Documentação de Projeto

para o sistema

GearUp

Versão 1.0

Projeto de sistema elaborado pelos alunos: Diogo Henrique Moreira da Silva,
Gustavo Ignácio Moreira da Silva,
Pedro Marçal Ballesteros,
Marcos Oliveira Antunes
como parte da disciplina **Projeto de Software**.

<07/04/2025>

Tabela de Conteúdo

Histórico de Revisões

Nome	Data	Razões para Mudança	Versão
Diogo Henrique Moreira da Silva	01/05/2025	Criação inicial do repositório e estrutura base do projeto GearUp	1.0
Diogo Henrique Moreira da Silva	03/05/2025	Inclusão dos primeiros diagramas de caso de uso e classe	1.1
Diogo Henrique Moreira da Silva	06/05/2025	Adição dos arquivos de relatórios e modelos de entidade no backend	1.2
Diogo Henrique Moreira da Silva	09/05/2025	Implementação de funcionalidades de agendamento e cadastro de clientes	1.3
Diogo Henrique Moreira da Silva	12/05/2025	Atualização dos diagramas de classe e casos de uso	1.4
Diogo Henrique Moreira da Silva	14/05/2025	Integração de backend e frontend; início do fluxo de login	1.5
Diogo Henrique Moreira da Silva	16/05/2025	Adição dos diagramas de sequência e diagrama de estados	1.6
Diogo Henrique Moreira da Silva	17/05/2025	Revisão final do documento de projeto e inclusão dos modelos de arquitetura e banco de dados	2.0

1. Introdução

O projeto GearUp tem como objetivo facilitar o gerenciamento de serviços automotivos por meio de uma plataforma web intuitiva e eficiente. A aplicação permite que clientes cadastrem seus veículos, agendem serviços com facilidade e acompanhem o andamento de seus atendimentos em tempo real. Já os funcionários também podem realizar cadastros de serviços, além de visualizar os agendamentos, atualizar o status dos serviços e salvar observações nos agendamentos.

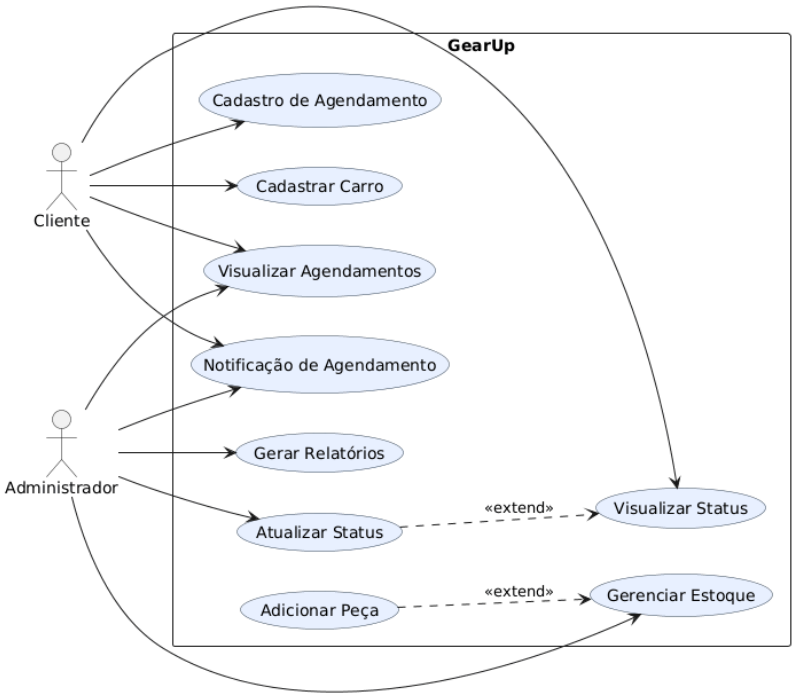
Além disso, o sistema oferece recursos de controle de peças e geração de relatórios mensais para auxiliar na tomada de decisões.

2. Modelos de Usuário e Requisitos

2.1 Descrição de Atores

• Cliente – usuário que agenda serviços, cadastra veículos e acompanha o andamento.
• Funcionário – gerencia agendamentos, estoque de peças e atualiza status.
• Sistema de E-mail – componente externo que envia notificações e confirmações automáticas.

2.2 Modelo de Casos de Uso



UC-01: Cadastrar Carro
UC-02: Cadastro de Agendamento
UC-03: Visualizar Agendamentos
UC-04: Visualizar Status
UC-05: Atualizar Status (extend de UC-04)
UC-06: Gerenciar Estoque
UC-07: Adicionar Peça (extend de UC-06)
UC-08: Gerar Relatórios
UC-09: Notificação de Agendamento (automatizado)
UC-12: Administrador gerencia pedidos
Código (PlantUML):

```
@startuml

left to right direction

skinparam usecase {

  BackgroundColor #e8f0ff

  BorderColor #345

}


actor "Cliente" as Cliente

actor "Administrador" as Admin


rectangle "GearUp" {

  usecase "Gerar Relatórios" as UC_Relatorios

  usecase "Gerenciar Estoque" as UC_GerenciarEstoque

  usecase "Adicionar Peça" as UC_AdicionarPeca

  usecase "Visualizar Agendamentos" as UC_VisualizarAg

  usecase "Notificação de Agendamento" as UC_Notificacao

  usecase "Cadastro de Agendamento" as UC_CadastrarAg
```

```
  usecase "Visualizar Status" as UC_VisualizarStatus
  usecase "Atualizar Status" as UC_AtualizarStatus
  usecase "Cadastrar Carro" as UC_CadastrarCarro
}

' ligações de atores (seguindo seu desenho)
Cliente --> UC_VisualizarAg
Cliente --> UC_Notificacao
Cliente --> UC_CadastrarAg
Cliente --> UC_VisualizarStatus
Cliente --> UC_CadastrarCarro

Admin --> UC_Relatorios
Admin --> UC_GerenciarEstoque
Admin --> UC_VisualizarAg
Admin --> UC_Notificacao
Admin --> UC_AtualizarStatus

' relações <<extend>> conforme o rótulo do seu desenho
UC_AdicionarPeca ..> UC_GerenciarEstoque : <<extend>>
UC_AtualizarStatus ..> UC_VisualizarStatus : <<extend>>
@enduml
```

