Documentação do Aplicativo: Hommy

2021

Sumário

1. Introdução

Este documento tem como objetivo demonstrar as tecnologias utilizadas, seus componentes, funcionalidades e como os mesmos foram utilizados para auxiliar a criação do aplicativo Hommy, desenvolvido para auxiliar os estudantes a divulgarem e encontrarem vagas em repúblicas na sua cidade, bem como caronas, prestação de serviços variados e eventos que ocorrem na região.

Consideraremos que você já possua o ambiente pronto, qualquer dúvida pode consultar a outra documentação sobre a configuração: (https://drive.google.com/file/d/1XB5LhZUFpLAtG3BJkHV1i-nXhqxupfcf/view?usp=sharing).

2. Tecnologias utilizadas

Para o desenvolvimento do aplicativo, utilizamos as seguintes tecnologias:

React Native;
React Navigator;
Redux;
Hooks;
NodeJS
JSON;
MongoDB;
Firebase.

Falaremos sobre cada uma delas abaixo.

Observação: Na pasta do seu projeto, crie uma pasta chamada src. Dentro dela colocaremos todos os códigos criados aqui.

2.1. React Native

O React Native é um framework para desenvolvimento de aplicativos multiplataforma. Com ele, poderemos criar aplicativos tanto para sistema Android quanto para IOS utilizando um único código em JavaScript, pois todo o código desenvolvido com o React Native será convertido para linguagem nativa do sistema operacional. Ele foi responsável pela parte visual do projeto, sendo a base de todo o frontend.

Abordaremos algumas de suas funcionalidades abaixo.

2.1.1. Importação

Ao completar a instalação do React Native, ele já traz consigo diversas bibliotecas, mas você pode adicionar outras a medida que for necessário. Para isso, utilizamos o comando “Import”.

Como exemplo, importaremos a biblioteca do react:

```
import React from 'react';
```

A importação também é usada para adicionar componentes ao projeto, veremos isso adiante.

2.1.2. Classes

Precisaremos definir uma classe sempre que formos utilizar um componente. Para isso, se faz necessária a importação do React e {Component}, da seguinte forma:

```
import React, {Component} from 'react';
```

Após a importação, podemos criar a nossa classe. O padrão para a criação é:

```
class Nome_da_Classe extends Component{ };
```

Tudo o que será definido para essa classe fazer deve ser escrito dentro das chaves. Também precisaremos importar o que for necessário para o desenvolvimento da classe. Como exemplo:

```
import {Text, Image, Button} from 'react-native';
```

Por padrão, fazemos a exportação da classe no final do código através do comando:

```
export default Nome_da_classe;
```

Porém, a exportação também pode ser feita logo na definição da classe, como a seguir:

```
export default class Nome_da_Classe extends Component{ };
```

2.1.3. Componentes

O React já traz vários componentes que nos ajudam a criar nossa interface de usuário, utilizaremos como exemplo de demonstração os componentes View, Text e Button. Para usá-los, primeiro faremos a importação dos mesmos:

```
import {View, Text, Button} from 'react-native';
```

É importante que os componentes sejam definidos dentro de chaves e separados por vírgula, como no exemplo acima.

Agora utilizaremos o que foi dito neste item e no 2.1.2 para definirmos uma classe e usarmos os componentes View, Text e Button. O código completo ficaria da seguinte forma:

```
import React from 'react';
import React, {Component} from 'react';
import {View, Text, Button} from 'react-native';
class exemplo_classe extends Component{
    render(){
        return(
            <View>
                <Button title = "Título do Botão"/>
                <Text> Texto de exemplo</Text>
            </View>
        );
    }
 };
export default exemplo_classe;
```

2.2. React Navigation

O React Navigation é uma biblioteca que nos permitirá navegar e fazer a conexão entre as páginas em uma aplicação React Native, trazendo as informações de uma para outra.

Para navegar entre telas, você usará um StackNavigator, que funciona exatamente como uma pilha, colocando sempre a tela que você está para o topo. Ao sair da tela, ela é retirada do topo da pilha.

2.2.1. Utilizando o Navigation

Para começarmos a utilização do Navigation, precisaremos criar um arquivo JavaScript, no nosso projeto ele foi nomeado de Routes.js. Dentro dele, faça a importação dos seguintes itens:

```
import React from 'react';
import { createAppContainer } from 'react-navigation';
import { createStackNavigator } from 'react-navigation-stack';
```

Importaremos também as telas que queremos criar a conexão, temos como exemplo a tela Login e a Cadastro (observação: para importação das telas, lembre-se de colocar o caminho corretamente em que o arquivo .js se encontra. No nosso projeto, as telas se encontram dentro da página denominada Pages).

```
import Login from './pages/Acesso/Login';
import Cadastro from './pages/Republica/Cadastro/index';
```

Agora precisaremos criar uma constante que receberá o createStackNavigator. A chamamos de Navegação.

```
const Navegação = createStackNavigator()
```

Dentro da const Navegação, definiremos as telas que adicionaremos ao Navigator (Login e Cadastro). O padrão para isso é *nome_da_tela : nome_do_componente_importado*.

```
const Navegação = createStackNavigator(
    {
        Login: Login,
        Cadastro: Cadastro,
    }
);
```

Podemos definir uma página que será a primeira a ser renderizada. Para isso, utilizamos o comando *initialRouterName : nome_do_pagina*

```
const Navegação = createStackNavigator(
    {
        Login: Login,
        Cadastro: Cadastro,
    }
    {
        initialRouteName: 'Login',
    }
);
```

.

