Arquitetura de Referência

Versão 3.3

Histórico da Revisão

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| 01/09/2006 | 1.0 | Versão inicial do documento. | Rodrigo Barreiros |
| 29/10/2007 | 1.1 | Alteração de frameworks na camada de apresentação. Alteração da estrutura de pacotes. | Rodrigo Barreiros |
| 13/08/2009 | 1.2 | Atualizações gerais do documento. | Anacélia Cunha |
| 14/05/2012 | 1.3 | Atualizações nas camadas de apresentação e persistência. | Anacélia Cunha Rodrigo Barreiros |
| 24/04/2018 | 1.4 | Revisão da versão inicial do documento. | Lucas Caldas de Oliva Rodrigues |
| 26/04/2018 | 2.0 | Revisão aprovada pela equipe | Lucas Caldas de Oliva Rodrigues |
| 06/11/2020 | 2.5 | Atualização da tecnología da aplicação cliente | Filipe Rodrigues Barretto |
| 17/03/2021 | 3.0 | Atualização das tecnologias de desenvolvimento | Filipe Rodrigues Mateus Guilardi Mikael Medeiros |
| 29/03/2021 | 3.2 | Adicionando tecnologias de infra e banco de dados | Filipe Rodrigues Mateus Guilardi Mikael Medeiros |
| 30/04/2024 | 3.3 | Atualização das versões do java 21 e angular 17 | Leonardo Oliveira |

Índice

[1. Introdução 4](#_Toc178685525)

[2. Estratégia Arquitetural 4](#_Toc178685526)

[2.1. Arquitetura Stateless 4](#_Toc178685527)

[3. Serviços Base da arquitetura 5](#_Toc178685528)

[3.1. API-BASI 5](#_Toc178685529)

[3.2. API-Autenticacao 5](#_Toc178685530)

[3.3. API-Corporativo 5](#_Toc178685531)

[4. Frameworks da arquitetura back-end 6](#_Toc178685532)

[4.1. Java 6](#_Toc178685533)

[4.1.1. Spring-boot 6](#_Toc178685534)

[4.1.2. Spring-data-jpa 6](#_Toc178685535)

[4.1.3. Hibernate 6](#_Toc178685536)

[4.1.4. Spring-test 6](#_Toc178685537)

[4.1.5. Spring-data-rest 6](#_Toc178685538)

[4.1.6. Lombok 6](#_Toc178685539)

[4.2. Node 7](#_Toc178685540)

[4.2.1. Nest.js 7](#_Toc178685541)

[4.2.2. TypeORM 7](#_Toc178685542)

[4.2.3. Jest 7](#_Toc178685543)

[4.2.4. Jest-junit 7](#_Toc178685544)

[4.2.5. Passport 7](#_Toc178685545)

[5. Estrutura de Pacotes back-end 8](#_Toc178685546)

[5.1. Java 8](#_Toc178685547)

[5.1.1. Domain 9](#_Toc178685548)

[5.1.2. Controller 9](#_Toc178685549)

[5.1.3. Service 9](#_Toc178685550)

[5.1.4. Repository 9](#_Toc178685551)

[5.1.5. Exception 9](#_Toc178685552)

[5.1.6. util 9](#_Toc178685553)

[5.1.7. Config 9](#_Toc178685554)

[5.1.8. Test 9](#_Toc178685555)

[5.2. Node 10](#_Toc178685556)

[5.2.1. Controller 11](#_Toc178685557)

[5.2.2. Service 11](#_Toc178685558)

[5.2.3. Entity 11](#_Toc178685559)

[5.2.4. Module 11](#_Toc178685560)

[5.2.5. Middleware 11](#_Toc178685561)

[5.2.6. Pipe 11](#_Toc178685562)

[5.2.7. Guard 11](#_Toc178685563)

[5.2.8. Spec 11](#_Toc178685564)

[6. framework da arquitetura front-end 12](#_Toc178685565)

[6.1. Angular 12](#_Toc178685566)

[6.2. React 12](#_Toc178685567)

[7. Estrutura de pacotes Front-end 13](#_Toc178685568)

