



Projeto Banco de Dados



Apresentado por: **Matheus Camargo**
& Pedro Guaita

Objetivo geral

Desenvolver uma plataforma digital que conecte clientes a prestadores de serviços domésticos, oferecendo uma experiência segura, personalizada e eficiente para contratação, gestão e avaliação de serviços diversos.



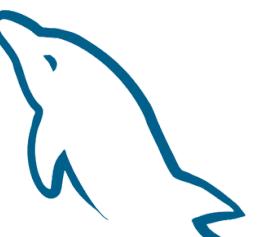
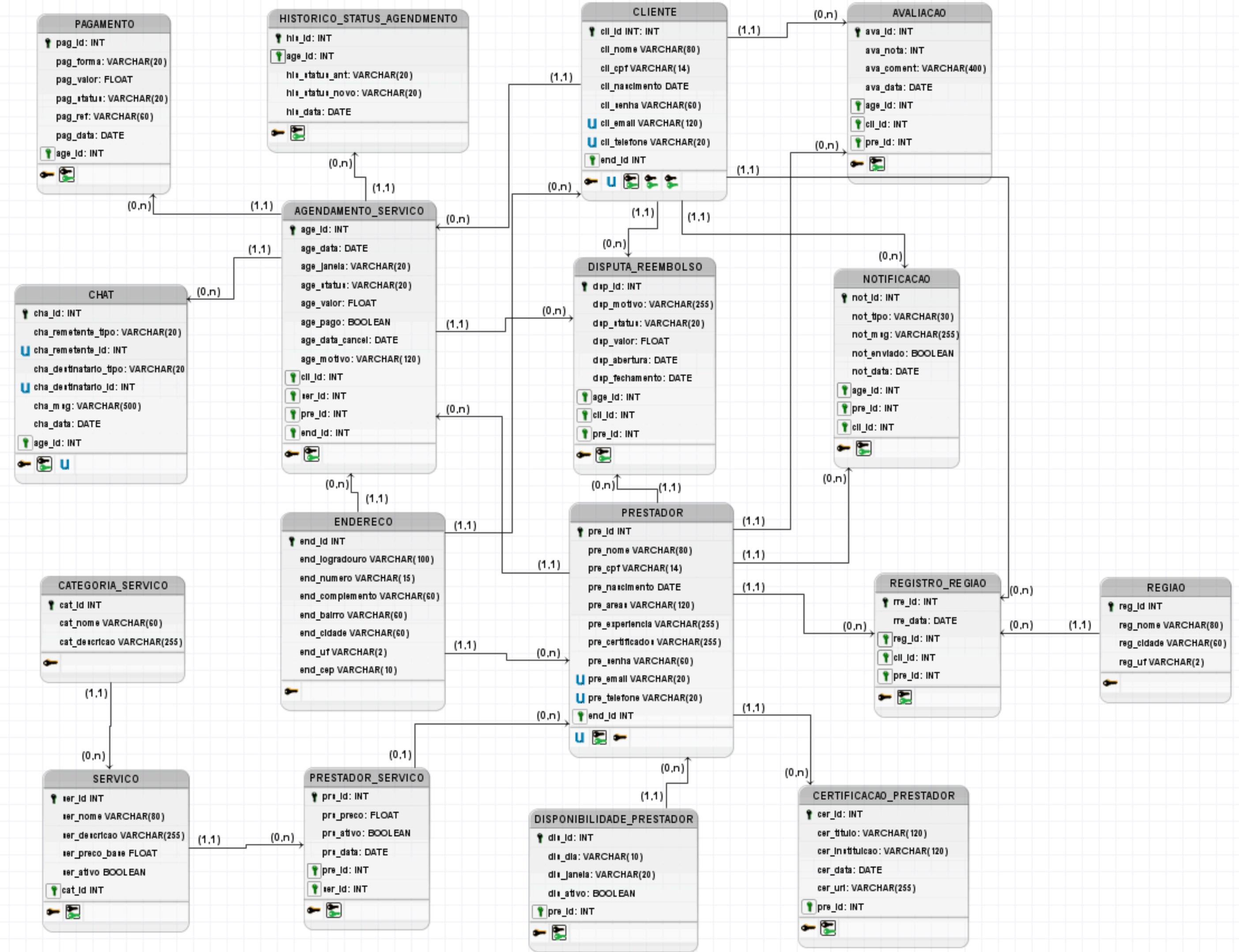
HomeHero



