**Semana OmniStack 11.0**

*Projeto: BE THE HERO*

# Introdução Semana

## Informações Stack JS

Vantagens JS: linguagem mais popular da web, possui a maior comunidade de programadores.

2015: mudanças que consolidou o JS.

O que é possível fazer utilizando a stack: implementar padrões de código, elimina o uso de muitas bibliotecas.

Possibilidade de desenvolver apps nativos e multiplataforma: produtividade sem abrir mão da performance.

Critérios para apostar nas tecnologias certas:

1. Entender o que o mercado está procurando.
2. Quais e quantas empresas estão utilizando.
3. Qual o tamanho da comunidade e do ecossistema ao redor dessas tecnologias.
4. Quem está por trás do desenvolvimento delas.
5. Quem são os casos de uso dessas tecnologias.
6. Quantidade de vagas disponíveis no mercado.
7. Reaproveitamento de conhecimento (linguagem, caso precise migrar para outras tecnologias).
8. O quanto é possível multiplicar suas oportunidades sem multiplicar seu esforço.
9. O quanto essas tecnologias estão alinhadas com os seus objetivos.
10. Vantagens técnicas de cada uma (ex: componentização, permormance, desenvolvimento multi plataforma, real time).

## Informações técnicas

Node: motor do chrome para conseguir interpretar JS. Criador: Rain Dow: possível de executar JS fora do navegador: motor do chrome desacoplado.

Instalar o node tendo mais controle de versão: deve utilizar algum gerenciador de pacotes (ferramenta para automatizar e facilitar a instalação de pacotes e ferramentas na máquina).

## Instalações iniciais necessárias

Homebrew: mac. Chocolatey: Windows – comando: cinst node.js-lts/choco install node.js-lts.

Gerenciador de pacotes do node: npm.

Criar um projeto node: npm init -y.

Framework: biblioteca/ferramenta que fornece um conjunto de funcionalidade que todo projeto node possui.

Instalação Express (configuração da aplicação: endpoints/parametros): npm install express.

Criar o projeto react: npx (executa um pacote externo sem instalar) npx create-react-app frontend. Rodar: npm start

Instalação nodemon: npm install nodemon -D (-D: dependência de desenvolvimento). Cria o script start: nodemon index.js. Roda o backend agora com npm start.

# Banco de Dados

O banco SQLite não precisa instalar nada, ele fica como um arquivo no projeto.

Duas abordagens:

1. Driver: instala o driver do banco e utiliza os comandos SQL puros. Ex: SELECT \* FROM users.
2. Query Builder: monta o SQL utilizando JS. Ex: table(‘users’).select(‘\*’).where(). Utilizando esta abordagem você pode utilizar qualquer banco SQL, a migração dos bancos não surte efeito no código utilizado.

Query Builder a ser utilizado: **knex**.

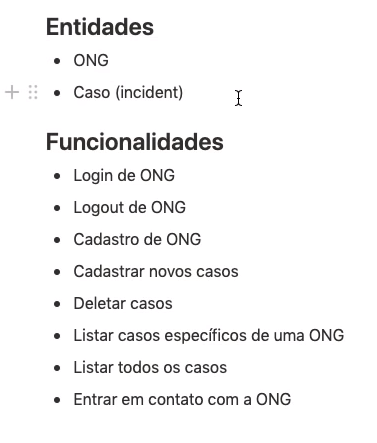
Instalação: npm install knex

Instalação do driver do banco: npm install sqlite3

Antes de configurar o banco rode o comando: npx knex init

# Projeto: Be The Hero

## Requisitos



## Banco de Dados

### Migrations

Criação de tabelas de forma que é armazenado um histórico das tabelas que foram criadas/alteradas. É um controle de versão do BD.

