

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CORNÉLIO PROCÓPIO
ENGENHARIA DA COMPUTAÇÃO

ALLAN GUILHERME DE OLIVEIRA SOARES DE SOUZA
MATHEUS FERREIRA ALPHONSE DOS ANJOS
VICTOR EHITI ITIMURA TAMAY

HOSPITAL

CORNÉLIO PROCÓPIO

JUNHO, 2025

ALLAN GUILHERME DE OLIVEIRA SOARES DE SOUZA

MATHEUS FERREIRA ALPHONSE DOS ANJOS

VICTOR EHITI ITIMURA TAMAY

HOSPITAL

Projeto elaborado na disciplina de Programação Orientada a Objetos 2 do curso de Engenharia da Computação, do Campus Cornélio Procópio da Universidade Tecnológica Federal do Paraná.

Professora: Gisele Alves Santa

CORNÉLIO PROCÓPIO

JUNHO, 2025

RESUMO

O presente projeto detalha o desenvolvimento do "HospitAll", um sistema de gerenciamento de informações hospitalares. O objetivo central é otimizar as operações administrativas de uma unidade de saúde, centralizando o cadastro e o controle de pacientes, médicos e estagiários. A metodologia envolveu o levantamento de requisitos funcionais e não funcionais, a modelagem do sistema com diagramas de Casos de Uso, Atividades e Classes, e a implementação seguindo boas práticas de programação. Como resultado, obteve-se um sistema capaz de realizar cadastros, consultas, edições e exclusões de registros, além de gerar relatórios e associar médicos a estagiários, contribuindo para um aprendizado prático em programação orientada a objetos.

Palavras-chave: Gerenciamento Hospitalar, Prontuário Eletrônico, Programação Orientada a Objetos, Java, Sistema de Saúde.

SUMÁRIO

1 Introdução	2
2 FERRAMENTAS E TECNOLOGIAS	3
3 DESENVOLVIMENTO	3
3.1 Levantamento dos Requisitos	3
3.2 Diagrama de Casos de Uso	5
3.3 Diagrama de Classes	6
3.4 Boas Práticas	6
3.5 Banco de Dados	12
3.6 Telas do sistema	13
3.7 Link do repositório	14
4 CONCLUSÕES	14
1 INTRODUÇÃO	

O gerenciamento de informações em unidades de saúde é uma tarefa complexa que exige precisão, segurança e agilidade. A transição de sistemas manuais para soluções digitais centralizadas é fundamental para otimizar processos e reduzir erros. Neste contexto, o projeto "HospitAll" foi desenvolvido.

O sistema tem como objetivo centralizar e facilitar o gerenciamento de informações hospitalares, permitindo o cadastro, controle e consulta de pacientes, médicos e estagiários. A aplicação foi projetada para oferecer uma interface intuitiva e funcionalidades robustas que auxiliam nas operações diárias de uma unidade de saúde, focando especificamente no controle das pessoas que trabalham ou são atendidas no local. O escopo do projeto não abrange a gestão administrativa de medicamentos ou a alocação de leitos.

2 FERRAMENTAS E TECNOLOGIAS

Para a elaboração do projeto, foram utilizadas as seguintes ferramentas e tecnologias:

Linguagem de Programação: *Java*

IDE (Ambiente de Desenvolvimento Integrado): *NetBeans*

Ferramenta de Modelagem UML: *Astah UML*

Sistema de Persistência de Dados: *PostgreSQL*

Sistema de Controle de Versão: *Git/GitHub*

3 DESENVOLVIMENTO

3.1 Levantamento dos Requisitos

Requisitos Funcionais

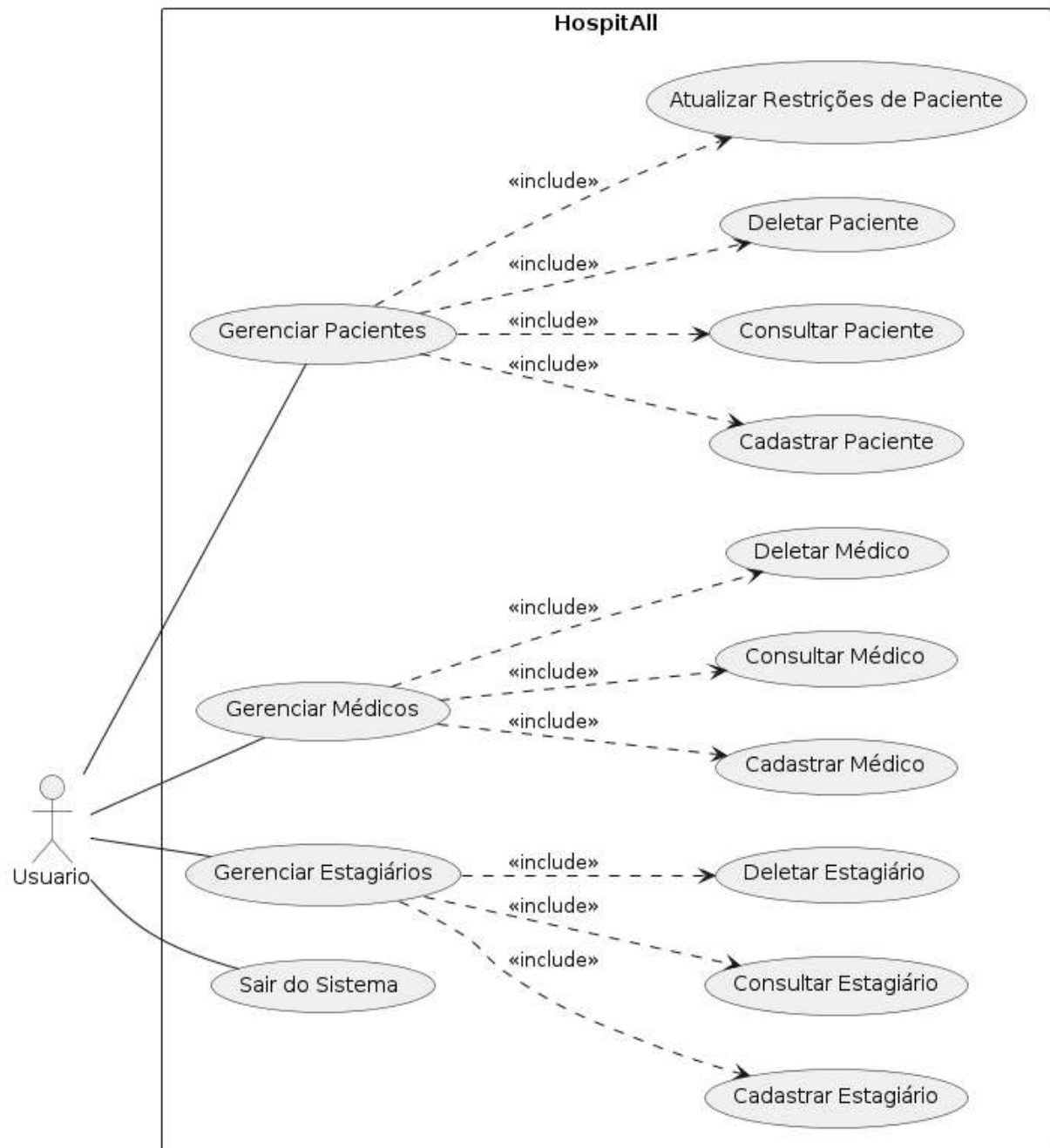
Identificador	Descrição	Prioridade
RF01	O sistema deve permitir o cadastro de pacientes com nome, CPF, idade, tipo de doença e sintomas.	Alta
RF02	O sistema deve permitir o registro de médicos com nome, CRM, especialidade e carga horária semanal.	Alta
RF03	O sistema deve permitir o cadastro de estagiários com nome, matrícula, curso e período.	Média
RF04	O sistema deve salvar automaticamente os dados inseridos pelo usuário em arquivos.	Média
RF05	O sistema deve exibir a listagem completa de pacientes, médicos e estagiários.	Alta
RF06	O sistema deve permitir a edição de informações dos registros previamente cadastrados.	Alta
RF07	O sistema deve permitir a exclusão de pacientes, médicos e estagiários.	Média

RF08	O sistema deve validar entradas, como CPF inválido, campos vazios, ou idade negativa.	Alta
RF09	Os dados cadastrados devem ser salvos em arquivos .txt, .csv, ou banco de dados simples.	Baixa
RF10	A idade do paciente deve ser calculada automaticamente com base na data de nascimento.	Alta
RF11	O sistema deve permitir associar estagiários a médicos responsáveis.	Média
RF12	O sistema deve permitir registrar um histórico de consultas ou atendimentos para cada paciente.	Média
RF13	O sistema deve permitir buscar registros por CPF e CRM em tempo real.	Alta
RF14	O sistema deve contar automaticamente a quantidade de pacientes, médicos e estagiários.	Média
RF15	O sistema deve gerar um relatório resumido em PDF ou texto.	Baixa

