Documento de Projeto de Sistema

CityGuiaGo

Vitória, ES

2024

Registro de Alterações:

Versão	Responsável	Data	Alterações
0.1	Gabriel Ferrari Wagnitz	27/11/2024	Versão inicial.
0.2	Mateus Freitas Couto	28/11/2024	Alterações em (1) e (2)
0.3	Gabriel Ferrari Wagnitz	08/12/2024	Desenho da arquitetura de Software
0.4	Gabriel Ferrari Wagnitz	10/12/2024	Detalhamento da Arquitetura de Software
0.5	Leonardo Franco Emerick Albergaria	11/12/2024	Detalhamento dos Componentes e Tecnologias Mobile
0.6	Mateus Freitas Couto	10/12/2024	Revisão de RNFs
1.0	Gabriel Ferrari Wagnitz	12/12/2024	Revisão final da arquitetura

1 Introdução

Este documento apresenta o projeto (*design*) do sistema. *CityGuiaGo*. A cidade de Vitória possui uma rica diversidade de atrações turísticas, mas muitos visitantes e até mesmo moradores enfrentam dificuldades ao planejar passeios e explorar os pontos turísticos disponíveis. Com o objetivo de solucionar esse problema, será desenvolvido um sistema interativo voltado para turistas que visitam a cidade.

O propósito do sistema é oferecer uma plataforma que facilite o planejamento de viagens e a exploração de atrações turísticas, promovendo experiências personalizadas para os usuários. Além disso, o sistema proporcionará suporte aos estabelecimentos locais, contribuindo para o fortalecimento do turismo regional, e oferecerá ferramentas administrativas para garantir a qualidade e segurança do conteúdo disponibilizado.

Essa solução busca promover uma interação mais eficiente entre turistas e a cidade, incentivando a descoberta de pontos turísticos e garantindo um planejamento prático e intuitivo.

Além desta introdução, este documento está organizado da seguinte forma: a Seção 3 apresenta a especificação dos requisitos não funcionais (atributos de qualidade), definindo as táticas e o tratamento a serem dados aos atributos de qualidade considerados condutores da arquitetura; a Seção 4 apresenta a arquitetura de software;

2 Plataforma de Desenvolvimento

Na Tabela 1 são listadas as tecnologias utilizadas no desenvolvimento da ferramenta, bem como o propósito de sua utilização.

Tabela 1 – Plataforma de Desenvolvimento e Tecnologias Utilizadas.

Tecnologia	**Versão**	**Descrição**	**Propósito**
Kotlin	2.0.0	Linguagem de programação moderna.	Desenvolver aplicativos multiplataforma e nativos para Android.
Kotlin Native	2.0.0	Extensão do Kotlin para compilação em código nativo.	Construir aplicativos que rodam sem uma JVM, como iOS ou sistemas embarcados.
SQLite	3.47.2	Banco de dados relacional leve.	Armazenar dados localmente em aplicativos móveis.
Python	3.13.1	Linguagem de programação de alto nível.	Desenvolver sistemas back-end e scripts de automação.

Tecnologia	Versão	Descrição	Propósito
Django	5.1.4	Framework Python para desenvolvimento de aplicações	Facilitar a estrutura do projeto, incluindo roteamento, servidor HTTP, Definição de Modelos e Regra de Negócio e integraçao com ORM.
DRF	3.15.2	Django Rest Framework - Framework para APIs baseado em Django.	Criar APIs RESTful para comunicação cliente-servidor.
Django Admin	5.1.4	Ferramenta administrativa automática do Django.	Gerenciar facilmente dados do sistema e autenticação.
Django ORM	5.1.4	Ferramenta de mapeamento objeto-relacional do Django.	Interagir com bancos de dados usando objetos Python.
PostgreSQL	17.2	Banco de dados relacional avançado.	Armazenar e gerenciar grandes volumes de dados de forma confiável.

Na Tabela 2 vemos os softwares que apoiaram o desenvolvimento de documentos e também do código fonte.