Criar migrate: knex migrate:make migration\_name

Rodar a migration (criar a tabela): npx knex migrate:latest

Remover a migrate (desfaz a última execução do migrate latest): npx knex migrate:rollback

Listar as migrations que foram executadas: npx knex migrate:status

## Login

Informações relacionadas a login normalmente vem através do cabeçalho da requisição (header).

Header: caracteriza o contexto da requisição. Ex: autenticação, login do usuário, informações relacionadas ao idioma (português, inglês.)

## Status HTTP:

Todos os códigos: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>

## Paginação

Paginação para não retornar todos os dados de uma vez. Retornada através do header da resposta.

//busca a paginação. Se não existir utiliza por padrão o valor 1 (buscar os dados da página 1).

        const { page = 1 } = request.query

        //conta todos os registros

        const [count] = await connection('incidents').count()

        //limit: limita a busca para 5 registros

        //offset: pula de 5 em 5 registros por página

        const incidents = await connection('incidents')

            .limit(5)

            .offset((page - 1) \* 5)

            .select('\*')

        //manda o count para o cabeçalho da resposta da requisição

        response.header('X-Total-Count', count['count(\*)'])

        return response.json(incidents)

## Join

Relacionar dados de duas tabelas.

const incidents = await connection('incidents')

            .join('ongs', 'ongs.id', '=', 'incidents.ong\_id')

            .limit(5)

            .offset((page - 1) \* 5)

            .select([

                'incidents.\*',

                'ongs.name',

                'ongs.email',

                'ongs.whatsapp',

                'ongs.city',

                'ongs.uf'

            ])

## Cors

Módulo de segurança. Determina quem poderá acessar a aplicação.

Instalação: npm install cors

Index.js

const cors = require('cors')

//em desenvolvimento: todas as aplicações podem acessar

app.use(cors())

//em produção:

app.use(cors({

    origin: 'endereço que pode acessar a aplicação. Ex: http://meuapp.com'

}))

# GitHub

Enviar os arquivos do projeto para o repositório no GitHub.

Crie o arquivo .gitignore: arquivos que não serão enviados ao repositório.

Node\_modules não é enviado, pois é pesado.. a pessoa que vai baixar apenas olha as dependências no package .json e as instala.

Digite remote e procure por: Git: Add Remote. Coloque o nome: origin, e cole a URL do repositório.

Para dar commit escreva algo na caixa de texto e clique no sinal de verde. Para dar o push .. e push.

Adicione o readme (tela inicial) para fazer a introdução do projeto com algumas informações e deixar mais atrativo, colocando as tecnologias utilizadas.

# FrontEnd

O React monta a estrutura da página (HTML, CSS..) a partir do JS e o JS só executa depois que o HTML está montado em tela. A página index.html (página que o cliente acessa) chama automaticamente o index.js (primeiro código a ser executado).

DOM: integração do react com a DOM (árvore de elementos), que renderiza (coloca em tela) o componente App dentro da div com id root.

## Componente

Função de retorna HTML. HTML em JS: JSX (JavaScript + XML).

Componentes: códigos que será reutilizados na aplicação e que possa isolar sem surtir efeito na aplicação.

Sempre é necessário importar o React quando for utilizar o JSX.