O Navigator por padrão insere um cabeçalho em todas as nossas páginas, porém utilizaremos um personalizado. Para isso, desabilitaremos o cabeçalho automático da seguinte forma:

```
const Navegação = createStackNavigator(
    {
        Login: Login,
        Cadastro: Cadastro,
    }
    {
        initialRouteName: 'Login',
        headerMode: 'none',
        navigationOptions: {
            headerVisible: false
        }
    }
);
```

Agora faremos a exportação da classe, mas um pouco diferente do que foi mencionado anteriormente. Será passado como default o *createAppContainer()* e, como parâmetro, a nossa const Navegação.

```
export default createAppContainer(Navegação);
```

Logo, nosso código completo ficará da seguinte forma:

```
import React from 'react';
import { createAppContainer } from 'react-navigation';
import { createStackNavigator } from 'react-navigation-stack';
import Login from './pages/Acesso/Login';
import Cadastro from './pages/Republica/Cadastro/index';

const Navegação = createStackNavigator(
    {
        Login: Login,
        Cadastro: Cadastro,
    }
    {
        initialRouteName: 'Login',
        headerMode: 'none',
        navigationOptions: {
            headerVisible: false
        }
    }
);
export default createAppContainer(Navegação);
```

2.2.2. Adicionando botões para navegar entre as telas

Para navegar entre as telas através de botões, vá até o arquivo da tela que deseja adicionar o botão e importe o *withNavigation do 'react-navigation' e Button do 'native-base'.*

```
import { withNavigation } from 'react-navigation';
import { Button } from 'native-base';
```

Como exemplo, adicionaremos na tela Login um botão para a tela Cadastro. A propriedade para isso é a onPress, utilizada dentro do <Button/>, que receberá uma função da tela que desejamos adicionar através do comando *this.props.navigation.navigate('nome_da_tela')*.

```
<Button
  title="Faça seu cadastro"
  onPress={() =>
    this.props.navigation.navigate('Cadastro')
  }
/>
```

Dessa forma, ao clicarmos no botão “Faça seu cadastro”, seremos redirecionados para a tela Cadastro.

Para fazer a exportação de uma classe utilizando o withNavigation, utilizaremos o padrão *export default withNavigation(nome_da_tela);*. O código de uma tela genérica de login utilizando essa propriedade ficaria assim:

```
import React from 'react';
import { withNavigation } from 'react-navigation';
import { Button } from 'native-base';

class Login extends React.Component {
  render() {
    return (
      <View>

        <Button
          title="Faça seu cadastro"
          onPress={() =>
            this.props.navigation.navigate('Cadastro')
          }
        />

      </View>
    );
  }
}
export default withNavigation(Login);
```

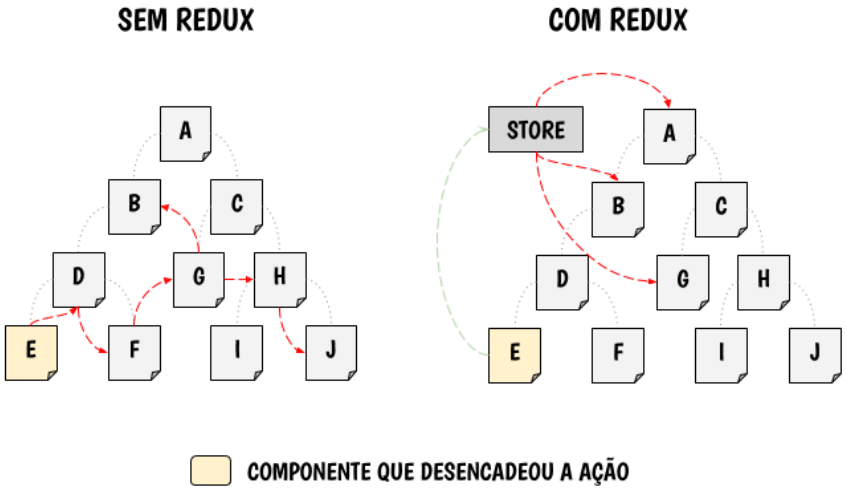
2.3. Redux

O Redux é uma biblioteca que nos auxilia no gerenciamento dos estados globais de uma aplicação, facilitando o compartilhamento entre componentes. O state de um componente nos retorna sobre um escopo específico no qual ele está inserido, e se precisarmos informar esse estado para outro componente que pode ou não estar no mesmo nível de hierarquia?

É aí que entra o Redux, ele utiliza estados globais: actions que serão disparadas, reducers que irão modificar o estado conforme necessário e stores para gerenciar tudo isso.

