UNIVERSIDADE DE MOGI DAS CRUZES

BACHARELADO EM ENGENHARIA DE SOFTWARE

B&D VIAGENS

SISTEMA CONTROLADOR DE RESERVA, ORGANIZAÇÃO E REGISTRO DE VIAGENS AO UNIVERSO DE PADRÕES DE PROJETO

BRUNO HIDEO SILVA SHIRAISHI

DIEGO ALVES DA SILVA FAGUNDES

MOGI DAS CRUZES - SP

2025

SUMÁRIO

RESUMO

O presente trabalho apresenta o desenvolvimento de um sistema informatizado de reserva de viagens para uma agência fictícia, elaborado no contexto da disciplina de padrões de projeto do curso de Bacharelado em Engenharia de Software da UMC. A solução foi concebida com base na arquitetura Model-View-Controller (MVC), incorporando padrões de projeto consolidados como Builder, Factory, Command e Facade, em conformidade com o princípio de substituição de Liskov. O sistema possibilita o gerenciamento integral de clientes e reservas de viagens, sendo estruturado para garantir extensibilidade e manutenibilidade. A implementação utilizou a linguagem Java em conjunto com o framework Spring Boot. Os resultados demonstram uma arquitetura coesa que atende aos requisitos de escalabilidade e facilidade de manutenção, validando a eficácia da aplicação estratégica dos padrões selecionados.

Palavras-chave: Sistema de viagens. MVC. Builder. Factory. Command. Facade. Liskov.

ABSTRACT

This project presents the development of a computerized travel reservation system designed for a fictitious agency, conducted as part of the design patterns discipline within the Software Engineering Bachelor's program at UMC. The solution was conceived based on the Model-View-Controller (MVC) architecture, incorporating established design patterns such as Builder, Factory, Command, and Facade, while adhering to the Liskov Substitution Principle. The system enables comprehensive management of customers and travel reservations, structured to ensure extensibility and maintainability. The implementation utilized Java programming language alongside the Spring Boot framework. Results demonstrate a cohesive architecture that meets scalability and maintenance requirements, validating the effectiveness of the strategic application of selected patterns.

Keywords: Travel system. MVC. Builder. Factory. Command. Facade. Liskov.

1 INTRODUÇÃO

As A transformação digital no setor turístico tem se consolidado como elemento fundamental para a competitividade empresarial, considerando que a maior parte das transações comerciais neste segmento migrou para plataformas digitais. Neste contexto, a informatização dos serviços de agências de viagem representa não apenas um diferencial competitivo, mas uma necessidade estratégica que possibilita maior controle sobre reservas, gestão eficiente de dados de clientes e otimização de processos operacionais.

O presente projeto propõe o desenvolvimento de um sistema de reserva de viagens fundamentado em padrões de projeto consolidados e boas práticas de arquitetura de software. O enfoque principal reside na aplicação da arquitetura Model-View-Controller (MVC) em conjunto com padrões específicos como Builder, Factory, Command e Facade, visando promover facilidade de manutenção, reusabilidade de componentes e adequada separação de responsabilidades.

Ademais, buscou-se assegurar o cumprimento rigoroso do princípio de substituição de Liskov, elemento essencial para uma arquitetura orientada á objetos coesa e robusta. Por conseguinte, o trabalho contribui tanto para a consolidação de conhecimentos teóricos quanto para a demonstração prática da aplicação de conceitos fundamentais da engenharia de software em um contexto real de desenvolvimento.

2 DESENVOLVIMENTO

2.1 Arquitetura MVC

O projeto foi desenvolvido seguindo rigorosamente o padrão arquitetural Model-View-Controller (MVC), proporcionando uma separação clara e lógica das responsabilidades do sistema em três camadas distintas:

Model (Modelo): Compreende as entidades fundamentais do domínio, representadas pelas classes Cliente, Reserva e Viagem, organizadas na estrutura de pacotes model.entity. Esta camada encapsula toda a lógica de negócio e as regras de integridade dos dados, garantindo a consistência das informações manipuladas pelo sistema.

