INSTITUTO FEDERAL DE SANTA CATARINA

ELISA KALIL ELIAS

AUTOMATIZAÇÃO DA ETAPA DE DIMENSIONAMENTO DE CARGA PARA NOVA LIGAÇÃO JUNTO À CELESC

Itajaí
2024

ELISA KALIL ELIAS

AUTOMATIZAÇÃO DA ETAPA DE DIMENSIONAMENTO DE CARGA PARA NOVA LIGAÇÃO JUNTO À CELESC

Monografia apresentada ao curso de Engenharia Elétrica do campus Itajaí do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro Eletricista.

Orientadora: Ana Elisa Ferreira Schmidt

Coorientador: Marcelo dos Santos Coutinho

Itajaí
2024

Ficha de Identificação da Obra
Sistema de Bibliotecas Integradas do IFSC - Campus Itajaí

Elias, Elisa Kalil
E42a Automatização da etapa de dimensionamento de carga para nova ligação junto à Celesc / Elisa Kalil Elias ; orientadora, Ana Elisa Ferreira Schmidt ; coorientador, Marcelo dos Santos Coutinho. -- 2024.
111 f.

Trabalho de Conclusão de Curso (Graduação) - Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, Campus Itajaí, Graduação em Engenharia Elétrica, Itajaí, 2024.
Inclui bibliografia.

1. Engenharia Elétrica. 2. Carga e distribuição elétrica. 3. Aplicativos móveis. I. Schmidt, Ana Elisa Ferreira. II. Coutinho, Marcelo dos Santos. III. Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina – Graduação em Engenharia Elétrica. IV. Título.

CDD 621.31

Elaborada pela Bibliotecária Eliane Pellegrini – CRB-14/1218

ELISA KALIL ELIAS

AUTOMATIZAÇÃO DA ETAPA DE DIMENSIONAMENTO DE CARGA PARA NOVA LIGAÇÃO JUNTO À CELESC

Monografia apresentada ao curso de Engenharia Elétrica do campus Itajaí do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro Eletricista.

Trabalho aprovado, Itajaí - SC, 04 de Junho de 2024.

BANCA EXAMINADORA:

gov.br Documento assinado digitalmente
FERNANDA ISABEL MARQUES ARGOUD DA SILV.
Data: 21/08/2024 11:09:55-0300
Verifique em https://validar.iti.gov.br

_____

Profa. Dra. Fernanda Isabel Marques Argoud Da Silv
Instituto Federal de Santa Catarina

gov.br Documento assinado digitalmente
SAIMON MIRANDA FAGUNDES
Data: 22/08/2024 13:48:38 -0300
Verifique em https://validar.iti.gov.br

_____

Prof. Dr. Saimon Miranda Fagundes
Instituto Federal de Santa Catarina

gov.br Documento assinado digitalmente
ANA ELISA FERREIRA SCHMIDT
Data: 19/08/2024 17:44:33-0300
Verifique em https://validar.iti.gov.br

Profa. Dra. Ana Elisa Ferreira Schmidt
Orientadora

AGRADECIMENTOS

Dedico este trabalho primeiramente aos meus pais, Patricia e Wilson pela forma como me apoiaram na conclusão de mais uma etapa da minha formação.

Aos meus irmãos que sempre me apoiaram nessa longa jornada.

Ao Instituto Federal de Ciência e Tecnologia onde tenho orgulho de ter estudado.

Aos bons amigos e colegas de trabalho que tive a oportunidade de conhecer graças às incontáveis oportunidades que a universidade me deu, e em especial, aos professores, que formaram não só mais uma cientista, mas um ser humano apto a construir um mundo melhor.

Muito especialmente, desejo agradecer a minha orientadora Professora Ana Elisa Ferreira Schmidt, pela sua orientação, disponibilidade e atenção dispensadas.

A programação é como resolver um quebra-cabeça onde você cria suas próprias peças, em vez de encaixá-las.

(Limor Fried, 2016)

RESUMO

Uma estratégia para minimizar equívocos na estimativa de dimensionamento de carga residencial e otimizar a distribuição de energia pela concessionária que consiste em automatizar um documento que estima o dimensionamento de carga requisitado pela Celesc, empresa fornecedora de energia, durante o procedimento de nova ligação.

Por meio do desenvolvimento de um aplicativo iOS, busca-se que a estimativa de dimensionamento de carga residencial seja gerado de forma automatizada mediante a escolha de equipamentos pelo usuário, tendo como objetivo validar a seguinte hipótese: é viável que um aplicativo seja concebido e implementado para aprimorar e automatizar as etapas para usuários leigos ao solicitarem uma nova ligação junto à concessionária de energia elétrica.

Os objetivos principais abarcam a comprovação da viabilidade do projeto, a apresentação de uma visão geral sobre aplicativos móveis para iOS, a condução de um estudo comparativo através de testes de usabilidade, modelos arquiteturais e a reestruturação do aplicativo de acordo com a arquitetura *Model View ViewModel* (MVVM).

Os resultados destacam uma melhoria substancial no dimensionamento de carga e uma resposta positiva à facilidade de uso do aplicativo. A contribuição original reside na criação de uma ferramenta específica para usuários leigos na estimativa de dimensionamento de carga, preenchendo uma lacuna na automatização desse processo.

Conclui-se que o desenvolvimento de aplicativos para automatizar o dimensionamento de carga é viável e vantajoso, sobretudo para usuários não versados em conceitos elétricos, contribuindo para a eficiência e exatidão do processo. As implicações práticas englobam uma simplificação considerável do procedimento para as concessionárias de energia elétrica, com a redução de equívocos e o potencial aumento da eficiência no sistema de distribuição da Celesc. A pesquisa se alinha a estudos prévios ao reconhecer a importância da automação no dimensionamento de carga, destacando-se por concentrar-se na experiência do usuário leigo e oferecer uma solução prática e acessível para esse público.

Palavras-chave: Mobile, iOS, Usabilidade

ABSTRACT

A strategy to minimize mistakes in load lifting, carried out predominantly by individuals who are not familiar with electrical concepts, and optimize energy distribution by the concessionaire consists of automating load sizing, requested by Celesc during the new connection procedure with the energy supplying company .

This project, through the development of an iOS application, seeks to generate residential load sizing through the user's choice of equipment, aiming to validate the following hypothesis: it is feasible to design and implement an application dedicated to improving and automating the steps for lay users when requesting a new connection from the electricity company?

The main objectives include proving the viability of the project, presenting an overview of mobile applications for iOS, conducting a comparative study through usability tests, architectural models and restructuring the application according to the Model View View architecture -Model (MVVM). The research adopts an applied approach, using the hypothetical-deductive method.

The findings highlight a substantial improvement in load sizing accuracy, reduced misunderstandings by lay users, and a positive response to the application's ease of use. The original contribution lies in the creation of a specific tool for lay users in load sizing, filling a gap in the automation of this process.

It is concluded that the development of applications to automate load sizing is viable and advantageous, especially for users not versed in electrical concepts, contributing to the efficiency and accuracy of the process. The practical implications include a considerable simplification of the procedure for electricity concessionaires, with the reduction of mistakes and the potential increase in efficiency in new connections. The challenge faced was to simplify complex electrical concepts for lay users. The research aligns with previous studies by recognizing the importance of automation in load sizing, standing out for focusing on the lay user experience and offering a practical solution for this audience.

Keywords: Mobile, iOS, Usability.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ABNT - Associação Brasileira de Normas Técnicas
APP - Application
CEB - Companhia Energética de Brasília
CELESC - Centrais Elétricas de Santa Catarina
HIG - Human Interface Guidelines
IDE - Integrated Development Environment
iOS - iPhone Operating System
MVC - Model View Controller
MVVM - Model View View-Model
OOP - Object Oriented Programming
PEP - Projeto Elétrico de Particulares
TCC - Trabalho de Conclusão de Curso
UC - Unidade Consumidora
UI - User Interface
UX - User Experience
VIP - View Interactor Presenter
WPF - Windows Presentation Foundation

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho de conclusão de curso insere-se no âmbito da obtenção do grau de bacharelado em Engenharia Elétrica no Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina sob a orientação da Professora Ana Elisa Ferreira Schmidt, e a coorientação do Professor Marcelo dos Santos Coutinho.

Neste capítulo serão enunciados a contextualização, justificativa e objetivos do trabalho.

1.1 Contextualização

A primeira cidade brasileira a receber luz elétrica nas ruas foi Campos dos Goytacazes localizada no interior do Rio de Janeiro, em 1883, tornando-se também a pioneira na inauguração da iluminação pública em toda a América Latina (QLUZ, 2021). Como resultado desse avanço, ocorreram as primeiras eletrocussões e incêndios relacionados a instalações elétricas no final do século XIX, evidenciando a necessidade técnica de normatização para tornar mais segura essa prática (SILVA, 2015).

A normatização pode ser definida como a atividade destinada a estabelecer, face a problemas reais ou potenciais, disposições para a utilização comum e repetida, tendo em vista a obtenção do grau ótimo de ordem, de acordo com Almacinha (2018). Por isso, segundo Silva (2015), o dimensionamento correto das instalações elétricas é fundamental para garantir a segurança e o funcionamento adequado, sendo sempre importante dimensionar a carga desejada corretamente, considerando margens de segurança, desde que estes considerem os casos mais desfavoráveis das variáveis desconhecidas.

Para imóveis sem medidor ou nunca habitados anteriormente em Santa Catarina, é necessário fazer a requisição de uma nova ligação junto às Centrais Elétricas de Santa Catarina (Celesc). Para a aprovação do requerimento, a Celesc solicita diversos documentos, dentre eles: especificação do tipo de ligação (monofásica, bifásica ou trifásica), a relação de equipamentos, e o dimensionamento de carga. A Celesc utiliza limites de potência em kilowatts (kW) para determinar o tipo de ligação mais adequado ao consumidor. Por exemplo, até 15 kW é classificado como uma ligação monofásica, enquanto que até 25 kW é considerado bifásico, conforme indicado na Tabela 02 do Manual Simplificado - Padrão de Entrada de Energia Elétrica em Instalações Consumidoras (Celesc, 2023). Já acima de 25 kW a ligação será trifásica. Portanto, para fazer a escolha correta, o usuário precisaria ter conhecimentos específicos sobre instalações elétricas. Além disso, qualquer usuário, independente do seu conhecimento na área, precisaria pesquisar os valores de potência de cada equipamento que possui e somar esses valores. Em seguida, ele terá que verificar em qual faixa de kilowatts (15 kW, 25 kW, ou mais) a ligação se encaixa, a fim de determinar o tipo de ligação mais apropriado junto à concessionária.

Através da experiência profissional como desenvolvedora de interface móvel, a autora deste trabalho pode perceber como aplicativos otimizam tempo e processos do dia-a-dia. Ainda que no site da Celesc seja possível encontrar uma planilha que sirva de exemplo para o usuário de como formular um dimensionamento de carga da sua residência, este pode encontrar dificuldades em especificar valores de potência (confundindo com valores de corrente e tensão mostrados na etiqueta dos equipamentos), bem como em especificar quais aparelhos devem ser listados ou qual tipo de ligação deve ser solicitada.

Buscando apoiar o usuário na tarefa de dimensionamento de cargas de sua residência, a autora desenvolveu o protótipo de um aplicativo móvel, apresentado como projeto final na disciplina de Projeto Integrador III (PI III), lecionada no 9° semestre do curso de Engenharia Elétrica do Instituto Federal de Educação, Ciência e Tecnologia do câmpus Itajaí. O objetivo da disciplina é instigar o aluno a desenvolver um projeto que integre diferentes disciplinas cursadas até o momento. O protótipo desenvolvido em PI III, teve o intuito de otimizar e automatizar a etapa de relação de equipamentos e dimensionamento de carga e tipo de ligação para usuários leigos em relação à eletricidade. Os alunos presentes participaram da apresentação relatando que, em alguns casos, os clientes de empresas onde adquiriram experiência profissional enfrentaram recusas em suas solicitações pois não foi anexado documento de responsabilidade técnica por profissional habilitado via sistema Projeto Elétrico de Particulares (PEP), documento este exigido pela Celesc para os casos em que o dimensionamento da carga ultrapassa 65 kW. Contudo, o usuário não estava ciente, e portanto não pode atender à exigência da concessionária.

O protótipo referido neste relatório como versão inicial do *App* de dimensionamento de carga, foi desenvolvido usando linguagem Swift, nativa iOS (funcional apenas para iPhone) e consiste em gerar de forma automatizada a relação de cada carga (em *watts*), bem como a soma total (em *kilowatts*) do imóvel através apenas dos equipamentos selecionados previamente pelo usuário.

O protótipo foi bem sucedido, sendo o *App* capaz de realizar as funções definidas. Tendo em vista o sucesso do protótipo, este trabalho de conclusão de curso tem como objetivo dar continuidade ao desenvolvimento do aplicativo, otimizando-o e adicionando a ele novas funcionalidades, além da realização de testes de usabilidade com profissionais com experiência na área, a fim de validar, entre outras coisas, sua eficácia.

Pretende-se com o desenvolvimento deste Trabalho de Conclusão de Curso (TCC) responder a seguinte hipótese: É viável projetar e desenvolver uma aplicação mobile para iOS nativo dedicado à otimizar e automatizar a etapa de relação de equipamentos, dimensionamento de carga e tipo de ligação para usuário leigos no que diz respeito à eletricidade, para fins de levantamento de carga para a solicitação de nova ligação residencial junto à concessionária de energia elétrica?

1.2 Justificativa

Aplicações móveis desempenham um papel cada vez mais crucial em diversas áreas, incluindo a gestão consciente de energia. No contexto de cálculo de demanda energética, esses aplicativos são uma ferramenta fundamental para auxiliar empresas e consumidores a compreenderem e gerenciarem seu consumo de forma responsável (SILVA; GUIMARÃES; LIMA; GUIMARÃES, 2020, p. 35).

Um aplicativo de dimensionamento de consumo energético se torna ainda mais relevante para pessoas sem conhecimento técnico, especialmente ao tentar solicitar uma nova ligação junto à Celesc. Muitas vezes, as pessoas desconhecem a quantidade de energia necessária para suas residências, resultando em conexões inadequadas e problemas como interrupções no fornecimento e contas mais elevadas.

Com um aplicativo de dimensionamento de consumo energético, seria mais intuitivo para as pessoas determinarem a quantidade correta de energia necessária em suas residências, garantindo uma conexão adequada. Isso não só evita que suas solicitações sejam negadas ou problemas relacionados ao mau dimensionamento de energia, mas também possibilita economia a longo prazo, já que um dimensionamento inadequado pode levar a contas de energia mais altas. Através de uma interface intuitiva e amigável, essa ferramenta capacita qualquer pessoa a tomar decisões baseadas em seu consumo de energia. Ao incorporar essas soluções tecnológicas, promove-se a preservação dos recursos naturais fomentando o uso responsável da energia, resultando em um impacto positivo tanto para o meio ambiente quanto para nossa economia.

Logo, este trabalho se mostra relevante uma vez que, através da criação e desenvolvimento de um aplicativo móvel, pretende-se amenizar os problemas discutidos. As funcionalidades que envolvem dimensionamento da carga total, especificação correta de todas as cargas e a orientação quanto à apresentação de documento de responsabilidade técnica, reduzem a quantidade de erros no requerimento, aumentando a segurança e precisão do fornecimento de energia.

Por fim, considera-se que o trabalho está de acordo com os valores do IFSC, em especial referente à inovação tecnológica e ao desenvolvimento de produtos e processos que possam ser transferidos para empresas do arranjo local, assim como para à comunidade local.

1.3 Objetivo

1.3.1 Gerais

Comprovar a viabilidade do projeto e desenvolvimento de um aplicativo iOS, cujas funcionalidades permitam automatizar e otimizar as etapas de relação de equipamentos, dimensionamento de carga, e tipo de ligação, para imóveis residenciais por usuários leigos no que diz respeito à eletricidade.

1.3.2 Específicos

No que diz respeito aos objetivos específicos, pretende-se:

1. Apresentar uma visão geral sobre aplicativos móveis para dispositivos iOS, assim como sobre os modelos arquiteturais mais empregados no seu desenvolvimento;
2. Desenvolver a primeira versão do *app* seguindo a arquitetura *Model View Controller* (MVC), de forma a otimizar este;
3. Realizar um estudo comparativo entre modelos arquiteturais para o desenvolvimento de aplicativos iOS;
4. Reestruturar classes, parâmetros e métodos seguindo a arquitetura *Model View View-Model* (MVVM);
5. Avaliar impactos arquitetônicos causados pela substituição do modelo arquitetural no aplicativo desenvolvido;
6. Validar e verificar do aplicativo em dois momentos, sendo um momento inicial e outro após melhorias propostas a partir de testes de usabilidade, compreendendo as seguintes etapas:
    a. Definição do plano de testes;
    b. Definição do local de testes;
    c. Definição e recrutamento dos participantes;
    d. Preparação dos materiais para testes;
    e. Condução dos testes;
    f. Coleta e análise dos dados;
    g. Adição de novas funcionalidades com base nos resultados dos testes de usabilidade.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo fornecer ao leitor uma introdução aos diversos conceitos, tecnologias e termos técnicos que serão abordados ao longo deste trabalho. Serão apresentados os fundamentos necessários para compreender as discussões e análises realizadas nas próximas seções. Será explorado o vocabulário técnico específico, de modo a garantir que todos os termos e conceitos sejam compreendidos de forma clara e precisa. Ao estabelecer essa base de conhecimento, busca-se facilitar a compreensão e o acompanhamento das informações e argumentos apresentados posteriormente.

2.1 Aplicativos Móveis

Existem duas linguagens principais no desenvolvimento de aplicativos móveis, a saber: a linguagem nativa e a híbrida. Aplicativos em linguagem nativa são desenvolvidos para uma plataforma específica, como iOS ou Android, utilizando

as linguagens de programação e ferramentas nativas disponíveis para cada sistema operacional. Já os aplicativos em linguagem híbrida são desenvolvidos usando tecnologias web encapsulados em um contêiner nativo (Usemobile, s.d.).

2.1.1 Aplicativos Nativos

Desenvolver aplicativos em linguagem nativa é interessante devido aos benefícios que essa abordagem traz. A linguagem nativa permite a criação de aplicativos específicos para cada plataforma, explorando todo o potencial e recursos disponíveis em cada sistema operacional. Isso resulta em um desempenho otimizado, melhor integração com o sistema, interface nativa e uma experiência de usuário mais fluida e intuitiva. Além disso, ao utilizar linguagem nativa, os desenvolvedores têm acesso a todas as ferramentas, bibliotecas e *Application Interface Programming* (APIs) oferecidas pela plataforma, permitindo a criação de recursos avançados e personalizados. Isso garante que o aplicativo esteja alinhado com as diretrizes e padrões de design específicos de cada plataforma, no caso da plataforma iOS o *Human Interface Guidelines (*HIG), conforme as diretrizes da Apple Inc. (2023), proporcionando uma experiência consistente e familiar para os usuários. Em resumo, desenvolver aplicativos em linguagem nativa maximiza o desempenho, a funcionalidade e a usabilidade, resultando em aplicativos de alta qualidade e satisfação dos usuários.

2.1.2 Sistema iOS

O iOS (iPhone Operating System) é o sistema operacional da Apple para dispositivos móveis, como iPhone, iPad e iPod Touch. Lançado em 2007 junto com o iPhone, o iOS trouxe uma interface inovadora baseada em toques na tela, substituindo os botões físicos. A App Store, lançada em 2008, permitiu o *download* de aplicativos de terceiros. Recursos como copiar e colar, multitarefa, FaceTime, iCloud e Siri foram adicionados nas versões subsequentes. O iOS 7.0 trouxe uma mudança visual marcante, e desde então, o sistema continua a evoluir, oferecendo melhorias de desempenho, segurança, privacidade e recursos de realidade aumentada. De acordo com a Apple Inc. (2023) o iOS é uma das principais plataformas móveis do mundo, com uma ampla variedade de aplicativos disponíveis na App Store e continua a desenvolver e aprimorar o iOS com atualizações regulares para proporcionar a melhor experiência aos usuários.

2.1.3 Anatomia de um Aplicativo iOS

A Apple disponibiliza *frameworks* essenciais para desenvolvedores de aplicativos no ecossistema iOS, visando interações de qualidade e responsividade. Destacam-se o UIKit, que permite a criação de interfaces de usuário atraentes e intuitivas, e a Foundation, que oferece recursos para gerenciamento de dados, arquivos e comunicação em rede. A introdução da linguagem Swift trouxe avanços

significativos, substituindo a linguagem Objective-C e proporcionando maior produtividade e legibilidade de código. Segundo Magalhães (2022), essas melhorias impactam profundamente a forma como os desenvolvedores criam e interagem com o sistema, resultando em aplicativos de alta qualidade e ótimo desempenho no iOS.

2.1.3.1 UIKit

Conforme a documentação da Apple Inc. (2023), o UIKit é um *framework* fundamental para a criação de interfaces de usuário no iOS. Ele oferece um conjunto abrangente de ferramentas e componentes de interface, permitindo o desenvolvimento de aplicativos visualmente atraentes e intuitivos. Com o UIKit, os desenvolvedores podem criar layouts, botões, controles de navegação, animações e muito mais, proporcionando uma experiência de usuário aprimorada.

2.1.3.2 Foundation

A Foundation é outro *framework* essencial que, conforme a documentação da Apple Inc. (2023) oferece uma variedade de classes e funcionalidades para o desenvolvimento de aplicativos iOS. Ele abrange áreas como gerenciamento de dados, gerenciamento de arquivos, comunicação em rede, internacionalização e muito mais. Com a Foundation, os desenvolvedores podem acessar recursos poderosos e essenciais para o desenvolvimento de aplicativos robustos e confiáveis.

2.1.4 UI, Layouts e Delegação

A interface do usuário (User Interface - UI) em aplicativos iOS é principalmente definida por meio de *Storyboards, Views* e *ViewControllers*. Uma *ViewController*, de acordo com Magalhães (2022), desempenha o papel de controlar os comportamentos e interações entre a interface do usuário e os dados da aplicação. Cada aplicativo possui pelo menos uma. As *Views* são os elementos fundamentais da UI, tais como imagens, textos, botões e componentes complexos. Elas podem ser criadas via código, programaticamente, ou através da interface gráfica chamada de *Storyboard*.

O conceito de *Delegates* permite a coordenação entre objetos, onde um objeto pode agir em resposta a outro. Essa coordenação ocorre por meio da troca de mensagens e define como os componentes devem reagir a eventos específicos.

Em resumo, a definição da UI em aplicativos iOS envolve o uso de *Storyboards* e *ViewControllers*, onde as *Views* desempenham um papel central na exibição de elementos visuais. Através do conceito de *Delegates*, é possível coordenar e controlar as interações e comportamentos dos componentes da interface do usuário.

2.1.5 Ciclo de vida de um aplicativo iOS

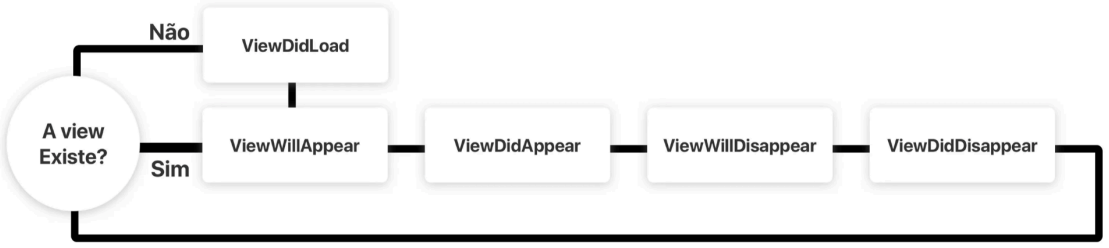
Os eventos que governam o ciclo de vida de um *UIViewController* fornecem pontos de entrada e saída que permitem aos desenvolvedores controlar e ajustar o comportamento de uma tela em diferentes momentos, conforme o trabalho de Mello (2018). Sendo eles:

1. *ViewWillAppear*(): Chamado logo antes da *View* aparecer;
2. *ViewDidAppear*(): Chamado logo depois da *View* aparecer;
3. *ViewDidLoad*(): Chamado quando, além de já ter aparecido, a *View* tiver sido completamente carregado;
4. *ViewWillDisappear*(): Chamado logo antes da *View* desaparecer;
5. *ViewDidDisappear*(): Chamado logo após a *View* desaparece.

Os eventos no ciclo de vida de uma tela em um aplicativo iOS ocorrem de acordo com a interação do usuário. Quando o usuário entra pela primeira vez no aplicativo, a função *ViewDidLoad*() é chamada para criar a tela inicial. Quando o usuário navega para uma próxima tela, a tela inicial permanece em segundo plano. Ao retornar para a tela inicial, as funções *ViewWillAppear*() e *ViewDidAppear*() são acionadas para confirmar que a tela foi apresentada ao usuário. Quando o usuário navega para outra tela, as funções *ViewWillDisappear*() e *ViewDidDisappear*() são chamadas para remover a tela atual da vista. Esses eventos são ilustrados na Figura 1.

De acordo com Sánchez (2023), os eventos citados fornecem controle sobre o ciclo de vida de uma *UIViewController*, permitindo que os desenvolvedores personalizem o comportamento de cada tela em diferentes estágios.

Figura 1 - Ciclo de Vida de Uma Aplicação iOS



Fonte: SÁNCHEZ, Jonattan Nieto. [Inédito]. 2023.

2.1.6 Padrões Arquiteturais

A Arquitetura de *Software* é o campo da Ciência da Computação que define a estrutura, funcionamento e interação entre as partes de um *software*, sendo o nível mais alto de design de um sistema. Uma arquitetura é selecionada ou projetada com

base nos requisitos e restrições do problema a ser resolvido. Ela abstratamente define os componentes, interfaces e comunicação entre eles, segundo o estudo de Sánchez (2023).

2.1.6.1 MVC

A Apple apresenta, segundo suas diretrizes em Apple Inc (2023), o Modelo-Visão-Controlador (MVC) como seu modelo de arquitetura padrão. Nesse modelo, o controlador atua como um mediador entre a visão e o modelo, sendo que a visão é responsável pela apresentação da interface do usuário (*Storyboards*), o controlador coordena as interações entre a visão e o modelo (*ViewControllers*), enquanto o modelo gerencia os dados e a lógica de negócios da aplicação (*Models)*.

No entanto, um desafio surge com o controlador, pois ele acaba reunindo toda a lógica de negócio devido à implementação da classe nativa da Apple, o *UIViewController*. Essa abordagem resulta em um problema conhecido como "*Massive View Controller*" (Controlador de Visão Massivo).

Segundo Magalhães (2022), no *Massive View Controller*, o controlador se torna sobrecarregado com uma quantidade significativa de responsabilidades, tornando-o difícil de gerenciar e manter. A aglomeração de lógica de negócio nessa camada compromete a testabilidade do código, dificultando a aplicação efetiva de testes. Essa limitação levou a comunidade de desenvolvedores a buscar soluções alternativas para lidar com o problema, a fim de tornar o código mais modular, escalável e testável. Essas soluções incluem a adoção de padrões arquitetônicos como MVVM (Modelo-Visão-Modelo de Visualização), que conforme Cadu (2010) distribuem as responsabilidades de forma mais equilibrada entre as camadas e facilitam a testabilidade.

Embora a abordagem da Apple com o Modelo-Visão-Controlador tenha sido amplamente adotada na comunidade iOS, é importante reconhecer suas limitações e considerar outras alternativas para garantir a manutenção de um código robusto e de fácil testabilidade.