Banco de Dados



MySQL®



MySQL®

Views

12

- Listar categorias de serviços
- Listar serviços cadastrados com suas categorias
- Histórico de agendamentos
- Resumo de agendamentos por período (manhã/tarde/noite)
- Resumo de agendamentos por região do cliente
- Resumo de pagamentos por status
- Média de avaliação por prestador
- Listar disputas abertas
- Listar disponibilidade de prestadores
- Resumo de receita por prestador
- Resumo de receita por período

Exemplos de View

Listar categorias

```
CREATE VIEW view_lista_categorias AS
SELECT
    cat_id,
    cat_nome,
    cat_descricao
FROM categoria_servico
ORDER BY cat_nome;
```

Resultado

```
--- Registro 1 ---
cat_id: 4
cat_nome: Ar Condicionado
cat_descricao: Instalação e manutenção de ar condicionado

--- Registro 2 ---
cat_id: 2
cat_nome: Marcenaria
cat_descricao: Móveis sob medida e reparos em madeira

--- Registro 3 ---
cat_id: 1
cat_nome: Pintura
cat_descricao: Serviços de pintura residencial e comercial

--- Registro 4 ---
cat_id: 3
cat_nome: Piscina
cat_descricao: Limpeza e manutenção de piscinas

--- Registro 5 ---
cat_id: 5
cat_nome: Segurança
cat_descricao: Instalação de sistemas de segurança
```



Exemplos de View

Listar serviços

```
CREATE VIEW view_lista_servicos AS
SELECT
    servico.ser_id,
    servico.ser_nome,
    servico.ser_descricao,
    servico.ser_preco_base,
    servico.ser_ativo,
    categoria_servico.cat_id,
    categoria_servico.cat_nome,
    categoria_servico.cat_descricao
FROM servico
JOIN categoria_servico USING (cat_id)
ORDER BY servico.ser_nome;
```

Resultado

```
--- Registro 1 ---
ser_id: 5
ser_nome: Câmeras de Segurança
ser_descricao: Instalação de sistema de câmeras
ser_preco_base: 600.0
ser_ativo: 1
cat_id: 5
cat_nome: Segurança
cat_descricao: Instalação de sistemas de segurança

--- Registro 2 ---
ser_id: 4
ser_nome: Instalação Split
ser_descricao: Instalação de ar condicionado split
ser_preco_base: 400.0
ser_ativo: 1
cat_id: 4
cat_nome: Ar Condicionado
cat_descricao: Instalação e manutenção de ar condicionado
```



Procedures

12

- Inserir cliente
- Inserir prestador
- Buscar dados pessoais do cliente (por ID)
- Listar agendamentos de um cliente (por ID)
- Buscar dados pessoais do prestador (por ID)
- Inserir novo agendamento de serviço
- Cancelar agendamento de serviço
- Listar agendamentos por período e status
- Registrar avaliação de prestador
- Abrir disputa de reembolso
- Fechar disputa de reembolso
- Inserir pagamento de agendamento

