Tech Challenge - FIAP

Projeto: API de Locação de Veículos;
Curso: ARQUITETURA E DESENVOLVIMENTO EM JAVA - Fase 1;
Escola: FIAP;

Grupo:
Julio Cesar de Almeida
Victor Luiz Montibeller
Leonardo Arantes Di Nizo
Tiago Germano Pereira
Rafael Sugahara Francisco

Ferramentas utilizadas:

Event Storming: Miro;
API:
- Linguagem: JAVA 17;
- Frameworks: Spring, JPA, Hibernate, Lombok;
- Banco de dados: H2;

Objetivo do Projeto

Desenvolvimento de um sistema para digitalizar o negócio de uma locadora de veículos, otimizando os seus serviços.

Uma locadora aluga seus veículos para clientes previamente cadastrados ou cadastra novos clientes. A partir deste cadastro, um cliente pode escolher um veículo e realizar a reserva, de acordo com o período escolhido, o status do veículo e a confirmação de valores contratuais.

Após a reserva, se o cliente concordar com os termos, um contrato é gerado. A partir do contrato, dados do veículo e período de locação, é calculado o valor de locação e solicitado para o cliente os dados do cartão de crédito para que seja possível o pagamento, que será processado por uma API externa (PagBank).

Na retirada do veículo é gerado também um checklist para conferência pelo cliente e no retorno do mesmo a área de manutenção preenche um novo checklist para verificar o que precisa ser feito no veículo e se haverá alguma cobrança extra, sempre atualizando o status do veículo (disponível, alugado ou em manutenção).

Event Storming

Para este Tech Challenge foi realizado um projeto de API de Locação de Veículos, na qual iniciamos através de um Event Storming.

O Event Storming foi iniciado com um *Brainstorming*, onde discutimos, entre os participantes do grupo, ideias que descrevessem os eventos da nossa API, simulando uma reunião com especialistas do negócio. Após essa fase, iniciamos a organização desses eventos em uma linha do tempo, demonstrando assim a ordem desses eventos e suas dependências. Nesta fase, começamos a pensar também alguns pontos de atenção da API. Após a organização dessas ideias em uma linha do tempo, separamos o mesmo em Eventos Pivotais, definindo onde termina uma parte do processo e iniciava outra.

Terminada essa fase, adicionamos as ações, atores, interfaces e outros pontos de atenção que surgiram.

Após a finalização desse fluxo, identificamos nosso objeto principal e agregados, separando-os para melhor visualização e representação.

E por fim separamos os Contextos Delimitados do nosso projeto.

Em cada fase, novas ideias foram surgindo e assim fomos refinando o processo ao longo de sua construção.

Todo esse processo de desenvolvimento do Event Storming e suas fases, podem ser visualizados através do link abaixo:
https://miro.com/app/board/uXjVMgUnBIU=/?share_link_id=244023865826