2.1.6.2 MVVM

A arquitetura MVVM (*Model-View-ViewModel*), foi criada por um dos arquitetos por trás das tecnologias WPF (*Windows Presentation Foundation)* e *Silverlight* da Microsoft em 2005, John Gossman, e é definida por estabelecer uma clara separação de responsabilidades em uma aplicação, conforme Cadu (2010). No MVVM, a camada *Model* representa a lógica de negócios e os dados subjacentes à aplicação e não tem conhecimento direto da *View* (Camada de apresentação) e vice-versa. Em vez disso, a *View* se relaciona com a *ViewModel*, que atua como intermediária entre a *Model* e a *View*, essa separação estrita garante que a lógica de apresentação e a lógica de negócios permaneçam desacopladas.

A comunicação entre a *View* e a *ViewModel* no MVVM acontece através do *binding*, um mecanismo que permite que as propriedades da *ViewModel* sejam

vinculadas diretamente aos elementos de interface na *View*. Isso significa que as atualizações feitas na *ViewModel* são refletidas automaticamente na *View*, e vice-versa, tornando a manutenção e a atualização da interface do usuário mais eficientes e livres de erros.

Em resumo, o MVVM é uma arquitetura que promove uma organização clara do código, com a separação de responsabilidades entre *Model*, *View* e *ViewModel*. Isso resulta em aplicativos mais fáceis de manter, testar e evoluir, tornando-o uma escolha valiosa para o desenvolvimento de software moderno.

2.1.6.3 VIP

O *Clean Swift*, também conhecido como VIP (*View*, *Interactor*, *Presenter*), é uma arquitetura que traz as ideias da Arquitetura Limpa para o ambiente de desenvolvimento iOS, segundo Carvalho (2021). A sua origem está na busca por maior modularização e testabilidade do código, um problema comum enfrentado no desenvolvimento de aplicativos para iOS, onde os *View Controllers* frequentemente se tornam "*Massive View Controllers*," ou seja, classes com uma quantidade excessiva de responsabilidades.

Carvalho (2021) também explica que para resolver esse problema, o *Clean* Swift adota uma abordagem de separação de responsabilidades. O ciclo VIP define como a comunicação ocorre no VIP, seguindo um fluxo unidirecional e utilizando protocolos para estabelecer essa comunicação. Nesse modelo, a View é responsável pela interface do usuário e pela captura de interações do usuário. O Interactor contém a lógica de negócios, processando dados e tomando decisões. O Presenter atua como intermediário, recebendo ações da View, solicitando operações ao Interactor e atualizando a View com os resultados. Ao contrário do MVVM, o VIP é unidirecional, as informações navegam da *Interactor* para a *View*, passando pelo *Presenter*, sempre nesta ordem.

Essa divisão clara de tarefas facilita a testabilidade, tornando possível testar as partes do aplicativo separadamente. Além disso, ajuda na manutenção, permitindo que as mudanças na lógica de negócios sejam feitas sem afetar a interface do usuário. O *Clean* Swift/VIP é uma arquitetura eficaz para criar aplicativos iOS mais organizados, escaláveis e robustos.

2.1.7 Padrões de Navegação

Para o desenvolvimento de um aplicativo é fundamental estipular a abordagem correta para o tipo de navegação. De acordo com Rausch (2024), quando os aplicativos possuem uma boa navegação, as pessoas conseguem se concentrar no conteúdo e na experiência, sem perceber os detalhes da navegação, o objetivo é familiarizar os usuários com o funcionamento do aplicativo, permitindo que eles descubram facilmente o conteúdo e interajam intuitivamente, resultando em uma experiência mais satisfatória e eficiente (Apple Inc.,2022).

Entre os padrões de navegação comumente utilizados estão os *drill-downs*, modais, pirâmides, passo a passo e outros.

Figura 2 - Exemplo de navegação Passo a Passo

Fonte: Rausch (2023)

2.2 Conceitos Computacionais

Com o objetivo de familiarizar o leitor com conceitos computacionais, nesta seção são apresentados os principais tópicos que fundamentam as tecnologias utilizadas no desenvolvimento prático do trabalho. Isso inclui a orientação a objetos, que é o paradigma da linguagem Swift, bem como o site GitHub, amplamente utilizado pela comunidade para o versionamento de código.

2.2.1 Programação Orientada à Objeto

Programação Orientada a Objetos (OOP) é um método de organização de programas que visa simplificar o gerenciamento da complexidade do código. Conforme o artigo *'Object-Oriented Programming, Functional Programming, and R*' publicado na revista *Statistical Science* (Volume 29, Número 2), em OOP, os elementos de um programa são estruturados em objetos, que são como unidades autônomas que contêm tanto dados quanto as ações que podem ser realizadas com esses dados. Essa abordagem ajuda a tornar o código mais organizado e legível, especialmente em projetos de grande complexidade. Em essência, OOP permite que o desenvolvedor crie 'objetos' que representem entidades do mundo real ou conceitos abstratos, tornando mais fácil modelar e resolver problemas complexos de maneira mais intuitiva.

2.2.2 IDE

Uma IDE (Ambiente de Desenvolvimento Integrado) segundo Mattos et al. (2019) é uma ferramenta de *software* que reúne um editor de texto para inserir código-fonte, capacidade de execução de código, depurador para correção de erros, identificação de erros sintáticos e semânticos, além de um interpretador ou compilador integrado. Essa combinação de recursos permite aos desenvolvedores criar, depurar e executar programas de maneira eficiente em um único ambiente, tornando-o fundamental para o processo de desenvolvimento de *software*.

2.2.3 Mocar dados

Um "*mock*" (pronuncia-se como "mawk" em inglês) é um termo comumente usado em desenvolvimento de *software* para se referir a um objeto ou componente simulado que é usado durante testes de unidade e integração. De acordo com Felipe (2021), um *mock* é uma representação fictícia de um objeto ou serviço real, criado com o propósito de simular seu comportamento em um ambiente controlado.

A principal função de um *mock* é permitir que os desenvolvedores testem partes específicas de um *software*, isoladamente, sem depender de componentes ou serviços externos reais. Isso ajuda a identificar problemas e erros em um componente individual, sem a complexidade de testar todo o sistema.

2.2.3 Softwares e plataformas

2.2.3.1 GitHub

GitHub é considerado a maior coleção de *software* de código aberto do mundo. De acordo com Borges, Hora e Valente (2016), é um local onde projetos de *software* são hospedados e compartilhados publicamente. Os desenvolvedores

podem usar o GitHub para armazenar, colaborar e controlar versões de seus projetos de código aberto. Ele possui também um recurso chamado "*stargazers button*" (botão de "*stargazers*"), que permite aos usuários manifestarem sua satisfação com um repositório específico, indicando assim sua popularidade.

Em resumo, o GitHub é uma plataforma de hospedagem de código aberto que desempenha um papel importante na colaboração e disseminação de projetos de software de código aberto.

2.2.3.2 XCode

De acordo com Kelly e Nozz, no livro "*Mastering* Xcode" (2014, p. 166), o Xcode é um ambiente de desenvolvimento integrado (IDE) desenvolvido pela Apple para a criação de aplicativos para seus sistemas operacionais, como iOS, macOS, watchOS e tvOS. Este poderoso *software* oferece uma variedade de ferramentas que simplificam e agilizam o processo de desenvolvimento de aplicativos.

O Xcode inclui um editor de código que suporta várias linguagens, como Swift e Objective-C, permitindo que os desenvolvedores escrevam o código-fonte do aplicativo. Além disso, ele possui recursos de depuração que ajudam a identificar e corrigir erros no código, também inclui ferramentas de análise de desempenho e testes, facilitando a otimização e a verificação da qualidade do aplicativo antes de ser lançado no mercado (Apple Inc., 2023).

Em resumo, o Xcode é uma ferramenta essencial para desenvolvedores de aplicativos da Apple, proporcionando um ambiente completo e eficiente para criar, depurar e testar aplicativos para uma ampla gama de dispositivos e sistemas operacionais da Apple.

2.2.3.3 Figma

O Figma é uma ferramenta vetorial que se concentra principalmente no design de interfaces. Segundo Saga (2020), uma ferramenta vetorial é uma aplicação de design que utiliza formas geométricas primitivas, como linhas, curvas, pontos e polígonos, baseada em fórmulas matemáticas específicas para criar imagens com precisão, onde os elementos são construídos por vetores, representados por pontos ou nós, definindo características como início, fim e curvatura das linhas. Além disso, de acordo com o artigo da XP Educação (2022), o Figma se destaca pelo seu recurso de trabalho colaborativo. Isso o torna uma ferramenta poderosa para equipes que trabalham remotamente, pois facilita a integração de ideias, reduz o risco de perda de arquivos e simplifica o processo de criação em grupo. A capacidade de fazer alterações em tempo real e compartilhar projetos de forma instantânea é uma vantagem significativa, tornando o Figma uma escolha sólida para equipes de design que buscam eficiência e colaboração na criação de interfaces atraentes e funcionais.

2.3 Demais Conceitos

2.3.1 Usabilidade

Usabilidade é a medida de quão fácil e eficaz é a utilização de um produto ou serviço digital por parte dos usuários para alcançar seus objetivos específicos, levando em consideração eficácia, eficiência e satisfação, dentro de um contexto de uso específico.

Ela envolve a facilitação da interação entre o usuário e a tecnologia, considerando fatores como design, recursos financeiros disponíveis, contexto dos usuários e suas competências. No decorrer do tempo, a compreensão da usabilidade evoluiu de uma qualidade binária para uma característica contínua que abrange várias dimensões, incorporando aspectos como diversão, bem-estar, estética e suporte para o desenvolvimento humano, refletindo uma visão mais ampla e holística da interação humano-computador (MARTINS et al., 2013, p. 15).

2.3.1.1 Teste de Usabilidade

Segundo os estudos de Martins et al. (2013, p. 35), o teste de usabilidade é um método de avaliação que envolve a observação dos usuários enquanto eles realizam tarefas com um determinado produto ou serviço. Durante esse processo, são coletados dados principalmente quantitativos para buscar evidências empíricas sobre como melhorar a usabilidade da interface de interação. O teste de usabilidade pode abordar questões simples, como a conclusão bem-sucedida de tarefas, ou questões mais complexas, como o nível de satisfação dos usuários finais.

2.3.2 Dimensionamento de Carga Elétrica

A NBR 5410 é a norma brasileira que estabelece as diretrizes para a instalação elétrica de baixa tensão, garantindo segurança e eficiência nas edificações. O dimensionamento de carga, segundo a NBR 5410, envolve a determinação da capacidade adequada dos condutores, dispositivos de proteção e outros componentes do sistema elétrico para suportar a demanda energética prevista. Este processo considera fatores como o tipo de edificação, a quantidade e o tipo de equipamentos elétricos, além das condições ambientais e de instalação. O objetivo é assegurar que o sistema elétrico opere dentro de parâmetros seguros, evitando sobrecargas, quedas de tensão e riscos de incêndio ou choque elétrico. O correto dimensionamento é fundamental para a durabilidade e funcionalidade das instalações, além de atender aos requisitos regulamentares e de segurança.

2.3.3 UI/UX

UI (User Interface) e UX (User Experience) são dois conceitos fundamentais no design de produtos digitais. De acordo com Rodrigues (2023), enquanto UI se

concentra na criação da aparência e interação visual de um produto, UX está mais preocupado com a experiência geral do usuário. Em resumo, UI trata da parte visual e interativa, enquanto UX envolve todos os aspectos da interação do usuário com o produto, buscando torná-la eficiente, eficaz e agradável. Juntos, UI e UX trabalham para criar produtos digitais que atendam às necessidades dos usuários e superem suas expectativas.UI (User Interface) e UX (User Experience) são dois conceitos fundamentais no design de produtos digitais. Enquanto UI se concentra na criação da aparência e interação visual de um produto, UX está mais preocupado com a experiência geral do usuário. Em resumo, UI trata da parte visual e interativa, enquanto UX envolve todos os aspectos da interação do usuário com o produto, buscando torná-la eficiente, eficaz e agradável. Juntos, UI e UX trabalham para criar produtos digitais que atendam às necessidades dos usuários e superem suas expectativas.

2.4 Trabalhos Relacionados

2.4.1 Introdução aos Trabalhos Relacionados

A revisão da literatura desempenha um papel fundamental na elaboração da pesquisa fornecendo uma base sólida de conhecimento, contextualizando o TCC sobre o tema escolhido - o desenvolvimento de um aplicativo móvel para dimensionamento de carga. E permitindo à pesquisadora explorar os estudos, projetos e inovações anteriores relacionados ao tema.

A revisão da literatura também ajuda a justificar a relevância do TCC, mostrando que o projeto não está replicando o que já foi feito, mas contribuindo com avanços específicos para atender às necessidades do dimensionamento de carga automatizado.

Nesse contexto, esta seção apresentará uma introdução aos trabalhos relacionados ao desenvolvimento de um aplicativo móvel usando o sistema iOS para automatizar o dimensionamento de carga, destacando a importância desta revisão na pesquisa em questão.

2.4.1.1 Dimensionamento de potência de geração fotovoltaica visando consumidores do grupo A

Xavier (2019) também desenvolveu, em seu TCC do curso de Engenharia Elétrica, um estudo sobre o dimensionamento de cargas. Seu projeto propõe uma forma economicamente viável de dimensionar sistemas de geração fotovoltaica, direcionada a consumidores do grupo A, com o objetivo de ajudá-los a minimizar os custos com energia elétrica. O Grupo A refere-se a consumidores de energia elétrica de grande porte, como empresas e indústrias, sujeitos a tarifas diferenciadas. Para alcançar esse objetivo, o projeto envolve análises da demanda contratada, a fim de verificar sua adequação às demandas reais dos consumidores, bem como a

identificação da melhor modalidade tarifária com base no perfil energético do grupo A.

O aplicativo proposto se diferencia do estudo de otimização de Xavier por ter foco em fornecer ao consumidor controle sobre o dimensionamento de carga e não sobre o dimensionamento de sistemas de geração fotovoltaica, e também por ter seu foco em baixa tensão, considerando UC's residenciais do grupo B.

No aplicativo desenvolvido neste TCC é esperado que o consumidor possa, através do seu uso, conhecer os fatores envolvidos no dimensionamento de sua demanda energética e escolher com clareza e facilidade os valores que compõem seu gasto energético mensal, bem como sua demanda contratada.

2.4.1.2 Aplicativo para cálculo de demandas segundo as normas de distribuição da CEB e dimensionamentos básicos

O projeto de Martins e Komatsu (2016) visa aprimorar um aplicativo anterior para cálculo de demanda elétrica desenvolvido pela Microsoft. Este aplicativo, em conformidade com normas da Companhia Energética de Brasília (CEB), tem como objetivo calcular a demanda elétrica, determinar o tipo de fornecimento e fornecer dados de dimensionamento básico de cabos elétricos para circuitos internos. A intenção é criar uma ferramenta intuitiva e de fácil uso para aumentar a produtividade em projetos de instalações elétricas, com a vantagem de ser compacta e compatível por ser desenvolvida em Excel. O foco está em garantir que o dimensionamento de carga seja preciso, evitando subdimensionamento ou superdimensionamento dos circuitos elétricos, considerando tanto a economia quanto a segurança das instalações elétricas.

O aplicativo proposto neste TCC se diferencia do aplicativo de Martins e Komatsu em sua natureza de concepção. Enquanto um é um aplicativo a ser usado por profissionais da área, em sua maioria, e em microcomputadores (Microsoft Excel), outro é concebido para ser utilizado por usuários leigos em dispositivos móveis que possuem sistema iOS, ou seja, iPhones.

2.4.1.3 Desenvolvimento de aplicativo computacional para cálculo de demanda segundo as normas de distribuição da CEB e dimensionamento de condutores e dispositivos de proteção segundo a ABNT NBR 5410:2004

O projeto de Neves Júnior (2016) trata do desenvolvimento de um aplicativo computacional chamado "ACADIE" para cálculos e dimensionamentos em instalações elétricas. O objetivo principal do projeto é auxiliar projetistas de instalações elétricas a realizar cálculos de demanda, determinar o tipo de fornecimento e dimensionar o ramal de entrada, seguindo as normas técnicas de distribuição da Companhia Energética de Brasília (CEB) para diferentes tipos de unidades consumidoras, incluindo individuais, prédios de múltiplas unidades e instalações com distribuição em tensão primária. Além disso, o aplicativo também

realiza o dimensionamento de condutores e dispositivos de proteção contra sobrecargas de acordo com os critérios da ABNT NBR 5410.

Já o atual projeto trata de um aplicativo *mobile*, como já mencionado, com o principal objetivo de orientar qualquer usuário a determinar seu dimensionamento de carga de forma fácil e intuitiva, gerando, se necessário, a demanda de maneira automatizada.

2.4.2 Identificação de Lacunas na Literatura

A análise dos projetos de Neves Júnior, Martins e Komatsu, bem como de Xavier, revela algumas lacunas identificadas na literatura em relação ao dimensionamento de carga e geração de energia. Enquanto Neves Júnior aborda o desenvolvimento de um aplicativo para computador, focando em projetistas e profissionais da área, Xavier se concentra na geração fotovoltaica para consumidores do grupo A, também pensado para projetistas e profissionais da área.

A diferença entre esses projetos mostra a necessidade de aplicativos mais direcionados para consumidores leigos, no que diz respeito ao dimensionamento de sua própria carga. Portanto, a literatura carece de ferramentas acessíveis para a gestão da demanda de energia e para auxiliar consumidores comuns na escolha de seus gastos energéticos e potência contratada. Essas lacunas ressaltam a importância de futuras pesquisas na criação de aplicativos mais abrangentes e amigáveis para atender às diferentes necessidades dos consumidores, promovendo a eficiência energética.

2.4.3 Conclusão da Revisão da Literatura

Portanto, conclui-se que há espaço para futuras pesquisas que abordem uma gama mais ampla de necessidades dos consumidores e promovam a eficiência energética de forma abrangente.

3 METODOLOGIA

3.1 Pesquisa

A pesquisa tem caráter qualitativo, que, segundo Silveira e Córdova (2009, p. 33-34), é uma abordagem que se diferencia da pesquisa quantitativa por se concentrar na análise de dados não-métricos, ou seja, dados que não podem ser quantificados em números ou medidas precisas. Em vez disso, a pesquisa qualitativa lida com informações mais subjetivas e descritivas, como opiniões, percepções, experiências e narrativas.

Uma das características que diferencia a pesquisa qualitativa é o uso do pesquisador como o principal instrumento de coleta de dados. Isso significa que o pesquisador desempenha um papel ativo na coleta, interpretação e análise dos dados, sendo que essas interações podem ocorrer por meio de entrevistas,

observações participantes, grupos focais, entre outras técnicas. Além disso, na pesquisa qualitativa, há uma ênfase no raciocínio intuitivo. Isso significa que os pesquisadores se baseiam em suas habilidades interpretativas para compreender e dar significado aos dados coletados.

3.2 Natureza

Pesquisa de natureza aplicada, que foca na solução de problemas específicos de maneira prática, bem como em gerar conhecimentos para aplicação prática, envolvendo verdades e interesses locais. (SILVEIRA; CÓRDOVA, [Data de publicação não informada], p. 35).

3.3 Objetivo

O objetivo da pesquisa é de carácter exploratório, pois proporciona familiaridade com o problema proposto e cria hipóteses de solução a partir de pessoas que tiveram experiências práticas com o produto estudado. (SILVEIRA; CÓRDOVA, [Data de publicação não informada], p. 35).

3.4 Método

A metodologia empregada segue o princípio da abordagem hipotético-dedutiva, na qual a identificação de um problema estimula a formulação de uma hipótese. Essa hipótese, por sua vez, é submetida a testes empíricos meticulosos para determinar sua validade ou refutação (GIL, 2008, p. 49-59). Esse método, amplamente reconhecido na pesquisa científica, permite uma análise criteriosa e fundamentada, contribuindo para o avanço do conhecimento e a compreensão mais profunda das questões investigadas.

3.5 Procedimentos Técnicos

De acordo com Silveira e Córdova ([Data de publicação não informada], p. 39), para se desenvolver uma pesquisa, é indispensável selecionar o método de pesquisa a utilizar. De acordo com as características por este citadas, esta pesquisa se caracteriza por utilizar procedimentos de pesquisa com *survey,* onde existe a busca de informação diretamente com um grupo de interesse para obtenção dos dados que se deseja, utilizando um questionário como instrumento de pesquisa.

3.6 Etapas do Desenvolvimento do Trabalho

A estrutura deste trabalho compreende quatro partes fundamentais, cada uma delas detalhada nas subseções subsequentes. Essas seções constituem os pilares que sustentam a abordagem deste estudo, oferecendo uma visão abrangente e aprofundada do assunto em análise.

3.6.1 Levantamento Bibliográfico e Documental

Com o objetivo de enriquecer e aprofundar o conhecimento técnico em relação aos conceitos, tecnologias e termos específicos que permeiam o escopo deste trabalho, foram conduzidas pesquisas bibliográficas e documentais. Durante o processo foram analisados artigos e livros relevantes à temática em questão, com ênfase na seleção de publicações atualizadas. Além disso, a investigação se estendeu ao âmbito das pesquisas abrangendo a análise crítica de documentos normativos, leis e regulamentos que guardam pertinência com o assunto em estudo.

A compilação desse levantamento e a síntese das informações obtidas são apresentadas de forma sistemática no Capítulo 2 - Fundamentação Teórica, contribuindo significativamente para a fundamentação teórica e contextualização do presente trabalho.

3.6.2 Design UI/UX

A otimização da usabilidade do aplicativo móvel teve início com uma exploração detalhada dos padrões de design mais prevalentes no contexto de UI/UX (interação do usuário/experiência do usuário). Esse processo não apenas serviu como um ponto de partida, mas também como um alicerce sólido para orientar o desenvolvimento de cada interface de forma única e autêntica. A compreensão profunda desses padrões proporcionou as bases necessárias para a criação de experiências de usuário envolventes e eficazes.

3.6.3 Definição e Recrutamento dos Participantes

Além das definições teóricas, foi elaborada a concepção de uma *persona* que melhor se alinha com o perfil do usuário primário do aplicativo.

Em paralelo, foi realizado o recrutamento criterioso de participantes dispostos a colaborar no processo de teste de usabilidade, selecionados com base na conformidade com o perfil delineado para a *persona*.

Essa abordagem garantiu que o grupo de participantes, composto por 8 pessoas e abordado no Capítulo 5, Testes de Usabilidade, representasse de maneira precisa o público-alvo do aplicativo. Isso otimizou a relevância e a eficácia dos testes de usabilidade realizados.

3.6.4 Desenvolvimento do aplicativo

A concepção da primeira edição do aplicativo teve início durante o curso de Projeto Integrador III, disciplina ministrada no 9º período do curso de engenharia elétrica no IFSC câmpus Itajaí, conforme mencionado no Capítulo 1 - Introdução. As diversas etapas desse processo, incluindo a criação de cada tela e suas respectivas funcionalidades, são detalhadas na Seção 4.

Durante o desenvolvimento deste projeto, houve aprimoramentos significativos que culminaram na criação da versão 2.0 do aplicativo. Essas melhorias abrangeram diversos aspectos, desde a otimização da interface do usuário (UI), focada nos conceitos de experiência do usuário (UX), até a realização de testes de usabilidade, refatoração da arquitetura e a introdução de novas funcionalidades, como será descrito na Seção 6.

3.6.5 Teste de Usabilidade

A estruturação das etapas relacionadas aos testes de usabilidade seguem uma sequência lógica, que compreende a definição do plano de teste, a seleção do local adequado para a realização dos testes, o estabelecimento dos critérios a serem avaliados durante os testes e o recrutamento dos participantes. Em seguida, ocorre a preparação dos materiais essenciais para a condução eficaz dos testes, seguida pela realização das sessões propriamente ditas.

Após a conclusão das sessões de teste, os dados coletados passam por uma análise detalhada, proporcionando *insights* valiosos ao pesquisador. Por fim, os resultados são apresentados e as conclusões, consequentemente, derivam dessas análises, proporcionando uma visão abrangente do desempenho do aplicativo em termos de usabilidade. Na Seção 5 o processo e os resultados obtidos são aprofundados, e pode-se obter mais informações sobre esses testes de usabilidade.

3.6.6 Análise dos Resultados, Refatoração e Conclusão

A fase final deste projeto, após a análise dos resultados obtidos, abrange a concepção do projeto de protótipo que culminará na versão 2.0 do aplicativo, além de sua conclusão.

Neste estágio, a comparação dos resultados com os objetivos inicialmente propostos desempenha um papel essencial, proporcionando a validação da hipótese originalmente formulada. Ao analisar os dados coletados e suas implicações, os *insights* e aprendizados adquiridos durante o processo de teste de usabilidade permitem orientar de forma mais precisa e informada a evolução do aplicativo. Essa etapa serve para o aprimoramento contínuo do produto, garantindo que as melhorias se alinhem com as necessidades e expectativas dos usuários. Essa evolução representa não apenas um avanço técnico, mas também a validação prática das suposições e metas iniciais.

4 DESENVOLVIMENTO DO APLICATIVO

Conforme mencionado anteriormente no capítulo 1 - "Introdução", o presente trabalho é uma continuação da versão inicial do aplicativo desenvolvido no Projeto Integrador III, disciplina ministrada no 9º período do curso de engenharia elétrica no IFSC câmpus Itajaí e nomeado como Power Scale. Com o objetivo de fornecer uma orientação mais completa ao leitor sobre o desenvolvimento do aplicativo, esta

seção descreve as funcionalidades já desenvolvidas, abrangendo a concepção do aplicativo e seu desenvolvimento.

Assim, são definidas as seguintes subseções de acordo com a Tabela 1, tendo como base o momento em que foram desenvolvidas:

Tabela 1 - Etapas do desenvolvimento da aplicação Power Scale

Versionamento	Etapas de desenvolvimento
Protótipo Power Scale - Projeto Integrador III	4.1 Concepção Inicial 4.2 Modelagem de Dados 4.3 Desenvolvimento da Lógica de Funcionamento 4.4 Criação da Tela de Tipo de Negócio 4.5 Criação da Tabela de Eletrodomésticos 4.6 Criação da Tela de Resultados 4.7 Criação da Tela de PDF - Exportar Dados.

Fonte: Autoria própria.

4.1 Concepção Inicial

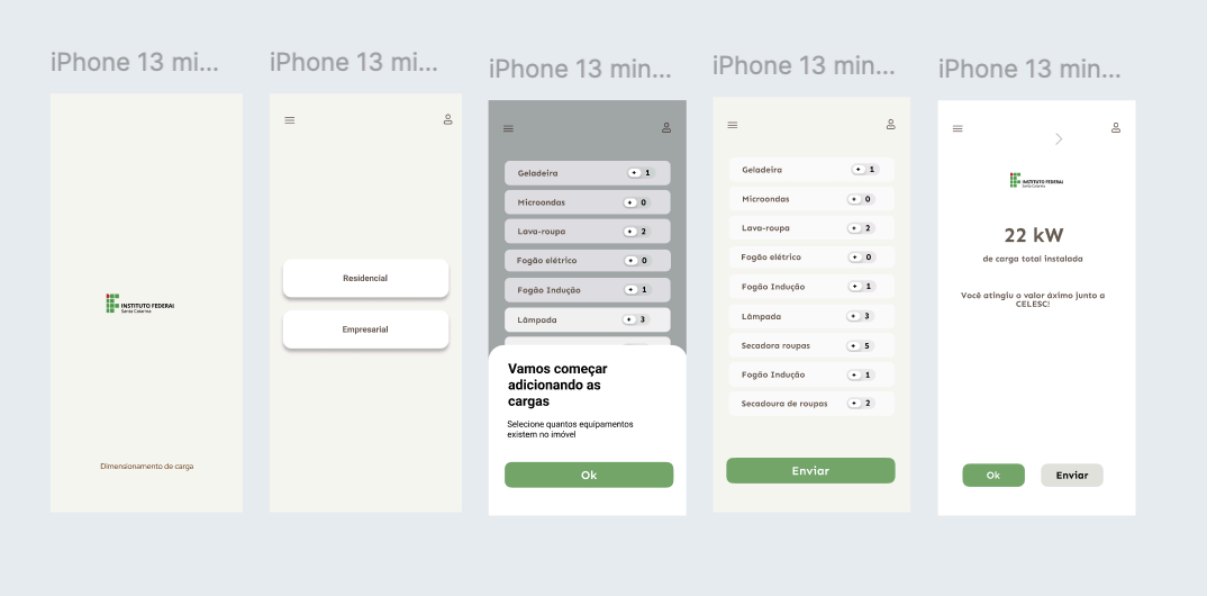
A criação de um aplicativo iOS é um processo complexo que requer um profundo conhecimento de desenvolvimento de software e familiaridade com as tecnologias fornecidas pela Apple. O ponto de partida para o desenvolvimento do Power Scale foi ter uma visão clara do propósito do aplicativo, o que envolveu identificar o problema a ser resolvido e elaborar um planejamento detalhado do design UI/UX utilizando o Figma. Isso envolveu a elaboração de wireframes, que segundo Miro são representações visuais esquemáticas da interface do usuário, proporcionando uma visualização da estrutura planejada para o aplicativo.

