UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GABRIEL RODRIGUES SHIOTA
GUILHERME DIEGO VIEIRA

ANÁLISE DA CAPACIDADE DO DESENVOLVIMENTO DO JAVAFX EM APLICAÇÕES MÓVEIS ATRAVÉS DE UM ESTUDO DE CASO

PONTA GROSSA
2022

GABRIEL RODRIGUES SHIOTA
GUILHERME DIEGO VIEIRA

ANÁLISE DA CAPACIDADE DO DESENVOLVIMENTO DO JAVAFX EM APLICAÇÕES MÓVEIS ATRAVÉS DE UM ESTUDO DE CASO

Analysis of JavaFX development capacity in mobile applications through a case study

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Bacharel em Ciências da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador(a): Profa. Dra. Mônica Hoeldtke Pietruchinski.

PONTA GROSSA
2022

GABRIEL RODRIGUES SHIOTA

GUILHERME DIEGO VIEIRA

ANÁLISE DA CAPACIDADE DO DESENVOLVIMENTO DO JAVAFX EM APLICAÇÕES MÓVEIS ATRAVÉS DE UM ESTUDO DE CASO

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciências da Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 25/05/2022

Mônica Hoeldtke Pietruchinski
Doutorado
Universidade Tecnológica Federal do Paraná

Diego Roberto Antunes
Doutorado
Universidade Tecnológica Federal do Paraná

Luiz Rafael Schmitke
Mestrado
Universidade Tecnológica Federal do Paraná

PONTA GROSSA

2022

Dedicamos este trabalho as nossas famílias
que nos auxiliaram em todos os nossos
momentos de dificuldades.

RESUMO

Ao decorrer dos anos o uso de dispositivos móveis vem se intensificando, fazendo com que tecnologias ao entorno deles se desenvolvam rapidamente, em meio as ferramentas possíveis, pode ser encontrado o JavaFX, que atualmente é a maneira mais moderna para a construção de interfaces gráficas ricas e dinâmicas utilizando a linguagem Java, e apesar de seguir o conceito de multiplataforma, ainda não se desenvolveu de forma efetiva para o ambiente móvel. A pesquisa apresentada neste trabalho envolveu analisar a capacidade de desenvolvimento dessa plataforma para dispositivos móveis, utilizando a ferramenta Gluon Mobile que possibilita sua implementação, a fim de obter uma aplicação multiplataforma. Buscou-se na literatura métricas que possibilitam analisar diversos critérios a respeito do desenvolvimento com a ferramenta. Como auxílio na análise da capacidade da ferramenta, foi realizado um estudo de caso, que forneceu dados suficientes para entender o estado atual da plataforma e suas capacidades. Como resultado concluiu-se que o uso da ferramenta não possui a mesma facilidade que o desenvolvimento em outras multiplataforma, porém apresenta um grande potencial de estudo.

Palavras-chave: Aplicação Móvel; Gluon Mobile; JavaFX.

ABSTRACT

Over the years, the use of mobile devices has intensified, causing technologies around them to develop quickly, among the possible tools, JavaFX can be found, which is currently the most modern way to build rich and dynamics graphical interfaces using the Java language, and despite following the multiplatform concept, it has not yet developed effectively for the mobile environment. The research presented in this work involved analyzing the development capacity of this platform for mobile devices, using the Gluon Mobile tool that enables its implementation, in order to obtain a multiplatform application. Metrics were searched in the literature that make it possible to analyze several criteria regarding the development with the tool. As an aid in the analysis of the tool's capability, a case study was carried out, which provided enough data to understand the current state of the platform and its capabilities. As a result, it was concluded that the use of the tool does not have the same ease as the development in other multiplatform, but it presents a great potential for study.

Keywords: Mobile Applications; Gluon Mobile; JavaFX.

LISTA DE ILUSTRAÇÕES

Figura 1 - Arquitetura JavaFX 17
Figura 2 - Exemplo de código FXML 19
Figura 3 - Visualização FXML 19
Figura 4 - Arquitetura do Plugin Gluon 20
Figura 6 - Tela de *login* 29
Figura 7 – Tela cadastro de pessoas 30
Figura 8 - Tela cadastro de produtos 31
Figura 9 - Tela cadastro de transação 32
Figura 10 - Telas de listagem 33
Figura 10 - Pacotes para construção no Linux 37
Figura 11 – Aba *Plugins* 51
Figura 12 – Aba *Projects* 52
Figura 13 - Tela novo projeto 53
Figura 14 - Tela definições do projeto 53
Figura 15 - Tela *views* do projeto 54
Figura 16 - Tela seleção do *SDK* 54
Figura 17 - Caminho do *JDK* 55
Figura 18 - Tela nome do projeto 55
Figura 19 - Menu Maven 56
Figura 20 - *Maven Settings* 56
Figura 21 - Variável de ambiente 57
Figura 22 - Gluonfx *run* 58
Figura 23 - Guia criar pacote 59
Figura 24 - Tela de configuração Scene Builder 60
Figura 25 - Atalho para o Scene Builder 60
Figura 26 - Classe de persistência Gluon 62
Figura 27 - Exemplo de uso da persistência 63

LISTA DE ABREVIATURAS E SIGLAS

ADB	*Android Debug Bridge*
API	*Application Programming Interface*
CSS	*Cascading Style Sheets*
F3	*Form Follows Function*
FXML	*Fx Markup Language*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
JDK	*Java Development Kit*
POM	*Project Object Model*
RAM	*Random Access Memory*
SDK	*Software Development Kit*
SQL	*Standart Query Language*
USB	*Universal serial bus*
WORA	*Write Once, Run Anywhere*
XML	*eXtensible Markup Language*

SUMÁRIO

1. INTRODUÇÃO 13
1.1 Objetivo 14
1.2 Justificativa 14
1.3 Organização do Trabalho 14
2 REFERENCIAL TEÓRICO 16
2.1 Programação móvel 16
2.2 JavaFX 16
2.2.1 Arquitetura e Características 17
2.2.2 *Scene Builder* 18
2.2.3 FXML 18
2.3 Gluon 19
2.4 GraalVM 20
2.5 Maven 21
2.6 Ferramentas de desenvolvimento Java 21
2.7 Métricas de análise de desenvolvimento de *software* 22
3 ESTUDO DE CASO 24
3.1 Aplicação 24
3.2 Funcionalidades do Sistema 24
4 PROCESSO DE DESENVOLVIMENTO DA APLICAÇÃO 26
4.1 Ferramentas utilizadas 26
4.2 Fases do desenvolvimento 26
4.3 Fase: Preparação do Ambiente de Desenvolvimento 27
4.3.1 Caminho percorrido 27
4.3.2 Problemas na configuração do ambiente 28
4.4 Fase: Implementação das interfaces gráficas 28
4.4.1 Caminho percorrido 28
4.4.2 Problemas encontrados 33
4.5 Fase: Configuração e criação da base de dados 34
4.5.1 Caminho percorrido 34
4.5.2 Problemas encontrados 34
4.6 Fase: Utilização de recurso nativo 35
4.6.1 Caminho percorrido 35
4.6.2 Problemas encontrados 36

4.7 Desenvolvimento desktop 36
5 ANÁLISE 38
5.1 Quantidade de Linhas de código, arquivos e dependências 38
5.2 Tamanho do pacote da aplicação e rastro de instalação 38
5.3 Perfil de uso da memória 39
5.4 Tempo de inicialização da aplicação e tempo de transição de tela 39
5.5 Estilo e elementos da interface do usuário 39
5.6 Curva de aprendizado 40
5.7 Comportamento de recarga em tempo real 41
5.8 Acesso ao sistema de arquivos e *hardware* do dispositivo 41
5.9 Portabilidade 42
5.10 Compilação e desenvolvimento da aplicação 42
6 CONCLUSÃO 43
REFERÊNCIAS 45
APÊNDICE A - GUIA PARA CRIAÇÃO DE APLICAÇÕES MULTIPLATAFORMA UTILIZANDO FERRAMENTAS GLUON 47

1. INTRODUÇÃO

A programação móvel vem sendo desenvolvida desde a década de 90 para os dispositivos da época (VOLTOLINI, 2014), mas devido aos sistemas operacionais existentes, os aplicativos eram implementados apenas pelos fabricantes.

Em 2008 houve o lançamento do iPhone OS SDK (*Software development kit*) e da App Store, que fez com que o mercado móvel ganhasse força, abrindo as portas até mesmo para os pequenos desenvolvedores de aplicativos e, posteriormente, o lançamento do Android SDK acelerou ainda mais esse movimento (BONNINGTON, 2014).

Atualmente a presença de dispositivos móveis é ainda maior, no ano de 2020, o Brasil registrou 234,07 milhões de acessos móveis (Anatel e Ministério da Comunicação, 2020).

O aumento desse mercado móvel anexa novas plataformas operacionais e uma variedade de dispositivos que precisam ser atendidos, necessitando criar uma nova solução para desenvolver aplicações que possam ser executadas na maioria delas, conceito conhecido como multiplataforma.

Fomentado por essas necessidades do mercado crescente, surge o JavaFX, definido por Dea *et al.* (2017) como uma nova geração de interface gráfica multiplataforma que aliada ao conceito inicial do Java de criar uma vez e executar em qualquer lugar, pode-se tornar uma escolha favorável, porém, a premissa vem de uma época onde dispositivos móveis com a versatilidade atual não existiam, sendo eles deixados de fora desse conceito.

Mesmo ficando atrás na programação móvel, o Java continuou popular no desenvolvimento para *desktop* e sistemas empresariais (GITHUT, 2022), abrindo espaço para que a organização Gluon expandisse uma linguagem já conhecida agregada a estrutura de interface JavaFX, para atuar em diversos dispositivos (DEA *et al.*, 2017).

Este projeto pretende demonstrar através de um estudo de caso utilizando o *framework* JavaFX na implementação de um sistema móvel, como é a utilização deste e analisando seus pontos positivos e negativos.

1.1 Objetivo

O objetivo geral deste trabalho é analisar a utilização do *framework* JavaFX em plataformas móveis, focado na etapa de desenvolvimento, de modo que possam ser identificados seus pontos positivos e negativos assim como seu posicionamento no mercado atual. Como objetivos específicos tem-se:

- Desenvolver um sistema móvel para controle de estoque utilizando a tecnologia JavaFX.
- Estabelecer critérios de avaliação da ferramenta JavaFX em aplicações móveis.
- Gerar um guia de desenvolvimento de um aplicativo com JavaFX.

1.2 Justificativa

Dentro do cenário atual de desenvolvimento onde o reuso de código é necessário para atender as demandas do mercado (SOMMERVILLE, 2011), estranha-se o fato do JavaFX, *framework* criado justamente para esse propósito, não esteja presente em plataformas móveis (STATISTA, 2022).

O *framework* JavaFX possibilita criar um código na linguagem Java e utilizá-lo em diferentes plataformas, como Windows, Linux, Android e IOS, resultando assim em um alto nível de produtividade, reduzindo o tempo de desenvolvimento de aplicações, bem como o seu custo (DEA *et al.*, 2017).

Justifica-se realizar essa análise para propiciar aos desenvolvedores de aplicações móveis, o conhecimento sobre o desenvolvimento com esse *framework.*

1.3 Organização do Trabalho

Este trabalho está organizado em capítulos. O capítulo 1 trata-se da introdução do trabalho, seus objetivos e justificativa. O capítulo 2 apresenta o referencial teórico utilizado para a elaboração do trabalho. No capítulo 3, é apresentado o tema da aplicação e as funcionalidades do sistema. No capítulo 4 são citadas as ferramentas utilizadas e as fases de desenvolvimento da aplicação, especificando o caminho percorrido e os problemas encontrados em cada fase. No capítulo 5 é apresentada a análise das métricas obtidas na aplicação. O capítulo 6 contém a conclusão sobre a ferramenta analisada. No apêndice A encontra-se um

guia do caminho percorrido para obtenção de uma aplicação móvel utilizando o JavaFX e o *plugin* Gluon Mobile

2 REFERENCIAL TEÓRICO

Para poder desenvolver um sistema utilizando o *framework* JavaFx é necessário compreender alguns conceitos e tecnologias importantes, apresentando os conceitos de: Programação móvel, Java e JavaFX, Gluon, ferramentas de desenvolvimento e métricas de análise de desenvolvimento.

2.1 Programação móvel

A programação móvel trata do desenvolvimento de aplicativos para dispositivo móvel, na literatura, pode ser dividido em três categorias sendo elas: nativa, híbrida e *web* (PRADO; SILVA, 2019).

Na aplicação nativa o desenvolvedor utiliza a linguagem do próprio sistema operacional, essa característica possibilita uma maior confiabilidade para o usuário, muitas vezes com melhor desempenho (PRADO; SILVA, 2019).

Apesar de levar o nome de aplicação *web*, não se trata de um aplicativo, é um *site* que disponibiliza uma formatação adequada para dispositivos móveis. Suas características positivas são de não necessitar de *download* e instalação e ser utilizável por outros sistemas operacionais, entretanto, normalmente necessita do uso constante de *internet*, são mais lentos e normalmente não possuem acesso a recursos nativos (PRADO; SILVA, 2019).

Semelhante a aplicação *web*, existe a aplicação híbrida, que se difere pelo *site* estar contido em uma aplicação, desse modo facilita o acesso a recursos nativos do sistema sem perder as características de um *site* que pode ser executado em diversos sistemas operacionais (PRADO; SILVA, 2019).

2.2 JavaFX

Desde suas primeiras versões o Java contava com duas bibliotecas para interface gráfica, mas em 2007 a Sun Microsystems revela uma nova plataforma denominada JavaFX, que abriu possibilidades de criar interfaces ricas, contendo diversos recursos para aprimorar a experiência do usuário (SRIVASTAVA, 2016). Inicialmente pertencia a empresa SeeBeyond com o nome de F3 (*Form follows function*), que possuía uma linguagem própria de *script* (DEA et al., 2017).