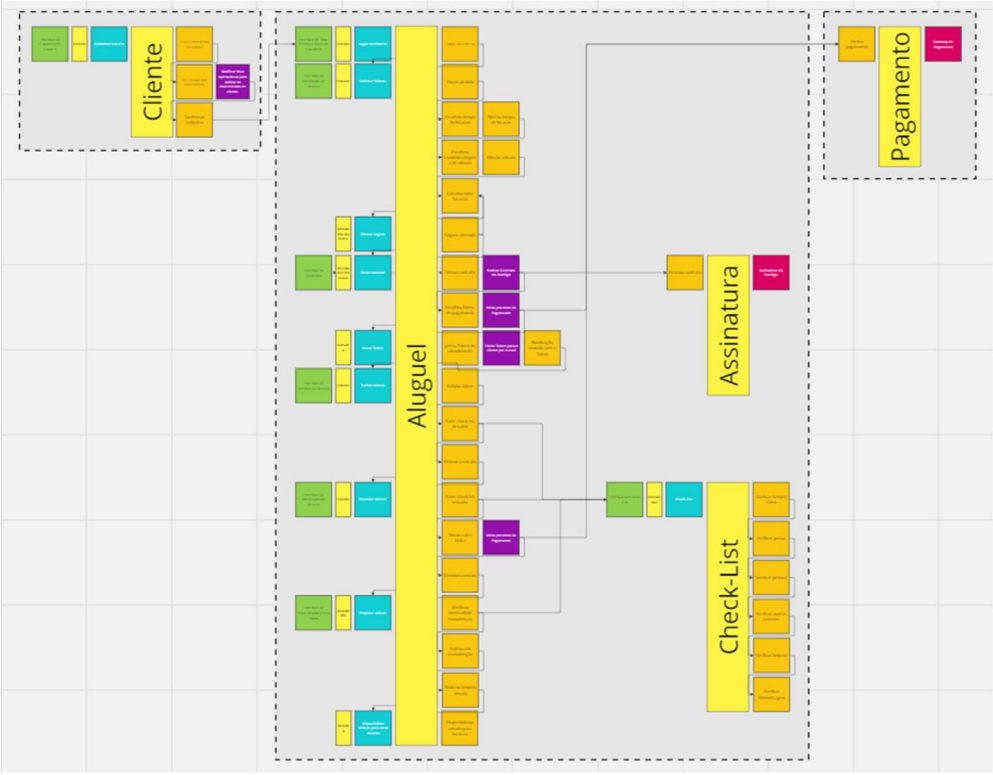

Figura 1 – Print da etapa final do Event Storming

Após a finalização do Event Storming, demos início a codificação de nossa API, seguindo as definições do Event Storming. Essa API pode ser visualizada no seguinte repositório GitHub:
https://github.com/JulioBCP/TechChallenge/tree/master

Arquitetura do Projeto

Para a arquitetura da API, nós separamos primeiramente os 3 contextos delimitados do nosso projeto (Cliente, Aluguel e Pagamento). Como a parte de Assinatura e CheckList são agregados que fazem parte do mesmo contexto delimitado, deixamos eles dentro do mesmo contexto arquitetural.

Dentro de cada contexto delimitado, escolhemos utilizar um padrão de arquitetura MVC, separando o código em: entities, controllers, services e repositories. Adicionamos ainda DTOs para separarmos os parâmetros das entidades que seriam passiveis de update, para assim não expor todos os campos das entidades na camada de controle, abstraindo somente os campos necessários para tal. Criamos ainda um pacote exceptions, para construção das classes, que darão um tratamento mais específico a cada entidade. Para classes utilitárias, como validações e cálculos, criamos também o pacote utils.

No contexto delimitado de pagamentos, realizamos uma integração com API externa (PagBank) para processamento do pagamento.

Melhorias futuras

Após a entrega da fase 1 deste Tech Challenge, deixamos planejado algumas melhorias futuras, tais como:

- Validações de todos os campos convenientes das entidades do nosso projeto.
- Outras validações, como a de alguns documentos duplicados, ainda não realizados.
- Possibilidade de integração com outros meios de pagamento, além do cartão de crédito.
- Interface gráfica (Front-end) para integração com a API, possibilitando uma melhor visualização dos processos.
- Implementação do status do veículo, para verificar se ele já está alugado, em manutenção ou disponível.

Testando a API

Para testar nossa API, temos alguns exemplos de payloads do tipo Json para o envio de requisições via Postman ou outra ferramenta semelhante. Esses payloads se encontram no projeto que pode ser clonado à partir do seguinte repositório:

https://github.com/JulioBCP/TechChallenge/tree/master

Após clonar este repositório, vá até o seguinte caminho no projeto: car-rental\car-rental\src\test\java\com\fiap\techchallenge\carrental\payloadsJson

Note que este caminho se encontra na área de testes automatizados da API. Utilizando qualquer um dos payloads disponibilizados para estes testes manuais, poderemos fazer as seguintes requisições, conforme veremos abaixo.