- Actions: são fontes de informações que são enviadas da aplicação para a Store. São disparadas pelas Action Creators, que são simples funções que, ao serem executadas, ativam os Reducers.
- Reducers: recebem e tratam as informações para que sejam ou não enviadas à Store.
- Store: é o container que armazena e centraliza o estado geral da aplicação. Ela é única, podemos ter apenas uma Store por aplicação

Dessa forma, teremos uma maneira de recuperar os estados dos componentes de uma forma mais ágil e simplificada, como pode ser observado na imagem a seguir:



Observando a imagem acima, podemos perceber que o Redux simplifica o acesso aos estados dos componentes de uma aplicação, repassando essa responsabilidade uma Store.

2.4. Hooks

Hooks é uma funcionalidade introduzida no React 16.8 que nos oferece uma nova forma de utilizar estado, ciclo de vida, entre outras funcionalidades sem a necessidade de escrevermos componentes com classe. Através dela, reestruturamos o código a fim de permitir a reutilização do estado e da lógica entre componentes, além do código com Hooks ser mais simples e fácil de ser lido.

2.5. NodeJS

O Node.js é um ambiente de execução que, diferente do que estamos acostumados, nos permite criar aplicações Javascript para rodar sem a necessidade de um navegador. Foi utilizado para construir o backend do projeto através de módulos, o que nos oferece a possibilidade de reutilizar o código, tendo em vista que os trechos são executados em contextos isolados, executando uma única função.

2.6. Json

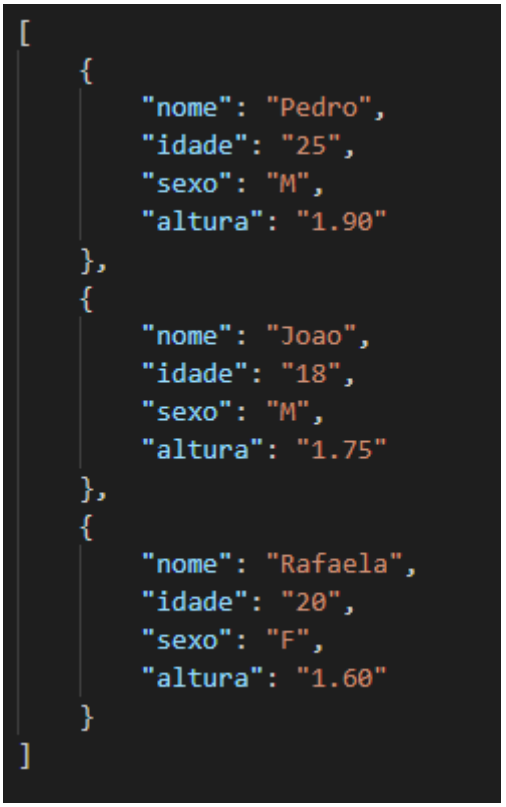
O Json é um formato, leve e fácil de ler, de troca de informações/dados entre sistemas, que foi utilizado para comunicar as informações entre o banco de dados, backend e frontend.

Utiliza-se uma estrutura de Array e Objetos que possuem chaves (que representam os nomes dos atributos) e valores (referentes ao valor do objeto).

Os dados em Json são escritos em pares “chave”:”valor” .A sintaxe é bem simples, temos os elementos:

- { e } - delimita um objeto.
- [e] - delimita um array.
- : - separa chaves de valores.
- , - separa os atributos chave/valor.

Para exemplificar, criaremos um array com objetos que representarão pessoas, as quais possuem os atributos: nome, idade, sexo e altura.



2.7. MongoDB

Atualmente, temos no mercado vários problemas de persistência de dados que o SQL tradicional não resolve. Para essa aplicação, temos como alternativa os bancos de dados não-relacionais, como é o caso do MongoDB, que pode ser considerado o principal da categoria.

Os bancos não-relacionais armazenam seus dados em documentos autossuficientes, ou seja, cada documento contém todos os dados necessários para o seu funcionamento, sem depender de outras tabelas (através de chaves estrangeiras) como funciona nos bancos que seguem o modelo relacional.

Esses documentos, por sua vez, podem ser escritos utilizando a estrutura do Json, como explicado no tópico 2.6 deste documento.

Para exemplificar a diferença entre os modelos, criaremos uma estrutura com os atributos Nome, Cidade e Estado. Utilizando bancos relacionais, primeiro precisaremos criar uma tabela com os campos e definir o tamanho e tipo de dado de cada um.

```
CREATE TABLE Usuarios (
    Usuarios_id serial PRIMARY KEY,
    Nome varchar(30),
    Cidade varchar(30),
    Estado varchar(2)
);
```

Depois de criada a tabela, faz-se necessário mais um comando para inserir os dados desejados.

```
INSERT INTO Usuarios (Nome, Cidade, Estado)
VALUES ('Antonio', 'Alegre', 'ES');
INSERT INTO Usuarios (Nome, Cidade, Estado)
VALUES ('Pedro', 'Salvador', 'BA');
```

Já utilizando o MongoDB, ficaria da seguinte forma:

```
db.usuarios.insert(
    {
        nome: "Antonio",
        cidade: "Alegre",
        estado: "ES"
    },

    {
        nome: "Pedro",
        cidade: "Salvador",
        estado: "BA"
    }
);
```

No documento criado para o MongoDB não temos nenhuma definição

de tipos dos dados, tamanho máximo para cada atributo, regras de validação ou restrições. Dessa forma, podemos armazenar qualquer tipo de documento no banco de dados e, diferentemente do relacional onde a coluna deve ser preenchida em todos os registros, caso seja necessário adicionar um novo atributo, podemos incluí-lo apenas no documento onde ele é necessário.