Requisitos Não Funcionais

Identificador	Descrição	Tipo
RNF01	O sistema deve restringir o uso de determinadas funcionalidades com base no nível de acesso do usuário.	Segurança
RNF02	O tempo de resposta para operações como salvar ou excluir deve ser inferior a 2 segundos.	Desempenho
RNF03	O sistema deve validar automaticamente os campos obrigatórios antes de permitir o envio.	Usabilidade
RNF04	A consulta de dados deve estar acessível por meio da Internet, compatível com os navegadores mais populares.	Portabilidade
RNF05	O código-fonte deve ser documentado com comentários e JavaDoc.	Documentação

3.2 Diagrama de Casos de Uso



Java

```
public class PacienteDAO {

    public void save(Paciente paciente) {
        DoencaDAO doencaDAO = new DoencaDAO();
        doencaDAO.save(paciente.getDoenca());

        String sql = "INSERT INTO hospital.pessoa (nome, cpf, idade,
tipo_pessoa, doenca_id) VALUES (?, ?, ?, ?, ?)";

        try (Connection conn = ConnectionFactory.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            stmt.setString(1, paciente.getNome());
            stmt.setInt(2, paciente.getCpf());
            stmt.setInt(3, paciente.getIdade());

            stmt.setString(4, "PACIENTE");
            stmt.setInt(5, paciente.getDoenca().getId());

            stmt.executeUpdate();

            try (ResultSet rs = stmt.getGeneratedKeys()) {
                if (rs.next()) {
                    paciente.setId(rs.getInt(1));
                }
            }

        } catch (SQLException e) {
            throw new RuntimeException("Erro ao salvar paciente.", e);
        }
    }

    public List<Paciente> listarTodos() {
```

```

String sql = "SELECT p.*, d.tipo as doenca_tipo, d.sintomas, d.restricoes
" +
    "FROM hospital.pessoa p " +
    "LEFT JOIN hospital.doenca d ON p.doenca_id = d.id " +
    "WHERE p.tipo_pessoa = 'PACIENTE'";

List<Paciente> pacientes = new ArrayList<>();

try (Connection conn = ConnectionFactory.getConnection();
    PreparedStatement stmt = conn.prepareStatement(sql);
    ResultSet rs = stmt.executeQuery()) {

    while (rs.next()) {

        Paciente paciente = new Paciente();
        paciente.setId(rs.getInt("id"));
        paciente.setNome(rs.getString("nome"));
        paciente.setCpf(rs.getInt("cpf"));
        paciente.setIdade(rs.getInt("idade"));

        int doencaId = rs.getInt("doenca_id");
        if (!rs.isNull()) {
            Doenca doenca = new Doenca();
            doenca.setId(doencaId);
            doenca.setTipo(rs.getString("doenca_tipo"));
            doenca.setSint(rs.getString("sintomas"));
            doenca.setRestr(rs.getString("restricoes"));
            paciente.setDoenca(doenca);
        }

        pacientes.add(paciente);
    }
} catch (Exception e) {

```

```

        throw new RuntimeException("Erro ao listar pacientes.", e);
    }

    return pacientes;
}

public Paciente buscarPorId(int id) {
    String sql = "SELECT p.*, d.tipo as doenca_tipo, d.sintomas,
d.restricoes " +
                "FROM hospital.pessoa p " +
                "LEFT JOIN hospital.doenca d ON p.doenca_id = d.id " +
                "WHERE p.id = ? AND p.tipo_pessoa = 'PACIENTE'";

    try (Connection conn = ConnectionFactory.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, id);

        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                Paciente paciente = new Paciente();
                paciente.setId(rs.getInt("id"));
                paciente.setNome(rs.getString("nome"));
                paciente.setCpf(rs.getInt("cpf"));
                paciente.setIdade(rs.getInt("idade"));

                int doencaId = rs.getInt("doenca_id");
                if (!rs.isNull()) {
                    Doenca doenca = new Doenca();
                    doenca.setId(doencaId);
                    doenca.setTipo(rs.getString("doenca_tipo"));
                    doenca.setSint(rs.getString("sintomas"));
                    doenca.setRestr(rs.getString("restricoes"));
                    paciente.setDoenca(doenca);
                }
                return paciente;
            }
        }
    }
}