[7.1. Angular 13](#_Toc178685569)

[7.1.1. Component 14](#_Toc178685570)

[7.1.2. Router 14](#_Toc178685571)

[7.1.3. Module 14](#_Toc178685572)

[7.1.4. Service 14](#_Toc178685573)

[7.2. React 15](#_Toc178685574)

[7.2.1. Assets 16](#_Toc178685575)

[7.2.2. Config 16](#_Toc178685576)

[7.2.3. Pages 16](#_Toc178685577)

[7.2.4. Services 16](#_Toc178685578)

[7.2.5. Shared-components 16](#_Toc178685579)

[7.2.6. Store 16](#_Toc178685580)

[7.2.7. Styles 16](#_Toc178685581)

[8. Segurança 17](#_Toc178685582)

[8.1. Características 17](#_Toc178685583)

[8.2. Fluxos de Autorização 17](#_Toc178685584)

[9. Infra estrutura 18](#_Toc178685585)

[9.1. Container 18](#_Toc178685586)

[9.2. Orquestradores 18](#_Toc178685587)

[10. Banco de dados 18](#_Toc178685588)

[10.1. SQL Server 18](#_Toc178685589)

[10.2. MONGODB 18](#_Toc178685590)

[11. Referências 18](#_Toc178685591)

# Introdução

Uma Arquitetura de Referência contém um consistente conjunto das melhores práticas arquiteturais que devem ser adotadas, sistematicamente, em todos os projetos de uma organização.

A missão de uma arquitetura de referência é fornecer uma base a partir da qual os projetos possam iniciar seu ciclo de vida. Com ela, o arquiteto do projeto poderá concentrar seu esforço em resolver questões arquiteturais específicas de seu projeto sem se preocupar em projetar a resolução de problemas já solucionados.

Um projeto que não possui uma arquitetura de referência como base, não necessariamente vai falhar, ele vai demandar um esforço muito maior por parte da equipe de arquitetura no sentido de pesquisar, investigar e ponderar sobre as decisões arquiteturais.

# Estratégia Arquitetural

A Arquitetura de Referência define uma abordagem de desenvolvimento baseada em serviços REST. Modelo no qual disponibiliza serviços, manipulando recursos através de métodos HTTP, que poderão compor a solução de qualquer sistema, mesmo que em outra tecnologia. A intenção é maximizar a reutilização de esforço, diminuir custo e aumentar a produtividade na medida em que serviços existentes poderão ser utilizados para montar novos serviços ou para construir novos sistemas

## Arquitetura Stateless

Toda requisição deve ser autossuficiente, ou seja, as informações do usuário devem ficar no cliente e não no servidor. O servidor deverá ser capaz de responder por novas requisições sem se preocupar em buscar dados de requisições anteriores.

A principal ideia é a separação do desenvolvimento do Back-end e Front-end, possibilitando e flexibilizando a reutilização das regras de acesso aos recursos (dados) por diversos dispositivos.

