revisão é realizada pelos membros da equipe de desenvolvimento.

Figura 6 – Fluxo do versionamento de código Git



Fonte: Adaptado de (FLAVIO ANTUNES, 2021).

O projeto em questão foi desenvolvido inteiramente utilizando o fluxo Git supracitado. Vale ressaltar que, a equipe do aplicativo para iOS utiliza duas *branches* principais, sendo denominadas Dev e Master. A Master está diretamente associada ao aplicativo disponibilizado para o cliente, enquanto a Dev é a *branch* principal da equipe de desenvolvimento, sendo atualizada constantemente. Em datas pré-determinadas há o processo de *merge* entre as duas *branches*, para que a Master seja atualizada e uma nova versão do aplicativo possa ser lançada na loja; esse processo é denominado Release.

3.4 ARQUITETURA DE SOFTWARE

A arquitetura de *software* compreende a estrutura e organização de um sistema de *software*. Em outras palavras, define a forma como os componentes do sistema são arranjados e como interagem entre si.

O propósito do uso de alguma arquitetura de *software* não diz respeito a sua funcionalidade, mas sim em dar suporte ao ciclo de vida, facilitando a implantação,

manutenção e desenvolvimento contínuo do *software* contida nela (MARTIN, R., 2017); minimizando o custo de vida útil do sistema e maximizando a produtividade do programador.

A escolha da arquitetura de *software* adequada depende dos requisitos e objetivos do sistema, bem como das restrições e necessidades do projeto. Existem várias abordagens e estilos de arquitetura de *software*, como a arquitetura em camadas, arquitetura cliente-servidor, arquitetura orientada a serviços (SOA), entre outros. Cada uma dessas abordagens tem suas vantagens e desvantagens, e a escolha correta depende do contexto específico do projeto.

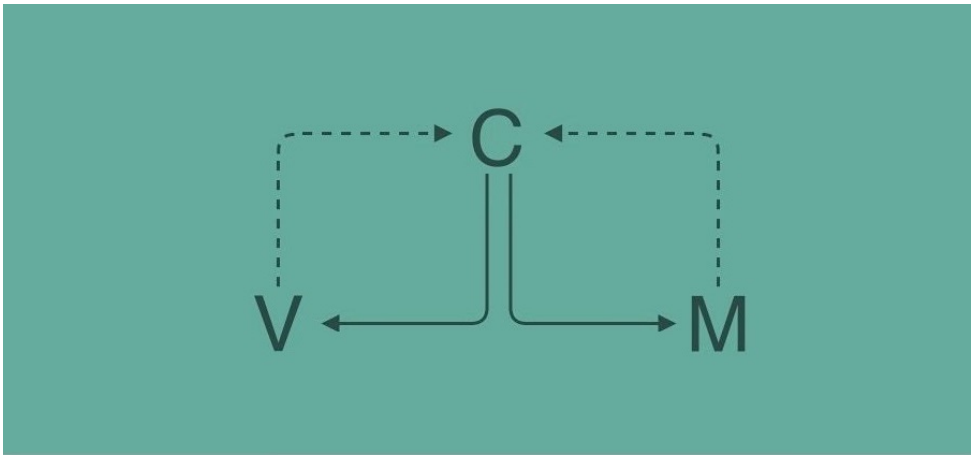
Nesse contexto, a presente subseção abordará diferentes padrões de arquitetura utilizados em aplicativos para sistema operacional iOS.

3.4.1 MVC

O MVC (*Model*, *View*, *Controller*) é um padrão de arquitetura utilizado para construir *software* no iOS. Ele consiste em três componentes principais: *Models*, que são responsáveis pelo armazenamento de dados e lógica de negócios; *Views*, que são responsáveis pela interface de usuário; e *Controllers*, que são responsáveis por conectar os *Models* e as *Views* em um aplicativo funcional (HUDSON, 2018).

O relacionamento entre as classes do MVC pode ser melhor entendido através do fluxo da Figura 7. A entrada de um módulo MVC se dá pelo *Controller* que é responsável por atualizar a *View* e solicitar informações ao *Model*, enquanto o *Model* retorna notificações e a *View* retorna ações do usuário ao *Controler*.

Figura 7 – Ilustração do fluxo do MVC



Fonte: Adaptado de (PROSZAK, 2016).

Embora seja fácil de aprender e amplamente utilizado no desenvolvimento iOS, o MVC tem algumas desvantagens e pode sofrer complicações devido à mistura de

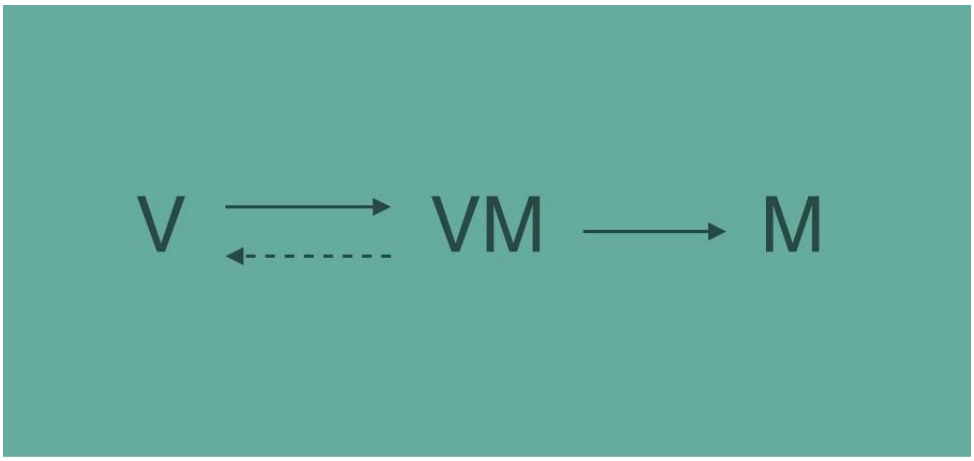
View e *Controllers* no iOS, sendo por padrão apenas um componente denominado *View Controller*. Além disso, podem haver funcionalidades que não se encaixam perfeitamente em nenhum dos componentes.

3.4.2 MVVM

O MVC é a abordagem padrão para desenvolver aplicativos Swift, mas muitos desenvolvedores acham que o design se torna muito complicado, com um grande número de códigos em um só lugar. O MVVM é uma alternativa ao MVC, que adiciona uma nova classe. A *View Model*, é a classe responsável pela validação de entradas, formatação de dados, tratamento de erros e converter dados do *Model* em valores formatados para as *Views*. As *Views* ou *View Controllers*, representados pelo acrônimo "V", representa uma junção entre *View* e *Controller* supracitados (HUDSON, 2018).

O MVVM pode ser melhor entendido através do fluxo da Figura 8. A entrada de um módulo em MVVM se dá pela *View*, que é responsável por receber ações do usuário. Dependendo da ação pode ser necessário o tratamento lógico de dados, que deve ser feito na *View Model*. Ainda, caso seja necessário acesso a algum serviço, como banco de dados ou a lógica de negócio, a *View Model* deve solicitar que o *Model* retorne essas informações para então enviá-las à *View*.

Figura 8 – Ilustração do fluxo do MVVM



Fonte: (PROSZAK, 2016).

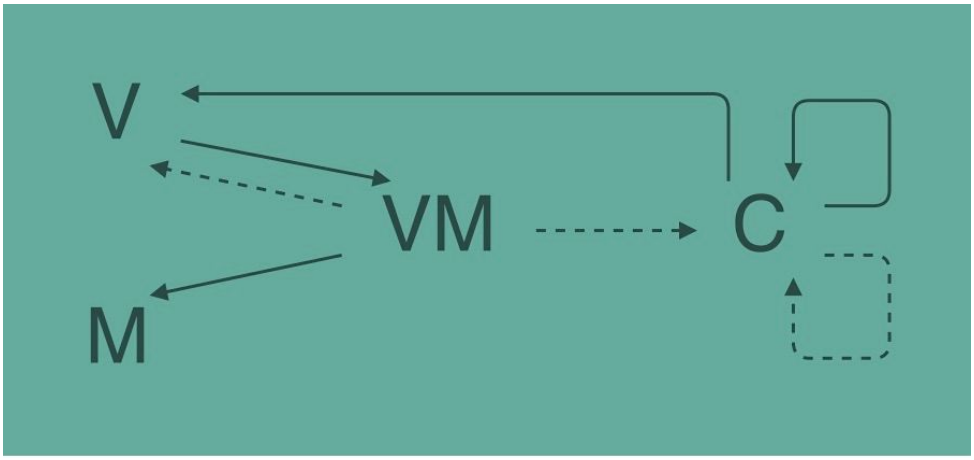
Em suma, o MVVM é uma boa alternativa para tornar a *View* menos massiva, separando a lógica de apresentação, feita na *View Model*, da apresentação em si, feita na *View*. Entretanto, ainda há críticas ao MVVM no que tange a busca por um código mais flexível e reutilizável, visto que no MVVM (assim como no MVC) a navegação é feita junto a estrutura de exibição, criando um forte acoplamento entre as *Views* de diferentes módulos, o que dificulta a reutilização de código (PROSZAK, 2016).

3.4.3 MVVM-C

O MVVM-C surge como uma alternativa para contornar o problema de acoplamento de *Views* supracitado. Para isso, o MVVM-C adiciona uma nova classe denominada *Coordinator*, que é responsável por gerenciar o fluxo de navegação entre os módulos (ŠARAVANJA, 2019).

O relacionamento entre os componentes do MVVM-C pode ser entendido através do fluxo da Figura 9. Os componentes *View*, *View Model* e *Model* têm as mesmas atribuições do modelo MVVM, salvo o fato de que a navegação não é mais responsabilidade da *View*. Assim, sempre que o módulo é acessado ou finalizado, a responsabilidade é do *Coordinator*. É importante destacar que muitos eventos de fechamento são iniciados na *View* (usuário clica em um botão "X" por exemplo), nesses casos, a *View* deve enviar os eventos recebidos para a *View Model*, que por sua vez os envia para o *Coordinator*, onde são devidamente tratados.

Figura 9 – Ilustração do fluxo do MVVM-C



Fonte: (PROSZAK, 2016).

Considerando seus benefícios, o padrão MVVM-C foi o escolhido para ser o padrão de arquitetura utilizado no trabalho em questão.

3.5 PRINCÍPIOS SOLID

A atualização do código é inerente à evolução de um *software*, e caso não haja o planejamento prévio e a devida preocupação na utilização de princípios e padrões de design, à medida que o *software* evolui, começa a apodrecer (MARTIN, R. C., 2000). Alguns sintomas desse apodrecimento são:

- **Rigidez:** diz respeito à dificuldade de realizar modificações. Em um design rígido, cada simples alteração causa uma cascata de outras alterações, que podem

se alastrar por outros módulos do sistemas, elevando o nível de dificuldade e o tempo necessário para o desenvolvimento;

- **Fragilidade:** é a probabilidade do código quebrar em diversos lugares sempre que é alterado, sendo que muitas vezes os locais em que ocorre a quebra não deveriam ser relacionados ao local em que ocorreu a modificação;
- **Imobilidade:** incapacidade em reutilizar partes do projeto (funções, classes ou mesmo *softwares* de outros projetos);
- **Viscosidade:** é a dificuldade de preservar o design de *software* ao fazer mudanças. Pode ocorrer em duas formas: viscosidade do projeto e viscosidade do ambiente. A viscosidade do projeto acontece quando é mais fácil usar soluções de contorno do que preservar o design adequado. A viscosidade do ambiente ocorre quando o ambiente de desenvolvimento é lento e ineficiente, levando a decisões de curto prazo que comprometem o design.

Nesse contexto, o SOLID é um conjunto de princípios utilizados durante o desenvolvimento de um *software* orientado a objetos que busca evitar o apodrecimento do design do sistema.

Os cinco princípios que compõem o SOLID serão descritos a seguir:

3.5.1 Princípio da Responsabilidade Única (SRP)

O SRP (Single-Responsibility Principle - S), diz que “cada classe deve ter uma única responsabilidade: deve ter um único propósito no sistema e deve haver apenas um motivo para alterá-la” (FEATHERS, 2004)

Em outras palavras, tal princípio tem foco no desacoplamento de responsabilidades, visando que uma entidade (classe ou função) realize apenas uma função específica (Figura 10).

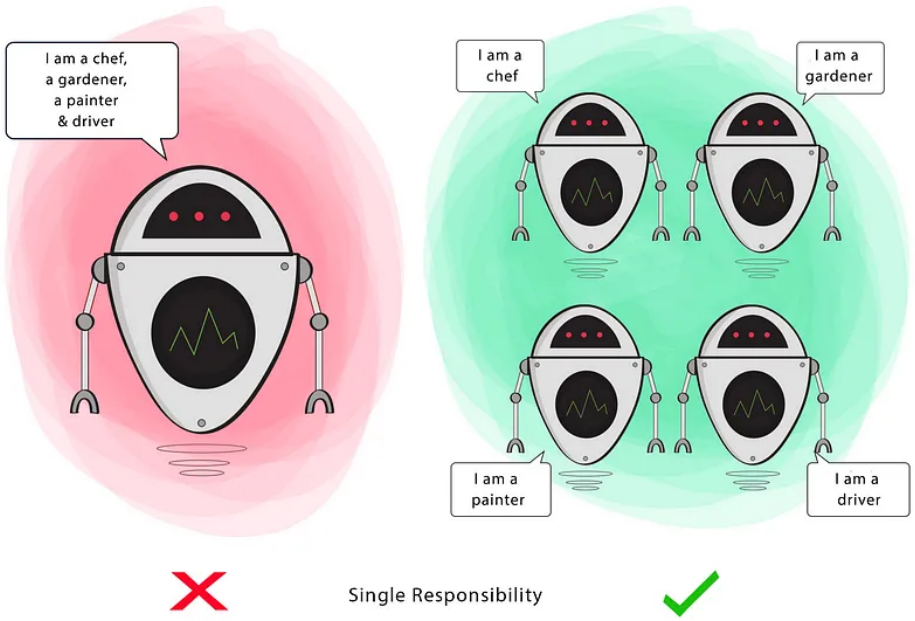
Caso o sistema desenvolvido aplique o princípio de responsabilidade única corretamente, espera-se que: caso ocorra alguma falha, necessidade de atualização ou reutilização de uma funcionalidade do projeto, apenas as entidades responsáveis por executar determinada ação sejam afetadas, trazendo menor complexidade e menos riscos ao projeto.

3.5.2 Princípio Aberto Fechado (OCP)

De acordo com o OCP (Open Closed Principle - O) “um módulo deve estar aberto para extensão, mas fechado para modificação” (MARTIN, R. C., 2000).

Em outras palavras, deve ser possível adicionar novas funcionalidades ou estender um módulo sem que seja necessário modificar as funcionalidades que o mesmo já desempenha (Figura 11).

Figura 10 – Ilustração do SRP



Fonte: (THELMA, 2020).

Com o uso do princípio em questão, espera-se que eventuais novas funcionalidades adicionadas a uma entidade não causem falhas nos lugares na qual a entidade já é utilizada, e nem comprometam as antigas funcionalidades.

3.5.3 Princípio da Substituição de Liskov (LSP)

O LSP (Liskov Substitution Principle - L) diz que “as subclasses devem ser substituíveis por suas classes base” (MARTIN, R. C., 2000).

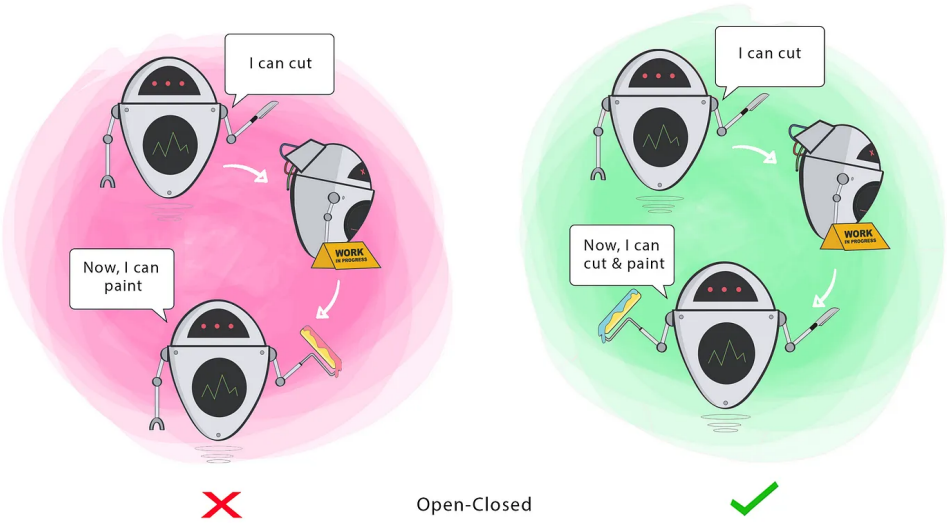
De acordo com o LSP toda subclasse construída como herança de uma superclasse deve ser capaz de executar as ações da classe herdada, entregando o mesmo resultado ou resultado do mesmo tipo sem causar falhas ou exceções.

Na Figura 12 pode-se ver um exemplo do LSP, onde existe uma superclasse que implementa um método de fazer café. Para que o princípio seja obedecido a subclasse também deve implementar esse método e retornar um resultado que faça sentido.

Outra perspectiva de entender o LSP é a partir de um contraexemplo: caso haja uma superclasse denominada “Ave” que implementa o método “voar”, não seria possível implementar uma subclasse do tipo “pinguim”, pois o método “voar” não poderia ser implementado. Nesse caso, a superclasse deveria ser reestruturada para obedecer o LSP.

Com o princípio sendo respeitado, espera-se que as subclasses possam usar

Figura 11 – Ilustração do OCP



Fonte: (THELMA, 2020).

os métodos herdados sem gerar falhas.

3.5.4 Princípio de Segregação de Interface (ISP)

O ISP (Interface Segregation Principle - I) diz respeito à ideia de que “muitas interfaces específicas do cliente são melhores do que uma interface de uso geral.” (MARTIN, R. C., 2000).

Cada classe deve apenas executar as ações necessárias para cumprir seu papel. Caso haja uma classe (classe principal) com métodos utilizados por várias outras classes (classes clientes), as classes cliente não devem ter acesso a todos os métodos da classe principal, e sim a abstrações (procolos na linguagem Swift) que implementam apenas os métodos utilizados (Figura 13).