2.8. Firebase

O Firebase é uma coleção de ferramentas oferecida pelo Google que auxilia o desenvolvimento e manipulação de aplicativos, disponibilizando serviços prontos para a utilização do desenvolvedor, poupando um tempo importante que poderá ser designado para concluir outra tarefa no projeto, ao invés de recriar algo que já existe. No nosso projeto, foi utilizado para armazenar fotos, enviar mensagens push e fazer login através de uma conta google, por exemplo.

3. Módulos

Quando tratamos com aplicações menores, é comum a escrita de todo o código em poucos arquivos contendo tudo que é necessário para o seu funcionamento. Porém, quando isso evolui para um cenário maior em aplicativos com muito código, pode acabar prejudicando a fluidez, demandando um maior tempo de carregamento, apresentando lentidão na exibição dos dados e até mesmo travamentos durante a execução.

Uma alternativa para esse problema seria dividir o código em módulos que serão chamados apenas em situações que fossem necessário o seu uso, diminuindo o tempo de carregamento da página inicial do aplicativo e até mesmo dos módulos, sempre que forem exibidos, tendo em vista que será trazido apenas o código necessário para uma determinada ação a ser executada.

No nosso projeto, o desenvolvimento dos módulos segue um fluxo muito parecido, portanto, utilizaremos como exemplo o módulo República para demonstrar o processo realizado em sua construção.

3.1. Repúblicas

Inicialmente, foi criada a parte visual do App que seria usada como base para os demais módulos, pois os elementos como cards, botões, telas de detalhes e modais serão reutilizados durante o desenvolvimento. Após o frontend de todas as páginas do módulo estarem prontas, foram criadas as conexões entre elas através do Navigator.

Para fazer a validação dos campos de cadastro no frontend, foi aplicado o Formik, uma biblioteca que facilita o manuseio de formulários. Para o armazenamento das imagens, bem como sua recuperação para

utilizar na exibição das telas posteriormente, foi feita a conexão com o Firebase.

Juntamente com o desenvolvimento do frontend, foi criada a estrutura básica de um CRUD (create, read, update e delete). De forma reduzida, um CRUD é a junção de 4 funções básicas de um sistema que trabalha com banco de dados:

- Create: criar um novo registro
- Read: ler as informações de um registro
- Update: atualizar os dados do registro
- Delete: apagar um registro

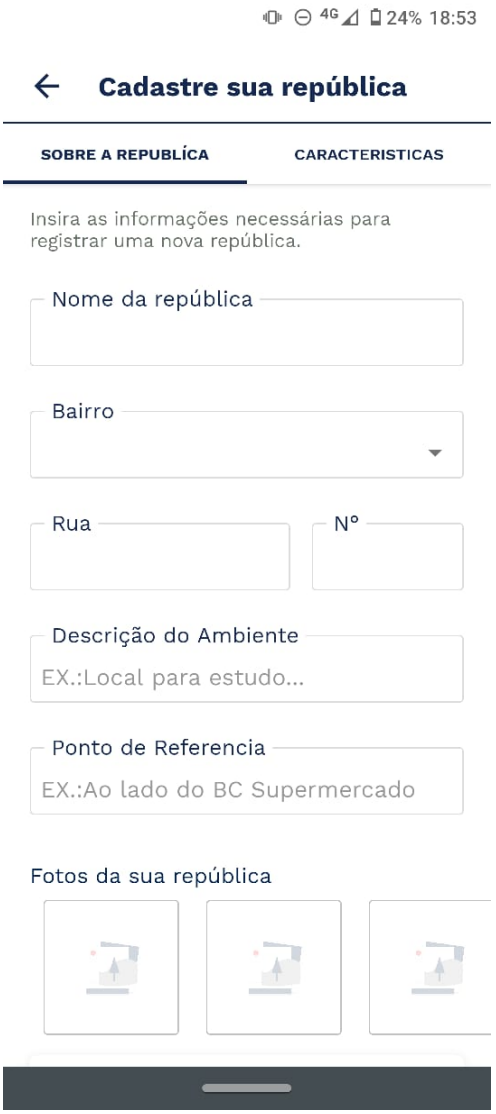
Todas as informações desse CRUD foram conectadas ao banco de dados MongoDB, através da biblioteca Mongoose. Após isso, foram criados os modelos para definir os campos que seriam necessários em cada entidade do banco de dados, como por exemplo, quais campos seriam interessantes o usuário preencher para realizar o cadastro de uma república a ser anunciada (nome, valor, descrição, quantidade de vagas, tipo de imóvel, etc).