View (Visão): Representada pelo conjunto de páginas HTML responsáveis pela interface do usuário, localizadas na estrutura resources/templates. Esta camada concentra exclusivamente os aspectos de apresentação, garantindo a separação entre lógica de negócio e interface gráfica.

Controller (Controlador): Implementado através de classes específicas localizadas no pacote controller, responsáveis por gerenciar as requisições provenientes da interface do usuário e coordenar as interações com os serviços de negócio. Esta camada atua como intermediária, garantindo o desacoplamento entre as demais camadas.

Esta estruturação arquitetural proporciona benefícios significativos em termos de manutenibilidade, testabilidade e evolução do sistema, permitindo modificações independentes em cada camada sem comprometer a integridade do conjunto.

2.2 Padrões de Projeto Aplicados

A implementação incorpora quatro padrões de projeto fundamentais, cada um selecionado estrategicamente para resolver problemas específicos da arquitetura:

Padrão Builder: Implementado através da classe ClienteBuilder, este padrão permite a construção fluente e controlada de objetos Cliente complexos. A escolha deste padrão justifica-se pela necessidade de criar instâncias de cliente com múltiplos atributos obrigatórios e opcionais, encapsulando as regras de validação e inicialização. Esta abordagem proporciona maior flexibilidade na construção de objetos e reduz significativamente a possibilidade de erros durante a instanciação, especialmente em cenários onde diferentes combinações de atributos são necessárias.

Padrão Factory: A classe ViagemFactory encapsula toda a lógica de instanciação de objetos Viagem, promovendo abstração e simplicidade na criação de diferentes tipos de viagem. Este padrão foi selecionado para centralizar a lógica de criação, facilitando futuras extensões do sistema que possam incluir categorias específicas de viagem (nacional, internacional, pacotes promocionais) sem necessidade de modificações extensivas no código cliente.

Padrão Command: Implementado através das classes ReservaCommand e CancelarReservaCommand, em conjunto com o executor CommandExecutor, este padrão encapsula operações como objetos independentes. A aplicação deste padrão permite funcionalidades avançadas como reversão de operações (undo), enfileiramento de comandos, registro de auditoria e execução assíncrona de operações. Esta abordagem é particularmente valiosa em sistemas de reserva, onde o controle transacional e a rastreabilidade de operações são requisitos críticos.

Padrão Facade: A classe ReservaFacade fornece uma interface unificada e simplificada para interação com o subsistema de reservas, ocultando a complexidade inerente às operações de negócio. Este padrão reduz o acoplamento entre o código cliente e os subsistemas internos, facilitando a manutenção e evolução do sistema. Ademais, centraliza pontos de controle para funcionalidades transversais como validação, logging e tratamento de exceções.

2.3 Princípio de Substituição de Liskov

O sistema demonstra aderência rigorosa ao princípio de substituição de Liskov, garantindo que subclasses possam ser utilizadas transparentemente no lugar de suas superclasses sem comprometer o comportamento ou a integridade do programa. Esta conformidade é evidenciada principalmente na implementação do padrão Command, onde as classes ReservaCommand e CancelarReservaCommand são completamente substituíveis pela interface Command.

A aplicação deste princípio assegura robustez arquitetural e facilita a extensibilidade do sistema, permitindo a inclusão de novas implementações de comando sem necessidade de modificações no código que utiliza a interface base. Por conseguinte, o sistema mantém alta coesão e baixo acoplamento, características fundamentais para arquiteturas orientadas a objetos de qualidade.

2.4 Tecnologias Utilizadas

A implementação do sistema baseou-se em um conjunto criteriosamente selecionado de tecnologias consolidadas no mercado:

Java 17: Linguagem principal de desenvolvimento, proporcionando robustez, portabilidade e recursos modernos de programação

Spring Boot: Framework que simplifica significativamente a configuração e o desenvolvimento de aplicações Java empresariais

HTML, CSS, JavaScript: Tecnologias fundamentais para construção da interface do usuário

Thymeleaf: Motor de templates que facilita a integração entre o backend Java e a apresentação HTML

Apache Maven: Ferramenta de gestão de dependências e automação de build

Git/GitHub: Sistema de controle de versão distribuído para gerenciamento do código-fonte

