ESS-DELIVERY-APP

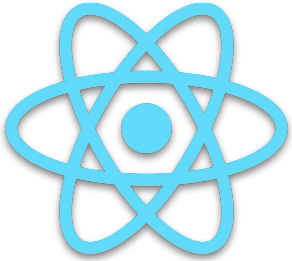
Enrique Laborão
Guilherme Morone
Lucca Gioia
Mateus Elias
Rafael Leite
Raul Coelho
Williams Santiago

Centro de Informática UFPE
Universidade Federal de Pernambuco

TECNOLOGIAS/FERRAMENTAS UTILIZADAS

SERVIDOR	INTERFACE GUI	TESTES
NodeJS	ReactJS	Jest
ExpressJS	Bootstrap	supertest
json-query	Redux	redux-saga-test-plan
	react-router-dom	enzyme
	styled-components	Cucumber/Puppeteer

PARA O PROJETO NO GERAL



CRONOGRAMA E DIVISÃO DE TAREFAS

DIVISÃO

ENRIQUE

- Roteamento das páginas
- Feature de carrinho (tela e endpoints)
- Implementação do header
- Implementação base do react-redux

GUILHERME

- Implementação da tela inicial
- Feature de avaliação de pedidos (inclui seus endpoints)
- Implementação da classe para manipulação dos arquivos JSON

LUCCA

- Feature de fazer pedidos (funcionalidade do menu)
- Endpoint de makeOrder

MATEUS

- Feature de histórico de pedidos
- Endpoint de orders (get-methods)

RAFAEL

- Feature de status do pedido
- Endpoint de restaurantes (get)

RAUL

- Feature de cancelamento de pedido
- Endpoint de cancelOrder
- Endpoint de usuário

WILLIAMS

- Feature de fazer pedidos (design do menu)
- Cálculo do tempo de entrega
- Endpoint de restaurantes (get)

Vale ressaltar as tarefas feitas em conjunto:

- Testes
- Slides
- Suporte ao time

CRONOGRAMA

SPRINT 1
(21/03 - 28/03)

- Implementação base da aplicação
- Desenvolvimento do design das telas

SPRINT 2
(28/03 - 04/04)

- Repositórios setados e organizados
- Reunião para divisão de tarefas

SPRINT 3
(04/04 - 11/04)

- Servidor e Cliente criados
- Templates feitos

SPRINT 4
(11/04 - 18/04)

- Implementação das telas

SPRINT 5
(18/04 - 25/04)

- Implementação das telas
- Refatoração de código

SPRINT 6
(25/04 - 02/05)

- Criação dos slides
- Criação do ambiente para testes
- Início dos testes
- Refatoração de código

SPRINT 7 (FINAL)
(02/04 - 04/05)

- Pequenos ajustes nos slides da apresentação final.
- Término dos testes

SERVIDOR

```
const express = require("express");
const bodyParser = require("body-parser");
const { getCart, postCart } = require("./resources/cart");
const { getUser } = require("./resources/user");
const { getRestaurants } = require("./resources/restaurant");
const {
  getOrders,
  postOrders,
  getOrderById,
  makeOrder,
  cancelOrder,
} = require("./resources/order");

const cors = require("cors");

require("dotenv").config();

const app = express();
app.use(bodyParser.json());
app.use(cors());

app.use((req, res, next) => {
  res.setHeader("Content-Type", "application/json");
  next();
});

app.get("/cart", getCart);
app.post("/cart", postCart);

app.get("/user", getUser);

app.get("/restaurants", getRestaurants);

app.get("/orders", getOrders);
app.post("/orders", postOrders);

app.get("/order-details", getOrderById);

app.post("/make-order", makeOrder);

app.post("/cancel-order", cancelOrder);
```

Definição da API e Criação dos endpoints

Centro de Informática UFPE

Universidade Federal de Pernambuco

```
const app = require("./app");

app.listen(1337, (_) => {
  console.log("Server running on port 1337");
});
```

Nos próximos slides, mostraremos com detalhes cada endpoint criado

Mas, antes de tudo, vamos mostrar a criação da classe para manipular “banco de dados” (arquivos JSON)

Testes

```
if (process.env.NODE_ENV == "test") {
  app.get("/resetTest", resetTest);
  app.post("/configTest", configTest);
}

module.exports = app;
```

