



**UNIVERSIDADE DE FORTALEZA**  
**VICE REITORIA DE GRADUAÇÃO**  
**CENTRO DE CIÊNCIAS TECNOLÓGICAS**  
**TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

MAYARA MEDINO MAIA  
GABRIEL MARFIM MENDES SILVA  
FRANCISCO LEVI DANTAS GOMES  
LIVIA CAVALCANTE BARROS RODRIGUES  
LUIZ EDUARDO BASTOS LIMA

**DOCUMENTAÇÃO TÉCNICA**  
Sistema TRAVO de Turismo e Eventos

FORTALEZA  
2025

**DOCUMENTAÇÃO TÉCNICA**  
Sistema TRAVO de Turismo e Eventos

Este documento contém a documentação técnica do Sistema TRAVO de Turismo e Eventos desenvolvido na componente curricular N393 - Projeto Aplicado Multiplataforma como requisito para obtenção de nota.

Supervisor: Prof. Bruno Lopes, Me

FORTALEZA  
2025

## SUMÁRIO

<b>1. INTRODUÇÃO</b>	<b>4</b>
1.1. CONTEXTO E JUSTIFICATIVA	4
1.2. OBJETIVOS	4
1.3. ESCOPO E DELIMITAÇÃO	5
<b>2. ENGENHARIA DE REQUISITOS</b>	<b>6</b>
2.1. REQUISITOS FUNCIONAIS (RFs)	6
2.2. REQUISITOS NÃO FUNCIONAIS (RNFs)	6
<b>3. PROJETO E ARQUITETURA DO SOFTWARE</b>	<b>8</b>
3.1. ARQUITETURA GERAL	8
3.2. PROJETO DO BANCO DE DADOS	8
3.3. PROJETO DE API	9
<b>4. TECNOLOGIAS E FERRAMENTAS</b>	<b>11</b>
4.1. STACK DE TECNOLOGIAS	11
4.2. FERRAMENTAS DE DESENVOLVIMENTO	11
<b>5. IMPLEMENTAÇÃO E RESULTADOS</b>	<b>13</b>
5.1. TELAS DO SISTEMA	13
<b>6. AMBIENTE E GUIA DE IMPLANTAÇÃO</b>	<b>14</b>
6.1. REQUISITOS DO AMBIENTE	14
6.2. PROCESSO DE IMPLANTAÇÃO	14
6.3. ACESSO À APLICAÇÃO IMPLANTADA	15
<b>7. CONCLUSÃO</b>	<b>16</b>
7.1. TRABALHOS FUTUROS	16
7.2. LIÇÕES APRENDIDAS	16

# 1. INTRODUÇÃO

## 1.1. CONTEXTO E JUSTIFICATIVA

*O turismo urbano e a participação em eventos culturais, gastronômicos e de entretenimento são atividades que movimentam significativamente a economia local. No entanto, turistas e até mesmo moradores enfrentam dificuldades para descobrir quais eventos estão acontecendo próximos de sua localização, além de encontrar informações consolidadas sobre locais de interesse, cardápios, avaliações e benefícios promocionais.*

*Atualmente, essas informações encontram-se fragmentadas: parte delas está em redes sociais, outras em sites específicos de eventos ou mesmo dependem de indicações boca a boca. Isso gera perda de oportunidades tanto para os usuários, que deixam de aproveitar experiências relevantes, quanto para os estabelecimentos, que perdem visibilidade e potenciais clientes.*

*O **TRAVO** surge como uma solução mobile que centraliza essas informações em um só lugar, proporcionando ao usuário uma experiência simplificada e prática. Com o TRAVO, o cliente poderá visualizar os eventos acontecendo ao seu redor, explorar locais próximos, consultar avaliações, cardápios e ainda utilizar cupons de desconto exclusivos.*

*Assim, o projeto busca atender tanto os **usuários finais (clientes)** que desejam explorar opções de lazer de forma prática, quanto os **estabelecimentos** que buscam ampliar sua divulgação e atrair novos consumidores.*

## 1.2. OBJETIVOS

*Desenvolver um aplicativo mobile que centralize informações sobre eventos e locais turísticos próximos ao usuário, oferecendo uma experiência personalizada que integre avaliações, cardápios, favoritos e cupons de desconto.*