Após criada toda a lógica dos cruds e as suas devidas relações com o banco de dados, foram criadas as rotas que são utilizadas como API's para fazer a ligação entre frontend e o backend.

Utilizando o React Native, dividimos esse módulo nos seguintes componentes:

3.1.1. Cadastro

É a tela que nos permite realizar o cadastro de uma república a fim de divulgar vagas disponíveis.


24% 18:53
Cadastre sua república
SOBRE A REPUBLÍCA
CARACTERISTICAS
Insira as informações necessárias para registrar uma nova república.
Nome da república
Bairro
Rua
N°
Descrição do Ambiente
EX.:Local para estudo...
Ponto de Referencia
EX.:Ao lado do BC Supermercado
Fotos da sua república

Após cadastrada a república, ela irá aparecer na tela a seguir.

3.1.2. República

É a tela inicial deste módulo e também do aplicativo como um todo, onde são exibidos os cards de todas as repúblicas cadastradas no sistema e suas informações mais importantes, como o nome, foto, descrição, valor e quantidade de vagas disponíveis. Clicando em um desses cards, vamos para a próxima tela: Detalhes Anúncio.

A tela República ficou da seguinte forma:

3.1.3. Detalhes Anúncio

Essa é a tela em que são exibidas todas as informações disponíveis no anúncio relacionadas ao tipo de república, localização, valores, quantidade de vagas e etc.

Lounge

Lounge

Ambiente aconchegante e completo

Representante
Testando

Tipo de imóvel
Casa

Gênero
Mista

Aceita animais?
sim

Disponibilidade
2 Vaga(s)

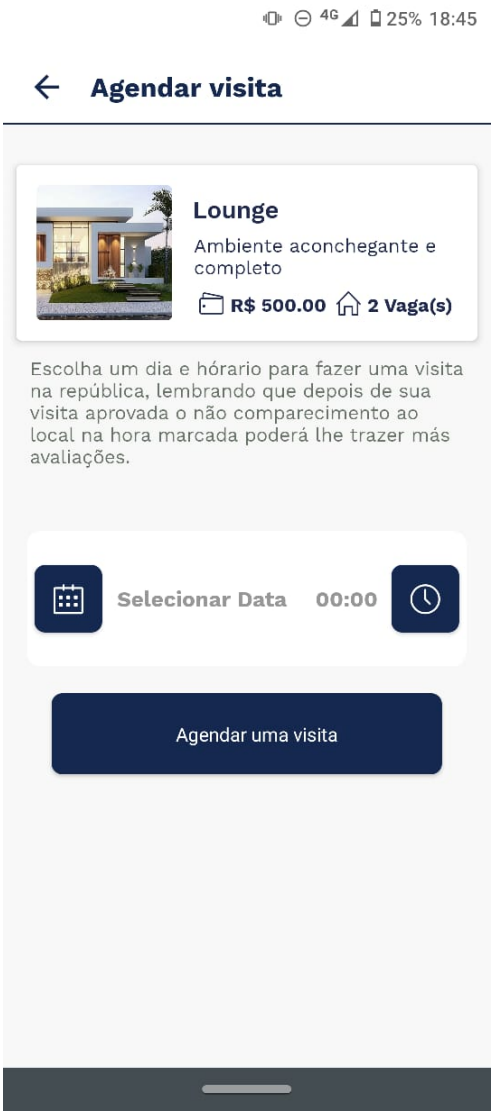
Endereço

Ponto de referencia
Próximo ao mercado

Endereço
Colinas, Centro, Nº155

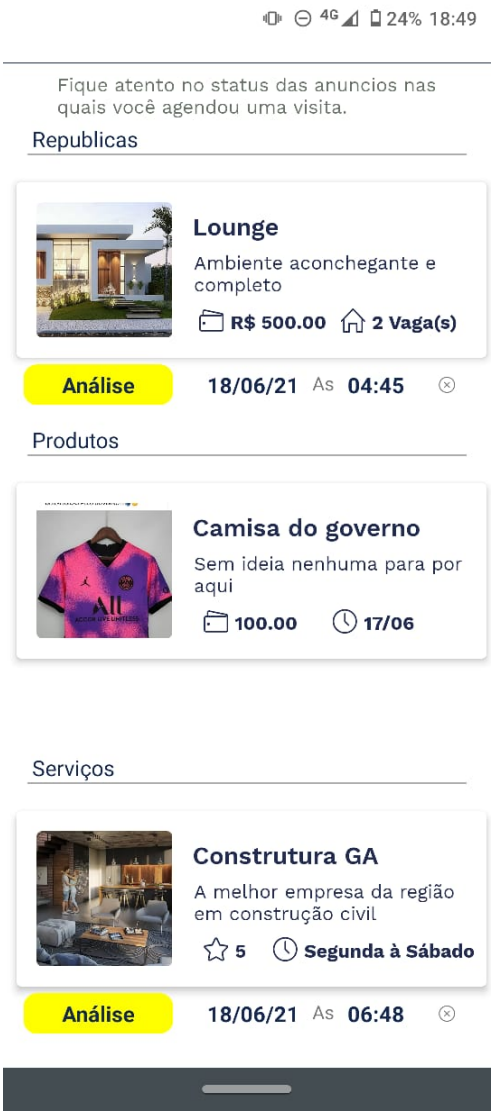
3.1.4. Agendar

É nessa tela que o usuário interessado em alguma república pode escolher um dia e horário para agendar uma visita para conhecê-la. É acessada através do botão “Agendar uma visita” no final da tela Detalhes Anúncio.