CONSTRUÇÃO DA CLASSE

```
const fs = require("fs");
const jsonQuery = require("json-query");

const path = process.env.NODE_ENV === "test" ? "./test_data/" : "./data/";


exports.ManipulateDatabase = class {
  tableName;
  document;
  filePath;

  constructor(tableName) {
    this.tableName = tableName;
    this.filePath = path + this.tableName + ".json";
    this.read();
  }
```

MÉTODOS BÁSICOS

```
read(match = null) {
  this.document = JSON.parse(fs.readFileSync(this.filePath, "utf8"));

  if (match) {
    return this.query(match);
  }
}

write(data) {
  fs.writeFileSync(this.filePath, JSON.stringify(data));
  this.read();
}

append(content) {
  this.document[this.tableName].push(content);
  this.saveChanges();
}

saveChanges() {
  this.write(this.document);
}

getArray() {
  return this.document[this.tableName];
}
```

MÉTODOS ESPECIAIS

```
query(match) {
  let qStr = "";

  if (match.inner !== undefined) {
    const name = match.inner.nameObjToQuery;
    qStr = `[${match.inner.matchId}]`;

    return jsonQuery(name + qStr, {
      data: this.document,
    }).value;
  } else if (match.deep !== undefined) {
    if (match.deep.deepSearch) qStr = "[**]";
    match.deep.booleans.forEach((element) => {
      if (element.findOne) {
        qStr += "[" + element.expr + "]";
      } else {
        qStr += "[*" + element.expr + "]";
      }
    });

    return jsonQuery(this.tableName + qStr, {
      data: this.document,
    }).value;
  }
}
```

```
findAndReplace(compareFunction, newItem, append = true) {
  const index = this.document[this.tableName].findIndex(compareFunction);
  if (index !== -1) {
    if (newItem) this.document[this.tableName].splice(index, 1, newItem);
    else this.document[this.tableName].splice(index, 1);
  } else if (append) {
    this.append(newItem);
  } else {
    throw new Error("Item não encontrado no banco de dados");
  }
  this.saveChanges();
}
```

USER e RESTAURANT


```
const { ManipulateDatabase } = require("../utils/db");
const jwt_decode = require("jwt-decode");

exports.getUser = async (req, res) => {
  try {
    decoded_auth = jwt_decode(req.headers.authorization);

    const table = new ManipulateDatabase("users");

    user_data = table.query({
      inner: {
        nameObjToQuery: "users",
        matchId: `id=${decoded_auth.userId}`,
      },
    });

    res.status(200).send(JSON.stringify(user_data));
  } catch (err) {
    console.error(err);
    res.status(500).send(err);
  }
};
```

```
const { ManipulateDatabase } = require("../utils/db");
const { getRandomSlice } = require("../utils/misc");

exports.getRestaurants = async (req, res) => {
  try {
    const restaurants = new ManipulateDatabase("restaurants");
    if (req.query.id !== undefined) {
      const restId = JSON.parse(req.query.id);

      rest_data = restaurants.query({
        inner: {
          nameObjToQuery: "restaurants",
          matchId: `id=${restId}`,
        },
      });
      res.status(200).send(rest_data);
    } else {
      const arr = getRandomSlice(restaurants.getArray(), 3);
      res.status(200).send(arr);
    }
  } catch (err) {
    console.error(err);
    res.status(500).send(err);
  }
};
```

ORDER

Inicialização

```
const { ManipulateDatabase } = require("../utils/db");
const {
  dateToString,
  dateStrToInt,
  createOrder,
  dispatchOrderStatusWorker,
} = require("../utils/misc");
const jwt_decode = require("jwt-decode");

function queryOrdersByDate(days, user_id) {
  const startDate = new Date();
  const table = new ManipulateDatabase("orders");

  startDate.setDate(startDate.getDate() - days);
  const resp = table.query({
    deep: {
      deepSearch: true,
      booleans: [
        {
          findOne: false,
          expr: `date≥${dateToString(startDate)}`,
        },
        {
          findOne: false,
          expr: `user_id≥${user_id}`,
        },
      ],
    },
  });
  resp.sort((a, b) => dateStrToInt(b.date) - dateStrToInt(a.date));
  return resp;
}
```

