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:

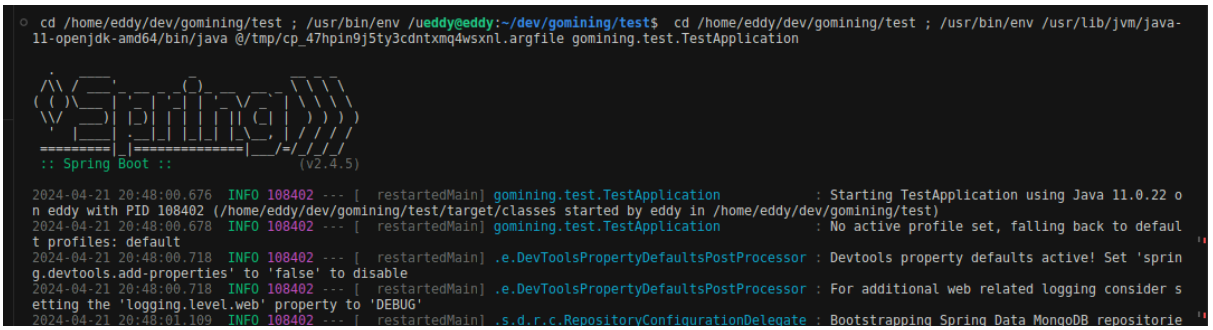
- Java 11 - Utilizei o Java 11 pois era um requisito do teste.
- Maven - Utilizei o maven para o gerenciamento de dependências.
- Spring Boot - O framework Spring com suas respectivas dependências.
- MongoDB - Banco de dados não relacional em documentos
- JJWT - Biblioteca para com funções para manipulação 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.

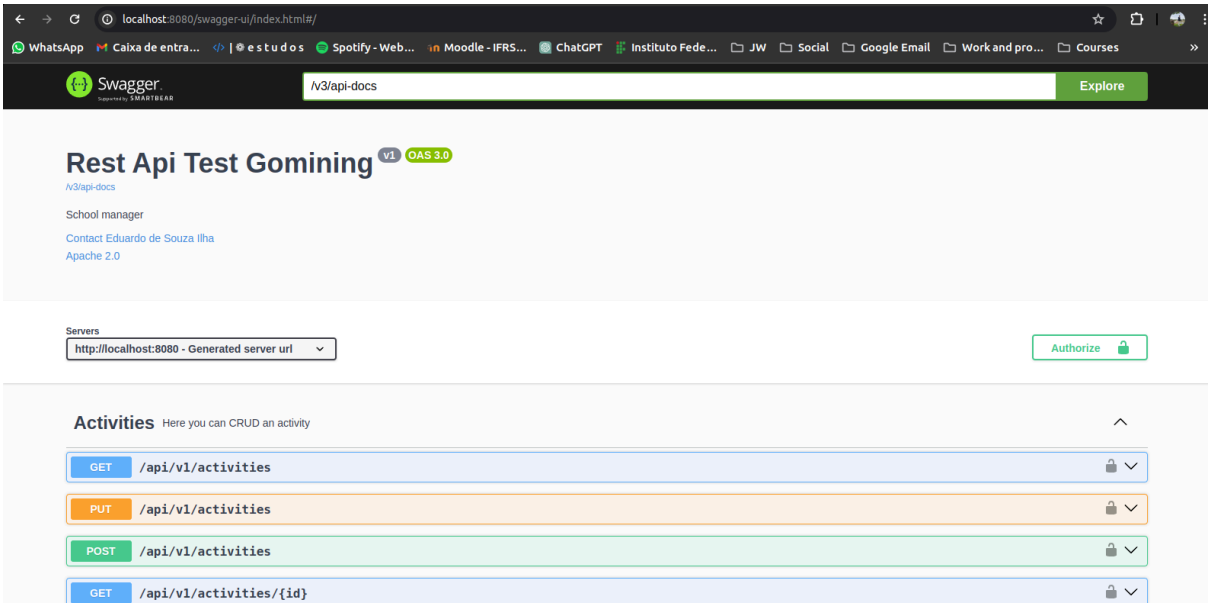
Link para instalação: https://www.mongodb.com/try/download/community

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.