Exemplos de Procedures

Inserir Clientes

```
CREATE PROCEDURE inserir_cliente(
    IN p_nome          VARCHAR(80),
    IN p_cpf           VARCHAR(14),
    IN p_nasc          DATE,
    IN p_senha         VARCHAR(60),
    IN p_end_logradouro VARCHAR(100),
    IN p_end_numero    VARCHAR(15),
    IN p_end_complemento VARCHAR(60),
    IN p_end_bairro    VARCHAR(60),
    IN p_end_cidade    VARCHAR(60),
    IN p_end_uf        VARCHAR(2),
    IN p_end_cep       VARCHAR(10),
    IN p_email         VARCHAR(120),
    IN p_telefone      VARCHAR(20)
)
BEGIN
    INSERT INTO endereco (
        end_logradouro,
        end_numero,
        end_complemento,
        end_bairro,
        end_cidade,
        end_uf,
        end_cep
    ) VALUES (
        p_end_logradouro,
        p_end_numero,
        p_end_complemento,
        p_end_bairro,
        p_end_cidade,
        p_end_uf,
        p_end_cep
    );

```

Resultado

Exemplos de Procedures

Dados cliente

```
CREATE PROCEDURE buscar_dados_pessoais_cliente(
    IN p_cli_id INT
)
BEGIN
    SELECT
        cliente.cli_id,
        cliente.cli_nome,
        cliente.cli_cpf,
        cliente.cli_nascimento,
        cliente.cli_email,
        cliente.cli_telefone,
        cliente.end_id,
        endereco.end_logradouro,
        endereco.end_numero,
        endereco.end_complemento,
        endereco.end_bairro,
        endereco.end_cidade,
        endereco.end_uf,
        endereco.end_cep
    FROM cliente
    JOIN endereco USING (end_id)
    WHERE cliente.cli_id = p_cli_id;
END $$
```

Resultado



Triggers

6

- Registrar status inicial do agendamento no histórico
- Registrar mudança de status do agendamento no histórico
- Criar notificação após nova avaliação de prestador
- Criar notificação após pagamento confirmado
- Criar notificação de novo agendamento para o prestador
- Criar notificações para cliente e prestador ao abrir disputa



Exemplos de Trigger

Status inicial Agendamento

```
CREATE TRIGGER trigger_pos_inserir_agendamento_registrar_status_inicial
AFTER INSERT ON agendamento_servico
FOR EACH ROW
BEGIN
    INSERT INTO historico_status_agendamento
        (age_id, his_status_ant, his_status_novo, his_data)
    VALUES
        (NEW.age_id, 'Criado', NEW.age_status, CURDATE());
END $$
```

Resultado



Exemplos de Trigger

Mudança de status

```
CREATE TRIGGER trigger_pos_atualizar_agendamento_registrar_mudanca_de_status
AFTER UPDATE ON agendamento_servico
FOR EACH ROW
BEGIN
    IF OLD.age_status != NEW.age_status THEN
        INSERT INTO historico_status_agendamento
        (age_id, his_status_ant, his_status_novo, his_data)
        VALUES
        (NEW.age_id, OLD.age_status, NEW.age_status, CURDATE());
    END IF;
END $$
```

Resultado



Spring Boot

Java



Contexto

O **Spring** nasceu no início dos anos 2000 como uma alternativa ao Java EE, que na época era muito burocrático e difícil de configurar. Ele trouxe uma abordagem mais **leve** e **flexível**, permitindo que desenvolvedores criassem aplicações robustas sem tanta complexidade



Diferenciais

- **Autoconfiguração:** Você não precisa configurar tudo manualmente. O Spring Boot detecta automaticamente as dependências e as configura para você.
- **Servidor embutido:** Diferente do Java EE tradicional, onde você precisava de um servidor como Tomcat ou WildFly, o Spring Boot já traz um servidor embutido, como o Tomcat, Jetty ou Undertow. Isso significa que você pode rodar sua aplicação como um simples arquivo JAR.
- **Starter Packs:** São pacotes de dependências prontos para facilitar a integração com bancos de dados, segurança, mensageria e muito mais.