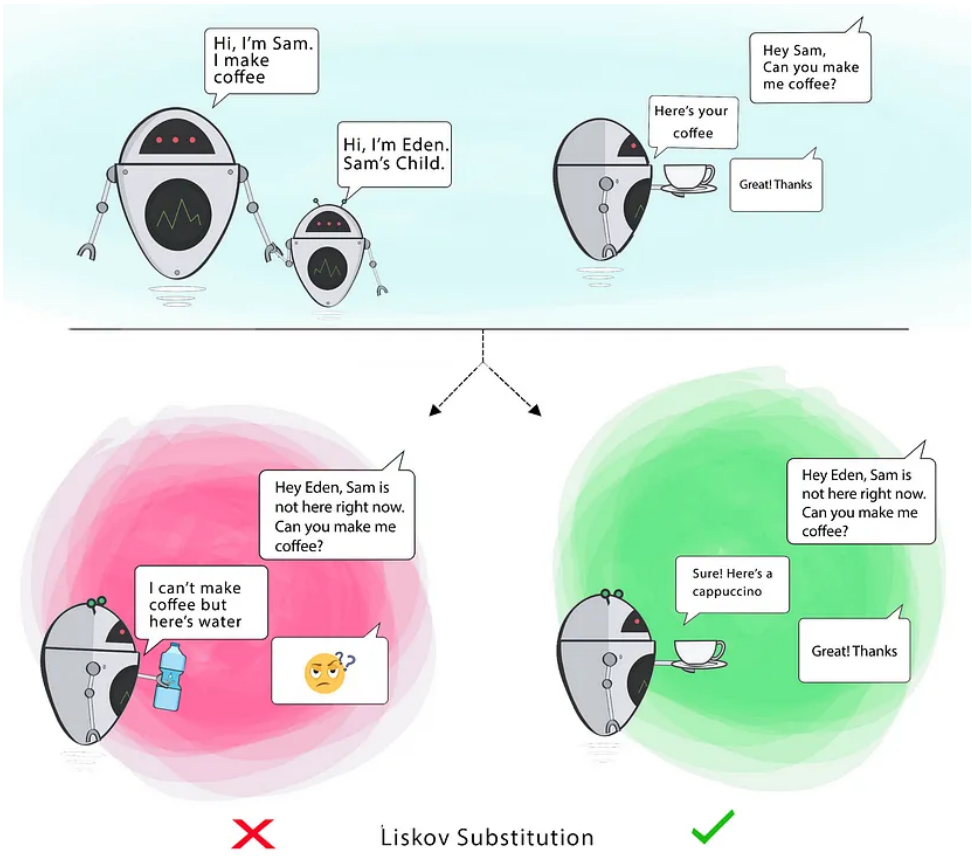
Caso o ISP não seja respeitado, uma classe cliente tem permissão para executar ações inúteis, que podem causar falhas ou fornecer resultados sem sentido.

3.5.5 Princípio da Inversão de Dependência (DIP)

O DIP (Dependency Inversion Principle - D) é descrito como: “Depender de abstrações. Não dependa de concreções.” (MARTIN, R. C., 2000).

O DIP pode ser exemplificado fazendo o uso de dois módulos. O primeiro executa uma ação e depende de uma ferramenta que é representada pelo segundo. Para o DIP ser respeitado, o módulo que executa a ação não deve depender de detalhes da implementação de outro módulo diretamente, e sim deve haver uma abstração nessa conexão (Figura 14).

Figura 12 – Ilustração do LSP



Fonte: (THELMA, 2020).

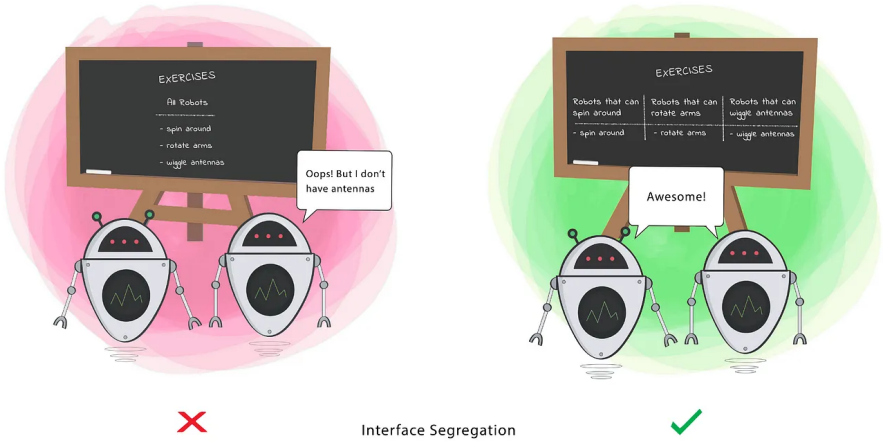
Com o DIP espera-se uma maior versatilidade, permitindo que uma classe possa implementar e utilizar métodos de diferentes classes auxiliares sem maiores problemas.

3.6 TESTES AUTOMATIZADOS

Os testes automatizados tem o foco principal de aprimorar e garantir a qualidade dos sistemas de software. Eles são programas ou *scripts* simples que executam as funcionalidades do sistema em teste e realizam verificações automáticas nos resultados obtidos. A grande vantagem dessa abordagem é que todos os casos de teste podem ser facilmente repetidos a qualquer momento, com pouco esforço e de forma rápida.(BERNARDO; KON, 2008).

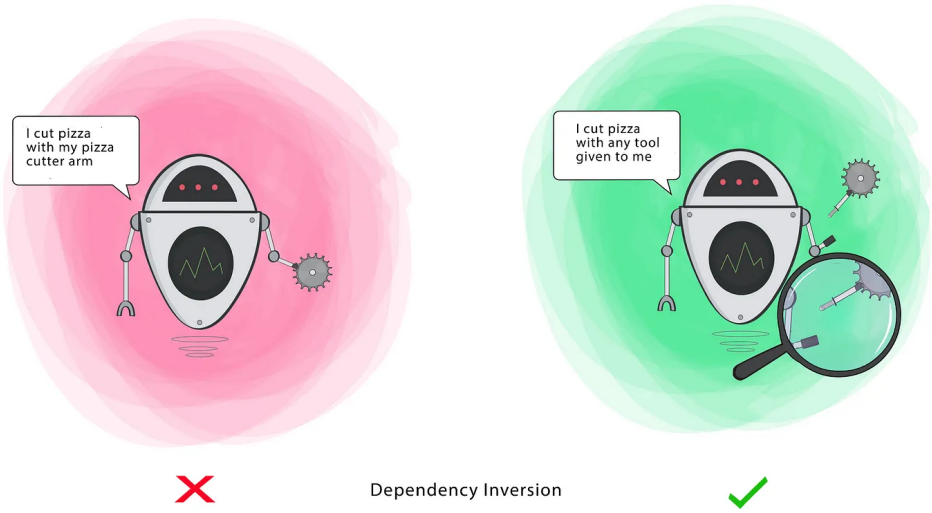
Até a década de 1990, os testes de *software* eram considerados menos importantes, sendo comum apenas pedir a um desenvolvedor para testar o produto. Entretanto, com as mudanças na indústria em busca de maior qualidade nos produtos, os testes automatizados se tornaram necessários, haja visto os possíveis erros humanos,

Figura 13 – Ilustração do ISP



Fonte: (THELMA, 2020).

Figura 14 – Ilustração do DIP



Fonte: (THELMA, 2020).

especificações malfeitas, esquecimentos ou inconsistências durante a concepção ou manutenção de um *software* (WAZLAWICK, 2010).

Os testes de *software* podem ser categorizados em diferentes grupos com base em seus objetivos, duas dessas categorias serão apresentadas nessa seção.

3.6.1 Teste de unidade

Por definição: “um teste de unidade é uma parte de um código (geralmente um método) que invoca outra parte do código e depois verifica a exatidão de algumas suposições. Se as suposições estiverem erradas, o teste de unidade falhou” (OSHEROVE, 2009). O teste de unidade recebe esse nome pois o objeto testado (unidade) refere-se a uma parte isolada e independente do código.

Nesse e nos demais testes a parte de código testada é constantemente chamada de SUT (*System Under Test*). Assim, o teste de unidade, assim como outros, é realizado contra um SUT.

No que diz respeito a boas práticas de implementação, um teste de unidade deve ser automatizado, repetível, fácil de implementar, reutilizável, acessível a qualquer pessoa, executado com um único toque de botão e executado rapidamente (OSHEROVE, 2009).

3.6.2 Teste de integração

Um teste de integração consiste em testar duas ou mais partes de código dependentes como um grupo (OSHEROVE, 2009). Esse tipo de teste tem como objetivo avaliar se os objetos testados se comunicam adequadamente (WAZLAWICK, 2010).

Nos testes de integração, assim como nos testes de unidade, considera-se o sucesso de operação quando o resultado do SUT está em conformidade com suposições previamente estabelecidas.

Portanto, os testes de unidade e de integração visam garantir a qualidade do *software*, entretanto, atuam de formas distintas. O teste de unidade testa funcionalidades de maneira isolada. Enquanto o teste de integração busca avaliar se a comunicação entre os objetos e funcionalidades estão corretas.

4 PLANEJAMENTO

No presente capítulo serão apresentados os aspectos relacionados ao pré-desenvolvimento do projeto. Visto que a metodologia de gerenciamento utilizada para o projeto é o Scrum, a etapa de planejamento não foi inteiramente realizada de forma prévia ao desenvolvimento do projeto, ocorrendo de maneira cíclica, como apresentado em seções anteriores.

As tarefas foram elencadas a cada *Sprint*, concomitante ao desenvolvimento de etapas anteriores. O levantamento dos requisitos para a criação de tarefas sempre ocorria, no mínimo, um *Sprint* antes do desenvolvimento das mesmas, gerando o *Product Backlog* do projeto.

As tarefas com descrições que representam seus requisitos e objetivos estão contidas em cartões de um quadro Kanban, o que gera informações modulares que condizem com a tarefa realizada, e não do projeto como um todo. Para melhor organização, todas as tarefas relacionadas ao projeto estão associadas a uma tarefa pai.

É importante mencionar que boa parte do planejamento e descrição dessas tarefas foram realizados pelo Coordenador e pelo Gerente de Projetos da equipe, apenas ficando a cargo do autor a criação de tarefas que eventualmente surgiram durante a execução de uma outra.

Com isso, o presente capítulo aborda inicialmente uma visão generalista do projeto desenvolvido e posteriormente apresenta cada uma das tarefas que serviram de base para o desenvolvimento do projeto.

4.1 VISÃO GERAL

As ferramentas desenvolvidas visam disponibilizar aos usuário do aplicativo DynaPredict a possibilidade da realização de monitoramento para análise espectral/análise de forma de onda em mais de um DynaLogger simultaneamente.

Para melhor entendimento, o projeto pode ser dividido em algumas etapas que serão abordadas a seguir.

4.1.1 Tour da nova funcionalidade

O Tour da nova funcionalidade consiste em um conjunto de textos e imagens, que expõem aos leitores informações gerais sobre o novo módulo e auxiliam na utilização.

Vale ressaltar que o tour deve aparecer apenas na primeira vez que o usuário acessa o módulo, e assim que é completado não deve mais estar disponível.

Enquanto o usuário não realiza o tour o botão de acesso da nova funcionalidade deve ficar marcado com um ponto azul, como uma indicação de novidade no aplicativo.

4.1.2 Agendamento do monitoramento

O primeiro conjunto de funcionalidades tem por objetivo principal agendar o monitoramento nos DynaLoggers. Para tal, conta com 3 diferentes passos:

1. **Escolher Spots:** o primeiro passo possibilita ao usuário selecionar ponto de monitoramento (*spots*) com DynaLoggers em que o monitoramento será agendado. Inicialmente há uma varredura (*scan*) que visa encontrar os DynaLoggers próximos (a uma distância que os dispositivos consigam se comunicar via Bluetooth) ao dispositivo móvel. A primeira varredura ocorre assim que o passo de Escolher Spots é acessado, entretanto o usuário pode executá-la novamente quando desejar.

   Com os DynaLoggers devidamente encontrados, o usuário deve selecionar de um a quatro periféricos de mesmo modelo ou de modelos compatíveis antes de prosseguir para o próximo passo;

2. **Confirmar Medições:** o segundo passo é iniciado com uma tentativa de conexão aos DynaLoggers anteriormente selecionados. Caso ocorra sucesso e o DynaLogger tenha dados de monitoramentos anteriores, estes dados são baixados e salvos no aplicativo, liberando espaço na memória do DynaLogger para a realização do novo monitoramento. Caso haja falha em algum desses processos, o DynaLogger não poderá ser utilizado

   Ainda, no mesmo passo é configurado o monitoramento que será realizado. Nessa etapa são definidos os eixos, podendo assumir o valor de “Eixos X, Y e Z”, “X”, “Y” ou “Z”; a frequência máxima do monitoramento, com valores disponíveis que variam com o modelo, versão de *firmware* do DynaLogger e eixo selecionado; duração do monitoramento, podendo assumir valores que variam de acordo com o modelo, versão, eixo e frequência anteriormente selecionada; e tempo até o início do monitoramento, que pode ser de dez a sessenta minutos;

3. **Confirmar Agendamento:** durante essa etapa o monitoramento previamente configurado é agendado nos DynaLoggers.

   Caso haja falha no agendamento, o usuário poderá realizar novamente a tentativa de agendamento, desde que o momento da falha não tenha ocorrido posteriormente ao tempo estipulado para início do monitoramento, configurado na etapa anterior. Caso um ou mais DynaLoggers obtenham sucesso no agendamento o usuário poderá encerrar com sucesso a etapa de agendamento.

4.1.3 Notificações

Assim que a etapa de agendamento é finalizada, uma notificação confirmando o agendamento deve ser apresentada para o usuário. A notificação deve apresentar o horário em que o monitoramento será iniciado. O módulo fica inacessível até que o monitoramento seja finalizado.

Ainda, outra notificação deve ser apresentada assim que o monitoramento for finalizado, informando o sucesso da operação. Após essa notificação, o botão de acesso da nova funcionalidade deve ficar marcado com um ponto vermelho, indicando a necessidade da coleta dos dados adquiridos pelos DynaLoggers durante o monitoramento.

Após a notificação o acesso ao módulo é liberado, porém, caso acessado, direciona o usuário para a etapa de Coleta dos Dados dos DynaLoggers utilizados.

4.1.4 Coleta dos dados

Após todas as etapas supracitadas serem realizadas, a etapa de coleta pode ser acessada. Esse conjunto de funcionalidades tem por objetivo principal a coleta e salvamento dos dados do monitoramento realizado pelos DynaLoggers, e pode ser dividido em dois passos:

1. **Coleta de dados:** durante essa etapa o aplicativo se conecta aos DynaLoggers utilizados. Quando a conexão é estabelecida o aplicativo baixa e salva os dados do monitoramento. Vale ressaltar que, erros podem ocorrer durante a etapa e o aplicativo deve possibilitar que o usuário realize o processo novamente em caso de falha. Ainda, caso as operações sejam finalizadas com sucesso, não deve ser possível coletar os dados novamente (visto que terão sido apagados do DynaLogger).

2. **Visualização dos dados:** após o sucesso na etapa de coleta, o usuário deve ter a possibilidade de visualizar graficamente os dados obtidos, sendo capaz de realizar interações com os gráficos gerados.

4.1.5 Envio dos dados para o servidor

Após todas as etapas anteriores serem realizadas, o aplicativo deve enviar os dados do monitoramento para a nuvem, juntamente com atributos que identificam os DynaLoggers responsáveis pelo monitoramento.

4.2 TAREFAS PROPOSTAS

Na presente seção serão apresentadas todas as tarefas propostas para o desenvolvimento do módulo. Como anteriormente descrito, tais tarefas eram criadas no

Product Backlog durante a etapa de desenvolvimento de tarefas anteriores. A cada reunião de Sprint Planning as tarefas eram introduzidas no *Sprint*.

As tarefas criadas tem tipos diferentes que facilitam a organização, podendo ser de um dos tipos descritos abaixo:

- **História:** as tarefas do tipo história são utilizadas para representar um requisito de alto nível do usuário ou do cliente (nesse contexto o cliente não é necessariamente externo à empresa, podendo ser uma outra equipe, por exemplo). Tarefas desse tipo podem apresentar textos indicando a descrição inicial, o que deve ser realizado, porque será realizado e o que é esperado para a tarefa ser considerada realizada (DoD);
- ***Feature:*** as tarefas desse tipo representam novas funcionalidades a serem implementadas e podem conter campos com informações que dizem respeito ao que deve ser feito, como está no aplicativo e sugestão de como deve ficar o aplicativo após a implementação;
- **Problema:** representam problemas que devem ser solucionados. É comum evidenciar em tarefas desse tipo textos que indicam o problema, versão do código em que o problema ocorre, fluxo para reproduzir, o que ocorre ao realizar o fluxo e o que deveria ocorrer;
- **Épico:** tarefas desse tipo são utilizadas para novos módulos no aplicativo ou refatorações que envolvem grande quantidade de código. Essas tarefas têm como característica conter regras de negócio, *layouts* de telas e tarefas filhas associadas.

Para o desenvolvimento do projeto, inicialmente foram criadas pelo Coordenador e pelo Gerente de Projetos algumas tarefas do tipo história. Em etapas posteriores do desenvolvimento foram criadas pelo autor algumas tarefas do tipo *feature*, devido a requisitos que não foram concluídos nas tarefas inicialmente criadas. Todas foram associadas a uma tarefa do tipo Épico com o nome Espectral Simultânea, que é o nome comercial do módulo desenvolvido no presente projeto.

Todas as tarefas utilizadas para o desenvolvimento podem ser visualizadas em ordem cronológica nas subseções apresentadas abaixo.

4.2.1 Botão de acesso para a Espectral Simultânea e marcador no menu

"Como usuário do DynaPredict, eu quero visualizar a *feature* Espectral Simultânea no menu lateral do App."

- **O quê**/**Quero:** visualizar o novo módulo no menu lateral do DynaPredict e o ícone indicativo de novidade (marcador azul) no menu;

- **Por quê/Para quê:** utilizar a Espectral Simultânea para realizar diversas espectrais simultaneamente.

4.2.2 Tela navegável

"Como usuário do DynaPredict, eu quero navegar entre as páginas que compõem a etapa de agendamento da Espectral Simultânea."

- **O quê/Quero:** desejo uma tela navegável (*view pagger*) com suporte aos passos da etapa de agendamento;
- **Por quê/Para quê:** poder navegar entre as páginas 1 a 3 do agendamento da Espectral simultânea;
- **DoD:** tela navegável que permita acessar os passos através de botões de avançar e voltar, indicando o passo atual (Escolher Spots, Confirmar Medições ou Confirmar Agendamento) ao usuário.

4.2.3 Step 1 - scan/seleção de dynaLogger

"Como usuário do DynaPredict, eu quero realizar o *scan* e seleção de DynaLogger (*step* 1) da Espectral Simultânea."

- **O quê/Quero:** visualizar DynaLoggers próximos sendo capaz de selecioná-los;
- **Por quê/Para quê:** selecionar DynaLoggers para realização da Espectral Simultânea;
- **DoD:** listar somente DynaLoggers próximos e com modelo que dê suporte para Espectral Simultânea (modelos capazes de realizar o monitoramento de forma agendada), sendo capaz de selecionar os DynaLoggers.

4.2.4 Step 2.1 - funcionalidade de conexão com DynaLogger

"Como usuário do DynaPredict, eu quero poder conectar com dynalogger (*step* 2.1) para posteriormente agendar espectral simultânea."