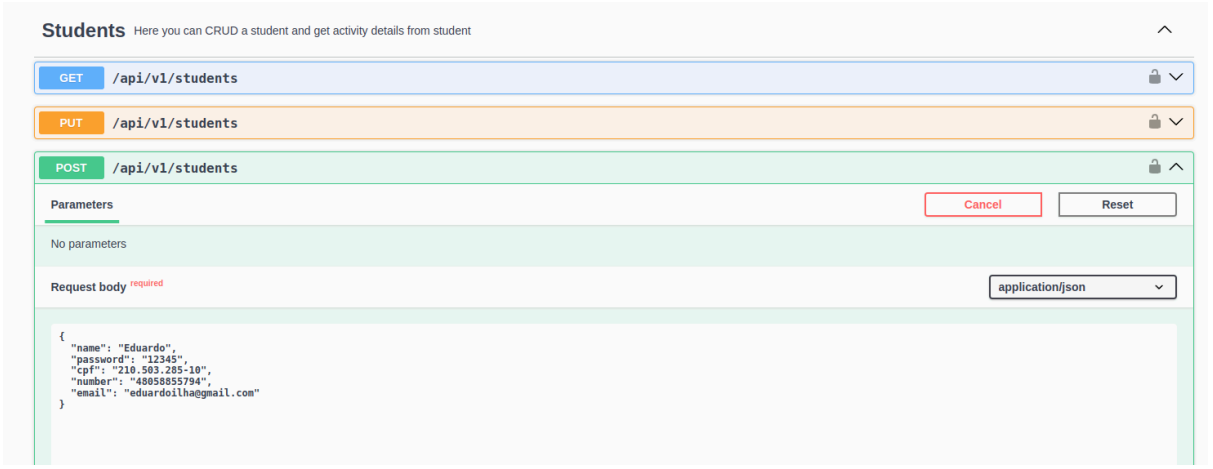
Após inicializar a aplicação (usando a porta padrão 8080) é possível acessar no navegador a rota :

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

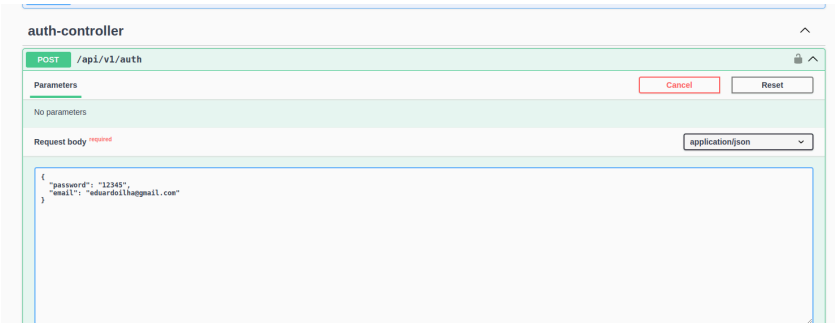


Aqui está disponível a documentação da nossa API, sendo que em primeiro momento as únicas rotas disponíveis são api/v1/students com o método POST para o cadastro do estudante e a rota api/v1/auth para a autenticação e geração do JWT.

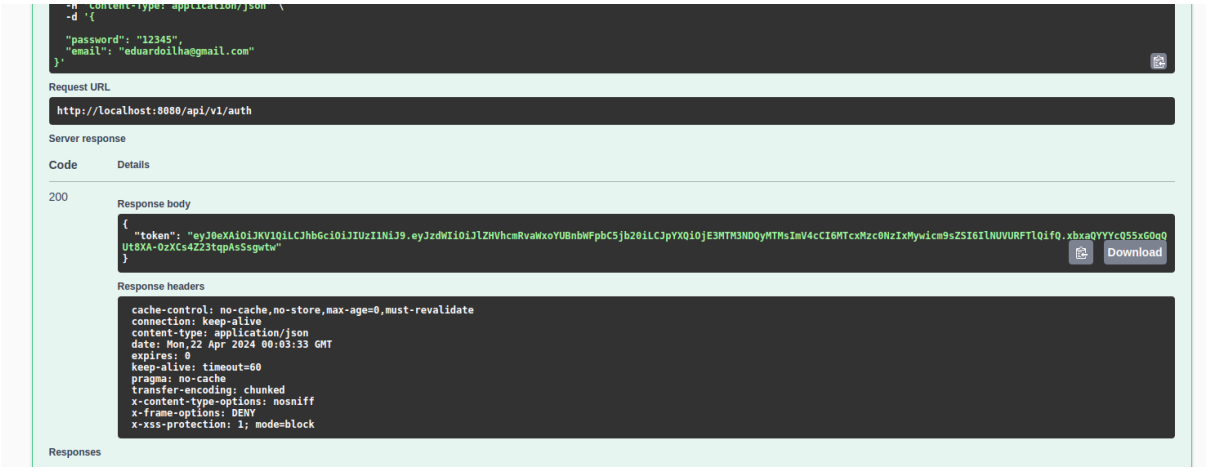
A aplicação possui diversos tipos de exceções possibilitando um retorno seguindo o modelo de APIs Rest. Para criar um usuário basta acessar Students e acessar o metodo post



