Projeto SYSAGUA PIES1 - Estrutura, tecnologias e APIs

Para a implementação deste projeto vamos precisar usar as seguintes tecnologias:

Linguagens e Frameworks

- Java 21 (LTS)
- Spring Framework
- JavaFX
- Postgres SQL
- Docker

JavaSpring com gradle: Dependências e estrutura

Dependências para desenvolvimento e funcionalidade principal:

- **JPA (Java Persistence API)** Para mapear classes Java para tabelas no banco de dados e facilitar operações.
- **Spring WebMVC** Para criar uma interface de API interativa (UI Swagger) que documenta e permite testar os endpoints
- **JSON Web Tokens (JWT)** Para gerar, assinar, decodificar e validar tokens JWT em aplicações Spring.

- **spring boot starter security** Para proteger rotas, implementar login, gerenciar permissões de usuário.
- **Bean Validation** Para validar automaticamente os dados enviados em requisições (e.g., nos DTOs).
- **Spring MVC** Para criar APIs RESTful ou aplicações Web.
- **Flyway** Para executar migrações em um banco PostgreSQL.

Dependências auxiliares e de conveniência:

- **Lombok** Para simplificar a escrita de classes com menos código manual.
- **spring boot devtools** Para aumentar a produtividade no desenvolvimento local.
- **Driver JDBC** para conectar aplicações Java ao banco de dados **PostgreSQL**

Dependências para testes:

- **spring boot starter test** Para criar e executar testes no projeto.
- **spring security test** Para criar testes que validam a segurança da aplicação.
- **JUnit Platform Launcher** Para executar os testes durante a fase de runtime, necessário quando múltiplos frameworks de teste estão sendo usados.

Estrutura de diretório JavaSpring

JavaSpring

```
project-root/
├── src/
│   ├── config/
│   ├── controllers/
│   ├── middleware/
│   ├── models/
│   ├── services/
```

```
│   ├── app.java
│   └── server.java
├── tests/
├── .env
├── .gitignore
└── buid.gradle
```

JavaFx com Maven: Dependências e estrutura

Ferramentas Necessárias

Frontend (JavaFX)

- Scene Builder (para interface gráfica)
- MaterialFX

Estrutura Inicial do Projeto

```
pedido-agua-frontend/
├── src/
│   ├── controllers/   # Controladores JavaFX
│   ├── models/        # Modelos para ligação com o backend
│   ├── services/      # Chamadas à API REST
│   ├── views/         # Arquivos FXML para telas
│   ├── App.java       # Classe principal
│   ├── styles.css     # Estilos
```

Response status code

- 200 OK
  - Indica que a API REST executou com êxito qualquer ação solicitada pelo cliente
  - Ao contrário do código de status 204, uma 200 deve incluir um corpo de resposta
- 201 CREATED
  - Indica que a requisição foi bem sucedida e que um novo recurso foi criado
- 204 No content
  - O código de status 204 geralmente é enviado em resposta a uma solicitação PUT ou DELETE quando a API se recusa a retornar qualquer corpo de mensagem no response
  - A resposta 204 NÃO DEVE incluir um corpo de mensagem
- 400 Bad Request
  - Indica que o servidor não pode ou não irá processar a requisição devido a alguma coisa que foi entendida como um erro do cliente
- 401 Unauthorized
  - Indica que a solicitação não foi aplicada porque não possui credenciais de autenticação válidas para o recurso de destino
- 404 Not Found
  - Indica que o servidor não conseguiu encontrar o recurso solicitado

---

Seção 01 - Implementar o banco de dados da aplicação

▼ **Atv001 - Criar a tabela de usuários**

O objetivo deste requisito é criar a tabela de usuários no banco de dados utilizando o Hibernate ORM. A tabela deve conter as colunas a seguir:

