Centro Universitário Senac Santo Amaro

Plataforma De Reservas Em Hotel

Nomes: Eric Paixão Andrade e Leonardo Gama Pereira

Professor: Carlos Verissimo

São Paulo

2023

Sumário

1 Descrição do Domínio do problema

Criação de um sistema para reservas de hotéis que facilite o processo de escolha de quartos e ofereça sugestões personalizadas com base nas preferências dos hóspedes.

2 Planejamento

3 Requisitos do Cliente

3.1 Requisitos Funcionais

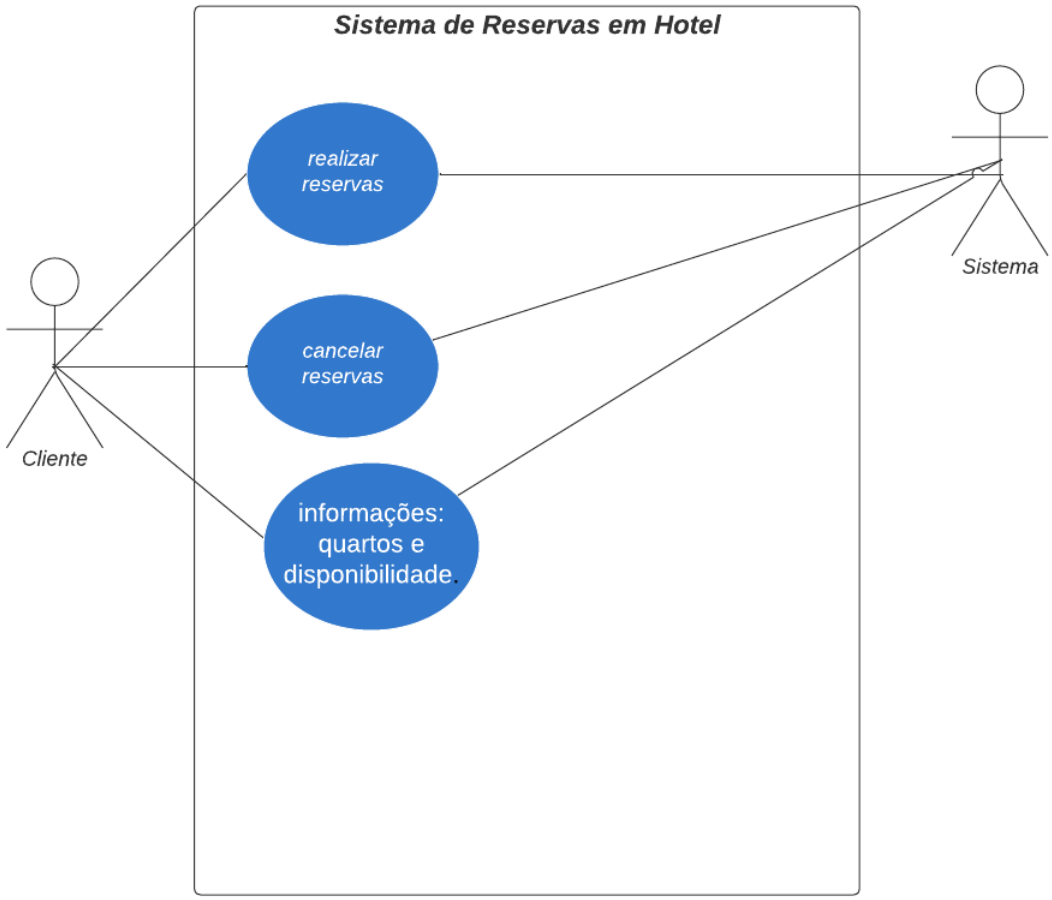
- RF1: **Registro de Usuário:** Permitir que os usuários se cadastrem na plataforma com informações básicas, como nome, e-mail e senha.
- RF2: **Pesquisa de Quartos:** Possibilitar aos usuários buscar quartos disponíveis com base em datas de check-in e check-out, bem como outros filtros, como tipo de quarto e comodidades.
- RF3: **Seleção de Quartos:** Permitir que os usuários escolham quartos disponíveis com base em suas preferências e requisitos.
- RF4: **Reserva de Quartos:** Capacitar os usuários a fazerem reservas para quartos selecionados, inserindo detalhes como datas, número de hóspedes e informações de contato.
- RF5: **Cálculo de Preços:** Calcular automaticamente o preço total da reserva com base nas datas escolhidas, número de hóspedes e tarifas do quarto.
- RF6: **Gestão de Reservas:** Permitir que os usuários visualizem e gerenciem suas reservas, incluindo a possibilidade de alterar datas, adicionar serviços extras e efetuar cancelamentos.
- RF7: **Pagamento Seguro:** Integrar um sistema de pagamento seguro que aceite diferentes métodos de pagamento para confirmar as reservas.
- RF8: **Confirmações por E-mail:** Enviar e-mails de confirmação aos clientes após a conclusão da reserva, fornecendo detalhes da reserva e informações importantes.
- RF9: **Sugestões Personalizadas:** Oferecer sugestões de quartos e comodidades com base nas preferências do usuário e em reservas anteriores.
- RF10: **Gestão de Disponibilidade:** Atualizar a disponibilidade de quartos em tempo real e bloquear quartos durante o processo de reserva para evitar duplicações.
- RF11: **Integração de Avaliações:** Permitir que os hóspedes deixem avaliações e comentários sobre suas estadias após o check-out.
- RF12: **Suporte ao Cliente:** Incluir um canal de suporte para que os usuários possam entrar em contato em caso de dúvidas, problemas ou assistência durante o processo de reserva.

