UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

Documento de Arquitetura de Software

Versão 1.0

Alexson Almeida - 508294

Daniel Almeida de Freitas - 508077

José Vitor - 509295

Histórico de Revisões

Data	Versão	Descrição	Autor
03/09/2023	1.0	Início do documento com sessões importantes.	Daniel Almeida

Indice Analitico

Documento de Arquitetura de Software

1. Introdução

Para se obter uma visão arquitetural do aplicativo de troca de livros <Nome do Aplicativo>, este documento vem para elucidar os diversos aspectos do sistema. Suas colocações são para transmitir as escolhas dadas em termos arquitetônicos, o qual a equipe adotou para o desenvolvimento do mesmo.

O objetivo desse documento é classificar de forma clara a arquitetura, o qual a equipe estabeleceu para o desenvolvimento do <NOME DO APLICATIVO> APLICATIVO DE TROCA DE LIVROS, o foco é seguir o desenvolvimento no padrão de arquitetura cliente e servidor uma vez ela se adequa melhor para os requisitos elicitados para a aplicação. Ademais, o lado do servidor - responsável por responder solicitações dos clientes - será construído utilizando a arquitetura clean architecture, baseada em camadas. Por último, o lado do cliente será construído utilizando uma arquitetura Model-View-ViewModel (MVVM), devido ela conseguir separar bem a camada de UI das camadas de dados subjacentes.

No documento está definido o que o <NOME DO APLICATIVO> estará contemplando em termos arquitetônicos, especificando seus casos de usos, visões lógicas, camadas assim como sua implementação e implantação. A equipe se atentou em todas as decisões arquitetônicas presentes neste documento, o que é único para o desenvolvimento do Aplicativo de troca de livros. A aplicação vem se tratando de um ambiente MOBILE, que tende a utilização de um banco de dados o qual estará dentro do ambiente de implementação que é um servidor web, seguindo princípios de boas práticas.

1.1 Finalidade

A finalidade deste documento, é oferecer de modo claro as diversas visões o qual os modelos arquitetônicos que vincula o aplicativo vem a possuir, trazendo consigo as características necessárias para os controles de suas atividades arquitetônicas, assim moldando todo procedimento para o desenvolvimento do sistema.

1.2 Escopo

A arquitetura do aplicativo <NOME DO APLICATIVO> é totalmente fundamentada e detalhada, para moldar uma base para a equipe de desenvolvimento, apresentando como será o comportamento do sistema. Esse documento refere-se às características identificadas no “App_Doc_Riq*” o qual tratam das necessidades e as regras do negócios o qual o <NOME DO APLICATIVO> deve atender, nesse documento contém os registros e tópicos importantes relacionado a arquitetura do sistema, dando diretrizes, e total apoio para o uso de suas tecnologias.

Definições, Acrônimos e Abreviações

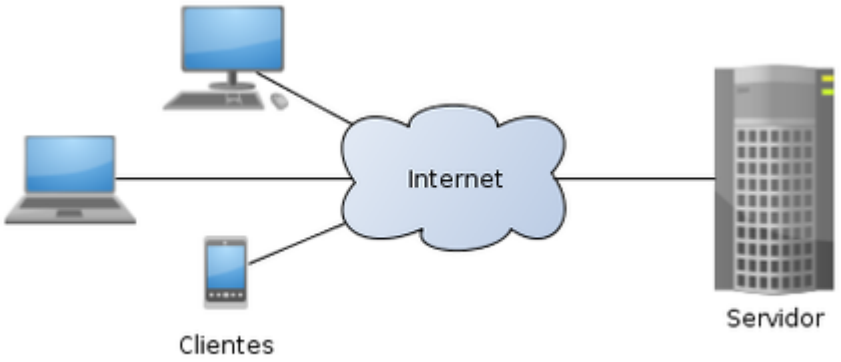
Abreviação	Acrônimos	Descrição
MVVM	Model-View-ViewModel	Padrão de arquitetura do Software

API	Application Programming Interface	Conjunto de funções e procedimentos que permitem a integração de um ou mais sistemas.
App_Doc_Req	Aplicativo Documento Requisitos	Identificador do documento de Requisitos do <NOME DO APLICATIVO>
RNF	Requisito Não Funcional	Identificador de um requisito não funcional.
RF	Requisito Funcional	Identificador de um requisito funcional.
SO	Sistema Operacional	Software responsável por alocar e gerenciar recursos de um sistema.