Mais tarde com a compra da Sun Microsystems pela Oracle, anunciaram planos de modificar a plataforma para integrar com o Java, culminando na criação do

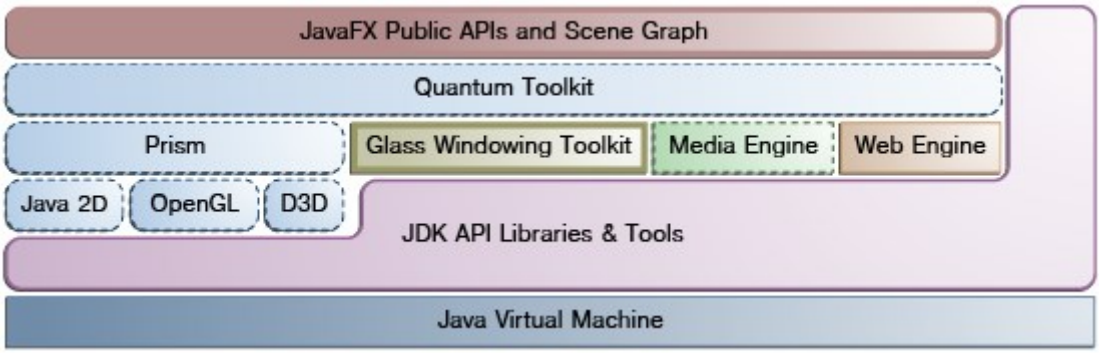
JavaFX 2.0, apresentado na conferência JavaOne de 2011 (DEA et al., 2017). Nesse mesmo evento, a Oracle se comprometeu a disponibilizar o código para a comunidade, com a ideia de que o *software* cresceria mais rapidamente e melhoraria seu suporte (DEA et al., 2017).

Em 2013, Johan Vos reuniu uma equipe e deu início ao projeto JavaFXPorts que tinha como objetivo portar a plataforma Java juntamente ao JavaFX, para Android (DEA et al., 2017). Ao mesmo tempo, a RoboVM trabalhava em um projeto que tinha o mesmo objetivo, porém para realizar a integração na plataforma iOS (DEA et al., 2017). Visando unificar os projetos, no ano de 2015 foi criada a organização Gluon que buscava centralizar esforços de toda a comunidade e, atualmente, disponibiliza integrações do Java para sistemas móveis, em nuvem e embarcados (DEA et al., 2017).

2.2.1 Arquitetura e Características

Na Figura 1, é apresentada a arquitetura e características de funcionamento do JavaFX dividida em camadas de recursos visuais e integração com sistemas operacionais.

Figura 1 - Arquitetura JavaFX



Fonte: Castillo (2013).

O primeiro bloco da Figura 1 (*Scene Graph*), representa todos elementos que compõem a cena, são divididos em nós, onde um ou mais elementos integram o elemento pai (CASTILLO, 2013).

Ainda na mesma camada a ferramenta fornece um conjunto de APIs (*Application Programming Interface*) que suportam o desenvolvimento de aplicações ricas e amplo acesso a recursos da plataforma Java (CASTILLO, 2013).

Os blocos abaixo indicados como *Quantun Toolkit* e *Prism*, são sistemas responsáveis por renderizar as cenas a partir de recursos do *software,* quando o

hardware disponível é incapaz. Essa camada é responsável pela exibição de gráficos 2D e 3D, se adaptando com base no dispositivo utilizado (CASTILLO, 2013).

A camada mais inferior dos gráficos denominada *Glass Windowing Toolkit* é responsável por fornecer serviços que integram a aplicação ao sistema operacional onde está sendo executada, gerencia recursos como a fila de execução de eventos e janelas (CASTILLO, 2013).

Midia Engine e *Web Engine* são camadas de abstração que permitem o uso de recursos de multimidia e *web* respectivamente, permitindo integrar áudio, imagem, vídeo, páginas *web* completas e aplicação de estilos com CSS (*Cascading Style Sheets*) (CASTILLO, 2013).

O diagrama apresenta também a camada JDK API Libraries & Tools, que contém recursos que permitem ampliar o JavaFX por meio de APIs e ferramentas para integrar maiores funcionalidades e recursos atualizados (CASTILLO, 2013).

2.2.2 *Scene Builder*

O *Scene Builder* é um *software* de auxílio a criação de telas, atualmente disponibilizado pela Gluon e introduzido no mercado a partir de 2012, sua principal funcionalidade é gerar arquivos de tela ".fxml" para serem utilizados em aplicações com JavaFX.

A característica principal do software é a capacidade de produzir telas a partir do recurso de clicar e arrastar que facilitam a visualização da tela em tempo de criação. Fornece também a capacidade de gerenciar variáveis e propriedades referentes aos componentes contidos nas telas.

2.2.3 FXML

O FXML (*FX Markup Language*) é uma linguagem de marcação para a criação de interfaces, baseada na linguagem XML (*eXtensible Markup Language*), foi desenvolvida pela Oracle para construir a interface do usuário em uma aplicação JavaFX (ORACLE, 2008-2014).

O uso dessa linguagem permite melhor separação entre a interface gráfica e a lógica do programa, se destaca principalmente por ser utilizada pelo Scene Builder como arquivo de saída para cada tela criada (ORACLE, 2012-2014).

Na Figura 2, é apresentado um exemplo de código de um arquivo FXML, em seguida na Figura 3, é apresentado o resultado gerado a partir do código.

Figura 2 - Exemplo de código FXML

```
<?import com.gluonhq.charm.glisten.mvc.View?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.BorderPane?>


<View maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
      prefHeight="600.0" prefWidth="335.0" xmlns="http://javafx.com/javafx/18" xmlns:fx="http://javafx.com/fxml/1">
   <center>
      <Button mnemonicParsing="false" text="Hello World!" BorderPane.alignment="CENTER" />
   </center>
</View>
```

Fonte: Autoria própria.

Figura 3 - Visualização FXML

Fonte: Autoria própria.

2.3 Gluon

A Gluon é uma organização mantida por diversos desenvolvedores da comunidade, tendo como fundadores, especialistas reconhecidos pela indústria de *software*, principalmente desenvolvedores Java, que desde 2015 vem integrando e atualizando tecnologias que permitem o uso do JavaFX em dispositivos como Android, IOS e sistemas embarcados (SRIVASTAVA, 2016).

A fim de facilitar a criação de aplicativos utilizando essa plataforma, foi criada uma biblioteca chamada Gluon Mobile, na qual existe um conjunto de abstrações do *hardware*, denominadas como *attachs*, que permitem acessar os recursos nativos dos dispositivos móveis, essa biblioteca permite também a execução em dispositivos de forma nativa priorizando o desempenho (GLUON, 2019).

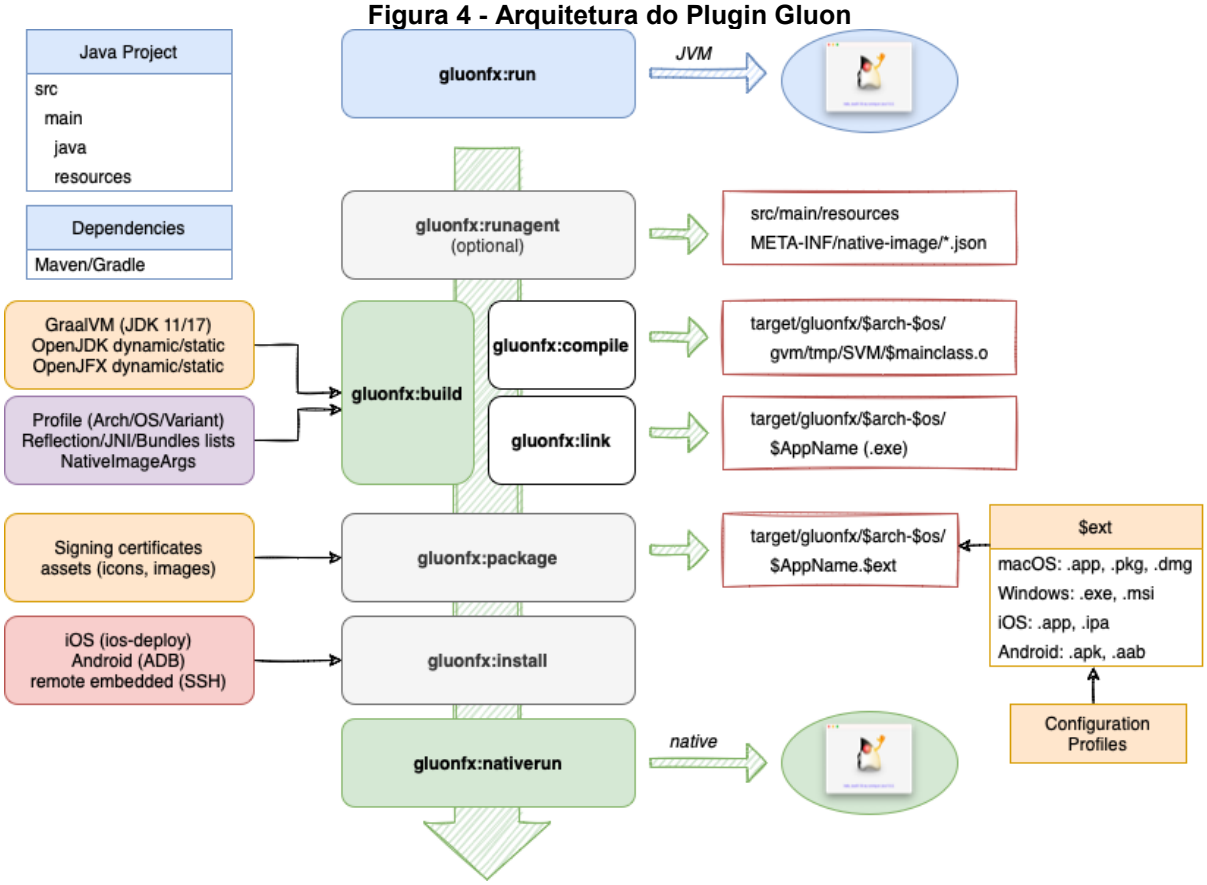
O *Plugin* disponibilizado pela Gluon consiste em um recurso que fornece estruturas bases como ponto de partida para iniciar projetos com o Gluon Mobile, sendo mantido para as principais IDEs (*Integrated Development Environment*) Java (IntelliJ, Apache NetBeans e Eclipse) (GLUON, 2022).

Atualmente disponibiliza quatro tipos de projetos que podem suprir necessidades iniciais para aplicações, são eles: *single view, multi view, multi view with fxml (fx markup language), multiple view with glisten afterburner.* Por meio do plugin é

possível escolher qual será o gerenciador do projeto, suportando os principais Maven e Gradle (GLUON, 2022).

Na Figura 4 é apresentado o diagrama de funcionamento do plugin Gluon, sua estrutura principal contempla rotinas responsáveis pela criação, compilação, empacotamento e execução da aplicação.

Figura 4 - Arquitetura do Plugin Gluon



Fonte: Gluon (2022).

As rotinas apresentadas, utilizam bibliotecas e ferramentas auxiliares quando executadas. O caso do “gluonfx:build” faz uso do GraalVM, arquivos de tradução dos componentes JavaFX e outros arquivos de configuração do projeto, para gerar um código nativo. O “gluonfx:package” é responsável por unir as dependências do projeto em um único pacote (GLUON, 2022).

2.4 GraalVM

O GraalVM trata-se de uma versão modificada do *kit* de desenvolvimento do Java, esse pacote tem grande importância no desenvolvimento de aplicativos com Gluon, pois sua aplicação é o que possibilita a conversão dos arquivos binários em Java para código nativo de diversas plataformas, permitindo executar compilações

tanto para a máquina virtual Java, quanto para execução nativa em dispositivos específicos (GRAALVM, 2022).

Este *kit* de desenvolvimento é estendido a partir do OpenJDK, e adiciona recursos que visam otimizar o desempenho, oferecendo uma inicialização mais rápida e um menor espaço na memória das aplicações em imagem nativa (GRAALVM, 2022).

A versão utilizada no projeto trata-se de uma extensão exclusiva para o desenvolvimento com o *plugin* Gluon, atualmente são oferecidas duas versões, uma construída para funcionar com o Java 11 e outra para o Java 17, ambas funcionam nas plataformas Windows, Linux e MacOS.

2.5 Maven

A ferramenta Maven trata-se de um gerenciador de projeto que facilita o desenvolvimento e auxilia o entendimento de projetos, surgindo a partir da necessidade de se ter uma maneira padrão, clara e de fácil compartilhamento para criar projetos. A Maven foi criada especificamente para a linguagem Java e abrange desde a etapa de construção do projeto até sua documentação (MAVEN, 2022).

2.6 Ferramentas de desenvolvimento Java

Dentre os ambientes de desenvolvimento existentes para programar na linguagem Java, existem três que são as mais usadas, são elas: Eclipse, Apache Netbeans e IntelliJ.

A IBM lançou o Eclipse no ano de 2001, além do Java esta ferramenta oferece suporte para outras linguagens como C, C++, Python dentre outras. Esta ferramenta tem como principal característica o funcionamento baseado em *plugins*, é por meio deles que possibilitam o acréscimo de funcionalidades (ANDRADE, 2021).

O NetBeans foi lançado em 2000 pela Sun Microsystems, inicialmente desenvolvido como um *software* proprietário, adquirido em 2010 pela Oracle, no ano de 2016 iniciou um novo projeto na incubadora Apache, tornando-se assim de código aberto e passando a se chamar Apache NetBeans. Ele oferece suporte para criação de interfaces em aplicações *web*, *desktop* e *mobile*, além de dar melhor suporte ao Java por fazer parte do seu ecossistema (ANDRADE, 2021).

Em 2001 foi lançada a primeira versão do IntelliJ desenvolvida pela JetBrains, que ganhou destaque por conter diversos recursos nativos, foi construída em Java e

fornece suporte para as principais linguagens existentes no mercado. Possui versão gratuita e paga, ambas permitem o desenvolvimento comercial, porém a versão paga possui suporte para mais linguagens de programação assim como algumas ferramentas adicionais (ANDRADE, 2021).

2.7 Métricas de análise de desenvolvimento de *software*

Para que seja possível avaliar o desenvolvimento da aplicação com JavaFX, deve-se definir métricas que quantificam algumas características relevantes do *software*. Algumas métricas destacadas são:

- Linhas de código: refere-se à quantidade de linhas escritas no código da aplicação e podem significar o esforço necessário durante o desenvolvimento e a facilidade para realizar manutenções no código. No entanto, tal métrica pode não ser muito precisa, pois pode haver divergência na forma como cada linguagem de programação é estruturada (GONSALVES, 2019; WILSON, 2010; DUFOUR et al., 2002).
- Quantidade de arquivos: trata-se de uma contagem simples de arquivos, Gonsalves (2019) opta por contar somente arquivos criados pelo desenvolvedor, considerando que também sofre influência da estrutura de cada linguagem. Observa-se que ao mesmo tempo que muitos arquivos são difíceis de gerenciar, poucos arquivos tornam as delimitações do projeto confusas.
- Dependências: é uma contagem simples de bibliotecas externas necessárias para o funcionamento da aplicação, apesar de possibilitar a adição de diversas funcionalidades, seu uso em excesso dificulta o gerenciamento do *software*, podendo causar erros ou falhas de segurança que o desenvolvedor não tenha conhecimento (GONSALVES, 2019).
- Tamanho do pacote da aplicação e rastro de instalação: significam o tamanho da aplicação antes e depois de sua instalação, respectivamente. A diferença na capacidade de armazenamento em dispositivos móveis, faz com que os usuários optem por aplicativos menores (GONSALVES, 2019; CHHABRA; GUPTA, 2010).