- **O quê/Quero:** conectar com DynaLoggers;
- **Por quê/Para quê:** testar a conexão e limpar a memória dos DynaLoggers para possibilitar a realização do monitoramento;
- **DoD:** conectar com os DynaLoggers selecionados no *step* 1 da Espectral Simultânea, pausar monitoramento contínuo dos DynaLoggers, baixar dados e tratar erros de conexão. Quando ocorrer sucesso em todos os casos o usuário deve

ser redirecionado para tela de configurações do monitoramento da Espectral Simultânea.

4.2.5 Step 2.2 - tela de configuração de espectral

"Como usuário do DynaPredict, eu quero poder configurar o monitoramento para análise espectral (*step* 2) no novo módulo de Espectral Simultânea."

- **O quê/Quero:** visualizar formulário de configuração;
- **Por quê/Para quê:** configurar eixos, frequência máxima, duração e tempo até o início do monitoramento.

4.2.6 Step 3 - tela de progresso do agendamento

"Como usuário do DynaPredict, eu quero poder agendar o monitoramento no DynaLogger com base nas configurações do (*step* 2)."

- **O quê/Quero:** implementar a funcionalidade de conexão com os DynaLoggers, para comunicar o pedido de um monitoramento com as configurações selecionadas no passo anterior;
- **Por quê/Para quê:** realizar o agendamento do monitoramento da Espectral Simultânea;
- **DoD:** conectar com os DynaLoggers selecionados e configurados nos *steps* 1 e 2 da Espectral Simultânea. Enviar o pedido de monitoramento ao DynaLogger e tratar erros de conexão.

4.2.7 Gestão de espectrais agendadas, notificação e marcador no menu

"Como usuário do DynaPredict, eu quero visualizar notificações de espectral agendada com sucesso e espectral pronta pra *download*. Também quero um indicador no botão de menu e de acesso à Espectral Simultânea após o monitoramento finalizado."

- **O quê/Quero:** ao solicitar uma espectral simultânea, incluir *push notification* de espectral agendada e *push notification* de espectral concluída. Incluir marcador vermelha no menu e botão de acesso ao módulo;
- **Por quê/Para quê:** poder visualizar quando a espectral foi agendada e quando está pronta pra *download*;
- **DoD:** gerar notificações e adicionar ícone indicativo (bolinha vermelha) no botão de acesso ao menu e botão de acesso ao módulo.

4.2.8 Tour da Espectral Simultânea

"Como usuário do DynaPredict, eu quero visualizar uma única vez o tour de apresentação da espectral simultânea e do ícone de novidade (bolinha azul)."

- **O quê/Quero:** visualizar o tour de apresentação da Espectral Simultânea;
- **Por quê/Para quê:** apresentar ao usuário a *feature* desenvolvida;
- **DoD:** apresentar tour somente para usuários novos, tratar gestão da apresentação dos ícones, bolinha vermelha com maior prioridade.

4.2.9 Step 4.1 - tela de gestão de download

"Como usuário do DynaPredict, eu quero poder conectar com DynaLogger (step 4) e baixar os dados do monitoramento"

- **O quê/Quero:** obter os dados de monitoramento dos DynaLogger para salvar e visualizar resultados;
- **Por quê/Para quê:** guardar e visualizar os resultados da espectral;
- **DoD:** tratar erros de conexão e *download*, caso haja sucesso em todos os casos o marcador vermelho deve desaparecer do botão de acesso ao menu e botão de acesso a Espectral Simultânea. Deve haver a possibilidade de tentar novamente, indicadores de progresso e estado dos processos realizados.

4.2.10 Step 4.2 - implementar visualização de gráficos

"Como usuário do DynaPredict, eu quero poder visualizar graficamente os resultados do monitoramento da Espectral Simultânea."

- **O quê/Quero:** visualizar dados obtidos em gráficos do espectro e forma de onda;
- **Por quê/Para quê:** analisar prévia dos resultados;
- **DoD:** apresentação de gráficos do espectro e forma de onda, capacidade de esconder e aparecer eixos e capacidade de trocar visualização entre dados de velocidade e aceleração.

4.2.11 Implementar varredura de DynaLogger

- **O quê:** implementar varredura (*scan*) de DynaLoggers previsto para as etapas de Escolher Spots (*step* 1), Confirmar Agendamento (*step* 3) e Coleta de Dados (*step* 4.1) da Espectral Simultânea;

- **Como está:** *scan* ainda não implementado por conta da refatoração na comunicação com os DynaLoggers;
- **Sugestão de como deve ficar:** implementar *scan* com tempo de vinte segundos, indicativo de progresso de tempo do *scan* e atualização do sinal recebido (RSSI) nas células que representam os DynaLoggers.

4.2.12 Adicionar lógica de conexão na Espectral Simultânea

- **O quê:** adicionar lógica de conexão na espectral simultânea e adaptando ao *step* 2.1;
- **Como está:** a conexão real ainda não existe no módulo;
- **Sugestão de como deve ficar:** capacidade de se conectar aos DynaLoggers retornando possíveis erros.

4.2.13 Adicionar lógica de agendamento na Espectral Simultânea

- **O quê:** o usuário deve conseguir agendar os monitoramentos para que iniciem após o tempo pré selecionado no *step* 2.2;
- **Como está:** atualmente os valores apresentados na *view* são fictícios e não há envios para os DynaLoggers;
- **Sugestão de como deve ficar:** conseguir agendar o monitoramento nos DynaLoggers. O intervalo de tempo entre o início do monitoramento dos DynaLoggers deve ser o menor possível. Retornar informações para atualização da *view* do *step* 3.

4.2.14 Coleta de dados Espectral Simultânea

- **O quê:** adicionar a funcionalidade de coleta de dados dos DynaLoggers na espectral simultânea, ajustar retorno de progresso e estado para atualização da *view* do *step* 2.1 e *step* 4.1;
- **Como está:** são utilizadas informações fictícias para atualização das *views* e validação de fluxos;
- **Sugestão de como deve ficar:** coletar e salvar os dados do monitoramento dos DynaLoggers. Ajustar fluxos do *steps* 2.1 e 4.1 com os status reais da coleta de dados.

5 DESENVOLVIMENTO

No presente capítulo estão descritas as etapas que contemplam o desenvolvimento e validação do módulo de Espectral Simultânea. O capítulo inicia com a apresentação de uma visão geral do desenvolvimento, explicando o fluxo de trabalho e algumas ferramentas utilizadas durante todo o projeto. Após isso, são apresentadas e descritas as etapas específicas do desenvolvimento. Algumas dessas etapas são descritas com embasamento em diagramas de classes e diagramas de sequência, que expõem de forma clara a arquitetura e fluxos utilizados no projeto. Durante a apresentação dessas etapas, alguns conceitos abordados no Capítulo 3 são retomados, sendo possível ver de forma prática como foram utilizados durante a execução do projeto. Por fim, são apresentadas as validações utilizadas para assegurar a confiabilidade e eficiência do módulo em questão. As validações abordam o desenvolvimento dos testes automatizados, além do processo de revisão do código realizado pelos demais membros da equipe e testes realizados pela analista de qualidade da equipe.

5.1 FLUXO DE TRABALHO

Como anteriormente abordado, toda a etapa de desenvolvimento foi pautada no Scrum. As etapas do desenvolvimento foram guiadas pelos cartões de tarefas apresentados na Seção 4.2. O fluxo de trabalho foi pautado na metodologia de versionamento de código GIT, apresentado na Seção 3.3. A cada tarefa o autor criava uma nova *branch* a partir da *branch* principal, para que as alterações realizadas não afetassem em primeiro momento a estabilidade do código principal, o que poderia impactar de maneira negativa os demais desenvolvedores da equipe. A cada funcionalidade implementada durante a execução de uma tarefa, um *commit* era realizado na *branch* da tarefa, registrando as alterações feitas. Após o término da tarefa um PR era aberto, permitindo que os demais membros da equipe pudessem corrigir as alterações realizadas na *branch* da tarefa antes que fosse unida à *branch* principal.

As etapas supracitadas foram realizadas com o auxílio do *software* Sourcetree, que fornece uma interface visual que auxilia na gestão de *branches* e *commits*. Para realização do versionamento também foi utilizado o Bitbucket, que é uma plataforma de hospedagem de repositórios de controle de versão baseados em Git. Nesse contexto, a plataforma Bitbucket foi utilizada para armazenamento das versões do código de maneira remota e também foi utilizada para a correção dos PR.

Após as alterações da *branch* de tarefa serem aprovadas no PR e unidas à *branch* principal, o analista de qualidade da equipe era responsável por realizar testes no dispositivo. Caso o analista não encontrasse problemas a tarefa era finalizada com sucesso. Caso contrário, a tarefa era reaberta e o problema encontrado era sinalizado ao desenvolvedor, que deveria repetir o fluxo de trabalho.

5.2 DESENVOLVIMENTO DAS FUNCIONALIDADES

A presente seção descreve detalhadamente como as funcionalidades do sistema foram projetadas e implementadas. Durante toda etapa de desenvolvimento foi utilizada a linguagem de programação Swift, que foi desenvolvida especificamente para a criação de apps para iOS, Mac, Apple TV e Apple Watch (APPLE, 2023). Além disso, foi utilizada a plataforma de Desenvolvimento integrado (IDE) XCode, que também foi criada para auxiliar o desenvolvimento de projetos desse tipo.

Para facilitar o entendimento do trabalho, assim como em seções anteriores, o desenvolvimento das funcionalidades será dividido em etapas.

5.2.1 Tour da nova funcionalidade

Como abordado em seções anteriores, o tour da Espectral Simultânea tem por objetivo expor aos leitores informações relevantes sobre o módulo por meio de imagens e textos.

5.2.1.1 Desenvolvimento das telas

A etapa inicial do desenvolvimento do módulo foi pautada na coleta das imagens e textos que seriam apresentados. Nenhuma das imagens e textos utilizados foi criada pelo autor, ficando apenas a cargo desse, a implementação no aplicativo.

Visto que o DynaPredict está disponível em quatro idiomas, sendo esses: português, inglês, espanhol e francês, foi necessário utilizar quatro possíveis valores para cada texto apresentado. O gerenciamento dos idiomas foi realizado pelo “Localizable.strings”, que é um componente desenvolvido para suportar localização e internacionalização em aplicativos iOS. Ele consiste em arquivos separados por idioma, onde cada arquivo contém pares de chave-valor representando as traduções das sequências de caracteres (strings) utilizadas no aplicativo para um idioma específico. Durante a execução do aplicativo, o arquivo “Localizable.strings” apropriado é carregado, permitindo que o aplicativo seja exibido no idioma preferido do usuário. Essa abordagem é utilizada em todo aplicativo DynaPredict, incluindo o desenvolvimento de todas as etapas deste projeto.

Antes do projeto, o aplicativo DynaPredict já tinha um tour em um módulo diferente. Portanto, não foi necessário construir a funcionalidade do zero. No entanto, foi preciso adaptar o módulo para ser genérico e aceitar diferentes entradas de dados. Para isso, o número de páginas do tour foi convertido de um valor fixo para um valor atribuído durante a inicialização. Também, foi adicionado um parâmetro que representa o tipo de tour desejado. Esses tipos de tour foram criados usando um enumerador (enum), que é um tipo de dado que representa um conjunto fixo de valores relaciona-

dos, permitindo definir um tipo personalizado com opções. Um enum pode ser criado em Swift através da sintaxe:

```
enum Types {
    case typeOne
    case typeTwo
}
```

onde “Types” é o nome do enum e cada “case” declara uma nova opção do enum. O módulo toma como base a opção selecionada na inicialização para definir fluxos e funcionalidades como: os textos e imagens que serão apresentados, a tela para qual o usuário será direcionado após a finalização do tour e o tour que foi finalizado. Os textos e imagens de um determinado tipo de tour são dispostos em uma estrutura (struct), que é um tipo de dado utilizado para agrupar propriedades e métodos relacionados em uma única entidade. A struct é construída em Swift através da sintaxe:

```
struct SimultaneousSpectralTour {
    let screnOneText: String
    let screnOneImage: UIImage
    let screnTwoText: String
    let screnTwoImage: UIImage
//      ...

    func method() { /*...*/ }
}
```

onde “SimultaneousSpectralTour” é o nome da struct, que tem dados que se referem aos textos e imagens apresentados na tour, podendo também ter métodos específicos. Para cada tipo de tour uma struct diferente foi criada e selecionada de acordo com o tipo escolhido através do enum.

5.2.1.2 Gerenciamento da apresentação

O tour da Espectral Simultânea deve aparecer quando o usuário acessa a Espectral Simultânea, nas condições de nunca ter completado previamente esse tour. No DynaPredict, todos os dados do usuário são armazenados em um servidor remoto e em um banco de dados local. Todas as alterações realizadas no aplicativo são salvas no banco de dados local. Porém, quando o usuário sai do aplicativo (*log out*), que é diferente de simplesmente fechá-lo, os dados do banco de dados local são apagados. Por conta disso, é necessário que eventualmente ocorra a sincronização entre os dados locais e remotos, para que as informações não sejam perdidas e estejam atualizadas.

Tendo isso em vista, é necessário que a informação de que o usuário realizou o tour seja armazenada previamente no banco local e posteriormente sincronizada com o servidor.

Para garantir a persistência da informação supracitada, foi criado um novo atributo do tipo booleano no banco de dados local. O atributo simboliza se o usuário já realizou o tour, e tem o valor padrão "falso". Após o tour ser finalizado pelo usuário o valor que corresponde ao atributo é alterado para "verdadeiro". Isso é realizado por um código similar ao apresentado abaixo:

```
import RealmSwift
if let object = Realm().objects(Object.self)
    .filter("objectId = %@", id).first {
        try realm.write {
            object.parameter = true
        }
    }
}
```

de forma que, primeiramente é importada a biblioteca que diz respeito ao sistema de gerenciamento de banco de dados Realm, que é utilizado no aplicativo. Após isso, uma busca é realizada no banco de dados. No exemplo acima a busca é realizada pelo identificador (id) do objeto. Então, o parâmetro do objeto é atualizado através de uma inscrição no banco.

Para que o novo atributo pudesse ser sincronizado com o servidor, foi adicionado um campo do tipo booleano como parâmetro da mensagem utilizada para a comunicação entre o banco de dados local e o servidor. Para a validação do envio e recebimento desta mensagem foi utilizado o *software* Postman, que facilita o processo de testes rápidos nesse tipo de aplicação.

5.2.2 Fluxos de comunicação com DynaLoggers

Durante a criação do módulo foi necessária a implementação de fluxos (operações) para realizar o agendamento do monitoramento e a coleta/*download* de dados nos DynaLoggers. Cada operação é composta por uma série de comandos que são enviados para um DynaLogger. Assim que cada comando é concluído, é retornado um resultado que é utilizado para decidir o próximo comando e atualizar a visualização. Alguns resultados também são salvos no banco de dados.

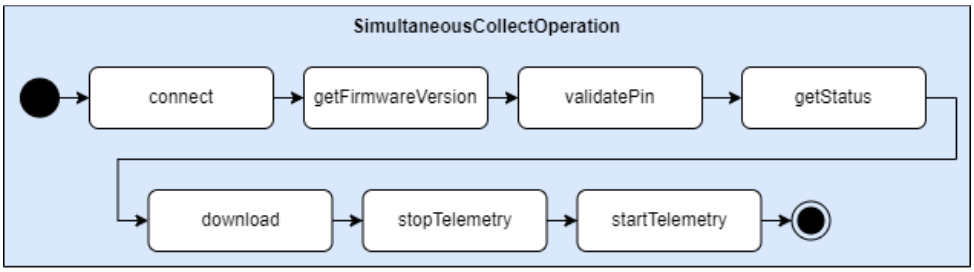
Existem duas gerações de DynaLoggers, denominadas DyP e DyL, a implementação dos comandos que interagem com os DynaLoggers são realizados de duas formas, que variam dependendo da geração:

- **Comandos para comunicação com DyP:** são utilizadas bibliotecas para iOS na realização dos comandos;
- **Comandos para comunicação com DyL:** é utilizada uma biblioteca própria da empresa, que é desenvolvida com o Kotlin Multiplatform for Mobile (KMM) e busca unificar a lógica de negócio utilizada no DynaPredict para Android e iOS.

Os comandos necessários para a construção da Espectral Simultânea já estavam previamente desenvolvidos. Entretanto, foi necessária a implementação de operações para a coleta de dados e para o agendamento do monitoramento. Visto que, cada geração de DynaLoggers utiliza métodos diferentes para execução dos comandos, cada geração precisou de operações específicas, sendo necessário o desenvolvimento de quatro diferentes operações.

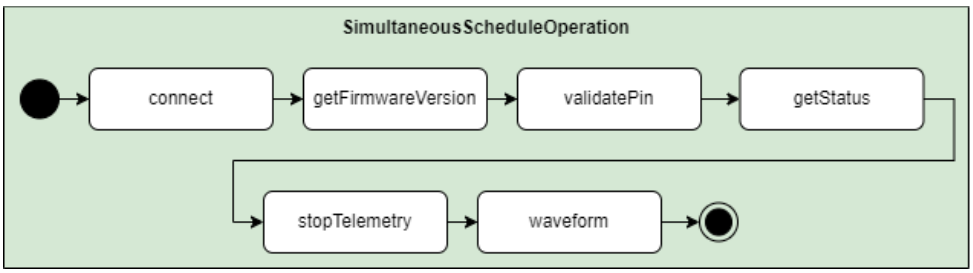
Embora tenham sido construídas operações diferentes para cada geração de DynaLogger, os comandos utilizados são similares, apenas divergindo na forma com que são implementados. Assim, as operações desenvolvidas para a coleta de dados e para o agendamento do monitoramento podem ser vistas de forma simplificada na Figura 15 e na Figura 16, respectivamente.

Figura 15 – Operação para a coleta de dados simplificada



Fonte: Arquivo pessoal.

Figura 16 – Operação para o agendamento do monitoramento simplificada



Fonte: Arquivo pessoal.

Ambas as operações iniciam com um comando responsável por estabelecer a conexão entre o aplicativo e o DynaLogger. Após isso, é obtida a versão de *firmware* do DynaLogger, com base nisso é possível determinar alguns recursos disponíveis para cada periférico. Então, é realizado um comando de autenticação, para que sejam liberados comandos mais sensíveis dos DynaLoggers (capazes de obter estado, dados de monitoramentos...). Com a finalização da autenticação, o estado do DynaLogger é requisitado, retornando informações importantes como qual o tipo de monitoramento que está armazenado no DynaLogger, memória, bateria...

Após o comando para obter o estado do DynaLogger, as operações se diferem. Na operação para a coleta de dados, é enviado o comando de “download” para baixar os dados residuais do DynaLogger. Assim que finalizada a coleta desses dados, o monitoramento contínuo é reiniciado, apagando os dados residuais. Já na operação para o agendamento do monitoramento para análise espectral, assim que o estado é adquirido, o monitoramento contínuo é interrompido e o agendamento de um monitoramento para análise espectral é realizado, o que também apaga os dados residuais do DynaLogger.