Outro ponto importante são as dependências de cada classe. Precisamos primeiramente criar um usuário, portanto a primeira requisição a ser feita será do Usuário. O qual contém login e senha, nome e tipo de usuário, conforme exemplo abaixo:

Inserções

Método: POST - inserirUsuario

URL: localhost:8080/usuarios

Exemplo de Body (json)

```
{
   "login": "joaodoe",
   "senha": "senha123",
   "nome": "João da Silva",
   "tipoUsuario": "CLIENTE"
}
```

Após termos um usuário cadastrado, podemos cadastrar todos os dados de um cliente no sistema, logo podemos fazer a seguinte requisição POST:

Método: POST – inserirCliente

URL: localhost:8080/clientes

Exemplo de Body (json)

```
{
   "numeroCnh": 12345678910,
   "dataNascimento": "1990-01-15",
   "email": "joaodoe@email.com",
   "sexo": "MASCULINO",
   "cpf": 12345678910,
   "telefone": 19998765362,
   "endereco": {
       "rua": "Rua das Flores",
       "numero": "123",
       "bairro": "Centro",
       "cidade": "Cidade",
       "estado": "UF",
       "pais": "Brasil",
       "CEP": "12345-678"
   },
   "usuario": {
       "id": 1,
       "login": "joaodoe",
       "senha": "senha123",
       "nome": "João da Silva",
       "tipoUsuario": "CLIENTE"
   }
}
```

A qualquer momento podemos também cadastrar um veículo, pois este não depende de nenhuma outra entidade.
Para cadastrarmos um veículo temos a url e o exemplo abaixo:

Método: POST – inserirVeiculo
URL: localhost:8080/veiculos
Exemplo de Body (json)

```
{
   "tipoVeiculo": "HATCH",
   "marca": "TOYOTA",
   "modelo": "Corolla",
   "anomodelo": 2022,
   "anoFabricacao": 2021,
   "km": 15500,
   "kmManutencao": 14000,
   "motor": "2.0L 4 cilindros",
   "cor": "Prata",
   "tracao": "T4X2",
   "arCondicionado": true,
   "trioEletrico": true,
   "tetoSolar": false,
   "combustivel": "GASOLINA",
   "cabineDupla": false,
   "capacidadeCarga": "500 kg"
}
```

Temos vários payloads de veículos como exemplo para serem cadastrados. Se precisar cadastrar mais veículo, vá a pasta do projeto “outrosVeiculos” para payloads extras.

Tendo um cliente cadastrado no sistema, podemos fazer uma reserva de um veículo. Para tal, temos a requisição abaixo:

Método: POST – inserirReserva
URL: localhost:8080/reservas
Exemplo de Body (json)


```
{
   "cliente": {
    "numeroCnh": 12345678910,
    "dataNascimento": "1990-01-15",
    "email": "joaodoe@email.com",
    "sexo": "MASCULINO",
    "cpf": 12345678910,
    "telefone": 19998765362,
    "endereco": {
        "rua": "Rua das Flores",
        "numero": "123",
        "bairro": "Centro",
        "cidade": "Cidade",
        "estado": "UF",
        "pais": "Brasil",
        "CEP": "12345-678"
    },
    "usuario": {
        "id": 1,
        "login": "joaodoe",
        "senha": "senha123",
        "nome": "João da Silva",
        "tipoUsuario": "CLIENTE"
    }
 },
   "veiculo": {
   "id": 1,
   "tipoVeiculo": "HATCH",
   "marca": "TOYOTA",
   "modelo": "Corolla",
   "anomodelo": 2022,
   "anoFabricacao": 2021,
   "km": 15500,
   "kmManutencao": 14000,
   "motor": "2.0L 4 cilindros",
   "cor": "Prata",
   "tracao": "T4X2",
   "arCondicionado": true,
   "trioEletrico": true,
   "tetoSolar": false,
   "combustivel": "GASOLINA",
   "cabineDupla": false,
   "capacidadeCarga": "500 kg"
   },
   "valorReserva": 1300.0,
   "dataInicio": "2023-01-15",
   "dataFim": "2023-01-20"
}
```