O design desempenhou um papel fundamental na criação do aplicativo móvel. Durante o processo, foi desenvolvida uma interface com cores que reflitam a identidade do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina. Além disso, o padrão escolhido para o desenvolvimento da aplicação foi o padrão passo a passo mostrado na Figura 2, conforme discutido no capítulo 2.1.7 - "Padrões de Navegação". Essa escolha foi motivada pela capacidade desse padrão em guiar os usuários por uma sequência de telas em um fluxo linear, oferecendo-lhes uma experiência orientada, pois de acordo com Raush (2023), o padrão "passo a passo" é comumente empregado em situações que exigem a condução dos usuários por uma série de etapas, como passeios guiados, configurações e tutoriais introdutórios. Além disso, é especialmente útil quando há a necessidade de inserir uma grande quantidade de dados, como no processo de

checkout de uma loja online, uma vez que divide o processo em etapas mais gerenciáveis.

Terminado o processo, foi alcançado o *design* de telas exemplificado na Figura 3 abaixo. Esse design exemplifica não apenas a estética desejada, mas também a funcionalidade intuitiva que foi almejada para a aplicação.

Figura 3 - Visão geral do aplicativo PowerScale.



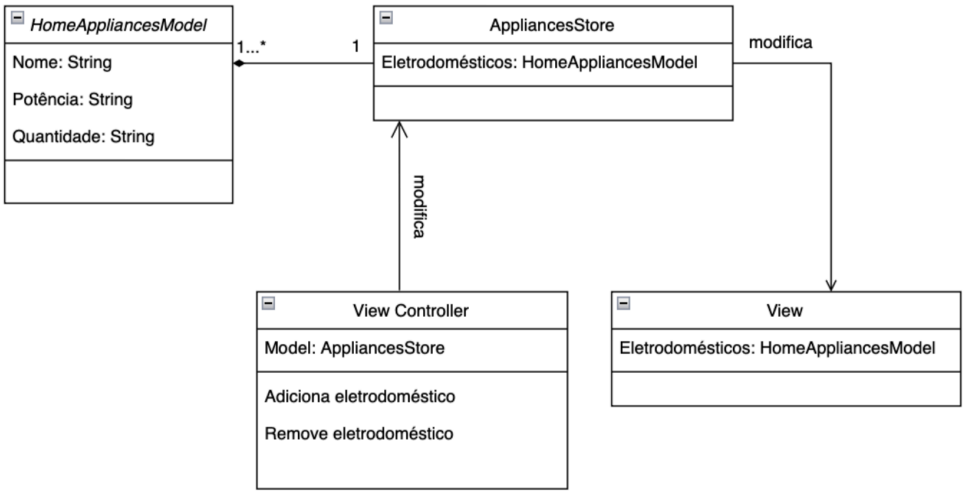
Fonte: Autoria própria.

4.2 Modelagem de Dados

Para a modelagem de dados desenvolveu-se a classe "Appliances Store", que inclui uma variável definida como "appliances" e configura um *array* de objetos do tipo "Home Appliance Model". É nesta classe que cada eletrodoméstico é inicializado (Figura 5), atribuindo a eles seus respectivos valores iniciais.

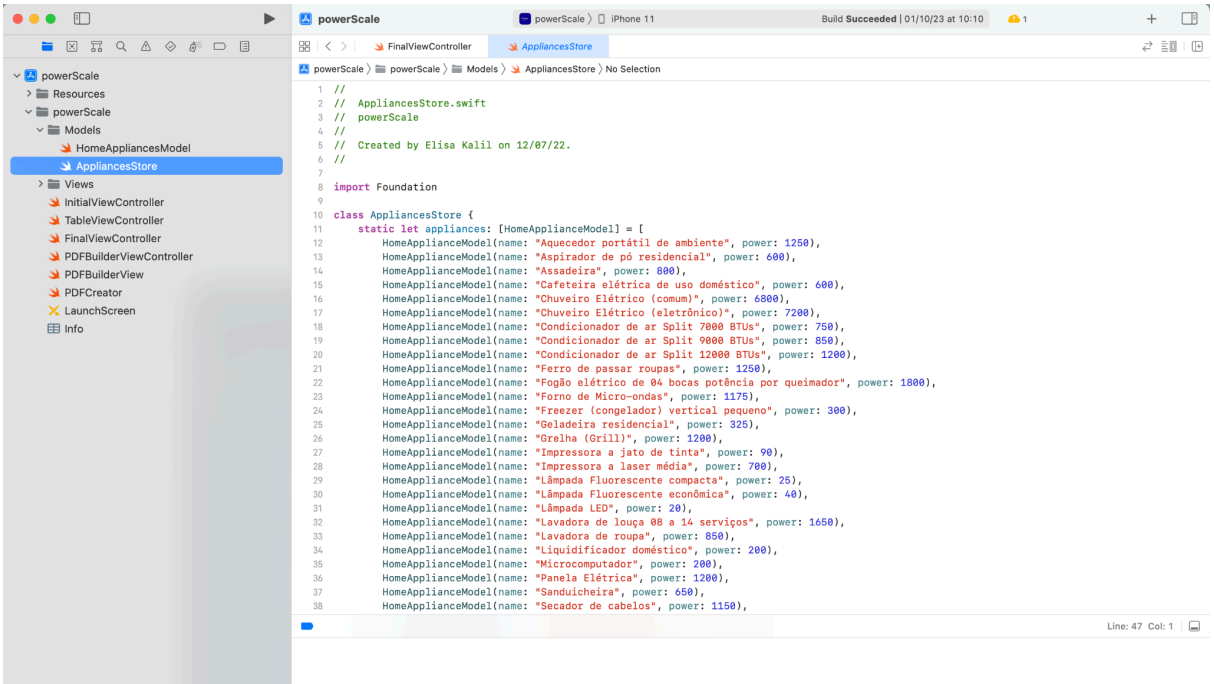
Essa variável atua como um repositório central que abriga todos os equipamentos disponíveis para manipulação pelo usuário. À medida que o usuário interage com a tela de equipamentos do aplicativo, os valores contidos nesta constante são atualizados dinamicamente em resposta às ações realizadas, todo o fluxo pode ser observado na Figura 4, que ilustra o diagrama de classes correspondente ao tratamento de dados da aplicação. Isso garante que todas as informações dos eletrodomésticos estejam sempre sincronizadas e prontas para serem utilizadas durante a interação do usuário com o aplicativo.

Figura 4 - Diagrama de Classes do Aplicativo V1.0



Fonte: Autoria própria.

Figura 5 - AppliancesStore



Fonte: Autoria própria.

No caso específico do aplicativo apresentado, a ideia é que ele seja uma solução em que os usuários não precisem criar contas, fazer login ou seguir procedimentos semelhantes. Essa abordagem visa atender à necessidade específica de fornecer aos usuários uma ferramenta simples e eficaz sem a complicação de processos de autenticação. Dessa forma, o aplicativo concentra-se

em sua função principal: ajudar os usuários a gerar o consumo de energia de sua residência de maneira rápida e eficiente.

Portanto, os dados necessários para o funcionamento do aplicativo se resumem a uma lista de equipamentos. Os usuários podem selecionar os equipamentos que possuem em casa, tendo seus respectivos valores de potência pré-setados pelo *app*. Não é necessário realizar um cadastro do usuário, o que simplifica significativamente a experiência do usuário e torna o uso do aplicativo mais prático e direto. Contudo os dados são mantidos durante a sessão de uso.

Para tal, tomou-se como referência a relação de equipamentos listados na tabela 01, localizada na página 24 do "Manual Simplificado - Padrão de Entrada de Energia Elétrica em Instalações Consumidoras" da CELESC, juntamente com seus respectivos valores de potência. Esta tabela apresenta os valores de potência padrão para diversos eletrodomésticos, como ilustrado na Figura 6 a seguir

Figura 6 - Potências típicas segundo a Celesc

TABELA **01**

Determinação da carga instalada e potências típicas dos eletrodomésticos

Aparelhos (1)	Faixa de Potência (W) (2)	Potência do Aparelho (3)	Quantidade (4)	Total da Potência (5)
Aquecedor portátil de ambiente	500 a 2000			
Aspirador de pó residencial	200 a 1000			
Assadeira	600 a 1000			
Cafeteira elétrica de uso doméstico	600			
Chuveiro Elétrico (comum)	6800			
Chuveiro Elétrico (eletrônico)	6000 a 8400			
Condicionador de ar Split 7000 BTUs (*)	750			
Condicionador de ar Split 9000 BTUs (*)	850			
Condicionador de ar Split 12000 BTUs (*)	1200			
Ferro de passar roupas	850 a 1650			
Fogão elétrico de 04 bocas potência por queimador	1500 a 2100			
Forno de Micro-ondas	850 a 1500			
Freezer (congelador) vertical pequeno	300			
Geladeira residencial	150 a 500			
Grelha (Gill)	1200			
Impressora a jato de tinta	90			
Impressora a laser média	700			
Lâmpada Fluorescente compacta	5 a 45			
Lâmpada Fluorescente econômica	15 a 65			
Lâmpada LED	3 a 35			
Lavadora de louça 08 a 14 serviços	1500 a 1800			
Lavadora de roupa	500 a 1200			
Liquidificador doméstico	200			
Microcomputador	200			
Panela Elétrica	1200			
Sanduicheira	650			
Secador de cabelos	500 a 1800			
Secadora de roupas	1500 a 4000			
TV LED 32 polegadas	75			
TV LED 55 polegadas	160			
Tomadas de uso geral	100			
Torneira Elétrica	4400			
Ventilador portátil	60 a 150			
Outros				
			Soma Total das Cargas (Watts)	
			Dividir por 1.000 (kW)	

(*) A potência elétrica e a capacidade em BTU podem variar um pouco por fabricante, o ideal é verificar os dados de placa ou catálogo.

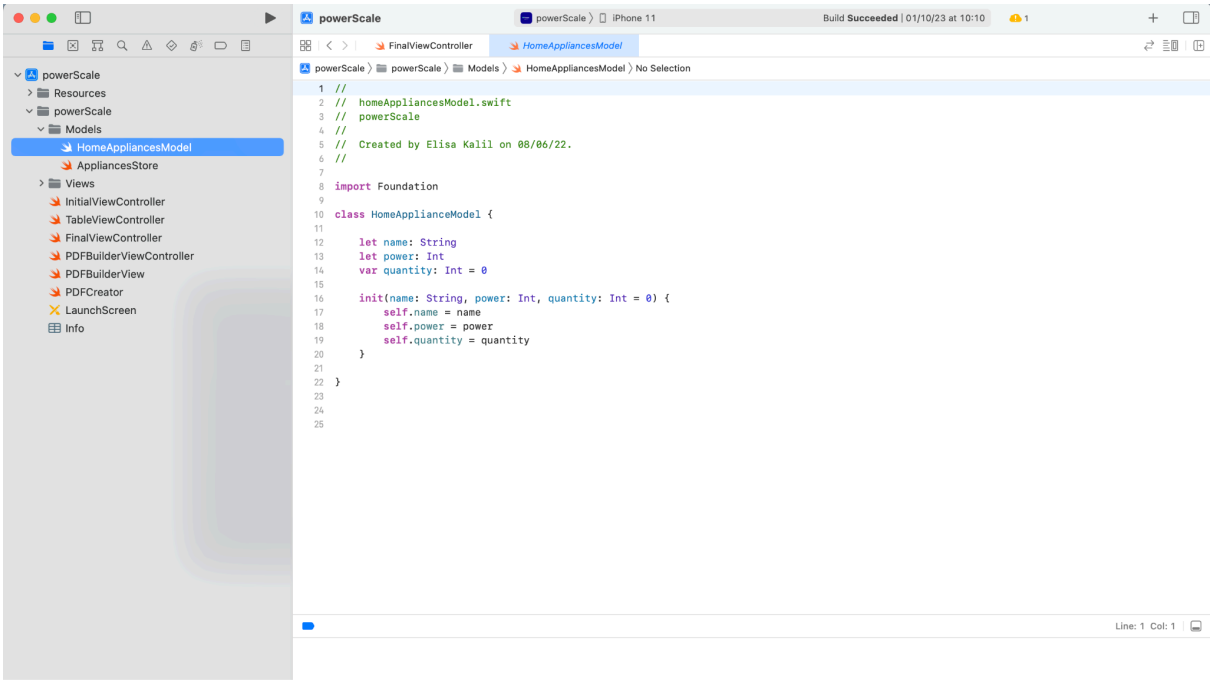
Fonte: CELESC. Padrão de Entrada de Energia Elétrica em Instalações Consumidoras. Santa Catarina: Celesc, 2023. pg. 24

Com base nessas informações, iniciou-se o processo de inclusão dos dados no aplicativo, o que foi feito de maneira manual, frequentemente referida como "mocada". A primeira etapa envolveu a criação de um modelo de dados por meio da declaração de uma classe denominada "Home Appliance Model". Nessa classe, foram definidos os seguintes atributos: nome, potência e quantidade. Esses atributos correspondem, respectivamente, ao nome do aparelho, ao seu consumo de energia e à quantidade selecionada pelo usuário.

É importante observar que, para classes (ao contrário de structs em programação orientada a objetos), é necessário incluir um inicializador, conforme demonstrado na Figura 7 a seguir.

Essa abordagem permitiu que o aplicativo manipulasse e exibisse os dados de maneira eficiente e sem atrasos durante o carregamento dos dados na UI, que se dá de maneira instantânea.

Figura 7 - HomeApplianceModel



Fonte: Autoria própria.

4.3 Desenvolvimento da Lógica de Funcionamento

A primeira versão do aplicativo (Power Scale), desenvolvida como parte da unidade curricular de Projeto Integrador III na nona fase, empregou uma arquitetura inicialmente simples conhecida como MVC (*Model, View e Controller*). Essa abordagem foi recomendada pela Apple como adequada para aplicativos de menor complexidade. Nesse contexto, toda a lógica do aplicativo foi originalmente implementada diretamente na *view controller* da tela de resultados.

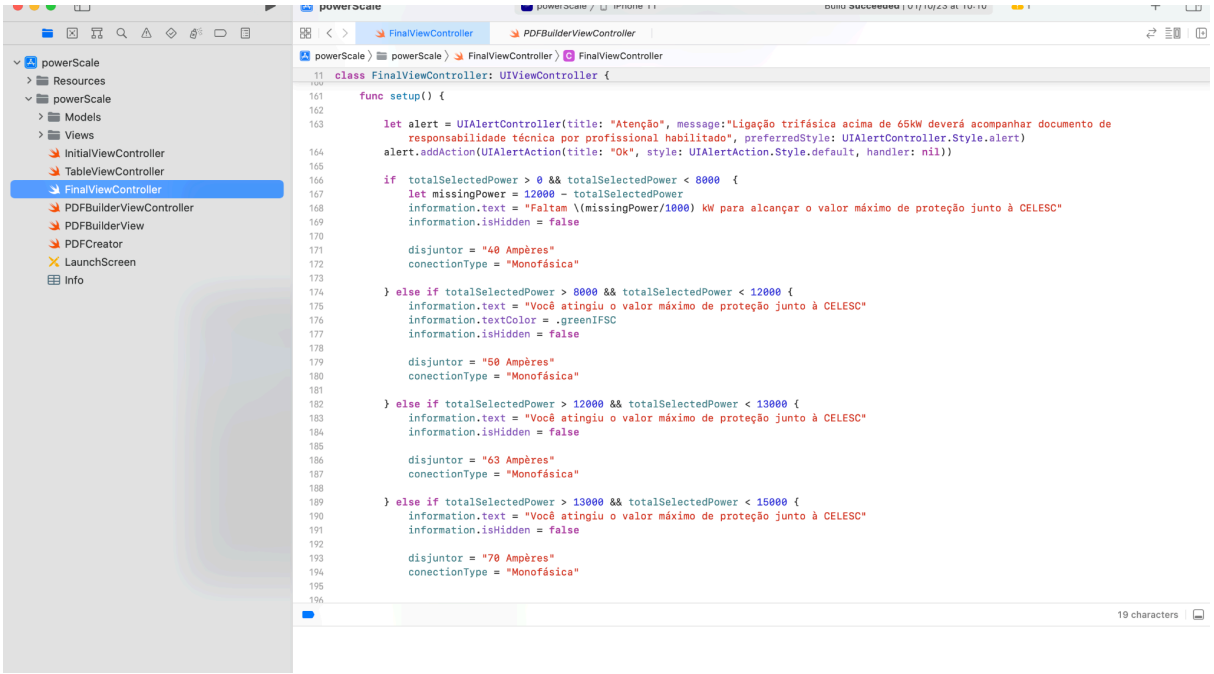
O processo envolveu a criação de uma sequência lógica de condições, geralmente expressa através de estruturas "*if*"- condicional, que avaliava o valor total de potência selecionado pelo usuário, somando as potências de todos os

equipamentos já previamente adicionados também pelo usuário. Com base nessa avaliação, o aplicativo V1.0 determina a mensagem de retorno a ser exibida na tela, gerando uma *string* informativa, por exemplo: "Você atingiu o valor máximo de proteção de acordo com os padrões da CELESC".

Essa abordagem também considerou os limites de potência associados a cada tamanho de disjuntor conforme especificado na Tabela 02 - Padrão de Entrada de Energia Elétrica (página 25) do Manual Simplificado - Padrão de Entrada de Energia Elétrica em Instalações Consumidoras da CELESC. No entanto, posteriormente, essa abordagem foi descartada, pois não era relevante para os usuários conhecerem os detalhes técnicos relacionados à definição do disjuntor pela CELESC, uma vez que o objetivo principal era solicitar um determinado consumo de energia.

Na Figura 8, é apresentada uma representação da estrutura de arquivos e lógica implementada no aplicativo utilizando a arquitetura MVC. Embora essa abordagem tenha gerado um número reduzido de arquivos e limitado a troca de dados, resultou em classes de controle (*ViewControllers*) extensas e complexas.

Figura 8 - Código: FinalViewController

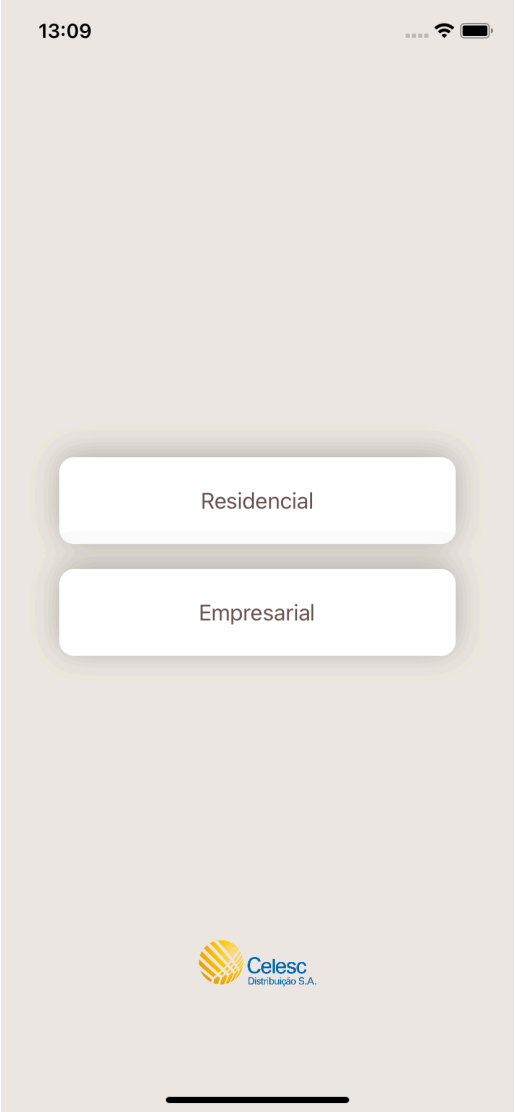


Fonte: Autoria própria.

4.4 Criação da tela de tipo de negócio

Na fase inicial do desenvolvimento, foi considerado crucial permitir que o usuário escolhesse entre dimensionar a potência elétrica para uma residência ou uma empresa. Como resultado, essa escolha foi implementada na primeira tela do Power Scale, onde disponibilizou-se dois botões distintos: "Residencial" e "Empresarial", como ilustrado na Figura 9.

Figura 9 - Tela de Tipo de Residência: *Initial View Controller*

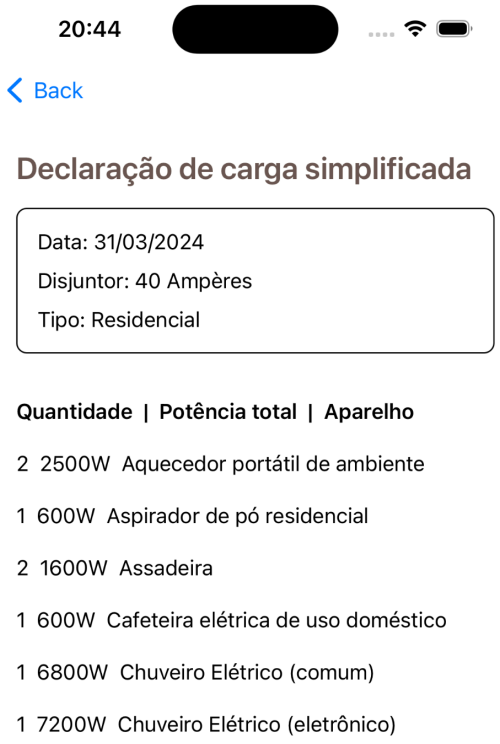


Fonte: Autoria própria.

Quando o usuário faz a seleção entre as opções, essa informação é transmitida, de maneira não expositiva, por todas as telas subsequentes, não interferindo diretamente na lógica do processo. No entanto, essa escolha se torna visível para o usuário ao final do processo, na tela de resultados, como indicado na Figura 10.

A decisão de dimensionar a potência para uma residência ou empresa não é tão relevante para a concessionária, logo, posteriormente optou-se por não apresentá-la, a fim de manter a interface limpa e de fácil compreensão e evitando que o usuário seja sobrecarregado com informações desnecessárias durante o uso do aplicativo V1.0.

Figura 10 - Tela de Tipo de Resultados: *Final View Controller*



Fonte: Autoria própria.

4.5 Criação da tela da tabela de eletrodomésticos

Uma das telas de grande relevância no aplicativo é aquela que abriga a tabela de eletrodomésticos, proporcionando ao usuário a possibilidade de realizar suas seleções. Nessa interface, os dados previamente criados e tratados na seção 4.2, denominada "Inserção dos Dados Mocados", são apresentados em forma de lista por meio de um componente conhecido na linguagem Swift como *UITableView*.

As *Table Views* são elementos de interface de usuário amplamente utilizados no desenvolvimento de aplicativos móveis. São responsáveis por compor listas de itens e são empregadas em diversos aplicativos populares, como o WhatsApp, para listar os contatos, entre outros exemplos.

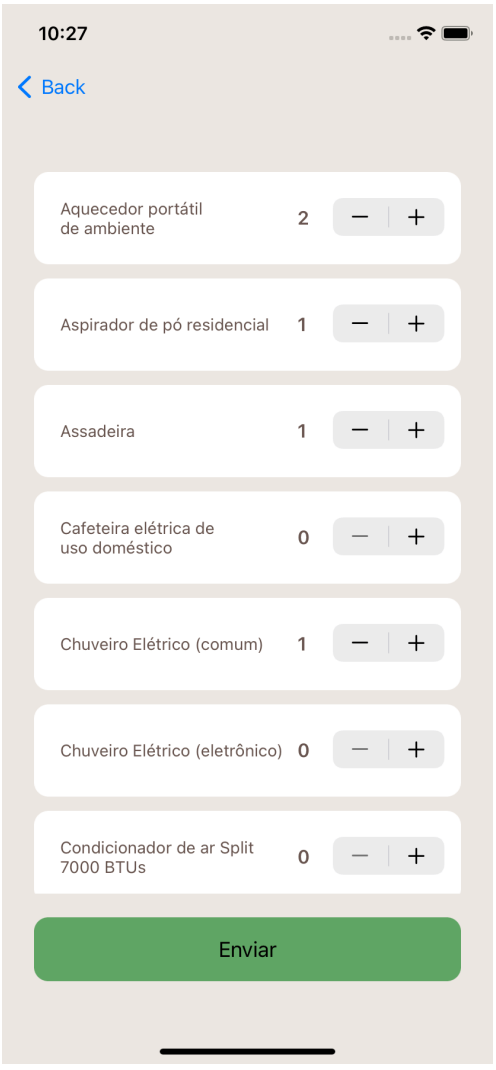
Ao criar uma *Table View*, incorpora-se dentro dela uma célula, denominada *UITableViewCell*, que pode ser personalizada para tornar a apresentação do

conteúdo mais atraente e informativa, incluindo a inserção de ícones, textos e outros elementos visuais. Cada célula é identificada por um rótulo único, e, nas configurações da tabela, é possível chamar o tipo específico de célula desejada por meio dessa identificação.

No contexto deste aplicativo, foi criada uma célula que contém informações relevantes para o usuário, como o nome do eletrodoméstico e a quantidade selecionada. Além disso, implementou-se um componente do tipo *UIStepper*, uma opção padrão da Apple, que inclui dois botões, cada um deles programado para realizar uma ação específica. Um botão foi configurado para adicionar itens, enquanto o outro para subtrair. Além disso, a célula foi estilizada com uma cor de fundo que a diferencia do plano de fundo da tela principal, além de possuir bordas arredondadas, proporcionando uma experiência visual mais agradável ao usuário.

A tela final é mostrada através da Figura 11.

Figura 11 - Tela de Seleção de Equipamentos: *Table View Controller*



Fonte: Autoria própria.

O código completo para implementação desta tela pode ser encontrado no GitHub, através do endereço eletrônico: https://github.com/elisakalil/loadWise-legacy.

4.6 Criação da tela de resultados

Ao concluir o processo, uma vez que o usuário tenha selecionado todos os eletrodomésticos necessários, ele é redirecionado para a tela de resultados. Nessa tela, são apresentadas informações cruciais, incluindo o valor total da carga elétrica selecionada, o tipo de ligação utilizada e se o usuário alcançou o limite máximo de carga disponível.

Para determinar se o usuário alcançou esse limite, o aplicativo faz referência à Tabela 02 - Padrão de Entrada de Energia Elétrica, conforme documentado no manual simplificado da Celesc na página 25. Essa tabela, previamente mencionada no capítulo 4.3 desta seção, é também destacada na Figura 12, exibindo os limites máximos de carga (em kW) para cada tipo de ligação, como monofásica, bifásica e trifásica.