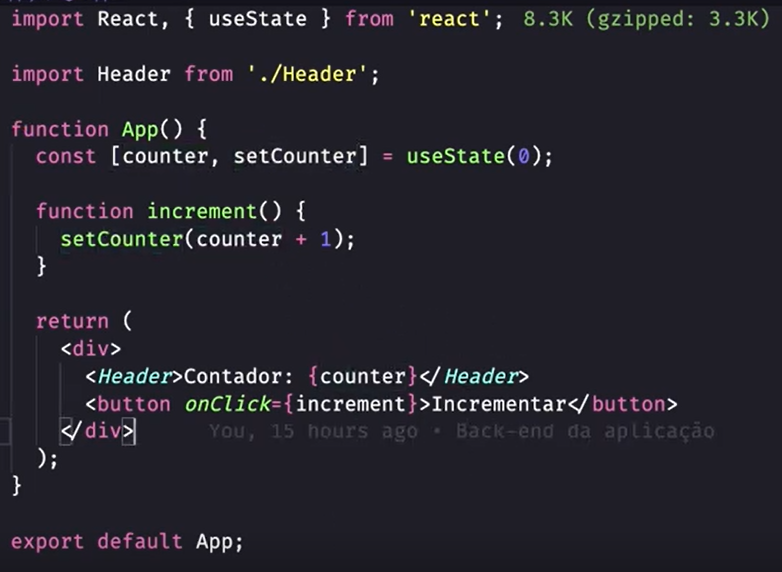
## Propriedade

Pegar o que está dentro da tag: props.children.

## Estado

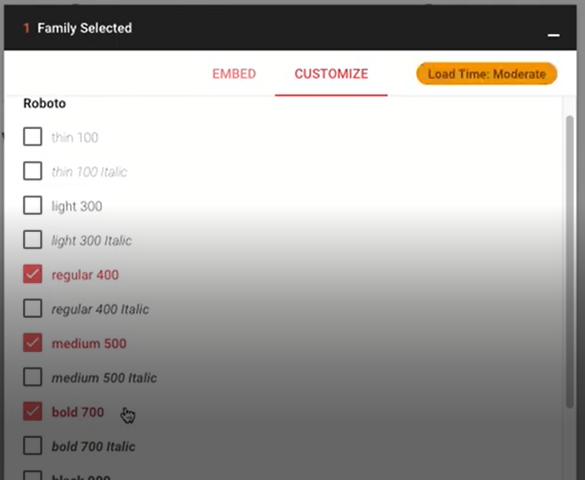
Informação que será mantida pelo componente: modificar, adicionar, listar, pegar dados de uma API.. Toda vez que o estado é alterado, o componente é renderizado novamente, por isso, os valores das variáveis de estado podem ser modificadas para visualização na interface.

Para utilizar é necessário importar o useState do React.



## Fontes

Google Fonts: <https://fonts.google.com/>

Selecionar os tipos: 

## Ícones

Material icon, fontawesome, feather icons.

Instalação: npm install react-icons

import { FiLogIn } from 'react-icons/fi'

<a *href*="/register">

                        <*FiLogIn* *size*={16} *color*={"#E02041"}/>

                        Não tenho cadastro

                    </a>

## Rotas

Instalação: npm install react-router-dom.

* Browser Router: controla todas as rotas para garantir o funcionamento.
* Route: cada uma das rotas.
* Switch: garante que apenas uma única rota execute por momento.

## Comportamento de SPA

O link não faz a página recarregar por completo, apenas muda a rota.

import { Link } from 'react-router-dom'

<*Link* *to*="/register">

                        <*FiLogIn* *size*={16} *color*={"#E02041"}/>

                        Não tenho cadastro

                    </*Link*>

## Conexão ao Backend

Cadastrar o cliente HTTP: realizar chamadas a API e obter respostas.

Npm install axios

### SPA

Tirar o recarregamento da página em submits:

function handleRegister(*e*) {

        //tira o carregamento da página - conceito de SPA

        e.preventDefault()

    }

### Chamada à API

Services/api.js

import axios from 'axios'

const api = axios.create({

    baseURL: 'http://localhost:3333'

})

export default api

index.js

import api from '../../services/api'

async function handleRegister(*e*) {

        //tira o carregamento da página - conceito de SPA

        e.preventDefault()

        const data = {

            name,

            email,

            whatsapp,

            city,

            uf

        }

        try {

            const response = await api.post('ongs', data)

            alert(`Seu ID de acesso: ${response.data.id}`)

            history.push('/')

        } catch (err) {

            alert('Erro no cadastro, tente novamente.')