Tendo uma reserva cadastrada no sistema, podemos fechar um contrato. A requisição para o contrato segue o formato abaixo:

Método: POST – inserirContrato
URL: localhost:8080/contratos
Exemplo de Body (json)


```
{
   "formaPagamento": "CREDITO",
   "multaDiaria": 50.0,
   "reserva": {
        "id": 1,
    "cliente": {
        "numeroCnh": 12345678910,
        "dataNascimento": "1990-01-15",
        "email": "joaodoe@email.com",
        "sexo": "MASCULINO",
        "telefone": 19998765362,
        "endereco": {
            "rua": "Rua das Flores",
            "numero": 123,
            "bairro": "Centro",
            "cidade": "Cidade",
            "estado": "UF",
            "pais": "Brasil",
            "cep": 0
        },
        "usuario": {
            "id": 1,
            "login": "joaodoe",
            "senha": "senha123",
            "nome": "João da Silva",
            "tipoUsuario": "CLIENTE"
        },
        "cpf": 12345678910
    },
    "veiculo": {
        "id": 1,
        "tipoVeiculo": "HATCH",
        "marca": "TOYOTA",
        "modelo": "Corolla",
        "anomodelo": 2022,
        "anoFabricacao": 2021,
        "km": 15500,
        "kmManutencao": 14000,
        "motor": "2.0L 4 cilindros",
        "cor": "Prata",
        "tracao": "T4X2",
        "arCondicionado": true,
        "trioEletrico": true,
        "tetoSolar": false,
        "combustivel": "GASOLINA",
        "cabineDupla": false,
        "capacidadeCarga": "500 kg"
    },
   "valorReserva": 1300.0,
   "dataInicio": "2023-01-15",
   "dataFim": "2023-01-20"
    }
}
```

Na saída ou retorno do veículo, a equipe de manutenção realiza o check-list do veículo, pois em caso de necessidade, uma cobrança extra seja feita. Para inserir um checklist no sistema, temos a seguinte requisição:

Método: POST – inserirCheckList
URL: localhost:8080/checkLists
Exemplo de Body (json)


```
{
   "contrato":{
        "id": 1,
        "formaPagamento": "CREDITO",
        "multaDiaria": 50.0,
        "reserva":{
            "id": 1,
            "cliente": {
                "numeroCnh": 12345,
                "dataNascimento": "1990-01-15",
                "email": "joaodoe@email.com",
                "sexo": "MASCULINO",
                "CPF": 12345678901,
                "telefone": 987654321,
                "endereco": {
                    "rua": "Rua das Flores",
                    "numero": "123",
                    "bairro": "Centro",
                    "cidade": "Cidade",
                    "estado": "UF",
                    "CEP": "12345-678"
                },
                "usuario": {
                    "login": "joaodoe",
                    "senha": "senha123",
                    "nome": "João da Silva",
                    "tipoUsuario": "CLIENTE"
                }
            },
            "veiculo": {
                "id": 1,
                "tipoVeiculo": "HATCH",
                "marca": "TOYOTA",
                "modelo": "Corolla",
                "anomodelo": 2022,
                "anoFabricacao": 2021,
                "km": 15500,
                "kmManutencao": 14000,
                "motor": "2.0L 4 cilindros",
                "cor": "Prata",
                "tracao": "T4X2",
                "arCondicionado": true,
                "trioEletrico": true,
```

```
                "tetoSolar": false,
                "combustivel": "GASOLINA",
                "cabineDupla": false,
                "capacidadeCarga": "500 kg"
            },
            "valorReserva": 1300.0,
            "dataInicio": "2023-01-15",
            "dataFim": "2023-01-20"
        }
    },
   "abastecido": true,
   "pressaoPneus": 32,
   "avariasPintura": "Arranhões na porta esquerda",
   "avariasInterna": "Manchas no banco traseiro",
   "limpeza": "Interior limpo, exterior sujo",
   "km": 15500,
   "tipoCheckList": "ENTRADA"
}
```