GET Methods

```
exports.getOrders = async (req, res) => {
  try {
    decoded_auth = jwt_decode(req.headers.authorization);
    const days = req.query.dateFilter;
    const data = queryOrdersByDate(days, decoded_auth.userId);
    res.status(200).send(JSON.stringify(data));
  } catch (err) {
    console.error(err);
    res.status(500).send(err);
  }
};
```

```
exports.getOrderById = async (req, res) => {
  try {
    const { id } = req.query;

    const table = new ManipulateDatabase("orders");
    const data = table.query({
      inner: {
        nameObjToQuery: "orders",
        matchId: `id=${id}`,
      },
    });
    if (!data) throw new Error("Pedido não existe!");

    res.status(200).send(JSON.stringify(data));
  } catch (err) {
    console.error(err);
    res.status(500).send(err);
  }
};
```

ORDER - POST METHODS

```
exports.postOrders = async (req, res) => {
  try {
    decoded_auth = jwt_decode(req.headers.authorization);

    // Restaurants update
    const restaurantsTable = new ManipulateDatabase("restaurants");

    const restaurantData = restaurantsTable.query({
      inner: {
        nameObjToQuery: "restaurants",
        matchId: `id=${req.body.restaurantId}`,
      },
    });

    const newRate = {
      user_id: decoded_auth.userId,
      stars: req.body.rate.stars,
      feedback_text: req.body.rate.feedback_text,
    };

    restaurantData.rates.push(newRate);

    const restaurantCompareFunction = (item) =>
      item.id == req.body.restaurantId;
    restaurantsTable.findAndReplace(restaurantCompareFunction, restaurantData);
```

```
exports.makeOrder = async (req, res) => {
  try {
    decoded_auth = jwt_decode(req.headers.authorization);

    const ordersTable = new ManipulateDatabase("orders");
    const cartTable = new ManipulateDatabase("carts");
    const restaurantsTable = new ManipulateDatabase("restaurants");
    const userTable = new ManipulateDatabase("users");

    let cart_data = cartTable.query({
      inner: {
        nameObjToQuery: "carts",
        matchId: `user_id=${decoded_auth.userId}`,
      },
    });
    if (!cart_data) throw new Error("User has no Cart");

    const user_data = userTable.query({
      inner: {
        nameObjToQuery: "users",
        matchId: `id=${decoded_auth.userId}`,
      },
    });

    const rest_data = restaurantsTable.query({
      inner: {
        nameObjToQuery: "restaurants",
        matchId: `id=${cart_data.rest_id}`,
      },
    });
```

```
    let order = createOrder(
      cart_data,
      rest_data.addresses,
      user_data.addresses
    );

    ordersTable.append(order);
    const cartCompareFunction = (item) => item.user_id == decoded_auth.userId;
    cartTable.findAndReplace(cartCompareFunction, null);

    const orderId = order.id;

    dispatchOrderStatusWorker(orderId);

    res.status(200).send(JSON.stringify({ id: orderId }));
  } catch (err) {
    console.error(err);
    res.status(500).send(err);
  }
};
```

```
    // Orders update
    const ordersTable = new ManipulateDatabase("orders");

    const orderData = ordersTable.query({
      inner: {
        nameObjToQuery: "orders",
        matchId: `id=${req.body.orderId}`,
      },
    });

    orderData.rate = {
      did: true,
      stars: req.body.rate.stars,
      feedback_text: req.body.rate.feedback_text,
    };

    const orderCompareFunction = (item) => item.id == req.body.orderId;
    ordersTable.findAndReplace(orderCompareFunction, orderData);

    const daysFilter = req.body.daysFilter;
    const resData = queryOrdersByDate(daysFilter, decoded_auth.userId);
    res.status(200).send(JSON.stringify(resData));
  } catch (err) {
    console.error(err);
    res.status(500).send(err);
  }
};
```

```
exports.cancelOrder = async (req, res) => {
  try {
    const decoded_auth = jwt_decode(req.headers.authorization);

    const ordersTable = new ManipulateDatabase("orders");
    const item = ordersTable.query({
      inner: {
        nameObjToQuery: "orders",
        matchId: `id=${req.body.id}`,
      },
    });

    if (item.user_id != decoded_auth.userId) throw new Error("Sem autorização");

    const DELIVERING_TIME = 90 * 60 * 1000;
    if (item.status.preparing && item.timestamp + DELIVERING_TIME > new Date()) {
      //preparing and not delayed
      throw new Error(
        "O pedido só pode ser cancelado se seu preparo não tiver sido iniciado ou se houver um atraso de 30 minutos ou mais"
      );
    }

    const orderCompareFunction = (item) => item.id == req.body.id;
    ordersTable.findAndReplace(orderCompareFunction, null, false);

    res.status(200).send(JSON.stringify({ msg: "Success" }));
  } catch (err) {
    console.error(err);
    res.status(500).send(JSON.stringify({ msg: err.message }));
  }
};
```