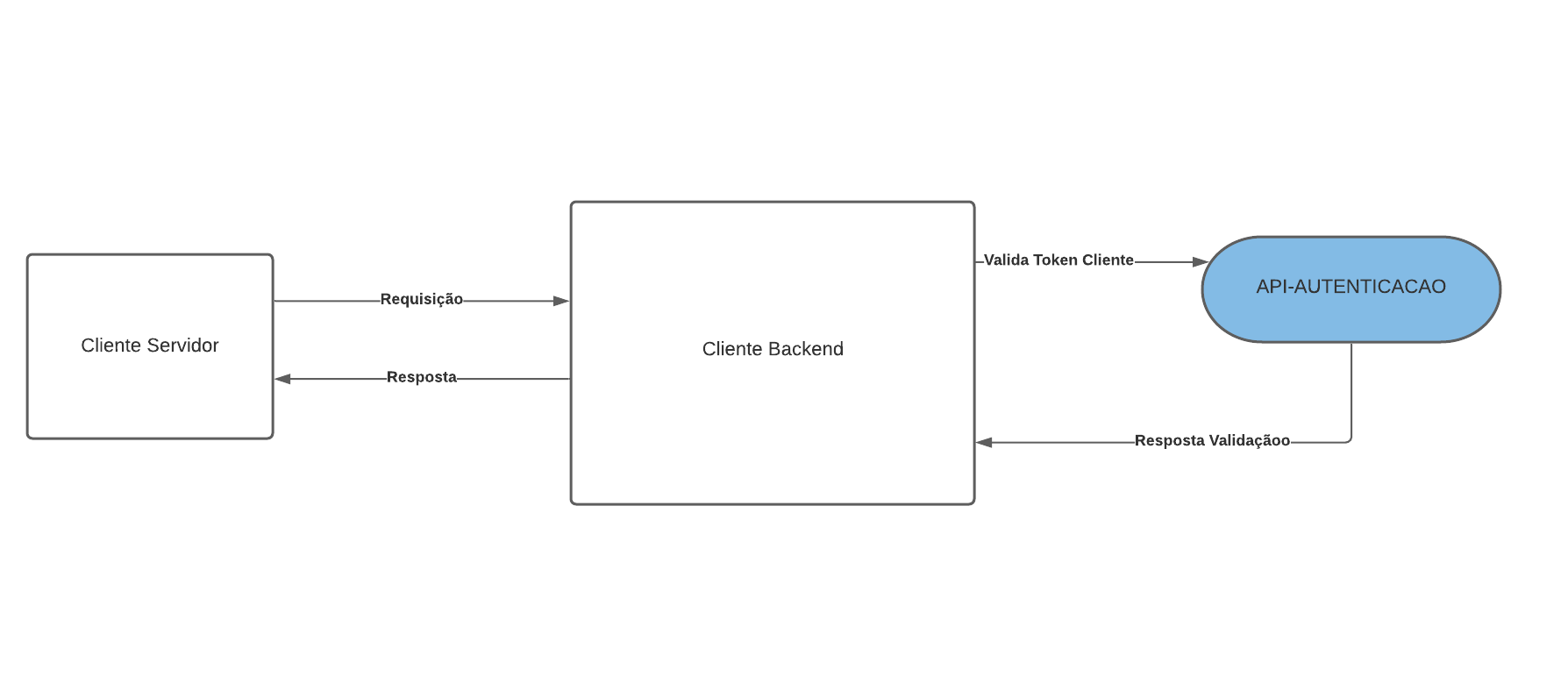


Figura 1 Requisição Stateless

# Serviços Base da arquitetura

A STI já possui alguns serviços que podem ser utilizados por qualquer sistema e que são Serviços base da arquitetura:

## API-BASI

Para acesso aos dados da base integrada;

## API-Autenticacao

Para autenticação de usuários no active directory;

