Documentação do Backend

O backend foi escrito em Python, usando o framework Flask e a biblioteca SQLAlchemy para lidar com um banco de dados SQLite. O objetivo deste backend é fornecer um serviço de gerenciamento de coordenadas com a capacidade de adicionar, listar e exibir as coordenadas.

Modelo:

```python
from flask import Flask, render_template, redirect, request
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///coordinates.db'
db = SQLAlchemy(app)

class Coordinate(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    x = db.Column(db.Float)
    y = db.Column(db.Float)
    z = db.Column(db.Float)
    r = db.Column(db.Float)
```

No começo está os imports, iniciação do banco de dados e a classe Coordinate define como serão os dados que serão armazenados, que é um banco de dados SQLite, com o arquivo ‘coordinates.db’ sendo armazenado localmente

```python
@app.route('/')
def index():
    coordinates = Coordinate.query.all()
    return render_template('index.html', coordinates=coordinates)
```

.

Rotas e funções:

```python
@app.cli.command()
def createdb():
    db.create_all()
```

@app.cli.command(): A função createdb cria o banco de dados caso ele ainda não exista. Basta executar “flask createdb” no terminal

@app.route('/'): A função index retorna uma página HTML com as informações de uma tabela com todas as coordenadas registradas no banco de dados.

```
@app.route('/coords', methods=['POST'])
def coords():
    coor = Coordinate(
        x = request.form['x'],
        y = request.form['y'],
        z = request.form['z'],
        r = request.form['r']
    )
    db.session.add(coor)
    db.session.commit()
    return redirect('/')
```

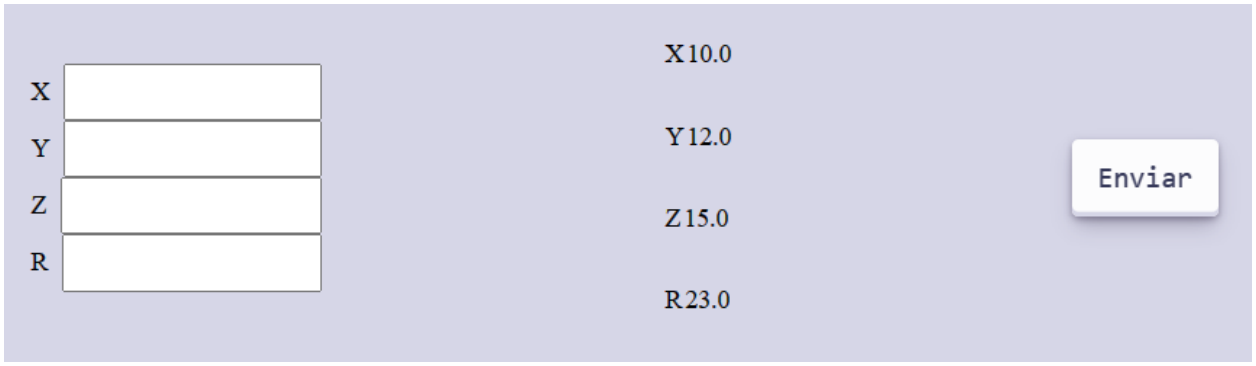
@app.route('/coords', methods=['POST']): A função coords é responsável por receber uma solicitação do tipo POST e adicionar uma nova entrada no banco de dados com os dados das novas coordenadas.

```
@app.route('/godot', methods=["GET", "POST"])
def godot_coords():
    coordenadas = Coordinate.query.all()
    if coordenadas:
        x = coordenadas[-1].x
        y = coordenadas[-1].y
        z = coordenadas[-1].z
        r = coordenadas[-1].r
        godotstring = f"{x}/{y}/{z}/{r}"
        return godotstring
```

@app.route('/godot', methods=['GET']): Quando a rota é acessada, ela executa a função godot_coords(), que recupera as coordenadas armazenadas no banco de dados e as retorna em formato de string para que o godot entenda.