- Perfil de uso da memória: indica quanto a aplicação gerada usa de memória RAM (*Random access memory*) durante seu ciclo de vida (GONSALVES, 2019; CHHABRA; GUPTA, 2010).
- Tempo de inicialização da aplicação e tempo de transição de tela: de acordo com Gonsalves (2019) é um dos parâmetros mais perceptíveis para o usuário, pois indica o tempo entre interação do usuário e resposta visual do aplicativo.
- Estilo e elementos da interface do usuário: a capacidade da ferramenta de adaptar e fornecer os recursos visuais de diferentes plataformas de forma coerente, para tornar a experiência do usuário mais natural (GONSALVES, 2019).
- Curva de Aprendizado: indica os conhecimentos necessários para utilização do *framework*. Esse fator impacta principalmente no custo de profissionais e tempo de treinamento (GONSALVES, 2019).
- Comportamento de recarga em tempo real: contribui diretamente para o tempo de desenvolvimento, considerando que durante o processo é necessário a rápida visualização do que está sendo feito e de como o *software* se comportaria em um dispositivo (GONSALVES, 2019).
- Acesso ao sistema de arquivos e *hardware* do dispositivo: trata-se da capacidade que a linguagem tem de acessar os recursos nativos de um dispositivo móvel (GONSALVES, 2019).
- Portabilidade: refere-se à capacidade de um mesmo código ser funcional em sistemas operacionais distintos, métrica que também favorece o custo de uma aplicação, já que uma mesma equipe de desenvolvedores atende a diversas plataformas com esforço mínimo (GOMES FILHO, 2005).
- Compilação e desenvolvimento da aplicação: classifica o quão simples a estrutura permite a obtenção do produto, que no caso dos dispositivos móveis refere-se a pacotes instaláveis (GONSALVES, 2019).

Para o estudo de caso apresentado, coletaram-se todas as métricas citadas, a fim de obter aspectos distintos e assim realizar uma análise mais completa sobre a ferramenta.

3 ESTUDO DE CASO

Para aprofundar os estudos referente à ferramenta proposta, selecionou-se um modelo de estudo de caso. Para tal estratégia, um *software* foi elaborado a fim de obter uma visão prática das vantagens e desvantagens da ferramenta.

Neste capítulo é abordado o tema da aplicação desenvolvida e seus requisitos funcionais e não funcionais.

3.1 Aplicação

Visando a utilização de grande parte dos recursos disponibilizados pela ferramenta, desenvolveu-se uma aplicação para gerenciamento de estoque, pois possibilita a utilização de recursos nativos, persistência de dados e formulários permitindo uma avaliação aprofundada de suas funcionalidades.

A escolha da aplicação teve origem na necessidade de uma empresa em encontrar um software simples, mas que atendessem os processos existentes. A operação consiste na venda de produtos em uma loja física, considerando vendas no atacado e varejo. Outra parte importante da operação é a venda externa de produtos no atacado, utilizando um veículo para a visita de clientes.

3.2 Funcionalidades do Sistema

A aplicação deve atender os seguintes requisitos funcionais:

- Deve possibilitar o cadastro de transações permitindo indicar uma pessoa e produtos pertinentes a instância da transação;
- Deve possibilitar o cadastro, remoção e atualização de usuários e produtos que serão usados nas transações;
- Permitir a utilização da câmera do dispositivo para facilitar a leitura do código de barras, através do *attach* disponibilizado pelo Gluon;
- Permitir filtrar as transações em um período definido: separar dados de acordo com uma classificação;
- A partir da lista de transações deve-se calcular o total dos itens exibidos, recurso que deve ser interligado com o filtro.

Existem também alguns requisitos importantes para um bom funcionamento da aplicação, mas que não estão ligadas diretamente a etapa de desenvolvimento, e

sim as restrições das funções oferecidas (SOMMERVILLE, 2011). Os requisitos não funcionais definidos são:

- Requisito de portabilidade: como proposta principal deve executar ao menos no *desktop* e Android;
- Requisito de interoperabilidade: o sistema deverá persistir os dados por meio de um banco de dados ou arquivos;
- Requisitos de implementação: o sistema deverá ser desenvolvido na linguagem Java, utilizando *framework* JavaFX para interface e o *plugin* Gluon Mobile;
- Requisitos de eficiência: o sistema deverá responder as ações do usuário instantaneamente.

4 PROCESSO DE DESENVOLVIMENTO DA APLICAÇÃO

Neste capítulo relata-se o processo entorno do desenvolvimento da aplicação de estudo de caso, abordando quais as ferramentas utilizadas e como configurá-las, além de tratar sobre o caminho percorrido e os problemas enfrentados em cada fase do processo.

4.1 Ferramentas utilizadas

O projeto foi desenvolvido utilizando a IDE disponibilizada pela empresa JetBrains, o IntelliJ (INTELLIJ, 2022). O critério para escolha foi a facilidade de interação com a interface, por exibir apenas os recursos necessários.

Atualmente, o desenvolvimento de aplicações móveis utilizando o JavaFX torna-se mais viável com a utilização dos meios disponibilizados pela Gluon, logo a ferramenta mais importante é o *plugin* da Gluon específico para a IDE utilizada (GLUON, 2022).

Para facilitar e agilizar o desenvolvimento das telas da aplicação, utilizou-se a ferramenta Scene Builder, que possibilita a manipulação de componentes gráficos, através de interações de “clica e arrasta” (GLUON, 2022). A sua integração ao código é feita com um arquivo FXML gerado, dessa forma proporciona uma melhor divisão entre código de lógica e de *design*.

4.2 Fases do desenvolvimento

O desenvolvimento foi dividido em fases para melhor apresentar o caminho a percorrer e as dificuldades encontradas nesses caminhos. As fases definidas foram as seguintes:

- Preparação do ambiente de desenvolvimento;
- Implementação das interfaces gráficas;
- Configuração e criação da base de dados;
- Utilização de recursos nativos.

Nas seções seguintes, são apresentadas as fases de desenvolvimento divididas em subseções, contendo orientações do caminho funcional seguido dos problemas encontrados na fase.

4.3 Fase: Preparação do Ambiente de Desenvolvimento

Nessa fase são descritos como o ambiente de desenvolvimento foi preparado, o sistema operacional necessário, os recursos adicionais e a correta instalação e configuração da IDE para iniciar os trabalhos.

4.3.1 Caminho percorrido

Os primeiros passos seguiram a partir da documentação da Gluon, que devido a necessidade de cobrir diversas possibilidades para o desenvolvimento, tornou-se complexa ao considerar muitas variáveis, como por exemplo, diferentes sistemas operacionais e diversos caminhos a serem seguidos. Para evitar problemas entre as possíveis configurações, neste projeto, selecionou-se um caminho mostrando como obter uma aplicação funcional.

No caminho escolhido, foi utilizado como sistema operacional o Linux que é obrigatório para gerar um pacote instalável para Android.

Para construir um programa em código nativo é necessário a compilação Gluon do GraalVM Java 11, que serve como pacote de desenvolvimento da linguagem, nele já contém o JDK (Java *development kit*), compiladores para linguagem nativa de diversas plataformas e ferramentas que permitem programar em mais de uma linguagem.

Como o ambiente utilizado é o Linux, é necessário conceder permissões de execução ao arquivo “mvn” dentro da instalação da IDE, em “plugins/maven/lib/maven3/bin”.

No ambiente Linux, são necessárias mais algumas ferramentas, no geral são bibliotecas adicionais que podem ser obtidas a partir de um terminal utilizando os comandos “sudo apt update”, “sudo apt install build-essential” e “sudo apt install adb”.

A integração entre as ferramentas se dá pela IDE escolhida, e a partir dela deve ser instalado o respectivo *plugin* do Gluon, no caso do IntelliJ, a página inicial do programa já oferece acesso a guia *plugins*, onde pode ser baixado junto com algumas dependências.

Como recurso auxiliar para a construção da interface gráfica, o aplicativo Scene Builder pode ser instalado e, geralmente, agiliza o processo de criação, já que fornece meios de construir e visualizar as telas de forma rápida, além de facilitar a construção de visualizações dinâmicas.

Com os requisitos básicos já adquiridos, pode-se iniciar um novo projeto na categoria Gluon utilizando o GraalVM como JDK. Também é necessária a criação de uma variável do sistema denominada GRAALVM_HOME, indicando o caminho de seu diretório.

Detalhes sobre os passos a serem seguidos são encontrados de forma prática no apêndice A.

4.3.2 Problemas na configuração do ambiente

O maior problema encontrado está relacionado com a versão do GraalVM para Java 17, que atualmente não consegue reunir as dependências na geração de um arquivo “apk”. A solução provisória é a utilização do GraalVM Java 11.

Um problema encontrado é citado na observação da documentação: trata-se da falta de permissão de execução do arquivo “mvn”. Isso se deve a forma que o Linux gerencia as permissões, logo deve-se habilitá-la manualmente, dado que a instalação padrão tem essa opção desabilitada.

Outro problema foi identificado após a tentativa de compilar o projeto, na qual foi gerado um registro indicando a falta da biblioteca “stdio.h”. A instalação do pacote “build-essential” soluciona o problema, já que contém as bibliotecas necessárias.

Na etapa de instalação do aplicativo em um dispositivo móvel ocorre a falha de comunicação, o registro acusa a falta do *plugin* ADB (Android *debug bridge*), porém o mesmo deveria ser instalado pelo próprio *plugin* do Gluon (GLUON, 2022). A instalação externa do *plugin* resolve o problema.

4.4 Fase: Implementação das interfaces gráficas

Nesta fase é apresentada a implementação das interfaces, que é o processo em que as visualizações do aplicativo são construídas.

4.4.1 Caminho percorrido

Para realizar a construção das interfaces utilizou-se a ferramenta Scene Builder (SCENE BUILDER, 2022), que permite a manipulação de objetos gráficos de forma interativa e dinâmica.

Primeiramente criou-se a tela de *login*, mostrada na Figura 5, a qual serviria para controlar a entrada de usuários, realizando a validação de *login* e senha no banco

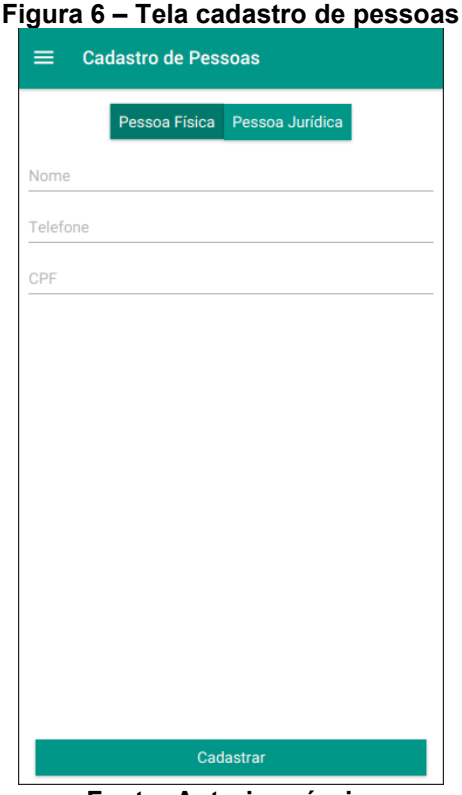
de dados. Porém, devido aos problemas encontrados na integração de um banco de dados, apresentados na subseção 4.5.2, esta tela tornou-se um controle do próprio usuário para proteger dados críticos da aplicação.

Figura 5 - Tela de *login*

Fonte: Autoria própria.

Em seguida foram criadas as telas de cadastro, sendo a primeira o cadastro de pessoas (Figura 6), a qual permite que sejam realizados os cadastros de pessoa física e jurídica. Os cadastros realizados servem à aplicação para vincular os clientes às transações realizadas, além de manter os registros dos clientes.

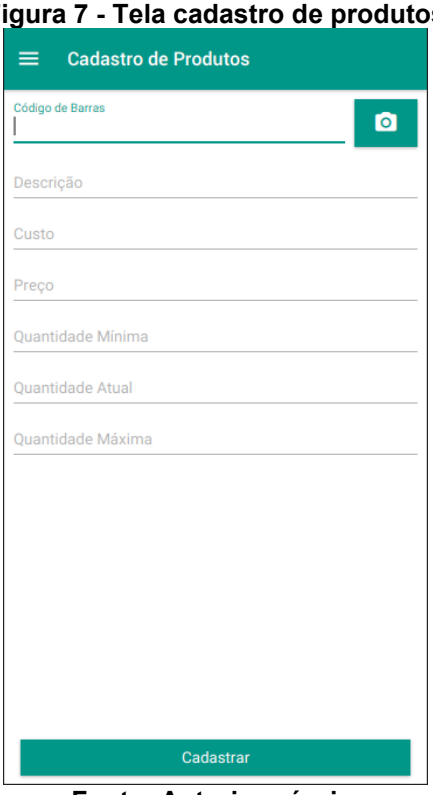
Figura 6 – Tela cadastro de pessoas



Fonte: Autoria própria.

Em seguida construiu-se a tela de cadastro de produtos, nela são inseridos os dados mais relevantes relacionados a ele, como o código de barras, a descrição, o custo, o valor, as quantidades de estoque mínima, atual e máxima, o resultado é mostrado na Figura 7.

Figura 7 - Tela cadastro de produtos



Fonte: Autoria própria.

A última tela de cadastro criada foi a de transação, apresentada na Figura 8, nela são inseridas a data, a hora, o tipo da transação, podendo ser de entrada ou saída, o cliente e a lista de produtos envolvidos no processo. Além dessas informações, também são exibidos os valores do subtotal e do total, que é calculado considerando a inserção de um desconto.

Figura 8 - Tela cadastro de transação

Fonte: Autoria própria.