2.5 Funcionalidades do Sistema

O sistema oferece um conjunto abrangente de funcionalidades que atendem aos requisitos operacionais de uma agência de viagens:

Gestão Integral de Clientes: Cadastro completo incluindo identificação (ID, nome, sobrenome), informações pessoais (data de nascimento, sexo), dados de contato (email, celular) e informações legais (CPF, endereço)

Catálogo de Viagens: Cadastro e manutenção de destinos turísticos diversos

Sistema de Reservas: Funcionalidade para associar clientes a destinos específicos, criando reservas controladas

Gestão de Cancelamentos: Implementação de cancelamento de reservas através do padrão Command, garantindo rastreabilidade

Relatórios Operacionais: Listagem detalhada de clientes cadastrados e reservas ativas no sistema

3 CONCLUSÃO

A implementação do sistema de reservas de viagens evidenciou a importância da aplicação criteriosa de padrões de projeto para o desenvolvimento de software robusto e manutenível. Entre as principais contribuições deste trabalho, destaca-se a demonstração prática de como a combinação estratégica de padrões distintos pode resultar em uma arquitetura coesa e bem estruturada.

A arquitetura MVC proporcionou uma separação clara das responsabilidades, facilitando tanto o desenvolvimento quanto a manutenção futura do sistema. Os padrões Builder, Factory, Command e Facade adicionaram camadas significativas de flexibilidade e extensibilidade, enquanto o cumprimento rigoroso do princípio de substituição de Liskov garantiu robustez e coesão no uso de herança e interfaces.

Como limitações do presente trabalho, identifica-se a necessidade de maior exploração de aspectos relacionados à performance do sistema, implementação de mecanismos de segurança mais robustos e integração com sistemas externos. Estas limitações constituem oportunidades valiosas para trabalhos futuros.

O projeto demonstra potencial significativo de expansão, podendo incorporar funcionalidades como autenticação avançada de usuários, integração com APIs de pagamento, desenvolvimento de dashboards administrativos e implementação de sistemas de recomendação personalizados. Tais extensões consolidariam o sistema como uma solução comercialmente viável para o segmento de turismo.

REFERÊNCIAS BIBLIOGRÁFICAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 10719**: informação e documentação: trabalhos acadêmicos.

Dessign Patterns. **Documentação de Dessgin Patterns**. Disponíveis em:

https://refactoring.guru/design-patterns/builder

https://refactoring.guru/design-patterns/factory

https://refactoring.guru/design-patterns/facade

https://refactoring.guru/design-patterns/command

SPRING.IO. **Documentação Oficial do Spring Boot**. Disponível em: https://spring.io/projects/spring-boot. Acesso em: 20 jan. 2025.

APÊNDICE A – TRECHOS DE CÓDIGO

Padrão Builder

```
public class ViagemBuilder {
    private String destino;
    private String localPartida;
    private Date dataPartida;
    private Date dataRetorno;
    private BigDecimal preco;
    private List<Reserva> reservas;

    public ViagemBuilder comDestino(String destino) {
        this.destino = destino;
        return this;
    }

    public ViagemBuilder comLocalPartida(String localPartida) {
        this.localPartida = localPartida;
        return this;
    }
    .
    .
    .
    .
    .
    .
    .
}
```

Padrão Factory

```
public class ViagemFactory {

    public static Viagem criarViagemSimples(String destino, String localPartida, Date
dataPartida, Date dataRetorno, BigDecimal preco) {

        return new ViagemBuilder()

                .comDestino(destino)

                .comLocalPartida(localPartida)

                .comDataPartida(dataPartida)

                .comDataRetorno(dataRetorno)

                .comPreco(preco)

                .comReservas(List.of())

                .build();

    }
```