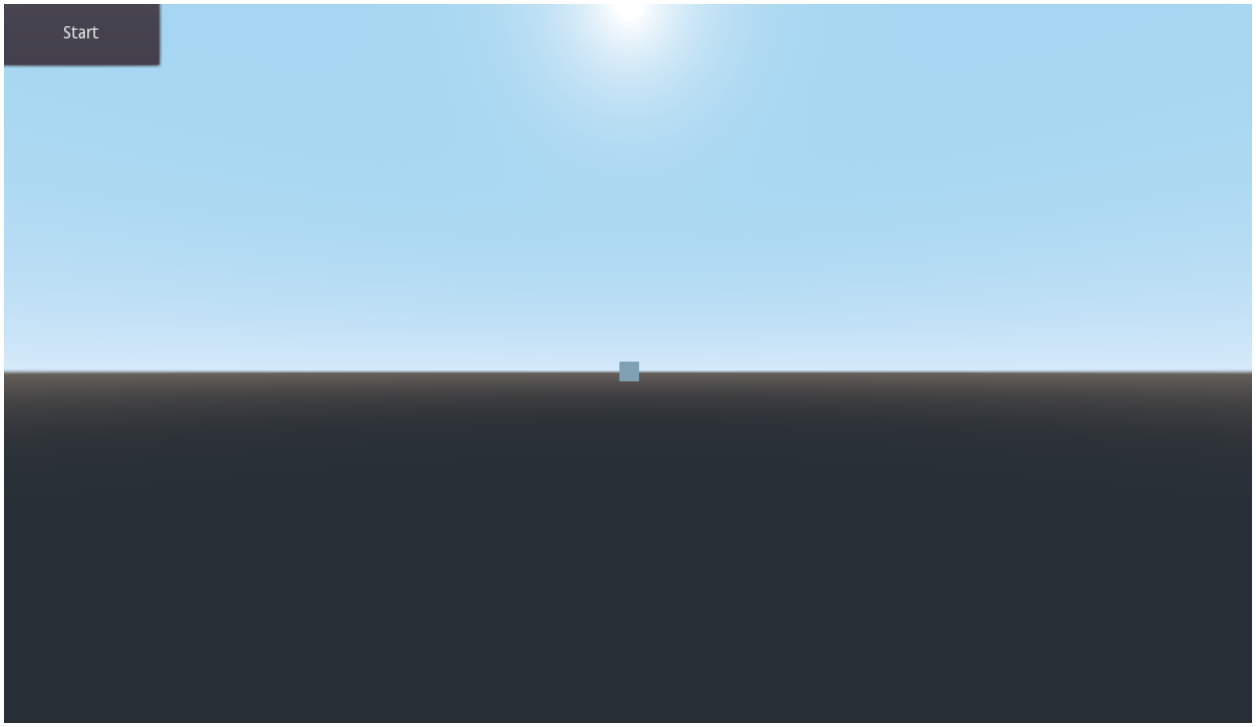
Documentação do Frontend

O frontend foi desenvolvido em html puro, com a estilização em CSS.



Essa é a única tela criada que cumpre o que foi solicitado, o primeiro componente, é o grupo de inputs. Você insere os valores que deseja para depois de clicar no botão de enviar, os valores inseridos nesse componentes são atualizados para o componente do meio, enquanto que os inputs são zerados.

Documentação do Godot



Essa é minha tela no godot. Ao clicar no botão “Start”, será feita a requisição e as coordenadas serão pegas e atualizadas na sua tela.

Esse é um exemplo após clicar no botão:

A cada 5 segundos, sem precisar clicar no botão, uma nova requisição é feita. Logo, se nada mudar no banco de dados, não terá alteração. Todavia, se os dados no banco forem diferentes, no godot os valores de x, y, z e r serão alterados, e o cubo irá se deslocar até sua nova posição.

Entretanto, uma nova alteração foi feita.

Após clicar no botão start, ele “se transforma” em um botão de stop. Afinal, ao clicar no botão de start, ele começa a fazer requisições repetidamente, e se clicar em stop, isso para.
Ou seja, se uma nova informação chegar, não será atualizado. Mas claro, ao clicar no botão de stop, o botão de start surge novamente para começar mais uma vez. O código referente ao godot está comentado.