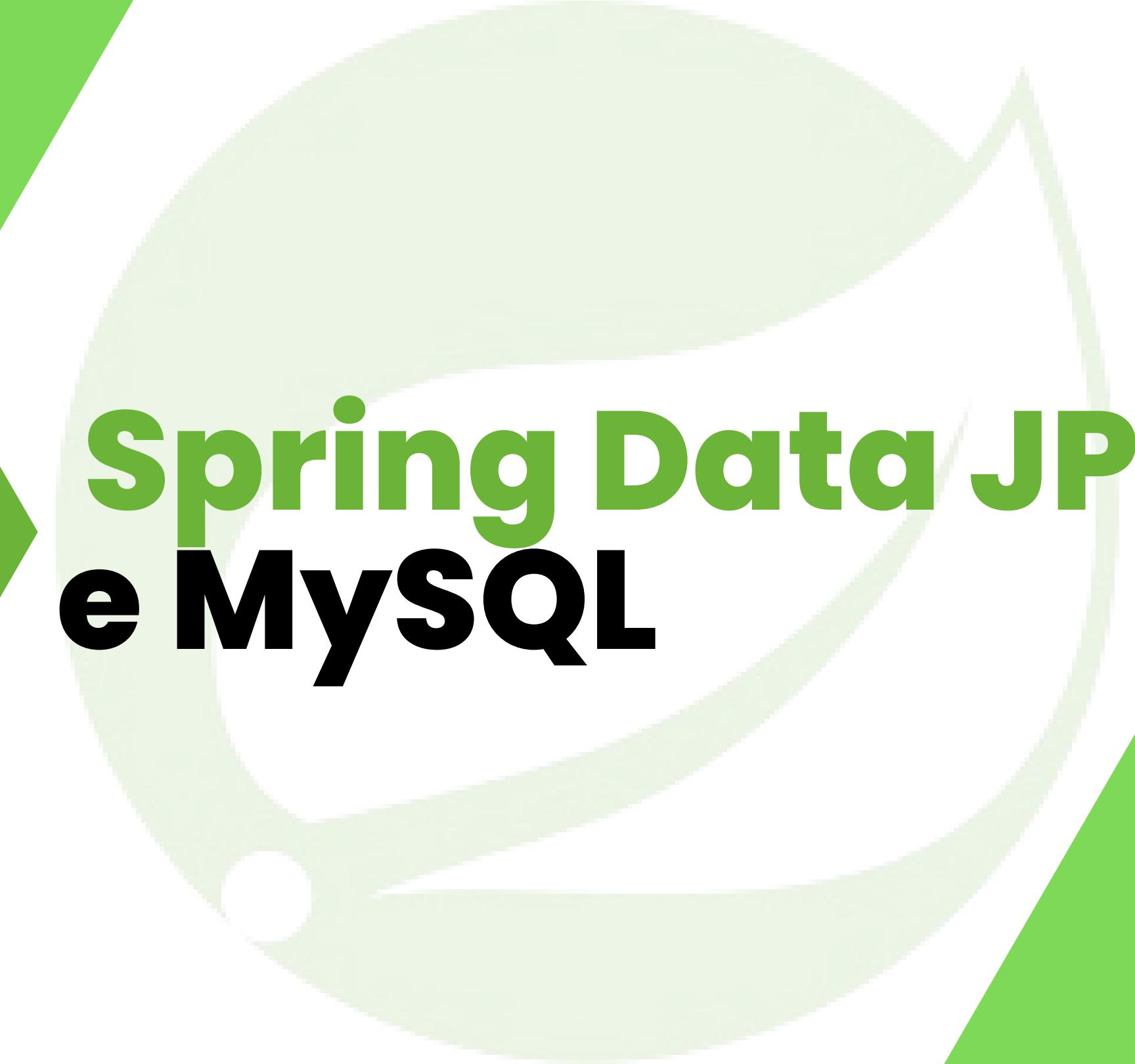
Estrutura do projeto

Arquitetura em camadas

-  **controller** → **Controladores REST**
-  **service** → **Regras de negócio**
-  **repository** → **Acesso a banco de dados**
-  **model** → **Modelos de dados**



Persistência
de Dados



Spring Data JPA
e MySQL

O que é JPA?