        }

    }

### Chamada à API automática

Dispara a função em um determinado momento do componente:

import React, { useEffect } from 'react'

Quando é mostrado em tela:

//1° param - qual função deseja ser executada

    //2° param - quando a função será executada: [] - executa uma única vez no fluxo do componente.

    //ex: [ongName] - sempre que o nome da ong mudar irá chama

    //.then - pega os dados

    useEffect(() => {

        api.get('profile', {

            headers: {

                Authorization: ongId

            }

        }).then(*response* => {

            setIncidents(response.data)

        })

    }, [ongId])

### Navegação Histórico

import { Link, useHistory } from 'react-router-dom'

const history = useHistory()

history.push('/profile')

### Storage Navegador

Dados que serão utilizados em toda a aplicação.

Salvar os dados:

localStorage.setItem('ongId', id)

localStorage.setItem('ongName', response.data.name)

Recuperar os dados:

const ongId = localStorage.getItem('ongId')

const ongName = localStorage.getItem('ongName')

### Formatação de datas e valores

Classe global JS: intl (internacionalização)

1. param: idioma utilizado.
2. Indica o formato: moeda
3. moeda

{*Intl*.NumberFormat('pt-BR', { style: 'currency', currency: 'BRL' }).format(incident.value)}

### Logout

Apenas remove os dados do local storage e redireciona o usuário.

function handleLogout() {

        localStorage.clear()

        history.push('/')

}

# React Native

Instalação do expo na máquima: npm install -g expo-cli

Criação projeto: expo init nome\_projeto – choose blank

## Executando no celular

Rode: yarn start. Baixe o app: expo. E leia o qrcode gerado.

## Diferenças React

Não é igual HTML com as tags: div, section, header, footer. O ReactNative possui suas próprias tags (componentes): view, text..

Não possui classes de estilização, é sempre a propriedade style criada do objeto StyleSheet.

Todos componentes do React Native possui por padrão o display flex.

Sempre que desejar estilizar um componentes deve criar um estilo próprio, pois, herança de estilização não existe no React Native (estilização do pai não passa para o filho).

## Ícone e Splash Screen

Necessário apenas substituir os arquivos na pasta assets com os nomes: icon e splash. Dimensões:

* icon: 192 x 192.
* splash: 1242 x 2436.

## Rotas

### React Navigation

Instalação: npm install @react-navigation/native

Rode o comando:

* Expo: expo install react-native-gesture-handler react-native-reanimated react-native-screens react-native-safe-area-context @react-native-community/masked-view
* Sem expo: npm install react-native-reanimated react-native-gesture-handler react-native-screens react-native-safe-area-context @react-native-community/masked-vie

#### Stack Navigator

Comando: npm install @react-navigation/stack

## Diferenciação IOS e Android

Instalação pacote: expo install expo-constants

Fornece informações fixas de cada projeto.

import Constants from 'expo-constants'

* Pegar o tamanho da status bar:

paddingTop: Constants.statusBarHeight + 20

## Formatação de números

Intl: instalação: npm install intl

## Conexão WhatsApp

Linking: abre um app localizado no próprio celular através de um endereço.

function sendWhatsapp() {

        Linking.openURL(`whatsapp://send?phone=+553171590068&text=${message}`)

    }

## Conexão E-mail

import \* as MailComposer from 'expo-mail-composer'

function sendMail() {

        MailComposer.composeAsync({

            //assunto da mensagem

            subject: 'Herói do caso: Cadelinha atropelada',

            //para quem o email sera enviado

            recipients: ['carlinha.ponciano@hotmail.com'],

            //conteúdo da mensagem

            body: message

        })

    }

## Scroll infinito

<*FlatList*

*style*={styles.incidentList}

*data*={incidents}

*keyExtractor*={*incident* => *String*(incident.id)}

                //showsVerticalScrollIndicator={false}

                //função que chama quando chega no final da lista (carregar novos casos)

*onEndReached*={loadIncidents}

                //o quanto o usuário precisa estar de distância do final da lista para carregar novamente