Embora existam operações diferentes para cada geração de DynaLogger, as classes que contém os métodos que executam essas operações foram abstraídas através da implementação de um protocolo genérico denominado “BLEDevice”. Sendo assim, ao fazer o uso desse protocolo, é possível invocar métodos genéricos, de forma que independam da geração do DynaLogger para realizar a operação desejada. Os retornos das operações também são genéricos e independem da geração do DynaLogger utilizado. Por fim, um “BLEDevice” pode ser obtido ao realizar a varredura de DynaLoggers.

5.2.3 Agendamento do monitoramento

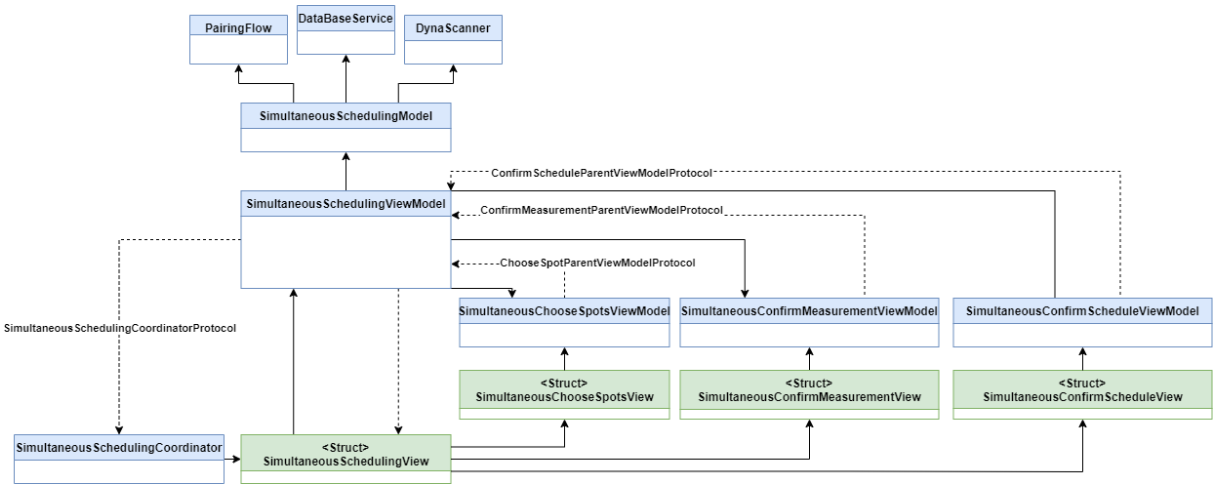
Para o agendamento do monitoramento alguns passos já explicados em seções anteriores devem ser realizados. Com isso, será apresentado aqui o desenvolvimento geral do Agendamento do Monitoramento, bem como, o desenvolvimento dos passos que compõem essa etapa.

5.2.3.1 Estrutura

Os passos do Agendamento do Monitoramento são traduzidos para o aplicativo como telas, que contêm algumas variáveis e funcionalidades específicas. Essas telas são gerenciadas por entidades que dispõem de funcionalidades que gerenciam o progresso dos passos, transitando informações entre esses passos. Essa entidade dispõe de métodos e informações que são necessárias para a realização e conclusão de cada passo.

A estruturação do agendamento do monitoramento pode ser mais facilmente entendida pelo diagrama de classes simplificado da Figura 17.

Figura 17 – Diagrama de classes simplificado do agendamento do monitoramento



Fonte: Arquivo pessoal.

No diagrama da Figura 17, as linhas cheias representam uma referência forte, em que a entidade de partida mantêm a entidade de chegada na memória. Por outro lado, as linhas tracejadas apresentam uma referência fraca, sendo possível encerrar a entidade de chegada enquanto a de partida ainda esteja em memória. Essas referências foram devidamente planejadas, uma vez que, caso haja um ciclo de referências fortes, as entidades nunca serão finalizadas, ocasionando um *memory leak*, em que as entidades ficam alocadas na memória até a finalização do aplicativo.

No diagrama é possível evidenciar as entidades de prefixo “SimultaneousScheduling”, que foram criadas para gerenciar a apresentação das entidades que representam os passos. Os passos nessa arquitetura fazem parte desse gerenciador (as entidades do gerenciador contêm as entidades dos passos), não sendo necessário haver uma navegação específica para cada passo. Por conta da não necessidade de navegação, cada um dos três passos são compostos apenas por uma *View* que está contida na “SimultaneousSchedulingView” e por uma *View Model* que está contida na “SimultaneousSchedulingViewModel”.

É possível evidenciar também o uso da arquitetura MVVM-C, apresentada na Seção 3.4.3, onde o “SimultaneousSchedulingCoordinator” é responsável pelo gerenciamento da navegação, tendo métodos responsáveis por finalizar e iniciar o módulo, instanciando as classes e estruturas do módulo durante a inicialização. A “SimultaneousSchedulingView” é responsável por apresentar a parte visual do módulo, contendo as demais estruturas de sufixo “View” dentro dela. A classe “SimultaneousSchedulingViewModel”, é responsável por conectar a lógica de negócio à visualização. Para

isso, contêm abstrações da lógica de negócio, criando fluxos que modificam as telas e tratam as interações do usuário. Ainda, por ser a *View Model* principal do módulo, também contém as demais classes com sufixo “ViewModel” e é responsável por transitar informações e gerenciar o progresso entre os passos. Por fim, a classe “SimultaneousSchedulingModel” contém a lógica de negócio. Podendo utilizar serviços para estabelecer a comunicação com os DynaLogger e acessar o banco de dados.

A comunicação entre as classes é realizada através de abstrações, que em Swift é feita por protocolos, obedecendo a DIP do SOLID, apresentado na Seção 3.5.5. Isso faz com que as classes e estruturas sejam desacopladas, ocasionando maior facilidade na manutenção do módulo. Com isso, caso uma das classes seja modificada, por decorrência de problemas ou novas funcionalidades, as demais não precisarão, desde que a classe modificada continue respeitando os métodos do protocolo. Além disso, o ISP do SOLID apresentado na Seção 3.5.4 também é respeitado, visto que cada relação entre duas entidades distintas tem seu protocolo específico, limitando a comunicação. Embora não tão evidentes pela representação do diagrama de classes, os demais princípios SOLID também foram utilizados como base para a criação dessas funcionalidades.

5.2.3.2 Gerenciamento das visualizações

Toda a etapa de construção das visualizações foi realizada com o uso da tecnologia SwiftUI, onde toda a estruturação da *View* é baseada em pilhas (*stacks*) de sub-visualizações. A *View* é atualizada de acordo com algumas variáveis predefinidas da *View Model*. Isso é realizado através de observadores, onde a *View Model* publica variáveis que são observadas pela *View*, com isso, assim que alguma variável publicada tem seu valor alterado, a visualização que observa essa variável é atualizada. Um exemplo dessa comunicação pode ser vista no código abaixo:

```
import SwiftUI

// ViewModel
class MyViewModel: ObservableObject {
    @Published var count = 0

    func incrementCount() {
        count += 1
    }
}

// View
```

```
struct MyView: View {
    @StateObject private var viewModel = MyViewModel()

    var body: some View {
        VStack {
            Text("Count: \(viewModel.count)")

            Button(action: {
                viewModel.incrementCount()
            }) {
                Text("Increment")
            }
        }
    }
}
```

onde a classe “MyViewModel” é uma classe observável, que herda de “ObservableObject”. A classe conta uma variável publicada “count”, e um método “incrementCount” que atualizada essa variável.

A estrutura “MyView” inicialmente instancia e observa a classe supracitada. A visualização é composta por um pilha vertical (“VStack”) que contém um texto que apresenta o valor da variável publicada da classe “MyViewModel” e um botão de incremento. Assim que o botão é pressionado, o método “incrementCount” é acionado, atualizando a variável “count” e acarretando a atualização do texto.

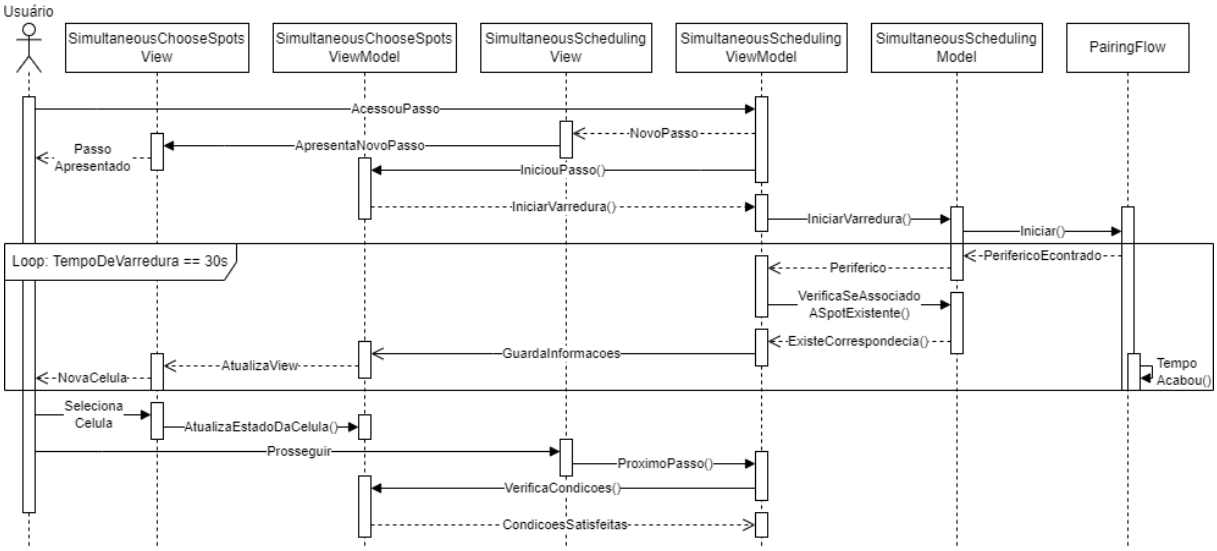
Muitas vezes é necessário o uso de serviços como banco de dados ou conexão com periféricos para atualização das visualizações. Nesse caso, é necessário que a *View Model* invoque métodos do *Model*. Um caso exemplo no projeto é durante a varredura de DynaLoggers, onde em decorrência do pressionar de um botão a *View* invoca um método da *View Model*, que por sua vez invoca um método do *Model*. O *Model* então invoca um método de um serviço, que é responsável por fazer a varredura dos periféricos. Os resultados são retornados e atribuídos a uma variável publicada da *View Model*, então a *View* é atualizada.

5.2.3.3 Funcionalidades da etapa de Escolher Spots

O funcionamento da presente etapa pode ser melhor entendido pelo diagrama de sequência apresentado na Figura 18, que apresenta de forma simplificada a comunicação e o fluxo de informações entre as entidades. Vale ressaltar que, para simplificar o diagrama, as exceções não são apresentadas. Ainda, a representação tem foco no passo de Escolher Spots, por conta disso, não são apresentadas as funcionalidades

de criação das entidades, não sendo necessária a presença do “SimultaneousSchedulingCoordinator”.

Figura 18 – Diagrama de sequência simplificado da etapa de Escolher Spots



Fonte: Arquivo pessoal.

As entidades que implementam as funcionalidades da etapa de Escolher Spots são denominadas pelo prefixo “SimultaneousChooseSpot” (veja também o diagrama da Figura 17).

A *View Model* específica do passo é responsável por armazenar uma lista de pontos de monitoramento com DynaLoggers associados, contendo informações como: o estado da célula (selecionada ou não selecionada), informações do ponto de monitoramento (nome do ponto de monitoramento e ativo em que se localiza o ponto de monitoramento) e informações relacionada ao DynaLogger (potência do sinal, identificador, modelo...). Essa lista é preenchida com base na varredura de DynaLoggers.

A varredura é invocada inicialmente pala “SimultaneousChooseSpotsViewModel”, que utiliza um método da “SimultaneousSchedulingViewModel”, que por sua vez, invoca um método do “SimultaneousSchedulingModel”, onde o serviço de varredura é utilizado para encontrar periféricos por Bluetooth. Os métodos que tratam os fluxos de varredura estão localizados na *View Model* principal pois alguns outros passos também podem realizar a varredura de periféricos, sendo assim, o “SimultaneousChooseSpotsViewModel” apenas invoca esse método.

À medida que a varredura ocorre, algumas células são criadas. Caso o usuário pressione alguma das células, ela terá seu estado alterado.

O “SimultaneousChooseSpotsViewModel” apresenta também um método para controle do progresso entre passos, que verifica a quantidade de DynaLoggers selecionados e se esses são de modelos compatíveis. Caso alguma condição não seja

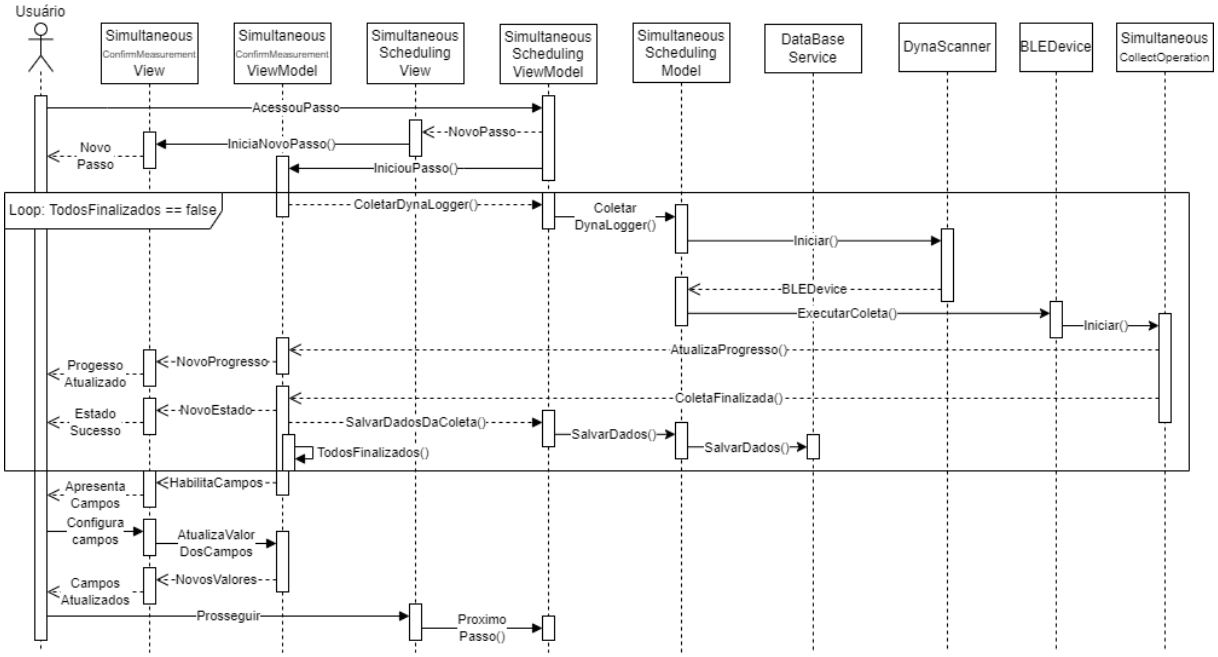
realizada essa classe aciona alertas que são apresentados na “SimultaneousChooseSpotView”.

Quando o passo é finalizado os DynaLoggers selecionados são enviados para o próximo passo, através de um método executado pela “SimultaneousSchedulingViewModel”, que passa as informações do “SimultaneousChooseSpotsViewModel” para o “SimultaneousConfirmMeasurementViewModel”.

5.2.3.4 Funcionalidades da etapa de Confirmar Medições

O passo em pode ser melhor compreendido através do diagrama da Figura 19.

Figura 19 – Diagrama de sequência simplificado da etapa de Confirmar Medições



Fonte: Arquivo pessoal.

O passo é implementado pelas entidades de prefixo “SimultaneousConfirmMeasurement” representadas na Figura 17 e Figura 19.

Com uma lista dos DynaLoggers selecionados, inicialmente o “SimultaneousCollectOperation” é acionado, sendo realizada a conexão e posterior coleta de dados residuais nesses DynaLoggers de forma sequencial. Assim que os dados de cada DynaLogger são coletados é efetuado o salvamento desses dados no banco de dados.

Toda parte comunicação com os DynaLoggers e salvamento dos dados da coleta no banco de dados ocorre em diferentes *threads* de fundo/segundo plano, para que essas tarefas sejam executadas de forma concorrente com as demais. Isso é necessário visto o processamento exigido durante a comunicação e a grande quantidade de dados que é salva no banco, que poderiam ocasionar lentidão no aplicativo caso

não fossem executadas de forma concorrente. O código utilizado para execução de tarefas em diferentes *threads* pode ser visto abaixo:

```
DispatchQueue(label: "QueueIdentification", qos: .background).async {
    // code running in background
}
```

que cria uma lista de despacho que executará um bloco de código em uma *thread* de segundo plano de forma assíncrona.

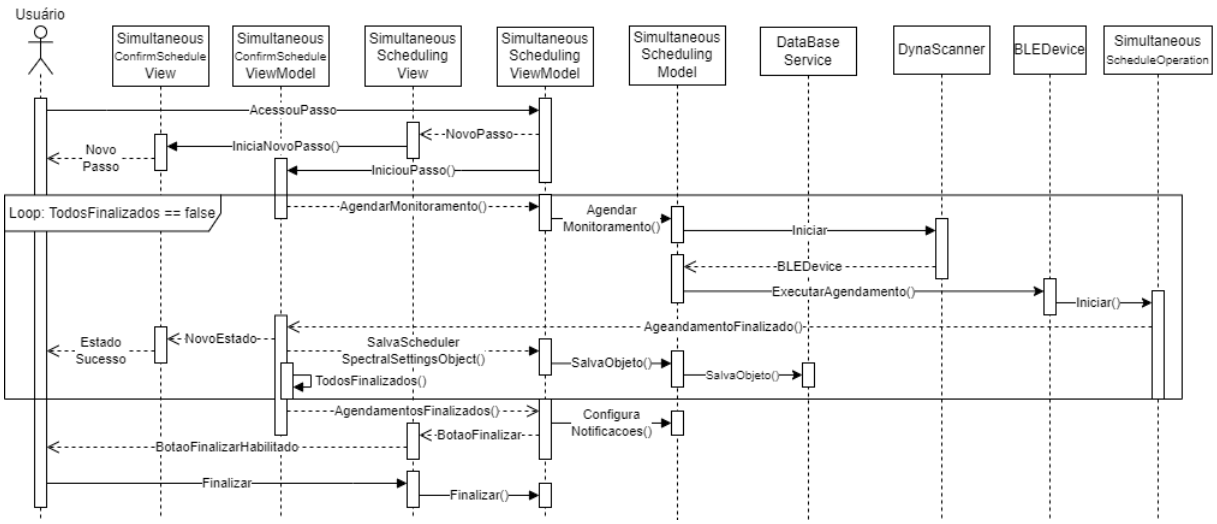
Assim que um DynaLogger obtém sucesso no processo é alocado em outra lista, que será utilizada para armazenar as informações do ponto de monitoramento e DynaLogger utilizado, bem como, as configurações do monitoramento que será realizado.