3.1.5. Agendamentos User

Após agendar visita em uma república, você poderá acompanhar o andamento através dessa tela, que no aplicativo é nomeada de “Meus agendamentos”. Nessa tela também ficam guardados os pedidos de agendamento em produtos e serviços, que são outros módulos do aplicativo.



3.1.6. Agendamentos

Nessa tela estão as informações sobre pessoas que demonstraram interesse na república que o usuário cadastrou, trazendo o seu nome, foto, dia e hora que deseja fazer uma visita. O responsável pela república pode aceitar ou recusar a visita.

O acesso a essa tela é feito através da aba Meus anúncios > Suas repúblicas > Ver interessados.

22% 19:00

Meus anúncios

Suas Republica

Lounge

Ambiente aconchegante e completo

R$ 500.00 **2 Vaga(s)**

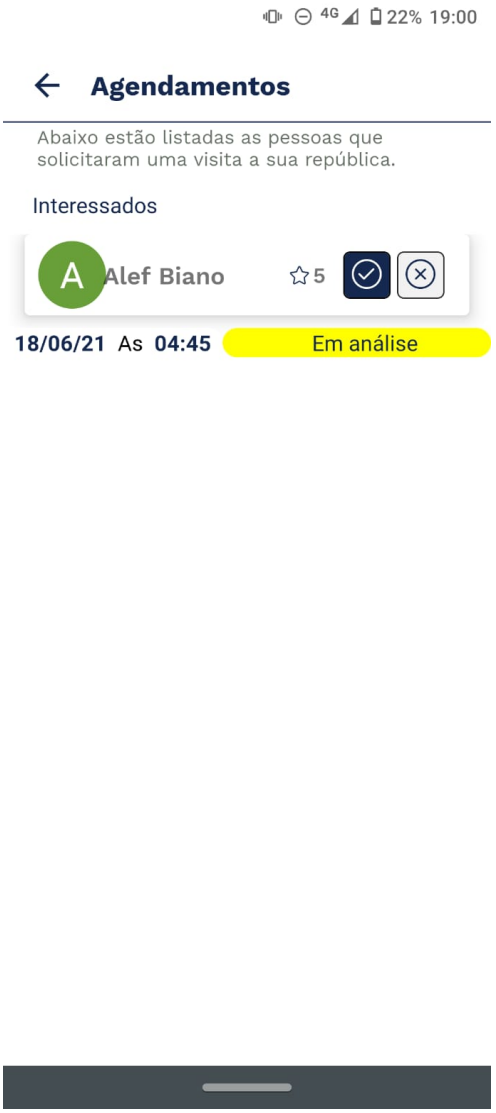
Editar | Ver interessados

Suas Caronas

Alegre X Guaçuí

Saida	Chegada	Valor	Data
12:00	**13:00**	**R$ 20.00**	**18/06**

Editar | Ver interessados


22% 19:00
Agendamentos
Abaixo estão listadas as pessoas que solicitaram uma visita a sua república.
Interessados
A
Alef Biano
5
18/06/21 As 04:45
Em análise

4. Requisitos Funcionais

dentificador	Requisito	Descrição
F01	Manter Repúblicas	Manter dados da República, atribuir um ID único e permitir a manipulação desses dados
F02	Manter Clientes (interessados em Repúblicas/vagas)	Manter informações dos usuários interessados nas Repúblicas e serviços ofertados pelo app, permitindo a manipulação das informações.
F03	Manter Produtos para troca	Manter informações relativas aos produtos das

		Repúblicas que serão ofertados para troca ou venda, bem como integrar a possíveis interessados entre usuários do app.
F04	Manter Serviços	Manter informações relativas à prestação de serviços para o contexto das Repúblicas, tais como serviços domésticos, de manutenção
F05	Manter Utilidades	Manter dados referentes a informações importantes no contexto da aplicação, tais como notícias, dicas,

		informes, etc. e permitir a sua manipulação.
F06	Controle de prestadores de serviços	Manter as informações sobre os prestadores de serviços e controles de sua atuação no app.
F07	Controle financeiro	Realizar o controle das finanças gerais da República e a possível divisão entre as partes.
F08	Controle de demandas internas	Realizar o controle de demandas pessoais internas da República e notificações aos membros.
F09	Emitir relação de serviços	Listagem dos serviços

		prestados pelo prestador e aqueles contraídos pelas Repúblicas
F10	Realizar buscas baseadas em filtros.	Implementação das diversas buscas em cada entidade presente.
F11	Efetuar a disponibilidade de vaga em República	Oferta de vagas por uma República e suas particularidades
F12	Efetuar o atendimento da vaga ao usuário	Possibilita a candidatura do cliente a uma vaga na República e o alinhamento do contato entre as partes
F13	Efetuar a disponibilidade de serviços pelo prestador	Oferta de serviços por um prestador e

		suas particularidades
F14	Efetuar o atendimento do serviço à República	Possibilita a marcação da República a um serviço de um prestador e o alinhamento do contato entre as partes
F15	Realizar a avaliação das operações realizadas	Realizar a avaliação pelas partes de cada comutação realizada pelo aplicativo, mediante regras específicas.