Padrão Facade

```
public class ReservaFacade {

    @Autowired

    private ClienteRepository clienteRepository;

    @Autowired

    private ViagemRepository viagemRepository;

    @Autowired

    private ReservaRepository reservaRepository;


    public String realizarReserva(Long clienteId, Integer viagemId, String situacao) {

        Optional<Cliente> clienteOpt = clienteRepository.findById(clienteId);

        Optional<Viagem> viagemOpt = viagemRepository.findById(viagemId);


        if (clienteOpt.isPresent() && viagemOpt.isPresent()) {
```

```
            Reserva reserva = new Reserva();
            reserva.setCliente(clienteOpt.get());
            reserva.setViagem(viagemOpt.get());
            reserva.setDataReserva(new Date()); // Data atual
            reserva.setSituacao(situacao);

            reservaRepository.save(reserva);
            return "Reserva realizada com sucesso!";
        } else {
            return "Cliente ou Viagem não encontrados.";
        }
    }
```

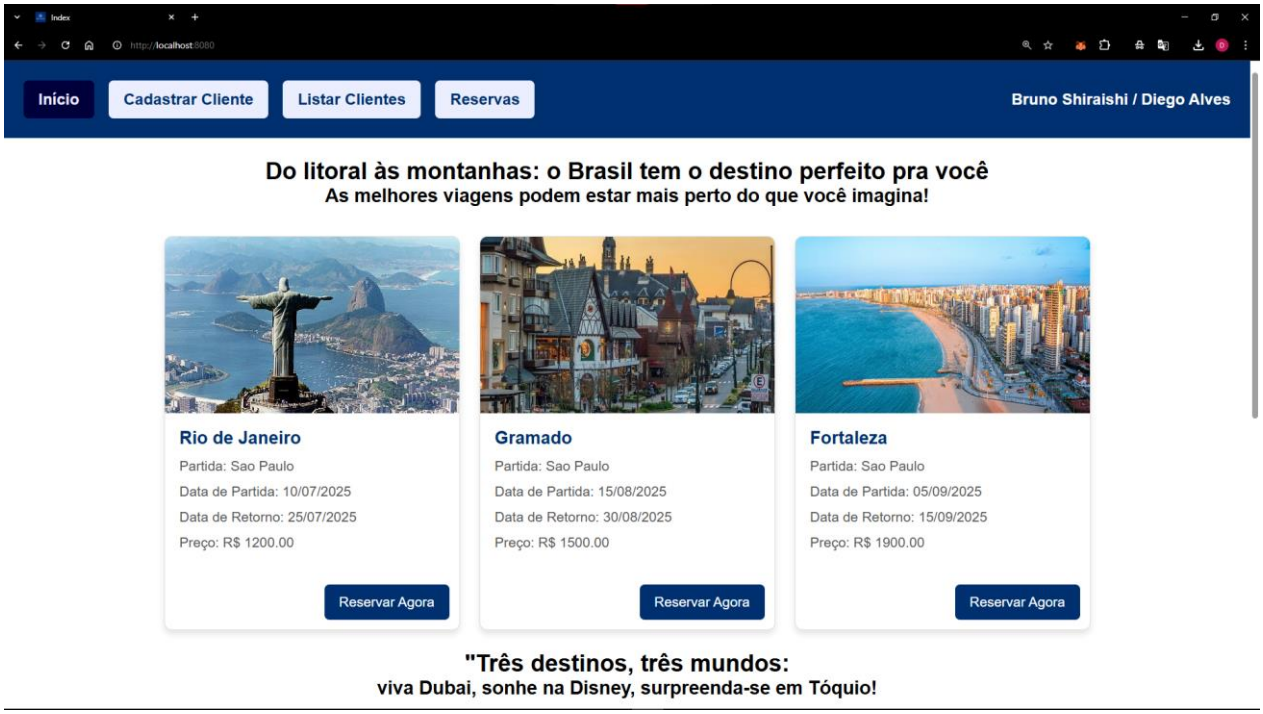
Padrão Command

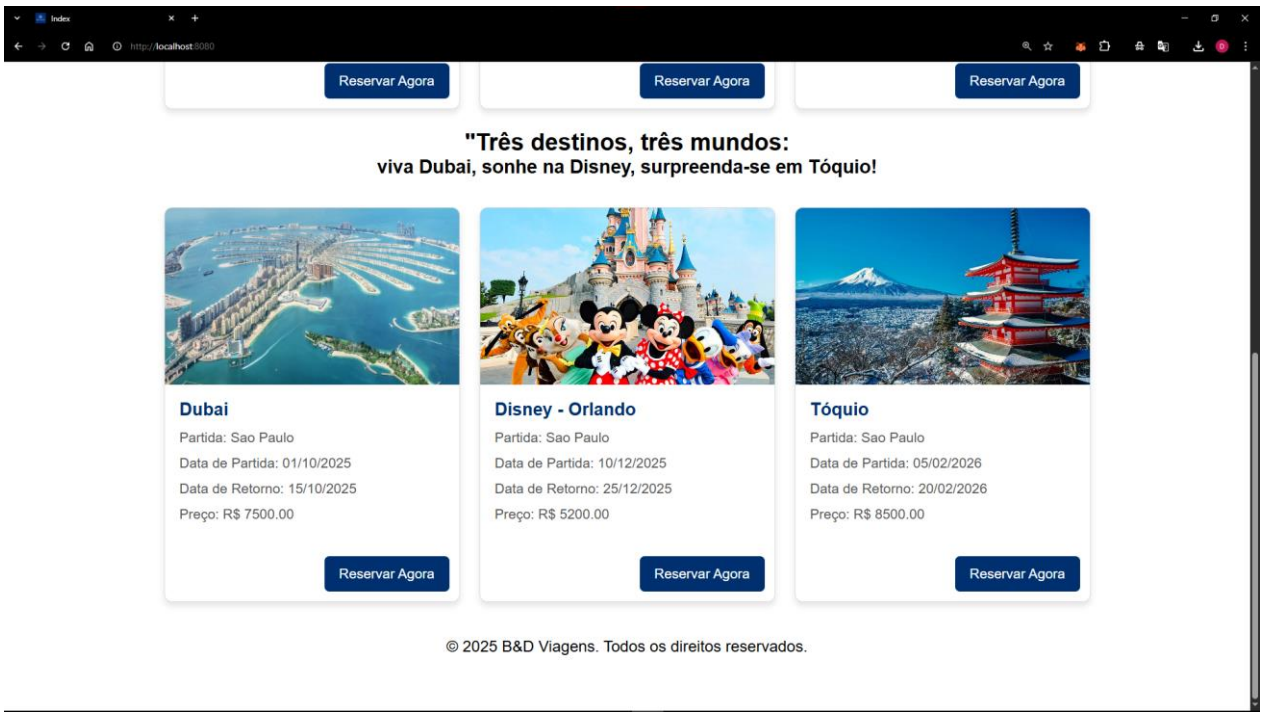
```
public class ReservaCommand implements Command {

    private final ReservaFacade reservaFacade;
    private final Lo
ng clienteId;
    private final Integer viagemId;
    private final String situacao;

    public ReservaCommand(ReservaFacade reservaFacade, Long clienteId, Integer
viagemId, String situacao) {
        this.reservaFacade = reservaFacade;
        this.clienteId = clienteId;
        this.viagemId = viagemId;
        this.situacao = situacao;
```

```
    }

    @Override
    public String executar() {
        return reservaFacade.realizarReserva(clienteId, viagemId, situacao);
    }
}
```