Após a construção das telas de cadastro, seguiu-se para as telas de lista, cada uma apresentando a lista referente a sua entidade, apresentadas na Figura 9, como: pessoas, produtos e transações da esquerda para a direita. As que exibem os produtos e as pessoas, permitem que os registros sejam excluídos ou editados. Já a transação permite apenas a sua visualização. Para acessar a operação de cadastro encontra-se um botão flutuante que direciona para a respectiva tela.

Figura 9 - Telas de listagem

Fonte: Autoria própria.

4.4.2 Problemas encontrados

O Gluon trata-se de um projeto em desenvolvimento na questão multiplataforma, fazendo com que recursos de conversão do arquivo gerado pelo Scene Builder, não sejam capazes de interpretar todos os elementos fornecidos pelo *framework*. Como consequência, diversos elementos fundamentais da tela devem ser implementados diretamente em código, fazendo com que não seja possível utilizar toda a capacidade do Scene Builder. Outra alternativa oferecida por um desenvolvedor do Gluon é adicionar o componente nas dependências do Maven, alternativa que impacta no tempo de construção do executável do projeto (PEREDA, 2020).

Com a adição de dispositivos móveis à plataforma, foram implementados pela equipe do Gluon alguns componentes com funcionalidades extras, mas que durante o seu uso apresentaram recursos incompletos, dentre eles destaca-se:

- A falta de propriedades do componente *appBar* acessíveis a partir do Scene Builder;
- O componente *floatButton* não é adicionável pelo Scene Builder;
- Não existe um método *setValue()* no *autoCompleteTextField*.

Esses e outros detalhes apresentados são justificáveis pela etapa de desenvolvimento em que o projeto se encontra, mas atrapalha na adesão da ferramenta já que é necessário encontrar alternativas para cada detalhe que se deseja implementar.

4.5 Fase: Configuração e criação da base de dados

Essa etapa trata o formato e estratégias empregadas na persistência de dados, assim como os problemas encontrados, suas causas e soluções.

4.5.1 Caminho percorrido

A primeira estratégia indicada pelo Gluon trata-se de um *plugin* proprietário denominado CloudLink, a princípio é colocado de lado pois sua página indica custos que fogem do código aberto, devido suas funcionalidades na nuvem. Porém, é possível utilizar alguns de seus recursos de forma local, permitindo o acesso ao armazenamento tanto de dispositivos móveis quanto de *desktops*.

O *plugin* abstrai o acesso ao armazenamento com funções simples que salvam dados em arquivos e os recuperam quando necessário em um formato de lista observável próprio, esse formato possibilita manter os dados atualizados sem a necessidade de instruções de atualização.

Apesar de ser um recurso de persistência, da forma como é usado não inclui um gerenciador de banco de dados, o que limita o uso eficiente de espaço, bem como outros recursos facilitadores para manipulação de dados.

4.5.2 Problemas encontrados

Antes de alcançar a solução final adotada no projeto, foram realizados testes com outras alternativas para integração com banco de dados, a fim de utilizar uma solução mais adequada e mais popular no meio de desenvolvimento.

A primeira tentativa foi a utilização da biblioteca SQLite, que se trata de um mecanismo simplificado de banco de dados SQL (*Standard query language*), útil para armazenar arquivos localmente.

Para sua implementação utilizou-se uma documentação antiga fornecida pelo Gluon, mas não anexada na documentação principal, a mesma informa para adicionar dependências necessárias ao projeto e implementar uma classe para a criação e

acesso ao banco de dados, além de configurações manuais no arquivo “AndroidManifest.xml” para fornecer permissões ao aplicativo.

A aplicação obteve um bom funcionamento, quando testada no *desktop*, no entanto, no sistema móvel houveram problemas de acesso ao armazenamento interno. A solução mais recente apresentada pela comunidade é o uso da ferramenta CloudLink, como apresentado na subseção anterior contém recursos pagos.

O problema geral consiste na falta de um *driver* atualizado que atenda a funcionalidade de multiplataforma. Apesar de outras ferramentas para desenvolvimento móvel implementarem *drivers* funcionais, a aplicação com Gluon não obteve o mesmo resultado.

Outra alternativa testada foi a utilização da plataforma móvel do Google, o Firebase, que permite o armazenamento de dados em nuvem. O problema se trata novamente das dependências, porém desse modo não chegam a ser reconhecidas dentro do projeto.

4.6 Fase: Utilização de recurso nativo

Nesta fase são descritos quais recursos nativos foram utilizados na aplicação e de que forma são abstraídos pela Gluon.

4.6.1 Caminho percorrido

Para ter acesso aos recursos nativos do dispositivo móvel, a Gluon disponibiliza um componente denominado Gluon Attach, responsável por abstrair o *hardware* independentemente do dispositivo (GLUON, 2019).

Logo na criação da aplicação são adicionados alguns *attachs* padrões que contém recursos obrigatórios de uso interno, são eles:

- *Display*: Fornece acesso às informações da tela do dispositivo (GLUON, 2019);
- *Lifecycle*: Traz eventos relacionados à aplicação referente a seu estado como *pause*, *resume* e *start* (GLUON, 2019);
- *Statusbar*: Permite acessar a barra de *status* do dispositivo (GLUON, 2019);
- *Storage*: Fornece o acesso do armazenamento do dispositivo (GLUON, 2019).

No estudo de caso foi aplicado um recurso que utiliza a câmera do dispositivo como leitor de código de barras, expandindo a funcionalidade da aplicação e permitindo testar a utilidade e facilidade por dentro do conceito de *attach*.

A aplicação do recurso envolve a criação de um botão que solicita o uso do recurso, quando acionado, disponibiliza a câmera até encontrar um código de barra, que quando encontrado, uma função de leitura retorna uma *string* com o valor lido.

Especificamente para o código de barras é necessário adicionar configurações no arquivo "AndroidManifest.xml", elas são responsáveis por solicitar permissões de uso da câmera ao usuário e também configurar algumas características do serviço.

4.6.2 Problemas encontrados

O uso repetido do recurso implementado causa problemas na aplicação até mesmo encerrando-a inesperadamente, o problema trata-se da falha de comunicação entre o serviço e a aplicação. Existem problemas semelhantes relatados na página principal do Github (GITHUB, 2022), mas até o momento não possuem solução.

Atualmente, o Gluon não faz alterações automaticamente no arquivo “AndroidManifest.xml” relacionado a plataforma Android, isso cria a necessidade de realizar configurações manuais que custam tempo no desenvolvimento.

4.7 Desenvolvimento desktop

As etapas demonstradas nas subseções anteriores demonstram o desenvolvimento voltado a uma aplicação para dispositivos móveis, porém seguindo o conceito da linguagem Java de se escrever uma vez e rodar em qualquer lugar, aproveitou-se o mesmo código fonte para obter-se um executável para a versão *desktop*.

O processo se difere na necessidade de obter pacotes de desenvolvimento adicionais no caso da plataforma Linux utilizado, podem ser adquiridos a partir dos comandos mostrados na Figura 10 utilizando um terminal.

Figura 10 - Pacotes para construção no Linux

```
apt-get install libasound2-dev libavcodec-dev
    libavformat-dev libavutil-dev libfreetype6-dev

apt-get install libgl-dev libglib2.0-dev
    libgtk-3-dev libpango1.0-dev
    libx11-dev libxtst-dev zlib1g-dev
```

Fonte: Autoria própria.

Ao final pode-se obter os mesmos resultados de um dispositivo móvel sem alterações no código fonte. As peculiaridades de cada dispositivo são automaticamente adaptadas para seu ambiente de execução. Exemplos que se destacam são os diferentes métodos de interação, como teclado, *mouse*, *touchscreen* e diferentes tamanhos de tela.

5 ANÁLISE

Este capítulo de análise é focado nas características do desenvolvimento da aplicação, usando as métricas selecionadas como guia e aplicando-as no estudo de caso.

5.1 Quantidade de Linhas de código, arquivos e dependências

Para captura dos valores, foi utilizado um *plugin* para a IDE IntelliJ chamado Statistic, que retorna o total de linhas de código dos arquivos selecionados e também a quantidade de arquivos que cada pacote contém. Em relação as dependências, foi realizada a contagem manual das bibliotecas importadas a partir de um arquivo gerado pelo *plugin* Gluon.

Foram criadas aproximadamente 1900 linhas de código, que são distribuídas em 31 arquivos, mas destaca-se que grande parte da construção do projeto está voltada a arquivos padrões, criados automaticamente pelo *plugin* da Gluon, esses arquivos não são comumente contabilizados em tais métricas.

Com relação as dependências o projeto utilizou um total de 31 bibliotecas externas, essa quantidade indica que a proposta do reuso de código é atendida, porém levanta alguns problemas de compatibilidade entre versões dessas dependências, tornando alguns recursos obsoletos e desatualizados, como citado na seção 4.3 e 4.5, no caso do Graal em sua versão 17 e a incompatibilidade do SQLite respectivamente.

5.2 Tamanho do pacote da aplicação e rastro de instalação

Para a coleta desses valores foram utilizados recursos nativos do Android, sendo o gerenciador de arquivos responsável pelo tamanho do arquivo antes da instalação e o gerenciador de aplicativos responsável pelo tamanho real da aplicação após instalação.

O pacote da aplicação gerado ao final do projeto obteve um tamanho de 33,6MB, que é relativamente alto considerando a baixa quantidade de funcionalidades e telas que a aplicação possui. Após a instalação da aplicação o espaço utilizado se torna ainda mais significativo, ficando em torno de 135MB, com isso pode-se perceber que o rastro na memória de armazenamento deixado pela aplicação desenvolvida foi considerável.

Para garantir a exatidão dos resultados obtidos, foram analisadas aplicações de exemplos fornecidos pela Gluon, tratam-se de aplicações simples com o objetivo de apresentar funcionalidades da ferramenta, porém com suas execuções foram obtidos número próximos da aplicação do estudo de caso, no que se refere ao uso de armazenamento.

Com mais amostras de aplicações analisadas, pode-se afirmar com mais assertividade que as aplicações desenvolvidas com esta ferramenta utilizando as dependências indicadas gera um produto que requer uso maior de armazenamento do que aplicações que utilizam outras tecnologias.

5.3 Perfil de uso da memória

O valor de memória foi obtido com o aplicativo Resource Monitor Mini, que rastreia o valor bruto de memória livre de todo o sistema e não só da aplicação, apesar de mais simples, não interfere no uso do dispositivo fornecendo valores mais precisos.

No uso da memória RAM a aplicação do estudo de caso obteve um consumo maior do que 200MB no dispositivo móvel, o mesmo resultado foi obtido para aplicações mais simples de exemplo. Ao decorrer do uso da aplicação a memória é gerenciada de forma aceitável, já que sua utilização aumenta consideravelmente, mas volta a reduzir quando a aplicação fica ociosa.

5.4 Tempo de inicialização da aplicação e tempo de transição de tela

Nesta etapa não foram obtidos valores numéricos, dado que o tempo de espera tanto para inicialização quanto para transições são totalmente imperceptíveis.

Quanto ao tempo de resposta e execução da aplicação, não foram encontradas divergências em comparação com outras tecnologias conhecidas. Esse comportamento já era esperado, dado que a construção final da aplicação é realizada em código nativo da plataforma que irá executá-la.

5.5 Estilo e elementos da interface do usuário

A análise desse parâmetro baseia-se na comparação entre os recursos que o JavaFX oferece e o que realmente foi possível usar na aplicação.

Os elementos gráficos estão ligados diretamente aos recursos do JavaFX, esperava-se que fosse a parte mais funcional da adaptação do Gluon, porém

apresentaram muitos problemas relacionados a execução em dispositivos móveis, causados por componentes incompletos ou mal adaptados.

A Gluon oferece alguns componentes que tem sua codificação estendida dos utilizados para a plataforma desktop, como exemplo o textField, no qual implementam a propriedade floatText, que permite inserir um texto sobre o elemento para identifica-lo, reduzindo o espaço ocupado em tela.

Em contrapartida, peca por deixar de lado funcionalidades comuns em dispositivos móveis, como a possibilidade de exibir um menu de contexto a partir da seleção de textos ou caixas de entrada, ou de aceitar caracteres desconhecidos do padrão dele.

Com relação aos ícones comuns no ambiente móvel, a ferramenta disponibiliza uma boa quantidade, apesar que, durante o desenvolvimento da aplicação do estudo de caso faltaram algumas opções.

Outra dificuldade acrescentada, é que as propriedades e componentes nativos do JavaFX não estão devidamente completas, fazendo com que funcionem quando executadas na máquina virtual, mas apresentem erros quando em dispositivos móveis. Algumas propriedades apresentam possibilidade de resolver manualmente ao custo de um tempo maior na construção dos códigos binários, já outras não apresentam nenhum tipo de solução.

5.6 Curva de aprendizado

A curva de aprendizado da ferramenta é alta para programadores que conhecem a linguagem Java, isso porque a plataforma JavaFX utiliza a mesma linguagem e a estrutura gerada em FXML é de fácil entendimento, caso seja necessária uma alteração manual no arquivo.

Apesar de ser uma tecnologia já conhecida, traz muitas diferenças relacionadas as modificações do Gluon, fazendo com que estratégias de programação utilizadas devam ser adaptadas, tais adaptações levam tempo e são agravadas pela falta de uma comunidade maior para auxílio, forçando o desenvolvedor a adotar alternativas que não são totalmente adequadas a solução do problema.

O estudo de caso apresenta exemplos de problemas, como a falta de conexão com bancos de dados de terceiros, a edição manual de arquivos para Android e também a falta de componentes gráficos adequados para o uso móvel.

A falta de uma comunidade expressiva em torno da ferramenta Gluon gera diversos problemas para o desenvolvedor final, principalmente pela falta de estabilidade com as diferentes versões das dependências e um desenvolvimento de funcionalidades mais lento.

5.7 Comportamento de recarga em tempo real

O recurso de recarga em tempo real é ausente na ferramenta, o que torna os processos de teste de comportamento e visualização da interface demorado, obrigando o desenvolvedor gerar o pacote executável da aplicação toda vez que necessitar visualizar os efeitos das alterações.

Apesar desse problema é possível visualizar o resultado no *desktop* por meio de uma rápida compilação para a máquina virtual Java, no entanto, essa alternativa não apresenta o resultado final de um código nativo, devido aos diferentes comportamentos dos componentes entre as plataformas, como citado na subseção 4.4.2.

5.8 Acesso ao sistema de arquivos e *hardware* do dispositivo

Na questão do acesso ao *hardware* dos dispositivos, a ferramenta disponibiliza diversos serviços que permitem a utilização das diferentes funcionalidades nativas existentes.