Tabela 2 - Padrão de Entrada de Energia Elétrica

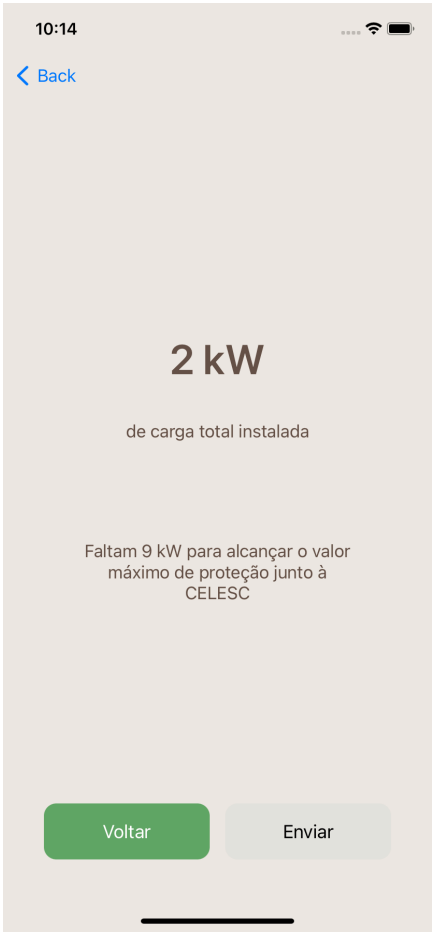
Tipo e Tensão	Carga Total Instalada na UC (kW)	Proteção Geral Disjuntor (A)
Monofásico 220 V	$0 < C \leq 8$	40
	$8 < C \leq 11$	50
	$11 < C \leq 13$	63/60
	$13 < C \leq 15$	70
Monofásico 440/220 V	$0 < C \leq 17$	50
	$17 < C \leq 22$	63/60
	$22 < C \leq 30$	70
	$30 < C \leq 40$	80-90/90
	$40 < C \leq 50$	100
Bifásico 380/220 V	$15 < C \leq 20$	50
	$20 < C \leq 25$	63/60
Trifásico 380/220 V	$25 < C \leq 30$	40
	$30 < C \leq 35$	50
	$35 < C \leq 45$	63/60
	$45 < C \leq 50$	70

	50 < C ≤ 60	80-90/90
	60 < C ≤ 70	100
	70 < C ≤ 75	125

Fonte: CELESC. Padrão de Entrada de Energia Elétrica. Santa Catarina: Celesc, 2023. pg. 25

Com base nesses parâmetros, o Power Scale verifica se a carga total selecionada pelo usuário atinge algum dos limites predefinidos. Se isso ocorrer, uma mensagem é exibida: "Você atingiu o limite junto à Celesc!". Caso contrário, uma mensagem indicando a quantidade que ainda pode ser adicionada sem aumento de faixa e custo é mostrada, conforme ilustrado na Figura 13. É desejável que o consumidor declare corretamente os eletrodomésticos presentes em sua residência à Celesc, atingindo assim a capacidade máxima de potência adequada. Dessa forma, garante-se que toda a demanda necessária estará disponível, prevenindo sobrecargas e evitando a necessidade de aumento do disjuntor para um sistema bifásico ou trifásico em caso de ultrapassagem da faixa de potência inicialmente solicitada.

Figura 12 - *Final View Controller*



Fonte: Autoria própria.

A Figura 13, denominada "*Final View Controller*", exemplifica uma situação em que o usuário escolheu eletrodomésticos que totalizam uma carga de 2 kW. Ao calcular, considera que, para uma conexão monofásica, o usuário pode solicitar até 11 kW. Isso evidencia que ainda faltam aproximadamente 9 kW para atingir o limite estabelecido pela Celesc.

Com o intuito de facilitar a compreensão do leitor, criou-se a Tabela 2, apresentada a seguir, que fornece uma visão clara dos limites aplicáveis a cada tipo de ligação elétrica, juntamente com os valores dos disjuntores correspondentes. Estes dados foram obtidos diretamente da Tabela 03 - Padrão de Entrada de Energia Elétrica da Celesc.

Além disso, a Tabela 3 - Limites Para Cada Tipo de Ligação, abaixo, permite uma análise simplificada das restrições de carga elétrica de acordo com a configuração de ligação, fornecendo informações essenciais para o entendimento do contexto e dos limites de uso de energia elétrica em diferentes cenários.

Tabela 3 - Limites Para Cada Tipo de Ligação

Tipo de Ligação	Carga total (kW)	Proteção Geral (Ampères)
Monofásica	0 - 8	40
Monofásica	8 - 11	50
Monofásica	11 - 13	63
Monofásica	13 - 15	70
Bifásico	15 - 20	50
Bifásico	20 - 25	63
Trifásico	25 - 30	40
Trifásico	30 - 35	50
Trifásico	35 - 45	63
Trifásico	45 - 50	70
Trifásico	50 - 60	90
Trifásico	60 - 70	100
Trifásico	70 < 75	125

Fonte: Autoria própria com base em dados fornecidos pela Celesc, 2020.

4.7 Criação da tela de PDF - exportar dados.

Ao dar um último clique em "Enviar" na tela de resultados, o usuário avança para a fase final da experiência: a tela de exportação de dados, mostrada na Figura 14 e Figura 15. Nesse ponto crucial, é oferecida a oportunidade de revisar todos os

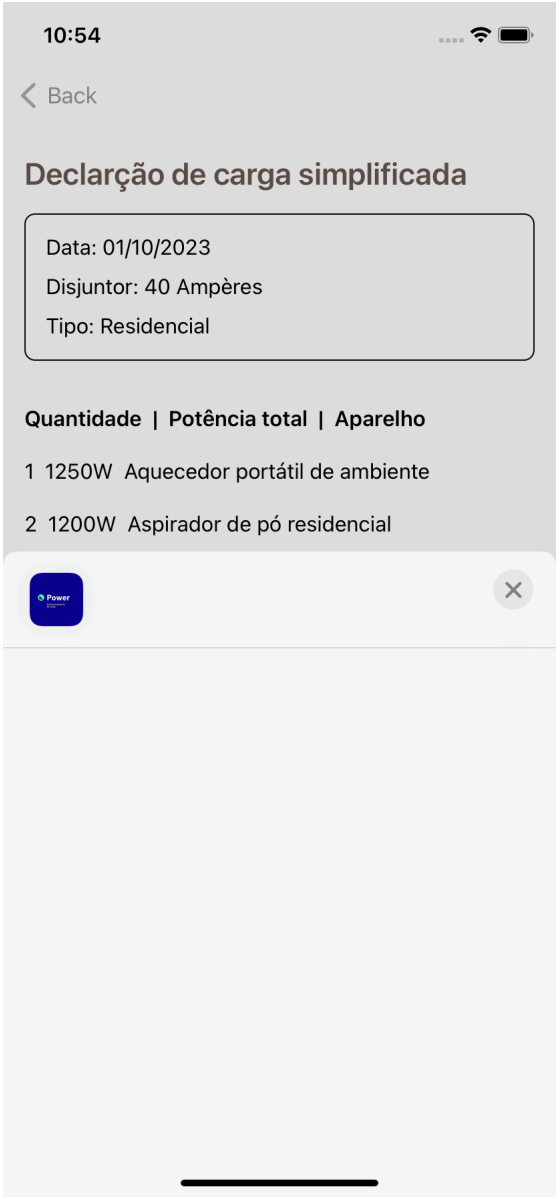
detalhes previamente especificados. Uma lista completa dos equipamentos selecionados é exibida, acompanhada de suas respectivas potências e as quantidades escolhidas para cada um deles. Além disso, todas as opções cruciais, incluindo o tipo de residência (residencial ou empresarial), tipo de ligação (monofásica, bifásica ou trifásica) e os valores dos disjuntores em ampères, também são mostrados.

O propósito final desta etapa é permitir que o usuário revise e confirme os dados antes de prosseguir. A partir daqui, ele têm a capacidade de visualizar o arquivo que será gerado. Esse arquivo pode ser salvo diretamente no dispositivo móvel ou até mesmo exportado de maneira automática para outros meios, como e-mail, WhatsApp, entre outros. Essa funcionalidade simplifica consideravelmente o processo de solicitação de uma nova ligação junto à Celesc, tornando-o mais eficiente e acessível para o usuário.

Figura 13 - Tela de Exportar Dados: *PDF Builder View Controller*

Fonte: Autoria própria.

Figura 14 - *Bottom Sheet* de Exportar Dados: PDF Builder ViewController



Fonte: Autoria própria.

5 TESTE DE USABILIDADE DO APLICATIVO POWER SCALE

As seguintes subseções introduzem o processo de definição dos procedimentos para a realização de testes de usabilidade no aplicativo Power Scale. Estes são cruciais para avaliar a experiência do usuário (Rosemberg, 2010), e cada etapa desse processo é detalhada para garantir resultados confiáveis. Inicia-se com a definição do plano de teste, objetivos e recrutamento de participantes. Em seguida, aborda-se o roteiro dos testes, a preparação de materiais, como cenário e questionário pós-testes, a condução dos testes e a coleta de dados, incluindo observações e *feedbacks* dos participantes. Posteriormente, a análise dos dados e a apresentação dos resultados são explicadas.

5.1 Definição do Plano de Teste

Nesta seção, aborda-se a criação do plano de teste, que é o roteiro que orienta a execução dos testes, discutindo os objetivos buscados e como planeja-se recrutar os participantes ideais.

5.1.1 Objetivos

Com os testes de usabilidade pretende-se:

- Realizar uma análise para mensurar a eficiência das funcionalidades que foram incorporadas ao sistema, com o intuito de garantir que elas desempenhem suas funções de maneira otimizada e efetiva, contribuindo para uma experiência do usuário mais satisfatória e eficiente;

- Examinar a usabilidade do aplicativo móvel, focando na sua capacidade de ser intuitivamente compreendido e utilizado pelos usuários;

- Detectar possíveis falhas ou problemas;

- Dificuldades no fluxo em que o usuário enfrenta durante o uso;

- Coletar sugestões para a incorporação de novas funcionalidades ao sistema, de forma a atender às crescentes demandas e expectativas dos usuários.

5.1.2 Definição e Recrutamento dos Participantes

Para alcançar resultados eficazes por meio dessa abordagem de pesquisa, é essencial conduzi-la com participantes que se assemelham à persona-alvo do produto.

Cada participante deve possuir características e necessidades que estejam alinhadas com o público-alvo pretendido. Durante o processo, os participantes são guiados através de tarefas que simulam de maneira autêntica a utilização do serviço, e essa abordagem meticulosa na seleção dos participantes e na observação minuciosa durante o teste de usabilidade garante insights inestimáveis para o processo de design e a melhoria contínua do produto (Vieira, 2019).

Iniciando, portanto, pelo processo de criação das personas, elas foram delineadas como representações fictícias arquetípicas, destinadas a encapsular grupos específicos de indivíduos. Essas *personas* foram construídas com base nas características do público-alvo da aplicação, englobando pessoas com idades entre 20 e 35 anos, residentes no estado de Santa Catarina e que estão alugando ou adquirindo seu primeiro imóvel, estando dispostas a realizar uma nova ligação junto à Celesc. No entanto, é importante ressaltar que o público real pode apresentar variações significativas, e a persona serve como uma representação conceitual que

orienta os testes de usabilidade da aplicação, mas não necessariamente captura todas as nuances e complexidades do público real.

Nas Tabelas 3 e 4, encontram-se delineadas as *personas* idealizadas para a execução dos testes, fornecendo informações específicas e abrangentes.

Tabela 4 - Persona 1

Persona 1	
Nome	Beatriz
Idade	23 anos
Gênero	Feminino
Estado Civil	Solteira
Ocupação	Publicitária
Sobre	Beatriz é uma publicitária curitibana. Aos 23 anos, ela acabou de se mudar para morar sozinha em Florianópolis e será a primeira locatária de um imóvel recém construído, que não possui medidor. Para solicitar uma nova ligação junto à CELESC, Beatriz acessou o site e verificou que deveria entregar junto aos documentos uma planilha com o levantamento de carga do imóvel. Por não ter conhecimento prévio, Beatriz pesquisou na internet e verificou que deveria elencar os aparelhos que consomem energia. Sendo assim, ela listou o chuveiro, geladeira, fogão e forno, sem especificar seus valores de potência (que devem ser em Watts) por não saber diferenciar corrente, tensão e potência nas etiquetas dos aparelhos e nem os demais aparelhos que consomem valores de energia consideráveis como air fryer's. Beatriz não pode usufruir do máximo fornecimento de energia para a secção de cabo utilizado.
Arquétipo	O Governante
Personalidade	Arquiteto
Objetivos implícitos	Solicitar fornecimento de energia de maneira otimizada.

Objetivo explícito	Solicitar fornecimento de energia.
Dores	Ter de lidar com informações fora do seu domínio. Ter a solicitação de nova ligação junto à CELESC negada, o que resultará em uma nova espera de 7 dias.
Argumentos	Com a utilização do *app*, o usuário garante a solicitação de energia otimizada (para cada bitola, há um valor máximo de energia que pode ser solicitado sem alteração de custo)

Fonte: Elaborado pelo autor, 2023.

Tabela 5 - Persona 2

Persona 2	
Nome	João Victor
Idade	33 anos
Gênero	Masculino
Estado Civil	Solteiro
Ocupação	Microempreendedor
Sobre	João Victor é blumenauense e decidiu investir no ramo de pet shops e irá abrir sua primeira franquia, a PetLovers. Alugou um pequeno galpão que não possui medidor e solicitou corretamente por formulário online, uma nova ligação junto à CELESC. Por mais que todas as cargas tenham sido elencadas corretamente, João não foi aprovado pois sua carga total ultrapassou o limite de 60 kW estipulado pela Celesc, e a requisição deveria ter sido acompanhada de laudo técnico de um profissional da área da elétrica. Por não saber dessa informação, João perdeu 7 dias e terá que refazer o processo.
Arquétipo	O Criador
Personalidade	Protagonista
Objetivos implícitos	Solicitar fornecimento de energia de maneira otimizada.
Objetivo explícito	Solicitar fornecimento de energia.

Dores	Ter a solicitação de nova ligação junto à CELESC negada, o que resultará em uma nova espera de 7 dias.
Argumentos	Com a utilização do app, o usuário é informado quando atinge o valor de 65 kW, e deverá anexar laudo técnico.

Fonte: Elaborado pelo autor, 2023.

Além das *personas* destinadas aos usuários finais, também foram criadas *personas* com o propósito de participarem da avaliação heurística do aplicativo.

De acordo com Rosemberg (2010, slide 4, pág. 17), a avaliação heurística envolve a submissão da interface de um aplicativo a especialistas em usabilidade, que seguem um conjunto predefinido de princípios de usabilidade bem estabelecidos e deve envolver, no mínimo, três avaliadores especializados, mas não requer a presença do usuário final.

Durante esse processo, esses especialistas aderem a diretrizes conhecidas como heurísticas, que orientam a avaliação de todos os elementos da interface do usuário. Segundo Oliveira e Savoine (2011) as avaliações heurísticas contém 10 princípios fundamentais de usabilidade, sendo elas, de forma resumida:

a. Clareza na Comunicação do Sistema:

- Visibilidade e Status do Sistema: O sistema mantém o usuário constantemente informado sobre as ações em andamento de maneira compreensível e dentro de um tempo razoável.
- Compatibilidade com a Linguagem do Usuário: O sistema utiliza uma linguagem familiar aos usuários, evitando jargões técnicos e termos específicos.
- Controle e Liberdade do Usuário: Oferece saídas de emergência claramente identificadas, permitindo que os usuários saiam facilmente de situações inesperadas.
- Consistência e Padrões: Garante que ações ou situações semelhantes sejam representadas de maneira uniforme, facilitando a compreensão do usuário.

b. Ajudando os Usuários em Caso de Erros:

- Identificação e Resolução de Erros: Apresenta mensagens de erro em linguagem simples e orienta os usuários sobre como corrigi-los.
- Prevenção de Erros: Procura evitar a ocorrência de erros sempre que possível.

c. Melhorando a Eficiência e a Experiência do Usuário:

- Reconhecimento: Tornar objetos, ações e opções na interface sempre visíveis e facilmente identificáveis.
- Flexibilidade e Eficiência de Uso: Oferece opções que otimizam a experiência de usuários mais experientes.
- Design Estético e Minimalista: Evita o uso de informações ou elementos irrelevantes na interface.
- Ajuda e Documentação Acessíveis: Fornece informações facilmente localizáveis e orienta os usuários por meio de instruções simples.

Na Tabela 5, é apresentado a persona desenvolvida especialmente para a execução da avaliação heurística, oferecendo informações detalhadas e abrangentes sobre esses perfis.

Tabela 6 - Persona Para Avaliação Heurística

Persona 3	
Nome	Caroline
Idade	29 anos
Gênero	Feminino
Estado Civil	Solteira
Ocupação	Desenvolvedor(a) Android
Arquétipo	O Governante
Personalidade	Arquiteto
Objetivos	Avaliar a aplicação quanto aos bons princípios de usabilidade.

Fonte: Elaborado pelo autor, 2023.

5.1.3 Definição do Local dos Testes

Os testes de usabilidade e as avaliações heurísticas, de acordo com Belisiario (2023), podem ser realizados tanto online como presencialmente.

Na abordagem presencial, o teste de usabilidade ocorre em um ambiente controlado, permitindo o registro e anotação detalhada das ações dos usuários. Um facilitador experiente acompanha de perto o participante, incentivando a verbalização de problemas e desconfortos encontrados durante o teste, o que proporciona *insights* valiosos para a melhoria do design do produto.

Desta forma, optou-se por realizar os testes de usabilidade com os usuários finais de maneira presencial, escolhido por sua acessibilidade tanto para os usuários quanto para os moderadores. Essa abordagem permitiu capturar de perto as

reações, críticas, desconfortos e dúvidas dos usuários durante sua primeira interação com o aplicativo, proporcionando uma compreensão mais profunda de sua experiência.

Por outro lado, optou-se por realizar a avaliação heurística de forma online, aproveitando a facilidade de compartilhamento de tela. Essa abordagem facilitou discussões aprofundadas sobre melhorias de código e otimização da interface do usuário, contribuindo para um desempenho mais eficiente do aplicativo.

Um resumo com as demais informações sobre o local dos testes pode ser encontrado nas Tabelas 6 e 7 abaixo.

Tabela 7 - Definição de Local de Testes de Usabilidade

Item	**Descrição**
Local	Qualquer local com rede Wifi disponível
Equipamentos	Macbook e iPhone
Softwares necessários	Xcode
Documentos	https://forms.gle/NZXoxCrk4kHkJzRu8 https://forms.gle/g5pHNWrtx4usdHYR6
Moderador(es)	Elisa Kalil (autora do trabalho)
Assistente(s)	-

Fonte: Elaborado pelo autor, 2023.

Tabela 8 - Definição de Local de Avaliação Heurística

Item	**Descrição**
Local	Online via plataforma Google Meet
Equipamentos	Macbook
Softwares necessários	Xcode
Documentos	https://forms.gle/2F2DfCL7253sWYpb8
Moderador(es)	Elisa Kalil (autora do trabalho)
Assistente(s)	-

Fonte: Elaborado pelo autor, 2023.

5.2 Roteiro dos Testes

Para garantir a condução adequada dos testes de usabilidade, elaborou-se um roteiro que proporciona uma introdução à problemática que o Power Scale visa

resolver. O usuário é orientado a imaginar que precisa solicitar uma nova ligação de energia à Celesc, levando em consideração os eletrodomésticos já existentes em sua residência ou comércio.

Em seguida, o usuário é apresentado à versão de teste do aplicativo V1.0, que está instalada no iPhone do moderador. Ele é encorajado a usá-lo sem orientações prévias, permitindo que perguntas relacionadas à usabilidade surjam naturalmente. O moderador, atento ao comportamento do usuário, registra observações cruciais, como o número de perguntas feitas pelo usuário e o tempo gasto em cada tela, em uma ficha de observação.

Após a conclusão do processo de interação com o aplicativo V1.0, o usuário recebe um formulário para preenchimento, proporcionando uma oportunidade valiosa para coletar feedbacks e percepções essenciais sobre a experiência do usuário.

Na avaliação heurística, seguiu-se a mesma abordagem da ambientação da problemática ao participante. No entanto, tanto o usuário quanto o moderador realizaram uma revisão detalhada, analisando cada tela minuciosamente. Durante essa análise, buscando identificar comportamentos ideais relacionados a diversos aspectos, como padrões de cores, disposição de botões, sistemas de navegação e muito mais.

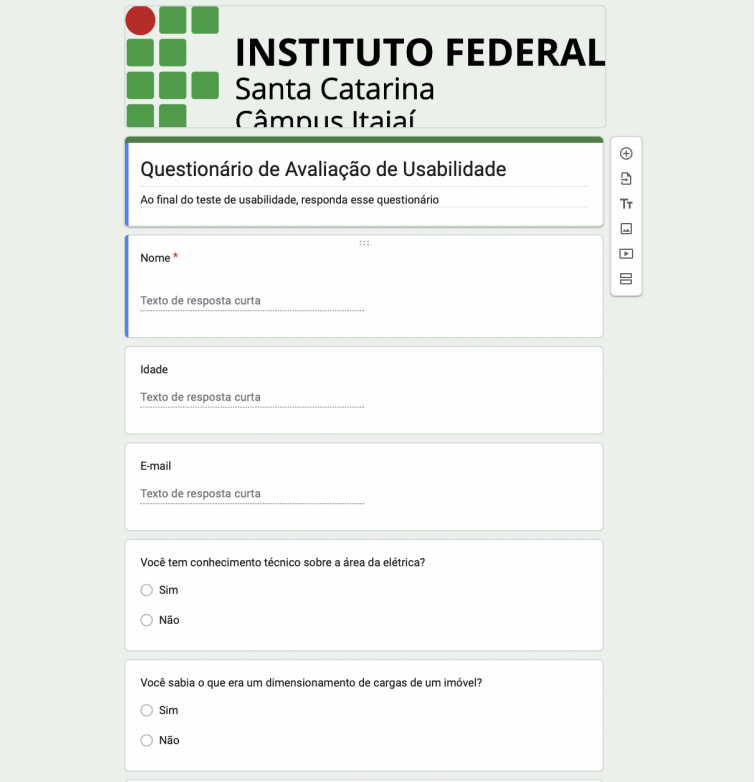
5.3 Preparação dos Materiais Para Testes

Dois questionários foram preparados para os participantes, visando obter informações sobre suas experiências.

O primeiro questionário, apresentado na Figura 16 abaixo, é direcionado aos usuários que participaram do teste de usabilidade. O segundo questionário, exibido na Figura 17 abaixo, é destinado aos participantes da avaliação heurística. Esses questionários são ferramentas essenciais para coletar dados sobre a usabilidade e a eficácia do aplicativo, suas respostas ajudarão a orientar nossos esforços de desenvolvimento futuros, estes podem ser acessados na íntegra, para fins de visualização, através dos endereços eletrônicos:

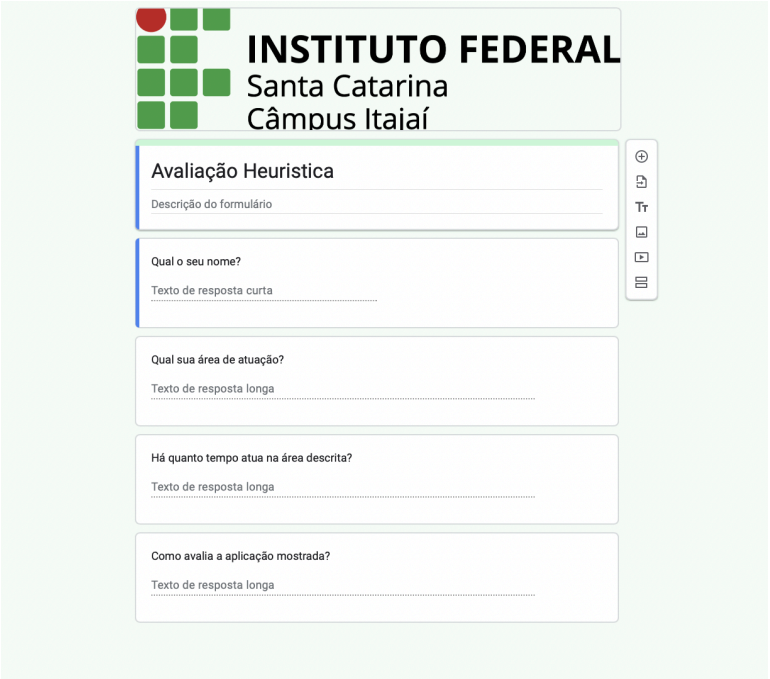
a. Teste de Usabilidade: https://forms.gle/NZXoxCrk4kHkJzRu8
b. Avaliação Heurística: https://forms.gle/2F2DfCL7253sWYpb8

Figura 15 - Questionário de Teste de Usabilidade



Fonte: Autoria própria.

Figura 16 - Avaliação Heurística



Fonte: Autoria própria.

Foi desenvolvido um terceiro questionário com o propósito de auxiliar o moderador na avaliação do comportamento do usuário durante o teste de

usabilidade. Este questionário foi preenchido em tempo real, à medida que o participante interagia com o aplicativo e expressava suas reações.

É possível visualizar o questionário na íntegra na Figura 18 abaixo, e também através do seguinte endereço eletrônico:

c. Observação Comportamental: https://forms.gle/g5pHNWrtx4usdHYR6

Figura 17 - Questionário de Observação Comportamental

Fonte: Autoria própria.

5.3.1 Questões

As questões pertinentes a cada questionário podem ser visualizadas nas Tabelas 8, 9 e 10 abaixo.

Tabela 9 - Descrição das Questões de Avaliação Heurística

Questão	**Descrição**
1	Qual seu nome?
2	Qual sua área de atuação?
3	Há quanto tempo atua na área descrita?
4	Como avalia a aplicação mostrada?

Fonte: Elaborado pelo autor, 2023.

Tabela 10 - Descrição das Questões de Teste de Usabilidade

Questão	**Descrição**
1	Nome
2	Idade
3	E-mail
4	Você tem conhecimento técnico sobre a área da elétrica?
5	Você sabia o que era um dimensionamento de cargas de um imóvel?
6	Você entende o que é o dimensionamento de cargas de um imóvel após a utilização do aplicativo?
7	Sentiu dificuldade em identificar os aparelhos da sua residência/empresa na lista?
8	Sentiu falta de mais informações sobre os aparelhos ou o processo? Sim, liste abaixo:
9	Você acha que o aplicativo te ajudou a otimizar a quantidade de energia que você irá solicitar?
10	Indicaria o *app* à um amigo que precisasse realizar o levantamento de carga da sua residência?
11	Comentário geral

Fonte: Elaborado pelo autor, 2023.

Tabela 11 - Descrição das Questões de Observação Comportamental

Questão	**Descrição**
1	Nome do usuário
2	Idade
3	O usuário se mostrou tenso durante o uso?
4	O usuário pediu informações?
5	Com base na pergunta anterior, caso a resposta tenha sido "sim", quais informações foram pedidas?
6	O usuário retornou às telas anteriores para checar informações?
7	O usuário demorou quanto tempo para fazer a simulação?

Fonte: Elaborado pelo autor, 2023.

5.3.2 Cenário dos Testes

Para garantir a fidelidade dos testes de usabilidade, era crucial que o usuário selecionado pudesse interagir com o aplicativo em um dispositivo físico real. Portanto, o aplicativo precisou ser instalado em um iPhone com um sistema operacional compatível, mesmo que ainda não estivesse disponível na App Store para download público.

A Apple oferece uma opção que permite compilar o código diretamente em um dispositivo físico conectado ao computador através do Xcode (projeto). Após selecionar a opção de desenvolvedor nas configurações do dispositivo, o aplicativo pode ser instalado no celular por um período limitado. Essa abordagem foi adotada para instalar o projeto do aplicativo em um iPhone 11, que foi posteriormente utilizado nos testes com o usuário, proporcionando uma experiência mais autêntica e realista.

