

Universidade Federal da Paraíba
Ciências da Computação
Banco de Dados I

Projeto Final(CRUD)

Aluno
Abraão Homualdo - 20200095558

1 Introdução

O objetivo principal do projeto era o desenvolvimento de um CRUD performático utilizando qualquer tipo de sGBD conectado à um interface[1] desenvolvida com tecnologias front-end ou diretamente ao terminal. Com base nos requisitos o SynchronoBank (CRUD desenvolvido) tem como característica principal o gerenciamento de contas e de cartões cadastrados no banco, além de conter algumas trigger's lançadas diretamente do DB (database) como também algumas excessões acionadas pela API (criada com Spring). Ao decorrer do relatório serão especificadas as tecnologias e framework's, endpoint's, problemas nas formas normais da criação dos relacionamentos do banco de dados, erro de requisições geradas na conexão com API e a solução de todas essas 'issues'.

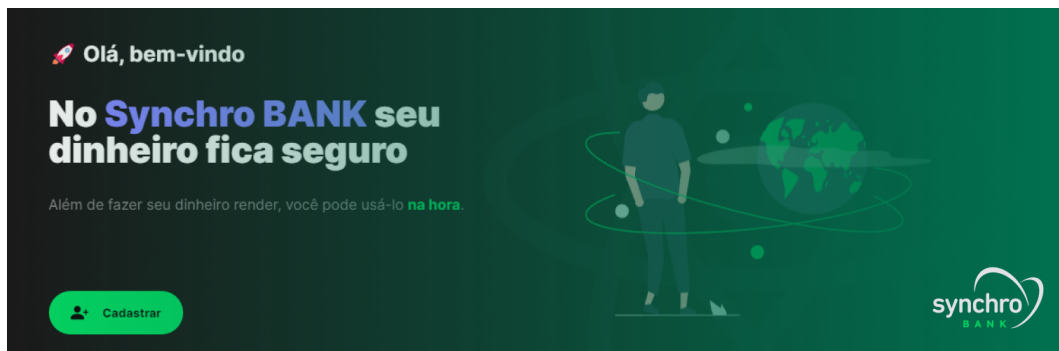


Figura 1: Capa do Readme do Projeto

2 Desenvolvimento

Com base no que foi solicitado, foi traçado as tecnologias que seriam utilizadas e implementadas. Seguindo a lógica da imagem anexada[2], de primeira instância foi selecionado como tecnologia front-end o Vite.JS (Framework ReactJs) que por sua vez trás um 'boost' para os motores de busca do APP em questão, no setor do back-end o Spring (Framework Java) foi o mais performático para uma criação em si do CRUD e para o Banco de Dados foi notório o uso do MySQL, tanto para lançamento de excessões e gatilhos como também de comunicação com o front. A comunicação para requisições do client-side para o server-side foi através do Axios (Promise HTTP client), no qual requisita o que o usuário solicita e entrega tudo no formato JSON tornando uma comunicação bem mais corrente.

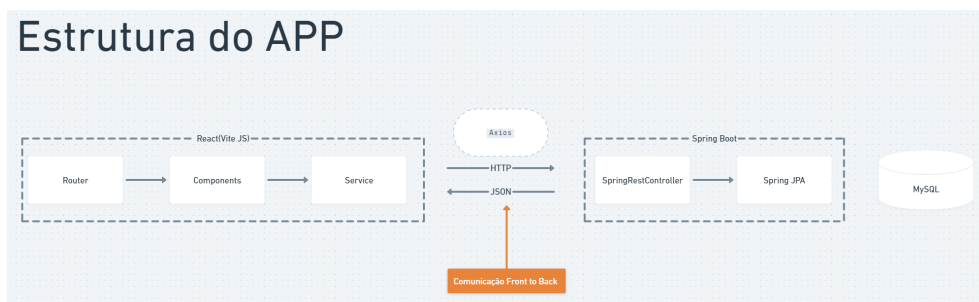


Figura 2: Estrutura do CRUD

Conectando a base teórica visto em sala de aula, era necessário uma verificação dos relacionamentos e cardinalidades como também os atributos das tabelas de tal forma que suprissem pelo menos a 3ª forma normal (3ªFN). Portanto, foram gerados três tipos de tabela para o problema em Diagrama E-R ser solucionado. Os critérios são mostrados nas imagens abaixo, tanto no modelo conceitual[3] e relacional[4], tendo em vista que no conceitual não se faz visível as 'stranger primary keys' do relacionamento de cada tabela, mostrado por completo no modelo relacional.