No estudo de caso foi utilizado, como exemplo, o recurso da câmera para escanear códigos de barras, que apresentou dificuldade na etapa de adição das dependências, por não estar presente na documentação, mas após configurado funcionou como o esperado.

Além deste, existem outros recursos como acelerômetro, *bluetooth*, bússola, notificações e também alguns que são adicionados por padrão ao projeto como o caso do *display* e *lifecycle*.

Seguindo para o acesso ao sistema de arquivos, a ferramenta traz consigo duas alternativas, a primeira é realizar o acesso através de um serviço do *attach* podendo fazer uso da memória interna ou externa do dispositivo, a segunda maneira é utilizando o Gluon CloudLink de forma *offline* salvando os dados localmente sem custos.

Mesmo com diversas tecnologias de banco de dados presentes no mercado, não se encontrou documentação para utilização, tornando a liberdade de escolha engessada em apenas uma ferramenta oficial e não gratuita.

5.9 Portabilidade

O Gluon Mobile surge justamente com a proposta de tornar o Java multiplataforma, mas ainda apresenta dificuldades nessa proposta. O produto atual é capaz de gerar pacotes para as principais plataformas, inclusive dispositivos embarcados, mas apresenta requisitos e limitações principalmente quando se trata da utilização de dependências intrínsecas a plataforma.

5.10 Compilação e desenvolvimento da aplicação

O plugin Gluon oferece quatro modelos base de projeto que estão pré-configurados e com algumas dependências já adicionadas, isso facilita para iniciar o desenvolvimento, além de fornecer um site (start.gluon.io) que permite diversas configurações para base de um projeto.

No que se refere ao processo de construção da aplicação final, as operações para realizar o build, package e install são agregadas pela ferramenta Maven e otimizam o processo.

6 CONCLUSÃO

O estudo de caso junto às pesquisas realizadas, fornecem uma visão nítida de como encontra-se a experiência de desenvolver uma aplicação multiplataforma, utilizando a linguagem Java em conjunto à plataforma gráfica JavaFX, e como única alternativa de integração, tem-se a organização Gluon.

Apesar de ser possível atingir o mercado móvel, fica evidente que ainda há um caminho a ser percorrido, sendo os principais problemas encontrados:

- Ausência da recarga em tempo real, que dificulta a visualização da aplicação final;
- Conversão dos componentes gráficos incompleta aumentando o tempo e o trabalho de desenvolvimento;
- Inconsistência no comportamento da aplicação em dispositivos diferentes;
- Problemas com integração com banco de dados, induzindo ao uso de *software* proprietário;
- Falta de uma documentação completa e organizada.

A falta de uma equipe de desenvolvedores maior junto à uma comunidade mais ativa, torna o desenvolvimento da ferramenta mais lenta e as soluções de problemas escassas. Essa falta é evidenciada principalmente no que se refere ao uso de recursos de terceiros, comumente utilizados por desenvolvedores em outras ferramentas, mas que ainda não fornecem compatibilidade com as aplicações Gluon, no estudo atual isso se refletiu na falta da integração com banco de dados.

Além disso, esse fator da equipe pequena também impacta na qualidade da documentação, sendo ela desorganizada e, muitas vezes, não anexada em apenas um local, isso acrescenta trabalho para o desenvolvedor que deveria encontrar informações na própria fonte, e não por meio de terceiros.

Outra implicação, são as soluções apresentadas pela comunidade que normalmente não se tratam de correções para problemas encontrados, mas sim de alternativas possíveis que podem ser adequadas somente para parte do problema.

As diversas dificuldades apresentadas no decorrer deste trabalho podem ocultar suas capacidades e afastar novos desenvolvedores, dado que não fica evidente oficialmente, que se trata de um projeto em desenvolvimento na questão de multiplataforma, atualmente em sua versão 0.0.53.

Mesmo sendo possível desenvolver uma aplicação com JavaFX que se aplique tanto a um dispositivo Android quanto ao *desktop*, pode-se concluir, que a ferramenta possui uma baixa confiabilidade devido aos diversos problemas encontrados, que atualmente a tornam de difícil utilização em aplicações reais.

Por se tratar de algo criado para adequar o Java já existente para executar em dispositivos móveis, gera o questionamento se a continuação do projeto desenvolvido neste trabalho é viável. Cabe uma análise entre a quantidade de trabalho necessária para concluir as ferramentas e mantê-las atualizadas, comparada com a quantidade de trabalho para se desenvolver algo novo com uma nova filosofia, como pode ser visto em outras ferramentas presentes no mercado de desenvolvimento para dispositivos móveis.

Porém, considerando o estado atual do *framework* JavaFX para aplicações móveis pode-se vislumbrar um potencial para pesquisa e desenvolvimento de novos *plug-ins, drivers* para bancos de dados e outras ferramentas para agregar funcionalidades ao ambiente, desde que se tenha o objetivo de pesquisa.

REFERÊNCIAS

ANATEL, Agência nacional de Telecomunicações. **Relatório de acompanhamento do setor de telecomunicações**. [S.l.], 2020. Disponível em: https://www.gov.br/anatel/pt-br/dados/acompanhamento/relatorios-de-acompanhamento/2020#R2020_90. Acesso em: 20 out. 2021.

ANDRADE, A. P. **Principais IDEs para desenvolvimento Java**. [S.l.], 2021. Disponível em: www.treinaweb.com.br/blog/principais-ides-para-desenvolvimento-java. Acesso em: 20 fev. 2022.

BONNINGTON, C. **5 Years On, the App Store Has Forever Changed the Face of Software**. [S. l.], 1 ago. 2014. Disponível em: <https://www.wired.com/2013/07/five-years-of-the-app-store/>. Acesso em: 24 out. 2021.

CASTILLO, C. **Arquitetura JavaFX**. [S.l.], abril de 2013. Disponível em: https://docs.oracle.com/javafx/2/architecture/jfxpub-architecture.htm#. Acesso em: 15 Jun. de 2022.

CHHABRA, J. K.; GUPTA, V. **A survey of dynamic software metrics.** Journal of computer science and technology, v. 25, n. 5, 2010.

CHIN, Stephen et al. **The Definitive Guide to Modern Java Clients with JavaFX 17**: Cross-Platform Mobile and Cloud Development. 2. ed. Apress, 2022. 636 p. ISBN 978-1-4842- 7267-1.

DEA, Carl et al. **JavaFX 9 by Example**: Rich-client applications for any platform. 3. ed. Apress, 2017. 557 p. ISBN 978-1-4842-1960-7.

DUFOUR, B. et al. **Dynamic metrics for Java**. Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications. 2003.

GITHUB, **Where the world builds software.** [S.l], 2022. Disponível em: github.com. Acesso em: 17 jun. 2022.

GITHUT, **Githut 2.0**: A small place to discover languages in GitHub. [S.l.], 2022. Disponível em: https://madnight.github.io/githut/. Acesso em: 18 fev. 2022.

GLUON. **Gluon Mobile**: criar aplicativos iOS e Android em Java. [S.l.], 2019. Disponível em: <gluonhq.com/products/mobile/>. Acesso em: 8 dez. 2021.

_____, **Gluon Documentation**. [S.l.], 2022. Disponível em: <docs.gluonhq.com>. Acesso em: 3 mar. 2022.

GOMES FILHO, M. J. A. **Um Processo de Avaliação Um Processo de Avaliação da Portabilidade de da Portabilidade de Unidades de Software Unidades de Software.** Universidade Federal de Pernambuco, 2005.

GONSALVES, Michael. **Evaluating the mobile development frameworks Apache Cordova and Flutter and their impact on the development process and application characteristics**. California State University, Chico. 2019.

GRAALVM, **Get Started with GraalVM**. [S.l.], 2022. Disponível em: https://www.graalvm.org/22.0/docs/getting-started/. Acesso em: 5 mar. 2022.

INTELLIJ, **IntelliJ IDEA:** o Java IDE capaz e ergonômico da JetBrains. [S.l.], 2022. Disponível em: jetbrains.com. Acesso em: 17 jun. 2022.

MAVEN, **Apache Maven Project**. [S.l.], 2022. Disponível em: maven.apache.org. Acesso em: 5 mar. 2022.

ORACLE. **JavaFX**: Getting Started with JavaFX. [entre 2008 e 2014]. Disponível em: docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm. Acesso em: 18 fev. 2022.

_____. **JavaFX Scene Builder**: Getting Started with JavaFX Scene Builder. [entre 2012 e 2014]. Disponível em: docs.oracle.com/javase/8/scene-builder-2/get-started-tutorial/overview.htm. Acesso em: 15 fev. 2022.

PEREDA, J. **How to solve fxml loading exceptions in compiled JavaFX project using GluonHQ client, Native Image and GraalVM?** [S. l.], 22 ago. 2020. <https://stackoverflow.com/questions/63527596/how-to-solve-fxml-loading-exceptions-in-compiled-javafx-project-using-gluonhq-cl>. Acesso em: 11 abril. 2021.

PRADO, E. F. do; SILVA, F. A. de S. **Análise teórica sobre o desenvolvimento de aplicativos nativos, híbridos e webapps**. Revista EduFatec: educação, tecnologia e gestão, Franca, 2019. Disponível em: https://revistaedufatec.fatecfranca.edu.br/wp-content/uploads/2019/09/AN%C3%81LISE-TE%C3%93RICA-SOBRE-O-DESENVOLVIMENTO-DE-APLICATIVOS-NATIVOS-H%C3%8DBRIDOS-E-WEBAPPS.pdf. Acesso em: 28 nov. 2021.

SOMMERVILLE, I. **Engenharia de Software**. 9 ed. Rio de Janeiro: Pearson, 2011.

SRIVASTAVA, P. **JavaFX:** A Rich Internet Application (RIA) Development Platform. [S. l.], 16 ago. 2016. Disponível em: www.opensourceforu.com/2016/08/javafx-rich-internet-application-ria-development-platform. Acesso em: 7 dez. 2021.

STATISTA, **Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021**. 21 fev. 2022. Disponível em: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/. Acesso em: 16 mar. 2021.

VOLTOLINI, R. **Conheça o primeiro smartphone da História** [galerias]. [S. l.], 1 ago. 2014. Disponível em: https://www.tecmundo.com.br/celular/59888-conheca-primeiro-smartphone-historia-galerias.htm. Acesso em: 25 out. 2021.

WILSON, G. **Making Software:** What Really Works, and Why We Believe It. [S.l]. O'Reilly Media, 2010.

APÊNDICE A - GUIA PARA CRIAÇÃO DE APLICAÇÕES MULTIPLATAFORMA UTILIZANDO FERRAMENTAS GLUON

Neste guia será abordado de forma prática um passo a passo detalhado de como obter um aplicativo utilizando a ferramenta Gluon Mobile, para isso dividiremos o tutorial em partes, começando pela obtenção das ferramentas necessárias, configuração do ambiente de desenvolvimento, iniciar o primeiro programa e ao final serão tratadas algumas funcionalidades extras que o Gluon disponibiliza, para serem acrescentadas ao programa base obtido no passo anterior.

Apesar de existir uma documentação oficial disponibilizada pela Gluon, ela contém algumas dificuldades que justificam a necessidade deste guia. Primeiramente possui diversos caminhos, que apresentam diferenças principalmente relacionadas ao sistema operacional utilizado no projeto. Outra questão importante é a constante atualização do projeto que altera pontos críticos tanto da ferramenta quanto da documentação. Por último, por se tratar de uma ferramenta de código aberto ainda em crescimento, contém uma comunidade pequena em comparação com outras tecnologias, fazendo com que problemas que seriam simples, se tornem um grande atraso no desenvolvimento.

O caminho selecionado envolve a criação multiplataforma de aplicações, porém o guia trará informações apenas da execução em sistemas Linux e Android, visto que plataformas como IOS e Mac Os necessitam de um dispositivo próprio.

1 OBTENÇÃO DAS FERRAMENTAS

Nesta etapa serão apresentadas as ferramentas necessárias para seguir o guia, abordando o sistema operacional, o ambiente de desenvolvimento e ferramentas adicionais.

1.1 Sistema Operacional

Adquira uma distribuição Linux do site oficial (Foi utilizado o *Linux Mint 20.3 Cinnamon*), recomenda-se uma instalação limpa que possibilite ao menos 8Gb de memória RAM disponíveis no sistema.

1.2 Ambiente de Desenvolvimento Integrado (IDE - *Integrated Development Environment*)

Baixe e instale o ambiente de desenvolvimento IntelliJ (Foi utilizado a versão *2021.3.2 Community Edition*) em um diretório conhecido que será necessário em passos seguintes. *Link* para *download*: jetbrains.com/pt-br/idea/download/

1.3 Scene Builder

Pode ser adquirido no *site* gluonhq.com/products/scene-builder/, sendo utilizado neste guia a versão 18.0.0, o local de instalação é importante ser anotado para ser conectado ao IntelliJ em passos futuros.

1.4 GraalVM - *Kit* de Ferramentas de Desenvolvimento (JDK)

Apesar de possuir o mesmo nome não queremos a versão oficial do GraalVM, mas sim uma compilação feita pelo Gluon que pode ser encontrada no *link* https://github.com/gluonhq/graal/releases/latest. No momento da elaboração do guia a versão que utiliza o Java 17 apresenta problemas para realizar o *link* de algumas dependências, devido a isso recomenda-se utilizar a versão do Java 11. Sua instalação se resume a extrair o pacote em um diretório conhecido.

2 CONFIGURAÇÃO DO AMBIENTE

Este tópico apresenta como deve ser configurado o ambiente, considerando que as ferramentas acima foram obtidas e instaladas corretamente.

2.1 Pacote *Build-Essential*

Sistemas baseados em Debian como é o caso do Linux Mint possuem um pacote de ferramentas que serão utilizadas na compilação do projeto. O pacote pode ser obtido utilizando um terminal e inserindo o comando "*sudo apt update*" e após "*sudo apt install build-essential*".