Com isso, após o processo de conexão e coleta ser finalizado, o monitoramento é configurado e salvo na lista supracitada. Essas informações são passadas do "SimultaneousConfirmMeasurementViewModel" para o "SimultaneousConfirmScheduleViewModel", essa ação é realizada pelo "SimultaneousSchedulingViewModel" assim que o usuário evolui para o passo de Confirmar Agendamento.

5.2.3.5 Funcionalidades da etapa de Confirmar Agendamento

Esse passo é implementado pelas entidades de sufixo "SimultaneousConfirmSchedule" e utiliza o "SimultaneousScheduleOperation" para agendar o monitoramento. Os fluxos deste passo podem ser melhor entendidos pelo diagrama da Figura 20.

Figura 20 – Diagrama de sequência simplificado da etapa de Confirmar Agendamento



Fonte: Arquivo pessoal.

O passo é iniciado com base nos resultados obtidos anteriormente. A partir desses dados o monitoramento é agendado fazendo o uso da operação supracitada. Assim como na etapa anterior, a *View Model* específica do passo é responsável por gerenciar o processo de agendamento do monitoramento nos DynaLogger utilizados. Quando esse monitoramento para análise espectral é agendado, o monitoramento contínuo (veja a Seção 2.1.1) é interrompido.

Assim que ocorre o agendamento, as informações referentes ao ponto de monitoramento e DynaLogger são salvas em uma tabela do banco de dados criada especificamente para essa operação, denominada “SchedulerSpectralSettingsObject”. Essas informações são persistidas localmente (não são enviadas para o servidor) e são utilizadas para verificar posteriormente quais os DynaLoggers que devem ser coletados após o fim do monitoramento, além de outros usos explicados posteriormente.

Assim que todos os agendamentos são finalizados, as notificações são configuradas e registradas. Além disso, um botão de finalizar é apresentado ao usuário, caso este botão seja pressionado, a etapa de agendamento do monitoramento da Espectral Simultânea é finalizada.

5.2.4 Notificações

Duas notificações são disparadas no aplicativo: uma após o agendamento e outra após a finalização do monitoramento. O tempo estimado para a finalização do monitoramento é calculado a partir da soma entre o tempo até o início do monitoramento, que é inserido na etapa de Confirmar Medições, com o tempo estimado para a realização do monitoramento, que é um resultado da operação de agendamento.

As notificações foram geradas a partir das ferramentas nativas do Swift, que oferece a biblioteca “UserNotifications”. Essa biblioteca contém métodos para o disparo e tratamento das notificações. Para uso da biblioteca, é necessário habilitar algumas permissões. Entretanto, no presente trabalho, isso não foi necessário, visto que o aplicativo estava previamente configurado pois já disparava algumas notificações anteriormente ao projeto.

Uma notificação genérica pode ser registrada a partir do código:

```
import UserNotifications

class Notification {
    func createNotification {
        let content = UNMutableNotificationContent()
        content.title = "Título"
        content.body = "Texto de corpo"
        let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5,
```

```
                                                    repeats: false)
        let request = UNNotificationRequest(identifier: "myNotification",
                                            content: content,
                                            trigger: trigger)

        let notificationCenter = UNUserNotificationCenter.current()
        notificationCenter.delegate = self
        notificationCenter.add(request) { (error) in
            if let error = error {
                print("Error: \(error.localizedDescription)")
            } else {
                print("Success")
            }
        }
    }
}
```

sendo que inicialmente é criado o conteúdo da notificação. Após isso, o gatilho da notificação é configurado. Nesse caso, o gatilho é configurado para disparar a notificação em cinco segundos e não repetir o disparo. Após as configurações, a requisição é criada. Então, é invocado o método “UNUserNotificationCenter.current()” responsável por retornar um objeto do tipo “UNUserNotificationCenter” que tem os métodos necessários para o gerenciamento de notificações. A esse objeto é atribuída uma classe capaz de lidar com os métodos de retorno das notificações. Isso é realizado a partir do trecho “notificationCenter.delegate = self” que atribui à própria classe que está configurando a notificação o papel de tratar esses métodos que serão delegados. Após todas as configurações, a notificação é adicionada ao gerenciador, e será disparada de acordo com as configurações.

A classe responsável por tratar os métodos que o “UNUserNotificationCenter” delega deve implementar um protocolo, como demonstrado no código abaixo:

```
extension Notification: UNUserNotificationCenterDelegate {
    func userNotificationCenter(_ center: UNUserNotificationCenter,
                                willPresent notification: UNNotification,
                                withCompletionHandler completionHandler:
                                @escaping (UNNotificationPresentationOptions)
                                -> Void) {
        completionHandler([.alert, .badge, .sound])
    }
```

```
    func userNotificationCenter(_ center: UNUserNotificationCenter,
                                didReceive response: UNNotificationResponse,
                                withCompletionHandler completionHandler:
                                @escaping () -> Void) {
        completionHandler()
    }
}
```

O protocolo indica que a classe é capaz de tratar os métodos delegados da "UNUserNotificationCenter". Nesse caso, são tratados dois métodos: o primeiro é chamado quando uma notificação é recebida enquanto o aplicativo está em primeiro plano. Já o segundo é chamado quando uma notificação é recebida enquanto o aplicativo está em segundo plano, fechado ou a partir da ação de pressionar a notificação. A lógica de criação e tratamento de notificações apresentada é similar à utilizada neste projeto.

A partir dos códigos implementados, assim que a notificação de finalização do monitoramento é disparada, o estado dos pontos de monitoramento associados aos DynaLoggers utilizados no processo é alterado para "Conexão Necessária", tendo em vista a necessidade da coleta dos dados e reativação do monitoramento contínuo.

O módulo é inacessível quando há objetos do tipo "SchedulerSpectralSettingsObject" no banco de dados e o estado dos DynaLoggers que correspondem a esses objetos não é "Conexão Necessária". Entretanto, caso haja objetos do tipo "SchedulerSpectralSettingsObject" no banco e o estado dos DynaLoggers que correspondem a esses objetos seja "Conexão Necessária" a etapa de coleta pode ser acessada.

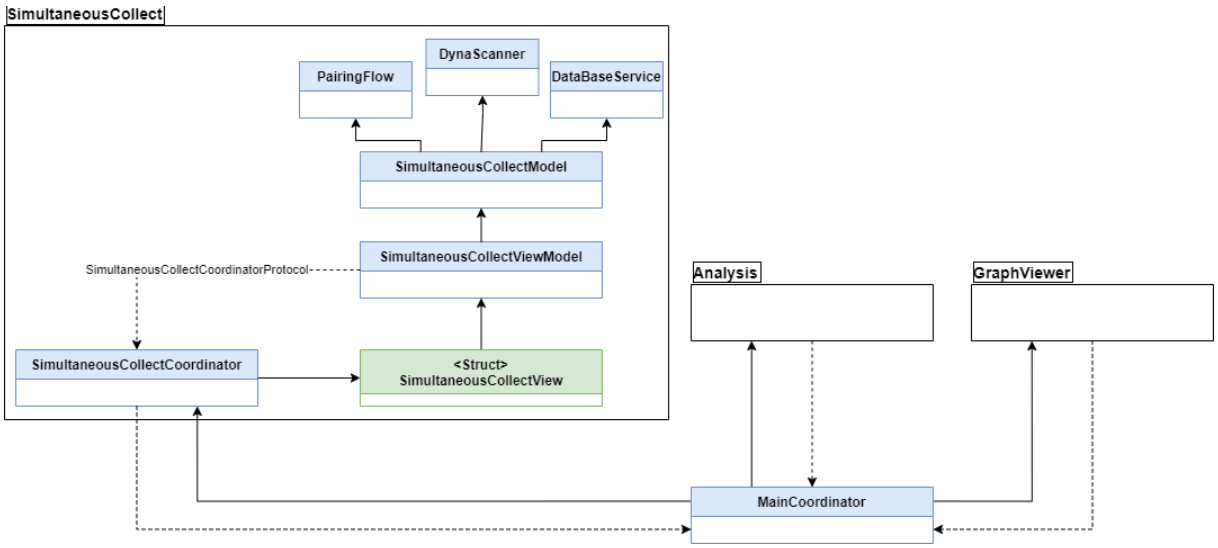
5.2.5 Coleta de dados

A estruturação da etapa de coleta de dados pode ser mais facilmente entendida ao fazer uso do diagrama da Figura 21.

É possível evidenciar pelo diagrama que a arquitetura MVVM-C é utilizada na organização das entidades do "SimultaneousCollect". Haja visto que a entidade "SimultaneousCollectCoordinator" é responsável pela criação e finalização das demais. A "SimultaneousCollectView" é responsável por apresentar as visualizações e também por captar os eventos do usuário, a "SimultaneousCollectModel" contém a lógica de negócio e a "SimultaneousCollectViewModel" faz uma ponte entre a lógica de negócio e a visualização.

Além disso, outros módulos preexistentes do aplicativo são utilizados para a funcionalidade de apresentação dos dados. Para isso, é necessário que os dados tenham sido previamente coletados e estejam formatados, sendo essas responsabilidades das entidades de prefixo "SimultaneousCollect" e dos serviços utilizados por

Figura 21 – Diagrama de classes simplificado da coleta de dados



Fonte: Arquivo pessoal.

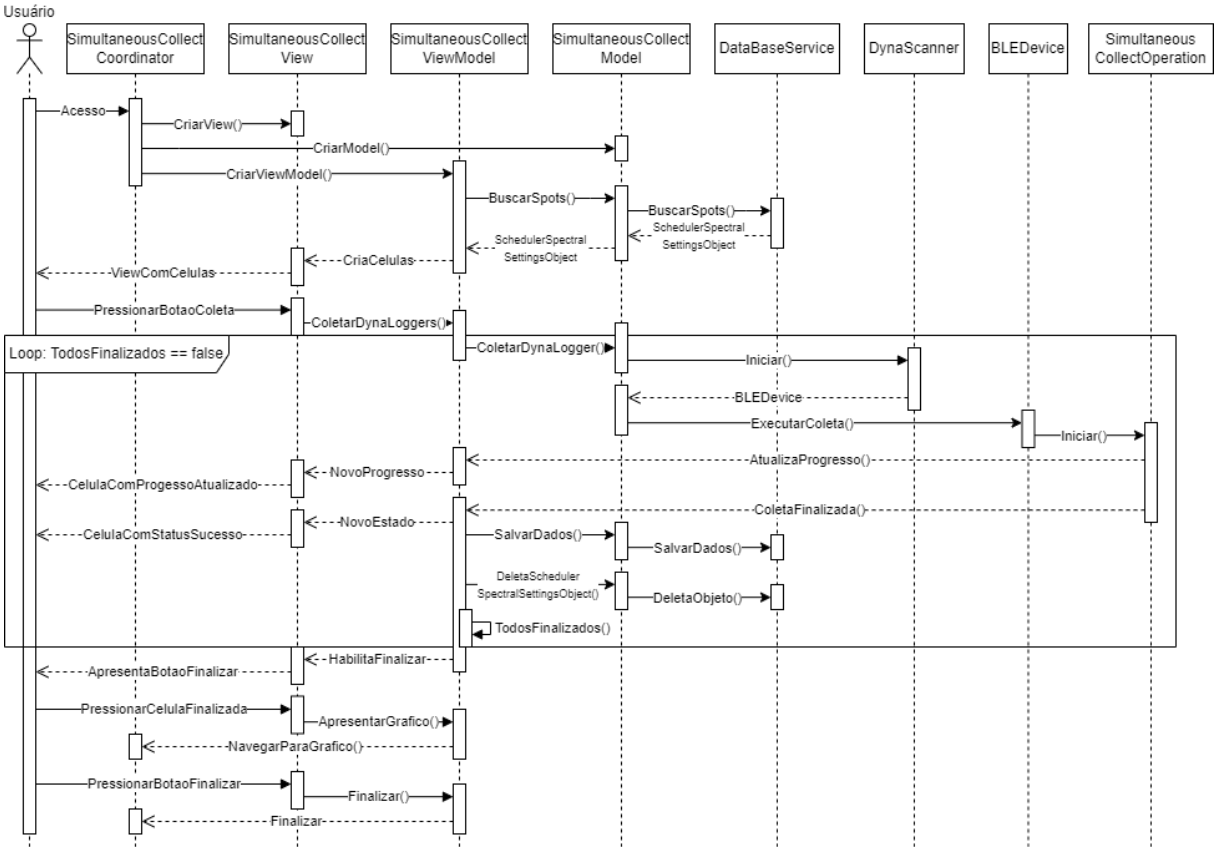
essas entidades.

O fluxo pode ser melhor entendido fazendo o uso da Figura 22. Ele é iniciado com a apresentação ao usuário dos pontos de monitoramento e DynaLoggers previamente utilizados, que são objetos do tipo “SchedulerSpectralSettingsObject” e estão salvos no banco de dados. Com essas informações é possível realizar a conexão e coleta nos DynaLoggers, tarefas que são realizadas similarmente as do passo de Confirmar Medição da etapa de Agendamento do Monitoramento (veja a Seção 5.2.3.4). Ainda, visto que a coleta ocorre de forma sequencial, assim que a coleta é finalizada em cada DynaLogger ocorre o salvamento dos dados do monitoramento no banco de dados, sendo também deletado do banco de dados o objeto “SchedulerSpectralSettingsObject” correspondente. Os dados coletados também são armazenados em uma lista de listas, para posteriormente serem apresentados graficamente sem a necessidade de acesso ao banco.

A apresentação gráfica ocorre assim que o usuário pressiona alguma das células que representam os pontos de monitoramento e DynaLoggers. Quando isso ocorre, os dados de uma das listas supracitadas são enviados como atributo para o módulo “Analysis” (veja a Figura 21), e esse módulo é acessado. O acesso do módulo ocorre através do “MainCoordinator” que é responsável por gerenciar toda a navegação do aplicativo. Caso o usuário pressione algum dos gráficos apresentado pelo módulo “Analysis”, ele é direcionado ao módulo “GraphViewer”, que apresenta o gráfico ampliado e fornece botões e funcionalidades para que usuário interaja com o gráfico apresentado.

Após a finalização da operação em todas as coletas, o botão de finalizar é

Figura 22 – Diagrama de sequência simplificado da coleta de dados



Fonte: Arquivo pessoal.

apresentado. Caso o usuário pressione esse botão a etapa é finalizada com sucesso.

5.2.6 Envio dos dados para o servidor

Visto que os dados de monitoramento foram salvos em uma classe já existente do banco de dados, a parte de sincronização deveria funcionar automaticamente. Entretanto, para garantir o envio foi utilizado o Postman. Além disso, alguns monitoramentos foram realizados com o uso de um calibrador que oscila com amplitude e frequência constantes. Esses monitoramentos, após enviados para o servidor, foram verificados no DynaPredict Web.

5.3 VALIDAÇÕES

As validações em um sistema de *software* têm como objetivo garantir que o sistema não apresente problemas e siga corretamente os requisitos especificados. Para a validação do módulo Espectral Simultânea, diferentes níveis de teste foram aplicados, sendo realizados desde testes de unidade e de integração (baixo nível), até

testes de fluxo e usabilidade (alto nível).

5.3.1 Testes de automatizados

Os testes automatizados foram desenvolvidos com o uso da linguagem Swift e o *framework* XCTest. A estrutura para desenvolvimento de uma classe de testes é iniciada com a declaração do *framework*. Após isso, a nova classe de testes implementada herda de uma classe específica do *framework* denominada "XCTestCase". A classe herdada "XCTestCase" fornece métodos e funcionalidades úteis para escrever e executar testes de forma estruturada, como:

- **XCTAssertEqual:** verifica se dois valores passados como parâmetros do método são iguais;
- **XCTAssertTrue e XCTAssertFalse:** verificam se uma condição é verdadeira ou falsa, respectivamente;
- **XCTAssertNil e XCTAssertNotNil:** verificam se um valor é nulo ou não nulo, respectivamente;
- **XCTAssertThrowsError:** verifica se uma determinada expressão lança um erro.

Como apresentado em capítulos anteriores, os teste automatizados são realizados contra um SUT, que é o sistema/pedaço de código que será testado. Para um teste efetivo, o SUT deve ser um pedaço de código do próprio sistema, que é realmente utilizado no aplicativo. Entretanto, é comum que o SUT necessite de métodos, objetos e classes auxiliares. Nesse caso, os dados e as ferramentas utilizadas pelo SUT são normalmente falsos, e simulam o comportamento real de maneira controlada. Isso possibilita que as entradas e parâmetros do sistema que está sendo testado sejam conhecidas, e por consequência, seja possível definir saídas esperadas. As ferramentas criadas para simular o comportamento real são conhecidas na área de testes de *software* como "mocks". Normalmente os testes foram realizados contra as *View Models* do módulo desenvolvido, visto que elas contêm os métodos que normalmente manipulam os dados.

5.3.1.1 Teste de unidade

A presente etapa demonstra a criação de um teste exemplo, criado com base nos conceitos citados durante o trabalho.

Inicialmente pode-se considerar um enum e uma classe "ViewModel":

```
enum Type {
    case typeA
```

```
        case typeB
    }

    class ViewModel {
        func isValidType(_ type: Type) -> Bool {
            switch type:
            case .typeA:
                return true
            case .typeB:
                return false
        }
    }
```

A classe executa um método intitulado “isValidType” que tem por objetivo retornar se o tipo escolhido é válido ou não, sendo “typeA” válido e “typeB” inválido. Esse método será a unidade testada. Com bases nisso, um exemplo de teste pode ser definido como:

```
    import XCTest

    class testClass: XCTestCase {
        func validModelTest() {
            let expectedResult = true
            let mockedType = .typeA

            let viewModelSut = createViewModelSUT()
            let result = viewModelSut.isValidType(mockedType)

            XCTAssertEqual(result, expectedResult)
        }

        private func createViewModelSUT() -> ViewModel {
            return ViewModel()
        }
    }
```

onde o método “validModelTest” executa a rotina de teste. Inicialmente é fornecido um resultado esperado (“expectedResult”). Após isso, uma entrada controlada é criada (“mockedType”). Então, o SUT é criado e o método “isValidType” executado com o dado controlado. Por fim, com o método “XCTAssertEqual” o resultado final pode ser comparado com o resultado esperado, se foram iguais, o teste passou.

O teste de unidade apresentado exemplifica os padrões de desenvolvimento dos testes utilizados no módulo Espectral Simultânea.