CART - GET

```
const { ManipulateDatabase } = require("../utils/db");
const jwt_decode = require("jwt-decode");

const DELIVERY_FEE = 5;

exports.getCart = async (req, res) => {
  try {
    decoded_auth = jwt_decode(req.headers.authorization);

    const table = new ManipulateDatabase("carts");

    cart_data = table.query({
      inner: {
        nameObjToQuery: "carts",
        matchId: `user_id=${decoded_auth.userId}`,
      },
    });

    res.status(200).send(cart_data);
  } catch (err) {
    console.log(err);
    res.status(500).send(err);
  }
};

function createNewCart(user_id, rest_id, rest_name, item, amountToChange) {
  return {
    user_id: user_id,
    rest_id: rest_id,
    rest_name: rest_name,
    total: item.price,
    items: [{ ...item, quantity: amountToChange }],
  };
}
```

CART - POST

```
exports.postCart = async (req, res) => {
  try {
    const body = req.body;

    const decoded_auth = jwt_decode(req.headers.authorization);
    const table = new ManipulateDatabase("carts");

    let cart_data = table.query({
      inner: {
        nameObjToQuery: "carts",
        matchId: `user_id=${decoded_auth.userId}`,
      },
    });

    //se usuario ja tiver um carrinho, atualiza, caso contrario cria um novo carrinho
    if (cart_data) {
      if (cart_data.rest_id !== body.rest_id)
        throw new Error("Restaurant Id doesn't match cart's current id");
      //caso o item ja esteja no carrinho, atualiza sua quantidade, caso contrario adiciona o novo item
      const index = cart_data.items.findIndex(
        (item) => item.item_id === body.item.item_id
      );

      if (index !== -1) {
        cart_data.items[index].quantity += body.amountToChange;

        if (cart_data.items[index].quantity <= 0)
          cart_data.items.splice(index, 1);

        if (cart_data.items.length == 0) cart_data = undefined;
        else {
          cart_data.total =
            DELIVERY_FEE +
            cart_data?.items.reduce(
              (acc, item) => acc + item.price * item.quantity,
              0
            );
        }
      } else {
        //previne que se retire um item q nao esta no carrinho
        if (body.amountToChange < 1)
          throw new Error("Impossible to remove non-existent item");
        cart_data.items.push({ ...body.item, quantity: body.amountToChange });
      }
    } else {
      cart_data = createNewCart(
        decoded_auth.userId,
        body.rest_id,
        body.rest_name,
        body.item,
        body.amountToChange
      );
    }

    const compareFunction = (item) => item.user_id == decoded_auth.userId;
    table.findAndReplace(compareFunction, cart_data);

    res.status(200).send(cart_data);
  } catch (err) {
    console.log(err);
    res.status(500).send(err);
  }
};
```

FRONTEND (GUI)