3.2 Requisitos Não Funcionais

- RNF1: **Usabilidade:** A interface do sistema deve ser intuitiva e de fácil utilização, mesmo para usuários sem experiência técnica.
- RFN2: **Desempenho:** O sistema deve ser responsivo, permitindo que os usuários realizem pesquisas, reservas e pagamentos de forma rápida e eficiente.
- RFN3: **Disponibilidade:** O sistema deve estar disponível 24/7 para permitir que os usuários acessem e façam reservas a qualquer momento.
- RFN4: **Segurança:** Dados pessoais e de pagamento dos usuários devem ser protegidos por medidas de segurança, como criptografia de dados.
- RFN5: **Tempo de Resposta:** O tempo de resposta do sistema, desde a pesquisa até a conclusão da reserva, deve ser rápido para evitar frustração dos usuários.
- RFN6: **Escalabilidade:** O sistema deve ser capaz de lidar com um aumento no número de usuários e reservas sem comprometer o desempenho.
- RFN7: **Compatibilidade:** O sistema deve ser compatível com diferentes dispositivos, navegadores e sistemas operacionais para atender a uma variedade de usuários.
- RFN8: **Documentação:** Fornecer documentação clara e simples sobre como usar o sistema, para que os usuários possam entender facilmente suas funcionalidades.
- RFN9: **Conformidade com Regulamentações:** O sistema deve seguir regulamentações de proteção de dados e leis de privacidade do setor de hospitalidade.
- RFN10: **Manutenção:** O sistema deve ser projetado de maneira a permitir manutenções regulares sem interromper o funcionamento geral.
- RFN11: **Backup e Recuperação:** Implementar rotinas de backup regulares e um plano de recuperação de desastres para proteger os dados dos usuários.
- RFN12: **Eficiência de Código:** O sistema deve ser otimizado para garantir um uso eficiente de recursos, minimizando tempos de carregamento.
- RFN13: **Acessibilidade:** Garantir que o sistema seja acessível para pessoas com deficiências, seguindo diretrizes de acessibilidade da web.
- RFN14: **Design Responsivo:** A interface do sistema deve se adaptar a diferentes tamanhos de tela, desde dispositivos móveis até desktops.
- RFN15: **Feedback ao Usuário:** Fornecer feedback claro aos usuários durante cada etapa do processo de reserva para evitar confusões.