Tabela 2 – Softwares de Apoio ao Desenvolvimento do Projeto

Tecnologia	Versão	Descrição	Propósito
Overleaf	-	Plataforma online para edição de documentos LaTeX.	Colaborar e criar documentos acadêmicos ou técnicos.
Github	-	Plataforma de controle de versão baseada em Git.	Armazenar, compartilhar e versionar projetos de código.
Android Studio	2024.2.1	Ambiente de desenvolvimento integrado (IDE).	Criar aplicativos Android com suporte a emulação e depuração.
VSCode	1.86	Editor de código leve e extensível.	Desenvolver e editar código em diversas linguagens.
Poetry	1.8.5	Gerenciador de dependências para Python.	Facilitar o gerenciamento de bibliotecas e pacotes em projetos Python.
Draw.io	-	Ferramenta para criação de diagramas.	Desenvolver diagramas UML, de fluxo, e outros gráficos técnicos.
Ruff	0.8.3	Ferramenta de linting para código Python.	Detectar e corrigir erros ou más práticas no código Python.
Gradle	8.11.1	Ferramenta de automação de build.	Compilar, testar e empacotar projetos de software, especialmente em Kotlin e Java.
Jetpack Compose	1.7.5	Toolkit de UI declarativo para Android.	Criar interfaces de usuário modernas e reativas.
Postman	11.13.2	Plataforma para teste de APIs.	Testar, depurar e monitorar APIs RESTful de forma eficiente.

3 Requisitos Não Funcionais

A Tabela 3 apresenta a especificação dos requisitos não funcionais identificados no Documento de Especificação de Requisitos, os quais foram considerados condutores da arquitetura.

Tabela 3 – Especificação de Requisitos Não Funcionais.

RNF-1 – Exibir resultados de buscas e filtros em até 2 segundos.	
Categoria:	Desempenho
Tática / Tratamento:	Utilizar o armazenamento em cache do lado do servidor e do lado do aplicativo, visando a diminuição do tempo de resposta e o aumento da eficiência computacional.
Medida:	Verificar o tempo de resposta que o sistema leva para retornar os resultados de buscas e filtros após a requisição do usuário.
Critério de Aceitação:	O tempo médio para operações de buscas e filtros requisitadas pelo usuário deve ser igual ou inferior a 2 segundos em 95% dos casos.

RNF-2 – Armazenar dados com criptografia e em conformidade com a LGPD.	
Categoria:	Segurança
Tática / Tratamento:	Utilizar criptografia robusta em dados em descanso (data at rest) e em trânsito (data in transit), como AES-256 para armazenamento e TLS 1.3 para comunicação, bem como garantir o anonimato ou a pseudonimização de dados que não exijam identificação direta do usuário.
Medida:	• Verificar que 100% dos dados sensíveis estejam armazenados em formato criptografado no banco de dados. • Gerar relatórios que demonstrem que a coleta, armazenamento e uso de dados pessoais seguem as normas da LGPD, incluindo consentimento explícito e possibilidade de exclusão.
Critério de Aceitação:	– CA1: Todos os dados sensíveis devem ser criptografados e protegidos contra acessos não autorizados. Testes de verificação devem comprovar que os dados armazenados não podem ser lidos sem as chaves apropriadas. – CA2: Deve haver um registro de consentimento explícito para cada usuário, assegurando que suas informações foram coletadas e processadas com base nas diretrizes da LGPD. – CA3: Logs de acesso e ações realizadas no banco de dados devem ser claros, completos e acessíveis apenas por administradores autorizados. – CA4: O sistema deve passar por uma auditoria de conformidade de segurança, realizada por uma empresa ou especialista certificado, com relatório aprovando sua aderência à LGPD e padrões de criptografia.

RNF-3 – Garantir 99,9% de disponibilidade mensal para turistas acessarem o sistema.	
Categoria:	Confiabilidade
Tática / Tratamento:	Uso de exceções para detectar falhas de comunicação entre o usuário e o servidor, em caso de falha utilizar de um ponto de verificação funcional recente para uma reversão, assim como o uso de um monitor de processos para verificar a disponibilidade do componente.
Medida:	Percentual de falhas de login de usuários, segundo a fórmula a seguir $$\text{Disponibilidade (\%)} = \left(\frac{\text{Tentativas de login} - \text{Tentativas de login falhas}}{\text{Tentativas de login}}\right) \times 100$$ As tentativas de login falhas levam em consideração apenas as exceções de falha de comunicação com o servidor, desconsiderando falhas por erro do usuário
Critério de Aceitação:	O resultado obtido com a fórmula de Disponibilidade deve ser >= 99,9%