2.2 Pacote do *substrate*

Para obter uma imagem nativa para Linux são necessários pacotes adicionais, que podem ser instalados com os seguintes comandos em um terminal Linux:

"*apt-get install libasound2-dev libavcodec-dev libavformat-dev libavutil-dev libfreetype6-dev"*

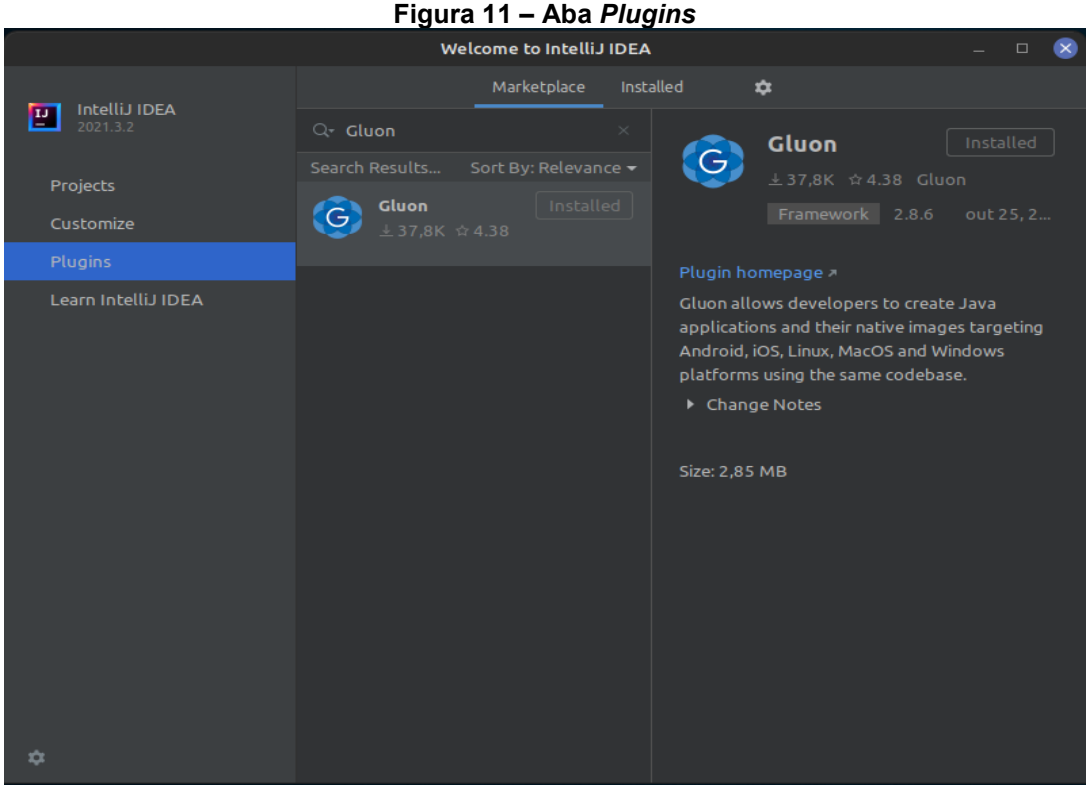
"apt-get install libgl-dev libglib2.0-dev libgtk-3-dev libpango1.0-dev libx11-dev libxtst-dev zlib1g-dev"

2.3 Plugin Gluon

O *plugin* Gluon contém ferramentas que auxiliam na construção de um aplicativo da Gluon, disponibilizando projetos base para começar uma aplicação e possibilitando a seleção de um gerenciador de projetos, sendo eles, Maven e Gradle.

Para instalá-lo, acesse a aba de *plugins* do IntelliJ e na barra de pesquisa digite "Gluon", como mostrado na Figura 11.

Figura 11 – Aba *Plugins*



Fonte: Autoria própria.

2.4 *Android Debug Bridge* (adb)

O *adb* é uma ferramenta que facilita a comunicação entre o computador e o dispositivo Android. A documentação indica que a ferramenta é instalada junto com com o *plugin*, porém a mesma apresentou problemas que foram resolvidos com a instalação manual a partir do comando "*sudo apt install adb*" em um terminal Linux.

2.5 Permissão para o MVN (Maven)

Como o ambiente utilizado é o Linux, é necessário conceder permissões de execução ao arquivo "*mvn*" dentro da instalação da IDE, em "*plugins/maven/lib/maven3/bin*".

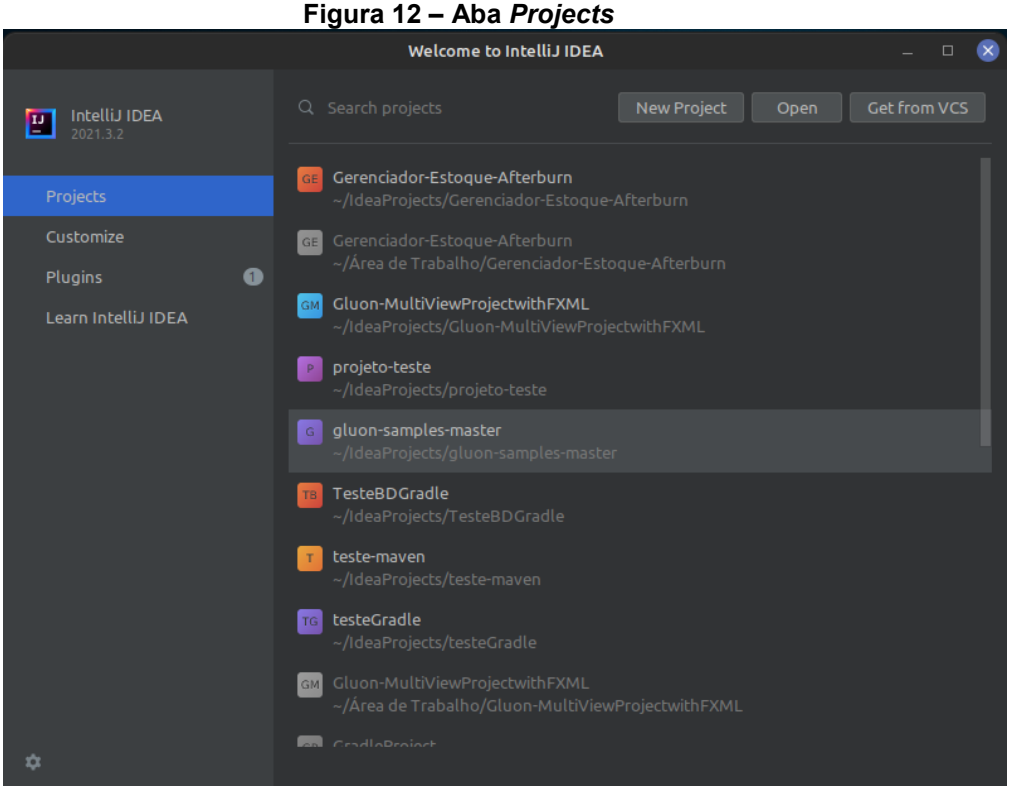
3 CRIANDO O PRIMEIRO PROJETO

Nesta etapa serão apresentados os passos para a criação de um projeto base e criação de um pacote instalável.

3.1 Iniciando o projeto

Com o *plugin* do Gluon já instalado, podemos encontrar na página inicial do IntelliJ selecionando a aba "*Projects*" selecione a opção "*New Project*" (Figura 12).

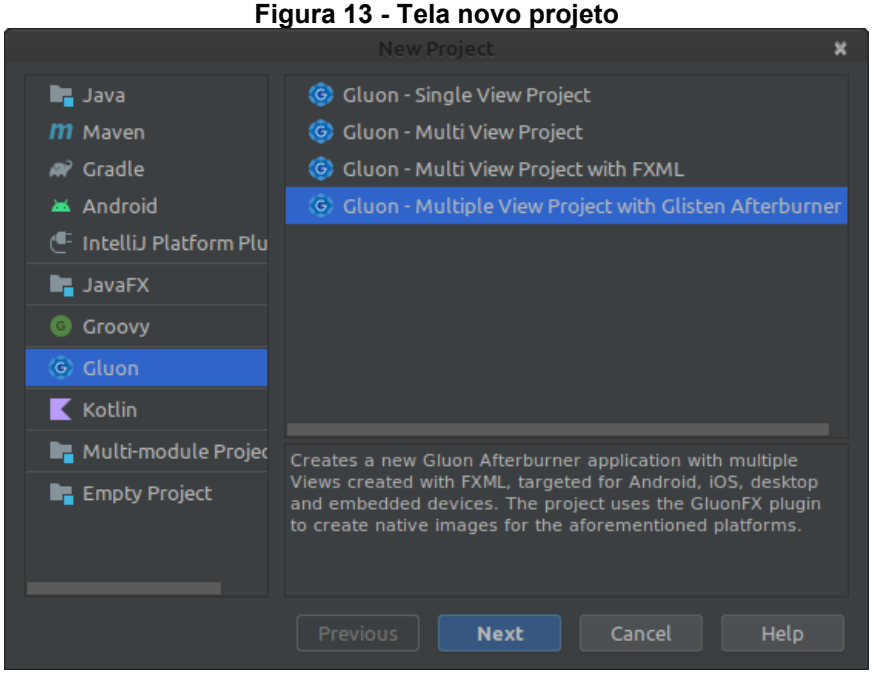
Figura 12 – Aba *Projects*



Fonte: Autoria própria.

Será aberta uma nova tela com uma nova categoria denominada "*Gluon*", nela serão fornecidas quatro opções de projetos iniciais que devem ser selecionados de acordo com a necessidade, para esse guia usaremos a opção "*Gluon - Multiple View Project with Glisten Afterburner*" mostrado na Figura 13, pois contém alguns recursos que serão utilizados como facilitador para a persistência de dados.

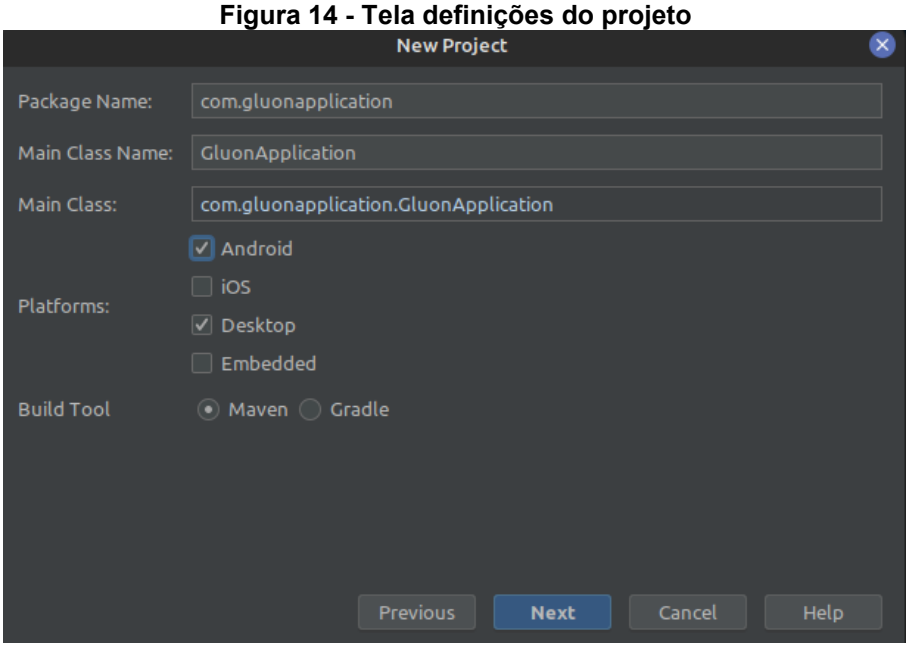
Figura 13 - Tela novo projeto



Fonte: Autoria própria.

Como mostrado na Figura 14, definiremos a opção "*Platforms*" para "*Android*" e "*Desktop*" e o "*Maven*" será selecionado como gerenciador de projeto, visto que a documentação atual do Gluon utiliza ele como principal. As demais opções são padrões de projetos Java e devem ser utilizadas de acordo com a necessidade do projeto.

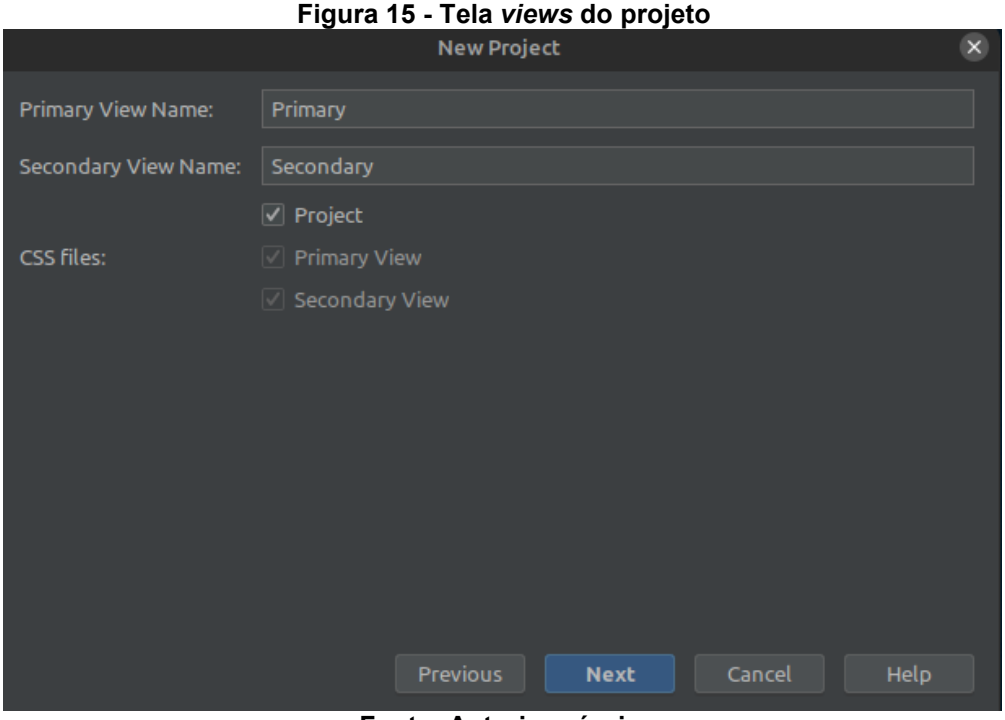
Figura 14 - Tela definições do projeto



Fonte: Autoria própria.

O projeto base traz duas telas iniciais como padrão, como mostrado na Figura 15. Os seus nomes podem ser definidos de acordo com a preferência do desenvolvedor e necessidade do projeto.