*onEndReachedThreshold*={0.2}

*renderItem*={({ item: *incident* }) => (

                    <*View* *style*={styles.incident}>

                        <*Text* *style*={styles.incidentProperty}>ONG:</*Text*>

                        <*Text* *style*={styles.incidentValue}>{incident.name}</*Text*>

                        <*Text* *style*={styles.incidentProperty}>CASO:</*Text*>

                        <*Text* *style*={styles.incidentValue}>{incident.title}</*Text*>

                        <*Text* *style*={styles.incidentProperty}>VALOR:</*Text*>

                        <*Text* *style*={styles.incidentValue}>{

*Intl*.NumberFormat(

                                'pt-BR', {

                                    style: 'currency',

                                    currency: 'BRL'

                                }).format(incident.value)}

                        </*Text*>

                        <*TouchableOpacity*

*style*={styles.detailsButton}

*onPress*={() => navigateToDetail(incident)}

                        >

                            <*Text* *style*={styles.detailsButtonText}>Ver mais detalhes</*Text*>

                            <*Feather* *name*="arrow-right" *size*={16} *color*="#E02041" />

                        </*TouchableOpacity*>

                    </*View*>

                )}

            />

async function loadIncidents() {

        //não deixa fazer vária requisições simultâneas

        if (loading) {

            return

        }

        if (total > 0 && incidents.length == total) {

            return

        }

        setLoading(true)

        const response = await api.get('incidents', {

            params: { page }

        })

        //adiciona os novos casos ao invés de sobrescrever e excluir os já existentes

        setIncidents([...incidents, ...response.data])

        setTotal(response.headers['x-total-count'])

        setPage(page + 1)

        setLoading(false)

    }

# ERROS

Quando der erro na versão do react: coloque a mesma versão do expo e do sdk version no app.json e package.json e rode: npm install e npm start.

# Dicas

## Dicais para construção de páginas:

Primeiro crie a estrutura da página, depois coloque a estilização, por fim, criar a funcionalidade (conexão a API no node).

# Curiosidades Importantes

## Tamanho Logo

Exportar em 3 tamanhos a logo em png que será utilizada dentro do app, pois, dentro do mobile, existem densidades de pixels diferentes entre as telas. Ex: 400px em uma tela de IphoneX na realidade são 1200px, pois a densidade no IphoneX é 3 (cabe 3 vexes mais pixels do que em um celular comum).

Quando for importar não precisa colocar o @2 e @3 pois ele importa o melhor formato de acordo com a tela que está rodando a aplicação.

## Identificadores

Existem alguns formatos padrões de identificação: uuid (Universal Unique ID) / regex (expressão regular para verificar se o id é válido).

# Funcionalidades Avançadas

## Validação

Celebrate: biblioteca para fazer validações: <https://github.com/arb/celebrate>

Importante utilizar para os controladores não serem chamados e não fazer requisições no servidor sem necessidade.

Instalação: npm install celebrate

Funcionamento: integra o joi (biblioteca de validação) com o express.

Normalmente as validações ficam na pasta: src/validators.

* Exemplo validando BODY:

routes.post('/ongs', celebrate({

    [Segments.BODY]: Joi.object().keys({

        name: Joi.string().required(),

        email: Joi.string().required().email(),

        whatsapp: Joi.string().required().min(10).max(11),

        city: Joi.string().required(),

        uf: Joi.string().required().length(2)

    })

}), OngController.create)

* Name: valida que é uma string e é um campo obrigatório.
* Email: valida que é uma string, um campo obrigatório e que possui o formato de email (@ e .\_\_)
* WhatsApp: valida que é um número, um campo obrigatório e que possui no mínimo 10 e máximo 11 dígitos.
* Exemplo validando Header:

routes.get('/profile', celebrate({

    [Segments.HEADERS]: Joi.object({

        authorization: Joi.string().required()

    }).unknown()

}), ProfileController.index)