5.4 Condução dos Testes

Durante a sessão de testes, adotou-se uma abordagem de moderação ativa, onde o moderador participa proativamente com os participantes. Isso significa que o moderador deve sempre estar pronto para fornecer orientações, esclarecer dúvidas e incentivar os participantes a compartilharem suas experiências e pensamentos enquanto interagem com o aplicativo.

Essa interação próxima entre o moderador e os participantes é fundamental para obter *insights* valiosos e garantir que possamos capturar feedbacks detalhados e observações críticas durante todo o processo de teste.

5.5 Coleta dos Dados

Para coletar os dados dos testes, foram utilizados questionários administrados por meio da plataforma Google Forms. Isso permitiu organizar e reunir as respostas dos participantes de maneira eficaz.

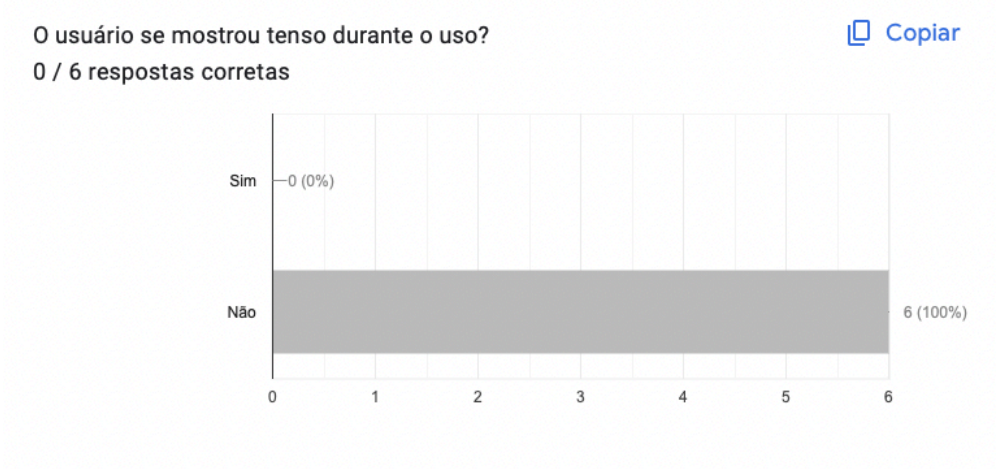
5.6 Análise dos Dados

A análise dos testes foi realizada com o objetivo de atingir os objetivos listados na seção 5.1.1, que incluem avaliar a eficiência e eficácia das funcionalidades implementadas, bem como a facilidade de uso do aplicativo. Além disso, buscou-se identificar erros e obter sugestões para aprimorar ainda mais a experiência do usuário. Para alcançar esses objetivos, foram empregadas técnicas e métodos específicos de análise de dados que serão mostradas na seção seguinte, 5.8, garantindo uma compreensão completa dos resultados dos testes.

5.7 Apresentação dos Dados

Como previsto, várias informações valiosas foram obtidas a partir das respostas dos testes e da ficha de observação. A Figura 19, que se encontra na ficha de observação do moderador, revela que nenhum dos usuários demonstrou tensão ao utilizar o aplicativo. Isso indica que a abordagem de navegação tipo "passo a passo", discutida na seção 2.1.7 Padrões de Navegação, mostrou-se eficaz ao orientar os usuários durante todo o processo, proporcionando uma experiência tranquila e sem contratempos.

Figura 18 - O Usuário se mostrou tenso?

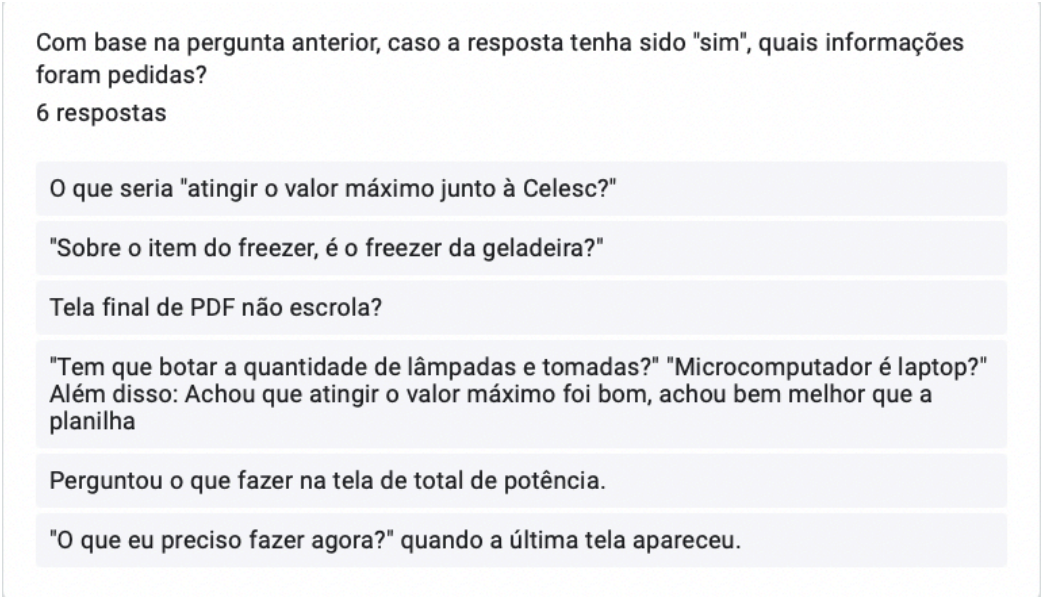


Fonte: Autoria própria

Analisando a Figura 20, fica evidente que todos os usuários apresentaram dúvidas e fizeram perguntas em relação a diferentes aspectos do aplicativo. Com base nessas perguntas, identificou-se a necessidade de introduzir novas funcionalidades e realizar melhorias, que são detalhadamente abordadas na Seção 6, Resultados.

Três das perguntas se referem à tela final, onde os resultados são exibidos, juntamente com a informação sobre se o usuário atingiu o limite máximo de carga permitido pela Celesc. Isso levou a concluir que essa tela precisa de aprimoramentos. Além disso, a segunda pergunta da lista, "Sobre o item do freezer, é o freezer da geladeira?", e a terceira pergunta, "Microcomputador é *laptop*?", indicaram a necessidade de melhorar a nomenclatura dos dispositivos listados na tabela, que atualmente é formal e gera confusão. Estudou-se também a viabilidade de adicionar ícones em cada célula para representar os aparelhos de forma mais visual e clara.

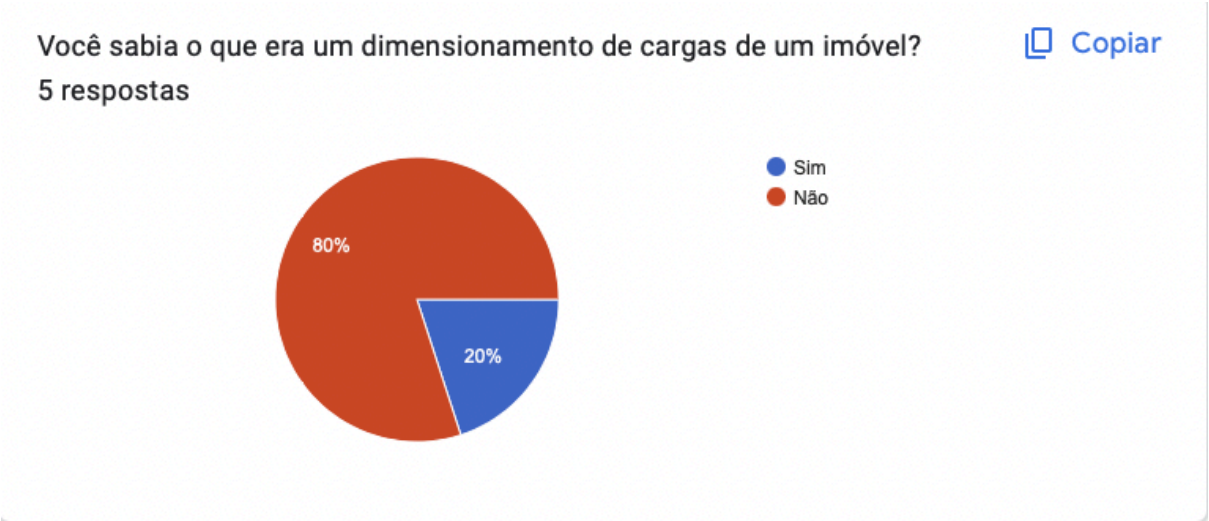
Figura 19 - O Usuário se mostrou tenso?



Fonte: Autoria própria

Nas Figuras 21 e 22 a seguir, um fato interessante se destaca. Na Figura 20, é evidente que a maioria dos participantes não possuía conhecimento sobre o conceito de dimensionamento de carga, tampouco sabia como realizá-lo de maneira adequada. No entanto, após a utilização do aplicativo, todos os participantes afirmaram ter adquirido compreensão sobre o dimensionamento de carga em residências ou imóveis.

Figura 20 - Entende o Que é Dimensionamento de Carga?



Fonte: Autoria própria

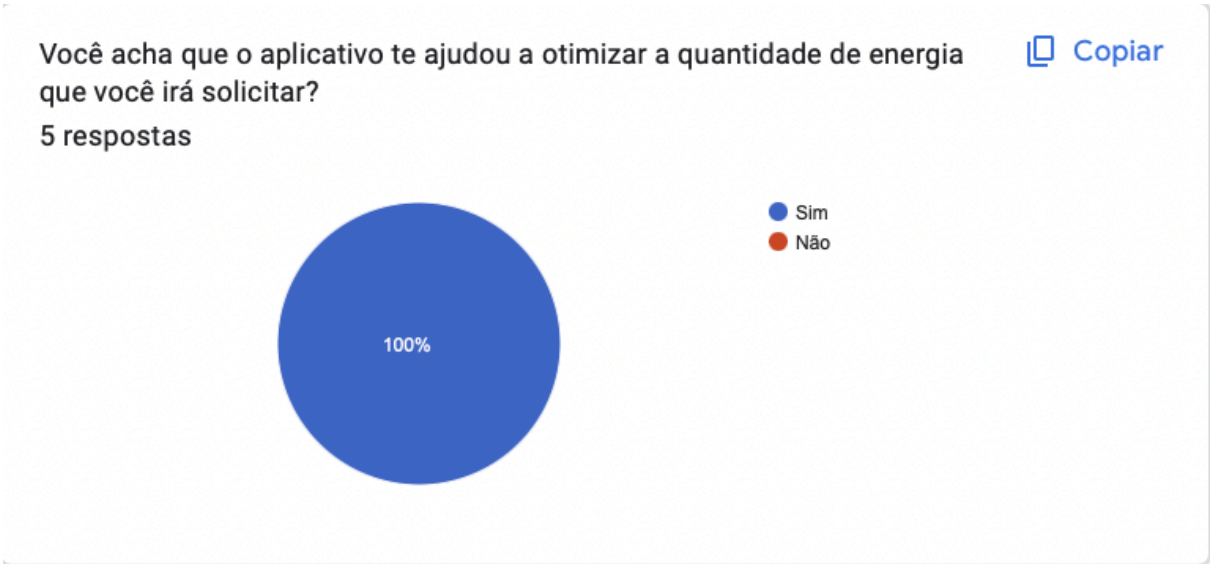
Figura 21 - Entende o Que é Dimensionamento Após o Uso do *App*?

Fonte: Autoria própria

Por fim, conforme ilustrado nas Figuras 23, 24 e 25 subsequentes, o aplicativo teve, de modo geral, um impacto positivo em todos os participantes. Eles compartilharam a percepção de que o aplicativo não apenas simplificou o processo de dimensionamento de carga, mas também manifestaram prontidão em recomendar entusiasticamente o aplicativo a amigos e conhecidos.

Esse feedback positivo valida com êxito a eficácia e a utilidade do aplicativo na tarefa em questão.

Figura 22 - Ajudou a Otimizar?

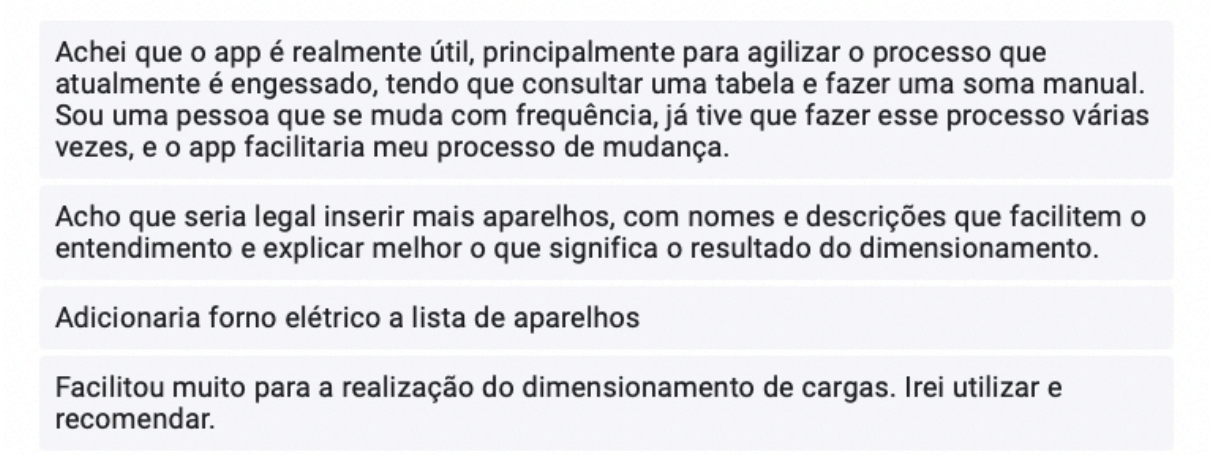


Fonte: Autoria própria

Figura 23 - Indicaria a um Amigo?

Fonte: Autoria própria

Figura 24 - Comentário Geral



Fonte: Autoria própria

6 RESULTADOS

Nesta seção, o objetivo é apresentar a nova versão do aplicativo, o Load Wise, bem como fornecer uma explicação detalhada de cada item listado para refatoração, que se originou dos testes de usabilidade e da avaliação heurística conduzidos tendo como base o aplicativo Power Scale. Nas subseções, cada ponto será abordado em profundidade.

6.1 Aprimoramentos elencados para nova versão Load Wise

Ao avaliar os resultados dos testes, conforme apresentado na seção 5.8 - Apresentação dos Dados, identificaram-se várias áreas do aplicativo Power Scale que necessitam de melhorias. Esses aprimoramentos são organizados e detalhados

na Tabela 11, que será discutida de forma mais abrangente na subseção 6.1.3 - Novas Funcionalidades do Aplicativo Load Wise.

Tabela 12 - Novas Funcionalidades

Item	Local	Melhoria
1	Tela Onboarding	Criar uma tela de introdução informativa para orientar o usuário no contexto.
2	Tela de Tipo de Residência	Reformular as opções de escolha do usuário, substituindo "Residencial" e "Empresarial" por "Área Rural" e "Área Urbana".
3	Tela de Seleção de Equipamentos	Fornecer ao usuário informações em tempo real, como a potência total selecionada e o tipo de ligação em que ele se encaixa, à medida que ele seleciona os equipamentos.
4	Tela de Seleção de Equipamentos	Exibir ícones representando visualmente cada equipamento, juntamente com os valores de potência, a quantidade selecionada e um nome mais informal e facilmente associado a cada dispositivo.
5	Tela de Resultados	Refatorar a mensagem final exibida ao usuário e enriquecê-la com informações adicionais que capacitem o usuário a acessar e explorar o assunto por conta própria.
6	Tela de Resultados	Realizar cálculo de demanda automaticamente
7	Todas as telas	Refatorar paleta de cores

Fonte: Elaborado pelo autor, 2023.

6.1.1 Reestruturação dos Código para MVVM

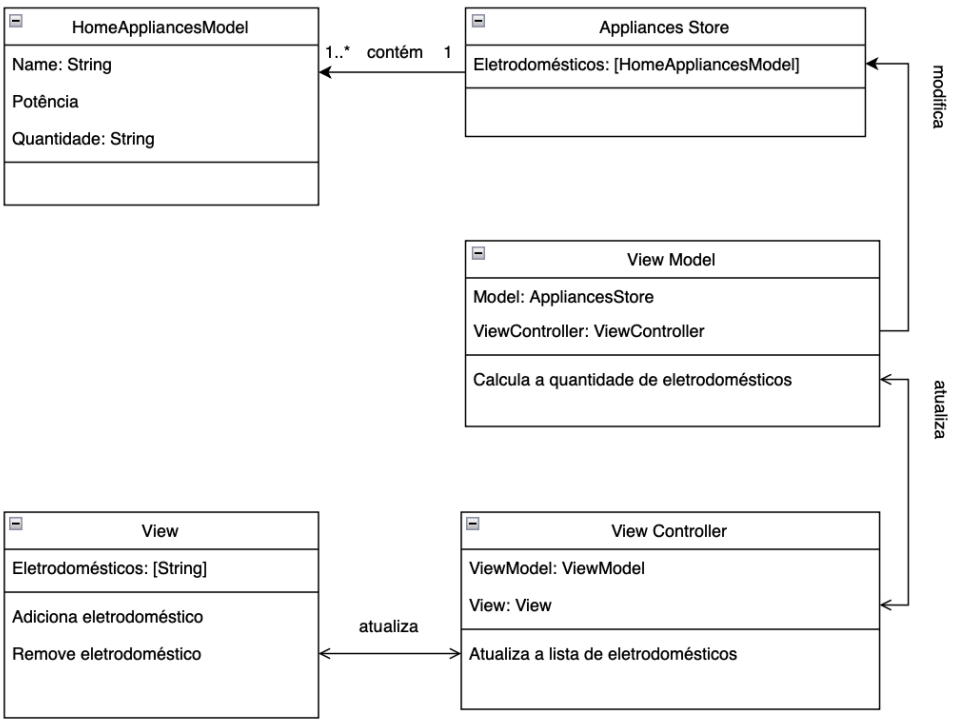
Conforme descrito no capítulo 2.1.6.2, MVVM (*Model-View-ViewModel*) é um padrão arquitetural de software que se destaca em cenários que demandam manutenção contínua, correção de bugs e refatorações, pois ele estrutura o código em classes bem delimitadas, cada qual com funções específicas. Dessa forma, tomou-se a decisão de adotar esse padrão para a nova versão do aplicativo.

No padrão MVVM (Model-View-ViewModel), a modelagem dos dados passa a ter um novo nível de abstração: a View Model. Nesta classe, são implementadas as funções lógicas, como o cálculo da quantidade total de eletrodomésticos, por exemplo. Desta forma, diferentemente do projeto Power Scale, onde a View Controller calculava, modificava o modelo e atualizava a view, no projeto Load Wise, a View Model calcula e modifica o modelo, enquanto a view controller fica encarregada apenas de atualizar a view e seus diferentes estados, criando um código mais organizado, como mostra a Figura 26.

Para preservar o código anterior e mantê-lo acessível para futuras referências, o projeto existente foi mantido e renomeado no GitHub como

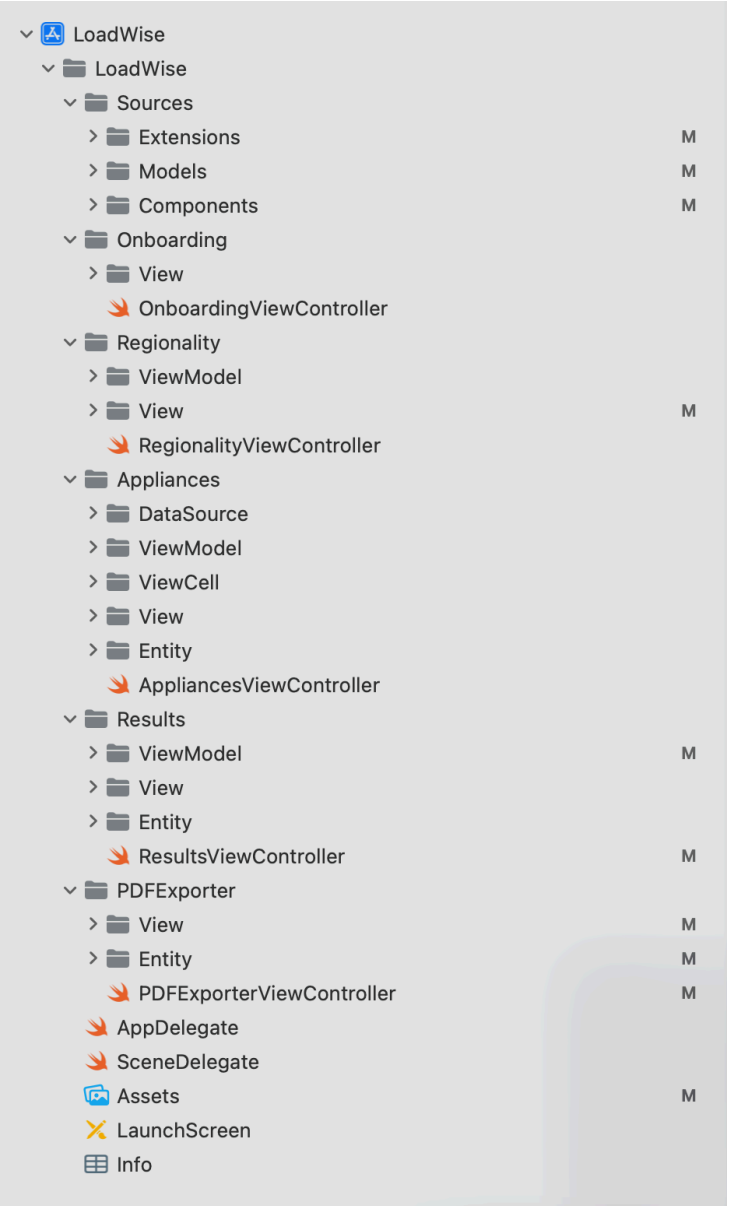
"PowerScale - Código Legado." Simultaneamente, para a nova iteração do aplicativo, foi criado um projeto totalmente novo intitulado "Load Wise", desenvolvido integralmente em conformidade com a arquitetura MVVM, empregando protocolos para aprimorar a organização do código, conforme apresentado no diagrama de classes da Figura 26. A disposição aprimorada dos arquivos, que se tornou substancialmente mais extensa, é demonstrada na Figura 27 apresentada a seguir.

Figura 25 - Refatoração diagrama de classes para MVVM



Fonte: Autoria própria

Figura 26 - Refatoração para MVVM

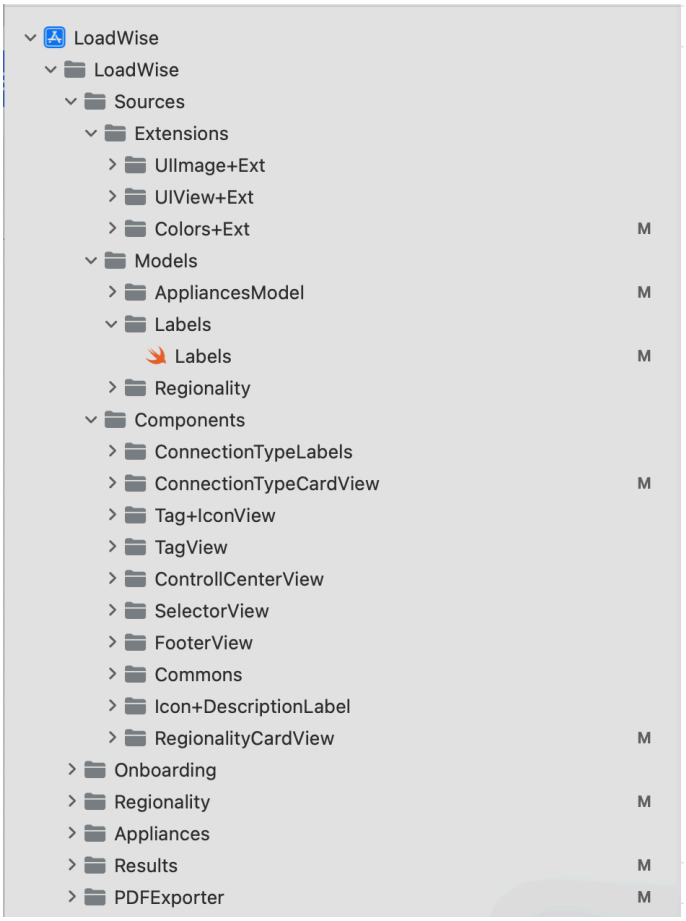


Fonte: Autoria própria

Além da refatoração das classes, o aplicativo passou por um processo de componentização, que consiste em transformar cada elemento da tela em um componente. Por exemplo, os botões "Continuar" de cada tela tem o mesmo design e costumam ser programados diretamente no arquivo da tela em que era inserido, ou seja, tínhamos 4 códigos de botão repetidos no projeto, um em cada tela. Ao componentizar, uma classe Button foi criada para o código do botão, e foi chamada em todas as telas que o utilizavam.

Desta forma, para cada item da tela, foi criado um componente exclusivo. Isso possibilita a inserção desses componentes na tela atual e, quando necessário, a reutilização em outras telas por meio de protocolos, conforme exemplificado na Figura 28 apresentada a seguir.

Figura 27 - Componentização



Fonte: Autoria própria

6.1.2 Reestruturação da UI

Na segunda versão do aplicativo, o Load Wise, houve uma reavaliação significativa da interação com os usuários que possuem limitações visuais, como daltonismo ou deficiência visual. A paleta de cores anterior, notoriamente composta por tons muito claros e com pouca distinção entre eles, como pode ser observado na Figura 29, podia representar um desafio para a visualização desses usuários. Portanto, uma nova paleta de cores foi cuidadosamente redesenhada para melhorar a acessibilidade e tornar a experiência de uso mais inclusiva.

Figura 28: Power Scale - Paleta de Cores

Fonte: Autoria própria

Seguindo as Diretrizes de Acessibilidade para Conteúdo Web (WCAG) 2.0 de 2008, foram adotados três níveis de conformidade: nível A, nível AA e o mais elevado, nível AAA. Para assegurar a conformidade com o nível AAA, várias diretrizes foram aplicadas, incluindo a seleção da nova paleta de cores que proporciona alto contraste.

A nova interface adota um fundo preto com texto em branco, o que naturalmente resulta em alto contraste (nível AAA). Além disso, o design é flat e incorpora ícones e cores adicionais que são facilmente distinguíveis, como apresentado na paleta de cores da Figura 30.

Para tal, foram utilizadas ferramentas online, como a Calculadora de Contraste desenvolvida pela Universidade Federal do Rio Grande do Sul (disponível no endereço eletrônico: https://www.ufrgs.br/calculacontraste), para determinar as melhores combinações de cores. Esse esforço visa tornar a interface mais inclusiva, garantindo a acessibilidade para todos os usuários.

Figura 29: Load Wise - Paleta de Cores

Fonte: Autoria própria

Para criar o estilo de design *flat*, caracterizado por elementos visuais minimalistas e sem sombras, resultando em uma aparência limpa e simplificada,

foram desenvolvidos ícones a partir de desenhos vetorizados dos equipamentos disponíveis no aplicativo, como mostra a Figura 31. Isso foi realizado usando a ferramenta de design online Canva, acessível em: https://www.canva.com.