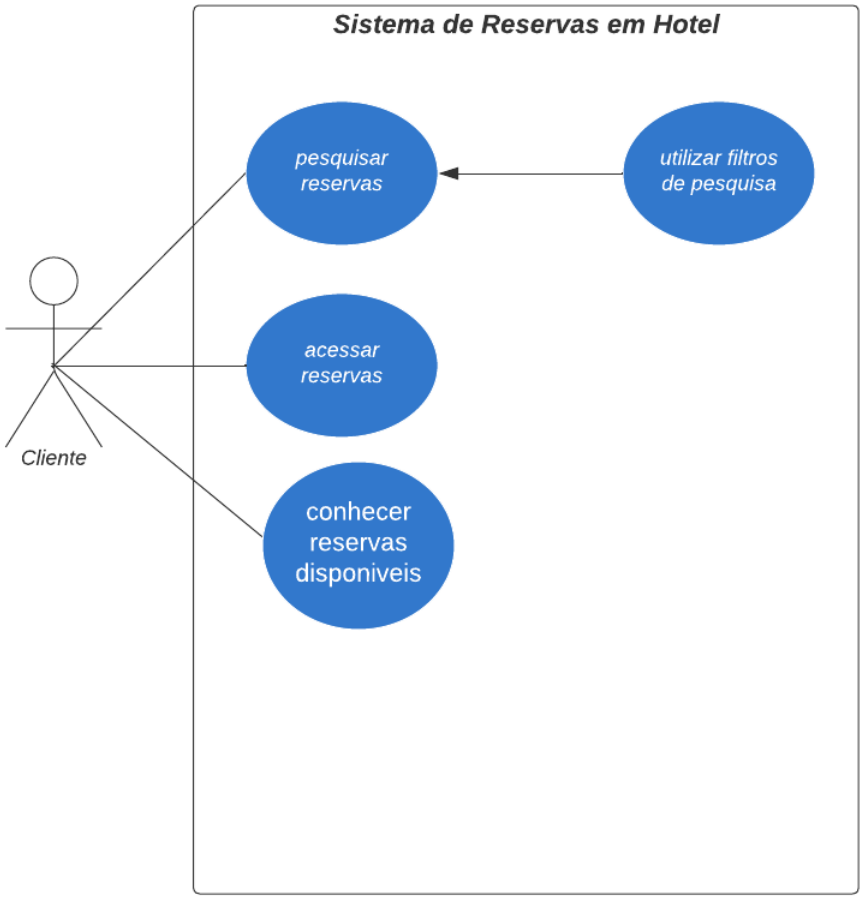
4 Casos de Uso (UC)

4.1 Diagrama de Caso de Uso - Visão Geral



4.2 Caso de Uso - Caso de Uso #1 – Realizar Reserva

4.2.1 Diagrama do Caso de uso #1



4.2.2 Detalhamento do Caso de uso #1.1 – Pesquisar reservas

Nome do Caso de Uso:	1.1 – Pesquisar reservas
Atores:	Clientes
Trigger:	Necessidade de reserva do cliente
Pré-requisito	Logar no sistema
Fluxo de eventos	Cliente entra no sistema; pesquisa por datas e quartos disponíveis.

4.2.3 Detalhamento do Caso de uso #1.2 – Utilizar filtro

Nome do Caso de Uso:	1.2 – Utilizar filtros
Atores:	Cliente
Trigger:	Estar buscando por reserva
Pré-requisito:	Escolher data e quarto
Fluxo de eventos	Cliente estar logado no sistema; filtrar datas e quartos de seu interesse e disponibilidade

