Enunciado do Projeto – Sistema de Aluguel de Carros

Contexto

A empresa fictícia **EasyCar** deseja informatizar o processo de aluguel de veículos. Atualmente, os registros de clientes, carros e contratos de aluguel são feitos manualmente, o que dificulta o controle e a consulta de informações.

O objetivo deste projeto é desenvolver uma **API REST em Django** que permita gerenciar perfis de clientes, carros e alugueis, com autenticação e controle de acesso baseado em grupos de usuários.

Objetivos

- Implementar um sistema de backend que permita:
    - Usuários do grupo **Funcionários** cadastrarem carros e registrarem alugueis.
    - Usuários do grupo **Clientes** consultarem apenas seus próprios alugueis e os carros que já alugaram, não fazem edição nem inserção de dados.
- Garantir segurança e organização dos dados com autenticação e permissões.
- Disponibilizar documentação clara da API para facilitar o uso por outros sistemas.
- Para criação dos tokens de acesso será utilizado DRF Token Authentication.
- Todos os usuários serão criados na tabela User do Django.
- Todos os grupos serão criados na funcionalidade nativa de grupos do django via interface admin.

Estrutura de Dados

O sistema pode conter **4 tabelas principais (sugestão)**:

1. **User** - modelo padrão do Django para autenticação.
2. **PerfilCliente** - informações adicionais do cliente (CNH, telefone, endereço).
    - Relacionamento **1:1** com User.
3. **Carro** - cadastro de veículos disponíveis para aluguel.
    - Carros são cadastrados apenas por usuários do grupo Funcionários.
4. **Aluguel** - registros de alugueis realizados.
    - Relacionamentos:
        - **N:N** com **Carro** via **Aluguel** (um perfilCliente pode alugar vários carros, e um carro pode ser alugado por vários clientes em momentos diferentes)
        - **1:N** entre User (Funcionário) ↔ Aluguel (um funcionário pode registrar vários alugueis).

Requisitos Mínimos

- **Rotas de backend** implementando métodos **GET, POST, PUT, DELETE**.
- **CRUD completo** para cada tabela.
- **No mínimo 3 rotas de relacionamento**:
    - PerfilCliente ↔ Carro (A-B)
    - Carro ↔ Aluguel (B-C)
    - PerfilCliente ↔ Carro ↔ Aluguel (A-B-C)
- **Relacionamentos obrigatórios**:
    - 1:1 (User ↔ PerfilCliente)
    - 1:N (Funcionário ↔ Aluguel)
    - N:N (PerfilCliente ↔ Carro via Aluguel)
- **Autenticação**:
    - Login via web (Django Auth).
    - **Token Authentication com DRF** (`rest_framework.authtoken`).
- **Controle de acesso por grupos**:
    - Funcionários → CRUD completo em Carro e Aluguel, além de cadastrar PerfilCliente.
    - Clientes → apenas consultas de seus alugueis e carros alugados.
- **Registro no admin**: todas as tabelas e tokens devem estar disponíveis.
- **Documentação da API**: gerada com **drf-spectacular** e disponível em **Redoc**.

Sugestão de modelagem:

Entidade: User - Já existe no Django (não deve ser criada/alterada)

- **Campos principais**:
    - `id` (PK)
    - `username`
    - `email`
    - `password`
    - `is_active`, `is_staff`, `is_superuser`
- **Relacionamentos**:
    - 1:1 com **PerfilCliente** (quando o usuário pertence ao grupo Clientes)
    - 1:N com **Aluguel** (quando o usuário pertence ao grupo Funcionários e registra vários alugueis)

Entidade: PerfilCliente

- **Campos principais**:
    - `id` (PK) - models gera automaticamente
    - `user_id` (FK → User, relacionamento 1:1)
    - `cnh`
    - `telefone`
    - `endereco`
- **Relacionamentos**:

    - N:N com **Carro** via **Aluguel** (um perfilCliente pode alugar vários carros, e um carro pode ser alugado por vários clientes em momentos diferentes)

Entidade: Carro

- **Campos principais**:
    - `id` (PK) - models gera automaticamente
    - `modelo`
    - `placa`
    - `ano`
    - `status` (ex.: disponível, alugado)
- **Observação**:
    - Carros são cadastrados apenas por usuários do grupo Funcionários.

Entidade: Aluguel

- **Campos principais**:
    - `id` (PK) - models gera automaticamente
    - `perfil_cliente_id` (FK → PerfilCliente)
    - `carro_id` (FK → Carro)
    - `funcionario_id` (FK → User, grupo Funcionários) ← **novo campo**
    - `data_inicio`
    - `data_fim`
    - `valor`
- **Relacionamentos**:
    - N:N entre **PerfilCliente** e **Carro**
    - 1:N entre **User (Funcionário)** e **Aluguel** (um funcionário pode registrar vários alugueis)

Sugestão de Rotas da API

Rota principal

- `/api/` → ponto de entrada da API (pode listar recursos disponíveis ou redirecionar para documentação).

Usuários e Perfis

- `/api/users/` → CRUD de usuários (apenas funcionários).
- `/api/perfis-clientes/` → CRUD de perfis de clientes (apenas funcionários).
- `/api/perfis-clientes/{id}/` → detalhes de um perfil específico (apenas funcionários).
- `/api/perfis-clientes/{id}/alugueis/` → listar alugueis de um cliente específico (apenas funcionários).

- `/api/me/alugueis/` → listar **apenas os alugueis do cliente autenticado** (rota exclusiva para clientes).

Carros

- `/api/carros/` → CRUD de carros (funcionários podem cadastrar/editar; clientes podem apenas listar).
- `/api/carros/{id}/` → detalhes de um carro específico.

Alugueis

- `/api/alugueis/` → CRUD de alugueis (funcionários registram).
- `/api/alugueis/{id}/` → detalhes de um aluguel específico.
- `/api/funcionarios/{id}/alugueis/` → listar todos os alugueis registrados por um funcionário.

Autenticação

- `/api/auth/token/` → obtenção de token (DRF Token Authentication).

Documentação

- `/a/api/schema/` → schema OpenAPI gerado pelo **drf-spectacular**.
- `/api/docs/` → documentação interativa.
- `/api/docs/redoc/` → documentação interativa.

Sugestão de melhorias

- Implementação de status do aluguel (ativo, finalizado).
- Implementação de funcionalidade para devolução do veículo e finalização do aluguel.
- Implementação de django-filter, para combinação de filtros.

Entregáveis

Código fonte

- Repositório GitHub organizado
- Modelos, rotas, controllers/services
- Arquivo de dependências
- Scripts de migração (se houver)
- Estrutura de pastas limpa

Documentação

- README com (template: https://github.com/claulis/templateBFD):
    - descrição do sistema
    - criar ambiente
    - instalar pacotes necessários
    - instruções de instalação
    - como rodar o servidor
- Documentação da API (Markdown ou Swagger)
- Modelo de dados (MER + DER)