Figura 30: Load Wise - Ícones

Fonte: Autoria própria

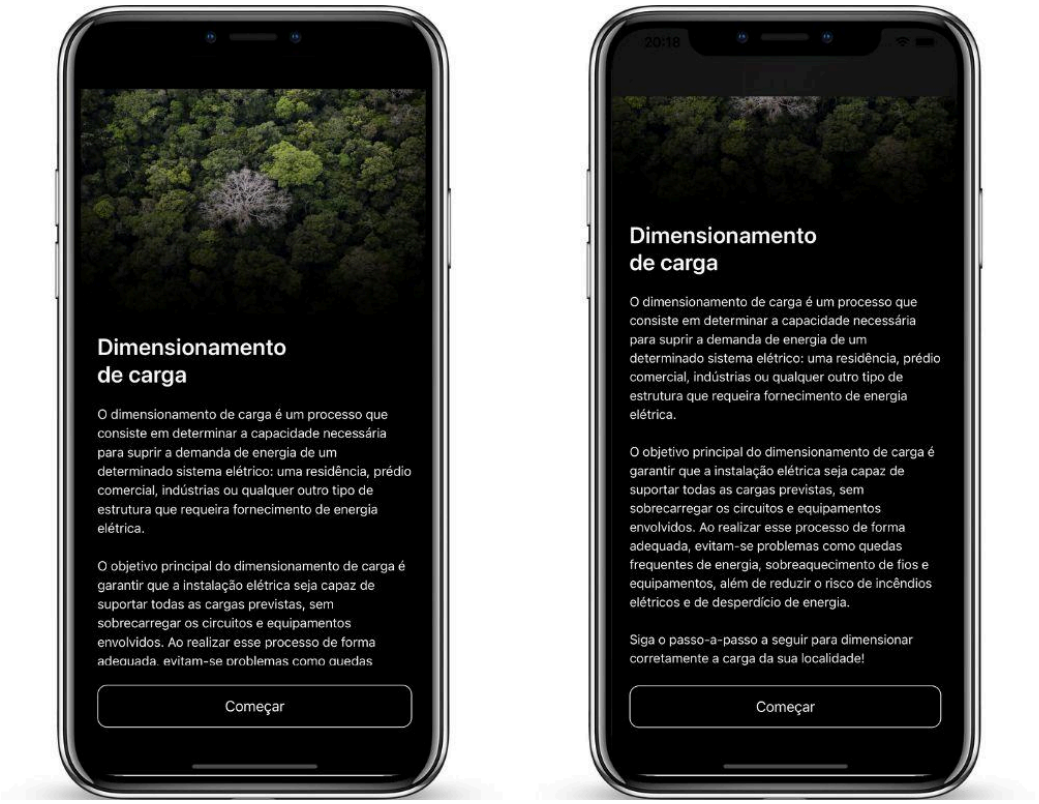
6.1.3 Novas Funcionalidades do Aplicativo

Nas subseções, serão abordados detalhadamente cada nova funcionalidade que faz parte da segunda versão do aplicativo.

6.1.3.1 Implementação de Tela de *Onboarding*

A maioria dos usuários que participaram dos testes demonstrou desconhecimento em relação ao dimensionamento de carga. Após usar o aplicativo, eles relataram uma melhor compreensão desse conceito. No entanto, compreender não apenas o que é o dimensionamento de carga, mas também o seu impacto no meio ambiente e na rede elétrica é essencial. Para abordar essa necessidade, foi criada uma tela de onboarding como a primeira etapa do aplicativo. Nessa tela, um texto informativo foi desenvolvido, abrangendo os tópicos mencionados. Você pode conferir o conteúdo na Figura 32 abaixo:

Figura 31: Load Wise - *Onboarding*



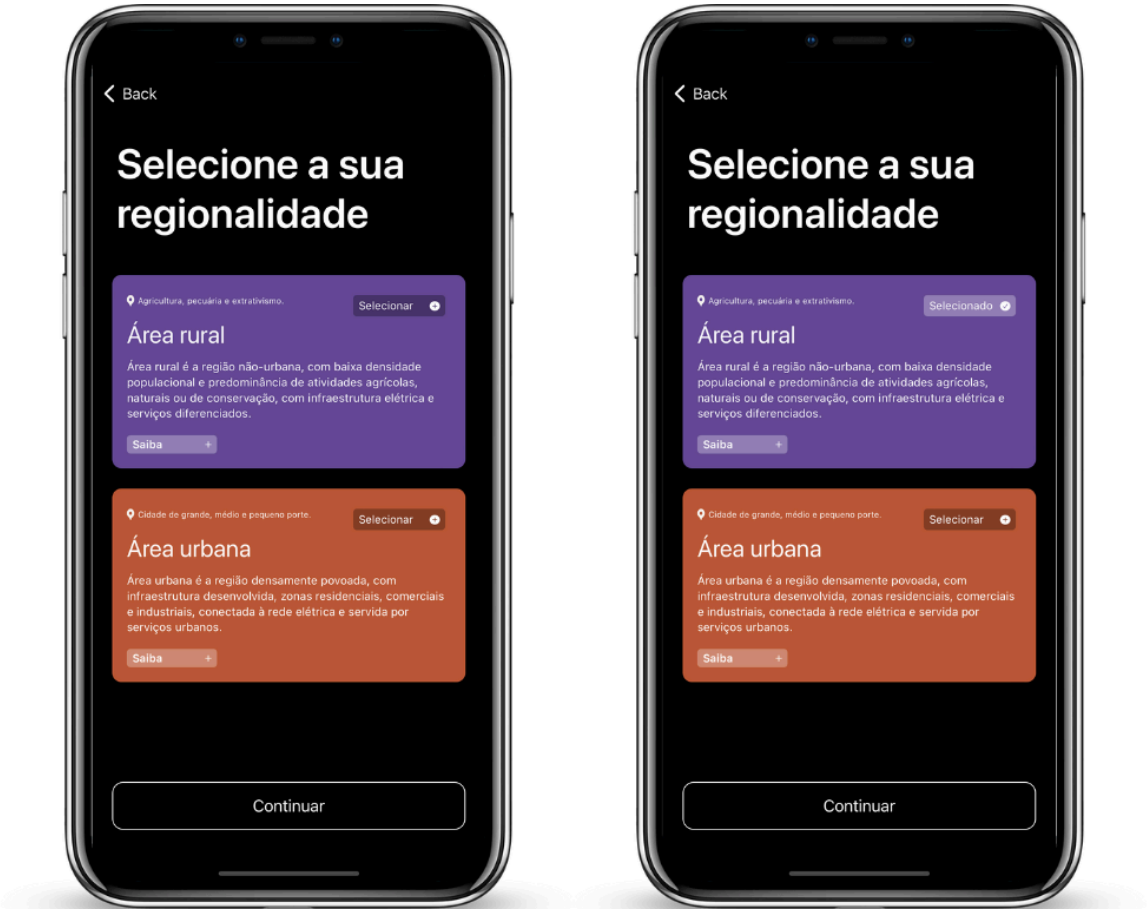
Fonte: Autoria própria

Isso visa fornecer aos usuários uma compreensão mais abrangente do dimensionamento de carga e sua relevância. Além disso, essa abordagem contribui para a conscientização sobre a importância das escolhas de consumo de energia.

6.1.3.2 Implementação de Tela de Regionalidade

A diferenciação entre imóveis comerciais e residenciais é de pouca relevância para a Celesc. No entanto, as ligações de energia podem variar significativamente entre áreas rurais e urbanas. Portanto, a tela correspondente a essa seleção foi aprimorada para apresentar as opções "Área Rural" e "Área Urbana" ao usuário, como ilustrado na Figura 33. Essa melhoria visa fornecer ao usuário uma escolha mais precisa, alinhada com as características de sua localização, garantindo que o dimensionamento de carga seja adequado às condições reais. Isso contribui para uma melhor experiência do usuário e resultados mais precisos.

Figura 32: Load Wise - Tela de Regionalidade



Fonte: Autoria própria

Cada *card* agora apresenta uma breve descrição explicativa sobre como se define uma área rural e uma área urbana, e também inclui um botão intitulado "Saiba +". Ao clicar neste botão, o usuário é direcionado, por meio da implementação de deep link no código, para a página de solicitação de nova ligação junto à Celesc. A Figura 34 ilustra essa melhoria na interface, proporcionando ao usuário fácil acesso a informações adicionais e recursos relevantes para o processo de ligação elétrica.

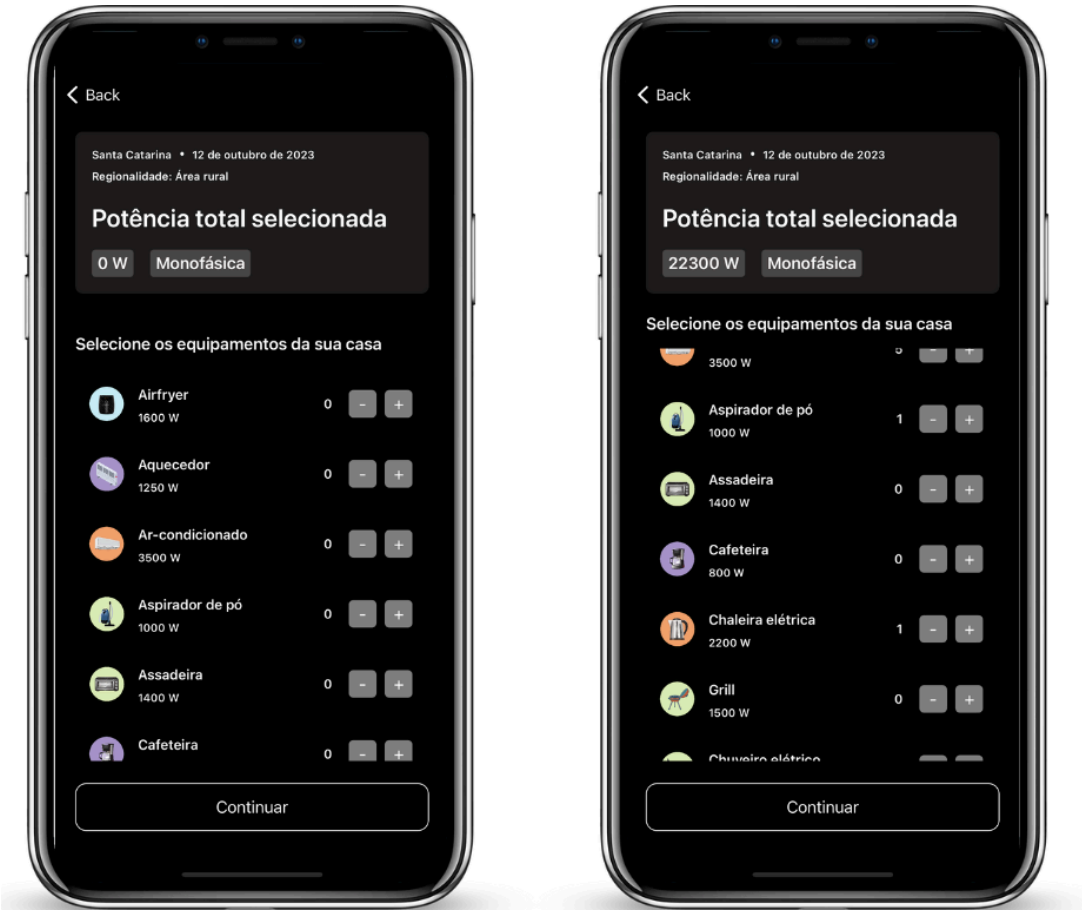
Figura 33: Load Wise - Deep Link Nova Ligação

Fonte: Autoria própria

6.1.3.3 Implementação do Controle de Dados e Refatoração da Table View

Conforme discutido no item 3 da Tabela 11 - Novas Funcionalidades, o Load Wiseagora fornece informações em tempo real para auxiliar o usuário na seleção de equipamentos. Essas informações incluem a potência total já selecionada e o tipo de ligação ao qual o usuário se enquadra. Para melhorar a experiência do usuário e permitir um controle mais eficiente das escolhas, foi incorporado um *card* informativo no cabeçalho da tabela de equipamentos. Esse *card* permanece visível mesmo quando o usuário rola a tela, facilitando o acesso constante a essas informações essenciais. A Figura 35 apresenta essa adição, ilustrando como o aplicativo tornou as informações mais acessíveis e em tempo real durante a seleção dos equipamentos.

Figura 34: Load Wise - Tela de Equipamentos



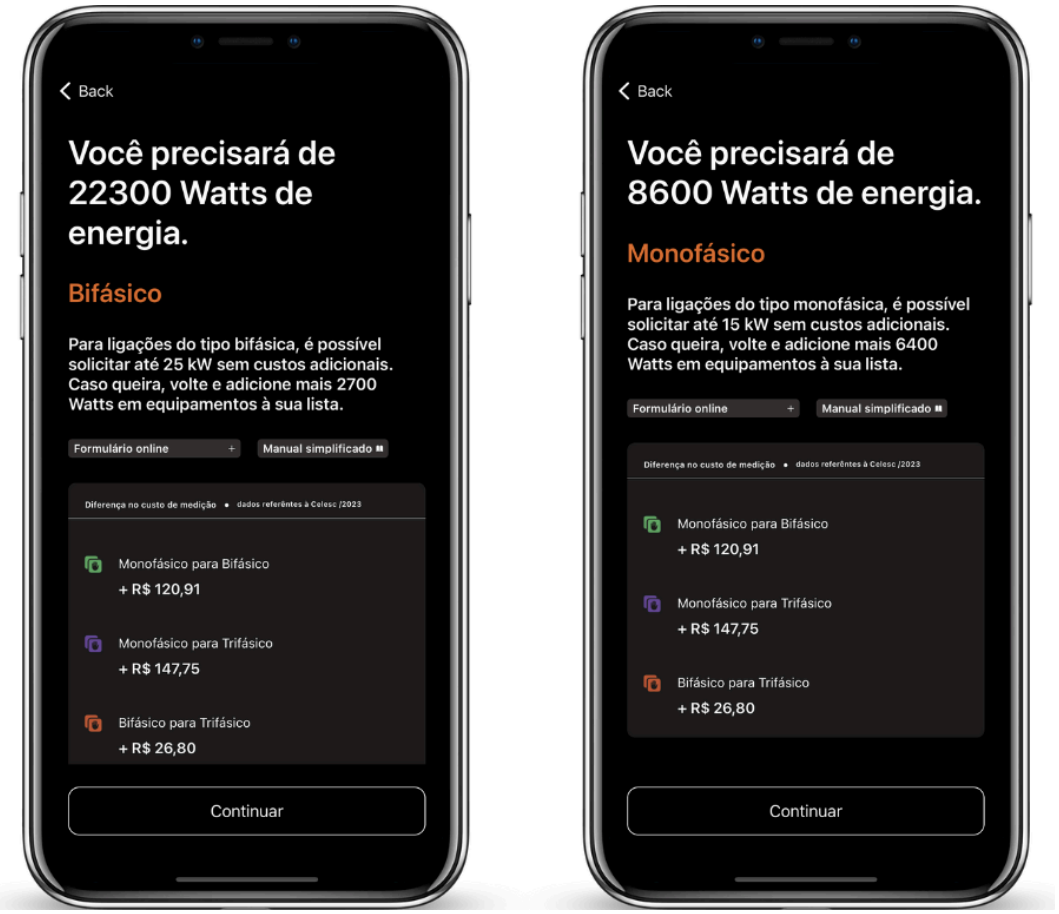
Fonte: Autoria própria

Pode-se notar também a adaptação feita da célula, que agora mostra um ícone personalizado ilustrando cada equipamento, informa também a potência de cada um, e os nomes foram repensado para serem mais acessíveis e informais, se encaixando melhor no dia-a-dia do usuário.

6.1.3.5 Refatoração da Tela de Resultados

Com o intuito de possibilitar ao usuário leigo um controle completo sobre sua solicitação de energia, a tela de resultados passou por uma reformulação significativa. Agora, ela exibe informações que orientam o usuário quanto à quantidade de energia necessária e ao limite disponível sem custos adicionais, como mostrado na Figura 36. Além disso, são disponibilizados botões de acesso rápido para o Manual Simplificado da Celesc e o formulário online, permitindo que os usuários obtenham esclarecimentos para possíveis dúvidas de maneira fácil e conveniente.

Figura 35: Load Wise - Tela de Resultados



Fonte: Autoria própria

6.1.3.6 Adição de Cálculo de Demanda

A inclusão do cálculo de demanda no aplicativo de geração automatizada de dimensionamento de carga de imóveis residenciais é fundamental para garantir a precisão e conformidade com as normas técnicas. Conforme a Normativa I-321.0027, a demanda é definida como a razão entre a demanda máxima num intervalo de tempo especificado e a carga instalada na unidade consumidora (Normativa I-321.0027, 4.21). Em particular, para instalações com carga trifásica instalada entre 25 e 75 kW, é necessário realizar o cálculo da demanda para dimensionar corretamente os componentes elétricos, conforme especificado na mesma normativa (Normativa I-321.0027, 5.3.3.5).

A responsabilidade pelo cálculo recai sobre o projetista ou técnico encarregado, que deve garantir que os componentes atendam às necessidades da instalação. Incorporar essa funcionalidade no aplicativo não apenas simplifica o trabalho dos profissionais envolvidos, mas também aumenta a precisão e segurança do dimensionamento elétrico, em conformidade com as exigências regulamentares. A funcionalidade leva em conta o fator de demanda de 0,2135 estabelecido pela Celesc em 2019 para o grupo A. A partir dessa data, a distribuidora passou a permitir essa escolha livremente.

7 CONCLUSÃO

Na conclusão deste Trabalho de Conclusão de Curso (TCC), destacam-se os resultados significativos alcançados no desenvolvimento e implementação do aplicativo iOS dedicado à automatização do dimensionamento de carga para novas ligações junto à Celesc, nomeado de Load Wise.

Os principais objetivos foram atingidos, com a comprovação da viabilidade do projeto conforme apresentado na seção 6 Resultados; a apresentação de uma visão abrangente sobre aplicativos móveis para iOS, conforme apresentado na seção 2 Referência Teórica; a realização de um estudo comparativo entre modelos arquiteturais, conforme apresentado na seção 2.1.6 Padrões Arquiteturais; a reestruturação do aplicativo seguindo a arquitetura MVVM, conforme apresentado na seção 4 Desenvolvimento do Aplicativo; e sobre o layout, através de testes de usabilidade e aplicação de técnicas de aprimoramento UI/UX, conforme apresentado na seção 5 Teste de Usabilidade.

O método hipotético-dedutivo, utilizado nesta pesquisa de natureza aplicada, proporcionou uma análise criteriosa e fundamentada dos resultados.

As descobertas evidenciaram uma melhoria substancial na precisão do dimensionamento de carga, a redução de equívocos por parte de usuários leigos e uma resposta positiva à facilidade de uso do aplicativo. A contribuição original deste trabalho reside na criação de uma ferramenta específica para usuários leigos, preenchendo uma lacuna na automatização desse processo, simplificando conceitos elétricos complexos.

Concluiu-se que o desenvolvimento de aplicativos para automatizar o dimensionamento de carga é não apenas viável, mas também benéfico, especialmente para usuários não familiarizados com conceitos elétricos. As implicações práticas incluem uma simplificação considerável do processo para as concessionárias de energia elétrica, com a redução de erros e um potencial aumento da eficiência nas novas ligações.

Apesar do desafio inicial de simplificar conceitos complexos de eletricidade para usuários leigos, esta pesquisa se alinha e destaca-se em relação a estudos anteriores, focando na experiência do usuário leigo e oferecendo uma solução prática para esse público. Assim, este TCC contribui não apenas para o avanço do conhecimento na área, mas também para a aplicação prática e eficiente do dimensionamento de carga em novas ligações de energia elétrica.

REFERÊNCIAS

ALMACINHA, José António. **Introdução ao Conceito de Normatização em Geral e sua Importância na Engenharia**. Porto, Portugal: Inegi, 2018. 21 p.

APPLE INC. **Apple Developer**. Disponível em: https://developer.apple.com. Acesso em: 23 maio 2023.

APPLE INC. **Documentation - Foundation**. Disponível em: https://developer.apple.com/documentation/foundation. Acesso em: 22 de maio de 2023.

APPLE INC. **Documentation - UIKit.** Disponível em: https://developer.apple.com/documentation/uikit. Acesso em: 12 de maio de 2023.

Apple Inc. (2022). **WWDC 2022 - Vídeos - Apple Developer**. Recuperado de https://developer.apple.com/videos/play/wwdc2022/10001/. Acesso em: 28 maio 2023.

BELISIARIO, G. (2023). **UX Baseado em Fatos: Teste de Usabilidade**. UX Collective. Recuperado de https://brasil.uxdesign.cc/ux-baseado-em-fatos-teste-de-usabilidade-ce8dc58841a3

BORGES, Hudson; HORA, Andre; VALENTE, Marco Tulio. **Understanding the Factors That Impact the Popularity of GitHub Repositories**. In: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), 02-07 October 2016. IEEE, 2016. DOI: 10.1109/ICSME.2016.31. Disponível em: https://ieeexplore.ieee.org/abstract/document/7816479. Acesso em: 11 de setembro de 2023.

CADU. **Entendendo o Pattern Model-View-ViewModel**, 2010. Disponível em: https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411. Acesso em: 2 set. 2023.

CARVALHO, Lucas. **Clean Swift e uma Pitada de Arquitetura em iOS**, 2021. Disponível em: https://medium.com/@lucasscarvalho/clean-swift-e-uma-pitada-de-arquitetura-em-ios-6d97ed55b61d. Acesso em: 2 de setembro de 2023.

CELESC. Ligação Nova. 2023. **Como solicitar o serviço por formulário online?**. Disponível em: https://www.celesc.com.br/ligacao-nova. Acesso em: 6 de maio de 2023.

CELESC. **Padrão de Entrada de Energia Elétrica em Instalações Consumidoras**. Santa Catarina: Celesc, 2020. 26 p. Disponível em: https://www.celesc.com.br/arquivos/normas-tecnicas/padrao-entrada/manual-simplificado-padrao-entrada-energia-eletrica.pdf. Acesso em: 5 maio 2023.

CHAMBERS, John M. **Object-Oriented Programming, Functional Programming, and R. Statistical Science**, Volume 29 (Número 2). Disponível em: https://projecteuclid.org/journals/statistical-science/volume-29/issue-2/Object-Oriented-Programming-Functional-Programming-and-R/10.1214/13-STS452.full?tab=ArticleLink. Acesso em: 10 set. 2023.

SAGA. **Entenda o que são desenhos vetoriasi.** Disponível em: https://blog.saga.art.br/entenda-o-que-sao-desenhos-vetoriais/#:~:text=De%20maneira%20resumida%2C%20os%20desenhos,matemáticas%20específicas%20para%20serem%20construídos.. Acesso em: 31 mar. 2024.

FELIPE, Alan. **Mockando dados para criação de interface**. 4 de jan. de 2021. Disponível em: https://dev.to/alanflpns/mockando-dados-para-criacao-de-interface-4i7e. Acesso em: 23 de setembro de 2023.

GIL, Antônio Carlos. **Métodos e Técnicas de Pesquisa Social**. 6a. ed. São Paulo: [s.n.], 2008. Disponível em: https://ayanrafael.files.wordpress.com/2011/08/gil-a-c-mc3a9todos-e-tc3a9cnicas-de-pesquisa-social.pdf. Acesso em: 15 de setembro de 2023.

KELLY, Maurice; NOZZI, Joshua. **Mastering Xcode**. 2ª edição. Peachpit Press, 2014. 89 páginas. Acesso em: 11 set. 2023.

MAGALHÃES, Ícaro. **Um estudo comparativo entre padrões arquitetônicos para o desenvolvimento de aplicativos para a plataforma iOS.** 2022. 80 p. Monografia (Bacharelado em Ciência da Computação do Centro de Informática, Universidade Federal da Paraíba). Disponível em: <https://repositorio.ufpb.br/jspui/bitstream/123456789/15651/1/ILM12122018.pdf>. Acesso em: 15 de maio de 2023.

MARTINS, Ana Isabel et al. **Avaliação de Usabilidade: Uma Revisão Sistemática da Literatura**. Revista Ibérica de Sistemas e Tecnologias de Informação, [ano de publicação não informado], p. 47. Disponível em: https://www.researchgate.net/profile/Nelson-Rocha-4/publication/254558973_Avaliacao_de_Usabilidade_Uma_Revisao_Sistematica_da_Literatura/links/0deec52651165142ed000000/Avaliacao-de-Usabilidade-Uma-Revisao-Sistematica-da-Literatura.pdf. Acesso em: 11 set 2023.

MARTINS, Michell Viudes Garcia; KOMATSU, Paulo Takeo. **Aplicativo para cálculo de demandas segundo as normas de distribuição da CEB e dimensionamentos básicos.** 2016. 94 f. TCC - Universidade de Brasília. Disponível em: https://bdm.unb.br/handle/10483/13444. Acesso em: 16 de setembro de 2023.

MATTOS, Mauro M. et al. **Ambiente de programação para a introdução da lógica de programação**, 2019. Acesso em: 11 set. 2023. Disponível em: https://sol.sbc.org.br/index.php/wie/article/view/13297/13150.

MELLO, Flávio Luís. **Desenvolvimento de Aplicativos Nativos Android e iOS para o Restaurante Universitário da UFRJ.** [S.l.], 2018. Disponível em: <https://pantheon.ufrj.br/handle/11422/17206>. Acesso em: 27 de maio de 2023.

MIRO. **Wireframes**. Disponível em: https://miro.com/pt/wireframe/o-que-e-wireframe/. Acesso em: 31 mar. 2024.

NEVES JÚNIOR, Fernando José das. **Desenvolvimento de aplicativo computacional para cálculo de demanda**, 2016. Acesso em: 25 de maio de 2023

Normativa I-321.0027. Fornecimento de Energia Elétrica em Tensão Secundária de Distribuição. 2019. Número da norma. Disponível em: https://www.celesc.com.br/arquivos/normas-tecnicas/padrao-entrada/N3210001-Fornecimento-Energia-Eletrica-Tensao-Secundaria.pdf. Acesso em: 26 fev. 2024.

RAUSCH, F. **Modern iOS Navigation Patterns**. Disponível em: https://frankrausch.com/ios-navigation. Acesso em: 04 jan. 2024.

RODRIGUES, K. **O que é UX/UI Design? Um guia completo para iniciantes.** 24 fev. 2023. Disponível em: https://blog.cubos.academy/ux-ui-design-guia-completo/#. Acesso em: 31 mar. 2024.

SILVEIRA, Denise Tolfo; CÓRDOVA, Fernanda Peixoto. **Métodos de pesquisa**. Porto Alegre: Editora da UFRGS, 2009. pp. 33-44.