```

```

        }
    } catch (Exception e) {
        throw new RuntimeException("Erro ao buscar paciente por ID.", e);
    }

    return null;
}

public boolean deletar(int id) {
    String sql = "DELETE FROM hospital.pessoa WHERE id = ? AND
tipo_pessoa = 'PACIENTE'";

    try (Connection conn = ConnectionFactory.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setInt(1, id);

        int linhasAfetadas = stmt.executeUpdate();

        return linhasAfetadas > 0;

    } catch (SQLException e) {
        throw new RuntimeException("Erro ao deletar paciente.", e);
    }
}

public boolean atualizarRestricoes(int pacienteId, String
novasRestricoes) {

    String sql = "UPDATE hospital.doenca SET restricoes = ? WHERE id = "
+
        "(SELECT doenca_id FROM hospital.pessoa WHERE id = ? AND
tipo_pessoa = 'PACIENTE')";

    try (Connection conn = ConnectionFactory.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {

```

```

        stmt.setString(1, novasRestricoes);
        stmt.setInt(2, pacienteId);

        int linhasAfetadas = stmt.executeUpdate();

        return linhasAfetadas > 0;

    } catch (SQLException e) {
        throw new RuntimeException("Erro ao atualizar restrições do
paciente.", e);
    }
}

```

Outra boa prática que nós utilizamos no desenvolvimento do nosso projeto, Foi o Princípio Aberto/Fechado (OCP). O Princípio Aberto/Fechado afirma que as entidades de software (classes, módulos, funções, etc.) devem ser abertas para extensão, mas fechadas para modificação. Isso significa que o comportamento de um módulo pode ser estendido sem a necessidade de modificar seu código-fonte existente.

O projeto utiliza uma hierarquia de classes com a classe abstrata Humano e suas subclasses Paciente , Medico e Estagiario . A classe Humano define atributos e comportamentos comuns a todos os tipos de pessoas no sistema. Se, no futuro, for necessário adicionar um novo tipo de "humano" (por exemplo, um "Enfermeiro"), basta criar uma nova classe Enfermeiro que estenda Humano . Não é necessário modificar a classe Humano existente ou as classes Paciente , Medico e Estagiario . Comprovação (Trecho de Código - Humano.java e Paciente.java):

Java

```
public abstract class Humano {
    private String nome;
    private int cpf;
    private int idade;
    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }
    public int getCpf() { return cpf; }
    public void setCpf(int cpf) { this.cpf = cpf; }
    public int getIdade() { return idade; }
    public void setIdade(int idade) { this.idade = idade; }
}
```

Paciente.java (Subclasse de Humano)

```
public class Paciente extends Humano {
    private int id;
    private Doenca doenca;
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public Doenca getDoenca() { return doenca; }
    public void setDoenca(Doenca doenca) { this.doenca = doenca; }
}
```

3.5 Banco de Dados

	id [PK] integer	nome character varying (255)	cpf integer	idade integer	tipo_pessoa character varying (20)	doenca_id integer	espec character varying (100)	instituicao_ensino character varying (150)	periodo integer	crm integer	salario numeric (10,2)	curso character varying (255)
1	4	Ana Silva	111222333	28	PACIENTE		[null]	[null]	[null]	[null]	[null]	[null]
2	5	Bruno Costa	444555666	45	PACIENTE		[null]	[null]	[null]	[null]	[null]	[null]
3	6	Carlos de Andrade	777888999	62	PACIENTE		[null]	[null]	[null]	[null]	[null]	[null]
4	7	Daniela Martins	123123123	35	PACIENTE		[null]	[null]	[null]	[null]	[null]	[null]
5	8	Eduarda Farias	456456456	19	PACIENTE		[null]	[null]	[null]	[null]	[null]	[null]
6	10	Cleber	429353638	42	PACIENTE		[null]	[null]	[null]	[null]	[null]	[null]
7	12	Diego	15897548	24	MEDICO		[null]	[null]	[null]	1259	2599.00	[null]
8	13	Leticia	47895214	58	MEDICO		[null]	[null]	[null]	6987	12500.00	[null]
9	14	Elton	58236749	44	MEDICO		[null]	[null]	[null]	7411	10200.00	[null]
10	15	Mel	55284310	29	MEDICO		[null]	[null]	[null]	1278	5500.00	[null]
11	16	Bruno	96102998	41	MEDICO		[null]	[null]	[null]	6924	21000.00	[null]
12	50	Martins	45207381	22	ESTAGIARIO		[null]	[null]	Unoceste	7	[null]	Biomedicina
13	51	Luana	45003817	21	ESTAGIARIO		[null]	[null]	Unimar	5	[null]	Farmacia
14	52	Gustavo	44119021	23	ESTAGIARIO		[null]	[null]	Santa Cecilia	6	[null]	Medicina
15	53	Jose	55033217	19	ESTAGIARIO		[null]	[null]	Univem	3	[null]	Biomedicina
16	54	Juliana	69250741	21	ESTAGIARIO		[null]	[null]	PUC	4	[null]	Medicina
17	57	Sophia	45781205	21	PACIENTE		[null]	[null]	[null]	[null]	[null]	[null]