É importante observar os ids de cada entidade. Nos exemplos acima, temos todas as entidades sendo criadas com id =1, porém esses ids são gerados automaticamente, portanto, se for cadastrado mais de um cliente, por exemplo, é necessário verificar a qual cliente está sendo vinculado a reserva, ou a qual reserva está sendo vinculado um contrato e assim por diante.

Consultas

Com todos os cadastros realizados, caso queira buscar algum dado de qualquer entidade, temos os seguintes métodos GET.

Método: GET - buscarUsuario
URL: localhost:8080/usuarios/{id}

Método: GET - buscarCliente
URL: localhost:8080/clientes/{numeroCnh}

Método: GET - buscarVeiculo
URL: localhost:8080/veiculos/{id}

Método: GET - buscarReserva
URL: localhost:8080/reservas/{id}

Método: GET - buscarContrato
URL: localhost:8080/contratos/{id}

Método: GET - buscarCheckList
URL: localhost:8080/checkLists/{id}

Atualizações

Da mesma forma temos os métodos PUT para updates. Note que não temos update de contrato, pois em caso de necessidade de alteração, deve ser feito um novo contrato:

Método: PUT- atualizarUsuario
URL: localhost:8080/usuarios/{id}
Para a entidade Usuários só é possível alterar login e senha, de acordo com o DTO recebido neste método.

Método: PUT- atualizarCliente
URL: localhost:8080/clientes/{numeroCnh}
Para a entidade Clientes só é possível alterar nome e endereço, de acordo com o DTO recebido.

Método: PUT - atualizarVeiculo
URL: localhost:8080/veiculos/{id}
Para a entidade Veiculos só é possível alterar km atual e km da próxima manutenção, de acordo com o DTO recebido neste método.

Método: PUT- atualizarReserva
URL: localhost:8080/reservas/{id}
Para a entidade Reservas é possível alterar o veículo, valor da reserva, data de início e data fim, de acordo com o DTO recebido neste método.

Método: PUT- atualizarCheckList
URL: localhost:8080/checkLists/{id}
Para a entidade CheckList é possível alterar os seguintes campos: abastecido, pressão dos pneus, avarias de pintura, avarias interna, limpeza e km atual, conforme DTO do mesmo.

Exclusões

E por fim temos também os métodos Delete, para exclusão de qualquer entidade do sistema, a partir de seu id:

Método: DELETE- deletarUsuario
URL: localhost:8080/usuarios/{id}

Método: DELETE- deletarCliente
URL: localhost:8080/clientes/{numeroCnh}

Método: DELETE- deletarVeiculo
URL: localhost:8080/veiculos/{id}

Método: DELETE- deletarReserva
URL: localhost:8080/reservas/{id}

Método: DELETE- deletarContrato
URL: localhost:8080/contratos/{id}

Método: DELETE– deletarCheckList
URL: localhost:8080/checkLists/{id}

Considerações Finais

Durante o desenvolvimento deste projeto, pudemos exercitar todos os conceitos aprendidos na fase 1 da Pós Tech, como Conceitos de DDD, Orientação a Objetos, Controle de Versionamento e Framework Spring.

Este trabalho teve como objetivo propor teoricamente e tecnicamente a digitalização do negócio de uma locadora de veículos, aprimorando nossas habilidades como desenvolvedores de software, conforme aplicamos todos os conceitos aprendidos nas aulas. Durante o processo de desenvolvimento, conseguimos construir uma visão mais consistente do que deve ser o conteúdo interação humano-computador do sistema.

O objetivo final do projeto é a criação de um sistema interativo com uma maior qualidade de uso, atendendo bem as necessidades, desejos e preferencias do usuário.