* Unknown: descarta todas as propriedades que não foram informadas para pular a validação, pois, em uma requisição HTTP é passado várias propriedades no header.
* Exemplo validando Params:

routes.delete('/incidents/:id', celebrate({

    [Segments.PARAMS]: Joi.object().keys({

        id: Joi.number().required()

    })

}),IncidentController.delete)

* Exemplo validando Query:

routes.get('/incidents', celebrate({

    [Segments.QUERY]: Joi.object().keys({

        page: Joi.number()

    })

}), IncidentController.index)

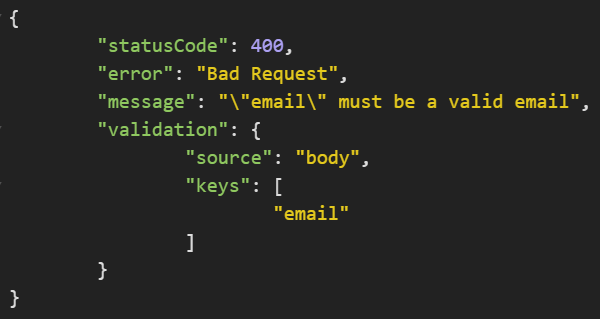
Tratamento dos erros para não retornar erro 500 (Internal Server Error).

Index.js

const { errors } = require('celebrate')

app.use(errors())

Erro retornado:



É possível configurar a mensagem a ser exibida. Através do valores informados pelo Keys, é possível utilizar no frontend para mostrar mensagens de erro nos inputs desejados.

## Testes

Em aplicações grandes e complexas (ex: 3.000 componentes), um componente é utilizado por diversos outros, assim, quando você faz uma simples modificação em um determinado componente você deve garantir com toda certeza que o restante da aplicação continua funcionando sem dar erro na produção.

### TDD (Test-Driven Development) - Desenvolvimento dirigido a Testes

Os testes funcionam como um guia no desenvolvimento. Verifica antes do desenvolvimento se as regras de negócio estão sendo consideradas no momento do desenvolvimento.

É muito importante para o funcionamento correto do sistema. Se os testes foram realizados somente após o desenvolvimento das funcionalidades a porcentagem de esquecimento e priorização de outras atividades são altas.

### Jest – Framework de Testes

Instalação: npm install jest -D

Inicialização: npx jest –init

Executar os testes: npm test

Documentação: <https://jestjs.io/docs/en/api>

### Tipos de Testes

#### Testes Unitários

Testa um pequeno pedaço da aplicação de forma isolada, funcionalidade específicas. Não atinge outras ferramentas, por exemplo, o banco de dados, outras APIs. Não possui side-effects (efeitos colaterais).

Exemplo base:

Nome do arquivo: termina com .spec.js

//define a categoria que será testada

describe('Generate Unique ID', () => {

    //it (isto): descrição do teste, colocar significado

    it('should generate an unique ID', () => {

        //definição do teste

        //expect: espera que ocorra:

        expect(2 + 2).toBe(4)

    })

})

Exemplo 2:

const generateUniqueId = require('../../src/utils/generateUniqueId')

describe('Generate Unique ID', () => {

    it('should generate an unique ID', () => {

        const id = generateUniqueId()

        expect(id).toHaveLength(8)

    })

})

#### Testes de Integração

Testa o fluxo de uma funcionalidade/rota por completo da aplicação.

##### Configuração do banco de dados

Necessário ter um banco de testes específico para rodar os testes para não interferir na integridade dos dados de desenvolvimento/produção da aplicação.

Nova configuração no knexfile.js:

test: {

    client: 'sqlite3',

    connection: {

      filename: './test.sqlite3'

    },

    migrations: {

      directory: './src/database/migrations'

    },

    useNullAsDefault: true,

  },

Adição no package.json:

Scripts:

"test": "cross-env NODE\_ENV=test jest"

Como diferenciar se está em ambiente de desenvolvimento ou de testes?