INICIALIZAÇÃO DO REACT-APP

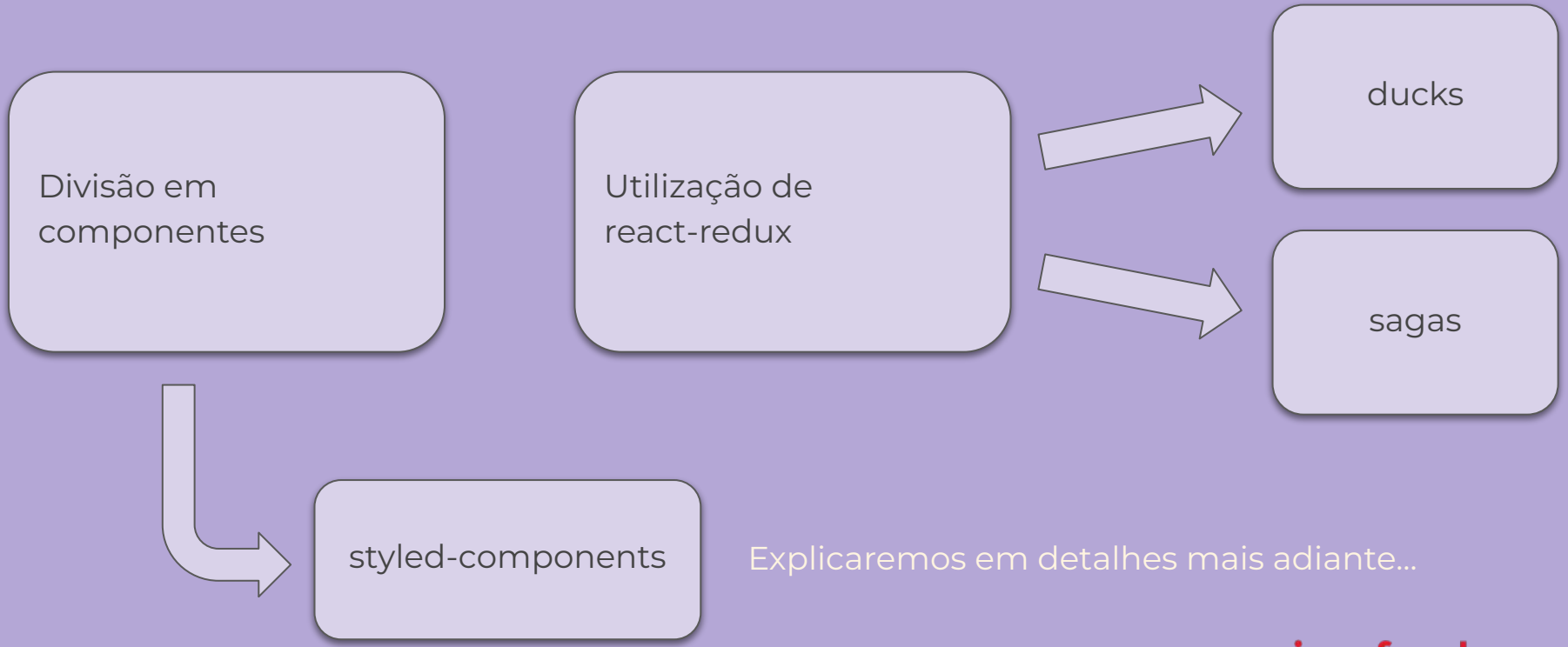
```jsx
import React from "react";
import ReactDOM from "react-dom";
import App from "./components/App/App";
import "bootstrap/dist/css/bootstrap.min.css";
import store from "./store";
import { Provider } from "react-redux";

ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>,
  document.getElementById("root")
);
```

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <link
      href="https://fonts.googleapis.com/css?family=Montserrat"
      rel="stylesheet"
    />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="description" content="Breno, por favor passa a gente :)" />
    <title>Ifood????</title>
  </head>
  <body style="background-color: #fdf1d6">
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

CONSTRUÇÃO E ESTRUTURA DA APLICAÇÃO

Uma aplicação react pode ser construída de diversas formas. Porém, optamos por esta que explicaremos a seguir:



Explicaremos em detalhes mais adiante...

COMPONENTE PRINCIPAL E ROTAS

Centro de Informática UFPE

Universidade Federal de Pernambuco

```jsx
import React, { Component } from "react";
import { StyledApp } from "./App.style";
import { BrowserRouter } from "react-router-dom";
import RouteOptions from "../routes";
import ReduxToastr from "react-redux-toastr";

import "react-redux-toastr/lib/css/react-redux-toastr.min.css";

Enrique, mês passado | 1 author (Enrique)
class App extends Component {
  render() {
    return (
      <StyledApp>
        <BrowserRouter>
          <RouteOptions />
        </BrowserRouter>
        <ReduxToastr
          timeOut={4000}
          newestOnTop={false}
          preventDuplicates
          position="top-right"
          getState={(state) => state.toastr} // This is the default
          transitionIn="fadeIn"
          transitionOut="fadeOut"
          progressBar
          closeOnToastrClick
        />
      </StyledApp>
    );
  }
}

export default App;
```

```jsx
import styled from "styled-components";

export const DEFAULT_RED = "#91091e";
export const LIGHT_RED = "#C40C28";
export const DARK_RED = "#720717";

export const DEFAULT_GREEN = "#60D394";
export const DARK_GREEN = "#34BF73";

export const DEFAULT_BLUE = "#48ACF0";
export const DARK_BLUE = "#1391E5";

export const StyledApp = styled.div`
  font-family: Montserrat;
  width: 90%;
  margin-left: auto;
  margin-right: auto;
`;
```

```jsx
import React, { Component } from "react";
import { Route, Routes, Navigate } from "react-router-dom";

import Home from "./Home";
import Header from "./Header";
import History from "./History";
import Cart from "./Cart";
import Menu from "./Menu";
import NotFound from "./NotFound";
import OrderDetails from "./OrderDetails";

Guilherme Morone Araujo, semana passada | 3 authors (Guilherme Morone Araujo and others)
class RouteOptions extends Component {
  render() {
    return (
      <>
        <Header />
        <Routes>
          <Route exact path="/home" element={<Home />} />
          <Route path="/history" element={<History />} />
          <Route path="/cart" element={<Cart />} />
          <Route path="/menu/:id" element={<Menu />} />
          <Route path="/details/:id" element={<OrderDetails />} />
          <Route path="/" element={<Navigate replace to="/home" />} />
          <Route path="*" element={<NotFound />} />
        </Routes>
      </>
    );
  }
}

export default RouteOptions;
```