*Dito isso, os objetivos específicos do projeto são:*

- *Implementar um sistema de autenticação com login, cadastro e recuperação de senha.*
- *Desenvolver um dashboard que exiba mapa com locais próximos, eventos em destaque e informações personalizadas para o usuário.*
- *Criar tela de visualização de locais, contendo resumo, cardápio, avaliações, redes sociais e cupons vinculados ao estabelecimento.*
- *Implementar uma área de cupons com listagem geral e funcionalidade de resgate.*
- *Desenvolver funcionalidade de favoritos para que o usuário possa salvar locais e acessá-los rapidamente.*
- *Criar tela de perfil para gerenciamento de dados pessoais e configurações básicas.*
- *Garantir feedback em tempo real ao usuário (mensagens de erro, confirmações de ações, carregamento dinâmico).*
- *Promover integração futura com clubes e descontos exclusivos para usuários assinantes.*

### 1.3. ESCOPO E DELIMITAÇÃO

- **Escopo:**
  - **Tela de Login:** autenticação com validação de credenciais e mensagens de erro.
  - **Tela de Cadastro:** formulário para criação de conta com campos essenciais.
  - **Tela de Recuperar Senha:** fluxo para redefinição de senha.
  - **Tela Dashboard:** exibição de mapa interativo, locais próximos, eventos em destaque e mensagem personalizada de boas-vindas.
  - **Tela de Local:** informações sobre estabelecimentos (resumo, cardápio, avaliações, cupons disponíveis).
  - **Tela de Cupons:** listagem completa de cupons de todos os estabelecimentos, com opção de resgate.
  - **Tela de Favoritos:** locais favoritados pelo usuário para acesso rápido.
  - **Tela de Perfil:** edição de informações pessoais (nome, e-mail, foto de perfil, etc.).

- **Configurações básicas:** *logout, notificações e preferências gerais.*
- **Delimitação (Fora do Escopo):**
  - *O aplicativo não incluirá gerenciamento financeiro, emissão de notas fiscais ou integração com sistemas de contabilidade.*
  - *Não será desenvolvido um módulo para cadastro direto de estabelecimentos (nesta versão, o foco é no cliente final).*
  - *Não haverá integração inicial com meios de pagamento ou reservas online.*
  - *O app não fará gestão logística ou controle de entregas.*

## 2. ENGENHARIA DE REQUISITOS

### 2.1. REQUISITOS FUNCIONAIS (RFs)

ID	Nome do Requisito	Descrição
RF01	Autenticação de Usuário	O aplicativo deve permitir que um usuário se autentique com e-mail e senha. Após o login bem-sucedido, o app deve manter a sessão ativa até que o usuário escolha sair.
RF02	Cadastro de Usuário	O aplicativo deve permitir que novos usuários se cadastrem informando nome, e-mail, data de nascimento e senha.
RF03	Recuperação de Senha	O aplicativo deve permitir que o usuário recupere a senha através do e-mail cadastrado.
RF04	Dashboard Interativo	O aplicativo deve exibir um dashboard inicial com mapa mostrando locais próximos à localização do usuário, eventos em destaque e mensagem de boas-vindas personalizada.
RF05	Localização	O aplicativo deve utilizar a localização do dispositivo para recomendar locais e eventos próximos.
RF06	Visualização de Locais	O aplicativo deve exibir informações de cada local, incluindo resumo, cardápio, avaliações e cupons disponíveis.
RF07	Favoritar Locais	O aplicativo deve permitir que o usuário favorite locais para acessá-los rapidamente depois.
RF08	Visualização de Favoritos	O aplicativo deve exibir a lista de locais favoritos salvos pelo usuário.
RF09	Listagem de Cupons	O aplicativo deve exibir todos os cupons disponíveis para o usuário, permitindo busca e filtros.
RF10	Resgate de Cupom	O aplicativo deve permitir que o usuário resgate cupons e exibir confirmação de sucesso.
RF11	Avaliações de Locais	O aplicativo deve exibir avaliações feitas por outros usuários sobre os locais.

RF12	Perfil do Usuário	O aplicativo deve permitir que o usuário edite informações pessoais como e-mail, senha e foto de perfil.
RF13	Logout	O aplicativo deve permitir que o usuário encerre sua sessão manualmente.
RF14	Configurações Básicas	O aplicativo deve permitir acesso às configurações, como notificações e preferências gerais.
RF15	Busca e Filtros no Mapa	O usuário deve poder buscar locais/eventos pelo nome e aplicar filtros (ex: tipo de local, categoria do evento).
RF16	Notificações Push	O aplicativo deve enviar notificações push para alertar sobre novos eventos e cupons disponíveis próximos ao usuário.