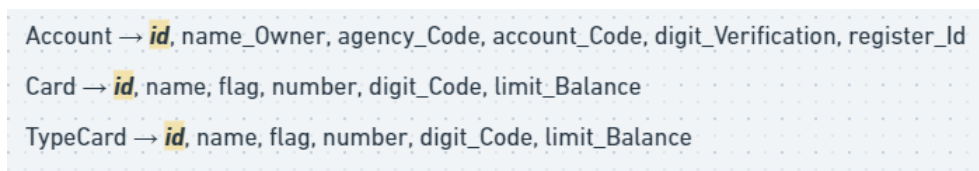


Figura 3: Modelo Conceitual do CRUD

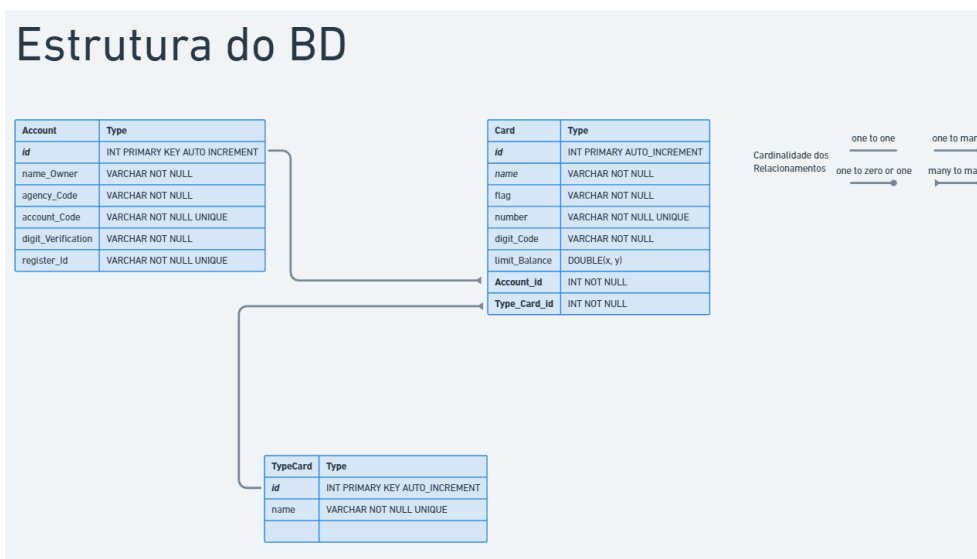


Figura 4: Modelo Relacional do CRUD

Envolvendo com tópico, as conexões tinham que ser feitas através de alguma porta padrão[5] e solicitadas diretamente na API, dessa forma foi utilizado o XAMPP (Server Open-Source) para conectar o banco e assim alimentar a API[6] para armazenamento dos dados que seriam requisitados posteriormente pelo usuário.

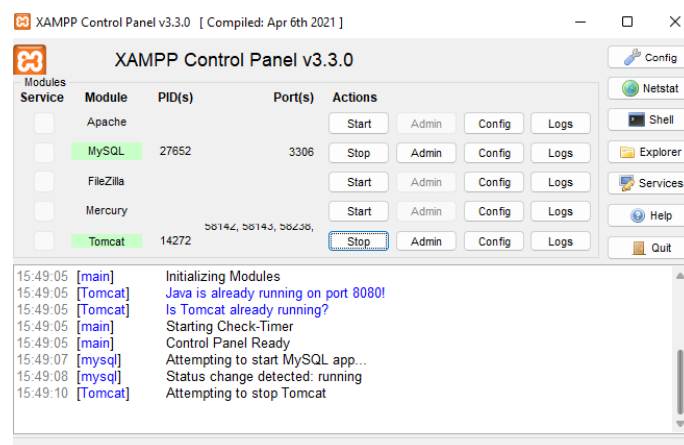
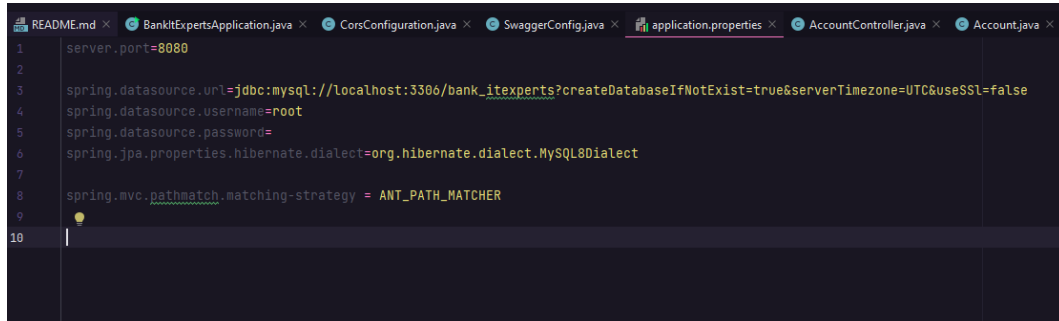



Figura 5: Porta:3306 MySQL através do XAMPP



```
1 server.port=8080
2
3 spring.datasource.url=jdbc:mysql://localhost:3306/bank_itexperts?createDatabaseIfNotExist=true&serverTimezone=UTC&useSSL=false
4 spring.datasource.username=root
5 spring.datasource.password=
6 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
7
8 spring.mvc.pathmatch.matching-strategy = ANT_PATH_MATCHER
9
10
```

