# MANUAL DE MANUTENÇÃO DE CÓDIGO

## ‹ YAMAHA PLANNING SYSTEM ›

# Sumário

# 1. Introdução

O Yamaha Planning System é um sistema de código aberto criado desenvolvido pelo Grupo Akatonbo, no primeiro semestre do Instituto de Tecnologia e Liderança – Inteli. O respositório do sistema pode ser encontrado no GitHub pelo link https://github.com/2022M2T3/Projeto4 e está estruturado no padrão de arquitetura MVC – Model, View e Controller. Esse padrão foi escolhido para otimização na velocidade das requisições feitas no sistema.

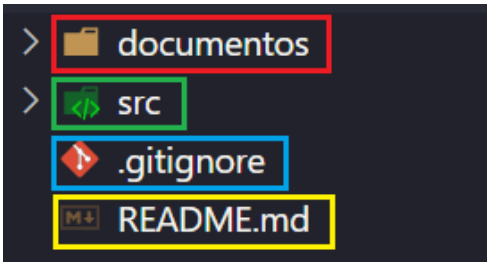
Foram utilizados NodeJS e JavaScript como linguagens de programação e Embedded JavaScript (EJS) como linguagem de modelagem para HTML. Além dessas, outras tecnologias também foram utilizadas e podem ser consultadas no Web Application Document (WAD) do sistema.

O seguinte manual é uma documentação de apoio aos comentários feitos ao longo do código, ou seja, é recomendado que, juntamente com a leitura do manual, o código seja acompanhado. Também é sugerido que os documentos WAD e Manual de Usuário sejam lidos antes de realizar a manutenção do código para que o contexto de criação do sistema seja totalmente compreendido.

O roteiro de apresentação seguirá de acordo com os caminhos das requisições do usuário feitas através do sistema, além de mostrar todas as pastas e seus referentes arquivos que contém.

Mais informações sobre os diretórios podem ser encontradas no documento Readme.md, também encontrado no GitHub. Para mais informações sobre a usabilidade do sistema, também é disponibilizado o Manual do Usuário, onde podem ser encontradas mais informações sobre a finalidade do sistema, assim como suas funcionalidades.

## 2. Estruturação de pastas e arquivos




A primeiro momento, o repositório contém duas pastas – *documentos* e *src* – e dois arquivos – *.gitgnore* e *README.md*.

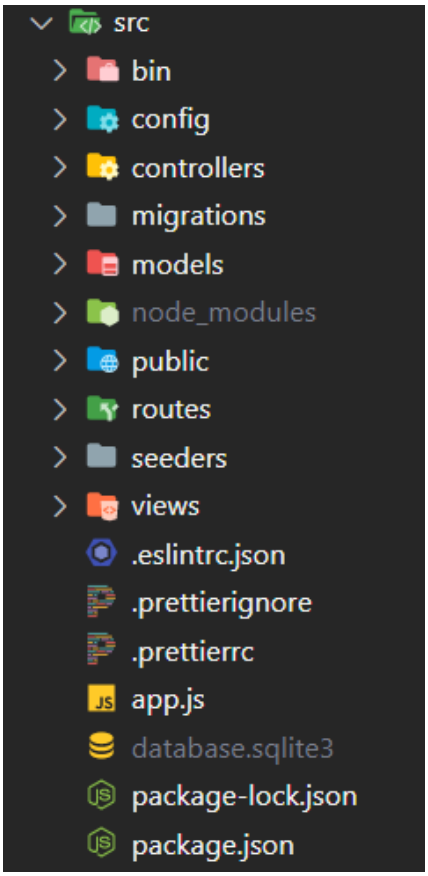
A pasta documentos contém o WAD do projeto, onde estão reunidas todas as informações de desenvolvimento do projeto do início. Além do WAD, estão localizados também o Manual do Usuário e o presente documento, Manual de Manutenção do Código.

A pasta *src* contém todos os arquivos de funcionalidade do sistema.

O arquivo *.gitgnore* é o arquivo onde é feita a identificação de todos os arquivos que não devem ser puxados a cada commit do GitHub. Para mais informações, cheque a documentação dos arquivos .gitgnore.

O arquivo *README.md* contém informações sobre os integrantes desenvolvedores do Grupo Akatonbo, descrição, estruturação de pastas, instalação, configuração para desenvolvimento, versões, exemplo de uso, licenças e referências utilizadas.

