

Teste de desenvolvimento Gomining

Análise do problema e tecnologias utilizadas

Seguindo a proposta estabelecida o objetivo da aplicação é a simulação de uma escola. Utilizei como tecnologias base:

- Java 11 - Utilizei o Java 11 pois era um requisito do teste.
- Maven - Utilizei o maven para o gerenciamento de dependências.
- Spring Boot - O framework Spring com suas respectivas dependências.
- MongoDB - Banco de dados não relacional em documentos
- JWT - Biblioteca para com funções para manipulação e criação do JWT
- Lombok - Criação de Getter, Setter, Construtores entre outros
- Modelmapper - Possibilitou o mapeamento dos objetos DTO, tanto de request como response, entre as camadas e para a entity.
- OpenApi - O openApi juntamente com o swagger quando configurados geram a documentação da API
- Spring-Boot-tester - O spring boot tester nos dá acesso também ao junit possibilitando a realização de testes.

Utilização

Para rodar a aplicação é necessário possuir o Java 11 e o maven instalados, além da necessidade de instalar o banco de dados MongoDB, a versão que utilizei foi a 7.0.8 além disso utilizei MongoDB compass na versão 1.42.5 como GUI.

Após possuir as ferramentas instaladas a aplicação spring se comunicará com o banco pela porta padrão 27017 e o database de desenvolvimento será chamado de *gomining*, sendo que mais tarde ao realizar os testes será criado outro banco chamado *test*, sendo assim nossos testes na aplicação não afetarão nossa aplicação de desenvolvimento.

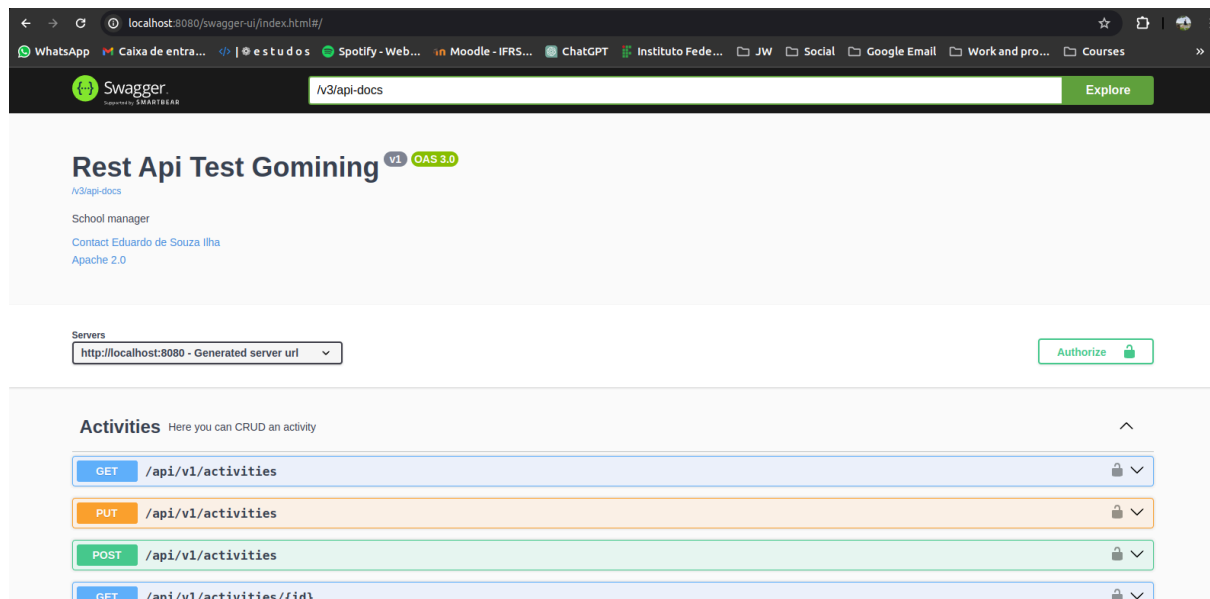
```
o cd /home/eddy/dev/gomining/test ; /usr/bin/env /ueddy@eddy:/dev/gomining/test$ cd /home/eddy/dev/gomining/test ; /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java @/tmp/cp_47hpin9j5ty3cdntxm4wsxnl.argfile gomining.test.TestApplication

[Spring]
:: Spring Boot :: (v2.4.5)

2024-04-21 20:48:00.676 INFO 108402 --- [ restartedMain] gomining.test.TestApplication : Starting TestApplication using Java 11.0.22 o
n eddy with PID 108402 (/home/eddy/dev/gomining/test/target/classes started by eddy in /home/eddy/dev/gomining/test)
2024-04-21 20:48:00.678 INFO 108402 --- [ restartedMain] gomining.test.TestApplication : No active profile set, falling back to default
t profiles: default
2024-04-21 20:48:00.718 INFO 108402 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'sprin
g.devtools.add-properties' to 'false' to disable
2024-04-21 20:48:00.718 INFO 108402 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider s
etting the 'logging.level.web' property to 'DEBUG'
2024-04-21 20:48:01.109 INFO 108402 --- [ restartedMain] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data MongoDB repository
```