4.2.4 Detalhamento do Caso de uso #1.3 – Acessar reservas

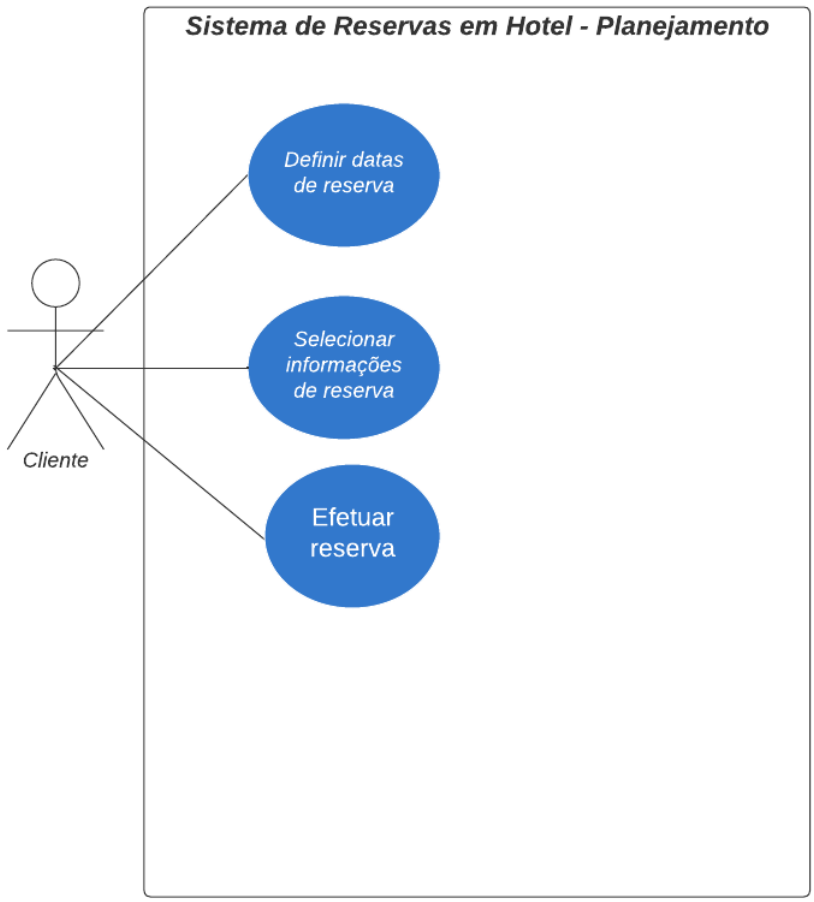
Nome do Caso de Uso:	1.3 – Acessar reservas
Atores:	Cliente
Trigger:	Necessidade de uma reserva
Pré-requisito:	Estar logado no sistema / Ter selecionado as datas
Fluxo de eventos:	Cliente logo no sistema; Verifica as opções de reservas;

4.2.5 Detalhamento do Caso de uso #1.4 – Conhecer reservas disponíveis.

Nome do Caso de Uso:	1.4 – Conhecer reservas disponíveis.
Atores:	Cliente
Trigger:	Conhecer as reservas disponíveis
Pré-requisito:	Ter preenchido os filtros
Fluxo de eventos:	Preencher os filtros

4.3 Caso de Uso - Caso de Uso #2 – Configurar planejamento – Informações: quartos e disponibilidade.

4.3.1 Diagrama do Caso de uso #2



4.3.2 Detalhamento do Caso de uso #2.1 – Definir datas de reserva

Nome do Caso de Uso:	2.1 – Definir datas de reserva
Atores:	Cliente
Trigger:	Definir datas
Pré-requisito:	Estar efetuando a reserva
Fluxo de eventos:	Cliente entra no sistema; Entra na página de reserva; Escolhe as datas de reserva;

4.3.3 Detalhamento do Caso de uso #2.2 – Selecionar informações de reserva

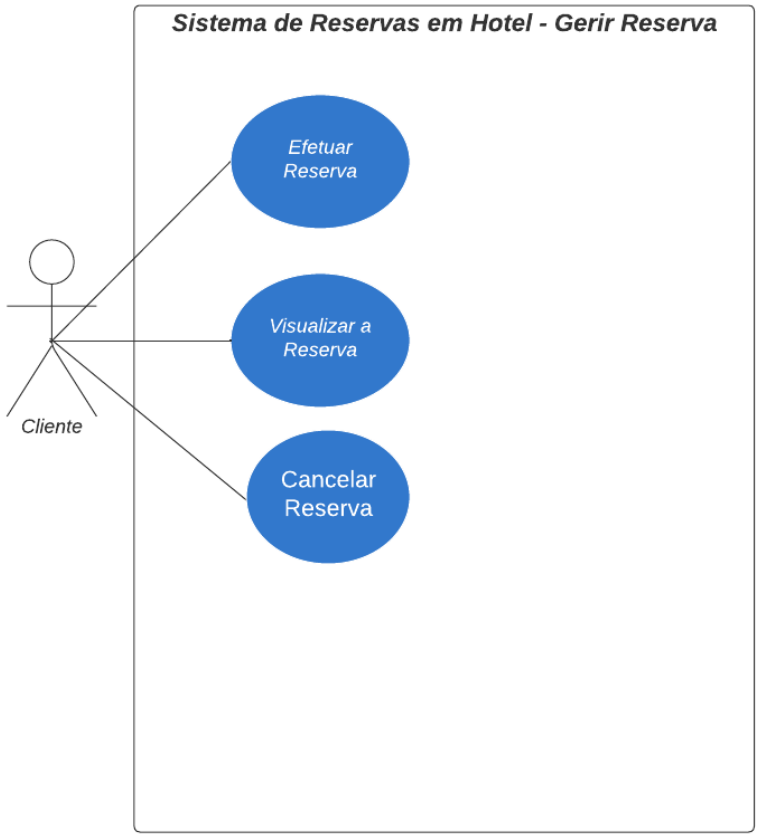
Nome do Caso de Uso:	2.2 – Selecionar informações de reserva
Atores:	Cliente
Trigger:	Selecionar quarto e datas
Pré-requisito:	Estar realizando a reserva
Fluxo de eventos	Cliente loga no sistema; Busca por datas e quartos disponíveis.