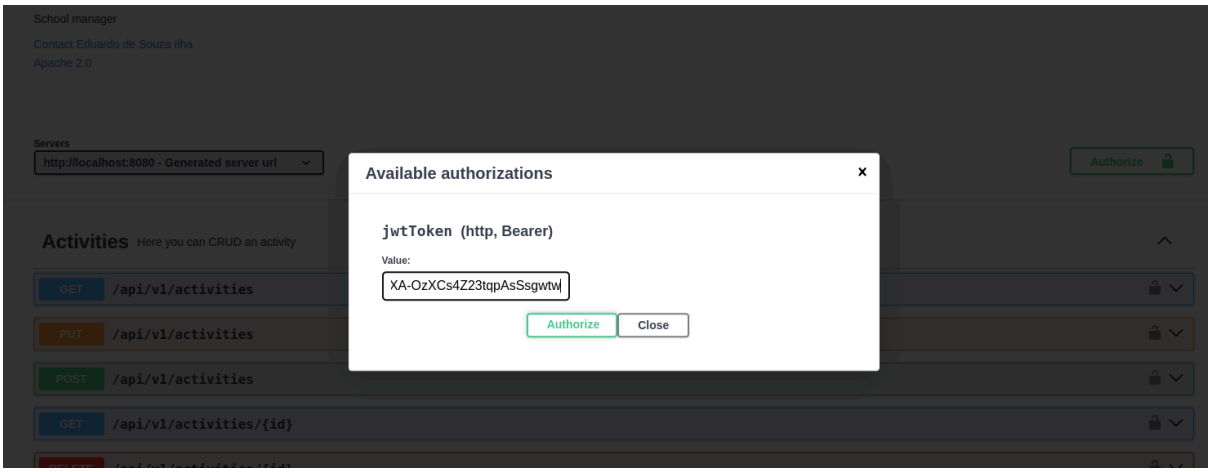
Após o envio da e confirmação da criação do usuário é possível realizar a autenticação com email e senha



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



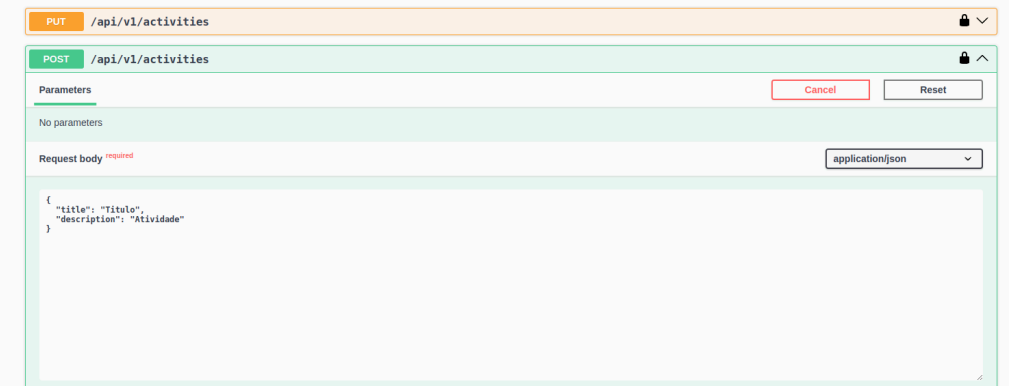
Basta copiar esse token e colar na parte de cima da página e assim o token será enviado no header de todas as requisições, portanto sendo um usuário logado.