Figura 6: Configuração Spring(API) com o Banco

No decorrer do projeto foi solicitado um lançamento de uma trigger, e gerou inúmeros conflitos com a API que por sua vez já tinha algumas excessões[7] e tratamentos para inserção de dados do usuário. Isso foi um dos principais empecilhos para mal funcionamento do app, como alternativa foi implementado uma trigger 'before insert and update row'[8] para o tratamento na geração de um cartão de crédito, onde não era possível ter menos do que 16 caracteres no 'number', assim privando e entregando resultados peformáticos no deploy.



```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity<StandardError> methodArgumentNotValid(MethodArgumentNotValidException exception, HttpServletRequest request){
    List<ValidationError> errors = exception.getBindingResult().getFieldErrors().stream().stream() Stream<FieldError>
    .map(error -> new ValidationError(error.getObjectName(), error.getField(), error.getDefaultMessage())) Stream<ValidationError>
    .collect(Collectors.toList());

    StandardError err = new StandardError();
    err.setTimestamp(Instant.now());
    err.setCode(HttpStatus.BAD_REQUEST.value());
    err.setStatus(HttpStatus.BAD_REQUEST.name());
    err.setMessage("Um ou mais dados não estão no formato esperado.");
    err.setPath(request.getRequestURI());
    err.setErrors(errors);

    return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(err);
}
```

Figura 7: Exceção de Conflito

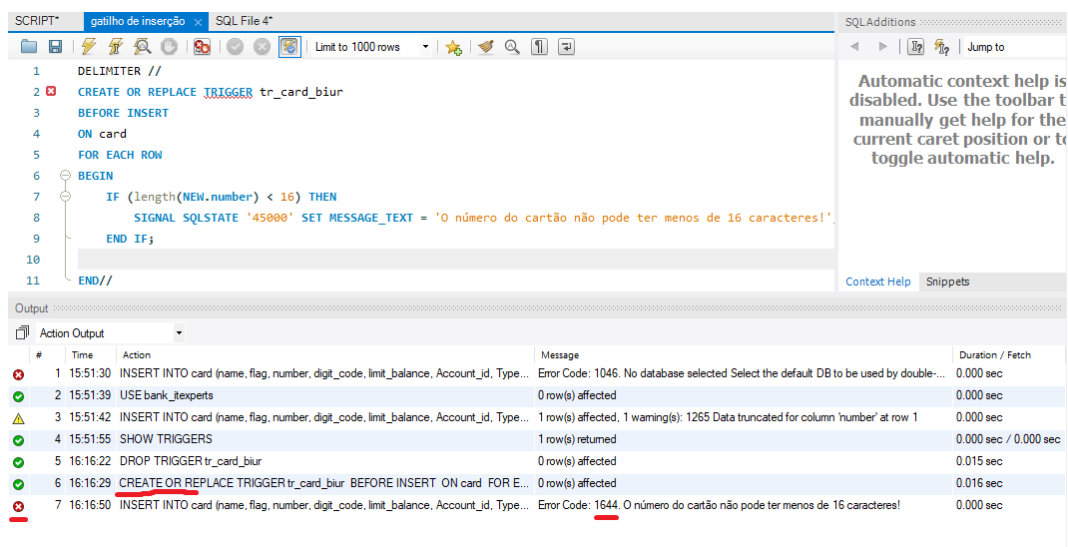


Figura 8: Trigger Criada e Output de teste

Outro problema que estava privando as requisições do servidor, foi o erro comum de CORS (Cross-origin Resource Sharing) no qual o Spring barrava as requisições (GET, POST, PUT, DELETE) do Axios para o front-end, como alternativa foi criado o arquivo CorsConfigurator[9] que trouxe flexibilidade na entrega dos arquivos json para alimentar as telas com as informações necessárias.



Figura 9: Desabilitando limitações do CORS

Por fim, com aplicabilidade completa e funcional foram gerados as rotas de acesso e requisição dos dados que seriam armazenados no banco de dados. [10]

Endpoint	Router	Description
post	/api/v1/accounts	Criar uma nova conta
get	/api/v1/accounts/{id}	Buscar Conta[ID]
delete	/api/v1/accounts/{id}	Deleta uma conta existente[ID]
post	/api/v1/accounts/{id}/cards	Criar cartão em uma conta[ID]
delete	/api/v1/accounts/{id}/cards/{id}	Deleta um cartão de uma conta existente[ID]

Figura 10: Endpoint's da API

3 Conclusão

Desde o início dos trabalhos, a grande motivação para a realização deste projeto é o fato de que o aprendizado seria bastante enriquecedor, onde o interesse no projeto era de finalidade e propósito de utilizá-lo para falhas de ausência de tecnologia pessoal. Então a forma que foi tratada foi bastante desafiador, superando limites que vistos no desenvolvimento e conclusão do mesmo seriam impossíveis. No mais, a aplicação[11] ainda necessita de melhorias e de algumas telas não inseridas(cadastro de cartões de crédito) e containerização(ou containerização) do banco de dados utilizando Docker.

Repositório

AbraaoDEV, <https://github.com/AbraaoDev/synchrobank>