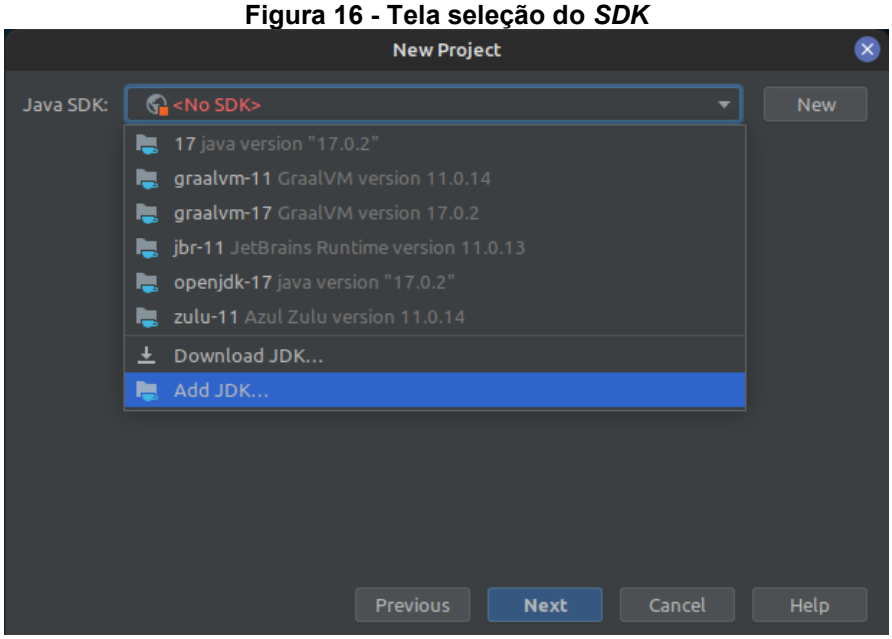
Figura 15 - Tela *views* do projeto



Fonte: Autoria própria.

Na criação do primeiro projeto será necessário adicionar o GraalVM instalado no tópico 1.4 deste guia como SDK, para isso na tela seguinte abra a lista e selecione "*Add JDK*", como mostra a Figura 16.

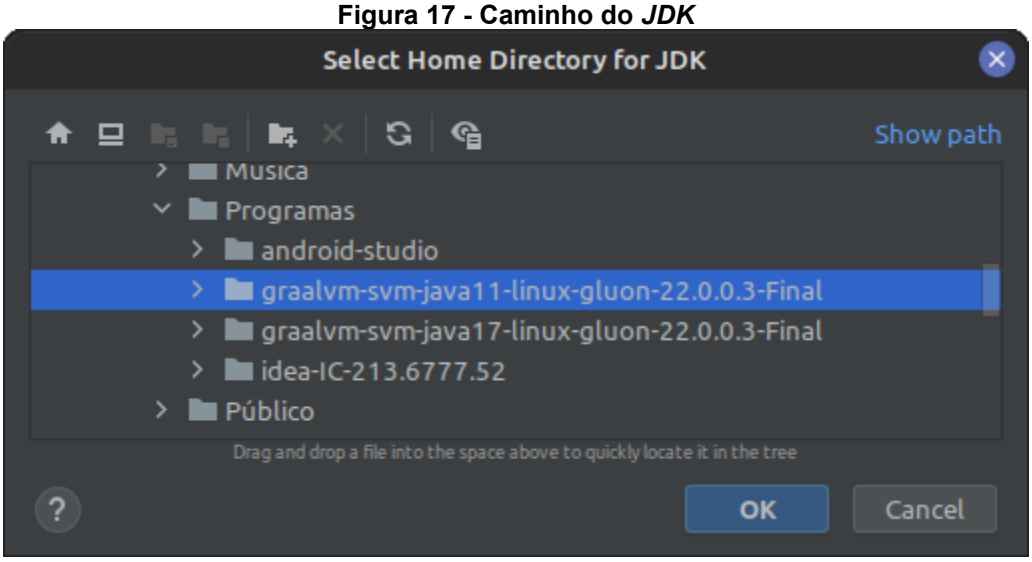
Figura 16 - Tela seleção do *SDK*



Fonte: Autoria própria.

Na tela que se abre mostrada na Figura 17 devemos selecionar o local de instalação do GraalVM e em seguida clicar em *OK*, você será enviado para tela anterior e a partir desse projeto, a versão já estará disponível na lista para ser selecionada diretamente, não necessitando mais selecionar seu caminho.

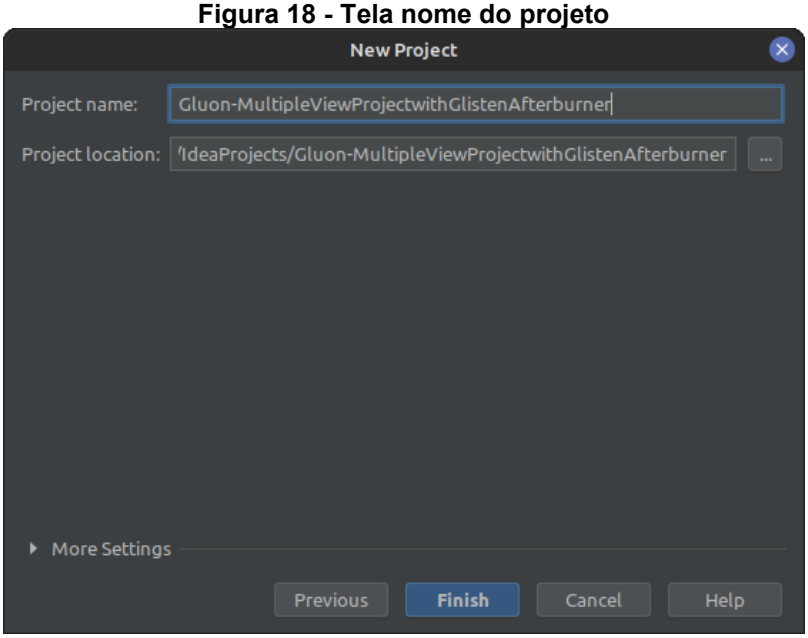
Figura 17 - Caminho do *JDK*



Fonte: Autoria própria.

Tendo selecionado o *SDK* podemos seguir para a próxima tela, nela podem ser definidos o nome do projeto e o local que ele será salvo, como mostrado na Figura 18. Feito isso, basta finalizar a criação do projeto.

Figura 18 - Tela nome do projeto

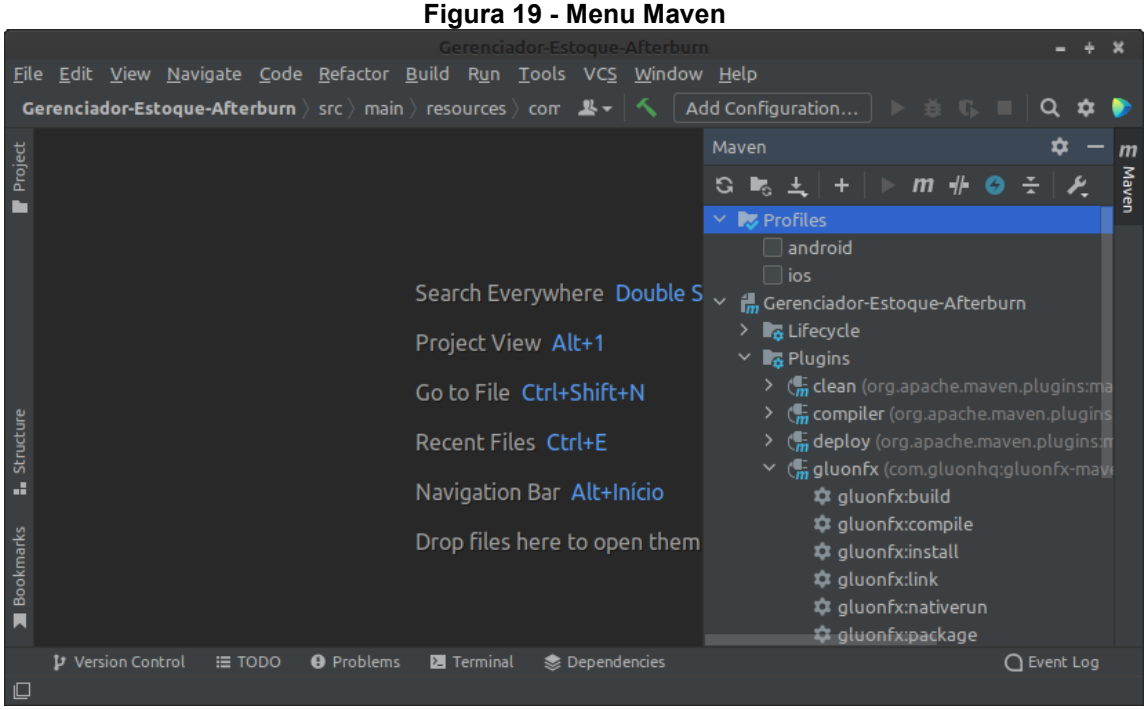


Fonte: Autoria própria.

3.2 Menu Maven

O menu Maven está disposto no lado direito da tela principal como mostrado na Figura 19, nele encontraremos opções importantes relacionadas à construção e execução do projeto, assim como algumas configurações do próprio Maven.

Figura 19 - Menu Maven



Fonte: Autoria própria.

3.3 Configurando o GraalVM

Para a correta configuração do Maven é necessário configurar uma variável de ambiente que aponte para o diretório de instalação do GraalVM, a maneira mais simplificada de configurá-lo é acessando o menu "*Maven Settings*" mostrado na Figura 20.

Figura 20 - *Maven Settings*

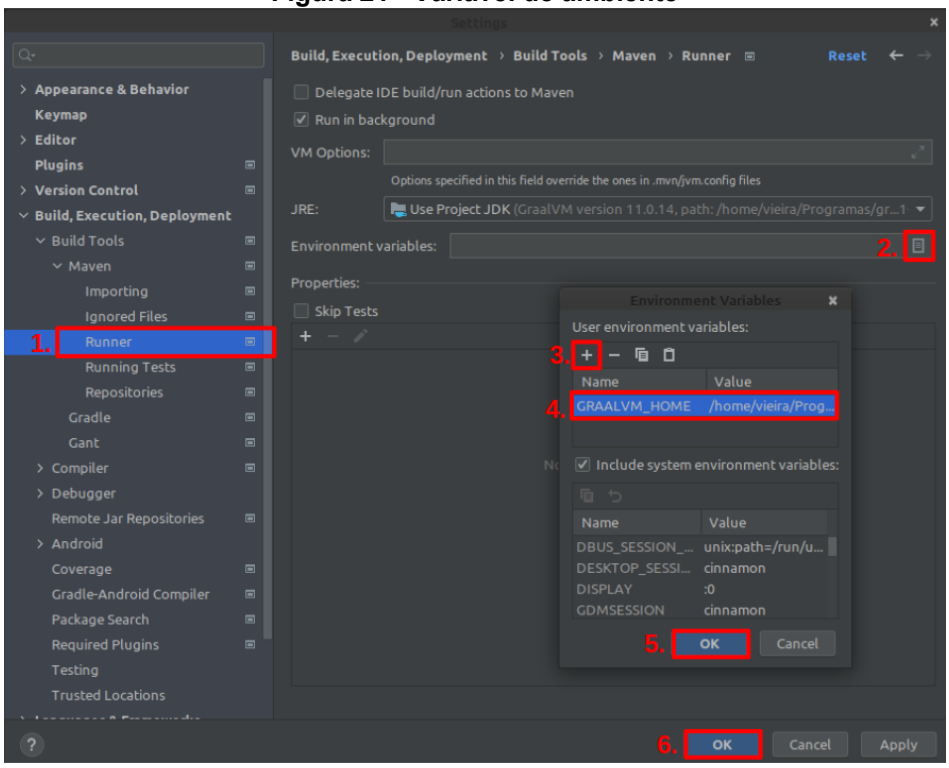
Fonte: Autoria própria.

A Figura 21 apresenta uma sequência de passos para adicionar a variável de ambiente ao Maven.

Seguindo a numeração apresentada acesse o menu "*Runner*" (1), em seguida localize e clique no ícone (2) que abrirá uma nova janela, selecionando o ícone (3)

habilitará uma nova linha (4) em branco, nela a coluna “*Name*” deve ser preenchida com “*GRAALVM_HOME*” e no campo “*Value*” deve ser inserido o caminho de instalação do GraalVM. Em seguida, selecione “*OK*” (5 e 6) para aplicar as alterações.

Figura 21 - Variável de ambiente

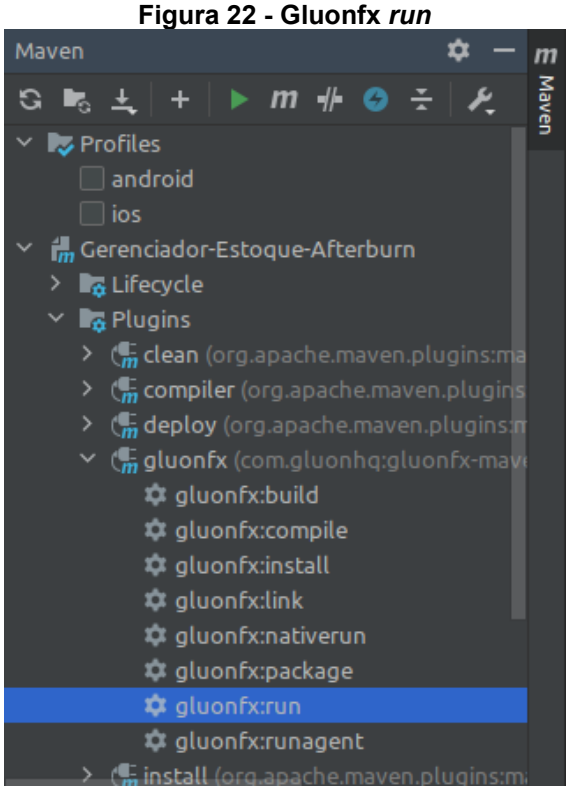


Fonte: Autoria própria.

3.4 Executando o projeto na Máquina Virtual Java (JVM)

A Figura 22 mostra o menu maven, nele existe a operação “*gluonfx:run*” que permite realizar rapidamente testes na etapa de desenvolvimento utilizando a máquina virtual Java, essa operação não converte códigos binários, agilizando o processo.

Figura 22 - Gluonfx *run*



Fonte: Autoria própria.

A execução realizada por meio deste método não representa realmente como será o resultado final, principalmente em dispositivos Android, onde eventualmente apresentou problemas na junção de dependências e proporção de tela.

3.5 Construindo para código nativo

Para obter uma imagem nativa do aplicativo é preciso selecionar a opção "*android*" (1) em "Profiles" no menu Maven, como mostrado na Figura 23. Note que "Gerenciador-Estoque-*Afterburn*" é o nome do projeto utilizado no exemplo.

Primeiro execute a operação "*gluonfx:build*" (2), essa etapa contempla a compilação e o link das dependências, e trata-se do processo mais demorado.