2.3 Diagrama de Sequência do Sistema

Contrato	UC-01: Cliente realiza login
Operação	RealizarLogin (email, senha)
Referências cruzadas	UC-01 – Cliente realiza login
Pré-condições	• Usuário possui e-mail cadastrado e senha válida. • Sistema disponível.
Pós-condições	• Uma sessão/autenticação é criada para o usuário. • O sistema registra o login (data/hora/IP). • Usuário é redirecionado para a área autenticada. • Em caso de falha, mensagem "Credenciais inválidas" é exibida.
Contrato	UC-02: Cadastro de Agendamento
Operação	SolicitarAgendamento (dto)
Referências cruzadas	UC-02 – Cadastro de Agendamento, UC-09 – Notificação de Agendamento
Pré-condições	• Cliente autenticado. • Cliente possui ao menos um carro cadastrado. • Horário solicitado não está reservado.
Pós-condições	• Agendamento é criado com `status = AGUARDANDO_INICIO`. • Dados persistidos (cliente, carro, tipo de serviço, data/hora e observação). • Confirmação apresentada ao cliente. • Notificação de confirmação disparada (e-mail).
Contrato	UC-03: Visualizar Agendamentos
Operação	ListarAgendamentos (filtros)
Referências cruzadas	UC-03 – Visualizar Agendamentos
Pré-condições	• Usuário autenticado (Cliente ou Administrador).
Pós-condições	• Lista de agendamentos é exibida conforme filtros (período/status). • Nenhuma alteração de estado do sistema.

Contrato	UC-04: Visualizar Status
Operação	ConsultarStatus (agendamentoId)
Referências cruzadas	UC-04 – Visualizar Status
Pré-condições	• Agendamento existe e pertence ao cliente (ou perfil tem permissão).
Pós-condições	• Status atual e histórico são exibidos. • Nenhuma alteração de estado do sistema.
Contrato	UC-05: Atualizar Status
Operação	AtualizarStatus (agendamentoId, novoStatus)
Referências cruzadas	UC-05 – Atualizar Status (extend de UC-04), UC-09 – Notificação de Agendamento
Pré-condições	• Administrador/Funcionário autenticado e autorizado. • Agendamento existe. • Transição de status é válida (regras de negócio).
Pós-condições	• Status do agendamento é atualizado (ex.: EM_ANDAMENTO → CONCLUIDO ou CANCELADO). • Registro de auditoria é criado (quem/quando). • Cliente pode ser notificado sobre a atualização.
Contrato	UC-06: Cancelar Agendamento
Operação	CancelarAgendamento (agendamentoId)
Referências cruzadas	parte do fluxo de UC-02/UC-04
Pré-condições	• Cliente autenticado e proprietário do agendamento (ou Administrador). • Agendamento ainda não concluído.
Pós-condições	• Agendamento marcado como `CANCELADO`. • Sistema apresenta confirmação de cancelamento. • (Opcional) Notificação enviada ao cliente.
Contrato	UC-07: Gerenciar Estoque
Operação	AtualizarEstoque (pecaId, quantidade, preco, localizacao, estoqueMinimo)
Referências cruzadas	UC-06 – Gerenciar Estoque
Pré-condições	• Administrador autenticado. • Peça existente no catálogo.

Pós-condições	• Quantidade/preço/localização/estoque mínimo atualizados. • Registro de alteração (auditoria) gravado
Contrato	UC-08: Adicionar Peça
Operação	AdicionarPeca (dadosPeca)
Referências cruzadas	UC-07 – Adicionar Peça (extend de Gerenciar Estoque)
Pré-condições	• Administrador autenticado. • Código da peça ainda não cadastrado.
Pós-condições	• Nova peça é incluída no catálogo e estoque. • Sistema confirma inclusão; fica disponível para consultas e relatórios.
Contrato	UC-09: Gerar Relatórios
Operação	GerarRelatorio (tipo, periodo, filtros)
Referências cruzadas	UC-08 – Gerar Relatórios
Pré-condições	• Administrador autenticado.
Pós-condições	• Relatório é produzido (tela/PDF/CSV). • Nenhuma alteração de estado do domínio; apenas leitura/compilação.

3. Modelos de Projeto

3.1 Arquitetura

O GearUp adota o modelo cliente-servidor em três camadas, com troca de dados via REST/JSON sobre HTTPS:

1. **Frontend (Camada de Apresentação)**
    - Tecnologias: HTML, CSS e JavaScript (site estático).
    - Responsabilidades: interface com o usuário; formulários de login, cadastro de carro e criação/consulta de agendamentos; chamadas REST ao backend; feedback de validação e estados de carregamento.
2. **Backend (Camada de Lógica/Serviços)**

    - Tecnologias: Spring Boot 3.3.4 (Java 17), arquitetura MVC + Services + Repositories.
    - Responsabilidades: autenticação/autorização (JWT), regras de negócio (criar/editar/cancelar agendamentos; atualização de status; gestão de peças/estoque; geração de relatórios), orquestração de persistência e envio de notificações via SMTP.
3. **Banco de Dados (Camada de Persistência)**
    - Tecnologia: PostgreSQL hospedado na Railway Cloud.
    - Responsabilidades: armazenamento de usuários, clientes, funcionários, carros, agendamentos, peças/estoque e relatórios, com integridade relacional e transações ACID.

Comunicação e integrações

- **Cliente** → Backend: requisições REST via HTTPS (JSON).
- **Backend** → Banco: JDBC/TCP-IP.
- **Backend** → E-mail: SMTP (ex.: Gmail) para confirmações e lembretes.