## API-Corporativo

Para Buscar configuração de projetos previamente cadastrada na base integrada

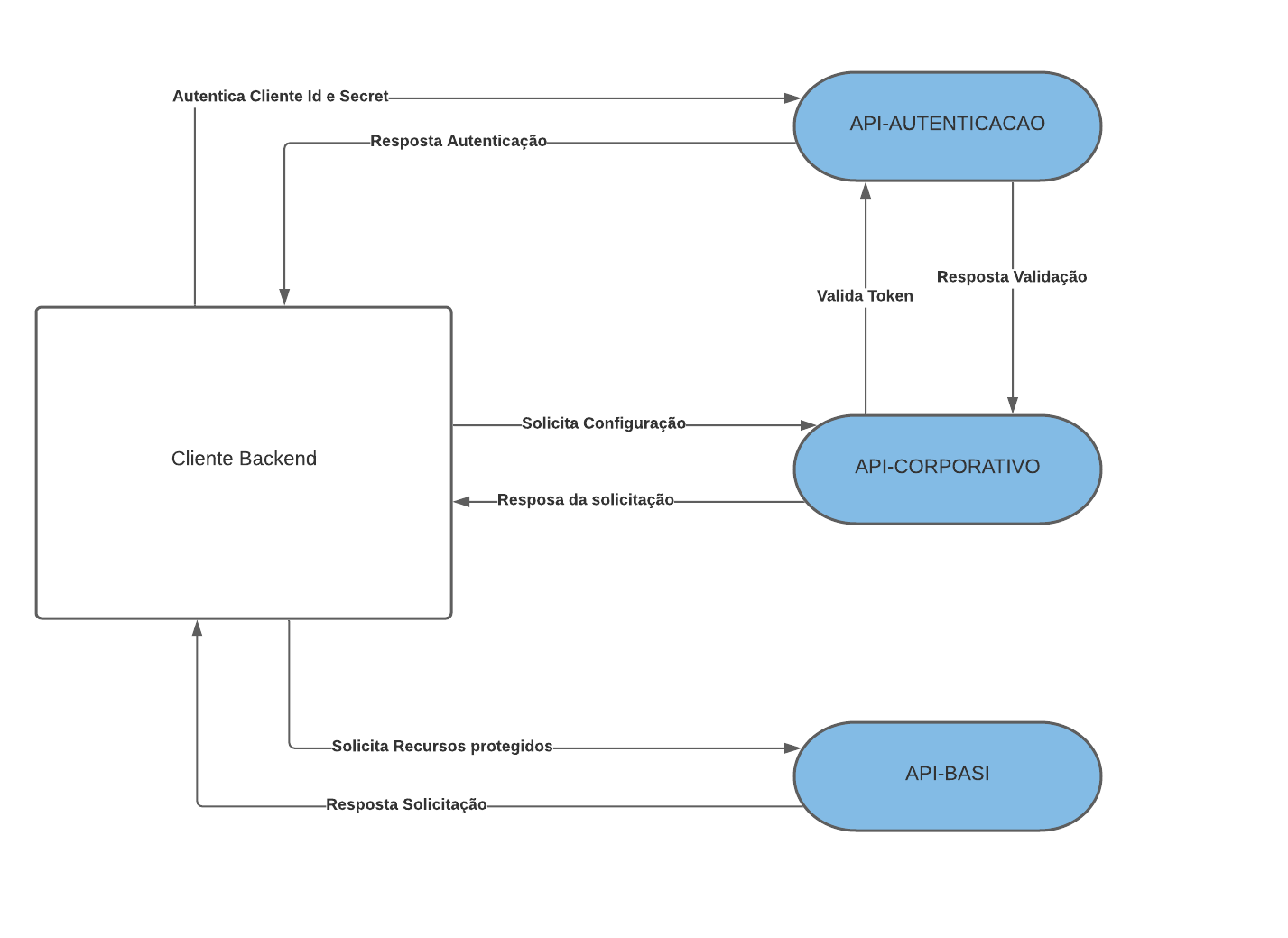


Figura 2 Comunicação dos serviços com as APIs ao iniciar a aplicação

# Frameworks da arquitetura back-end

## Java

Linguagem de programação mundialmente famosa criada em 1991 implementando os conceitos de desenvolvimento orientado a objeto, ao longo dos anos o Java sempre esteve entre as cinco linguagens mais utilizadas no mercado de desenvolvimento de software.

### Spring-boot

O Spring-boot Framework é uma plataforma Java que fornece suporte abrangente à infraestrutura para o desenvolvimento de aplicativos Java.

### Spring-data-jpa

O Spring criou o framework Spring Data JPA, cujo principal objetivo é permitir que o desenvolvimento com JPA se torne mais fácil, prático e menos repetitivo.

### Hibernate

Hibernate é uma ferramenta para mapeamento objeto/relacional para ambientes Java.

### Spring-test

Biblioteca de Teste do Spring, que proporciona diversas funcionalidades para a criação de testes dentro do framework Spring

### Spring-data-rest

Responsável por facilitar a criação dos endpoints para fazermos todas as operações que declaramos nos nossos repositórios.

### Lombok

Responsável pela automação da construção de getters e setters e equals costumeiramente escritos em classes de Domínio.

## Node

Node é um ambiente de execução Javascript no servidor implementado o motor V8 sendo orientado a eventos assíncrono, visando a melhoria de resposta e performasse de eventos de entrada e saída de dados (I/O).