## 2.2. REQUISITOS NÃO FUNCIONAIS (RNFs)

### Desempenho

- **RNF01:** O tempo de inicialização do aplicativo não deve exceder **3 segundos** em dispositivos compatíveis.
- **RNF02:** O mapa no dashboard deve ser carregado em até **2 segundos** após a inicialização da tela.
- **RNF03:** A navegação entre telas deve ocorrer em menos de **1 segundo**.

### Usabilidade

- **RNF04:** A interface deve seguir as diretrizes de **Material Design 3 (Android)**, garantindo uma experiência consistente e intuitiva.
- **RNF05:** O aplicativo deve oferecer feedback imediato para todas as ações do usuário (ex: mensagens de erro, confirmações, loaders).

### Compatibilidade

- **RNF06:** O aplicativo deve ser totalmente funcional em dispositivos Android a partir da versão **9.0 (API 28)**.
- **RNF07:** O app deve se adaptar automaticamente a diferentes tamanhos de tela (smartphones e tablets).



## Segurança

- **RNF08:** O token de autenticação do usuário deve ser armazenado de forma segura utilizando **EncryptedSharedPreferences (Android Jetpack)**.
- **RNF09:** Todas as comunicações com o backend devem ser realizadas via **HTTPS (TLS 1.2 ou superior)**.
- **RNF10:** As senhas devem ser armazenadas de forma criptografada no servidor (ex: **bcrypt**).

## Consumo de Recursos

- **RNF11:** O aplicativo deve minimizar o consumo de dados móveis utilizando cache local para imagens e informações já carregadas.

### 3. PROJETO E ARQUITETURA DO SOFTWARE

#### 3.1. ARQUITETURA GERAL

**Instrução:** *Aqui você deve apresentar a arquitetura interna do seu aplicativo. Em vez da arquitetura do servidor, descreva o padrão de design que você usou para organizar o código do app (ex: MVVM, MVI, Clean Architecture). O diagrama deve ilustrar as camadas do aplicativo e como elas interagem entre si.*

**Exemplo de Texto:** O aplicativo RochaForte foi desenvolvido seguindo a arquitetura **MVVM (Model-View-ViewModel)**, recomendada oficialmente pelo Google para o desenvolvimento Android. Esta arquitetura promove a separação de responsabilidades, facilitando a testabilidade e a manutenção do código.

As camadas são divididas da seguinte forma:

1. **View (Apresentação):** Composta por Activities/Fragments e construída com Jetpack Compose. É responsável por exibir os dados na tela e capturar as interações do usuário, delegando toda a lógica para a ViewModel.
2. **ViewModel:** Contém a lógica de apresentação e gerencia o estado da UI. Ela sobrevive a mudanças de configuração (como a rotação da tela) e expõe os dados para a View através de **StateFlows**. A ViewModel não conhece a View, apenas notifica sobre mudanças de estado.
3. **Model (Dados):** Composta pela camada de **Repositório** (Repository), que é a única fonte da verdade para os dados. O Repositório decide de onde buscar os dados: de uma fonte remota (**Remote Data Source**, que consome a API REST) ou de uma fonte local (**Local Data Source**, que acessa o banco de dados Room para dados em cache).

**[INSERIR IMAGEM DO DIAGRAMA DA ARQUITETURA MVVM AQUI]**

#### 3.2. PROJETO DE DADOS: INTEGRAÇÃO COM API E PERSISTÊNCIA LOCAL

**Instrução:** *Nesta seção, explique como seu aplicativo gerencia os dados. A fonte primária de informações é a API REST, mas um aplicativo moderno quase sempre precisa de uma estratégia de persistência local (um banco de dados no dispositivo) para funcionar como um cache. Descreva como essas duas fontes de dados trabalham juntas. Explique sua estratégia: o app busca os dados da API e os salva localmente para depois exibi-los? Isso permite que o app funcione offline e melhora*

*drasticamente a performance, pois nem sempre é necessário buscar dados na internet.*

### Exemplo de Texto:

A estratégia de dados do aplicativo RochaForte é projetada para performance e resiliência, utilizando a API REST como a fonte da verdade e um banco de dados local como um cache otimizado. A interação entre essas duas fontes é orquestrada pela camada de Repositório, seguindo o padrão "Single Source of Truth" (Fonte Única da Verdade) no cliente.