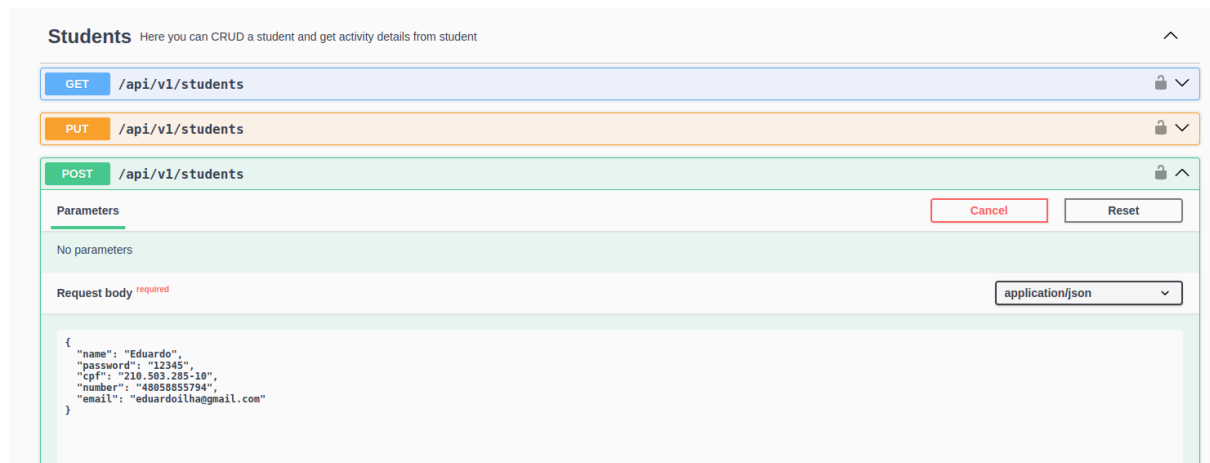
Após inicializar a aplicação (usando a porta padrão 8080) é possível acessar no navegador a rota :

- <http://localhost:8080/swagger-ui/index.html#/>



Aqui está disponível a documentação da nossa API, sendo que em primeiro momento as únicas rotas disponíveis são `api/v1/students` com o método POST para o cadastro do estudante e a rota `api/v1/auth` para a autenticação e geração do JWT.

A aplicação possui diversos tipos de exceções possibilitando um retorno seguindo o modelo de APIs Rest. Para criar um usuário basta acessar Students e acessar o método post



Após o envio da e confirmação da criação do usuário é possível realizar a autenticação com email e senha

auth-controller

POST /api/v1/auth

Parameters

No parameters

Request body *required* application/json

```
{
  "password": "12345",
  "email": "eduardoilha@gmail.com"
}
```

Sendo assim a aplicação irá gerar um token com validade de 50 min:

Content-type: application/json

```
-d '{
  "password": "12345",
  "email": "eduardoilha@gmail.com"
}'
```

Request URL

http://localhost:8080/api/v1/auth

Server response

| Code | Details |
|------|---|
| 200 | <p>Response body</p> <pre>{ "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ3dWUiOiJlZHVhcmRvaXoYURnbnFpbC5jb20iLCJpYXQiOiJlZDQ0MTM3NDQyNTMsImV4cCI6MTcxMzc0NzIxMywicm9sZSI6ImlwVURFTlQ1fQ..xbxaQYYyc055x60q0Ut8XA-OzXC4Z23tqpAsSsgwtw" }</pre> <p>Response headers</p> <pre>cache-control: no-cache,no-store,max-age=0,must-revalidate connection: keep-alive content-type: application/json date: Mon, 22 Apr 2024 00:03:33 GMT expires: 0 keep-alive: timeout=60 pragma: no-cache transfer-encoding: chunked x-content-type-options: nosniff x-frame-options: DENY x-xss-protection: 1; mode=block</pre> |

Responses

Basta copiar esse token e colar na parte de cima da página e assim o token será enviado no header de todas as requisições, portanto sendo um usuário logado.

School manager

Contact Eduardo de Souza Iha

Apache 2.0

Servers

http://localhost:8080 - Generated server url

Activities Here you can CRUD an activity

GET /api/v1/activities

PUT /api/v1/activities

POST /api/v1/activities

GET /api/v1/activities/{id}

Available authorizations

jwtToken (http, Bearer)

Value:

XA-OzXC4Z23tqpAsSsgwtw

Authorize Close

A partir desse momento já é possível criar atividades, excluir atividades entre outras alternativas de CRUD, porém a atribuição de atividade aos estudantes ocorre ao realizar o update do usuário, sendo que o update pode ser realizado sem ter os dados das atividades, devido às liberdades que o banco de dados não relacional oferece, porém também é possível adicionar mais atividades ao array e suas respectivas notas.

LEMBRE-SE sempre que uma atividade for excluída ela excluirá seus dados e notas do estudante também.

Ao criar uma atividade é possível usar seu id para inserir uma nova atividade no estudante (Não é possível atribuir atividades inexistentes ao estudante)

Aqui vemos um exemplo de uma atualização no estudante (passando dados dentro de `activityAndGrades`)

Por meio desse método não é possível fazer alteração na senha nem mesmo no email pois devem ser rotas diferentes já que tratam de dados únicos, pois o email é o identificador do usuário e está contido no payload do JWT.

Agora já é possível verificar por exemplo a média de notas e todas as notas de um estudante.

Média entre 10 e 5

Utilizei o Mapper para mapear os objetos de Request e response possibilitando uma comunicação entre as camadas, sendo assim é possível evitar envio de dados desnecessários como resposta dos endpoints.

A atualização dos usuários somente podem ser realizadas se o usuário logado é o mesmo que está sendo atualizado evitando a modificação de dados entre os usuários (Não tive tempo de completar a criação de papéis dentro da aplicação, sendo que somente o admin ou professor poderia fazer a atualização dos estudantes, mas fica para uma próxima implementação)