## 3. Pasta *src*




Dentro da pasta src estão localizadas as pastas do padrão MVC, nomeadas como *routes*, *controllers*, *models* e *views*, respectivamente o caminho de uma requisição. Existem também arquivos na pasta

Já o arquivo *database.sqlite3* contém o banco de dados, que está estruturado em 7 tabelas: Departments, Locations, Roles, Employees, Projects, Assignments e sqlite_sequence.

O arquivo *app.js* é a aplicação central, também localizada na pasta *src*, que roda o sistema, redireciona as requisições dos arquivos de rotas para as models e views e declara todas as tecnologias utilizadas no código.

No trecho de código seguinte, a o arquivo *app.js* verifica qual o endereço da requisição para direcioná-la

aos arquivos correspondentes da pasta *routes*, declarados em variáveis anteriormente no mesmo arquivo:

```
52   app.use('/', indexRouter);
53   app.use('/', authRouter);
54   app.use('/employees', employeesRouter);
55   app.use('/projects', projectsRouter);
56   app.use('/roles', rolesRouter);
57   app.use('/assignments', assignmentsRouter);
58   app.use('/locations', locationsRouter);
59   app.use('/departments', departmentsRouter);
60   app.use('/dashboard', dashboardRouter);
```

A partir desse primeiro redirecionamento, a rota da requisição vai sendo construída.

## 4. Caminho de uma requisição

Como já mencionado, o banco de dados está separado em sete tabelas. Nas pastas que a requisição vai percorrer, existem arquivos para cada endereço de requisição. Por exemplo, o caminho de uma requisição que utiliza dados da entidade *roles* irá acessar os arquivos */routes/roles.js*, */controllers/roleController.js* e */models/role.js*.


Dentro de *roles.js* da pasta *routes*, a requisição percorre o arquivo até o endereço da requisição recebido através da URL corresponder com a linha de código, o que é possível com a utilização do Express, uma das bibliotecas do NodeJS.

Como existe mais de um tipo de requisição para cada método na tabela *roles*, foram necessários diferentes endereços para as requisições. O código abaixo

exemplifica isso bem, pois existem dois métodos GET diferentes para a tabela *Roles*, por isso estão em rotas diferentes para evitar conflitos de informação.

```
12  router
13    .route('/')
14    .get(roleController.getAllRoles) // GET /roles
15    .post(roleController.createRole); // POST /roles
16
17  router
18    .route('/new')
19    .get(roleController.newRole); // GET /roles/new
20
21  router
22    // Métodos que selecionam o role pela Pk id
23    .route('/:id')
24    .patch(roleController.updateRole) // PATCH /roles/:id
25    .delete(roleController.deleteRole); // DELETE /roles/:id
```

Dentro da pasta *routes* existe o arquivo *index.js*, que recebe o método GET para carregar a página inicial, sendo a rota padrão.

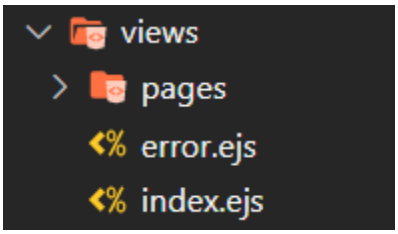
Após ser direcionada para o endpoint correto dentro do arquivo *roleController.js* com base no endereço fornecido pela URL, o método é executado e entra em contato com o banco de dados. Logo no início de cada arquivo da pasta *controllers*, os arquivos da pasta *models* são importados para receber os dados solicitados pela requisição. Abaixo, podemos ver o exemplo da entidade Roles:

```
// Importa o index.js do Model gerado automaticamente pelo Sequelize
const Role = require('../models').Role;
```

Nos arquivos da pasta *models*, é utilizada a biblioteca Sequelize, que manipula o banco de dados e dispensa o uso da linguagem Structured Query Language (SQL).

Ainda no arquivo *roleController.js*, foram utilizadas funções assíncronas para alteração do banco de dados de acordo com a requisição para que sua resposta chege antes de executar as funções seguintes, que são, principalmente, as renderizações das views.