2. Representação Arquitetural

2.1. Arquitetura Cliente e Servidor

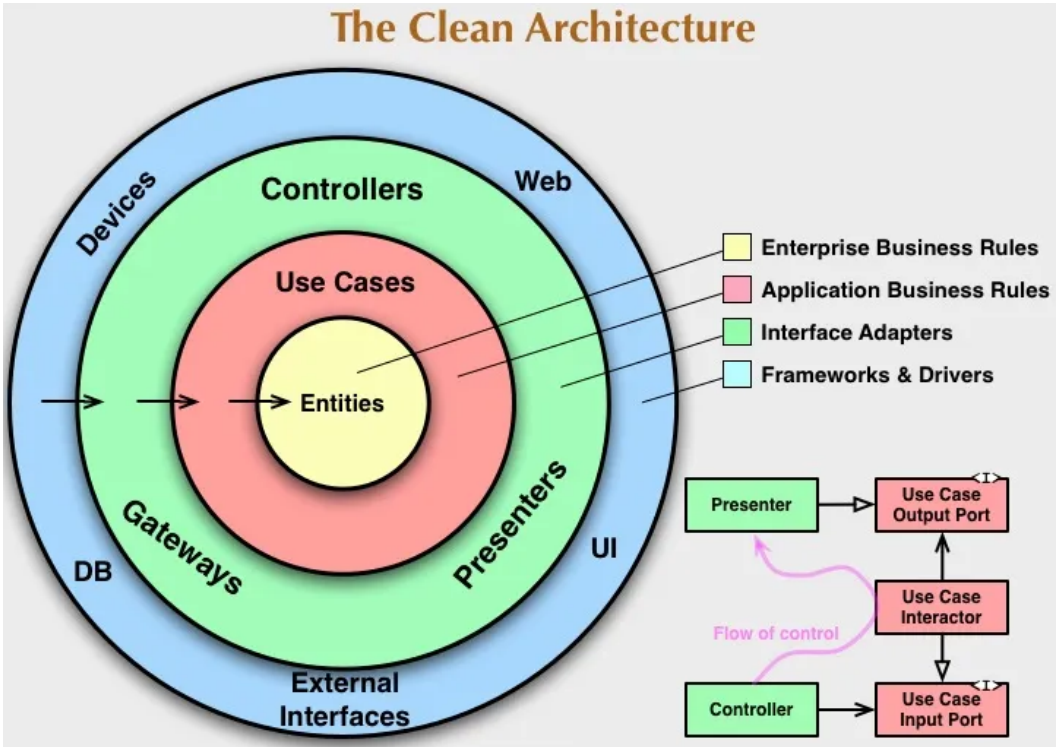
Como dito anteriormente o projeto será desenvolvido usando uma arquitetura cliente e servidor que é dividida em mais dois tipos de arquiteturas subsequentes, sendo conectadas por meio de uma API. A arquitetura cliente servidor é uma arquitetura de aplicação distribuída, ou seja, na rede existem os fornecedores de recursos ou serviços a rede, que são chamados de servidores, e existem os requerentes dos recursos ou serviços, denominados clientes. Essa escolha se dá principalmente por esse tipo de arquitetura se adequar bem aos requisitos do projeto e por ser esse modelo permite a separação das responsabilidades entre o cliente e o servidor, possibilitando um melhor controle sobre o fluxo de informações e expansão do sistema no futuro.



2.2. Clean Architecture (Lado do Servidor)

Uma vez que estamos seguindo uma arquitetura cliente-servidor, temos a oportunidade de separar nossa aplicação em dois fluxos diferentes, por consequência, também podemos escolher tecnologias e arquiteturas diferentes para cada fluxo. Sendo assim, o nosso lado do servidor será construído utilizando uma arquitetura baseada em camadas chamada de Clean Architecture. Falando sobre suas camadas, a camada mais interna é chamada de “Entities”, tem como objetivo organizar as classes responsáveis pelas regras de negócio comum a vários sistemas e implementar regras de negócios genéricas. Sua segunda camada é a “Uses Cases” e tem como objetivo implementar regras de negócio específicas do sistema. Na terceira camada de dentro para fora temos a camada de “Controllers”, camada responsável por mediar a interação entre a camada mais externa da arquitetura (sistemas externos) e as camadas centrais (Casos de Uso e Entidades) e implementação dos endpoints da REST da API que iremos criar. Já na camada mais externa temos a camada de “Frameworks Externos”, camada essa onde ficam classes de bibliotecas, frameworks e quaisquer sistemas externos.