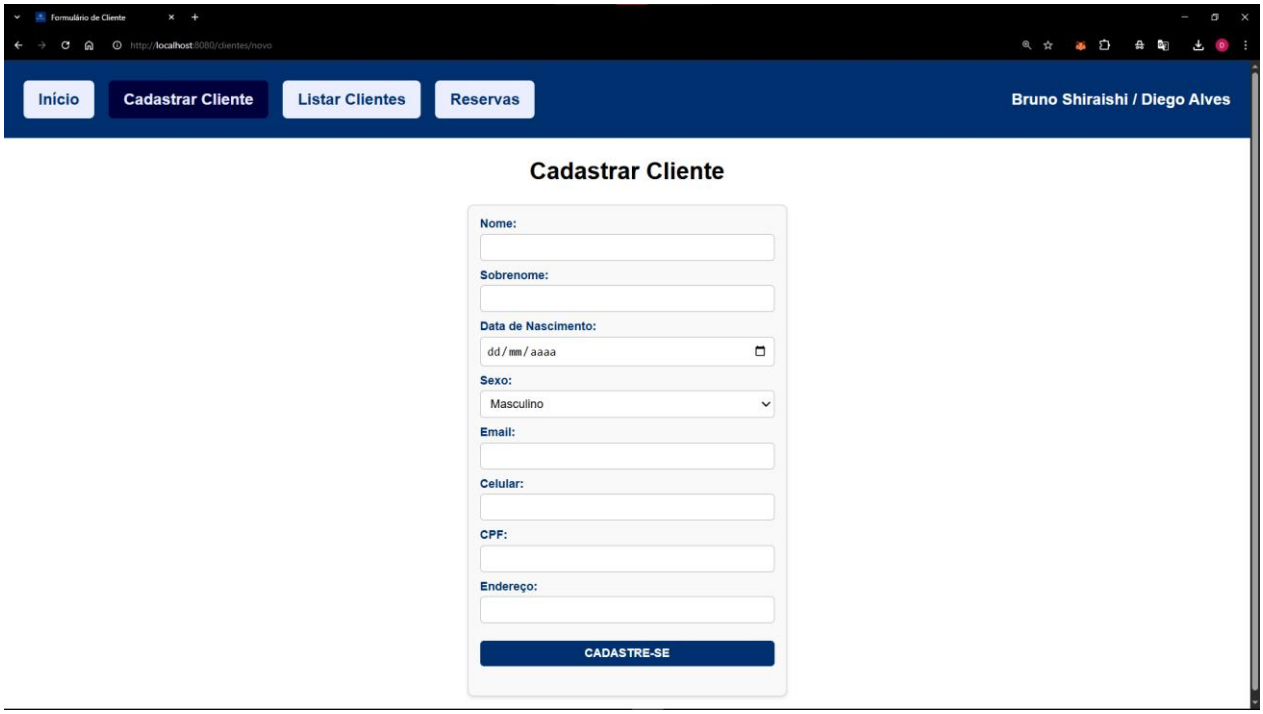
ANEXO A – PRINT DE TELAS DO SISTEMA

Tela Início (index.html)

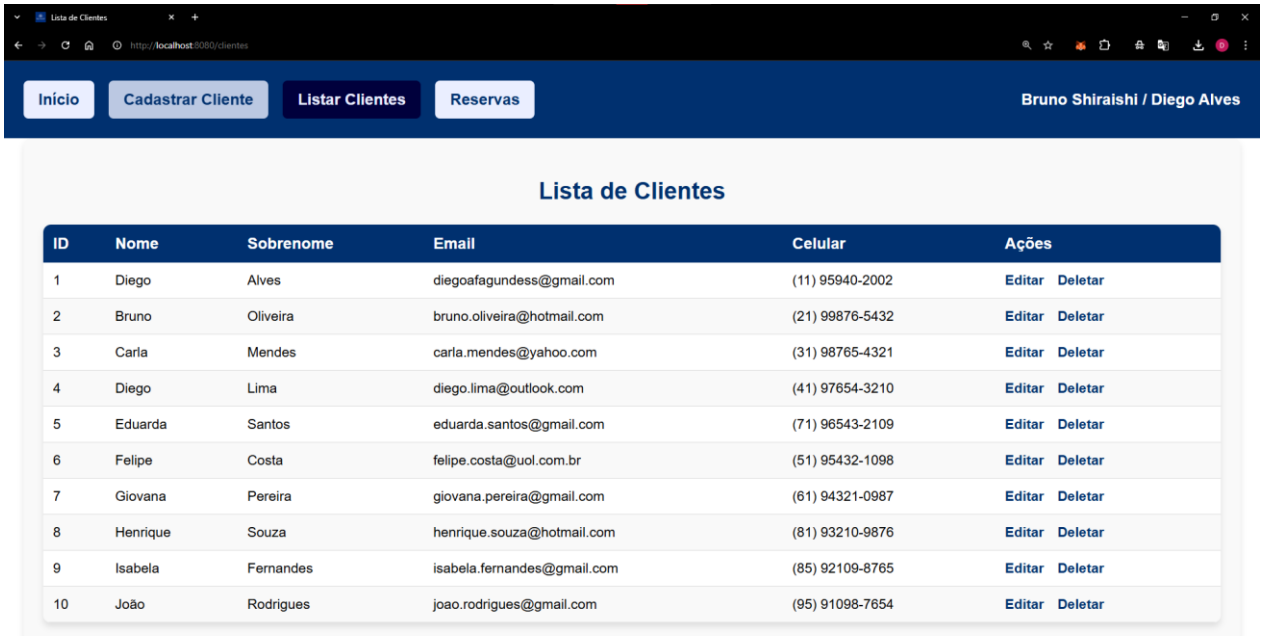




Cadastro de Clientes (form.html)