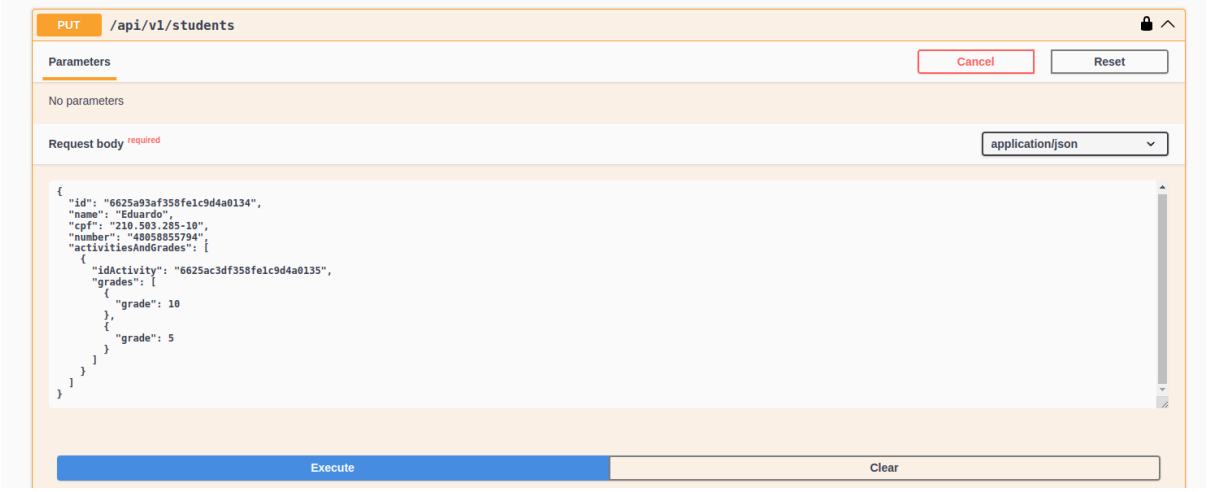
A partir desse momento já é possível criar atividades, excluir atividades entres outras alternativas de CRUD, porém a atribuição de atividade aos estudantes ocorre ao realizar o update do usuário, sendo que o update pode ser realizado sem ter os dados das atividades, devido às liberdades que o banco 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)

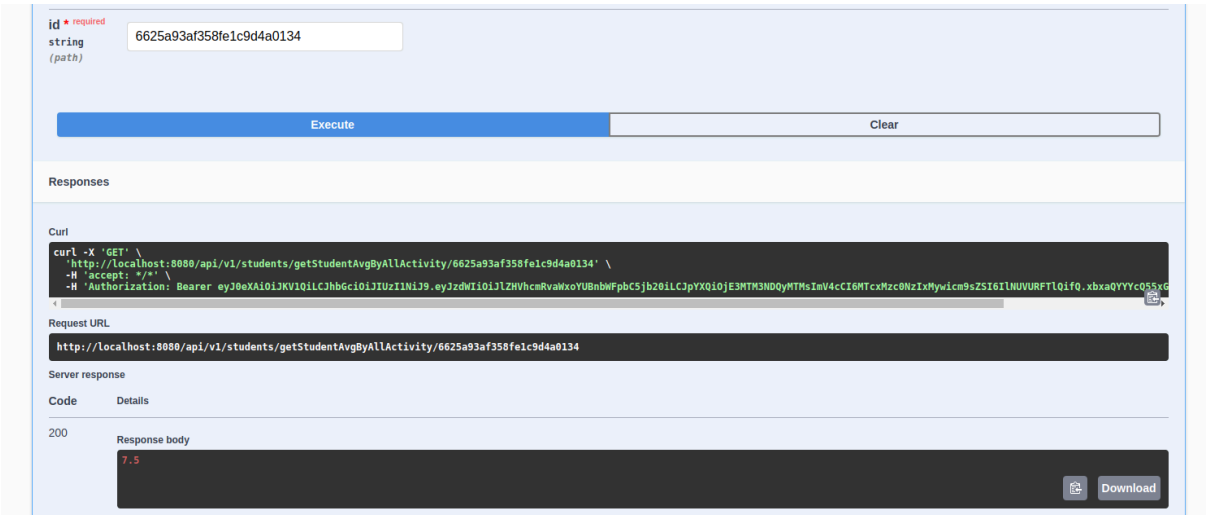


Aqui vemos um exemplo de uma atualização no estudante (passando dados dentro de activityAndGrades)