Segurança

- Tráfego externo cifrado (TLS/HTTPS).
- JWT para sessões, políticas de autorização por perfil (Cliente/Administrador).
- CORS configurado no backend; hashing de senhas; validação de entrada.

Escalabilidade & Manutenibilidade

- Camadas desacopladas: o frontend pode ser servido por CDN/estático, o backend escala horizontalmente (stateless) e o banco em instância gerenciada.
- Separação clara de Controllers → Services → Repositories → Entities facilita testes e evolução.

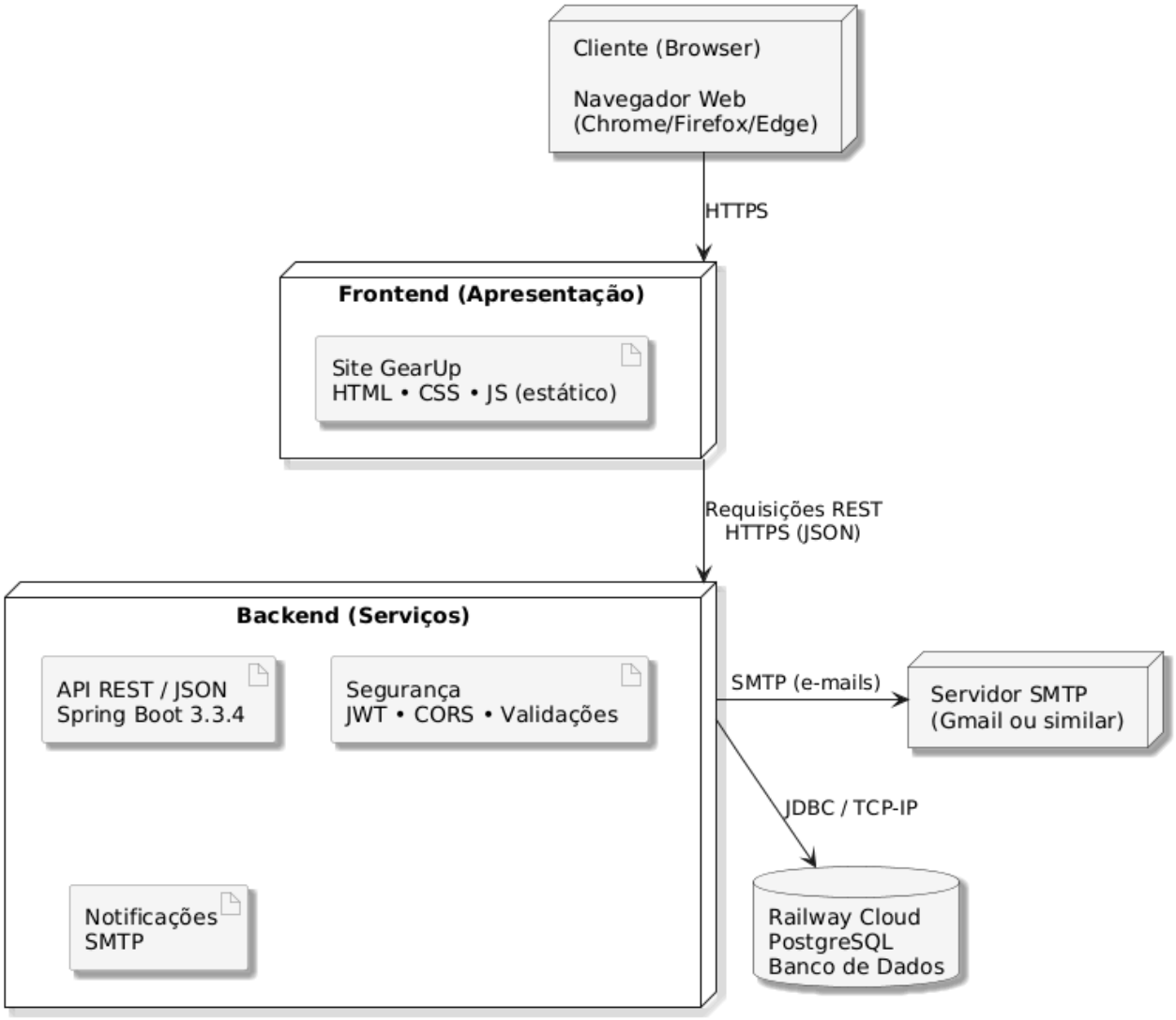
Observabilidade & Operação

- Logs estruturados e correlação por requestId; métricas de saúde do Spring Boot (actuator) opcionais; mensagens de erro padronizadas para o frontend.

Fluxo resumido

1. O usuário acessa o site estático (HTTPS).
2. O frontend envia REST ao Spring Boot para autenticar, cadastrar carro, criar/consultar agendamentos etc.
3. O backend persiste dados no PostgreSQL (Railway) e, quando aplicável, dispara e-mails de confirmação/lembrete via SMTP.

Essa arquitetura garante separação de responsabilidades, segurança, escalabilidade e facilidade de manutenção.



Código:

```
@startuml

' --- estilo geral ---

skinparam shadowing true

skinparam rectangle {

  BorderColor #333

  FontColor #111

  FontSize 14
```

```
}
skinparam artifact {
  BackgroundColor #f5f5f5
  BorderColor #888
}
skinparam database {
  BackgroundColor #f5f5f5
}

node "Cliente (Browser)\n\nNavegador Web\n(Chrome/Firefox/Edge)" as Browser

node "Frontend (Apresentação)" as FE {
  artifact "Site GearUp\nHTML • CSS • JS (estático)" as Site
}

node "Backend (Serviços)" as BE {
  artifact "API REST / JSON\nSpring Boot 3.3.4" as Api
  artifact "Segurança\nJWT • CORS • Validações" as Sec
  artifact "Notificações\nSMTP" as MailMod
}

database "Railway Cloud\nPostgreSQL\nBanco de Dados" as DB
node "Servidor SMTP\n(Gmail ou similar)" as SMTP
```