5.3.1.2 Teste de integração

Diferente dos testes de unidade, os testes de integração têm por objetivo validar a comunicação entre métodos e classes. Dessa forma, um exemplo pode ser criado ao considerar parte da classe e do enumerador utilizados no exemplo anterior, apenas sendo adicionado alguns elementos como apresentado abaixo:

```
protocol CoordinatorProtocol {
    func goToNextModule()
}

class Coordinator: CoordinatorProtocol {
    func goToNextModule() { /*...*/ }
}

class MockedCoordinator: CoordinatorProtocol {
    var isGoToNextModule = false
    func goToNextModule() {
        isGoToNextModule = true
    }
}

class ViewModel {
    weak var coordinator: CoordinatorProtocol?

    init(coordinator: CoordinatorProtocol) {
        self.coordinator = coordinator
    }

    func isValidType(_ type: Type) -> Bool { /*...*/ }

    func finishStep(_ type: Type) {
        if isValidType(type) {
            coordinator?.goToNextModule()
        }
    }
}
```

No código apresentado é possível ver que uma nova classe intitulada “Coodinator” foi adicionada. A classe é utilizada de forma abstrata, por meio de protocolo, pela classe “ViewModel”. Para os testes será utilizada a classe “MockedCoordinator” que oferece métodos controlados.

O método “goToNextModule” tem o objetivo real de enviar o usuário para um outro módulo. Entretanto, na classe controlada ele apenas atribui valor a uma variável que será utilizada nos testes. Além disso, a classe “ViewModel” oferece agora um método intitulado “finishStep”, que direciona o usuário a um novo módulo caso o tipo seja válido.

Um exemplo de teste de integração pode ser visto abaixo:

```
class testClass: XCTestCase {
    func finishStepTest() {
        let mockedType = .typeA

        let mockedCoordinator = MockedCoordinator()
        let viewModelSut = createViewModelSUT(mockedCoordinator)
        let result = viewModelSut.finishStep(mockedType)

        let result = mockedCoordinator.isGoToNextModule
        XCTAssertTrue(result)
    }
}

private func createViewModelSUT(_ coordinator: CoordinatorProtocol)
  -> ViewModel {
    return ViewModel(coordinator: mockedCoordinator)
}
```

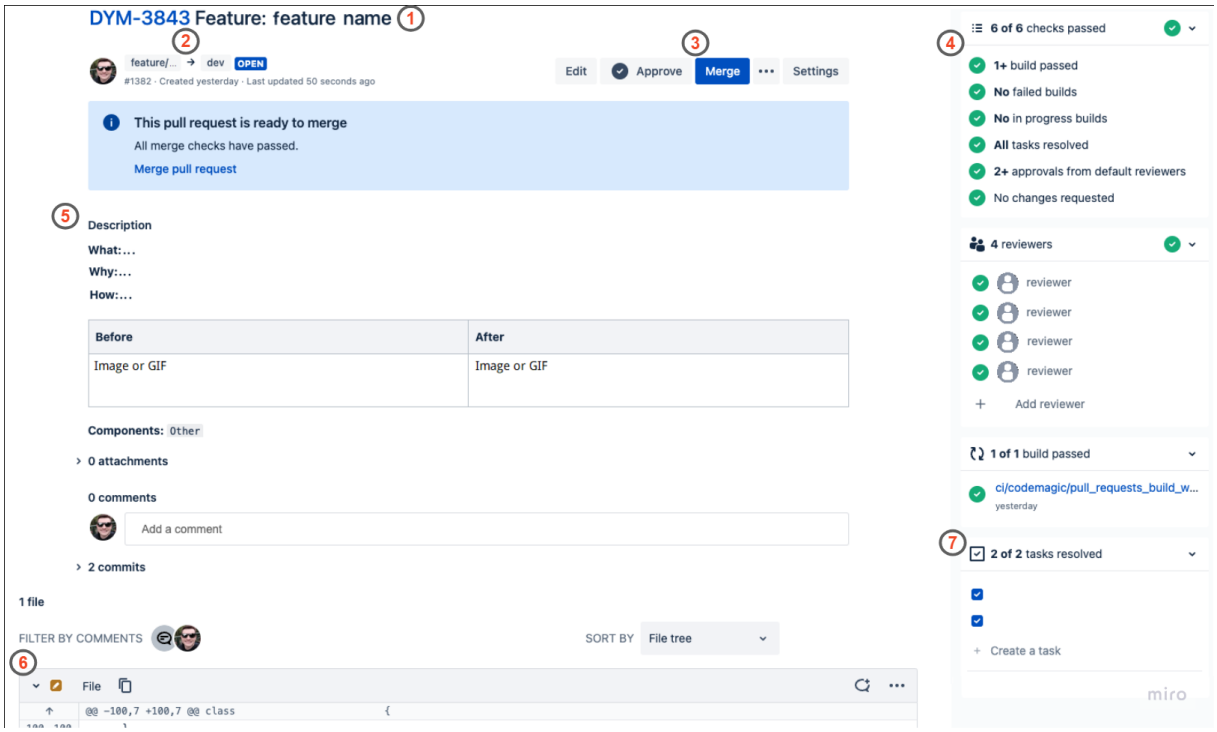
onde inicialmente um tipo válido é criado. Após isso, o Coordinator falso é criado, com o método que atribui valor “verdadeiro” à variável “isGoToNextModule”. O SUT então é criado com a classe falsa e então o método “finishStep” é acionado, tendo como entrada o outro dado controlado. De acordo com a lógica implementada, espera-se que com os dados utilizados o “finishStep” consiga direcionar o usuário para o próximo módulo (visto que o tipo utilizado é válido), o que significa atribuir valor verdadeiro a variável “isGoToNextModule”. Assim, o teste é finalizado ao verificar se o valor dessa variável é verdadeiro. O exemplo apresentado acima testa a integração entre os métodos “isValidType” e “finishStep”. Além disso, verifica a integração entre as classes utilizadas.

Por fim, os padrões utilizados são os mesmos utilizados durante o desenvolvimento dos testes de integração do presente projeto.

5.3.2 Criação de pull requests

Como explicado em capítulos anteriores, um PR é uma solicitação feita por um dos desenvolvedores para incorporar suas alterações ao código principal. No contexto deste trabalho, os PRs abertos deveriam seguir necessariamente uma padronização, que foi estruturada pelos desenvolvedores da equipe. O processo de PR utilizado pode ser entendido com o auxílio da Figura 23, onde podem ser evidenciados diversos fatores importantes como:

Figura 23 – Exemplo de *pull request*



Fonte: Arquivo pessoal.

1. **Título do PR:** que contém três principais informações: 1) o código da tarefa, que é relacionado à tarefa no quadro Kanban, facilitando a busca de informações; 2) o tipo da tarefa, que corresponde aos tipos já ressaltados na Seção 4.2; e 3) o título da tarefa realizada;

2. ***Branches*:** mostra ao revisor às *branches* a qual a solicitação de união está sendo feita;

3. **Controles:** permite ao revisor aprovar, rejeitar ou solicitar mudanças no PR. Além disso, permite que o responsável pelo PR realize a união entre as *branches*;

4. **Condições:** apresenta as condições necessárias para considerar o PR aprovado. No contexto do presente trabalho é necessário que o *build* automático[1] tenha sido bem sucedido, todas as tarefas criadas pelos revisores tenham sido resolvidas pelo responsável do PR, no mínimo dois corretores tenham aprovado o PR e não haja solicitação de mudanças no código;

5. **Descrição do PR:** o responsável do PR deve descrever de forma sucinta o que, o por que e como foram realizadas as alterações. Nessa etapa também podem ser colocadas imagens ou GIF do antes e depois da realização da tarefa, além dos componentes que sofreram as alterações;

6. **Alterações do código:** apresenta as modificações realizadas durante a execução da tarefa, para isso o código das *branches* são comparados e as diferenças sinalizadas;

7. **Tarefas do PR:** dá a possibilidade dos revisores adicionarem tarefas necessárias para considerar o PR bem sucedido. As tarefas podem ser desde mudança no código até testes que devem ser realizados pelo responsável do PR.

5.3.3 Testes realizados pelo analista de qualidade

O analista de qualidade (QA) é responsável por realizar testes sobre as alterações realizadas. Normalmente os testes realizados são de alto nível, avaliando a usabilidade e fluxo (não o código). Caso algum problema seja identificado, o QA reabre a tarefa problemática. O desenvolvedor deve realizar as alterações necessárias para corrigir o problema identificado, sendo necessário refazer o fluxo de trabalho, com abertura de nova *branch*, PR, *merge* e novos testes. A tarefa será reaberta quantas vezes forem necessárias.

[1] O *build* automático é uma ferramenta que tem por objetivo validar se o aplicativo é construído corretamente pela IDE, para isso, ele testa todos os ambientes alvos do aplicativo. Além disso, o *build* automático também executa os testes automatizados do aplicativo.

6 RESULTADOS

No presente capítulo são descritos os resultados obtidos a partir da proposta de desenvolvimento de um módulo para agendamento e coleta de dados de monitoramento simultâneo para análises espectrais.

6.1 FUNCIONALIDADE IMPLEMENTADAS

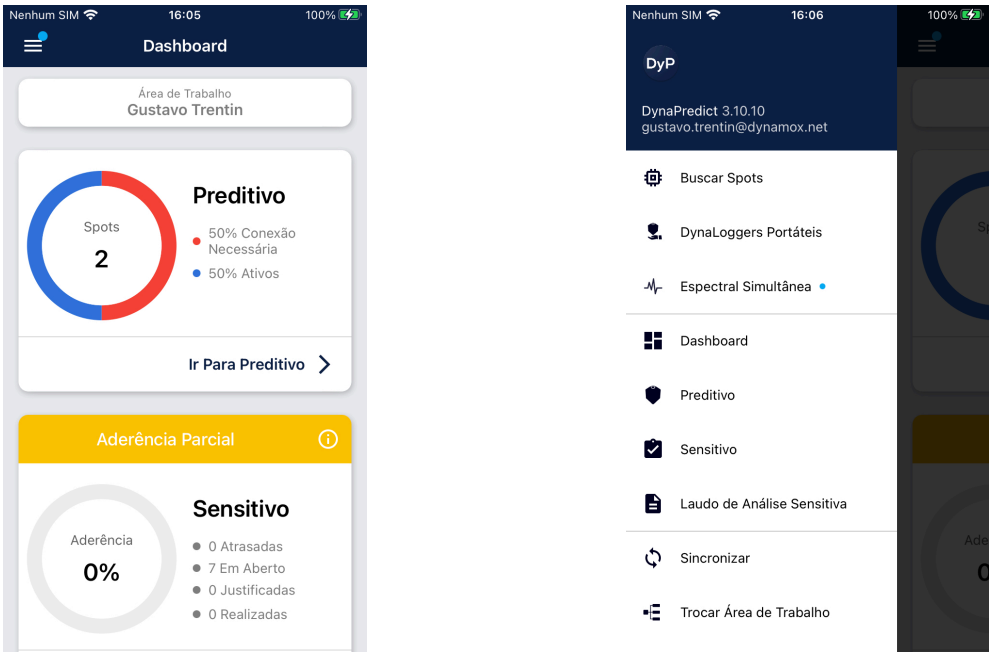
São apresentados na presente seção os resultados obtidos relacionados às funcionalidades e fluxos do módulo, sendo apresentados também os resultados visuais (telas) obtidos com a realização do projeto.

6.1.1 Tour da Espectral Simultânea

Na primeira vez que o usuário acessa o aplicativo, ou enquanto não acessar a Espectral Simultânea, um marcador aparecerá tanto no botão de acesso ao menu (Figura 24a) como no botão de acesso ao novo módulo (Figura 24b). Nessas condições, se o usuário acessa a novo módulo, é imediatamente direcionado para um tour da Espectral Simultânea.

Como visto na Figura 25 e Figura 26 o tour da Espectral Simultânea é composto por cinco etapas. Que explicam ao usuário o propósito da Espectral Simultânea e como usar esse módulo.

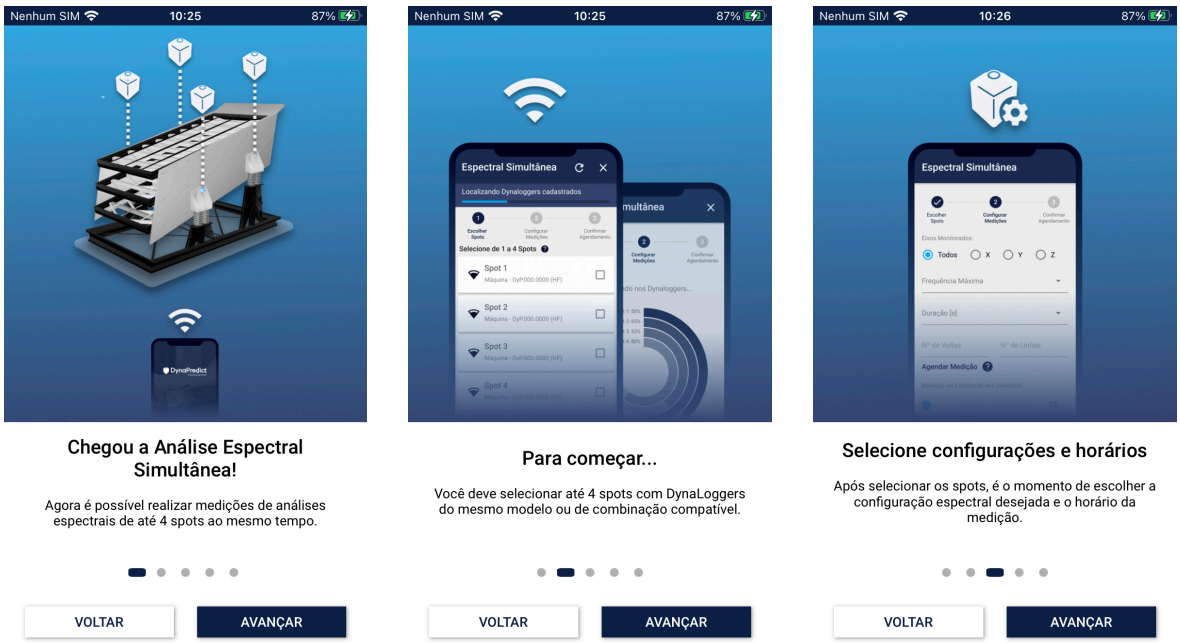
Figura 24 – Marcador de tour disponível



(a) Acesso ao menu. (b) Acesso ao módulo

Fonte: Arquivo pessoal.

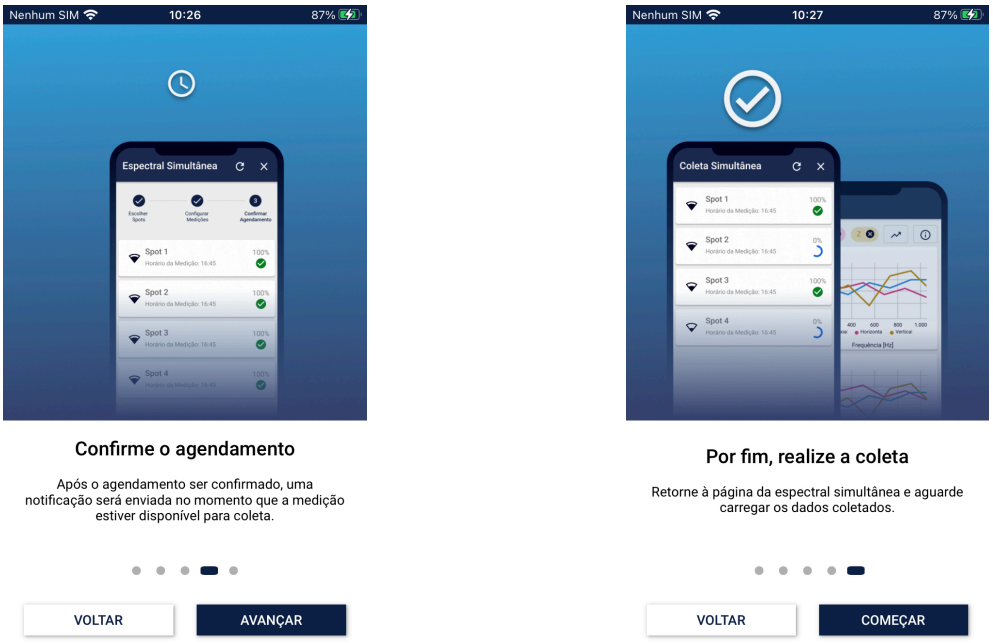
Figura 25 – Primeira, segunda e terceira telas do tour da Espectral Simultâneas.



(a) Primeira tela (b) Segunda tela (c) Terceira Tela

Fonte: Arquivo pessoal.

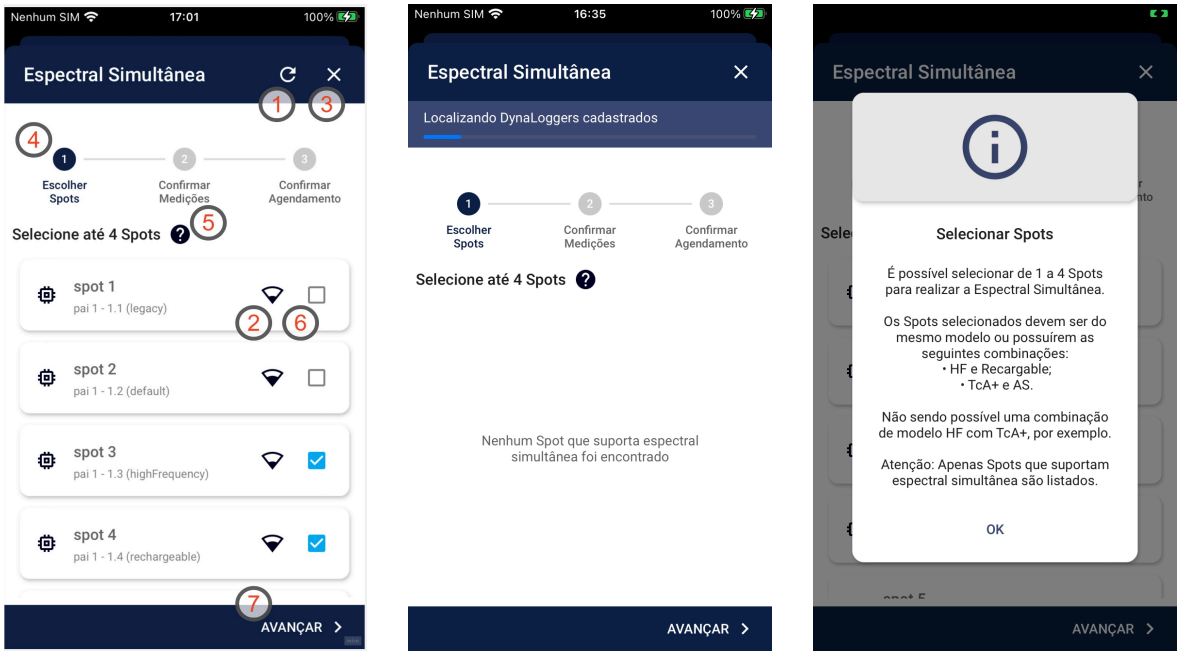
Figura 26 – Quarta e quinta telas do tour da Espectral Simultâneas.



(a) Quarta tela (b) Quinta tela

Fonte: Arquivo pessoal.