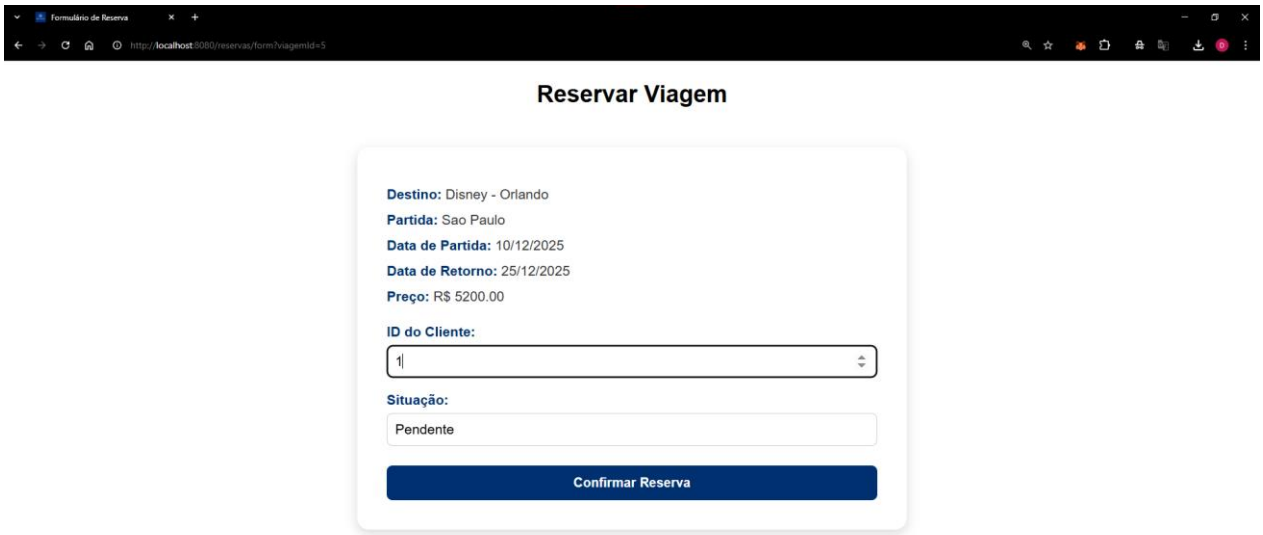


Listagem de Clientes (lista.html)



ID	Nome	Sobrenome	Email	Celular	Ações
1	Diego	Alves	diegoafagundess@gmail.com	(11) 95940-2002	Editar Deletar
2	Bruno	Oliveira	bruno.oliveira@hotmail.com	(21) 99876-5432	Editar Deletar
3	Carla	Mendes	carla.mendes@yahoo.com	(31) 98765-4321	Editar Deletar
4	Diego	Lima	diego.lima@outlook.com	(41) 97654-3210	Editar Deletar
5	Eduarda	Santos	eduarda.santos@gmail.com	(71) 96543-2109	Editar Deletar
6	Felipe	Costa	felipe.costa@uol.com.br	(51) 95432-1098	Editar Deletar
7	Giovana	Pereira	giovana.pereira@gmail.com	(61) 94321-0987	Editar Deletar
8	Henrique	Souza	henrique.souza@hotmail.com	(81) 93210-9876	Editar Deletar
9	Isabela	Fernandes	isabela.fernandes@gmail.com	(85) 92109-8765	Editar Deletar
10	João	Rodrigues	joao.rodrigues@gmail.com	(95) 91098-7654	Editar Deletar

Formulário de Cadastro de Reserva (formReserva.html)



Listagem de Reservas (listaReservas.html)