Tela Banco de Dados - Tabela Humano

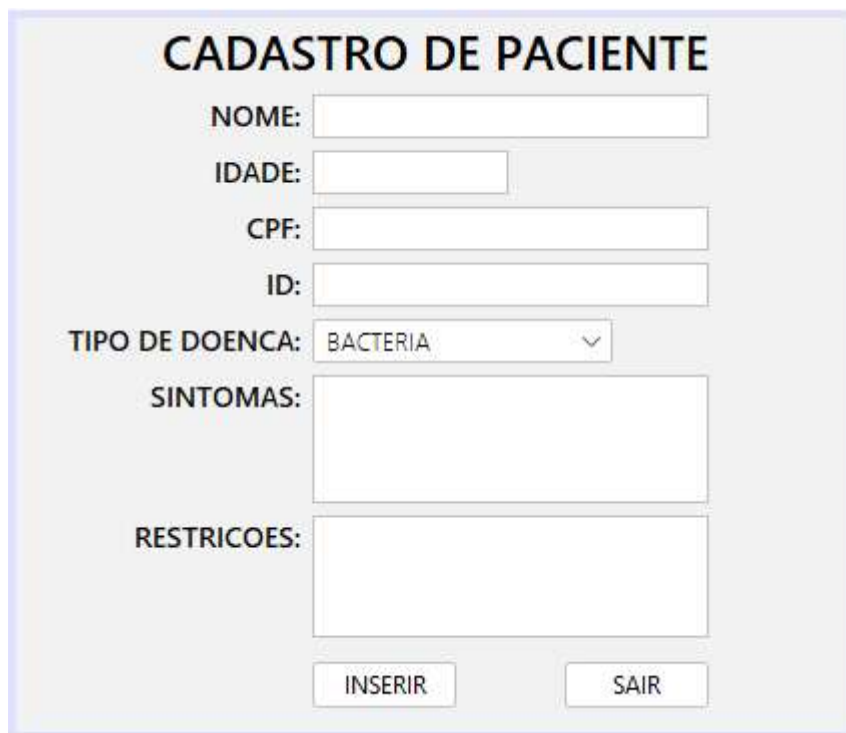
3.6 Telas do sistema



Tela inicial - HospitAll

The image shows a web application window titled "Paciente". The main content area has a light gray background and features the title "MENU PACIENTE" in bold uppercase letters. Below the title is a label "ID:" followed by a text input field. Underneath the input field is a button labeled "CADASTRAR PACIENTE". Below this button are four buttons arranged in a 2x2 grid: "CONSULTAR PACIENTE", "ALTERAR RESTRICOES", "APRESENTAR PACIENTES", and "DELETAR PACIENTE". At the bottom right of the window is a button labeled "SAIR".

Tela Paciente - HospitAll



CADASTRO DE PACIENTE

NOME:

IDADE:

CPF:

ID:

TIPO DE DOENÇA: BACTERIA ▼

SINTOMAS:

RESTRICOES:

INSERIR SAIR

Tela Cadastro de Paciente - HospitAll

3.7 Link do repositório

GitHub: <https://github.com/matheustm29/HospitAll>

Link do Vídeo:

<https://drive.google.com/file/d/1SiwKQHFEpGEx2AGNEzGOZzMSQuIXzNuj/view?usp=sharing>

4 CONCLUSÕES

O desenvolvimento do projeto "HospitAll" permitiu aplicar na prática os conceitos fundamentais da Programação Orientada a Objetos em um cenário realista. A implementação dos requisitos funcionais, como cadastro de entidades e geração de relatórios, foi concluída com sucesso.

REFERÊNCIAS

DEITEL, Paul; DEITEL, Harvey. *Java: como programar*. 10. ed. São Paulo: Pearson Education do Brasil, 2016.

LARMAN, Craig. *Utilizando UML e padrões: uma introdução à análise e ao projeto orientado a objetos e ao processo unificado*. 3. ed. Porto Alegre: Bookman, 2007.

HORSTMANN, Cay S. *Padrões de projeto orientados a objetos*. 2. ed. Rio de Janeiro: LTC, 2006.