RNF-4 – Garantir responsividade para uso fluido em dispositivos móveis.	
Categoria:	Usabilidade
Tática / Tratamento:	Implementar interfaces adaptativas seguindo as diretrizes de Material Design, com layouts flexíveis que se ajustam a diferentes tamanhos de tela. Utilizar componentes nativos do Android para garantir consistência com o sistema operacional.
Medida:	Usuários conseguirem acessar, navegando da mesma forma com tempo de execução semelhante, o sistema através das diferentes plataformas.
Critério de Aceitação:	O tempo para acessar e utilizar uma mesma funcionalidade do sistema não deve diferir em 15% em diferentes dispositivos com telas entre 4 e 7 polegadas

4 Arquitetura de Software

A arquitetura do sistema CityGuiaGo, ilustrada na Figura 1, é projetada em torno de dois componentes principais: um aplicativo móvel nativo para Android, desenvolvido em Kotlin, e uma API REST, construída com Python e Django. Essa divisão permite que o sistema seja escalável, mantendo cada componente especializado em sua função e utilizando as ferramentas mais adequadas para cada tarefa. A arquitetura foi concebida com o objetivo de garantir desempenho, segurança e usabilidade, aproveitando as vantagens de cada tecnologia empregada. Ambos se baseiam numa combinação dos estilos arquitetônicos Camadas e Partições (FALBO, 2018).

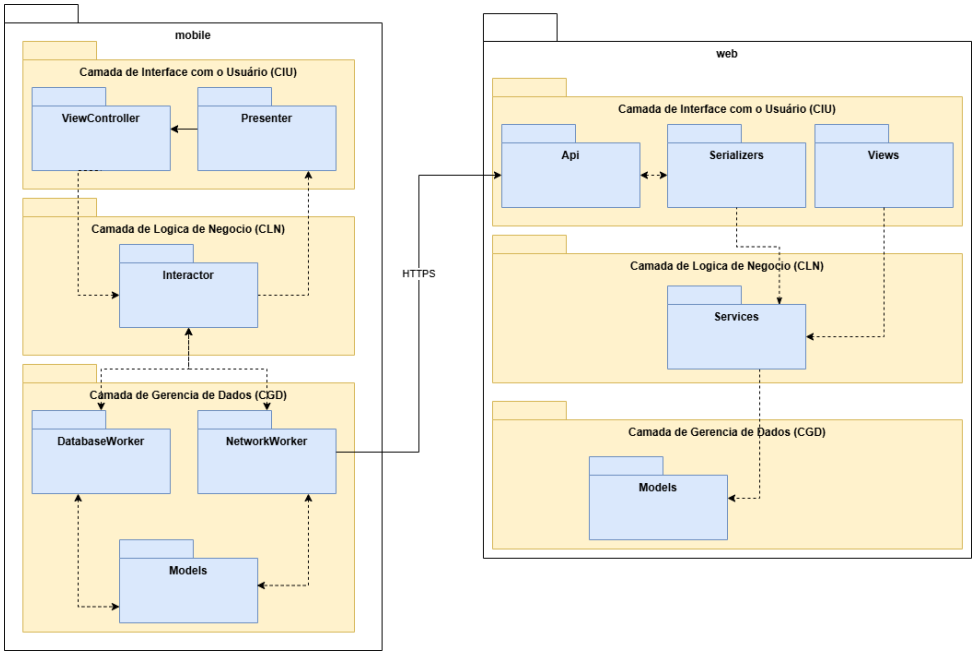

Figura 1 – Arquitetura do sistema CityGuiaGo.

4.1 Aplicativo Mobile Android

O aplicativo móvel, desenvolvido nativamente para Android utilizando a linguagem Kotlin e o framework Jetpack Compose para a interface de usuário, segue o padrão arquitetural VIP (View-Interactor-Presenter)(SOUZA, 2020), também conhecido como Clean Swift. Este padrão promove uma separação clara de responsabilidades, facilitando a testabilidade e a manutenibilidade do código.

4.1.1 Componentes do Aplicativo Mobile

- **ViewController:** Utilizando Jetpack Compose, este componente é responsável por renderizar a interface do usuário e capturar as interações do usuário, como cliques e entradas de texto. Sua principal função é apresentar a interface e delegar as ações do usuário para o Interactor.
- **Presenter:** Este componente atua como um intermediário entre o Interactor e o ViewController, recebendo os dados processados pelo Interactor e formatando-os para exibição na interface do usuário. O Presenter não contém lógica de negócio, concentrando-se apenas na preparação dos dados para apresentação.
- **Interactor:** O Interactor contém a lógica de negócio da aplicação, recebendo as requisições do ViewController, coordenando os Workers para obter e manipular dados, e devolvendo os resultados para o Presenter. Ele é independente da interface do usuário, o que facilita a testabilidade e o reuso de código.
- **Workers:**

  - *DatabaseWorker:* Utiliza o SQLite para gerenciar as operações de armazenamento e recuperação de dados localmente no dispositivo, o que permite o armazenamento em cache e o funcionamento parcial offline.
  - *NetworkWorker:* Responsável pela comunicação com a API REST, usando requisições HTTP para buscar e enviar dados.