Figura 27 – Seleção dos pontos de monitoramento



(a) Funcionalidades. (b) Varredura (c) Alerta informativo

Fonte: Arquivo pessoal.

6.1.2 Agendamento do monitoramento

Assim que o usuário acessa a espectral simultânea nas condições de já haver realizado o tour e não ter monitoramentos simultâneos que ainda não foram coletados, ele é direcionado às funcionalidades de agendamento do monitoramento. As funcionalidades são compostas por três diferentes passos já explicados anteriormente. Os resultados para as funcionalidades que correspondem ao agendamento do monitoramento podem ser vistos separadamente nesta subseção.

6.1.2.1 Escolher Spots

As primeiras funcionalidades previstas da etapa de agendamento dizem respeito à seleção dos pontos de monitoramento Figura 27.

É possível identificar através da Figura 27a as funcionalidades previstas para o agendamento, que são:

1. Botão de varredura: quando pressionado, uma varredura de DynaLoggers é acionada (Figura 27b). A varredura identifica os DynaLogger próximos ao usuário através do uso de Bluetooth, com base nos resultados dessa varredura novas células são criadas (caso o DynaLogger não tenha sido anteriormente localizado) ou ocorre a atualização do RSSI do DynaLogger (Item 2) nas células já existentes. A varredura também é acionada assim que o usuário acessa o presente passo.

O botão de varredura é utilizado em demais lugares do módulo, realizando ações similares;

2. RSSI do DynaLogger: visto que cada ponto de monitoramento tem um DynaLogger associado, esse item apresenta ao usuário o RSSI do DynaLogger associado ao ponto de monitoramento correspondente à célula;

3. Botão de fechar: tem como objetivo fechar o módulo. O presente botão pode ser visto nas demais telas que compõem as etapas necessárias para o agendamento do monitoramento;

4. Indicativo de etapa: apresenta ao usuário a etapa atual. O presente item também é visto nas próximas telas do módulo;

5. Botão de informação: assim que pressionado é apresentado ao usuário o alerta da Figura 27c;

6. Indicativo de seleção: assim que a célula é pressionada o indicativo tem seu estado alterado, informando ao usuário se o ponto de monitoramento está selecionado;

7. Botão de avanço: tem o objetivo finalizar a presente etapa e direcionar o usuário para a etapa posterior.

   Para que a etapa de Escolher spots possa ser finalizada o usuário deve ter selecionado no mínimo um e no máximo quatro pontos de monitoramento com DynaLoggers associados, sendo todos de modelos compatíveis, ou seja, que suportem monitoramentos com mesma frequência e duração. Caso alguma das condições supracitadas não seja realizada um dos alertas da Figura 28 é apresentado. O botão de avanço também é visto em outras etapas do módulo, porém, com condições diferentes para avançar entre as etapas.

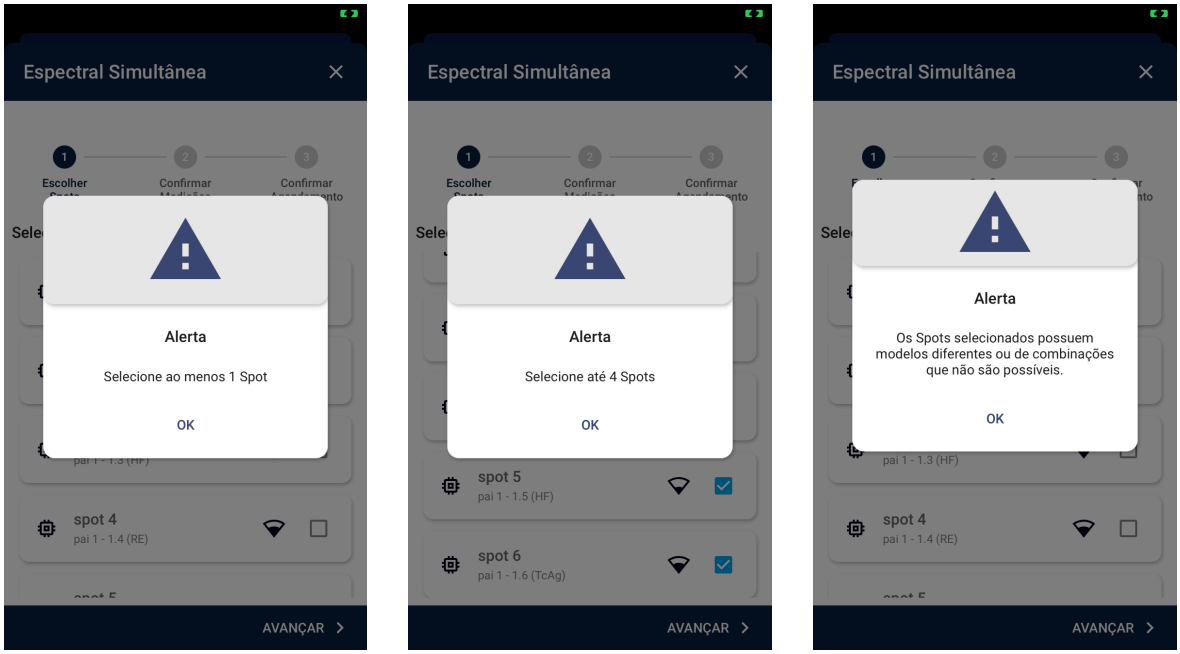
Após a finalização da etapa de Escolher Spots o usuário é direcionado para a etapa de Confirmar Medições.

6.1.2.2 Confirmar Medições

Após a etapa anterior há uma tentativa de conexão para baixar possíveis dados residuais dos DynaLoggers pré-selecionados.

Com base nos resultados dessa etapa, diferentes fluxos são considerados, e a tela sofre variações que dependem do sucesso da operação nos DynaLoggers. Inicialmente é apresentada a tela da Figura 29a, que apresenta o progresso da operação em cada DynaLogger. Nela é possível evidenciar falha na operação em um dos DynaLoggers (cor amarela), isso acarretará posteriormente no aparecimento do alerta

Figura 28 – Alertas de condições não realizadas



(a) Nenhum selecionado. (b) Muitos selecionados (c) Modelos incompatíveis

Fonte: Arquivo pessoal.

apresentado na Figura 29c. Já a tela da Figura 29b é apresentada em caso de falha na operação em todos os DynaLoggesr. Caso todos consigam realizar corretamente a operação, os campos de configuração do monitoramento (Figura 30a) são apresentados.

Novas funcionalidades são apresentadas ao usuário durante a conexão e *download* (Figura 29), como:

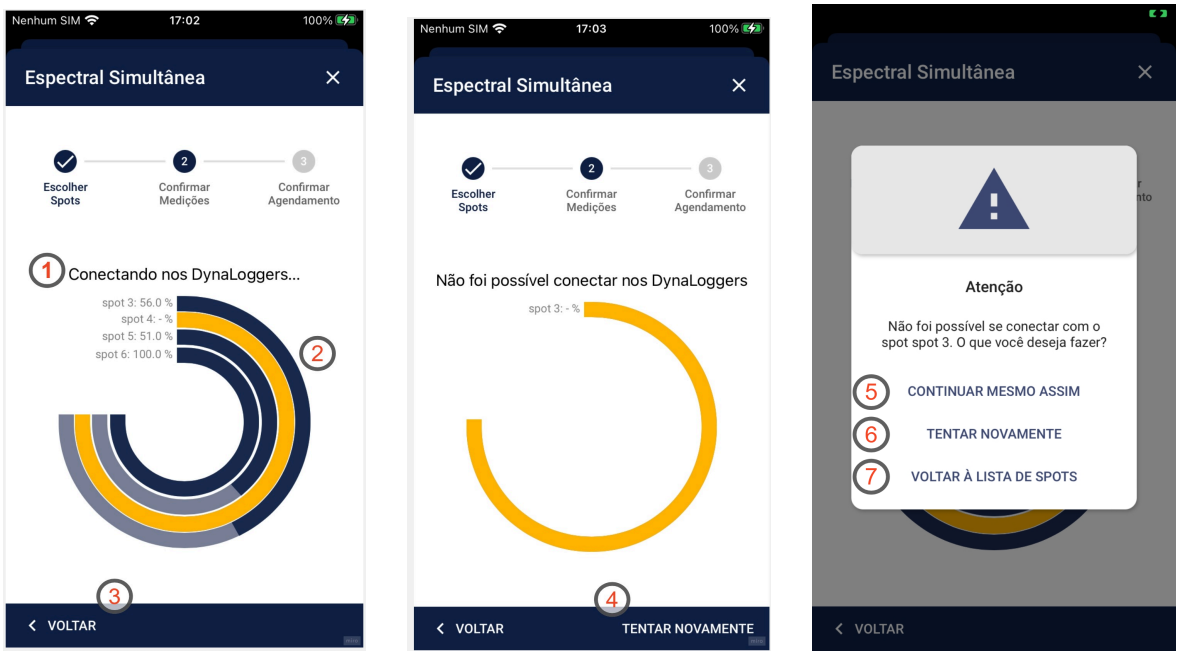
1. Texto de estado da operação: texto informativo sobre estado da operação, que pode ser diferente durante a execução da operação (Figura 29a), e caso haja falha na operação em todos os DynaLoggers (Figura 29b);

2. Indicativo de progresso: indica o progresso das operações de conexão e *download* de dados para cada DynaLogger que corresponde a um ponto de monitoramento. Caso haja falha durante o processo em algum DynaLogger, a barra de progresso correspondente assume a cor amarela e o indicador numérico de progresso assume o estado inválido (“-”);

3. Botão de voltar: permite ao usuário voltar à etapa anterior.

4. Botão de tentar novamente: caso haja falha durante o processo em todos os DynaLoggers o botão é apresentado. Se pressionado, a etapa de conexão e reiniciada e o botão desaparece;

Figura 29 – Telas de conexão e *download* de dados residuais



(a) Progresso da coleta. (b) Falha nas coletas. (c) Alerta de falha.

Fonte: Arquivo pessoal.

5. Botão de continuar mesmo assim do alerta: caso pressionado, os DynaLoggers que obtiveram falha na conexão são descartados do processo de agendamento;

6. Botão de tentar novamente do alerta: visto que o alerta aparece apenas quando houve falha em algum DynaLogger e sucesso em outros, se o botão for pressionado, o processo de conexão e *download* é reiniciado nos DynaLoggers que falharam;

7. Botão de voltar do alerta: se pressionado, retorna à etapa anterior (ação igual ao botão de voltar do Item 3).

Após o sucesso na operação em todos os DynaLoggers ou após o acionamento do “Botão de continuar mesmo assim do alerta” os campos de configuração do monitoramento (Figura 30a) são automaticamente habilitados. Nesse contexto, pode-se destacar algumas novas funcionalidades:

1. Seleção de eixo: com esse campo é possível selecionar os eixos que serão monitorados;

2. Frequência máxima: permite selecionar a frequência máxima do monitoramento. Na prática o valor enviado para o DynaLogger é o dobro do apresentado ao leitor, para evitar o efeito Aliasing. A lista de frequências máximas disponíveis

(Figura 30b) depende do modelo dos DynaLoggers selecionados (Seção 6.1.2.1) e dos eixos (Item 1);

3. Duração: permite ao usuário selecionar a duração do monitoramento. Assim como no item anterior, os possíveis valores são predefinidos (lista de seleção), porém, baseados no modelo (Seção 6.1.2.1), eixo (Item 1) e frequência máxima (Item 2);

4. Número de linhas: é um campo desabilitado para edição, que é calculado a partir do produto entre frequência máxima (Item 2) e duração (Item 3);

5. Botão de informação: apresenta um alerta similar aos demais já apresentados, contendo informações sobre a funcionalidade de agendar a medição;

6. Controle deslizante: permite alterar o tempo para o início do monitoramento, sendo permitidos valores entre dez e sessenta minutos. A edição afeta o valor apresentado no Item 7;

7. Campo numérico: assim como no item anterior, o campo numérico (Figura 30c) permite alterar o tempo para o início do monitoramento. O campo é limitado ao mesmo intervalo de valores do item anterior. Se um valor vazio ou menor que o limite inferior é inserido, o valor considerado é o do limite inferior. Se um valor maior que o limite superior é inserido, é considerado o valor máximo. A edição do campo afeta o progresso do Item 6.

Por fim, visto que todos os campos têm valores padrões atribuídos, não há condições que impeçam o usuário de avançar para a próxima etapa, sendo necessário apenas pressionar o botão de avanço.

6.1.2.3 Confirmar Agendamento

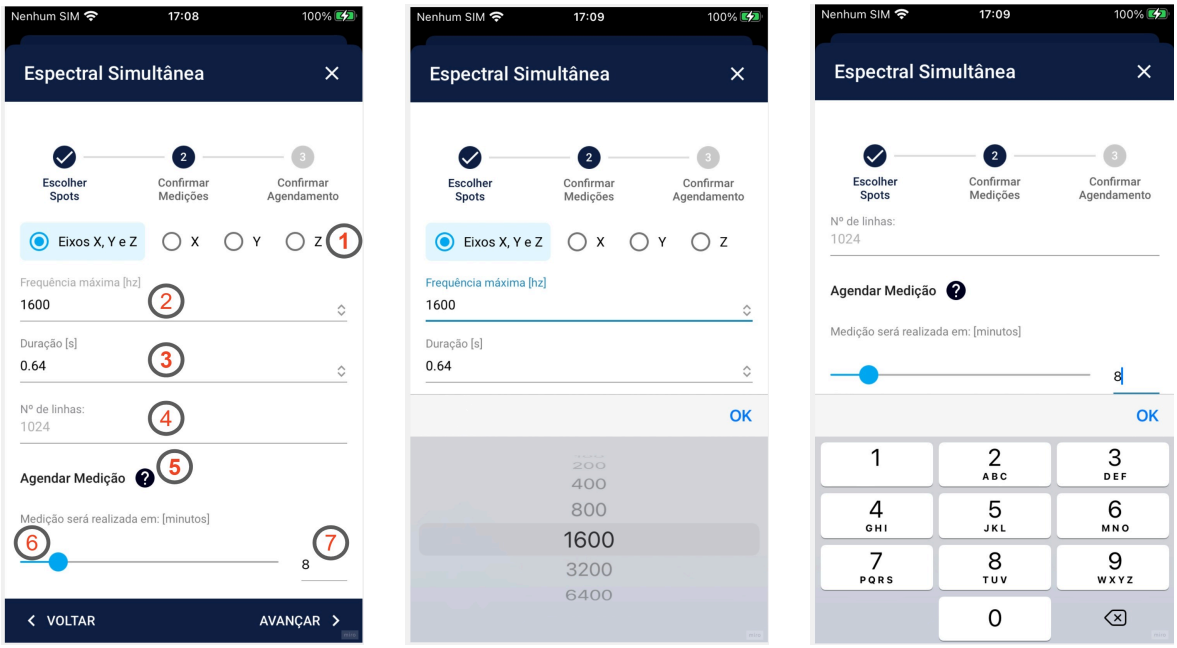
Com as configurações necessárias obtidas o agendamento do monitoramento é configurado em cada DynaLogger, sendo esse o principal objetivo da presente etapa. Além de confirmar o agendamento, a interface oferece retorno de informações ao usuário quanto o estado do agendamento (Figura 31a).

Alguns erros também são previstos nessa etapa, um deles ocorre quando uma ou mais tentativas de agendamento falham, acarretando na aparição do alerta da Figura 31b. Outro erro possível é obtido se o agendamento não é realizado antes da data prevista para o início do monitoramento, nesse caso é acionado o alerta da Figura 31c.

Tendo isso em vista, algumas novas funcionalidades podem ser citadas, como:

1. Indicador de estado: utilizado para dar um retorno visual ao usuário sobre o estado do agendamento. O primeiro estado apresentado é o de enviando dados

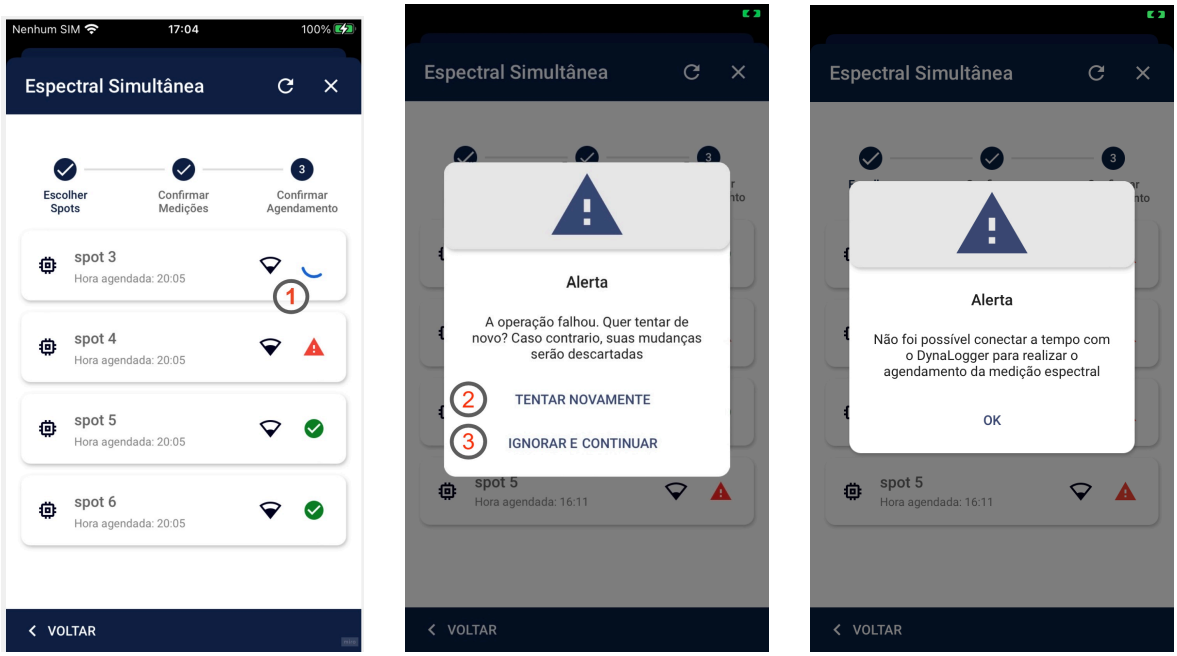
Figura 30 – Configuração do monitoramento



(a) Campos (b) Lista de seleção (c) Teclado numérico

Fonte: Arquivo pessoal.

Figura 31 – Confirmação do agendamento



(a) Agendamento (b) Alerta de falha (c) Alerta de *time out*

Fonte: Arquivo pessoal.

para o agendamento, representado pelo ícone de carregamento. Caso haja falha ou sucesso no agendamento, os demais ícones que aparecem na Figura 31a são apresentados;

2. Botão de tentar novamente do alerta: igualmente aos demais botões de tentar novamente, caso acionado, o sistema tenta realizar novamente a operação nos casos de falha;

3. Botão de ignorar e continuar do alerta: esse botão apenas aparece se houve sucesso em algum agendamento e falha em outros, seguindo a mesma lógica de negócio aplicada ao alerta da Figura 29c. Caso pressionado, o sistema desconsidera os DynaLoggers que obtiveram falha no agendamento e o fluxo é continuado.