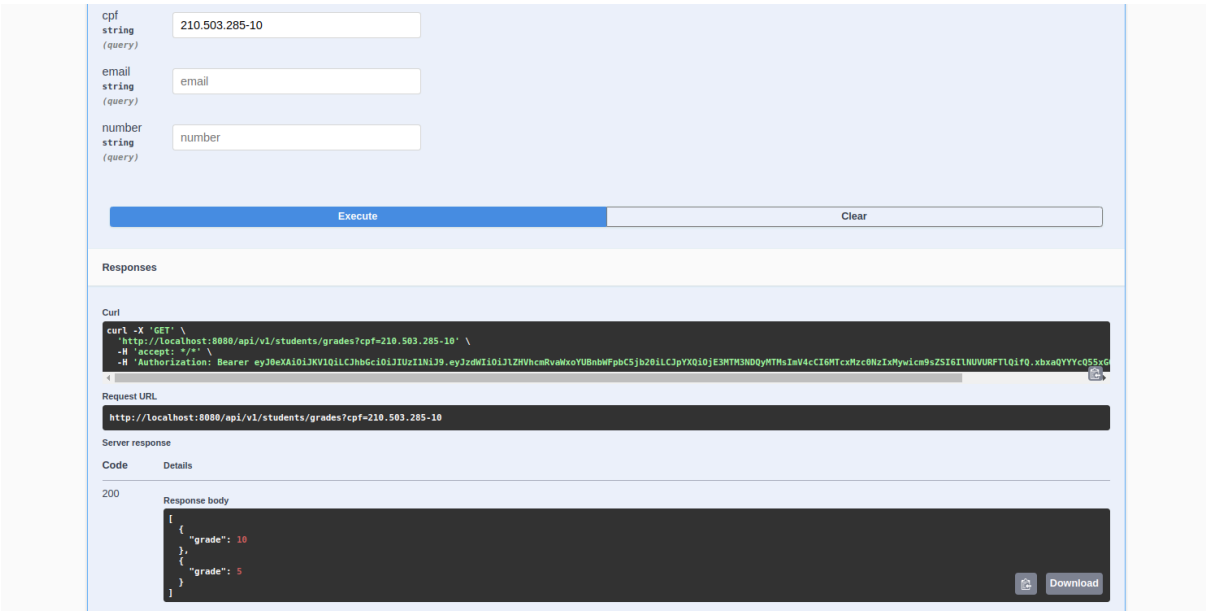


Por meio desse método não é possível fazer alteração na senha nem mesmo no email pois devem ser rotas diferentes já que tratam de dados únicos, pois o email é o identificador do usuário e está contido no payload do JWT.

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



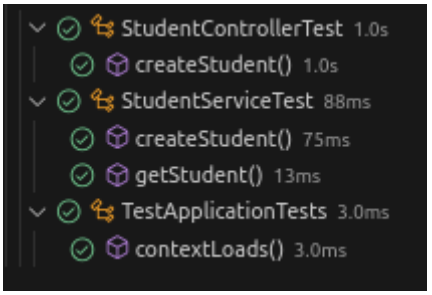
Média entre 10 e 5



Todas notas do estudante

Testes

Dois tipos de testes foram criados: um teste de ponta a ponta de criação de usuário pelo controller e dois testes unitários do studentService. Lembrando que tendo o mongoDB rodando no momento de rodar os testes será criado um banco de testes, porém seus valores não serão persistidos pois no final do teste os valores inseridos sempre são apagados.



Explicação

Optei por utilizar apenas dois documentos no banco de dados seguindo as regras do banco de dados não relacional (sem integridade referencial), sabendo que a aplicação possui estudantes, notas e atividades, dentro dos estudantes criei um novo objeto em forma de wrapper que possui o id da atividade e as notas atribuídas aquele estudante para aquela atividade. Sendo assim, a atividade pode ser reaproveitada para mais de um estudante sem a necessidade de criar atividades iguais para cada estudante.

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)