O **JPA** (**Java Persistence API**) é uma especificação do Java para trabalhar com persistência de dados de forma orientada a objetos. Ele abstrai a complexidade de escrever SQL puro e permite que usemos anotações para mapear entidades no banco de dados



O que é Hibernate?

O **Hibernate** é a implementação mais popular do JPA. Ele é responsável por traduzir nossas operações Java em comandos SQL



O QUE É SPRING DATA JPA?

O **Spring Data JPA** é um módulo do Spring que facilita ainda mais o uso do JPA. Com ele, conseguimos reduzir a quantidade de código necessário para realizar operações no banco de dados.



Configurando Banco de Dados

**Precisamos adicionar a configuração do banco de dados no nosso
application.properties**

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/homehero?useSSL=false&serverTimezone=UTC
2 spring.datasource.username=root
3 spring.datasource.password=root
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5 spring.jpa.hibernate.ddl-auto=update
6 spring.jpa.show-sql=true
7 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
8 server.port=8080
```



ENTIDADES

- Uma **Entidade** é uma classe Java que representa uma tabela no banco de dados.
- Marcada com **@Entity**.
- Cada **atributo** representa uma **coluna** da tabela.
- **@Id**: Indica a **chave Primária**
- **@GeneratedValue**: Gera o valor automaticamente

O que é JDBC?

JDBC é uma API do Java que permite executar comandos SQL diretamente no banco de dados.

- **Acesso direto ao banco via SQL**
- **Controle total sobre queries e procedures**
- **Ideal para procedures e views complexas**

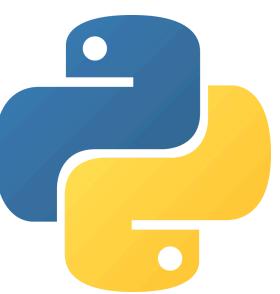


**Executa q procedure
para inserir um cliente.**





Python

The Python logo consists of the word "Python" in a bold, dark blue sans-serif font, centered within a light gray rounded rectangle. To the right of the text is the Python logo icon, which is a yellow and blue abstract shape resembling a pair of interlocking snakes.

Configurando Banco de Dados

Precisamos adicionar a configuração do banco de dados

```
def obter_conexao():
    """Abre uma conexão usando mysql-connector-python."""
    return mysql.connector.connect(
        host=os.environ.get("DB_HOST", "localhost"),
        port=int(os.environ.get("DB_PORT", "3306")),
        user=os.environ.get("DB_USER", "root"),
        password=os.environ.get("DB_PASSWORD", ""),
        database=os.environ.get("DB_NAME", "homehero"),
    )
```



Funções

Views

```
def listar_categorias_servicos():
    """Listar categorias de serviços."""
    with closing[Any](obter_conexao()) as conexao, closing[Any](conexao.cursor(dictionary=True)) as cursor:
        cursor.execute("SELECT * FROM view_lista_categorias ORDER BY cat_nome")
        imprimir_registros(cursor.fetchall())

def listar_servicos_com_categorias():
    """Listar serviços cadastrados com suas categorias."""
    with closing[Any](obter_conexao()) as conexao, closing[Any](conexao.cursor(dictionary=True)) as cursor:
        cursor.execute("SELECT * FROM view_lista_servicos ORDER BY ser_nome")
        imprimir_registros(cursor.fetchall())
```



Funções

Procedures

```
def inserir_cliente_procedure():
    """
    Insere um novo cliente no banco de dados através da stored procedure.
    """
    print("== Inserir Cliente ==")
    nome = input("Nome: ").strip()
    cpf = input("CPF: ").strip()

