Clean Code e Clean Architecture - Turma 13

19% CONCLUÍDO

☆ ☆ ☆ ☆ ☆

- **Informações Importantes**
- **Aula 1 (04/09/2023 às 19:00)**
- **Aula 2 (11/09/2023 às 19:00)**
  - ✓ Assistir aula completa
  - ✓ Hexagonal Architecture e SRP
  - ○ Test Patterns e DIP - Stub, Spy, Mock e Fake
  - ○ Implementação do Projeto com Hexagonal e TDD
  - ✓ **Próxima etapa do projeto**
  - ✓ Q&A
- **Aula 3 (18/09/2023 às 19:00)**
- **Aula 4 (25/09/2023 às 19:00)**
- **Aula 5 (02/10/2023 às 19:00)**
- **Aula 6 (09/10/2023 às 19:00)**
- **Aula 7 (16/10/2023 às 19:00)**

LS

CERTIFICADO

✓ **CONCLUÍD**

Próxima etapa do projeto

← **ANTERIOR** **PRÓXIMO** →

Nessa etapa vamos implementar o início da corrida, atualização do trajeto e encerramento da corrida.

UC4 - StartRide
Ator: Motorista
Input: ride_id
Output: void

Regras:

- Deve verificar se a corrida está em status "accepted", se não estiver lançar um erro
- Deve modificar o status da corrida para "in_progress"

UC5 - UpdatePosition
Ator: Sistema (atualiza a cada 10 segundos de forma automática)
Input: ride_id, lat, long
Output: void

- Deve verificar se a corrida está em status "in_progress", se não estiver lançar um erro
- Deve gerar o position_id
- Deve salvar na tabela position: position_id, ride_id, lat, long e date

UC6 - FinishRide
Ator: Motorista
Input: ride_id
Output: void

- Deve verificar se a corrida está em status "in_progress", se não estiver

Clean Code e Clean Architecture - Turma 13

19% CONCLUÍDO

Informações Importantes

Aula 1 (04/09/2023 às 19:00)

Aula 2 (11/09/2023 às 19:00)

- [x] Assistir aula completa
- [x] Hexagonal Architecture e SRP
- [ ] Test Patterns e DIP - Stub, Spy, Mock e Fake
- [ ] Implementação do Projeto com Hexagonal e TDD
- [x] **Próxima etapa do projeto**
- [x] Q&A

Aula 3 (18/09/2023 às 19:00)

Aula 4 (25/09/2023 às 19:00)

Aula 5 (02/10/2023 às 19:00)

Aula 6 (09/10/2023 às 19:00)

Aula 7 (16/10/2023 às 19:00)

LS

CERTIFICADO

lançar um erro

- Deve obter todas as positions e calcular a distância entre cada uma delas, para isso utilize um algoritmo que receba duas coordenadas (lat, long) e retorne a distância entre elas em km.
- Com a distância total calculada, calcule o valor da corrida (fare) multiplicando a distância por 2,1
- Atualizar a corrida com o status "completed", a distância e o valor da corrida (fare)

Considere o modelo de dados:

```
create table cccat13.position (
position_id uuid,
ride_id uuid,
lat numeric,
long numeric,
date timestamp
);
```

Utilize TDD e continue seguindo o mesmo design definido na aula 2, separando a API (driver) das regras de negócio e do banco de dados (resource).

Segue o link para o algoritmo de cálculo da distância entre duas coordenadas https://github.com/rodrigobranas/distance_calculator/blob/master/src/DistanceCalculator.ts

Participe da discussão...

Postar