- **Models:** Classes que representam as entidades do domínio da aplicação, como Atracao, Roteiro, Usuario, etc., servindo como modelo para dados que transitam entre as camadas do aplicativo.

4.2 API REST (Aplicação Web)

A API REST, desenvolvida com o framework Django e o Django REST Framework (DRF) em Python, utiliza o padrão REST para expor funcionalidades e dados do sistema. Ela é responsável por processar as requisições do aplicativo móvel, interagir com o banco de dados e retornar respostas apropriadas. A comunicação é feita através do protocolo HTTP e o formato de dados JSON.

4.2.1 Componentes da API REST

- **Api (Endpoints):** Definem os pontos de acesso da API, recebendo requisições HTTP e encaminhando-as para os serviços apropriados. As requisições são autenticadas com JWT para garantir a segurança e a autorização do acesso.

- **Serializers:** Responsáveis por serializar e desserializar os modelos, convertendo dados do banco de dados para JSON e vice-versa, facilitando a comunicação com o aplicativo móvel.

- **Views:** Gerenciadas pela biblioteca Django Admin, são responsáveis por possibilitar aos administradores do sistema uma forma fácil de visualizar e editar as informações do banco de dados (CRUD).

- **Services:** Implementam a lógica de negócio da aplicação, processando os dados e garantindo a integridade do sistema. Os serviços também interagem com o Django ORM para acessar e manipular os dados persistidos no banco de dados.

- **Models:** Representação das entidades do domínio no banco de dados PostgreSQL, definindo a estrutura e os relacionamentos entre os dados.

4.3 Comunicação entre Componentes

A comunicação entre o aplicativo móvel e a API REST é realizada através de requisições HTTP, utilizando o formato de dados JSON para o intercâmbio de informações. O aplicativo móvel, por meio do NetworkWorker, envia requisições para os endpoints da API. A API processa essas requisições, interage com o banco de dados PostgreSQL e retorna as respostas formatadas em JSON. O aplicativo móvel recebe essas respostas, desserializa os dados e os apresenta na interface do usuário.

A utilização de um cache local no aplicativo móvel, gerenciado pelo DatabaseWorker e armazenado em SQLite, melhora o desempenho da aplicação, reduzindo a necessidade de realizar requisições constantes à API. Isso também permite que a visualização das informações funcione de forma parcial mesmo em situações de falta de conexão com a internet, funcionalidade importante para os usuários que estarão em viagem.

A arquitetura do CityGuiaGo foi projetada levando em consideração os requisitos não-funcionais especificados na Seção 3:

- **Desempenho** (RNF-1): O cache local garante respostas rápidas e uma experiência de usuário fluida.
- **Segurança** (RNF-2): A utilização de JWT e HTTPS garante a segurança da comunicação.
- **Confiabilidade** (RNF-3): A separação entre a API (Web) e o cliente (Mobile) permite que cada componente seja escalado de forma independente, garantindo a disponibilidade e a estabilidade do sistema.
- **Usabilidade** (RNF-4): O uso do padrão VIP e do Jetpack Compose facilita a implementação de uma interface responsiva e seguindo os padrões de design nativos.

Referências

FALBO, R. A. *Projeto de Sistemas de Software - Notas de Aula*. 2018. Disponível em: <http://www.inf.ufes.br/~vitorsouza/falbo/Notas_Aula_Projeto_Sistemas_Falbo_2018.pdf>. Citado na página 5.

SOUZA, G. S. A. d. *Desenvolvimento de um aplicativo iOS para gestão de eventos com foco em qualidade*. 2020. Disponível em: <https://nemo.inf.ufes.br/wp-content/papercite-data/pdf/desenvolvimento_de_um_aplicativo_ios_para_gestao_de_eventos_com_foco_em_qualidade_2020.pdf>. Citado na página 6.