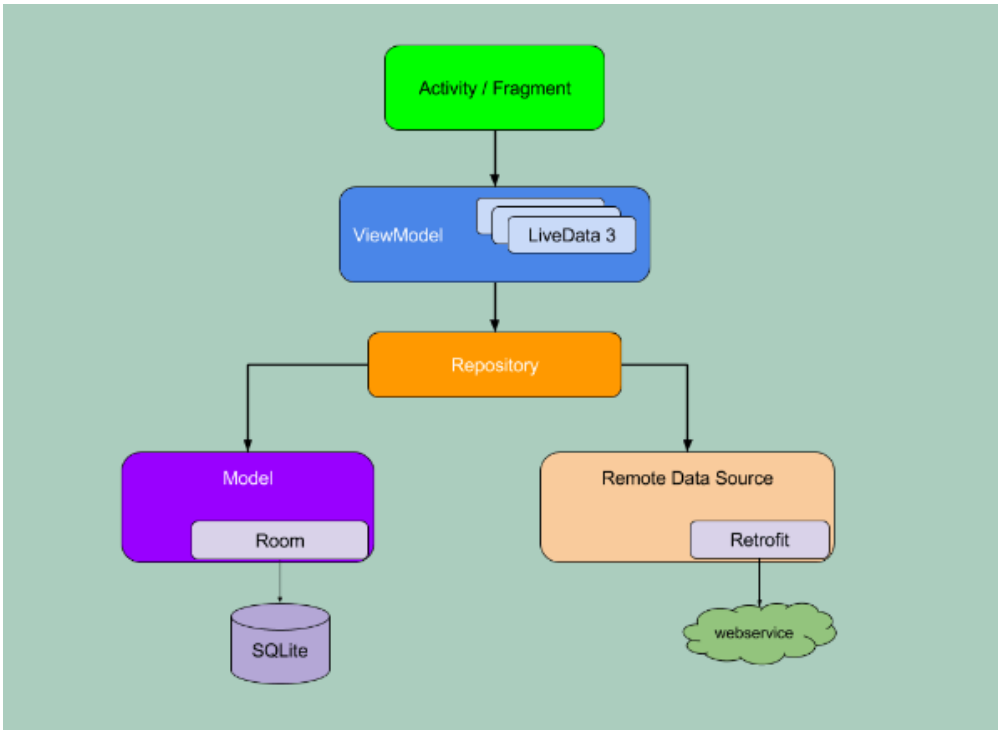
Sendo assim, escolhemos essa arquitetura por ela separar bem as responsabilidades de uma aplicação, além de facilitar o desenvolvimento de códigos, permitir uma melhor manutenção, uma boa atualização e menos dependências no projeto. Ademais, vale ressaltar que outros pontos importantes para a escolha foi o fato dessa arquitetura ser muito bem testável e independente de qualquer banco de dados ou agente externo.



2.3. Arquitetura Android MVVM (Lado do Cliente)

Seguindo a arquitetura cliente servidor, o lado do cliente será desenvolvido baseado na arquitetura MVVM - arquitetura Android baseada em responsabilidades - que visa separar a camada de UI das camadas de dados subjacentes. A primeira camada dessa arquitetura é a “Activity/ Fragment”, camada essa responsável pela visualização e interações com o usuário, a segunda é a camada ViewModel, responsável pelas regras de negócios da aplicação. Já a última camada é responsável pelos dados da aplicação, chamada por repository sua finalidade é fazer acessos a serviços externos por meio do retrofit ou acessar um banco de dados interno, exemplo o SQLite.

Sendo assim, escolhemos essa arquitetura por ela separar as responsabilidades da nossa camada de UI das camada de lógica e acesso a dados da aplicação. Além disso, esse tipo de arquitetura aumenta a testabilidade do aplicativo, uma vez que a separação dessas responsabilidades diminui o acoplamento entre o código escrito pela equipe e o código do SO, ou seja, diminui as chances seu de algum teste ficar impossibilitado de ser executado na JVM por existir alguma dependência que necessite de classes do SO para ser executada.



3. Tecnologias e Ferramentas

Levando em consideração os modelos de arquiteturas mostrados anteriormente, a equipe definiu as seguintes tecnologias e ferramentas para serem usadas durante o processo de desenvolvimento.

Nome	Descrição	Onde será usado(a)	Justificativa para o uso
Node.Js	Plataforma que interpreta o Javascript no lado do servidor.	Servidor	Familiaridade do time com a linguagem de programação JavaScript e experiência em construções de API REST utilizando Node.Js
NestJS	Framework para criar aplicativos baseado em Node.Js para o lado do servidor.	Servidor	Suporte para a criação de API REST, fácil integração com diferentes bancos de dados e ORMs, e possui uma arquitetura escalável baseada em conceitos modulares do Angular, que se encaixa perfeitamente com a arquitetura proposta para o projeto.
Typescript	Linguagem de programação fortemente tipada baseada em JavaScript.	Servidor	Aumenta a produtividade no desenvolvimento e previne erros de tipagens que possam vir a ter com o Javascript. Ademais, o typescript já vem por padrão no NestJs.
PostgreSQl	Banco de dados relacional.	Servidor	Se adequa melhor aos requisitos do projeto por ser relacional. Ademais, o time tem mais experiência em trabalhar com o Postgres.
Prisma	ORM para acesso ao banco de dados.	Servidor	Facilita o acesso ao banco de dados, tirando a necessidade de escrever queries em SQL puro.
Jest	Framework de testes baseado em JavaScript.	Servidor	Fornece uma estrutura de testes sem a necessidade de muitas configurações, com bom desempenho e boas opções de análise.
Kotlin	Linguagem de programação para o desenvolvimento de aplicativos Android.	Cliente	É mais performático em relação ao java e é uma das restrições da cadeira de mobile.

Aplicativo de Trocas de Livros	Versão: 1.0
Documento de Arquitetura de Software	Data: 03/09/2023
APP_Doc_Arq_V.1.0	