    # Validação de data com Loop até entrada válida
    while True:
        try:
            nascimento = date.fromisoformat(input("Data de nascimento (YYYY-MM-DD): ").strip())
            break
        except ValueError:
            print("Data inválida, use o formato YYYY-MM-DD.")

    senha = input("Senha: ").strip()

    # Dados do endereço
    print("\n--- Dados do Endereço ---")
    end_logradouro = input("Logradouro: ").strip()
    end_numero = input("Número: ").strip()
    end_complemento = input("Complemento (opcional): ").strip()
    end_bairro = input("Bairro: ").strip()
    end_cidade = input("Cidade: ").strip()
    end_uf = input("UF (2 letras): ").strip().upper()
    end_cep = input("CEP: ").strip()

    email = input("\nE-mail: ").strip()
    telefone = input("Telefone: ").strip()

    with closing[PooledMySQLConnection | MySQLConnectionAbs...obter_conexao() as conexao, closing(conexao.cursor()) as cursor]:
        cursor.callproc(
            "inserir_cliente",
            [nome, cpf, nascimento, senha,
             end_logradouro, end_numero, end_complemento, end_bairro, end_cidade, end_uf, end_cep,
             email, telefone],
        )
        conexao.commit()
    print("\nCliente inserido com sucesso.")
```

```
def buscar_dados_cliente():
    """
    Busca e exibe os dados pessoais completos de um cliente pelo ID.
    Utiliza a stored procedure buscar_dados_pessoais_cliente.
    """
    while True:
        try:
            cli_id = int(input("Informe o ID do cliente: ").strip())
            break
        except ValueError:
            print("Valor inválido, tente novamente.")

    with closing[PooledMySQLConnection | MySQLConnectionAbs...obter_conexao(), closing(conexao.cursor(dictionary=True)) as cursor]:
        cursor.callproc("buscar_dados_pessoais_cliente", [cli_id])
        # Processa todos os resultados retornados pela procedure
        for resultado in cursor.stored_results():
            imprimir_registros(resultado.fetchall())
```



Menu

```
def mostrar_menu():
    """Exibe o menu principal com todas as opções disponíveis."""
    print("===== Menu HomeHero MySQL =====")
    print("\n--- VIEWS ---")
    print("1 - Listar categorias de serviços")
    print("2 - Listar serviços cadastrados com suas categorias")
    print("3 - Listar serviços oferecidos por cada prestador")
    print("4 - Resumo de agendamentos por status")
    print("5 - Resumo de agendamentos por período")
    print("6 - Resumo de agendamentos por região do cliente")
    print("7 - Resumo de pagamentos por status")
    print("8 - Média de avaliação por prestador")
    print("9 - Listar disputas abertas")
    print("10 - Listar disponibilidade de prestadores")
    print("11 - Resumo de receita por prestador")
    print("12 - Resumo de receita por período")
    print("13 - Histórico de agendamentos (mudanças de status)")
    print("\n--- PROCEDURES ---")
    print("14 - Inserir cliente")
    print("15 - Inserir prestador")
    print("16 - Buscar dados pessoais do cliente (por ID)")
    print("17 - Listar agendamentos de um cliente (por ID)")
    print("18 - Buscar dados pessoais do prestador (por ID)")
    print("19 - Inserir novo agendamento de serviço")
    print("20 - Cancelar agendamento de serviço")
    print("21 - Confirmar agendamento de serviço")
    print("22 - Listar agendamentos por período e status")
    print("23 - Registrar avaliação de prestador")
    print("24 - Abrir disputa de reembolso")
    print("25 - Fechar disputa de reembolso")
    print("26 - Inserir pagamento de agendamento")
    print("27 - Pesquisar clientes por nome exato")
    print("28 - Buscar relacionamentos do cliente (por ID)")
    print("29 - Buscar relacionamentos do prestador (por ID)")
    print("\n0 - Sair")
```



Obrigado!