### Nest.js

Framework node desenvolvido com proposta de ser escalável, modular e eficiente no desenvolvimento de aplicações server-side. Sempre implementado com as conflagrações para Express.js.

### TypeORM

É um ORM que pode ser executado nas plataformas NodeJS. Seu objetivo é fornecer recursos que ajudem a desenvolver qualquer tipo de aplicativo que use bancos de dados, desde pequenos aplicativos com algumas tabelas até aplicativos corporativos de grande escala com múltiplos bancos de dados.

### Jest

É um framework de teste Javascript mantida pelo Facebook, Inc. projetada com foco na simplicidade e suporte para grandes aplicativos da web.

### Jest-junit

Biblioteca responsável por tornar nos testes em Jest.js compatíveis com o Junit.xml – muito importante na hora de exportar o relatório dos testes para o AzureDevops

### Passport

Passport é um middleware de autenticação para Node.js. Extremamente flexível e modular, o Passport pode ser inserido de forma discreta em qualquer aplicativo da web baseado em Express. Um conjunto abrangente de estratégias de autenticação.

# Estrutura de Pacotes back-end

## Java

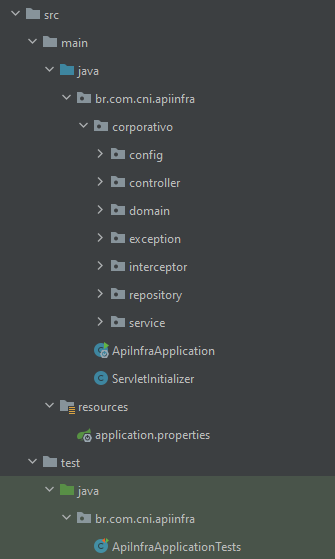


Figura 3 Estrutura projeto base Java

### Domain

São classes que definem os domínios dos Objetos negociais da solução

### Controller

Classes de controle, responsável por receber as requisições do usuário e chamar serviços para resolvê-los retornando a resposta para o usuário.

### Service

Classes responsáveis pela lógica do negócio do sistema, e nela que ficara as regras negociais da aplicação

### Repository

É onde fica a interface de persistência de dados da aplicação, quando houver necessidade

### Exception

Local onde deve ser implementado as classes que tratarão as exceções do sistema (Hadlers)

### util

Armazena as classes utilitárias que podem ser utilizadas pelos componentes.

### Config

Armazena as classes de suporte e configuração do sistema

### Test

Armazena as classes de teste do sistema, sendo que toda Service e Controller deve ter pelo menos uma classe de teste.

## Node

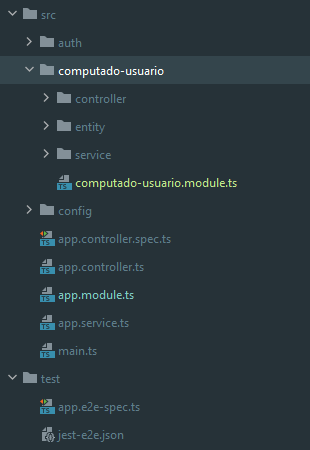


Figura 4 Estrutura API Node

### Controller

Os Controllers são classes responsáveis por lidar com as solicitações recebidas e retornar respostas ao cliente.

### Service

As Services são classes responsáveis pela lógica negocial dos módulos, ou serviços comum acesso que podem ser injetados em diversas classes do projeto.

### Entity

As entities são classes que representam o objeto do banco no sistema.

### Module

Um módulo é uma classe anotada com um decorador. O decorador fornece metadados para organizar a estrutura de aplicação.

### Middleware

Middleware é uma função que é chamada antes do manipulador de rotas. As funções do middleware têm acesso aos objetos de solicitação e resposta e à função middleware no ciclo de solicitação-resposta do aplicativo.

### Pipe

Um pipe é uma classe que intercepta o retorno de uma função podendo transformar o dado de retorno ou validá-lo antes de retorná-lo a sua execução padrão.