4.3.4 Detalhamento do Caso de uso #2.3 – Efetuar a reserva

Nome do Caso de Uso:	2.3 – Efetuar a reserva
Atores:	Cliente
Trigger:	Finalizar a reserva
Pré-requisito:	Ter selecionado data e quarto.
Fluxo de eventos:	Cliente loga no sistema; Entra na página de reservas; seleciona datas e quarto; efetua a reserva.

4.4 Caso de uso - Caso de Uso #3 – Acessar Reservas

4.4.1 Diagrama do Caso de uso #3



4.4.2 Detalhamento do Caso de uso #3.1 – Efetuar a reserva

Nome do Caso de Uso:	3.1 – Efetuar a reserva
Atores:	Cliente
Trigger:	Efetuar a reserva
Pré-requisito:	Ter preenchido todos os filtros e escolhido datas e quarto
Fluxo de eventos:	Preencher os filtros; Selecionar datas e quarto; Realizar reserva.

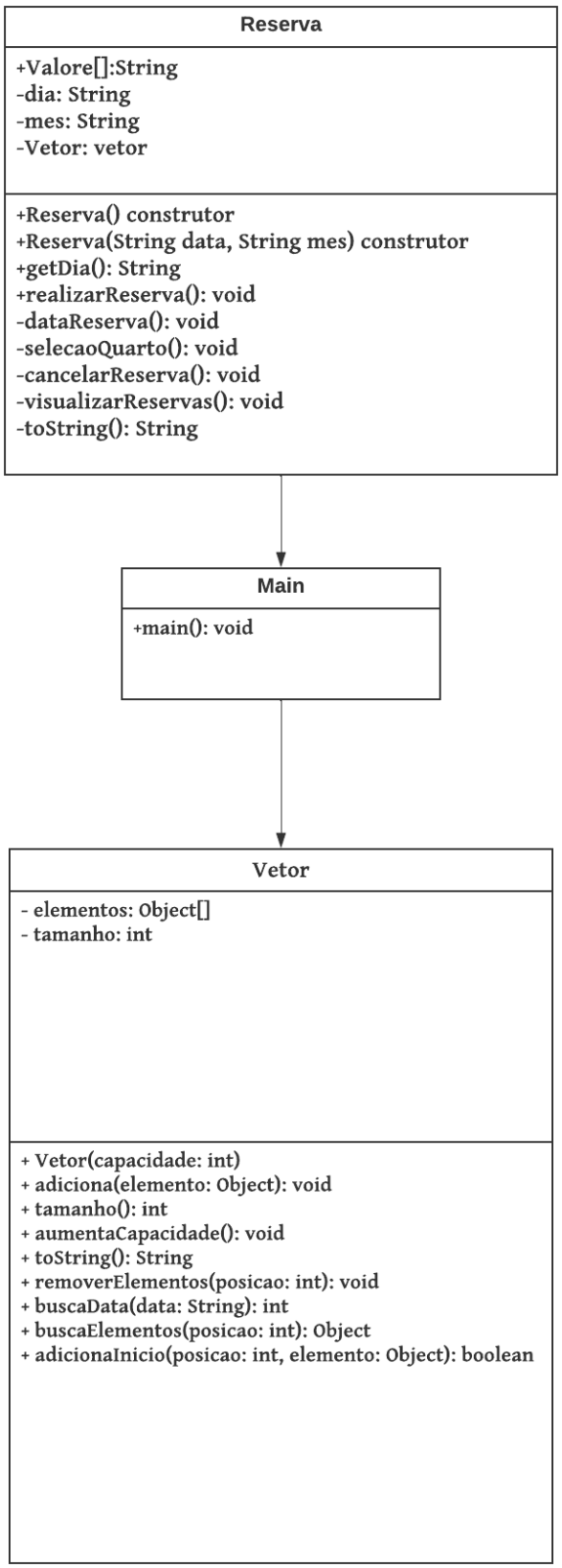
4.4.3 Detalhamento do Caso de uso #3.2 – Visualizar a reserva