Após aguardar a conclusão da etapa anterior, execute a operação "gluonfx:*package*" (3), esse processo levará mais alguns instantes.

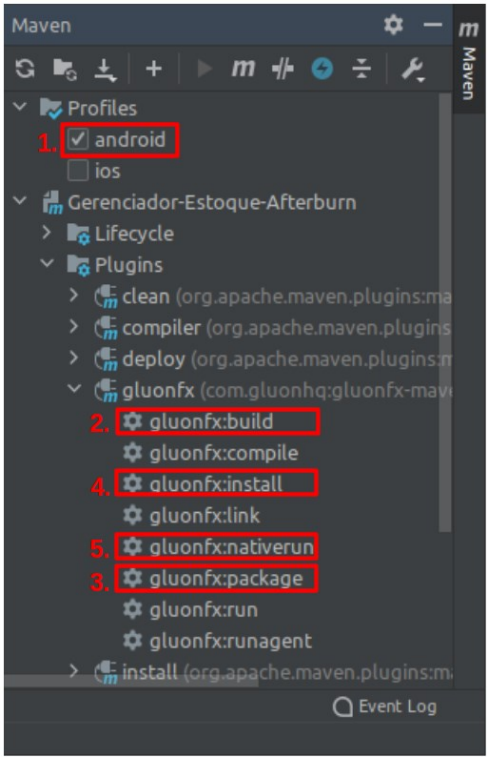
A partir desse ponto já temos um arquivo "*.apk*" gerado na pasta do projeto em *"/target/gluonfx/aarch64-android*" pronto para ser enviado e instalado em um dispositivo.

Para realizar os testes diretamente em um dispositivo móvel deve-se habilitar o modo de depuração do mesmo, esse passo não será contemplado neste guia, pois

varia para cada dispositivo. Em seguida, deve-se conectá-lo ao computador via cabo USB (*Universal serial bus*).

O comando “*gluonfx:install*” (4) instalará o aplicativo no dispositivo, note que no decorrer do processo, seu dispositivo Android irá pedir algumas permissões, lembre-se de aceitá-las, após a instalação execute o comando “*gluonfx:nativerun*” (5) executará a aplicação utilizando o console do IntelliJ como depurador.

Figura 23 - Guia criar pacote

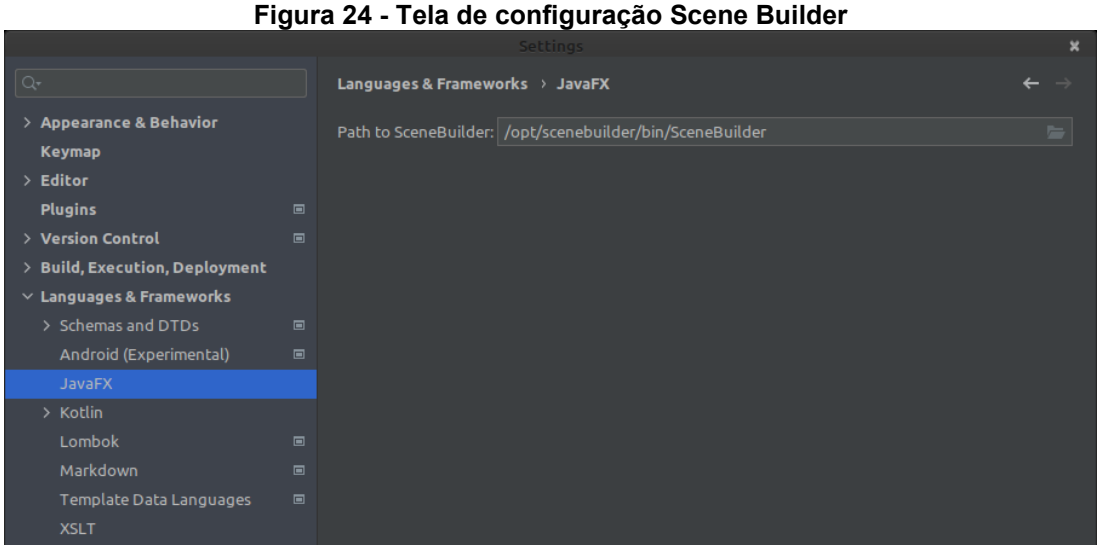


Fonte: Autoria própria.

4 CONFIGURANDO O SCENE BUILDER

Como recurso facilitador podemos acessar o Scene Builder a partir da IDE, isso pode ser feito no IntelliJ acessando “*File*” -> “*Settings*” na tela aberta acesse “*Languages & Frameworks*” -> “*JavaFX*” e adicione o caminho de instalação do Scene Builder no campo disponível mostrado na Figura 24.

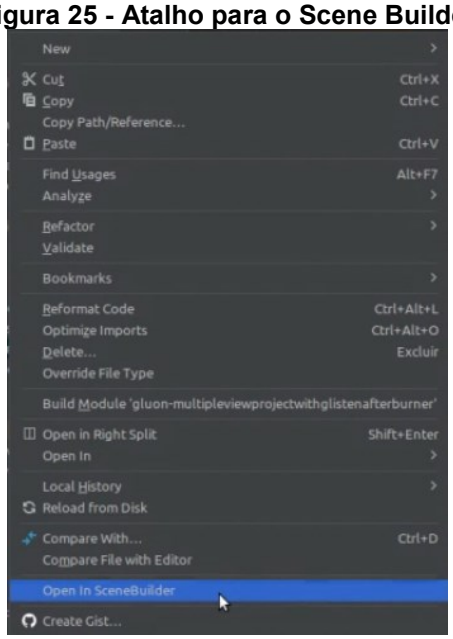
Figura 24 - Tela de configuração Scene Builder



Fonte: Autoria própria.

Isso permitirá que arquivos “*.fxml*” sejam abertos no Scene Builder, a partir da árvore de arquivos do IntelliJ, como apresentado na Figura 25, acessando o menu contexto do arquivo com o botão direito e clicando em “*Open In Scene Builder*”.

Figura 25 - Atalho para o Scene Builder



Fonte: Autoria própria.

5 CONFIGURAÇÕES DO PROJETO

O projeto foi criado utilizando o Maven como gerenciador e uma característica de seu funcionamento é utilizar um arquivo Projeto Modelo de Objeto (*POM - Project Object Model*), nele são adicionadas diversas informações sobre o projeto, como dependências, classes, recursos, versões entre outras.

Dentro do arquivo "*pom.xml*" encontraremos diversas *tags*, entre elas atente-se para:

- "*dependencies*": Deve conter informações de dependências que o Maven gerência, atente-se que nem todas as dependências estarão disponíveis para dispositivos Android, podendo causar erros.
- "*attachList*": Contém os recursos nativos a serem utilizados no projeto, serão abordados posteriormente neste guia.
- "*bundlesList*": Deve conter os caminhos dos arquivos "*.fxml*", por exemplo:

  <list>com.gluonapplication.nome_projeto.views.primary</list>
- "*reflectionList"*: Deve conter as classes do projeto seguindo a seguinte estrutura:

  <list>com.gluonapplication.nome_projeto.views.PrimaryPresenter</list>

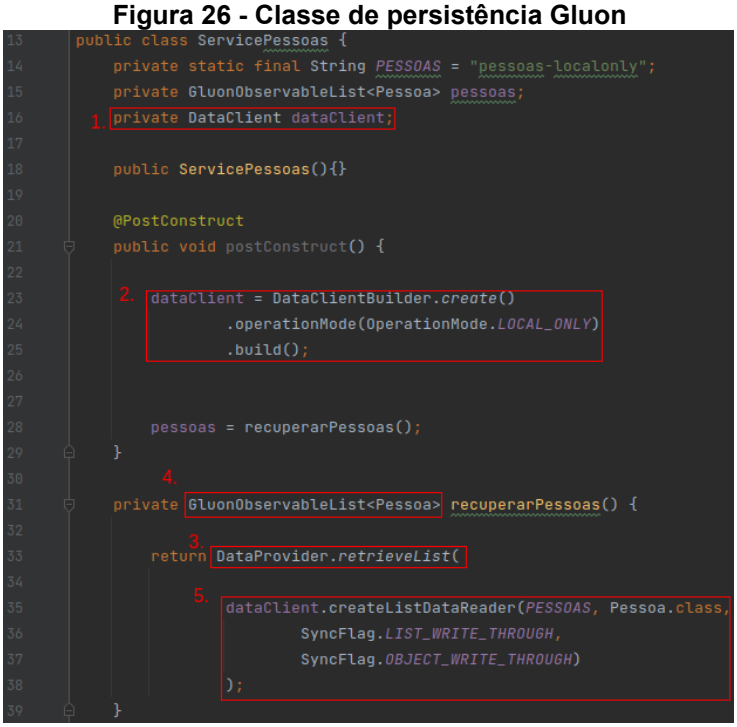
6 PERSISTÊNCIA COM GLUON LINK GRATUITO

A persistência dos dados foi realizada por meio da ferramenta CloudLink disponibilizada pela Gluon, consiste em um conector entre a aplicação e uma base de dados. Apesar de ser um recurso pago em suas operações *online*, ele possibilita o armazenamento local de forma gratuita gerenciando diretório e arquivos de forma simplificada e multiplataforma.

Para utilizá-la, existem classes específicas criadas pelo Gluon que permitem a criação e o armazenamento em arquivos, as quais definem a forma como os dados serão armazenados, além da classe do objeto que será armazenado que varia de acordo com o projeto.

Para exemplificar mostraremos uma codificação na Figura 26, o exemplo a seguir foi baseado no código disponibilizado em https://github.com/gluonhq/gluon-samples/tree/master/notes, que é um repositório oficial do Gluon.

Figura 26 - Classe de persistência Gluon



Fonte: Autoria própria.

A Figura 26 representa uma classe de serviço para controlar o armazenamento e a recuperação de dados, que no caso está armazenando objetos de uma classe chamada Pessoa. A classe responsável pela criação dos arquivos na memória é a denominada "*DataClient*" como mostrado em (1.), que é inicializada através das funções "*create()*" e "*build()*" da classe "*DataClientBuilder*", ao mesmo

tempo é realizada a configuração da forma de armazenamento, que no exemplo é realizado localmente, como mostrado em (2.).

Para recuperar os arquivos salvos é utilizada uma classe chamada "*DataProvider*", também criada pelo Gluon, que possui a função "*retrieveList()*" (3.), que recupera os arquivos em uma lista observável própria do Gluon, definida com o tipo do objeto utilizado (4.). Como parâmetro dessa função, é atribuído um objeto "*ListDataReader*" de um tipo especificado, o mesmo é obtido por meio da função "*createListDataReader()*" que recebe como parâmetro uma "*String*" que identifica o nome do arquivo salvo e a classe do objeto que está armazenada nele, além de duas "*flags*" que dão permissão para escrita como mostrado em (5.).

A partir da lista obtida como retorno, os objetos armazenados podem ser manipulados da forma que for necessária, podendo excluir, atualizar ou adicionar dados, essas operações são refletidas diretamente nos arquivos salvos. Exemplos de operações com a lista são mostrados na Figura 27.

Figura 27 - Exemplo de uso da persistência

```
42      public void addPessoa(Pessoa pessoa) {
43          pessoas.add(pessoa);
44      }
45      public void removePessoa(Pessoa pessoa) { pessoas.remove(pessoa); }
48
49      public GluonObservableList<Pessoa> getPessoas() { return pessoas; }
52
53  }
```

Fonte: Autoria própria.

É importante ressaltar que para cada tipo de objeto que for armazenar será necessário a criação de uma classe de serviço próprio para ele, isso porque cada uma terá uma lista com um tipo de objeto diferente.

7 GLUON ATTACH

Trata-se do uso de recursos nativos do dispositivo, sendo abstraídos pelo Gluon como *attach*, sua implementação depende do recurso que deseja utilizar, como exemplo foi utilizado a câmera do dispositivo móvel como um leitor de código de barras.

Primeiramente deve-se adicionar a dependência ao arquivo "*pom.xml*" dentro da *tag* "<*dependencies*>" da seguinte forma:

```
<dependency>
    <groupId>com.gluonhq.attach</groupId>
    <artifactId>barcode-scan</artifactId>
    <version>${attach.version}</version>
</dependency>
```

Ainda no arquivo "*pom.xml*" existe a *tag* "<*attachList*>", onde deve ser adicionado o *attach*, no caso:

```
<list>barcode-scan</list>
```

A documentação não informa as dependências, mas elas podem ser encontradas no *site start.gluon.io*, onde é gerado um arquivo "*pom.xml*" de acordo com as opções selecionadas.

O próximo passo é adicionar dados ao arquivo "*AndroidManifest.xml*", porém esse arquivo só é criado quando realizada a operação "*gluonfx:package*", é sugerido também que esse arquivo seja movido da pasta "*<pasta do projeto>/target/gluonfx/aarch64-android/gvm/android_project/app/src/main*" para a pasta caminho, para que as alterações não se percam nas próximas construções do projeto. Em seguida adicione as configurações fornecidas pela documentação do *attach*:

```
<manifest ...>
  <uses-permission android:name="android.permission.CAMERA"/>
  ...
  <application ...>
    ...
```

```
    <activity android:name="com.gluonhq.helloandroid.zxing.CaptureActivity"
            android:screenOrientation="sensorLandscape"
            android:clearTaskOnLaunch="true"
            android:stateNotNeeded="true"
            android:windowSoftInputMode="stateAlwaysHidden">
      <intent-filter>
        <action android:name = "com.gluonhq.attach.barcodescan.android.SCAN" />
        <category android:name="android.intent.category.DEFAULT"/>
      </intent-filter>
    </activity>
<activity
android:name="com.gluonhq.impl.attach.plugins.android.PermissionRequestActivity"
/>
   </application>
  </manifest>
```

A documentação referente a cada *attach* pode ser encontrada em *gluonhq.com/products/mobile/attach*.

Nesta etapa o *barcode* está devidamente configurado, restando apenas utilizá-lo chamando o serviço como no exemplo seguinte:

```
BarcodeScanService.create().ifPresent(service -> {
    Optional<String> barcode = barcodeScanService.scan();
    barcode.ifPresent(barcodeValue -> System.out.println("Scanned Bar Code: " +
barcodeValue));
  });
```

O exemplo de cada *attach* pode ser encontrado junto com sua documentação em *gluonhq.com/products/mobile/attach*.

O resultado final será a exibição de uma tela com a visualização da câmera configurada para leitura de códigos de barras ou códigos QR.