### Guard

Os Guards têm a responsabilidade de autorizar uma execução de um Controller, serviço e rotas dependendo de certas condições sempre presentes no tempo de execução.

### Spec

São classes de teste sempre vinculadas a um Controller e uma Service. Todo Controller e Service possui um Spec para implementação de teste.

# framework da arquitetura front-end

## Angular

É um framework JavaScript open-source, comumente utilizado para fazer aplicações SPA (single page aplication), mantido pelo Google, segue o padrão Componente da engenharia de Software. Angular traz serviços comumente designados ao lado do servidor da aplicação, como Controllers, Routers e services para o lado cliente da aplicação. Consequentemente, o peso do back-end é radicalmente reduzido, dividindo com a aplicação Front-end.

Através do Angular as aplicações clientes conseguem ser mais poderosas, gerenciáveis e escaláveis.

## React

React é uma biblioteca JavaScript de código aberto usada para construir interfaces de usuário. É usado para lidar com a camada de visualização para aplicativos da web e mobile. Também nos permite criar componentes de IU reutilizáveis. O principal objetivo do React é ser rápido, escalonável e simples.

# Estrutura de pacotes Front-end

## Angular

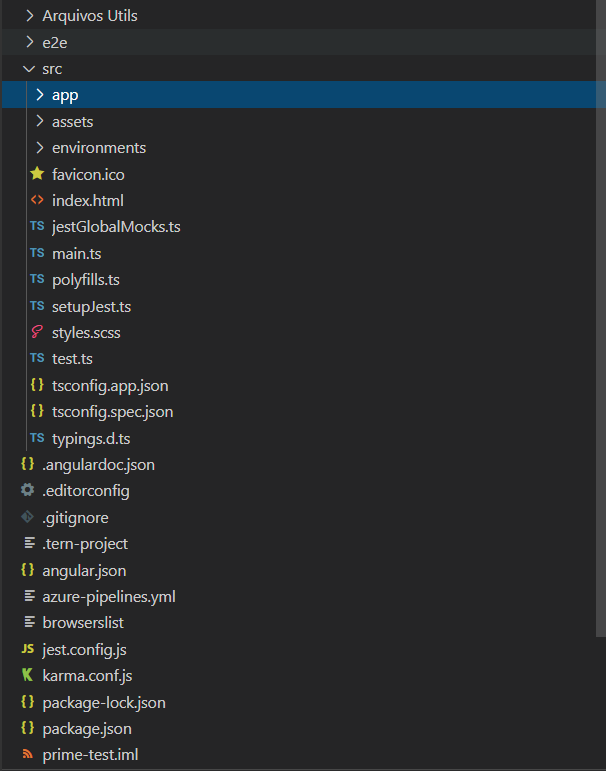


Figura 5 Estrutura de projeto Angular

### Component

Cada aplicativo Angular tem pelo menos um componente, o componente raiz que conecta uma hierarquia de componentes com o modelo de objeto de documento de página (DOM). Cada componente define uma classe que contém dados e lógica do aplicativo e está associada a um modelo HTML que define uma visualização a ser exibida em um ambiente de destino. Componentes geralmente são constituídos de:

* Html – onde fica a apresentação da aplicação
* Css – onde fica a customização dos elementos Html
* Spec – Classe de teste do componente
* Component – Logica de negócio do componente

### Router

O Angular Router fornece um serviço que permite definir um caminho de navegação entre os diferentes estados do aplicativo e visualizar hierarquias em seu aplicativo. É modelado nas convenções familiares de navegação do navegador.

### Module

Os módulos são conjuntos de Component que são dedicados a um domínio de aplicativo, um fluxo de trabalho ou um conjunto de recursos intimamente relacionado.

### Service

Uma Service Angular é um objeto sem estado e fornece algumas funções muito úteis. Essas funções podem ser chamadas de qualquer componente do Angular, como controladores, diretivas etc.