- **id**: Coluna do tipo LONG que representa a chave primária da tabela. Seu valor deve ser incrementado automaticamente pelo banco de dados.
- **nome**: Coluna do tipo STRING e de preenchimento obrigatório que armazena o primeiro nome do usuário.
- **sobrenome**: Coluna do tipo STRING e de preenchimento obrigatório que armazena o sobrenome do usuário.
- **email**: Coluna do tipo STRING e de preenchimento obrigatório que armazena o endereço de email do usuário.
- **senha**: Coluna do tipo STRING e de preenchimento obrigatório que armazena a senha do usuário. O valor a ser armazenado deve ser o hash da senha gerado pelo pacote bcrypt.
- **cargo:** Coluna do tipo STRING e de preenchimento obrigatório que armazena o cargo do usuário.
- **telefone:** Coluna do tipo STRING e de preenchimento opcional que armazena o telefone do usuário com DDD e nove dígitos.
- **status:** Coluna do tipo STRING e de preenchimento obrigatório que armazena o status do usuário sendo ativo ou inativo.

> Use a configuração `timestamps: true` para gerar as colunas **created_at** e **updated_at**

▼ Atv002 - Criar a tabela de clientes

O objetivo deste requisito é criar a tabela de clientes no banco de dados utilizando o Hibernate ORM. A tabela deve conter as colunas a seguir:

- **id**: Coluna do tipo LONG que representa a chave primária da tabela. Seu valor deve ser incrementado automaticamente pelo banco de dados
- **nome**: Coluna do tipo STRING e de preenchimento obrigatório que armazena o primeiro nome do usuário

- **sobrenome**: Coluna do tipo STRING e de preenchimento obrigatório que armazena o sobrenome do usuário.
- **endereco**: Coluna do tipo STRING e de preenchimento obrigatório que armazena o endereço da rua e número do usuário.
- **bairro**: Coluna do tipo STRING e de preenchimento obrigatório que armazena o bairro do usuário.
- **cidade**: Coluna do tipo STRING e de preenchimento obrigatório que armazena a cidade do usuário.
- **telefone:** Coluna do tipo STRING e de preenchimento opcional que armazena o telefone do usuário com DDD e nove dígitos.

> Use a configuração `timestamps: true` para gerar as colunas **created_at** e **updated_at**

Seção 02 - Implementar endpoints para o CRUD de usuarios

▼ **Atv001 - Criar endpoint para obter informações do usuário pelo ID**

- GET /v1/users/:id

Response body

```
{
  "id": 1,
  "nome": "user firstname",
  "sobrenome": "user surname",
  "email": "user@mail.com"
}
```

Response Status Code

- 200 OK - Deve ser retornado quando a requisição foi bem sucedida.
- 404 Not Found - Deve ser retornado quando o recurso solicitado não existe

▼ **Atv002 - Criar endpoint de cadastro de usuário**

- POST /v1/users

Headers

-
  - Content-type: application/json

Payload

```
{
  "nome": "user firstname",
  "sobrenome": "user surname",
  "email": "user@mail.com",
  "senha": "123@123",
  "confirmarSenha": "123@123",
  "cargo": "proprietario",
  "telefone": "85977889900"
}
```

Response Status Code

- 201 Created - Deve ser retornado quando o cadastro for bem sucedido.
- 400 Bad Request - Deve ser retornado quando a os dados da requisição estiverem incorreto.

▼ **Atv003 - Criar endpoint atualizar usuário**

- PUT /v1/users/:id

Headers

- Content-type: application/json

Payload

```
{
  "nome": "user firstname",
  "sobrenome": "user surname",
  "email": "user@mail.com",
  "telefone": "user@mail.com"
}
```

Response Status Code

- 204 No Content - Deve ser retornado quando a requisição foi bem sucedida mas nenhum corpo deve ser retornado.
- 400 Bad Request - Deve ser retornado quando a os dados da requisição estiverem incorretos
- 401 Unauthorized - Deve ser retornado quando o token de autorização não for enviado ou estiver incorreto
- 404 Not Found - Deve ser retornado quando o recurso solicitado não existe