O fluxo de dados funciona da seguinte maneira:

1. **Requisição de Dados:** A ViewModel solicita dados (ex: a lista de lajes) ao Repositório.
2. **Consulta à API:** O Repositório sempre busca a versão mais recente dos dados na API REST através da nossa camada de rede (Retrofit).
3. **Persistência no Cache:** Ao receber a resposta da API, o Repositório imediatamente salva os dados recebidos no banco de dados local (Room), sobrescrevendo os dados antigos daquela consulta.
4. **Exibição a Partir do Local:** A ViewModel observa os dados diretamente do banco de dados local. Assim que os novos dados são salvos, a interface é atualizada automaticamente de forma reativa.

Essa abordagem garante que:

- **Performance:** A UI é sempre populada a partir do banco de dados local, que é extremamente rápido.
- **Suporte Offline:** Se o dispositivo estiver sem conexão, o Repositório falha ao buscar na API, mas a UI continua exibindo os últimos dados que foram salvos no cache local, permitindo que o aplicativo continue funcional para consulta.

A tecnologia escolhida para a persistência local foi o Room Persistence Library. A principal entidade armazenada em cache é a [LajeCacheEntity](#).

**Dicionário de Dados** (Tabela [Lajes\\_Pedra](#)):

Nome do campo	Tipo de dados	Chave (PK/FK )	Nulo?	Descrição
id	UUID	PK	Não	UUID da laje, vindo da API.
tipo_material	VARCHAR(100)		Não	Ex: "Mármore Carrara", "Granito Preto Absoluto"

dimensoes_cm	JSONB		Não	Objeto JSON com altura, largura, espessura
url_foto_thumb	VARCHAR(255)		Sim	URL de uma versão em miniatura da foto.
status	VARCHAR(20)		Não	"Disponível", "Reservado", "Vendido"
last_updated	TIMESTAMPZ		Não	Timestamp de quando o dado foi Exportar para as Planilhas

### 3.3. CONSUMO DA API E FLUXO DE NAVEGAÇÃO

**instrução:** Esta seção tem dois objetivos. Primeiro, referenciar a documentação oficial da API que o seu aplicativo consome. Segundo, e mais importante, apresentar um **Diagrama de Fluxo de Navegação** que mostre as telas do seu aplicativo e como o usuário transita entre elas.

#### **Exemplo de Texto:**

O aplicativo consome a API REST do sistema RochaForte, cuja documentação completa, interativa e oficial, desenvolvida com OpenAPI, pode ser acessada através do seguinte link:

- **URL da Documentação da API:**

<https://myprofile.github.io/rochaforte-api-docs/>

O fluxo de navegação do usuário dentro do aplicativo foi projetado para ser simples e intuitivo, conforme ilustrado no diagrama a seguir:

**Figura 5: Diagrama de Fluxo de Navegação do Aplicativo** (Legenda: O fluxo principal inicia na tela de Login. Após o sucesso, o usuário é direcionado para a Home (lista de lajes), de onde pode navegar para a tela de Detalhes de uma laje específica ou para a tela de Cadastro de uma nova laje.)

[INSERIR IMAGEM DO DIAGRAMA DE FLUXO DE NAVEGAÇÃO AQUI (SPLASH -> LOGIN -> HOME -> DETALHES)]



## 4. TECNOLOGIAS E FERRAMENTAS

### 4.1. STACK DE TECNOLOGIAS

**Instrução:** Substitua completamente a stack web pelas tecnologias, linguagens e bibliotecas usadas no desenvolvimento do seu aplicativo mobile.

**Exemplo de Texto:**

- **Linguagem:** Kotlin, por ser a linguagem oficial para o desenvolvimento Android, oferecendo segurança (null-safety) u concisão.
- **Arquitetura:** Android Architecture Components (ViewModel, LiveData, StateFlow, Room).
- **UI Toolkit:** Jetpack Compose, para a construção declarativa e moderna da interface do usuário.
- **Injeção de Dependência:** Hilt, para simplificar a injeção de dependências no projeto.
- **Consumo de API:** Retrofit 2 e OkHttp 3, para realizar chamadas de rede à API REST de forma eficiente.
- **Banco de Dados Local:** Room, para persistência de dados e implementação de cache offline.
- **Carregamento de Imagens:** Coil, uma biblioteca moderna e performática para carregar imagens da rede.