## React

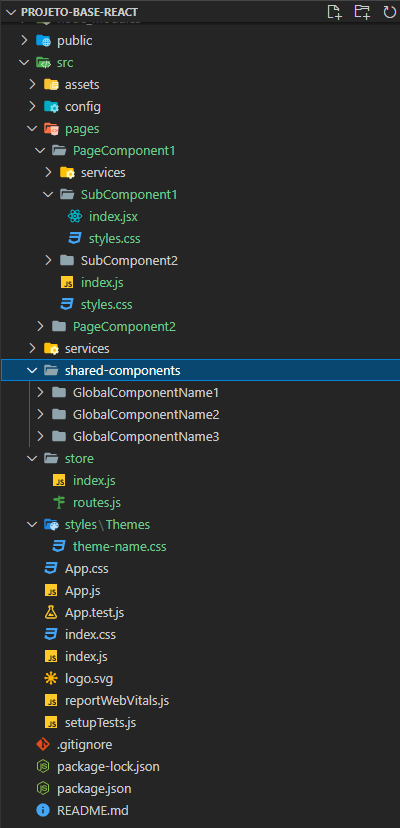


Figura 6 Estrutura de um projeto React

### Assets

Pasta onde se encontram os recursos estáticos, estruturais, do projeto, como imagens, ícones etc.

### Config

O diretório config e onde ficam os arquivos de configuração do projeto e algum plugin ou ferramenta utilizada para desenvolvimento e/ou configuração dos ambientes, caso seja necessário.

### Pages

Irá conter a relação de páginas e seus respectivos sub-componentes, juntamente com suas respectivas services.

### Services

As Services são classes responsáveis pela lógica negocial dos módulos, ou serviços comum acesso que podem ser injetados em diversas classes do projeto.

### Shared-components

Irá conter os as relações de componentes que são compartilhados na aplicação, que poderão ser reutilizados no decorrer do desenvolvimento.

### Store

Local onde irá conter os estados, compartilhados, da sua aplicação. Assim como poderá conter as rotas configuradas.

### Styles

Relação de temas e estilos da aplicação.

# Segurança

Para camada de segurança, foi definido a utilização do padrão OAuth 2, que permite uma aplicação acessar uma API no lugar do usuário, de forma segura e padronizada. Cliente acessa um recurso utilizando um Access Token. Access token é obtido com fluxos de autorização do OAuth2.

## **Características**

* Centralização e padronização
* Autenticação
* Papéis bem definidos:
* Resource Owner
* Resource Server
* Cliente
* Authorization Server
* Access Token / Refresh Token
* **HTTPS**

## Fluxos de Autorização



# INFRAESTRUTURA

## Container

Para facilitar o desenvolvimento e a entrega das apis e normalização de ambientes utilizamos Docker como nossa tecnologia para container. sempre dando preferencias para imagens em Linux e de distribuição oficial.

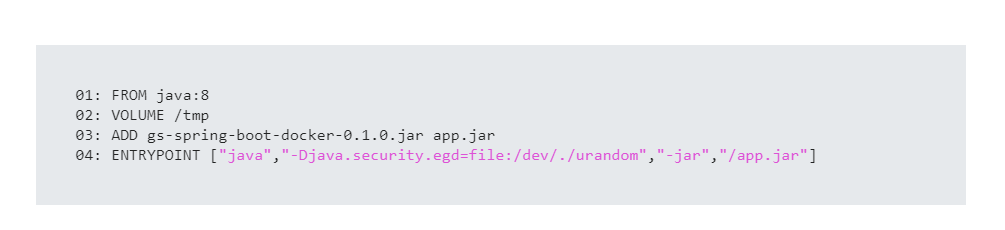


Figura 7 exemplo de dockerfile

## Orquestradores

Os orquestradores que são tecnologia que tomam conta dos contêineres e para cuidar do deploy de nosso container utilizamos de Kubernetes gerenciado através do Openshift.

# Banco de dados

## SQL Server

Banco de dados relacional desenvolvido pela Microsoft

## MONGODB

Banco de dados não relacional que trabalha através de estruturas desmoralizadas (Documentos)

# Referências

* <https://www.json.org/>
* <https://spring.io/>
* <https://angular.io/docs>
* <http://hibernate.org/orm/documentation/4.2/>
* <https://docs.nestjs.com/>