Por fim, assim que há o retorno de sucesso na operação, as informações de DynaLogger e ponto de monitoramento são armazenadas no banco de dados, para posteriormente serem coletadas. Caso haja sucesso em todos os casos ou o Item 3 seja pressionado, há a aparição do botão de finalizar, que se localiza na mesma posição da funcionalidade número sete da Figura 27a. Quando o botão de finalizar é pressionado o agendamento é considerado como concluído e a próxima etapa é considerada.

6.1.3 Notificações no dispositivo

As notificações implementadas para a Espectral Simultânea são disparadas assim que o agendamento é realizado (Figura 32a) e quando o monitoramento é finalizado (Figura 32b). Durante esse intervalo de tempo (agendamento realizado e monitoramento finalizado) o módulo fica indisponível. Caso o usuário tente acessar o módulo durante esse período, o alerta da Figura 32c é apresentado.

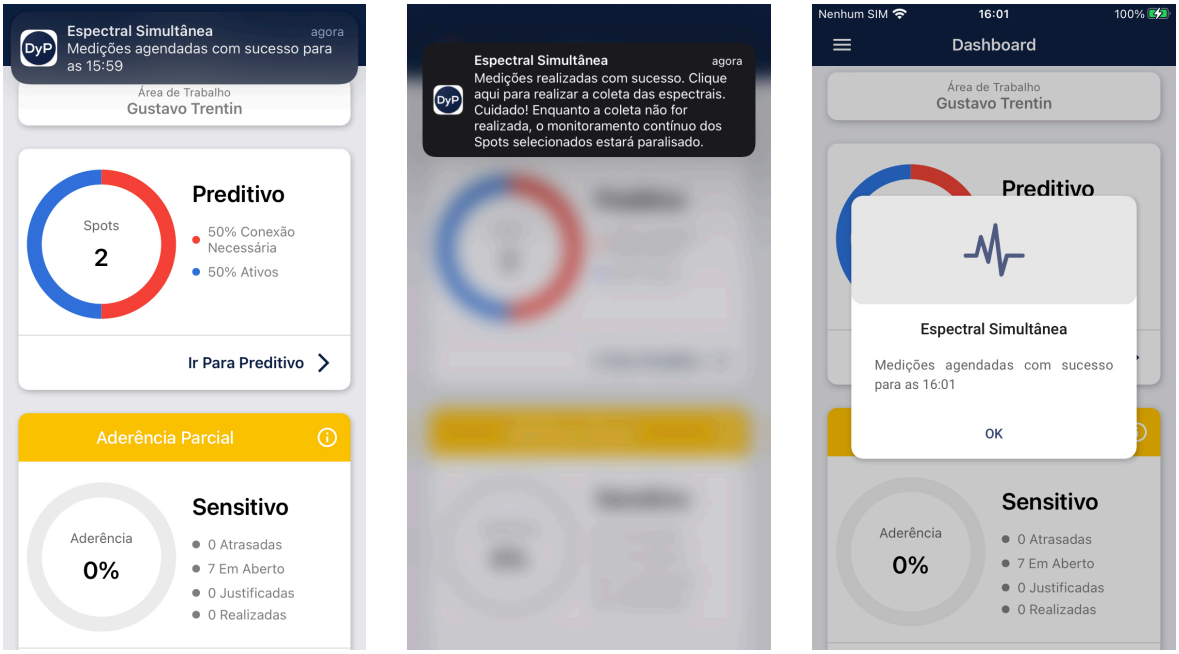
Assim que a notificação de monitoramento concluído (Figura 32b) é apresentada, o botão de acesso ao menu e o botão de acesso ao módulo ficam marcados com um ponto vermelho, similarmente aos marcadores do tour (veja respectivamente a Figura 24a e Figura 24b). Há também uma ação relacionada ao toque dessa notificação, que faz com que o usuário seja direcionado à etapa subsequente.

As notificações supracitadas aparecem mesmo que o aplicativo esteja em segundo plano ou fechado, mantendo suas funcionalidades.

6.1.4 Coleta de dados

Assim que o monitoramento é finalizado, a presente etapa pode ser acessada. Inicialmente o usuário é direcionado à tela de coleta dos dados (Figura 33a) onde é capaz de verificar o estado da operação. Caso a coleta seja finalizada com sucesso

Figura 32 – Notificações de agendamento e alerta de módulo indisponível.



(a) Medições agendadas (b) Medições realizadas (c) Módulo indisponível

Fonte: Arquivo pessoal.

em algum dos DynaLoggers, o usuário pode selecionar a célula que o representa, e assim visualizar os dados coletados de forma gráfica (Figura 33b). Para análise mais aprofundada dos dados coletados, o usuário pode ainda pressionar algum dos gráficos apresentados na Figura 33b. Isso irá direcionar o usuário para uma tela com gráfico expandido . O gráfico expandido e a visualização horizontal disponibilizam uma experiência visual mais imersiva, além de alguns campos para manipulação do gráfico serem habilitados.

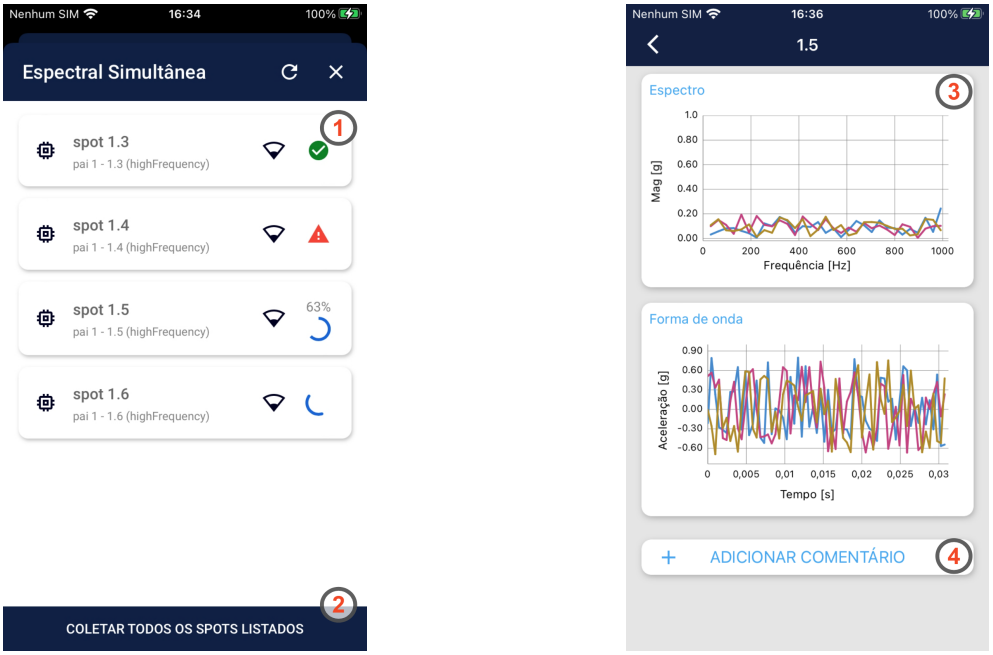
Ao verificar a Figura 33 e Figura 34 pode-se verificar alguns itens relevantes:

1. Indicador de estado: o item tem o objetivo de fornecer um retorno visual sobre o estado da coleta de dados. O item também adiciona um novo elemento de progresso à célula (veja a terceira célula da Figura 33a), que é mostrado durante a coleta de dados;

2. Botão de controle: o presente controle permite ao usuário iniciar, parar e finalizar a etapa. Dependendo do estado da operação, o texto e a cor do botão são alterados;

3. Imagem do gráfico: permite ao usuário realizar uma pré visualização do gráfico desejado, caso o usuário pressione alguma das imagens, ele é direcionado para à tela da Figura 34;

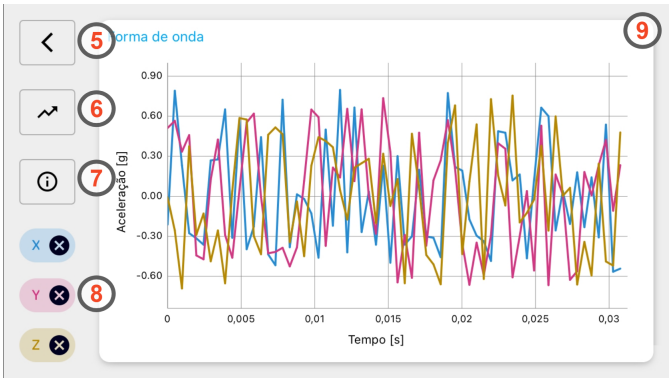
Figura 33 – Confirmação do agendamento e resultados gráficos.



(a) Estado da coleta (b) Gráficos

Fonte: Arquivo pessoal.

Figura 34 – Gráfico expandido.



Fonte: Arquivo pessoal.

4. Botão de adicionar comentários: permite ao usuário adicionar comentários que serão enviados junto aos dados do monitoramento;
5. Botão de voltar: quando pressionado o usuário é direcionado a tela da Figura 33b;
6. Botão de alterar métrica: quando pressionado possibilita ao usuários alterar os dados entre dados de velocidade e aceleração;
7. Botão de informação: quando pressionado apresenta ao usuário um alerta contendo as configurações do monitoramento em que esses dados foram obtidos;

8. Botões de eixos: ao pressioná-los, o usuário altera o estado do eixo para selecionado ou não, permitindo ao usuário filtrar apenas os eixos de interesse que serão apresentados no gráfico;

9. Gráfico interativo: permite ao usuário dar *zoom* e selecionar pontos de interesse do gráfico.

6.1.5 Envio de dados

A partir de testes realizados foi evidenciado que os dados de monitoramento estavam sendo enviados corretamente para o servidor.

6.2 VALIDAÇÕES

A partir das validações, puderam-se obter resultados que auxiliam a compreender se o *software* desenvolvido atende aos requisitos e expectativas definidos para seu funcionamento. Tendo isso em vista essa etapa apresenta ao leitor os resultados obtidos através das validações realizadas nesse trabalho.

6.2.1 Testes automatizados

Foram desenvolvidos mais de vinte testes automatizados de *software*, considerando os testes de unidade e de integração, que testaram aspectos como condições para o progresso entre telas, exatidão no preenchimento de variáveis, e exatidão na execução e retorno de métodos. Todas as funcionalidades desenvolvidas foram aprovadas nos testes antes de serem unidas ao código principal.

6.2.2 Pull requests

Durante essa etapa, o *build* automatizado falhou apenas uma vez, alertando o autor sobre problemas nos testes automatizados do aplicativo. Os desenvolvedores requisitaram algumas mudanças e testes adicionais através da criação de *tasks* no PR. Essas *tasks* diziam respeito a:

- Validar com o *designer* o marcador de tour e marcador de monitoramento pronto para a coleta;
- Nome errado nos passos durante o desenvolvimento das telas;
- Usar lógica já existente no aplicativo para preenchimento dos valores padrões dos campos de configuração do monitoramento;

Os código citados foram resolvidos durante o processo de PR. Nenhum dos corretores solicitou grandes mudanças no código que ocasionassem o declínio do PR.

6.2.3 Testes realizados pelo analista de qualidade

Os testes realizados pelo analista de qualidade (QA) da equipe evitaram que algumas inconformidades fosse esquecidas. Dessa forma, os testes realizados ocasionaram a reabertura de algumas tarefas, por conta de erros como:

- **Elemento visual da etapa de Confirmar Medições:** na primeira vez em que a tarefa de criar os campos de configuração do monitoramento, descrita na Seção 4.2.5, foi finalizada, algumas inconformidades foram destacadas. O componente de alteração de tempo não estava padronizado com o componente utilizado no aplicativo e o campo de inserir numericamente o tempo para o início do monitoramento estava bloqueado para edição, afetando respectivamente os itens 6 e 7 da Figura 30a. Tais inconformidades causaram a reabertura da tarefa;
- **Erro de fluxo durante a etapa de Confirmar Agendamento:** após a entrega da tarefa de agendamento do monitoramento, descrita na Seção 4.2.6, ser finalizada, foram encontradas inconformidades no fluxo. A opção de ignorar e continuar do alerta apresentado na Figura 31b estava sendo apresentada mesmo se todos os casos de agendamento falhassem. Essa inconformidade também ocasionou a reabertura da tarefa.

7 CONCLUSÃO

O módulo de Espectral Simultânea desenvolvido conseguiu satisfazer os requisitos propostos para o trabalho, permitindo que fossem realizados monitoramentos em vários DynaLoggers de forma simultânea, o que possibilita a análise de diferentes tipos de falhas que até então não eram possíveis ou eram mais difíceis de serem detectadas.

Ao analisar a construção do módulo pôde ser evidenciado o uso de diversas técnicas de programação e estruturação, como a arquitetura MVVM-C, os princípios SOLID, dentre outras boas práticas de programação. Com isso, espera-se uma boa manutenibilidade, com facilidade na resolução de possíveis falhas ou adição de novas funcionalidades no módulo.

Os testes realizados garantiram a exatidão das funcionalidades e fluxos desenvolvidos, evitando que algumas falhas chegassem ao usuário final, trazendo confiabilidade ao processo.

Embora os objetivos tenham sido atingidos, algumas melhorias poderiam ser realizadas no projeto, como a coleta simultânea em mais pontos de monitoramento, o que proporcionaria mais dados para a avaliação. Além disso, a comunicação com o sensores poderia ocorrer de forma concorrente, para que o processo fosse mais rápido.

Por fim, espera-se que o desenvolvimento do módulo traga maior eficiência no processo de manutenção, o que causa impacto positivo no âmbito econômico e ecológico, ampliando o espaço do Brasil no que diz respeito à área de tecnologias para a realização de manutenção preditiva.

REFERÊNCIAS

APPLE. **Apple Website**. [*S.l.*: *s.n.*], 2023. https://www.apple.com/br/swift/. [Online; acessado em 24 de Maio de 2023].

BERNARDO, Paulo Cheque; KON, Fabio. A importância dos testes automatizados. **Engenharia de Software Magazine**, v. 1, n. 3, p. 54–57, 2008.

CHACON, Scott; STRAUB, Ben. **Pro git**. [*S.l.*]: Springer Nature, 2014.

COCCO, Luisanna; MANNARO, Katiuscia; CONCAS, Giulio; MARCHESI, Michele. Simulating kanban and scrum vs. waterfall with system dynamics. *In*: SPRINGER. AGILE Processes in Software Engineering and Extreme Programming: 12th International Conference, XP 2011, Madrid, Spain, May 10-13, 2011. Proceedings 12. [*S.l.*: *s.n.*], 2011. P. 117–131.

DIRECT INDUSTRY. **Direct Industry Website**. [*S.l.*: *s.n.*], 2023. https://www.directindustry.com/prod/metso-corporation/product-9344-1663891.html. [Online; acessado em 01 de Junho de 2023].

DYNAMOX. **DynaLogger HF+**. [*S.l.*: *s.n.*], 2023a. https://content.dynamox.net/wp-content/uploads/2023/02/DataSheet-HF-082023-00-PT.pdf. [Online; acessado em 25 de Maio de 2023].

DYNAMOX. **Dynamox Website**. [*S.l.*: *s.n.*], 2023b. https://dynamox.net/. [Online; acessado em 12 de abril de 2023].

FARINELLI, Fernanda. Conceitos básicos de programação orientada a objetos. **Instituto Federal Sudeste de Minas Gerais**, 2007.

FEATHERS, Michael. **Working Effectively With Legacy Code: Work Effect Leg Code _p1**. [*S.l.*]: Prentice Hall Professional, 2004.

FLAVIO ANTUNES. **Git Workflow: o que é e principais tipos**. [*S.l.*: *s.n.*], 2021. https://www.zup.com.br/blog/git-workflow. [Online; acessado em 24 de Maio de 2023].

HUDSON, P. **Pro Swift - Swift 4. 1 Edition**. [*S.l.*]: CreateSpace Independent Publishing Platform, 2018. ISBN 9781985779785. Disponível em: https://books.google.com.br/books?id=OpibtAEACAAJ.

LIU, Chu-sheng; ZHANG, Shi-min; ZHOU, Hai-pei; LI, Jun; XIA, Yun-fei; PENG, Li-ping; WANG, Hong. Dynamic analysis and simulation of four-axis forced synchronizing banana vibrating screen of variable linear trajectory. **Journal of Central South University**, Springer, v. 19, n. 6, p. 1530–1536, 2012.

MACÊDO, Leticia Costa. Manutenção preditiva no contexto da indústria 4.0: um modelo preditivo em uma fábrica do ramo metalúrgico. Vitória, 2020.

MARTIN, R.C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. [*S.l.*]: Prentice Hall, 2017. (Robert C. Martin series). ISBN 9780134494272. Disponível em: https://books.google.com.br/books?id=fwvetAEACAAJ.

MARTIN, Robert C. Design principles and design patterns. **Object Mentor**, v. 1, n. 34, p. 597, 2000.

OSHEROVE, Roy. **The Art of Unit Testing: With Examples in. Net**. [*S.l.*]: Manning, 2009.

PROSZAK, Susann. **MVVM-C A simple way to navigate**. [*S.l.*: *s.n.*], 2016. https://tech.trivago.com/post/2016-08-26-mvvmc/. [Online; acessado em 18 de abril de 2023].

RICARTE, Ivan Luiz Marques. Programação Orientada a Objetos: uma abordagem com Java. **http://www. dca. fee. unicamp. br/cursos/PooJava/Aulas/poojava. pdf> Acesso em**, v. 29, n. 10, p. 2014, 2001.

SANTOS NETO, Manoel Ferreira; LEITE, Denisson Santana; NASCIMENTO, Willem Vieira. Revisão bibliográfica da manutenção preditiva e seus conceitos de tecnologia atrelados a Indústria 4.0. **Anais do X SIMPROD**, Departamento de Engenharia de Produção-Universidade Federal de Sergipe, 2018.

ŠARAVANJA, Kristian. **Chat aplikacija za iOS platformu**. 2019. Tese (Doutorado) – Josip Juraj Strossmayer University of Osijek. Faculty of Electrical . . .

SUSTO, Gian Antonio; BEGHI, Alessandro; DE LUCA, Cristina. A predictive maintenance system for epitaxy processes based on filtering and prediction techniques. **IEEE Transactions on Semiconductor Manufacturing**, IEEE, v. 25, n. 4, p. 638–649, 2012.

SUTHERLAND, Jeff. **SCRUM: A arte de fazer o dobro de trabalho na metade do tempo**. [*S.l.*]: Leya, 2014.

THELMA, Ugonna. **The S.O.L.I.D Principles in Pictures**. [*S.l.*: *s.n.*], 2020. `https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e898`. [Online; acessado em 16 de abril de 2023].

WAZLAWICK, Raul. **Análise e Projeto de Sistemas de Informação Orientados**. [*S.l.*]: Elsevier Brasil, 2010. v. 2.