Browser -down-> FE : HTTPS

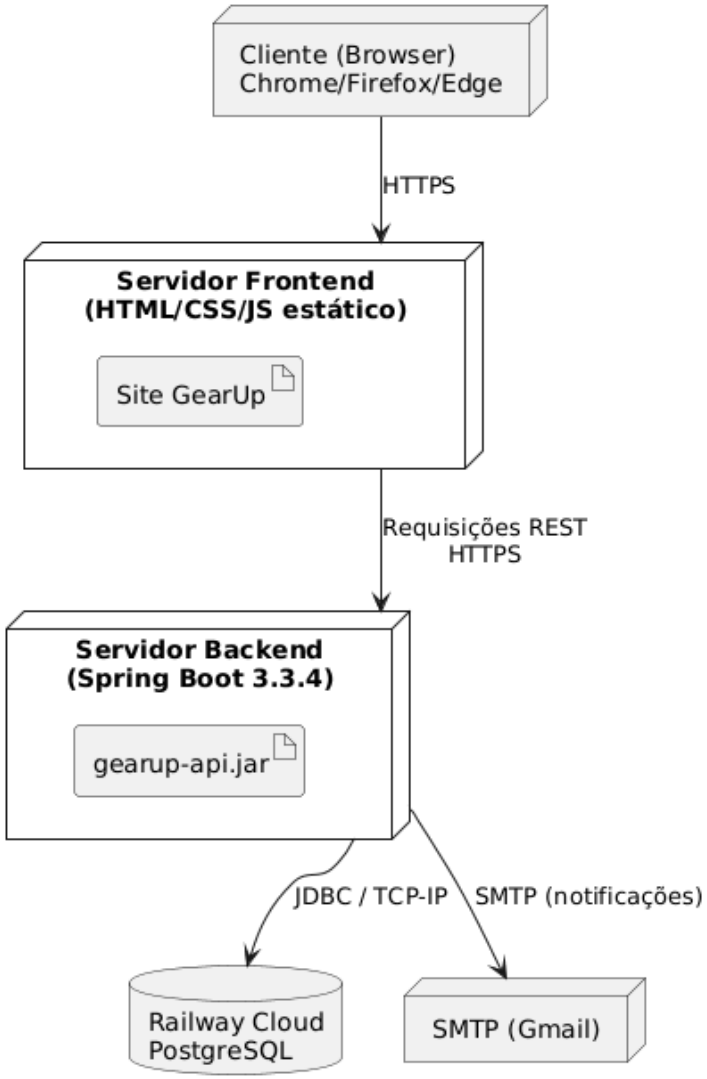
FE -down-> BE : Requisições REST\nHTTPS (JSON)

BE -down-> DB : JDBC / TCP-IP

BE -right-> SMTP : SMTP (e-mails)

@enduml

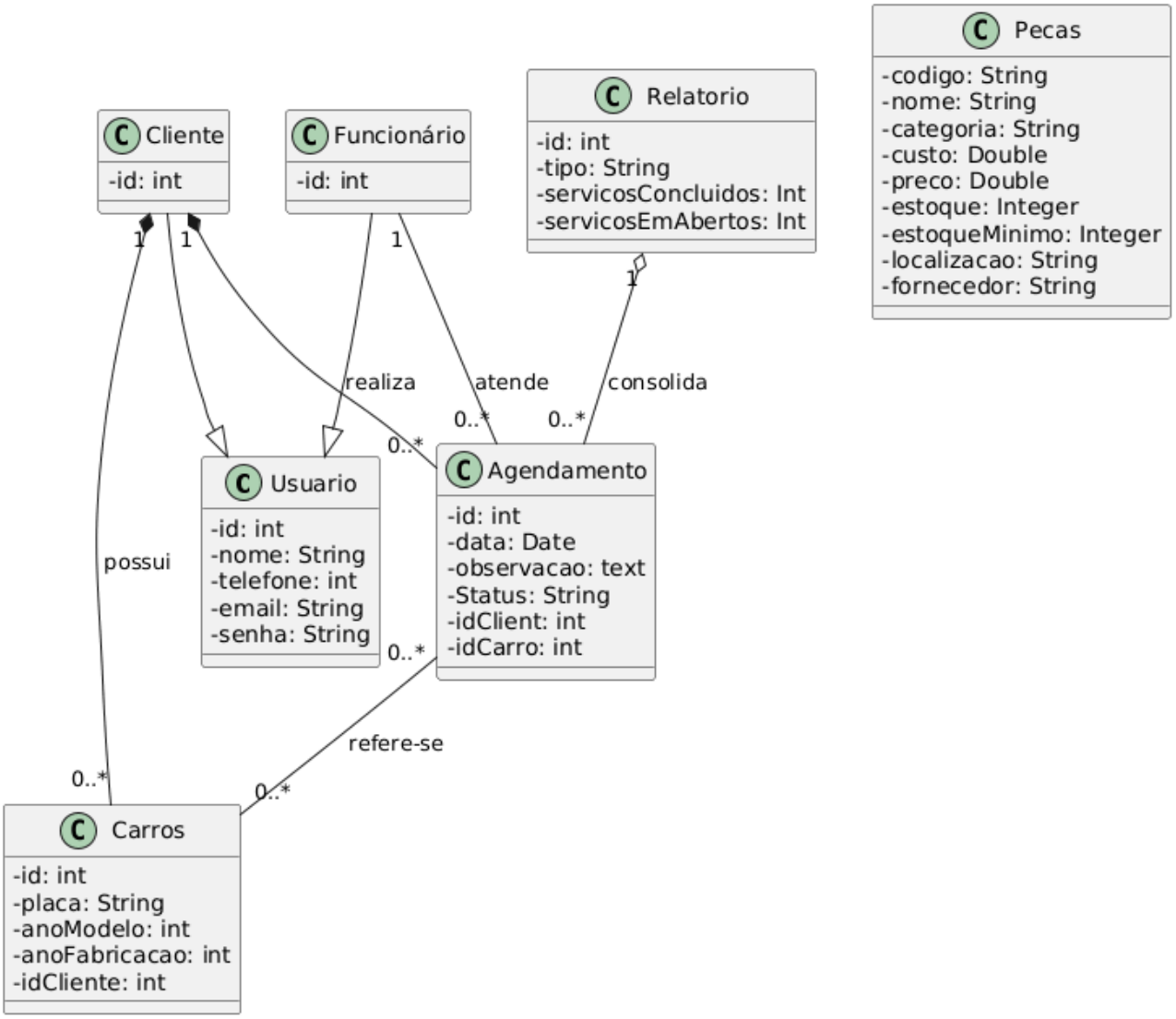
3.2 Diagrama de Componentes e Implantação.