5. Requisitos não Funcionais

RNF01: O software deverá ser de fácil manuseio, com uma interface bem intuitiva visando uma melhor interação com usuário;

RNF02: Cada usuário terá acesso a suas respectivas funcionalidades, sendo definidos três perfis distintos: República, cliente e prestador de serviço;

RNF03: O software deverá ser alocado em um servidor WEB, de preferência gratuito visando manter um custo baixo para a instituição, e mantendo assim uma disponibilidade maior para acesso;

RNF04: os dados serão armazenados em um SGBD em nuvem, sendo parte de seus dados embarcados no dispositivo do cliente quando de seu acionamento.

RNF05: o aplicativo será desenvolvido para multiplataformas de sistemas operacionais móveis.

RNF06: o aplicativo terá vários filtros em suas pesquisas a fim de facilitar a busca de informações pelo usuário.

RNF07: o aplicativo terá localização espacial usando a API Google Maps e serviços de geolocalização.

6. Regras de Negócio

RN01: haverá 03 perfis de usuários: república, cliente e prestador;

RN02: o número de imagens a serem inseridas no cadastro de um item terá um limite máximo de 03 imagens;

RN03: na exibição da vaga anunciada por uma República, haverá uma indicação de quantas pessoas se candidataram para a mesma, a fim de informar ao usuário a situação da concorrência por ela;

RN04: quando da candidatura do cliente por uma vaga, esta poderá ser monitorada pelos status (Aberta, Em análise, Em Contato, Atendida, Fechada);

RN05: dado uma candidatura, a República terá o limite de um número de dias definido para atender ao candidato. Caso contrário a vaga fica indisponível;

RN06: tanto republicas quanto o cliente à vaga poderão avaliar o contato feito e o aplicativo fará um ranking para medição de credibilidade das partes envolvidas;

RN07: serão definidas restrições para as partes envolvidas no processo de preenchimento de vagas (Repúblicas e clientes) em caso de não cumprimento do acordo por ambas.

RN08: da mesma forma, haverá um controle de reputação, baseado em avaliação de atendimento, para a prestação de serviços, ou seja, da república para o prestador e vice-versa. Após um número mínimo de avaliações negativas, o prestador poderá sofrer sanções como a inatividade no aplicativo ou sua exclusão.

RN09: quando um serviço é prestado haverá a contabilidade pela República do tempo gasto pelo prestador, em unidades de tempo fixas e definidas pelo aplicativo (janelas de tempo);

RN10: um serviço realizado será avaliado pelo seu custo/benefício, entre os valores fixos “Baixo, Moderado e Alto”;

RN11: um serviço poderá ser monitorado através dos valores “em atendimento”, “atendido”, “não concluído”, “não atendido”;

RN12: prestador de serviço registra qual janela de tempo em um dia estará disponível e o cliente, no caso a República, define qual dia da semana e janela de tempo será o desejado para a realização do serviço, até a confirmação do prestador ou sua solicitação de reagendamento.

RN13: haverá regras de moderação para serviços não atendidos.

RN14: a busca por vagas terão os filtros: por gênero, curso ou tipo de locação.

RN15: a busca por Repúblicas será por bairro ou região.

RN16: a busca por serviços será por tipo, tendo valores fixos para estes: pintura, eletricista, bombeiro hidráulico, faxina, etc.

RN17: produtos para troca ou venda serão permitidos materiais acadêmicos também, tais como livros, apostilas, instrumentos, etc.

7. O que não foi implementado

Dentro do que foi especificado nos itens 4, 5 e 6, alguns requisitos não foram totalmente implementados, seja por falta de investimento ou por tempo. Dentre eles, estão:

RF08: Controle de demandas internas;

RF09: Emitir relação de serviços;

RF15: Realizar a avaliação das operações realizadas;

RNF03(CONFIRMAR): O software deverá ser alocado em um servidor WEB, de preferência gratuito visando manter um custo baixo para a instituição, e mantendo assim uma disponibilidade maior para acesso;

RNF04(CONFIRMAR): os dados serão armazenados em um SGBD em nuvem, sendo parte de seus dados embarcados no dispositivo do cliente quando de seu acionamento;

RNF07: o aplicativo terá localização espacial usando a API Google Maps e serviços de geolocalização;

RN03: na exibição da vaga anunciada por uma República, haverá uma indicação de quantas pessoas se candidataram para a mesma, a fim de informar ao usuário a situação da concorrência por ela;

RN06: tanto republicas quanto o cliente à vaga poderão avaliar o contato feito e o aplicativo fará um ranking para medição de credibilidade das partes envolvidas;

RN07(CONFIRMAR): serão definidas restrições para as partes

envolvidas no processo de preenchimento de vagas (Repúblicas e clientes) em caso de não cumprimento do acordo por ambas.