Nome do Caso de Uso:	3.2 – Visualizar a reserva
Atores:	Cliente
Trigger:	Visualizar as reservas feitas
Pré-requisito:	Ter feito a reserva
Fluxo de eventos:	Fazer a reserva; Visualizar a reserva

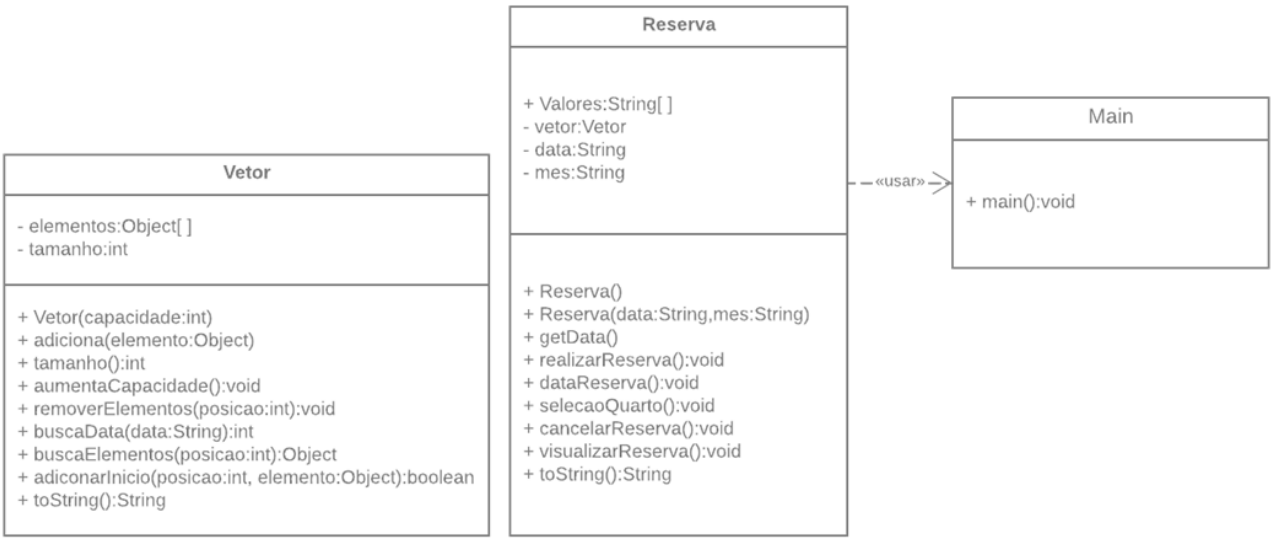
4.4.4 Detalhamento do Caso de uso #3.3 – Cancelar reserva

Nome do Caso de Uso:	3.3 – Cancelar reserva
Atores:	Cliente
Trigger:	Cancelar a reserva
Pré-requisito:	Ter realizado pelo menos uma reserva
Fluxo de eventos:	Ter realizado a reserva; Querer cancelar uma reserva

5 Diagrama UML



6 Implementação de Encapsulamento



6.1 Encapsulamento da classe Reserva:

Atributos Privados: Os atributos ‘data’ e ‘mês’ são declarados como privados com os modificadores de acesso ‘private’. Isso significa que esses atributos não podem ser acessados diretamente de fora da classe ‘Reserva’. Isso é uma prática de encapsulamento, pois oculta o estado interno da classe e restringe o acesso direto aos atributos.

```
private String data;
private String mes;
```