Código:

```
@startuml
skinparam nodeStyle rectangle
node "Cliente (Browser)\nChrome/Firefox/Edge" as Browser

node "Servidor Frontend\n(HTML/CSS/JS estático)" as Front {
  artifact "Site GearUp" as site
}

node "Servidor Backend\n(Spring Boot 3.3.4)" as Back {
  artifact "gearup-api.jar" as jar
}

database "Railway Cloud\nPostgreSQL" as DB
node "SMTP (Gmail)" as SMTP

Browser --> Front : HTTPS
Front --> Back : Requisições REST\nHTTPS
Back --> DB : JDBC / TCP-IP
Back --> SMTP : SMTP (notificações)
@enduml
```

3.3 Diagrama de Classes



Código:

```
@startuml
skinparam classAttributeIconSize 0

' ===== Classes e atributos =====
class "Usuario" as Usuario {
  - id: int
  - nome: String
  - telefone: int
  - email: String
  - senha: String
}
```

```
class "Cliente" as Cliente {
  - id: int
}

class "Funcionário" as Funcionario {
  - id: int
}

class "Carros" as Carro {
  - id: int
  - placa: String
  - anoModelo: int
  - anoFabricacao: int
  - idCliente: int
}

class "Agendamento" as Agendamento {
  - id: int
  - data: Date
  - observacao: text
  - Status: String
  - idClient: int
  - idCarro: int
}

class "Relatorio" as Relatorio {
  - id: int
  - tipo: String
  - servicosConcluidos: Int
  - servicosEmAbertos: Int
}

class "Pecas" as Pecas {
  - codigo: String
  - nome: String
  - categoria: String
  - custo: Double
  - preco: Double
  - estoque: Integer
  - estoqueMinimo: Integer
  - localizacao: String
  - fornecedor: String
}

' ===== Generalizações (herança) =====
Cliente --|> Usuario
```

Funcionario --|> Usuario

' ===== Relacionamentos (conforme cardinalidades e diamantes da imagem) =====
Cliente "1" *-- "0..*" Carro : possui
Cliente "1" *-- "0..*" Agendamento : realiza

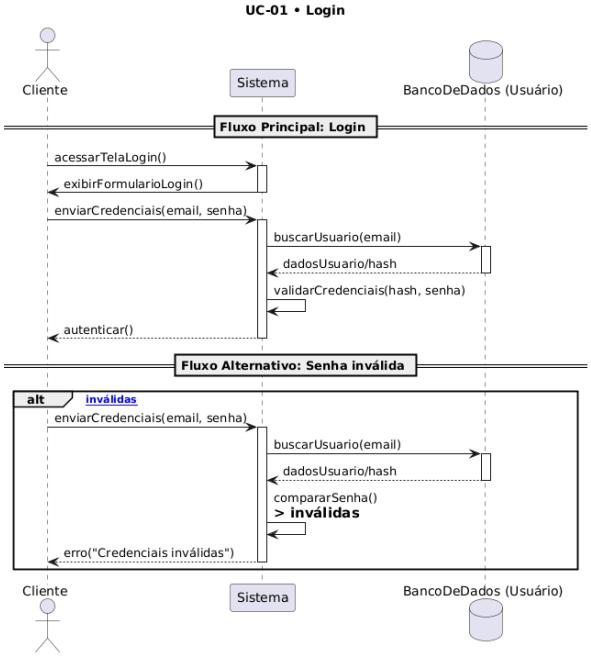
Funcionario "1" -- "0..*" Agendamento : atende

Relatorio "1" o-- "0..*" Agendamento : consolida

Agendamento "0..*" -- "0..*" Carro : refere-se
' (A classe Pecas aparece isolada no seu desenho; mantenho sem ligação)
@enduml

3.4 Diagramas de Sequência

UC-01 – Cliente realiza login:



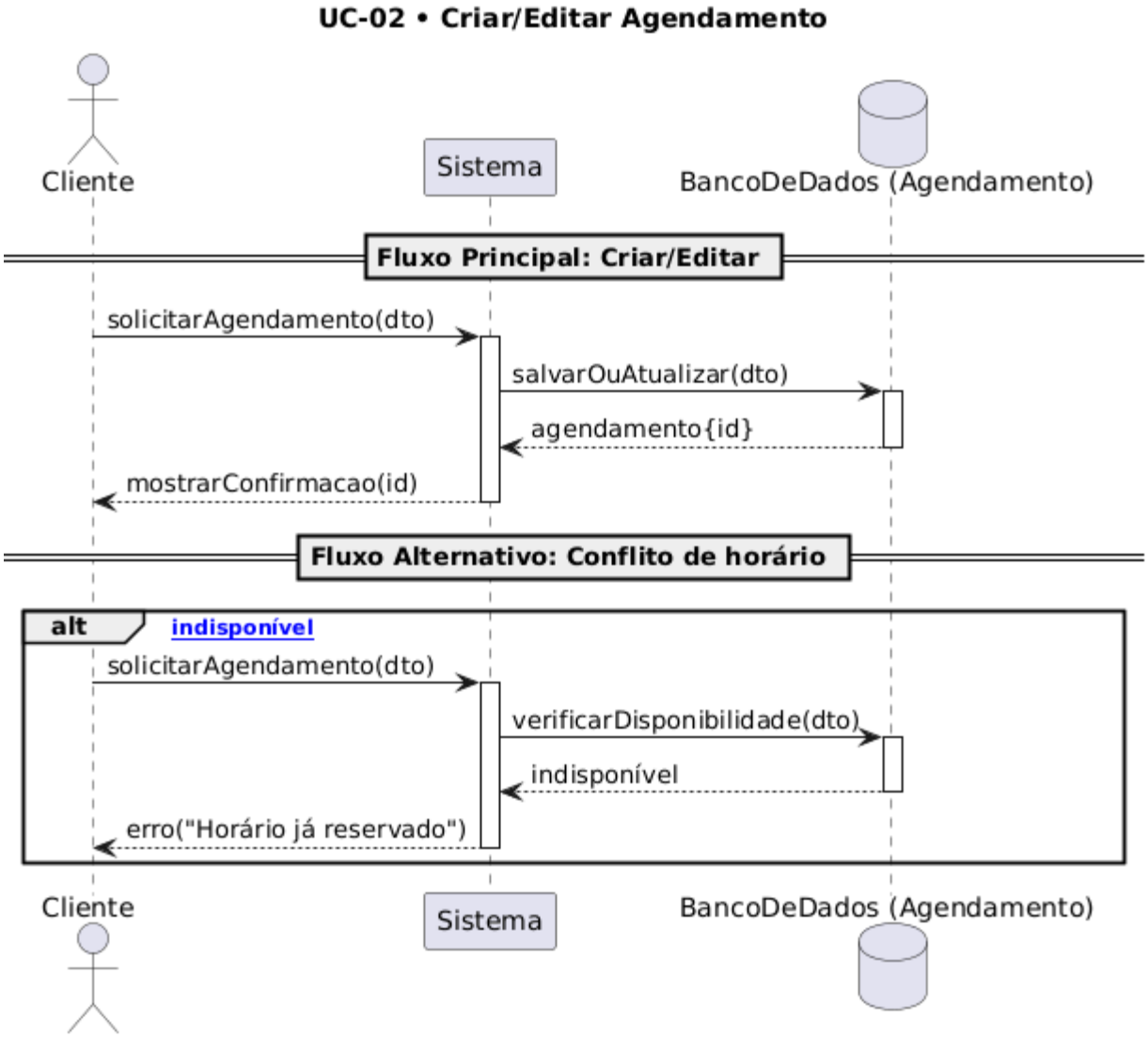
Código:
@startuml
title UC-01 • Login

actor Cliente
participant Sistema
database "BancoDeDados (Usuário)" as DBU

== Fluxo Principal: Login ==
Cliente -> Sistema : acessarTelaLogin()
activate Sistema

```
Sistema -> Cliente : exibirFormularioLogin()
deactivate Sistema

Cliente -> Sistema : enviarCredenciais(email, senha)
activate Sistema
Sistema -> DBU : buscarUsuario(email)
activate DBU
DBU --> Sistema : dadosUsuario/hash
deactivate DBU

Sistema -> Sistema : validarCredenciais(hash, senha)
Sistema --> Cliente : autenticar()
deactivate Sistema

== Fluxo Alternativo: Senha inválida ==
alt [Credenciais inválidas]
  Cliente -> Sistema : enviarCredenciais(email, senha)
  activate Sistema
  Sistema -> DBU : buscarUsuario(email)
  activate DBU
  DBU --> Sistema : dadosUsuario/hash
  deactivate DBU

  Sistema -> Sistema : compararSenha() \n==> inválidas
  Sistema --> Cliente : erro("Credenciais inválidas")
  deactivate Sistema
end
@enduml
```

UC-04 – Cliente cria/edita agendamento (gerenciar):



Código:

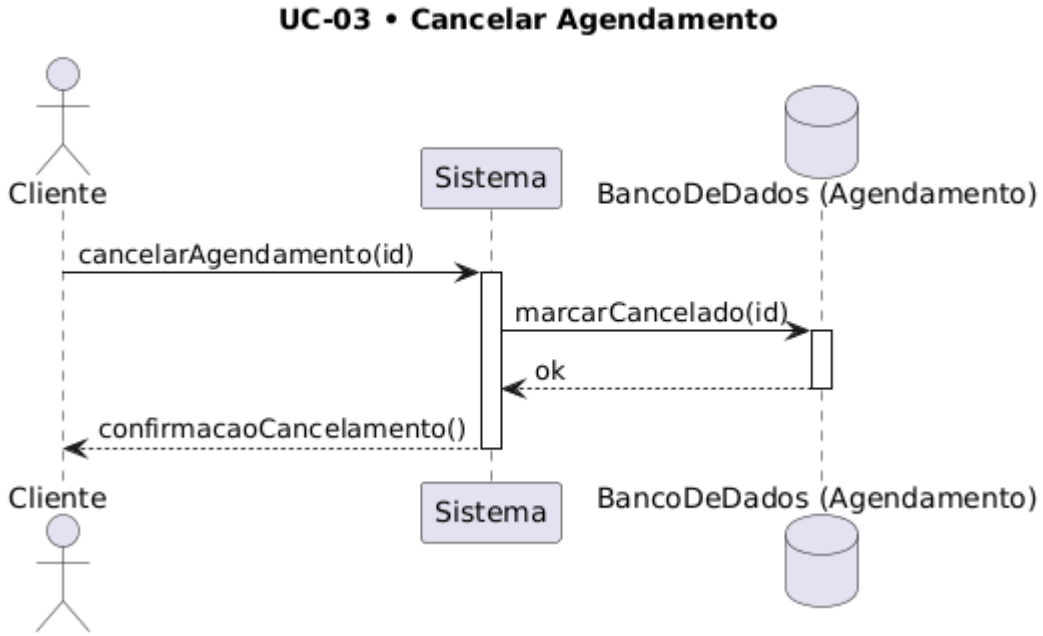
```
@startuml
title UC-02 • Criar/Editar Agendamento

actor Cliente
participant Sistema
database "BancoDeDados (Agendamento)" as DBA

== Fluxo Principal: Criar/Editar ==
Cliente -> Sistema : solicitarAgendamento(dto)
activate Sistema

Sistema -> DBA : salvarOuAtualizar(dto)
activate DBA
DBA --> Sistema : agendamento{id}
deactivate DBA
```