APÊNDICE A - Código Tela de Onboarding

```
import Foundation
import UIKit
class OnboardingView: UIView {

    private let headerImageView: UIImageView = {
        let headerImage = UIImageView()
        let image = UIImage(named: "florest-gradient")
        headerImage.image = image
        headerImage.translatesAutoresizingMaskIntoConstraints = false
        return headerImage
    }()

    private let headerTitleLabel: UILabel = {
        let label = UILabel()
        label.textAlignment = .left
        label.lineBreakMode = .byWordWrapping
        label.numberOfLines = 0
        label.textColor = .white
        label.font = UIFont.boldSystemFont(ofSize: 28)
        label.translatesAutoresizingMaskIntoConstraints = false
        return label
    }()

    private let scrollView: UIScrollView = {
        let scrollView = UIScrollView()
        scrollView.translatesAutoresizingMaskIntoConstraints = false
        scrollView.showsVerticalScrollIndicator = true
        scrollView.indicatorStyle = .white
        return scrollView
    }()

    private let contentView: UIView = {
        let content = UIView()
        content.backgroundColor = .black
        content.translatesAutoresizingMaskIntoConstraints = false
        return content
    }()
```

```swift
private let descriptionLabel: UILabel = {
    let label = UILabel()
    label.textAlignment = .left
    label.lineBreakMode = .byWordWrapping
    label.numberOfLines = 0
    label.textColor = .white
    label.translatesAutoresizingMaskIntoConstraints = false
    return label
}()

private lazy var footer: FooterView = {
    let footer = FooterView()
    footer.button.addTarget(self, action: #selector(buttonAction), for: .touchUpInside)
    footer.translatesAutoresizingMaskIntoConstraints = false
    return footer
}()

// MARK: INITIALIZERS

weak var delegate: OnboardingViewDelegate?

init() {
    super.init(frame: .zero)
    setup()
}

required init?(coder aDecoder: NSCoder) {
    fatalError("init(coder:) has not been implemented")
}

@objc public func buttonAction() {
    delegate?.buttonAction()
}

// MARK: PRIVATE FUNCTIONS

private func setup() {
    backgroundColor = .black

    buildViewHierarchy()
    addConstraints()
}

private func buildViewHierarchy() {
    addSubview(scrollView)
    addSubview(footer)

    scrollView.addSubview(contentView)

    contentView.addSubview(headerImageView)
    contentView.addSubview(headerTitleLabel)
    contentView.addSubview(descriptionLabel)
}

private func addConstraints() {
    NSLayoutConstraint.activate([
        scrollView.topAnchor.constraint(equalTo: topAnchor),
```

```
            scrollView.leadingAnchor.constraint(equalTo: leadingAnchor),
            scrollView.trailingAnchor.constraint(equalTo: trailingAnchor),
            scrollView.bottomAnchor.constraint(equalTo: footer.topAnchor),

            headerImageView.heightAnchor.constraint(equalToConstant: Metrics.ImageSize.heigth),

            contentView.topAnchor.constraint(equalTo: scrollView.topAnchor),
            contentView.trailingAnchor.constraint(equalTo: trailingAnchor),
            contentView.leadingAnchor.constraint(equalTo: leadingAnchor),
            contentView.bottomAnchor.constraint(equalTo: scrollView.bottomAnchor),

            headerImageView.topAnchor.constraint(equalTo: contentView.topAnchor),
            headerImageView.leadingAnchor.constraint(equalTo: contentView.leadingAnchor),
            headerImageView.trailingAnchor.constraint(equalTo: contentView.trailingAnchor),

            headerTitleLabel.topAnchor.constraint(equalTo: headerImageView.bottomAnchor, constant:
Metrics.Spacing.medium),
            headerTitleLabel.leadingAnchor.constraint(equalTo: contentView.leadingAnchor, constant:
Metrics.Spacing.medium),
            headerTitleLabel.trailingAnchor.constraint(equalTo: contentView.trailingAnchor, constant:
-Metrics.Spacing.giant),

            descriptionLabel.topAnchor.constraint(equalTo: headerTitleLabel.bottomAnchor, constant:
Metrics.Spacing.medium),
            descriptionLabel.leadingAnchor.constraint(equalTo: contentView.leadingAnchor, constant:
Metrics.Spacing.medium),
            descriptionLabel.trailingAnchor.constraint(equalTo: contentView.trailingAnchor, constant:
-Metrics.Spacing.medium),
            descriptionLabel.bottomAnchor.constraint(equalTo: contentView.bottomAnchor),

            footer.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
            footer.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium),
            footer.bottomAnchor.constraint(equalTo: safeAreaLayoutGuide.bottomAnchor)
        ])
    }

    private func setDescriptionLabelFontIfCould() {
        descriptionLabel.font = UIFont.systemFont(ofSize: Metrics.Font.small)
    }

    private func setDescriptionAttributeString() {
        let paragraphStyle = NSMutableParagraphStyle()
        paragraphStyle.firstLineHeadIndent = 0
        paragraphStyle.lineSpacing = 4

        let attributedText = NSAttributedString(string: .onboarding, attributes:
[NSAttributedString.Key.paragraphStyle: paragraphStyle])

        descriptionLabel.attributedText = attributedText
    }
}
//MARK: OnboardingViewProtocol
extension OnboardingView: OnboardingViewProtocol {
    func updateView() {
        headerTitleLabel.text = .loadBalancing
        footer.updateButton(with: .start)

        setDescriptionAttributeString()
```

```
        setDescriptionLabelFontIfCould()
    }
}

protocol OnboardingViewProtocol where Self: UIView {
    var delegate: OnboardingViewDelegate? { get set }

    func updateView()
}

protocol OnboardingViewDelegate: AnyObject {
    func buttonAction()
}

class OnboardingViewController: UIViewController {

    // MARK: Properties
    let contentView: OnboardingViewProtocol = {
        let view = OnboardingView()
        return view
    }()

    //MARK: Life cycle
    override func viewDidLoad() {
        super.viewDidLoad()

        contentView.delegate = self

        view = contentView
        view.backgroundColor = .black
        navigationController?.navigationBar.barTintColor = .black
        contentView.updateView()
    }
}
//MARK: OnboardingViewDelegate
extension OnboardingViewController: OnboardingViewDelegate {
    func buttonAction() {
        let viewModel = RegionalityViewModel()
        let nextVC = RegionalityViewController(viewModel: viewModel)
        navigationController?.pushViewController(nextVC, animated: true)
    }
}
```

APÊNDICE B - Código Tela de Regionalidade

```
import Foundation
import UIKit
class RegionalityView: UIView {

    private let headerTitleLabel: UILabel = {
        let label = UILabel()
        label.textAlignment = .left
        label.lineBreakMode = .byWordWrapping
        label.numberOfLines = 0
        label.textColor = .white
        label.font = UIFont.boldSystemFont(ofSize: 45)
        label.translatesAutoresizingMaskIntoConstraints = false
        return label
    }()

    private let contentStackView: UIStackView = {
        let content = UIStackView()
        content.backgroundColor = .black
        content.alignment = .center
        content.axis = .vertical
        content.spacing = Metrics.Spacing.small
        content.translatesAutoresizingMaskIntoConstraints = false
        return content
    }()

    private let countrysideCard: CardViewProtocol = {
        let card = CardView()
        card.translatesAutoresizingMaskIntoConstraints = false
        return card
    }()

    private let urbanCard: CardViewProtocol = {
        let card = CardView()
        card.translatesAutoresizingMaskIntoConstraints = false
        return card
    }()
```

```
    private lazy var footer: FooterView = {
        let footer = FooterView()
        footer.button.addTarget(self, action: #selector(buttonAction), for: .touchUpInside)
        footer.translatesAutoresizingMaskIntoConstraints = false
        return footer
    }()

    // MARK: INITIALIZERS

    private var selectedCard: String?
    weak var delegate: RegionalityViewDelegate?

    init() {
        super.init(frame: .zero)
        setup()
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    @objc public func buttonAction() {
        delegate?.buttonAction()
    }

    // MARK: PRIVATE FUNCTIONS

    private func setup() {
        backgroundColor = .black

        buildViewHierarchy()
        addConstraints()
        setupTapGesture()
    }

    private func buildViewHierarchy() {

        addSubview(headerTitleLabel)
        addSubview(footer)
        addSubview(urbanCard)
        addSubview(countrysideCard)
    }

    private func addConstraints() {
        NSLayoutConstraint.activate([
            headerTitleLabel.topAnchor.constraint(equalTo: safeAreaLayoutGuide.topAnchor, constant:
Metrics.Spacing.large),
            headerTitleLabel.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.large),
            headerTitleLabel.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.large),

            countrysideCard.topAnchor.constraint(equalTo: headerTitleLabel.bottomAnchor, constant:
Metrics.Spacing.bigger),
            countrysideCard.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
            countrysideCard.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium),

            urbanCard.topAnchor.constraint(equalTo: countrysideCard.bottomAnchor, constant:
Metrics.Spacing.medium),
            urbanCard.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
```

```
            urbanCard.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium),

            footer.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
            footer.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium),
            footer.bottomAnchor.constraint(equalTo: safeAreaLayoutGuide.bottomAnchor)
        ])
    }

    private func setupTapGesture() {
        let tapCountyGesture = UITapGestureRecognizer(target: self, action: #selector(didSelectCountrySide))
        countrysideCard.addGestureRecognizer(tapCountyGesture)
        countrysideCard.isUserInteractionEnabled = true

        let tapUrbanGesture = UITapGestureRecognizer(target: self, action: #selector(didSelectUrbanSide))
        urbanCard.addGestureRecognizer(tapUrbanGesture)
        urbanCard.isUserInteractionEnabled = true
    }

    @objc private func didSelectCountrySide() {
        countrysideCard.updateSelector(isSelect: true)
        delegate?.handlerCardSelection(with: RegionalityType.countrySide.rawValue)
    }

    @objc private func didSelectUrbanSide() {
        urbanCard.updateSelector(isSelect: true)
        delegate?.handlerCardSelection(with: RegionalityType.urbanSide.rawValue)
    }
}
//MARK: RegionalityViewProtocol
extension RegionalityView: RegionalityViewProtocol {
    func updateSelector(with regionality: RegionalityType, isSelect: Bool) {
        if regionality == RegionalityType.countrySide {
            countrysideCard.updateSelector(isSelect: isSelect)
        }

        if regionality == RegionalityType.urbanSide {
            urbanCard.updateSelector(isSelect: isSelect)
        }
    }

    func updateView() {
        headerTitleLabel.text = .selectYourRegionality

        countrysideCard.updateCardView(color: .lightPurple,
                                       descriptionImage: "florest",
                                       descriptionText: .countySideDescription,
                                       resumeText: .countySideResume,
                                       title: .countySideTitle)
        urbanCard.updateCardView(color: .darkOrange,
                                 descriptionImage: "urbanArea",
                                 descriptionText: .urbanSideDescription,
                                 resumeText: .urbanSideResume,
                                 title: .urbanSideTitle)

        footer.updateButton(with: .proceed)
    }
}
```

```swift
protocol RegionalityViewProtocol where Self: UIView {
    var delegate: RegionalityViewDelegate? { get set }

    func updateView()
    func updateSelector(with regionality: RegionalityType, isSelect: Bool)
}

protocol RegionalityViewDelegate: AnyObject {
    func handlerCardSelection(with selectedRegionality: String)
    func buttonAction()
}

class RegionalityViewController: UIViewController {

    let contentView: RegionalityViewProtocol = {
        RegionalityView()
    }()

    // MARK: PROPERTIES
    private let viewModel: RegionalityViewModelProtocol

    init(viewModel: RegionalityViewModel) {
        self.viewModel = viewModel
        super.init(nibName: nil , bundle: nil)

        self.viewModel.delegate = self
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        view = contentView
        contentView.delegate = self
        contentView.updateView()

        navigationController?.navigationBar.barTintColor = .black
        navigationController?.navigationBar.tintColor = .white
    }
}
//MARK: RegionalityViewDelegate
extension RegionalityViewController: RegionalityViewDelegate {
    func buttonAction() {
        let regionality = viewModel.getSelectedRegionality()
        let viewModel = AppliancesViewModel()
        let dataSource = AppliancesTableViewDataSource()
        let contentView = AppliancesView(dataSource: dataSource)

        let nextVC = AppliancesViewController(viewModel: viewModel,
                                              parameters: .init(regionalityType: regionality),
                                              contentView: contentView)

        navigationController?.pushViewController(nextVC, animated: true)
    }
```

```
    func handlerCardSelection(with selectedRegionality: String) {
        viewModel.handleSelectedRegionality(with: selectedRegionality)
    }
}
//MARK: RegionalityViewModelDelegate
extension RegionalityViewController: RegionalityViewModelDelegate {
    func updateSelector(with regionality: RegionalityType, isSelect: Bool) {
        contentView.updateSelector(with: regionality, isSelect: isSelect)
    }
}

class RegionalityViewModel {

    //MARK: PROPERTIES
    private var regionality: String?
    weak var delegate: RegionalityViewModelDelegate?

    private func toggleRegionalitySelection(selectedRegionality: String) {
        if selectedRegionality == RegionalityType.countrySide.rawValue {
            delegate?.updateSelector(with: RegionalityType.urbanSide, isSelect: false)
        } else {
            delegate?.updateSelector(with: RegionalityType.countrySide, isSelect: false)
        }
    }
}
//MARK: RegionalityViewModelProtocol
extension RegionalityViewModel: RegionalityViewModelProtocol {
    func getSelectedRegionality() -> String? {
        return regionality ?? nil
    }


    func handleSelectedRegionality(with selectedRegionality: String) {
        guard regionality != selectedRegionality else {
            toggleRegionalitySelection(selectedRegionality: selectedRegionality)
            regionality = nil
            return
        }
        toggleRegionalitySelection(selectedRegionality: selectedRegionality)
        regionality = selectedRegionality
    }
}

protocol RegionalityViewModelDelegate: AnyObject {
    func updateSelector(with regionality: RegionalityType, isSelect: Bool)
}

protocol RegionalityViewModelProtocol: AnyObject {
    var delegate: RegionalityViewModelDelegate? { get set }

    func handleSelectedRegionality(with selectedRegionality: String)
    func getSelectedRegionality() -> String?
}
```

APÊNDICE C - Código Tela de Equipamentos

```
import Foundation
```

```swift
import UIKit
class AppliancesView: UIView {

    private let controllCenter: ControllCenterProtocol = {
        let card = ControllCenter()
        card.translatesAutoresizingMaskIntoConstraints = false
        return card
    }()

    private let scrollView: UIScrollView = {
        let scrollView = UIScrollView()
        scrollView.showsVerticalScrollIndicator = true
        scrollView.indicatorStyle = .white
        scrollView.translatesAutoresizingMaskIntoConstraints = false
        return scrollView
    }()

    private let containerView: UIView = {
        let content = UIView()
        content.backgroundColor = .clear
        content.translatesAutoresizingMaskIntoConstraints = false
        return content
    }()

    private lazy var footer: FooterView = {
        let footer = FooterView()
        footer.button.addTarget(self, action: #selector(buttonAction), for: .touchUpInside)
        footer.translatesAutoresizingMaskIntoConstraints = false
        return footer
    }()

    private lazy var tableView: UITableView = {
        let tableView = UITableView()
        tableView.backgroundColor = .clear
        tableView.separatorStyle = .none
        tableView.showsVerticalScrollIndicator = false
        tableView.register(AppliancesViewCell.self, forCellReuseIdentifier: "appliancesCell")
        tableView.translatesAutoresizingMaskIntoConstraints  = false
        return tableView
    }()

    // MARK: Public Initializers
    weak var delegate: AppliancesViewDelegate?
    var previousDataModel: AppliancesViewCellEntity?
    var dataSource: AppliancesTableViewDataSource

    // MARK: Private Initializers
    private var items: [AppliancesViewCellEntity] = []

    init(dataSource: AppliancesTableViewDataSource) {
        self.dataSource = dataSource
        super.init(frame: .zero)
        setup()
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
```

```
    @objc public func buttonAction() {
        delegate?.buttonAction()
    }

    // MARK: Private Functions
    private func setup() {
        backgroundColor = .black
        footer.updateButton(with: .proceed)

        setupTable()
        buildViewHierarchy()
        addConstraints()
    }

    private func setupTable() {
        let entities = AppliancesStore.appliances.map({dataSource.convertAppliancesModelToEntity(withModel:
$0)})
        items = entities

        dataSource.items = entities
        dataSource.dataSourceDelegate = self

        tableView.dataSource = dataSource
        tableView.delegate = dataSource

        tableView.reloadData()
    }

    private func buildViewHierarchy() {
        addSubview(scrollView)
        addSubview(footer)
        scrollView.addSubview(containerView)

        containerView.addSubview(controllCenter)
        containerView.addSubview(tableView)
    }
    private func addConstraints() {
        NSLayoutConstraint.activate([
            scrollView.topAnchor.constraint(equalTo: topAnchor),
            scrollView.leadingAnchor.constraint(equalTo: leadingAnchor),
            scrollView.trailingAnchor.constraint(equalTo: trailingAnchor),
            scrollView.bottomAnchor.constraint(equalTo: footer.topAnchor),

            containerView.topAnchor.constraint(equalTo: scrollView.topAnchor),
            containerView.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
            containerView.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium),
            containerView.bottomAnchor.constraint(equalTo: scrollView.bottomAnchor),

            controllCenter.topAnchor.constraint(equalTo: containerView.topAnchor, constant: Metrics.Spacing.small),
            controllCenter.leadingAnchor.constraint(equalTo: containerView.leadingAnchor),
            controllCenter.trailingAnchor.constraint(equalTo: containerView.trailingAnchor),

            tableView.topAnchor.constraint(equalTo: controllCenter.bottomAnchor, constant:
Metrics.Spacing.medium),
            tableView.leadingAnchor.constraint(equalTo: containerView.leadingAnchor),
            tableView.trailingAnchor.constraint(equalTo: containerView.trailingAnchor),
            tableView.bottomAnchor.constraint(equalTo: containerView.bottomAnchor),
```

```
            footer.topAnchor.constraint(equalTo: containerView.bottomAnchor),
            footer.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
            footer.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium),
            footer.bottomAnchor.constraint(equalTo: safeAreaLayoutGuide.bottomAnchor)
        ])
    }
}
//MARK: EquipmentsViewProtocol
extension AppliancesView: AppliancesViewProtocol {
    func updateControllCenter(date: String,
                              local: String,
                              totalPower: String,
                              typeOfConnection: String,
                              regionality: String?)
    {
        controllCenter.updateControllCenter(date: date,
                                            local: local,
                                            totalPower: "\(totalPower)",
                                            typeOfConnection: typeOfConnection,
                                            regionality: regionality)
    }
}
//MARK: FooterViewDelegate
extension AppliancesView: FooterViewDelegate { }
//MARK: AppliancesTableViewDataSourceDelegate
extension AppliancesView: AppliancesTableViewDataSourceDelegate {
    func increaseButtonTapped(for dataModel: AppliancesViewCellEntity) {
        if let index = dataSource.items.firstIndex(where: { $0 === dataModel }) {
            dataSource.items[index].quantity += 1

            let indexPath = IndexPath(row: index, section: 0)
            tableView.reloadRows(at: [indexPath], with: .none)
            delegate?.calculateTotalPower(items: items)
            delegate?.updateItems(items: items)
        }
    }

    func decreaseButtonTapped(for dataModel: AppliancesViewCellEntity) {
        if let index = dataSource.items.firstIndex(where: { $0 === dataModel }) {
            if dataSource.items[index].quantity > 0 {
                dataSource.items[index].quantity -= 1

                let indexPath = IndexPath(row: index, section: 0)
                tableView.reloadRows(at: [indexPath], with: .none)

                delegate?.calculateTotalPower(items: items)
                delegate?.updateItems(items: items)
            }
        }
    }
}

protocol AppliancesViewProtocol where Self: UIView {
    var delegate: AppliancesViewDelegate? { get set }

    func updateControllCenter(date: String,
                              local: String,
```

```
                        totalPower: String,
                        typeOfConnection: String,
                        regionality: String?)
}

protocol AppliancesViewDelegate: AnyObject {
    func buttonAction()
    func calculateTotalPower(items: [AppliancesViewCellEntity])
    func updateItems(items: [AppliancesViewCellEntity])
    func updateControllCenter(date: String,
                        local: String,
                        totalPower: String,
                        typeOfConnection: String,
                        regionality: String?)
}

class AppliancesViewController: UIViewController {

    // MARK: Private Initializers
    private var viewModel: AppliancesViewModelProtocol
    private var parameters: AppliancesParameters
    private var alertShowed = false

    // MARK: Public Initializers
    var contentView: AppliancesViewProtocol
    let alertController = UIAlertController(title: "Atenção", message: .reachPowerLimit, preferredStyle: .alert)
    let doneAction = UIAlertAction(title: "Ok", style: .default) { action in }
    let moreAction = UIAlertAction(title: "Saiba mais", style: .default) { action in
        if let deepLinkURL = URL(string: "https://www.celesc.com.br/ligacao-nova") {
            if UIApplication.shared.canOpenURL(deepLinkURL) {
                UIApplication.shared.open(deepLinkURL, options: [:], completionHandler: nil)
            }
        }
    }

    init(viewModel: AppliancesViewModel,
         parameters: AppliancesParameters,
         contentView: AppliancesViewProtocol)
    {
        self.viewModel = viewModel
        self.parameters = parameters
        self.contentView = contentView
        super.init(nibName: nil , bundle: nil)

        self.viewModel.delegate = self
        self.contentView.delegate = self
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    //MARK: LIFE CYCLE
    override func viewDidLoad() {
        super.viewDidLoad()

        contentSetup()
        updateControlCenter()
```

```
    }

    private func contentSetup() {
        view.addSubview(contentView)
        contentView.bindFrameToSuperviewBounds()
    }

    private func updateControlCenter() {
        viewModel.updateControllCenter(regionality: parameters.regionalityType)
    }

    private func setupDeepLink() {
        viewModel.updateControllCenter(regionality: parameters.regionalityType)
    }

    private func showAlertIfNeeded() {
        if viewModel.getTotalPower() > 65000  && !alertShowed {
            alertController.addAction(doneAction)
            alertController.addAction(moreAction)
            present(alertController, animated: true, completion: nil)

            alertShowed = true
        }
    }
}
//MARK: AppliancesViewDelegate
extension AppliancesViewController: AppliancesViewDelegate {
    func calculateTotalPower(items: [AppliancesViewCellEntity]) {
        viewModel.calculateTotalPower(items: items)
    }

    func updateItems(items: [AppliancesViewCellEntity]) {
        viewModel.updateItems(items: items)
        showAlertIfNeeded()
    }

    func buttonAction() {
        let results = viewModel.getResults()
        let viewModel = ResultsViewModel()

        let nextVC = ResultsViewController(parameters: results, viewModel: viewModel)
        navigationController?.pushViewController(nextVC, animated: true)
    }
}
// MARK: EquipmentsViewController
extension AppliancesViewController: AppliancesViewModelDelegate {
    func updateControllCenter(date: String,
                              local: String,
                              totalPower: String,
                              typeOfConnection: String,
                              regionality: String?)
    {
        contentView.updateControllCenter(date: date,
                                         local: local,
                                         totalPower: totalPower,
                                         typeOfConnection: typeOfConnection,
                                         regionality: regionality)
    }
```

```swift
}

class AppliancesViewModel {
    // MARK: Public Initializers
    weak var delegate: AppliancesViewModelDelegate?

    // MARK: Private Initializers
    private var totalPower: Int = 0
    private var currentRegionality = ""
    private var selectedItems: [AppliancesEntity] = []

    private func getDate() -> String {
        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "dd 'de' MMMM 'de' yyyy"
        let currentDate = Date()
        let formattedDate = dateFormatter.string(from: currentDate)

        return formattedDate
    }

    private func getTotalPower() -> String {
        return "\(totalPower) W"
    }

    private func getTypeOfConnection() -> String {
        if currentRegionality == .countySideTitle {
            if totalPower <= 50000 {
                return "Monofásica"
            } else {
                return "Trifásica"
            }

        } else {
            if totalPower <= 15000 {
                return "Monofásica"
            }

            if totalPower <= 25000 {
                return "Bifásica"
            }

            if totalPower <= 25000 {
                return "Bifásica"
            }

            return "Trifásica"
        }
    }
}
// MARK: EquipmentsViewModelProtocol
extension AppliancesViewModel: AppliancesViewModelProtocol {

    func calculateTotalPower(items: [AppliancesViewCellEntity]) {
        totalPower = 0

        for item in items {
            let power = item.power
            let quantity = item.quantity
```

```
            let powerForItem = power * quantity
            totalPower += powerForItem
        }
        updateControllCenter(regionality: currentRegionality)
    }

    func updateItems(items: [AppliancesViewCellEntity]) {
        selectedItems = items.filter { $0.quantity != 0 }.map { item in
            return AppliancesEntity(appliance: item.title,
                                    power: "\(item.power)",
                                    quantity: "\(item.quantity)")
        }
    }

    func updateControllCenter(regionality: String?) {

        currentRegionality = regionality ?? .urbanSideTitle

        delegate?.updateControllCenter(date: getDate(),
                                       local: .santaCatarina,
                                       totalPower: getTotalPower(),
                                       typeOfConnection: getTypeOfConnection(),
                                       regionality: currentRegionality)
    }

    func getResults() -> ResultsEntity {
        .init(regionality: currentRegionality, totalPower: "\(totalPower)", type: getTypeOfConnection(), selectedItems:
selectedItems)
    }

    func getTotalPower() -> Int {
        return totalPower
    }
}

protocol AppliancesViewModelProtocol {
    var delegate: AppliancesViewModelDelegate? { get set }
    func updateControllCenter(regionality: String?)
    func calculateTotalPower(items: [AppliancesViewCellEntity])
    func getResults() -> ResultsEntity
    func getTotalPower() -> Int
    func updateItems(items: [AppliancesViewCellEntity])
}

protocol AppliancesViewModelDelegate: AnyObject {
    func updateControllCenter(date: String,
                              local: String,
                              totalPower: String,
                              typeOfConnection: String,
                              regionality: String?)
}

class AppliancesViewCell: UITableViewCell {
    private let storageView: UIView = {
        let view = UIView()
        view.backgroundColor = .black
        view.layer.cornerRadius = 8
        view.translatesAutoresizingMaskIntoConstraints = false
```

```
    return view
}()

private let icon: UIImageView = {
    let icon = UIImageView()
    icon.translatesAutoresizingMaskIntoConstraints = false
    return icon
}()

private let titleLabel: UILabel = {
    let label = UILabel()
    label.textColor = .white
    label.font = .boldSystemFont(ofSize: 15)
    label.numberOfLines = 0
    label.translatesAutoresizingMaskIntoConstraints = false
    return label
}()

private let powerDescriptionLabel: UILabel = {
    let label = UILabel()
    label.textColor = .white
    label.font = .boldSystemFont(ofSize: 12)
    label.translatesAutoresizingMaskIntoConstraints = false
    return label
}()

private let quantityLabel: UILabel = {
    let label = UILabel()
    label.textColor = .white
    label.text = "0"
    label.font = .boldSystemFont(ofSize: 14)
    label.translatesAutoresizingMaskIntoConstraints = false
    return label
}()

 let increaseButton: UIButton = {
    let button = UIButton()
    button.setTitle("+", for: .normal)
    button.setTitleColor(.white, for: .normal)
    button.backgroundColor = .gray
    button.layer.cornerRadius = 5
    button.addTarget(self, action: #selector(increaseButtonTapped), for: .touchUpInside)
    button.translatesAutoresizingMaskIntoConstraints = false
    return button
}()

 let decreaseButton: UIButton = {
    let button = UIButton()
    button.setTitle("-", for: .normal)
    button.setTitleColor(.white, for: .normal)
    button.backgroundColor = .gray
    button.layer.cornerRadius = 5
    button.addTarget(self, action: #selector(decreaseButtonTapped), for: .touchUpInside)
    button.translatesAutoresizingMaskIntoConstraints = false
    return button
}()

private var currentQuantityValue = 0
```

```swift
var dataModel: AppliancesViewCellEntity?
weak var dataSourceDelegate: AppliancesTableViewDataSourceDelegate?

// MARK: PROPERTIES
override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
    super.init(style: style, reuseIdentifier: reuseIdentifier)
    setup()
}

@available(*, unavailable)
required init?(coder: NSCoder) {
    fatalError("init(coder:) has not been implemented")
}

@objc func increaseButtonTapped() {
    if let dataModel = dataModel {
        dataSourceDelegate?.increaseButtonTapped(for: dataModel)
    }
}

@objc func decreaseButtonTapped() {
    if let dataModel = dataModel {
        dataSourceDelegate?.decreaseButtonTapped(for: dataModel)
    }
}

override func prepareForReuse() {
    super.prepareForReuse()
}

// MARK: Private Methods
private func setup() {
    backgroundColor = .clear
    buildHierarchy()
    setupConstraints()
}

private func buildHierarchy() {
    contentView.addSubview(storageView)
    storageView.addSubview(icon)
    storageView.addSubview(titleLabel)
    storageView.addSubview(powerDescriptionLabel)
    storageView.addSubview(quantityLabel)
    storageView.addSubview(increaseButton)
    storageView.addSubview(decreaseButton)
}

private func setupConstraints() {
    NSLayoutConstraint.activate([
        storageView.topAnchor.constraint(equalTo: topAnchor, constant: 4),
        storageView.leadingAnchor.constraint(equalTo: leadingAnchor),
        storageView.trailingAnchor.constraint(equalTo: trailingAnchor),
        storageView.bottomAnchor.constraint(equalTo: bottomAnchor, constant: -4),
        icon.topAnchor.constraint(equalTo: storageView.topAnchor, constant: Metrics.Spacing.tiny),
        icon.leadingAnchor.constraint(equalTo: storageView.leadingAnchor, constant: Metrics.Spacing.tiny),
        icon.bottomAnchor.constraint(equalTo: storageView.bottomAnchor, constant: -Metrics.Spacing.tiny),
        icon.widthAnchor.constraint(equalToConstant: 50),
        icon.heightAnchor.constraint(equalToConstant: 50),
```

```
            titleLabel.topAnchor.constraint(equalTo: storageView.topAnchor, constant: 15),
            titleLabel.leadingAnchor.constraint(equalTo: icon.trailingAnchor, constant: Metrics.Spacing.tiny),
            titleLabel.widthAnchor.constraint(equalToConstant: 200),

            powerDescriptionLabel.topAnchor.constraint(equalTo: titleLabel.bottomAnchor, constant:
Metrics.Spacing.tiny),
            powerDescriptionLabel.leadingAnchor.constraint(equalTo: icon.trailingAnchor, constant:
Metrics.Spacing.tiny),

            increaseButton.centerYAnchor.constraint(equalTo: storageView.centerYAnchor),
            increaseButton.trailingAnchor.constraint(equalTo: storageView.trailingAnchor, constant:
-Metrics.Spacing.small),
            increaseButton.widthAnchor.constraint(equalToConstant: 30),
            increaseButton.heightAnchor.constraint(equalToConstant: 30),

            decreaseButton.centerYAnchor.constraint(equalTo: storageView.centerYAnchor),
            decreaseButton.trailingAnchor.constraint(equalTo: increaseButton.leadingAnchor, constant:
-Metrics.Spacing.tiny),
            decreaseButton.widthAnchor.constraint(equalToConstant: 30),
            decreaseButton.heightAnchor.constraint(equalToConstant: 30),

            quantityLabel.topAnchor.constraint(equalTo: storageView.topAnchor, constant: Metrics.Spacing.medium),
            quantityLabel.trailingAnchor.constraint(equalTo: decreaseButton.leadingAnchor, constant:
-Metrics.Spacing.small),
            quantityLabel.bottomAnchor.constraint(equalTo: storageView.bottomAnchor, constant:
-Metrics.Spacing.medium),
        ])
    }

    // MARK: Setup Methods

    public func setDataCell(withData data: AppliancesViewCellEntity) {
        dataModel = data

        icon.image = UIImage(named: data.icon)
        titleLabel.text = data.title
        powerDescriptionLabel.text = "\(data.power) W"
        quantityLabel.text = "\(data.quantity)"

        increaseButton.isUserInteractionEnabled = true
        decreaseButton.isUserInteractionEnabled = true

    }
}

class AppliancesViewCellEntity {
    var title: String
    var power: Int
    var icon: String
    var quantity: Int

    init(title: String, power: Int, icon: String, quantity: Int) {
        self.title = title
        self.power = power
        self.icon = icon
        self.quantity = quantity
    }
```

```
}

protocol AppliancesTableViewDataSourceDelegate: AnyObject {
    func increaseButtonTapped(for dataModel: AppliancesViewCellEntity)
    func decreaseButtonTapped(for dataModel: AppliancesViewCellEntity)
}
final class AppliancesTableViewDataSource: NSObject, AppliancesTableViewDataSourceProtocol {
    public var items: [AppliancesViewCellEntity] = []
    weak var dataSourceDelegate: AppliancesTableViewDataSourceDelegate?

    func tableView(_ tableView: UITableView,
                   numberOfRowsInSection section: Int) -> Int {
        return items.count
    }

    func tableView(_ tableView: UITableView,
                   cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "appliancesCell", for: indexPath) as!
AppliancesViewCell
        let item = items[indexPath.row]

        cell.selectionStyle = .none
        cell.dataSourceDelegate = dataSourceDelegate
        cell.setDataCell(withData: item)
        cell.isUserInteractionEnabled = true
        return cell
    }

    func convertAppliancesModelToEntity(withModel model: AppliancesModel) -> AppliancesViewCellEntity {
        return AppliancesViewCellEntity(title: model.name, power: model.power, icon: model.icon, quantity:
model.quantity)
    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return 76
    }

    func tableView(_ tableView: UITableView, viewForHeaderInSection section: Int) -> UIView? {
        let headerView = UIView()
        headerView.backgroundColor = .black
        let label = UILabel()
        label.textAlignment = .left
        label.lineBreakMode = .byWordWrapping
        label.numberOfLines = 0
        label.textColor = .white
        label.font = UIFont.boldSystemFont(ofSize: 18)
        label.text = .selectYourEquipments
        label.translatesAutoresizingMaskIntoConstraints = false
        headerView.addSubview(label)
        NSLayoutConstraint.activate([
            label.topAnchor.constraint(equalTo: headerView.topAnchor),
            label.leadingAnchor.constraint(equalTo: headerView.leadingAnchor),
            label.trailingAnchor.constraint(equalTo: headerView.trailingAnchor),
            label.bottomAnchor.constraint(equalTo: headerView.bottomAnchor, constant: -16)
        ])
        return headerView
    }
}
```

```
protocol AppliancesTableViewDataSourceProtocol: UITableViewDataSource, UITableViewDelegate {
    var items: [AppliancesViewCellEntity] { get set }

    func convertAppliancesModelToEntity(withModel model: AppliancesModel) -> AppliancesViewCellEntity
}
```

