

≪ ONCLUÍE

# 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

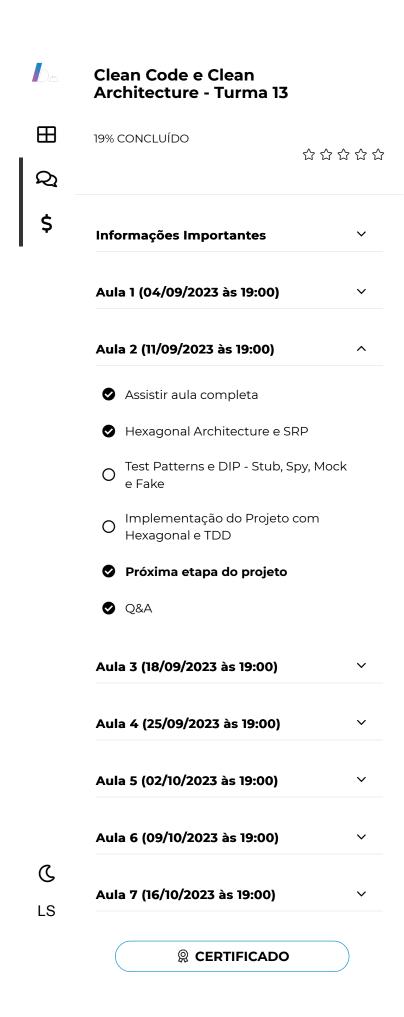
- Deve verificar se a corrida está em status "in\_progress", se não estiver lançar um erro
- Deve gerar o position\_id
- Deve salvar na tabela position: position\_id, ride\_id, lat, long e date

### **UC6 - FinishRide**

Ator: Motorista Input: ride\_id Output: void

• Deve verificar se a corrida está em status "in\_progress", se não estiver

1 of 2



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

2 of 2 20/09/23 20:59