Métodos Públicos de Acesso: A classe ‘Reserva’ fornece métodos públicos para acessar e manipular os atributos privados ‘data’ e ‘mes’. Os métodos ‘getData()’ permitem obter o valor desses atributos, enquanto os construtores e outros métodos, como ‘realizarReserva()’, ‘dataReserva()’, ‘selecaoQuarto()’, ‘cancelarReserva()’, ‘visualizarReservas()’, fornecem funcionalidades relacionadas ao objeto Reserva.

```
public String getData() {
    return data;
}
```

Ocultação de Detalhes Internos: Os detalhes internos de como uma reserva é manipulada, como a lógica para realizar uma reserva, cancelar uma reserva ou visualizar reservas, são encapsulados dentro da classe Reserva. Os métodos públicos são a única maneira de interagir com esses detalhes internos, fornecendo uma interface controlada para o mundo externo.

Utilização do Atributo Privado vetor: O atributo privado vetor, que é uma instância da classe Vetor, também segue o encapsulamento, pois não é diretamente acessível fora da classe Reserva. A classe Reserva pode interagir com vetor por meio de métodos da classe Vetor, presumivelmente encapsulados dentro dela.

```
private static Vetor vetor = new Vetor(10);
```

Em resumo, o código demonstra boas práticas de encapsulamento ao ocultar detalhes internos da classe Reserva, fornece métodos públicos para acesso controlado aos atributos e comportamentos e tratar exceções para manter a robustez do sistema sem expor detalhes de implementação.

6.2 Encapsulamento da classe Vetor:

Atributos Privados: Os atributos ‘elementos’ e ‘tamanho’ são declarados como privados com o modificador de acesso ‘private’. Isso significa que eles não podem ser acessados diretamente fora da classe ‘Vetor’, contribuindo para o encapsulamento, pois oculta o estado interno da classe.

```
private Object[] elementos;
private int tamanho;
```

Métodos Públicos de Acesso e Comportamento: A classe Vetor fornece métodos públicos para acessar e manipular seus atributos e comportamentos. Isso inclui métodos para adicionar elementos, verificar o tamanho, aumentar a capacidade, remover elementos, buscar elementos por posição e buscar elementos por data. Todos esses métodos são a interface controlada para interagir com a classe.

Tratamento de Exceções: Quando ocorrem erros, como tentar adicionar um elemento em um vetor cheio ou acessar uma posição inválida, a classe Vetor lança exceções e as trata internamente. Isso ajuda a manter a robustez do código e oculta os detalhes de implementação de como essas exceções são tratadas.

Método toString(): A classe Vetor substitui o método toString() para fornecer uma representação em string de seus elementos. Isso é uma prática comum para facilitar a visualização e depuração dos objetos.

Operações de Manipulação de Vetor: A classe Vetor encapsula várias operações comuns de manipulação de vetores, como adicionar, remover e buscar elementos. Essas operações são executadas internamente na classe, e os detalhes de implementação são ocultos do código cliente.

O encapsulamento aqui é evidenciado pelo fato de que os detalhes internos da estrutura de dados do vetor (como o array elementos) e sua manipulação são abstraídos e controlados por meio de métodos públicos. Isso permite que o código Reserva use a classe Vetor de forma segura, sem a necessidade de conhecer ou manipular diretamente os atributos internos do vetor.

7 Conclusão

Neste trabalho, abordamos a concepção e análise de um sistema de plataforma de reserva de hotel. Exploramos detalhadamente os requisitos funcionais e não funcionais, apresentamos um diagrama de caso de uso e um diagrama UML, e discutimos os princípios de encapsulamento.

Os requisitos funcionais delineados para o sistema incluem a capacidade de pesquisa de hotéis, visualização de informações detalhadas, reserva de quartos e gerenciamento de reservas. Os requisitos não funcionais ressaltam a importância da eficiência, escalabilidade, segurança e usabilidade do sistema.

O diagrama de caso de uso fornece uma representação visual das interações entre os atores e o sistema, esboçando as principais funcionalidades. O diagrama UML destaca a estrutura e relacionamentos das classes do sistema, incluindo atributos e métodos.

O princípio de encapsulamento é essencial para garantir a segurança, manutenção e operação confiável do sistema. Em resumo, este trabalho enfatiza a importância de uma abordagem abrangente no desenvolvimento do sistema de plataforma de reserva de hotel, garantindo a implementação eficaz das funcionalidades necessárias.