RN08(CONFIRMAR): da mesma forma, haverá um controle de reputação, baseado em avaliação de atendimento, para a prestação de serviços, ou seja, da república para o prestador e vice-versa. Após um número mínimo de avaliações negativas, o prestador poderá sofrer sanções como a inatividade no aplicativo ou sua exclusão.

RN09(CONFIRMAR): quando um serviço é prestado haverá a contabilidade pela República do tempo gasto pelo prestador, em unidades de tempo fixas e definidas pelo aplicativo (janelas de tempo);

RN10(CONFIRMAR): um serviço realizado será avaliado pelo seu custo/benefício, entre os valores fixos “Baixo, Moderado e Alto”;

RN11(CONFIRMAR): um serviço poderá ser monitorado através dos valores “em atendimento”, “atendido”, “não concluído”, “não atendido”;

RN13(CONFIRMAR): haverá regras de moderação para serviços não atendidos.

RN14(CONFIRMAR): a busca por vagas terão os filtros: por gênero, curso ou tipo de locação.

RN15(CONFIRMAR): a busca por Repúblicas será por bairro ou região.

RN16(CONFIRMAR): a busca por serviços será por tipo, tendo valores fixos para estes: pintura, eletricista, bombeiro hidráulico, faxina, etc.

8. O que foi implementado além do esperado

Além do que foi especificado anteriormente, durante o desenvolvimento surgiu a necessidade de ser criado um novo módulo, denominado Caronas. Esse módulo tem como objetivo permitir que os usuários publiquem caronas (informando a data, o local de saída e destino, bem como os horários previstos, pontos de embarque e desembarque, e o valor) e que outros usuários as encontrem, facilitando o contato entre prestador e pessoas interessadas.

9. Problemas enfrentados

PE01: O primeiro problema que encontramos ao desenvolver o aplicativo foi o aprendizado das linguagens que eram complexas para quem não tinha visto nada do tipo ainda.

PE02: A definição de uma estrutura para o projeto é algo que ainda persiste no desenvolvimento. A princípio, foi seguido um padrão se baseando em alguns outros projetos. Porém, com o decorrer do projeto, foi preciso realizar algumas modificações para que a arquitetura se adaptasse melhor ao nosso desenvolvimento.

PE03: Era necessário que alguém assumisse a liderança do desenvolvimento para dar umas direções sobre o que precisava ser feito e como seria feito, o que demorou um pouco para acontecer.

PE04: Manter um código limpo também é um problema que ainda persiste no desenvolvimento. Por se tratar de um primeiro aplicativo, muitas coisas dentro do código precisam ser re-fatoradas, seguindo alguns padrões. Porém, isso não é uma tarefa fácil de se executar, pois demanda muito tempo e esforço.

PE05: Montar o projeto com algumas libs ainda é um problema, pois no início foram incluídas algumas das quais o seu objetivo no projeto não é muito claro, porém a remoção da mesma se enquadra no mesmo ponto do custo de re-fatoração do **PE04**, e que precisa ser feita em algum momento.

PE06: A disponibilidade de tempo da equipe é algo que, no primeiro momento, foi ótimo e conseguimos avançar bastante com o desenvolvimento. Porém, na segunda parte o tempo ficou um pouco escasso e apertado. Mas nada que impeça a entrega.

PE07: A escolha de um servidor custo-benefício também foi um ponto difícil, pelo pouco conhecimento da equipe e pelos valores extraordinários de custo para mantê-lo.

PE08: O maior problema encontrado que resultou na não implementação de alguns requisitos previamente descritos foi a utilização de libs que são pagas, como exemplo o Google Maps, que é uma api do google necessária para fornecer algumas orientações de mapa e localidade para o usuário, mas que não foi possível manter o serviço devido ao seu custo. O mesmo acontece para o serviço de SMS que não foi implementado e E-mail que foi implementado, mas não é recomendado utilizar como esta.

10. Serviços a serem contratados

Para que o aplicativo alcance todo o seu potencial e cumpra com seus objetivos, faz-se necessária a aquisição de alguns serviços. Abaixo estão representados quais são e os custos para a utilização dos mesmos.

Serviço	Distribuidor	Valor	Referência
Places AutoComplete	Google	US$ 2.83	1000 requisições
Maps dinamicos	Google	US$ 7,00	1000 requisições
Maps Estaticos	Google	US$ 2,00	1000 requisições
Envio de SMS	Amazon	$0.00581	1 SMS
Envio de Email	Amazon	US$ 2,00	100.000 emails
Servidor sob demanda m6g.medium	Amazon	US$ 1,46	24 horas