Instalação: npm install cross-env

Quando rodar o script de teste, ao invés de start, estará disponível uma variável de ambiente: NODE\_ENV. Para ter acesso as variáveis de ambiente no node:

Connection.js:

const config = process.env.NODE\_ENV === 'test' ? configuration.test : configuration.development

const connection = knex(config)

Estudo variáveis de ambiente: <https://blog.rocketseat.com.br/variaveis-ambiente-nodejs/>

##### Testes da Ong

Biblioteca utilizada para realizar as chamadas à API: supertest.Não é recomendável utilizar o axios pois ele é voltado para chamadas específicas de front end.

Supertest: faz as requisições HTTP, utiliza validações ao pegar o retorno do API.

Documentação: <https://github.com/visionmedia/supertest>

Instalação: npm install supertest -D (dependência de desenvolvimento).

const request = require('supertest')

const app = require('../../src/app')

const connection = require('../../src/database/connection')

describe('ONG', () => {

    //antes de executar cada um dos testes

    beforeEach(async () => {

        //desfaz todas as migrations (zera o banco de dados antes de começar os testes)

        await connection.migrate.rollback()

        await connection.migrate.latest()

    })

    //executar depois de todos os testes. afterEach: executa no final de cada teste.

    afterAll(async () => {

        //desfaz a conexão do teste com o banco de dados.

        await connection.destroy()

    })

    it('should be able to create a new ONG', async () => {

        //send: quais dados deseja enviar

        const response = await request(app)

            .post('/ongs')

            .send({

                name: "APAD",

                email: "contato@test.com",

                whatsapp: "31998765432",

                city: "Rio do Sul",

                uf: "SC"

            })

        expect(response.body).toHaveProperty('id')

        expect(response.body.id).toHaveLength(8)

    })

})

Para colocar header no teste: set

const response = await request(app)

            .post('/ongs')

            .set('Authorization', 'asd')

            .send({

                name: "APAD",

                email: "contato@test.com",

                whatsapp: "31998765432",

                city: "Rio do Sul",

                uf: "SC"

            })

## Deploy

### Node

#### Experimental

Hospedar para fins experimentais, teste: herokuapp.com

O heroku faz o gerenciamento de toda a estrutura da aplicação: escalabilidade, por isso, possui valor mais alto. Possui plano gratuito para aplicações experimentais.

Como hospedar: <https://www.youtube.com/watch?v=-j7vLmBMsEU>

#### Comercial

##### Pequenas

Hospedar para que clientes possam utilizar: digital ocean.

Preços: é possível saber exatamente o quanto irá gastar: 5 dólares mensais.



O primeiro funciona muito bem para a maioria das aplicações.

Como hospedar: <https://www.youtube.com/watch?v=ICIz5dE3Xfg> – configuração de HTTPS, domínio..

##### Grandes

Aplicações exponenciais com grande número de usuários.

AWS (Amazon Web Services)

Google Cloud Platform

Microsoft Azure

### Front End (ReactJS)

#### Experimental e Comercial Pequenas

Netlify: específico para frontend.

Como hospedar: criar conta no netlify, informar qual o link do repositório no github e pronto.

### React Native apk

Gerando o apk para envio (google play, app store):<https://www.youtube.com/watch?v=wYMvzbfBdYI>

OBS: IOS: preço de 99 dólares por mês para ter uma conta. Google: 25 dólares para ter uma conta.

# Recomendações de Estudo

* Padrões de código: ESLint, Prettier. (possui vídeos no youtube no canal rocketseat)
* Autenticação JWT. Trabalha com a autenticação de parâmetros para aumentar a segurança e não deixar que qualquer pessoa possa acessar. (possui vídeos no youtube no canal rocketseat)
* Styled Components. Ferramenta utilizada para auxiliar na estilização do reactjs e react native. (possui vídeos no youtube no canal rocketseat)