Sistema --> Cliente : mostrarConfirmacao(id)
deactivate Sistema

== Fluxo Alternativo: Conflito de horário ==
alt [Horário indisponível]
  Cliente -> Sistema : solicitarAgendamento(dto)
  activate Sistema
  Sistema -> DBA : verificarDisponibilidade(dto)
  activate DBA
  DBA --> Sistema : indisponível
  deactivate DBA

  Sistema --> Cliente : erro("Horário já reservado")
  deactivate Sistema
end
@enduml

UC-07 – Cliente cancela agendamento:



Código:
@startuml
title UC-03 • Cancelar Agendamento

actor Cliente
participant Sistema
database "BancoDeDados (Agendamento)" as DBA

Cliente -> Sistema : cancelarAgendamento(id)

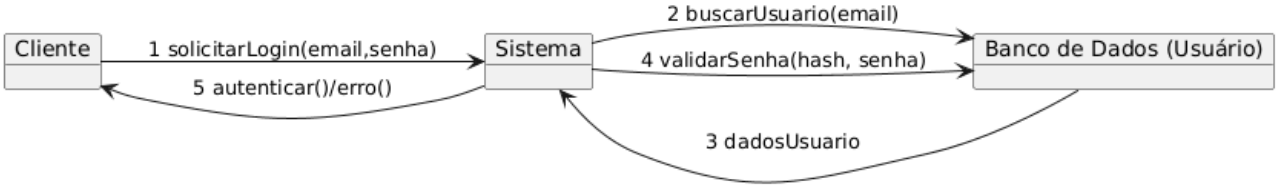
activate Sistema

Sistema -> DBA : marcarCancelado(id)
activate DBA
DBA --> Sistema : ok
deactivate DBA

Sistema --> Cliente : confirmacaoCancelamento()
deactivate Sistema
@enduml

3.5 Diagramas de Comunicação

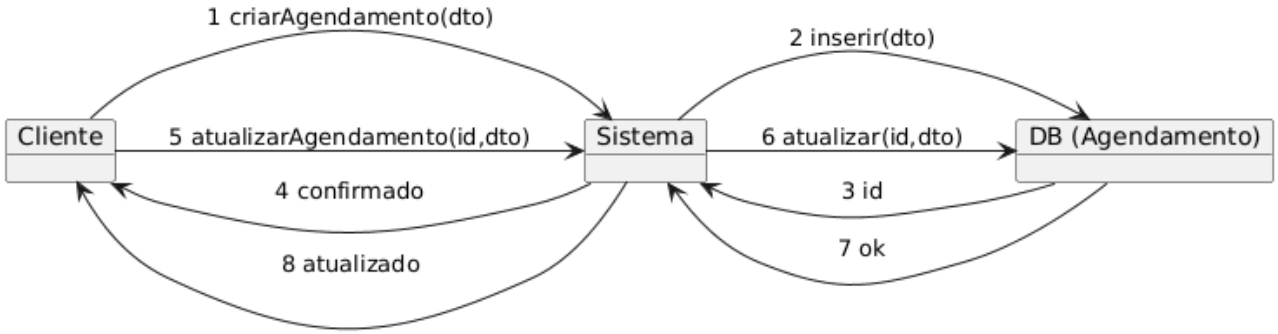
UC-01 – Login:



@startuml
left to right direction
object Cliente
object Sistema
object "Banco de Dados (Usuário)" as DB

Cliente --> Sistema : 1 solicitarLogin(email,senha)
Sistema --> DB : 2 buscarUsuario(email)
DB --> Sistema : 3 dadosUsuario
Sistema --> DB : 4 validarSenha(hash, senha)
Sistema --> Cliente : 5 autenticar()/erro()
@enduml

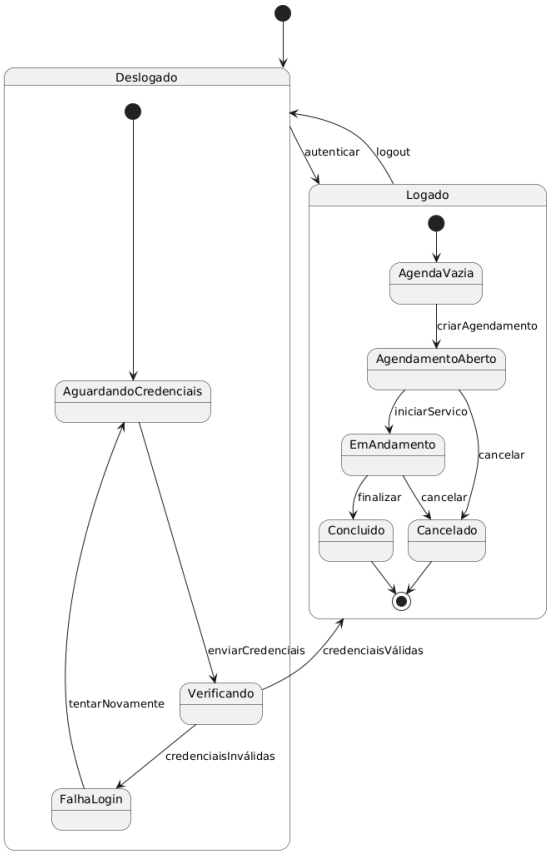
UC-04 – Gerenciar Agendamento:



Código:
@startuml
left to right direction
object Cliente

```
object Sistema
object "DB (Agendamento)" as DBA

Cliente --> Sistema : 1 criarAgendamento(dto)
Sistema --> DBA : 2 inserir(dto)
DBA --> Sistema : 3 id
Sistema --> Cliente : 4 confirmado

Cliente --> Sistema : 5 atualizarAgendamento(id,dto)
Sistema --> DBA : 6 atualizar(id,dto)
DBA --> Sistema : 7 ok
Sistema --> Cliente : 8 atualizado
@enduml
```

3.6 Diagramas de Estados

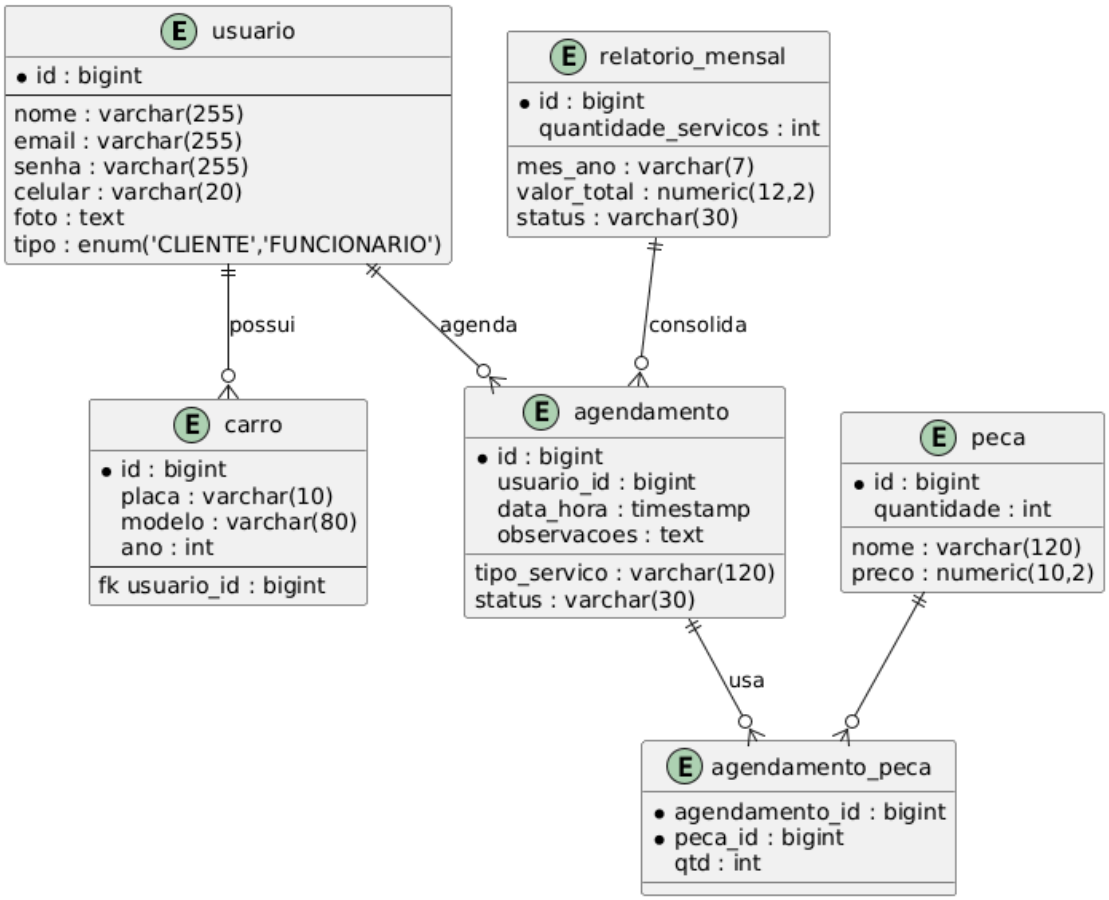


Código:

```
@startuml
[*] --> Deslogado

state Deslogado {
  [*] --> AguardandoCredenciais
  AguardandoCredenciais --> Verificando : enviarCredenciais
```

```
  Verificando --> Logado : credenciaisVálidas
  Verificando --> FalhaLogin : credenciaisInválidas
  FalhaLogin --> AguardandoCredenciais : tentarNovamente
}

state Logado {
  [*] --> AgendaVazia
  AgendaVazia --> AgendamentoAberto : criarAgendamento
  AgendamentoAberto --> EmAndamento : iniciarServico
  EmAndamento --> Concluido : finalizar
  AgendamentoAberto --> Cancelado : cancelar
  EmAndamento --> Cancelado : cancelar
  Concluido --> [*]
  Cancelado --> [*]
}

Deslogado --> Logado : autenticar
Logado --> Deslogado : logout
@enduml
```

4. Modelos de Dados



```
@startuml
!define cfoot
' (IE/crow's-foot style)
entity "usuario" as usuario {
  * id : bigint
  --
  nome : varchar(255)
  email : varchar(255)
  senha : varchar(255)
  celular : varchar(20)
  foto : text
  tipo : enum('CLIENTE','FUNCIONARIO')
}

entity "carro" as carro {
  * id : bigint
  placa : varchar(10)
  modelo : varchar(80)
  ano : int
  --
  fk usuario_id : bigint
}

entity "agendamento" as ag {
  * id : bigint
  usuario_id : bigint
  data_hora : timestamp
  tipo_servico : varchar(120)
  observacoes : text
  status : varchar(30)
}

entity "peca" as peca {
  * id : bigint
  nome : varchar(120)
  quantidade : int
  preco : numeric(10,2)
}

entity "agendamento_peca" as ap {
  * agendamento_id : bigint
  * peca_id : bigint
  qtd : int
}

entity "relatorio_mensal" as rel {
```

```
  * id : bigint
  mes_ano : varchar(7)
  quantidade_servicos : int
  valor_total : numeric(12,2)
  status : varchar(30)
}

usuario ||--o{ carro : possui
usuario ||--o{ ag : agenda
ag ||--o{ ap : usa
peca ||--o{ ap
rel ||--o{ ag : consolida
@enduml
```