Infelizmente, a aplicação é muito grande para ser mostrada inteiramente aqui.
Vamos seguir em diante diretamente pelo código.

Porém, vamos listar abaixo os componentes e features restantes.

COMPONENTES

Cart
Header
History
Home
Menu
OrderDetails
NotFound

REDUX (DUCKS/SAGAS)

Arquivos index.js
cart
history
menu
order
restaurant
user

OUTROS

utils/misc
utils/styles
services/api
constants/constants
assets/headerAssets

TESTES

BACKEND

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	82.39	50	78.57	83.76	
server	100	50	100	100	
app.js	100	50	100	100	45
server/resources	83.55	41.66	89.47	86.01	
cart.js	85.36	50	100	87.17	53,63,76-78
order.js	80.68	20	81.81	83.95	101-131
restaurant.js	84.61	100	100	84.61	22-23
user.js	100	100	100	100	
server/tests	57.14	0	33.33	57.14	
testResources.js	25	0	0	25	5-6,10-24
utils.js	100	100	100	100	
server/tests/cart	100	100	100	100	
dataUtils.js	100	100	100	100	
server/tests/orders	100	100	100	100	
dataUtils.js	100	100	100	100	
server/tests/restaurant	100	100	100	100	
dataUtils.js	100	100	100	100	
server/tests/user	100	100	100	100	
dataUtils.js	100	100	100	100	
server/utils	80	63.63	73.68	80.32	
db.js	92.1	70	100	91.66	21,48,75
misc.js	62.96	0	50	64	47-59,66,69,72

Test Suites: 8 passed, 8 total
Tests: 38 passed, 38 total
Snapshots: 0 total
Time: 4.61 s
Ran all test suites.

FRONTEND

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	75.64	38.14	64.1	75.48	
src	0	100	100	0	
index.js	0	100	100	0	8
src/assets	100	100	100	100	
headerAssets.js	100	100	100	100	
src/components	100	100	100	100	
routes.js	100	100	100	100	
src/components/App	100	100	100	100	
App.js	100	100	100	100	
App.style.js	100	100	100	100	
src/components/Cart	70.58	75	41.66	69.69	
index.js	37.5	75	36.36	33.33	40-65,109-134
styles.js	100	100	100	100	
src/components/Header	100	100	100	100	
index.js	100	100	100	100	
styles.js	100	100	100	100	
src/components/History	62.5	42.85	40	62.96	
index.js	44.73	42.85	37.5	44.44	57-108,128,142,187,208,278-36
styles.js	100	100	100	100	
src/components/Home	100	100	100	100	
index.js	100	100	100	100	
styles.js	100	100	100	100	
src/components/Menu	85.71	50	77.77	84.21	
index.js	76.92	50	75	72.72	31-32,106
styles.js	100	100	100	100	
styles.js	100	100	100	100	
src/constants	100	100	100	100	
constants.js	100	100	100	100	
src/services	100	100	100	100	
api.js	100	100	100	100	
src/store	100	100	100	100	
index.js	100	100	100	100	
src/store/ducks	100	100	100	100	
cart.js	100	100	100	100	
history.js	100	100	100	100	
index.js	0	0	0	0	
menu.js	100	100	100	100	
order.js	100	100	100	100	
restaurants.js	100	100	100	100	
user.js	100	100	100	100	
src/store/sagas	70.1	29.16	62.5	72.82	
cart.js	100	50	100	100	42-45
history.js	100	50	100	100	12-27
index.js	100	100	100	100	
menu.js	100	50	100	100	11

Agora, vamos mostrar os testes dos cenários/features diretamente pelo código.

Com isso, a apresentação termina por aqui.

OBRIGADO