▼ **Atv004 - Criar endpoint de deletar usuário**

- DELETE /v1/users/:id

Headers

- Content-type: application/json

Response Status Code

- 204 No Content - Deve ser retornado quando a requisição foi bem sucedida mas nenhum corpo deve ser retornado.
- 401 Unauthorized - Deve ser retornado quando o token de autorização não for enviado ou estiver incorreto
- 404 Not Found - Deve ser retornado quando o recurso solicitado não existe

Seção 03 - Implementar endpoints para o CRUD de clientes

▼ **Atv001 - Criar endpoint para obter informações do cliente pelo ID**

- GET /v1/clients/:id

Response body

```
{
  "id": 1,
  "nome": "client firstname",
  "sobrenome": "client  surname",
  "endereco": "rua abc 15",
  "bairro": "bairro1",
  "cidade": "cidade1",
  "telefone": "88922334455"
}
```

Response Status Code

- 200 OK - Deve ser retornado quando a requisição foi bem sucedida.
- 404 Not Found - Deve ser retornado quando o recurso solicitado não existe

▼ **Atv002 - Criar endpoint de cadastro de cliente**

- POST /v1/clients

Headers

-
  - Content-type: application/json

Payload

```
{
  "nome": "client firstname",
  "sobrenome": "client  surname",
  "endereco": "rua abc 15",
  "bairro": "bairro1",
  "cidade": "cidade1",
  "telefone": "88922334455"
}
```

Response Status Code

- 201 Created - Deve ser retornado quando o cadastro for bem sucedido.
- 400 Bad Request - Deve ser retornado quando a os dados da requisição estiverem incorreto.

▼ **Atv003 - Criar endpoint atualizar cliente**

- PUT /v1/clients/:id

Headers

- Content-type: application/json

Payload

```
{
  "nome": "clientAlter firstname",
  "sobrenome": "clientAlter  surname",
  "endereco": "ruaAlter abc 15",
  "bairro": "bairro1Alter",
  "cidade": "cidade1Alter",
  "telefone": "88922334455"
}
```

Response Status Code

- 204 No Content - Deve ser retornado quando a requisição foi bem sucedida mas nenhum corpo deve ser retornado.
- 400 Bad Request - Deve ser retornado quando a os dados da requisição estiverem incorretos
- 401 Unauthorized - Deve ser retornado quando o token de autorização não for enviado ou estiver incorreto
- 404 Not Found - Deve ser retornado quando o recurso solicitado não existe

▼ **Atv004 - Criar endpoint de deletar cliente**

- DELETE /v1/clients/:id

Headers

- Content-type: application/json

Response Status Code

- 204 No Content - Deve ser retornado quando a requisição foi bem sucedida mas nenhum corpo deve ser retornado.

- 401 Unauthorized - Deve ser retornado quando o token de autorização não for enviado ou estiver incorreto
- 404 Not Found - Deve ser retornado quando o recurso solicitado não existe

Seção 04 - Implementar endpoints para o CRUD de produtos

▼ **Atv001 - Criar endpoint para obter uma lista de produtos**

- GET /v1/product/search

Query params

- `fields=name,images,price`
  - Query string para limitar quais campos serão retornados
- `match=agua20L`
  - Query string usada para filtrar o resultado de produtos por um termo que combine com o nome ou descrição do produto
- `marca_ids=15,25`
  - Query string usada para filtrar o resultado de produtos pelo ID das marcas
- `linha_ids=1,2`
  - Query string usada para filtrar o resultado de produtos pelo ID das linha de produtos

Response body

```
{
  "data": [
    {
      "id": 1,
      "ativo": true,
      "nome": "Produto 01",
      "saldoInicial": 10,
      "descrição": "Descrição do produto 01",
      "preco": 119.90,
```

```
      "marca_ids": [1],
      "linha_ids": [1],
      "images": [
        {
          "id": 1,
          "content": "https://store.com/media/product-0
        }
      ]
    }
  ]
}
```

Response Status Code

- 200 OK - Deve ser retornado quando a requisição foi bem sucedida.
- 400 Bad Request - Deve ser retornado quando a os dados da requisição estiverem incorretos