### 4.2. FERRAMENTAS DE DESENVOLVIMENTO

**Instrução:** Liste as ferramentas que apoiaram o processo de desenvolvimento, desde a escrita do código até o gerenciamento das tarefas da equipe.

**Exemplo de Texto:**

- **IDE:** Visual Studio Code foi a IDE padrão para toda a equipe, devido à sua leveza, extensibilidade e terminal integrado.
- **Controle de Versão:** Git, com o repositório hospedado no GitHub. Adotamos o fluxo de trabalho "GitFlow", com branches separadas para **develop**, **features** e **main**.
- **Gerenciamento de Projeto:** Trello foi utilizado para gerenciar as tarefas. Criamos um quadro Kanban com as colunas "A Fazer", "Em Andamento", "Em Teste" e "Concluído".

- **Ferramenta de API:** *Insomnia* foi usado para testar os endpoints da API durante o desenvolvimento.

## 5. IMPLEMENTAÇÃO E RESULTADOS

### 5.1. TELAS DO SISTEMA

**Instrução:** Esta é a vitrine do seu projeto. Insira imagens (screenshots) das principais telas da sua aplicação. Cada imagem deve ter uma legenda curta e clara explicando sua finalidade. Dê preferência a telas que demonstrem as funcionalidades mais importantes (RFs) que você listou na seção 2.

#### **Exemplo de Texto:**

**Figura 1: Tela de Login** (Legenda: A tela de login é o ponto de entrada do sistema, garantindo o acesso seguro através de autenticação por e-mail e senha.)

**[INSERIR SCREENSHOT DA TELA DE LOGIN AQUI]**

---

**Figura 2: Dashboard Principal com Visão Geral do Estoque** (Legenda: Após o login, o administrador tem acesso a um dashboard com métricas rápidas sobre o inventário, como o número de peças disponíveis, reservadas e vendidas.)

**[INSERIR SCREENSHOT DO DASHBOARD AQUI]**

---

**Figura 3: Tela de Cadastro de Laje de Pedra** (Legenda: Formulário detalhado para o cadastro de uma nova peça no inventário, permitindo o upload de foto e a inserção de todas as especificações técnicas.)

**[INSERIR SCREENSHOT DO FORMULÁRIO DE CADASTRO AQUI]**



## 6. AMBIENTE E GUIA DE GERAÇÃO (BUILD)

**Instrução:** Esta seção detalha os requisitos e os passos para compilar o código-fonte e gerar o arquivo de instalação do aplicativo (o .apk).

### 6.1. REQUISITOS DO AMBIENTE

**Instrução:** Liste o software e as versões necessárias para que outra pessoa possa compilar seu projeto com sucesso.

**Exemplo de Texto:**

- **IDE:** Android Studio Iguana | 2023.2.1
- **Android Gradle Plugin (AGP):** 8.2.0
- **Gradle:** 8.2
- **Android SDK:** `compileSdk = 34`, `minSdk = 28`
- **JDK:** JDK 17 (embutido no Android Studio)

### 6.2. PROCESSO DE GERAÇÃO DE APLICATIVO

**instrução:** Forneça os comandos exatos para gerar uma versão de "release" do seu aplicativo a partir da linha de comando.

**Exemplo de Texto:**

1. Clone o repositório do projeto: `git clone https://github.com/equipe/rocha-forte-mobile.git`
  2. Entre na pasta do projeto: `cd rocha-forte-mobile`
  3. No Windows, execute o comando: `gradlew.bat assembleRelease`
  4. No Linux ou macOS, execute o comando: `./gradlew assembleRelease`
  5. Após a conclusão, o arquivo de instalação será gerado em:  
`app/build/outputs/apk/release/app-release.apk`
- 1.

### 6.3. ACESSO À APLICAÇÃO IMPLANTADA

**Instrução:** Forneça um link para o download direto do arquivo .apk ou para uma plataforma de testes (como Firebase App Distribution) onde a banca possa instalar o aplicativo.