APÊNDICE D - Código Tela de Resultados

```
import Foundation
import UIKit
class ResultsView: UIView {
    private let titleLabel: UILabel = {
        let label = UILabel()
        label.textAlignment = .left
        label.lineBreakMode = .byWordWrapping
        label.numberOfLines = 0
        label.textColor = .white
        label.font = UIFont.boldSystemFont(ofSize: 36)
        label.translatesAutoresizingMaskIntoConstraints = false
        return label
    }()

    private let scrollView: UIScrollView = {
        let scroll = UIScrollView()
        scroll.showsVerticalScrollIndicator = true
        scroll.indicatorStyle = .white
        scroll.translatesAutoresizingMaskIntoConstraints = false
        return scroll
    }()

    private let stackView: UIStackView = {
        let stack = UIStackView()
        stack.axis = .vertical
        stack.spacing = Metrics.Spacing.large
        stack.translatesAutoresizingMaskIntoConstraints = false
        return stack
    }()

    private let typeLabel: UILabel = {
        let label = UILabel()
        label.textAlignment = .left
        label.lineBreakMode = .byWordWrapping
        label.numberOfLines = 0
        label.textColor = .deepOrange
        label.font = UIFont.boldSystemFont(ofSize: 28)
        label.isHidden = false
```

```
    label.translatesAutoresizingMaskIntoConstraints = false
    return label
}()

private let descriptionLabel: UILabel = {
    let label = UILabel()
    label.textAlignment = .left
    label.lineBreakMode = .byWordWrapping
    label.numberOfLines = 0
    label.textColor = .white
    label.font = UIFont.boldSystemFont(ofSize: 18)
    label.isHidden = false
    label.translatesAutoresizingMaskIntoConstraints = false
    return label
}()

private let firstTag: TagWithIconViewProtocol = {
    let tag = TagWithIconView()
    tag.translatesAutoresizingMaskIntoConstraints = false
    return tag
}()

private let secondTag: TagWithIconViewProtocol = {
    let tag = TagWithIconView()
    tag.translatesAutoresizingMaskIntoConstraints = false
    return tag
}()

private let footer: FooterViewProtocol = {
    let footer = FooterView()
    footer.button.addTarget(self, action: #selector(buttonAction), for: .touchUpInside)
    footer.translatesAutoresizingMaskIntoConstraints = false
    return footer
}()

private let tagStackView: UIStackView = {
    let stack = UIStackView()
    stack.axis = .horizontal
    stack.spacing = Metrics.Spacing.small
    stack.translatesAutoresizingMaskIntoConstraints = false
    return stack
}()

private let connectionTypeCard: ConnectionTypeCardView = {
    let card = ConnectionTypeCardView()
    card.translatesAutoresizingMaskIntoConstraints = false
    return card
}()

// MARK: Properties
weak var delegate: ResultsViewDelegate?

// MARK: INITIALIZERS
init() {
    super.init(frame: .zero)
    setup()
}
```

```
    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    @objc public func buttonAction() {
        delegate?.buttonAction()
    }

    // MARK: PRIVATE FUNCTIONS

    private func setup() {
        backgroundColor = .black
        footer.updateButton(with: .proceed)

        buildViewHierarchy()
        addConstraints()

        setupFirstTapGesture()
        setupSecondTapGesture()
    }

    private func buildViewHierarchy() {
        addSubview(scrollView)
        addSubview(footer)

        scrollView.addSubview(stackView)

        stackView.addArrangedSubview(titleLabel)
        stackView.addArrangedSubview(typeLabel)
        stackView.addArrangedSubview(descriptionLabel)
        stackView.addArrangedSubview(tagStackView)
        stackView.addArrangedSubview(connectionTypeCard)

        tagStackView.addArrangedSubview(firstTag)
        tagStackView.addArrangedSubview(secondTag)
    }

    private func addConstraints() {
        NSLayoutConstraint.activate([
            scrollView.topAnchor.constraint(equalTo: safeAreaLayoutGuide.topAnchor),
            scrollView.leadingAnchor.constraint(equalTo: leadingAnchor),
            scrollView.trailingAnchor.constraint(equalTo: trailingAnchor),

            stackView.topAnchor.constraint(equalTo: scrollView.topAnchor, constant: Metrics.Spacing.medium),
            stackView.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
            stackView.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium),
            stackView.bottomAnchor.constraint(equalTo: scrollView.bottomAnchor),

            tagStackView.trailingAnchor.constraint(equalTo: stackView.trailingAnchor, constant:
-Metrics.Spacing.bigger),

            footer.topAnchor.constraint(equalTo: scrollView.bottomAnchor),
            footer.bottomAnchor.constraint(equalTo: safeAreaLayoutGuide.bottomAnchor),
            footer.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
            footer.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium)
        ])
    }
```

```swift
    private func setupFirstTapGesture() {
        let tapGesture = UITapGestureRecognizer(target: self, action: #selector(openFormLink))
        firstTag.addGestureRecognizer(tapGesture)
    }

    private func setupSecondTapGesture() {
        let tapGesture = UITapGestureRecognizer(target: self, action: #selector(openManualLink))
        secondTag.addGestureRecognizer(tapGesture)
    }

    @objc func openManualLink() {
        if let url = URL(string:
"https://www.celesc.com.br/arquivos/normas-tecnicas/padrao-entrada/manual-simplificado-padrao-entrada-energia-eletrica.pdf") {
            UIApplication.shared.open(url, options: [:], completionHandler: nil)
        }
    }

    @objc func openFormLink() {
        if let url = URL(string: "https://celescdvat.omd.com.br/dvat/externo/cadastro.do") {
            UIApplication.shared.open(url, options: [:], completionHandler: nil)
        }
    }

    private func updateTags() {
        firstTag.updateTag(text: "Formulário online",
                           iconName: "plus",
                           shoulShowBackground: true)

        secondTag.updateTag(text: "Manual simplificado",
                            iconName: "book.fill",
                            shoulShowBackground: true)
    }
}
extension ResultsView: ResultsViewProtocol {

    func updateView(with title: String,
                    with description: String?,
                    with typeDescription: String?)
    {
        updateTags()
        titleLabel.text = title

        guard let description = description else {
            descriptionLabel.isHidden = true
            return
        }
        descriptionLabel.text = description

        guard let typeDescription = typeDescription else {
            typeLabel.isHidden = true
            return
        }
        typeLabel.text = typeDescription
    }
}

protocol ResultsViewProtocol where Self: UIView {
```

```
    var delegate: ResultsViewDelegate? { get set }

    func updateView(with title: String, with description: String?, with typeDescription: String?)
}

protocol ResultsViewDelegate: AnyObject {
    func buttonAction()
}

class ResultsViewController: UIViewController {

    // MARK: Private Initializers
    private var parameters: ResultsEntity
    private let viewModel: ResultsViewModelProtocol

    // MARK: Public Initializers
    var contentView: ResultsViewProtocol = {
        let view = ResultsView()
        return view
    }()

    init(parameters: ResultsEntity,
         viewModel: ResultsViewModelProtocol)
    {
        self.parameters = parameters
        self.viewModel = viewModel
        super.init(nibName: nil , bundle: nil)
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    //MARK: Life cycle
    override func viewDidLoad() {
        super.viewDidLoad()

        contentSetup()
        updateView()
    }

    private func contentSetup() {
        contentView.delegate = self
        view = contentView
    }

    private func updateView() {
        guard let totalPower = Int(parameters.totalPower),
              totalPower > 0 else {
            contentView.updateView(with: .addEquipmentsTitle,
                                   with: .addEquipmentsDescription, with: nil)
            return
        }

        let title = viewModel.getTitle(with: totalPower)
        let description = viewModel.getDescription(with: totalPower)
        let type = viewModel.getType(with: totalPower)
        contentView.updateView(with: title, with: description, with: type)
```

```
    }
}
extension ResultsViewController: ResultsViewDelegate {
    func buttonAction() {

        let power = Int(parameters.totalPower) ?? 0

        let totalPower = parameters.totalPower
        let regionality = parameters.regionality
        let type = viewModel.getType(with: power)
        let selectedItems = parameters.selectedItems
        let demand = viewModel.getDemand(with: Double(power))


        let nextVC = PDFExporterViewController(parameters: .init(regionality: regionality,
                                                totalPower: totalPower,
                                                demand: demand,
                                                type: type,
                                                selectedItems: selectedItems))

        navigationController?.pushViewController(nextVC, animated: true)
    }
}

class ResultsViewModel { }
// MARK: EquipmentsViewModelProtocol
extension ResultsViewModel: ResultsViewModelProtocol {
    func getTitle(with totalPower: Int) -> String {
        return "Você precisará de \(totalPower) Watts de potência."
    }

    func getType(with totalPower: Int) -> String {
        if totalPower <= 15000 {
            return "Monofásico"
        } else if totalPower <= 25000 {
            return "Bifásico"
        } else {
            return "Trifásico"
        }
    }

    func getDemand(with totalPower: Double) -> String {
        let demand = totalPower * 0.2135
        return "\(Int(demand))"
    }

    func getDescription(with totalPower: Int) -> String {
        if totalPower == 15000 || totalPower == 25000 || totalPower == 75000 {
            return .reachLimit
        } else if totalPower < 15000 {
            let missingPower = 15000 - totalPower
            return "Para ligações do tipo monofásica, é possível solicitar até 15 kW sem custos adicionais. Caso
queira, volte e adicione mais \(missingPower) Watts em equipamentos à sua lista."
        } else if totalPower < 25000 {
            let missingPower = 25000 - totalPower
            return "Para ligações do tipo bifásica, é possível solicitar até 25 kW sem custos adicionais. Caso queira,
volte e adicione mais \(missingPower) Watts em equipamentos à sua lista."
        } else if totalPower < 65000 {
```

```
            let missingPower = 75000 - totalPower
            return "Para ligações do tipo trifásica, é possível solicitar até 75 kW sem custos adicionais. Caso queira, volte e adicione mais \(missingPower) Watts em equipamentos à sua lista."
        } else {
            return "Atenção, deverá ser justificada a necessidade por meio do cálculo da demanda e documento de responsabilidade técnica por profissional habilitado via sistema PEP."
        }
    }
}

protocol ResultsViewModelProtocol {
    func getTitle(with totalPower: Int) -> String
    func getType(with totalPower: Int) -> String
    func getDemand(with totalPower: Double) -> String
    func getDescription(with totalPower: Int) -> String
}

struct ResultsEntity {
    let regionality: String
    let totalPower: String
    let type: String
    let selectedItems: [AppliancesEntity]
}
```

APÊNDICE E - Código Tela de Exportar PDF

```
import UIKit
class PDFExporterView: UIView, PDFExporterViewProtocol {

    private let scrollView: UIScrollView = {
        let scroll = UIScrollView()
        scroll.showsVerticalScrollIndicator = true
        scroll.indicatorStyle = .white
        scroll.translatesAutoresizingMaskIntoConstraints = false
        return scroll
    }()

    private let separatorView: UILabel = {
        let separator = UILabel()
        separator.backgroundColor = .white
        separator.translatesAutoresizingMaskIntoConstraints = false
        return separator
    }()

    private let headerTitleLabel: UILabel = {
        let label = UILabel()
        label.lineBreakMode = .byWordWrapping
        label.numberOfLines = 0
        label.textColor = .white
        label.text = "Dimensionamento de carga"
        label.font = UIFont.boldSystemFont(ofSize: 26)
        label.translatesAutoresizingMaskIntoConstraints = false
        return label
    }()

    private let appliancesTitleLabel: UILabel = {
        let label = UILabel()
        label.lineBreakMode = .byWordWrapping
        label.numberOfLines = 0
        label.textColor = .white
        label.text = "Equipamentos"
        label.font = UIFont.boldSystemFont(ofSize: 16)
        label.translatesAutoresizingMaskIntoConstraints = false
        return label
```

```
}()

private let appliancesQuantityTitleLabel: UILabel = {
    let label = UILabel()
    label.lineBreakMode = .byWordWrapping
    label.numberOfLines = 0
    label.textColor = .white
    label.text = "Quantidade"
    label.font = UIFont.boldSystemFont(ofSize: 16)
    label.translatesAutoresizingMaskIntoConstraints = false
    return label
}()

private let appliancesPowerTitleLabel: UILabel = {
    let label = UILabel()
    label.lineBreakMode = .byWordWrapping
    label.numberOfLines = 0
    label.textColor = .white
    label.text = "Potência"
    label.font = UIFont.boldSystemFont(ofSize: 16)
    label.translatesAutoresizingMaskIntoConstraints = false
    return label
}()

private let descriptionStack: UIStackView = {
    let content = UIStackView()
    content.backgroundColor = .black
    content.alignment = .fill
    content.axis = .vertical
    content.spacing = Metrics.Spacing.small
    content.translatesAutoresizingMaskIntoConstraints = false
    return content
}()

private let appliancesNameStack: UIStackView = {
    let content = UIStackView()
    content.backgroundColor = .black
    content.alignment = .fill
    content.axis = .vertical
    content.spacing = Metrics.Spacing.tiny
    content.translatesAutoresizingMaskIntoConstraints = false
    return content
}()

private let appliancesPowerStack: UIStackView = {
    let content = UIStackView()
    content.backgroundColor = .black
    content.alignment = .fill
    content.axis = .vertical
    content.spacing = Metrics.Spacing.tiny
    content.translatesAutoresizingMaskIntoConstraints = false
    return content
}()

private let appliancesQuantityStack: UIStackView = {
    let content = UIStackView()
    content.backgroundColor = .black
    content.alignment = .fill
```

```
    content.axis = .vertical
    content.spacing = Metrics.Spacing.tiny
    content.translatesAutoresizingMaskIntoConstraints = false
    return content
}()

private let appliancesStack: UIStackView = {
    let content = UIStackView()
    content.backgroundColor = .black
    content.alignment = .fill
    content.axis = .horizontal
    content.spacing = Metrics.Spacing.small
    content.translatesAutoresizingMaskIntoConstraints = false
    return content
}()

// MARK: INITIALIZERS

init() {
    super.init(frame: .zero)
    setup()
}

required init?(coder aDecoder: NSCoder) {
    fatalError("init(coder:) has not been implemented")
}

// MARK: PRIVATE FUNCTIONS

private func setup() {
    backgroundColor = .black

    buildViewHierarchy()
    addConstraints()
}

private func buildViewHierarchy() {
    addSubview(scrollView)
    scrollView.addSubview(headerTitleLabel)
    scrollView.addSubview(descriptionStack)
    scrollView.addSubview(separatorView)
    scrollView.addSubview(appliancesStack)

    appliancesStack.addArrangedSubview(appliancesNameStack)
    appliancesStack.addArrangedSubview(appliancesPowerStack)
    appliancesStack.addArrangedSubview(appliancesQuantityStack)

    appliancesNameStack.addArrangedSubview(appliancesTitleLabel)
    appliancesPowerStack.addArrangedSubview(appliancesPowerTitleLabel)
    appliancesQuantityStack.addArrangedSubview(appliancesQuantityTitleLabel)
}

private func addConstraints() {
    NSLayoutConstraint.activate([
        scrollView.topAnchor.constraint(equalTo: safeAreaLayoutGuide.topAnchor),
        scrollView.leadingAnchor.constraint(equalTo: leadingAnchor),
        scrollView.trailingAnchor.constraint(equalTo: trailingAnchor),
        scrollView.bottomAnchor.constraint(equalTo: bottomAnchor),
```

```
            headerTitleLabel.topAnchor.constraint(equalTo: scrollView.topAnchor, constant: Metrics.Spacing.medium),
            headerTitleLabel.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
            headerTitleLabel.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium),

            descriptionStack.topAnchor.constraint(equalTo: headerTitleLabel.bottomAnchor, constant:
Metrics.Spacing.medium),
            descriptionStack.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
            descriptionStack.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium),

            separatorView.topAnchor.constraint(equalTo: descriptionStack.bottomAnchor, constant:
Metrics.Spacing.medium),
            separatorView.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
            separatorView.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium),
            separatorView.heightAnchor.constraint(equalToConstant: 0.3),

            appliancesStack.topAnchor.constraint(equalTo: separatorView.bottomAnchor, constant:
Metrics.Spacing.large),
            appliancesStack.leadingAnchor.constraint(equalTo: leadingAnchor, constant: Metrics.Spacing.medium),
            appliancesStack.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -Metrics.Spacing.medium),
            appliancesStack.bottomAnchor.constraint(equalTo: scrollView.bottomAnchor, constant:
-Metrics.Spacing.medium),
        ])
    }

    func exportToPDF() -> Data? {
        let pdfRenderer = UIGraphicsPDFRenderer(bounds: bounds)

        let pdfData = pdfRenderer.pdfData { context in
            context.beginPage()
            layer.render(in: context.cgContext)
        }

        return pdfData
    }

    func update(type: String,
                regionality: String,
                power: String,
                demand: String,
                appliances: [AppliancesEntity]) {

        let totalPower = Int(power) ?? 0

        let typeLabel = UILabel()
        typeLabel.numberOfLines = 0
        typeLabel.textColor = .white
        typeLabel.font = UIFont.systemFont(ofSize: 14)
        typeLabel.translatesAutoresizingMaskIntoConstraints = false
        typeLabel.text = "Tipe de ligação: \(type)"

        let regionalityLabel = UILabel()
        regionalityLabel.numberOfLines = 0
        regionalityLabel.textColor = .white
        regionalityLabel.font = UIFont.systemFont(ofSize: 14)
        regionalityLabel.translatesAutoresizingMaskIntoConstraints = false
        regionalityLabel.text = "Regionalidade: \(regionality)"
```

```
        let totalPowerLabel = UILabel()
        totalPowerLabel.numberOfLines = 0
        totalPowerLabel.textColor = .white
        totalPowerLabel.font = UIFont.systemFont(ofSize: 14)
        totalPowerLabel.translatesAutoresizingMaskIntoConstraints = false
        totalPowerLabel.text = "Potência total: \(power) kW"

        if totalPower > 22 {
            let demandLabel = UILabel()
            demandLabel.numberOfLines = 0
            demandLabel.textColor = .white
            demandLabel.font = UIFont.systemFont(ofSize: 14)
            demandLabel.translatesAutoresizingMaskIntoConstraints = false
            demandLabel.text = "Demanda: \(demand) kW"
            descriptionStack.addArrangedSubview(demandLabel)
        }

        descriptionStack.addArrangedSubview(typeLabel)
        descriptionStack.addArrangedSubview(regionalityLabel)
        descriptionStack.addArrangedSubview(totalPowerLabel)

        appliances.forEach { appliance in
            let applianceNameLabel = UILabel()
            applianceNameLabel.translatesAutoresizingMaskIntoConstraints = false
            applianceNameLabel.text = appliance.appliance
            applianceNameLabel.font = UIFont.systemFont(ofSize: 12)
            applianceNameLabel.textColor = .white

            let appliancePowerLabel = UILabel()
            appliancePowerLabel.translatesAutoresizingMaskIntoConstraints = false
            appliancePowerLabel.text = "\(appliance.power) W"
            appliancePowerLabel.font = UIFont.systemFont(ofSize: 12)
            appliancePowerLabel.textColor = .white

            let applianceQuantityLabel = UILabel()
            applianceQuantityLabel.translatesAutoresizingMaskIntoConstraints = false
            applianceQuantityLabel.text = appliance.quantity
            applianceQuantityLabel.font = UIFont.systemFont(ofSize: 12)
            applianceQuantityLabel.textColor = .white

            appliancesNameStack.addArrangedSubview(applianceNameLabel)
            appliancesPowerStack.addArrangedSubview(appliancePowerLabel)
            appliancesQuantityStack.addArrangedSubview(applianceQuantityLabel)
        }
    }
}

protocol PDFExporterViewProtocol where Self: UIView {
    func exportToPDF() -> Data?
    func update(type: String,
                regionality: String,
                power: String,
                demand: String,
                appliances: [AppliancesEntity])
}

class PDFExporterViewController: UIViewController {
```

```swift
    private let parameters: PDFExporterEntity

    // MARK: Properties
    let contentView: PDFExporterViewProtocol = {
        let view = PDFExporterView()
        return view
    }()

    init(parameters: PDFExporterEntity) {
        self.parameters = parameters
        super.init(nibName: nil , bundle: nil)
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    //MARK: Life cycle
    override func viewDidLoad() {
        super.viewDidLoad()

        view = contentView
        view.backgroundColor = .black
        navigationController?.navigationBar.barTintColor = .black

        setupExportBarButton()
        updateContentView()
    }

    private func setupExportBarButton() {
        let exportBarButton = UIBarButtonItem(image: UIImage(systemName: "square.and.arrow.up"), style: .plain,
target: self, action: #selector(exportToPDFButtonTapped))
        navigationItem.rightBarButtonItem = exportBarButton
    }

    private func updateContentView() {
        contentView.update(type: parameters.type,
                           regionality: parameters.regionality,
                           power: parameters.totalPower,
                           demand: parameters.demand,
                           appliances: parameters.selectedItems)
    }

    @objc private func exportToPDFButtonTapped() {

        if let pdfData = contentView.exportToPDF() {
            let pdfURL = FileManager.default.urls(for: .documentDirectory, in:
.userDomainMask)[0].appendingPathComponent("myPDF.pdf")

            do {
                try pdfData.write(to: pdfURL)
                print("PDF exportado com sucesso para \(pdfURL)")

                // Criar o UIActivityViewController
                let activityViewController = UIActivityViewController(activityItems: [pdfURL], applicationActivities: nil)
                activityViewController.popoverPresentationController?.barButtonItem =
navigationItem.rightBarButtonItem
                present(activityViewController, animated: true, completion: nil)
```

```
            } catch {
                print("Erro ao exportar PDF: \(error.localizedDescription)")
            }
        }

    }
}

import Foundation
struct PDFExporterEntity {
    let regionality: String
    let totalPower: String
    let demand: String
    let type: String
    let selectedItems: [AppliancesEntity]
}

struct AppliancesEntity {
    let appliance: String
    let power: String
    let quantity: String
}
```