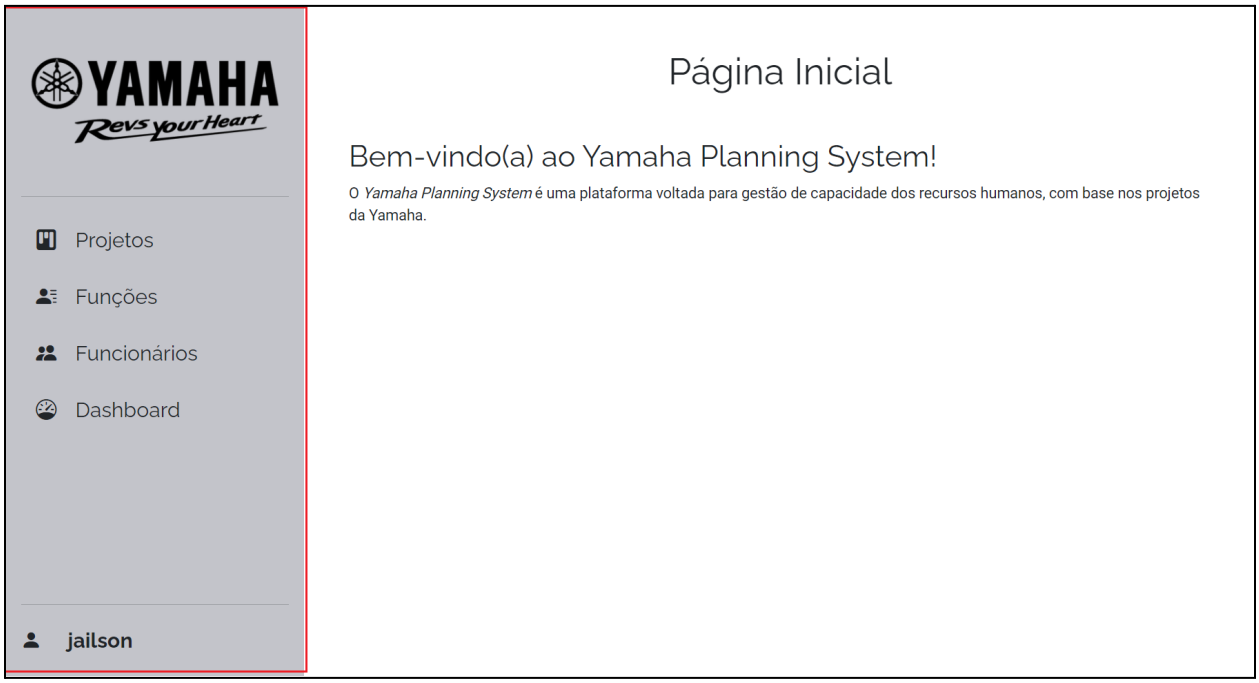
## 5. Views

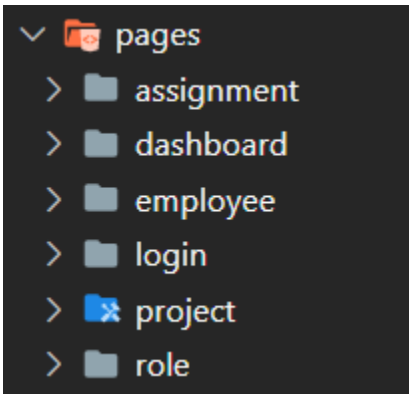



Na pasta *views*, estão localizados os arquivos do frontend do projeto. Foram utilizados arquivos do tipo Embedded JavaScript para que o JavaScript necessário de cada página seja embutido no código HTML para melhor organização.

Dois arquivos estão localizados na raíz da pasta *views*, *error.ejs* e *index.ejs*. O *index.ejs* é o arquivo de view padrão, a página inicial do sistema. Já o arquivo *error.js* é chamado quando algum erro impede que a aplicação seja carregada e retorna o código correspondente ao impedimento

Todos os arquivos contém uma barra de navegação vertical, localizada na esquerda de cada tela. Ela está presente em todas as telas e seu código se repete a cada arquivo *.ejs* das views.




A pasta *pages* está dividida em subpastas correspondentes aos atalhos da barra de navegação – Projetos, Funções, Funcionários e Dashboard correspondem às pastas *project*, *role*, *employee* e *dashboard*, respectivamente. As duas pastas restantes, *assignment* e *login*, correspondem, respectivamente, ao formulário de novas alocações de funcionários em um projeto e à página de login de abertura do sistema.

Na pasta *dashboard*, são efetuados os cálculos para inserção dos gráficos feitos utilizando ChartJS, uma biblioteca que facilita a implementação. Todos os cálculos são explicados pelos comentários ao longo dos arquivos *.ejs*.

# 6. Seeders

A pasta *seeders*, localizadas em *src*, contém arquivos *.js* que servem como um modelo para popular o banco de dados. As informações foram previamente disponibilizadas pela Yamaha e estão organizadas em formatos de objetos *json*. Para a execução dos arquivos, é necessária a instalação da biblioteca Sequelize na máquina e executar o seguinte comando:

```
npx sequelize-cli db:seed:all
```

Para mais informações, consultar as referências do arquivo WAD disponibilizado no repositório GitHub do sistema.