**Exemplo de Texto:**

- **Link para Download do APK:** [Link do Google Drive/Dropbox para o arquivo .apk]
- **Credenciais de Acesso (Perfil de Vendedor):**
  - **Usuário:** vendedor@teste.com
  - **Senha:** vendedor123

## 7. CONCLUSÃO

### 7.1. TRABALHOS FUTUROS

***Instrução:** Nenhum projeto está 100% completo. Liste aqui as melhorias e novas funcionalidades que você gostaria de implementar no futuro. Pense em como o aplicativo poderia se tornar ainda mais útil para o usuário. Isso demonstra visão de produto e consciência das limitações do trabalho atual.*

#### **Exemplo de Texto:**

Com a base sólida do aplicativo estabelecida, identificamos diversas oportunidades para evoluir e agregar ainda mais valor ao sistema RochaForte no futuro. As principais propostas são:

- **Funcionalidades Offline Avançadas:** Expandir a capacidade offline para permitir não apenas a consulta, mas também a **criação e edição de pedidos** sem conexão com a internet. Os dados seriam sincronizados automaticamente com o servidor assim que uma conexão fosse restabelecida.
- **Identificação de Lajes por QR Code:** Implementar uma funcionalidade que utilize a câmera do dispositivo para ler um QR Code fixado em cada laje de pedra. Isso permitiria ao vendedor ou gerente de estoque acessar instantaneamente os detalhes da peça, eliminando a necessidade de busca manual e agilizando o processo de inventário.
- **Notificações Push em Tempo Real:** Desenvolver um sistema de notificações push para alertar os vendedores sobre eventos importantes, como a confirmação de um pedido, a chegada de um novo lote de material ou uma alteração no status de uma laje que ele esteja monitorando.
- **Otimização para Tablets e Modo Paisagem:** Adaptar a interface do usuário para oferecer uma experiência otimizada em telas maiores, como as de tablets, que são frequentemente utilizados em balcões de vendas. Isso incluiria layouts de duas colunas e melhor aproveitamento do espaço horizontal.
- **Versão para a Plataforma iOS:** Iniciar o desenvolvimento da versão do aplicativo para iOS, a fim de atender a todos os potenciais usuários da empresa, independentemente do sistema operacional de seus dispositivos móveis.

## 7.2. LIÇÕES APRENDIDAS

***Instrução:** Faça uma reflexão sincera sobre a jornada de desenvolvimento do aplicativo. Quais foram os maiores desafios técnicos que a equipe enfrentou no universo mobile? Quais foram as dificuldades de integrar uma API externa? Como foi o trabalho em equipe? O que vocês fariam de diferente hoje? Esta seção valoriza o processo de aprendizado tanto quanto o resultado final.*

### **Exemplo de Texto:**

O desenvolvimento do aplicativo RochaForte foi uma experiência de aprendizado imensa, que nos levou muito além da simples escrita de código. Nossas principais lições aprendidas podem ser divididas em três áreas:

1. **Desafios Técnicos de Arquitetura:** A maior dificuldade técnica que enfrentamos foi, sem dúvida, o gerenciamento de estado e a orquestração de operações assíncronas. Inicialmente, tínhamos dificuldade em manter a interface consistente enquanto os dados eram buscados da API e salvos no banco de dados local. Compreender e aplicar corretamente a arquitetura MVVM, utilizando `StateFlows` para expor o estado da UI a partir da `ViewModel`, foi um divisor de águas. Isso nos ensinou que uma arquitetura bem definida não é opcional, mas sim essencial para criar um aplicativo robusto e manutenível.
2. **A Importância da Persistência Local:** No início, subestimamos a complexidade de oferecer um suporte offline funcional. Implementar o padrão de Repositório como a "Fonte Única da Verdade" que abstrai a origem dos dados (API ou cache local) foi o nosso maior "Aha! Moment". Aprendemos na prática que um aplicativo moderno não apenas consome uma API, mas gerencia dados de forma inteligente para ser rápido e resiliente, melhorando drasticamente a experiência do usuário.
3. **Comunicação entre Equipes (Frontend/Backend):** A integração com a API, desenvolvida pela equipe de Web, foi um grande exercício de comunicação. Depender da documentação OpenAPI foi fundamental e nos ensinou o valor de ter um "contrato" claro entre o cliente e o servidor. Tivemos momentos em que precisávamos de um campo extra ou de um formato de dado diferente, e a negociação com a outra equipe para evoluir a API foi um aprendizado valioso sobre o desenvolvimento colaborativo no mundo real.

A principal lição que levamos deste projeto é que a qualidade de um aplicativo mobile não está apenas em sua aparência, mas em sua arquitetura resiliente e na forma como ele lida com as incertezas do mundo real, como falhas de rede.

