Saphira: Sistema de gerenciamento de cinema

Miguel Lipreri Arnoldi*
Nícolas André Basotti **
José Matheus S. Lopes***

Resumo

"Saphira" representa um sistema de gerenciamento de cinema, integrado às disciplinas de Engenharia de Software I, Banco de Dados II e Programação II. Este projeto engloba funcionalidades essenciais, incluindo scripts para criação da base de dados e modelagem relacional, proporcionando uma gestão eficiente de informações no contexto cinematográfico. A utilização de Diagramas UML e scripts SQL contribuem para uma visão organizada do sistema, enquanto o dicionário de dados oferece uma referência detalhada para facilitar a compreensão e manipulação dos dados.

Palavras-chave: Saphira. Sistema de gerenciamento de cinema. Funcionalidades.

*Discente do Curso de Ciência da Computação
Unoesc-Campus de São Miguel do Oeste
Rua Lázaro da Costa, 167. São Miguel do Oeste-SC
miguellipreriarnold@gmail.com

**Discente do Curso de Ciência da Computação
Unoesc-Campus de São Miguel do Oeste
Rua XV Novembro, 1190. São Miguel do Oeste-SC
basottin@gmail.com

***Discente do Curso de Ciência da Computação
Unoesc-Campus de São Miguel do Oeste
Rua Almirante Barroso, 1306. São Miguel do Oeste-SC
josedocafe@gmail.com

1 INTRODUÇÃO

A evolução tecnológica tem desempenhado um papel fundamental na otimização de diversos setores, e o gerenciamento eficiente de cinemas não é exceção. Com o objetivo de facilitar e automatizar as operações cotidianas, visando um aproveitamento mais eficaz do tempo e uma organização mais precisa, surge a necessidade de um sistema de gerenciamento de cinema inovador.

Este trabalho tem como foco principal a criação de um software abrangente que utiliza princípios de engenharia de software, banco de dados e programação para aprimorar a gestão de cinemas. Denominado Saphira, nosso sistema emprega Diagramas UML para ilustrar, de maneira clara , o seu funcionamento. Além disso, a metodologia inclui a implementação de um banco de dados, com scripts para a criação da base de dados e modelagem relacional. Integramos o banco de dados ao sistema por meio do JDBC, aprimorando ainda mais a eficiência e a conectividade do Saphira.

A proposta é clara: criar uma solução que simplifica a forma como os cinemas são gerenciados. Através da automação de tarefas, busca-se proporcionar aos gestores de cinemas a capacidade de otimizar processos, ganhar tempo e promover uma gestão mais eficiente e organizada.

Este trabalho explora detalhadamente o desenvolvimento do Saphira, destacando como a integração de princípios de engenharia de software, banco de dados e programação pode transformar a dinâmica de gerenciamento em cinemas. A análise inclui desde os Diagramas UML, que esclarecem a estrutura do sistema, até a implementação prática do banco de dados e programação que sustentam esse sistema.

Ao longo deste artigo, vamos detalhar os elementos técnicos do Saphira. Com uma abordagem clara e objetiva, buscamos não apenas apresentar um sistema de gerenciamento de cinema, mas também destacar suas funcionalidades.

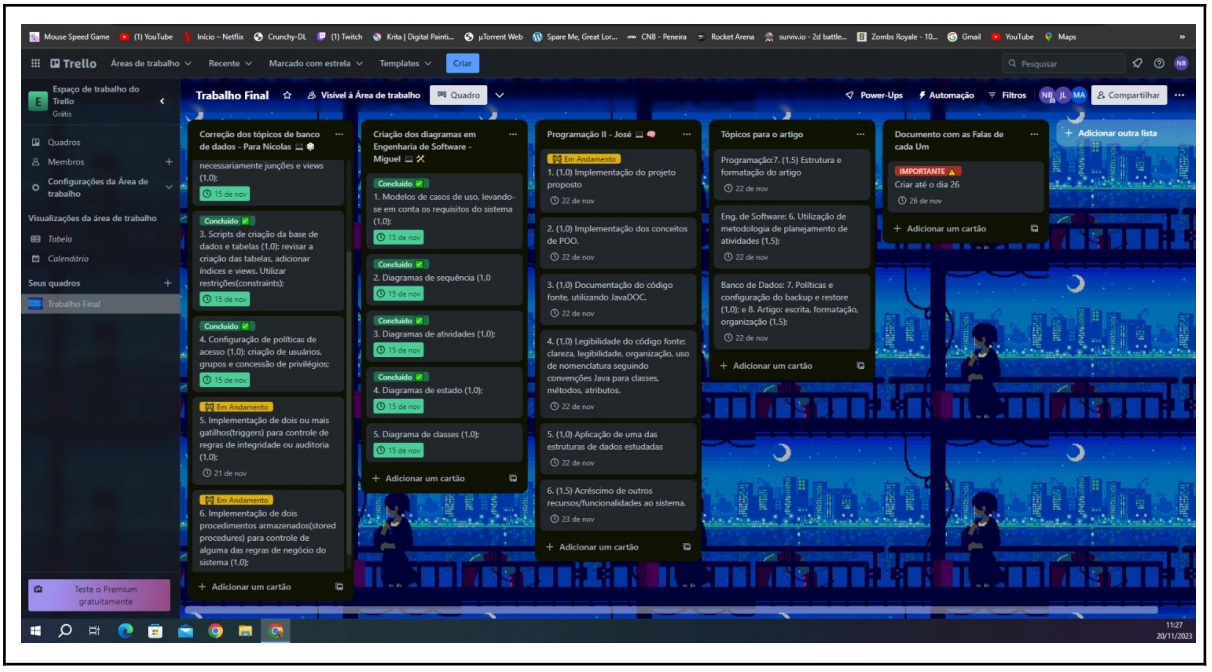
2 DESENVOLVIMENTO

2.1 METODOLOGIA DE PLANEJAMENTO DE ATIVIDADE

A Engenharia de Software desempenha um papel fundamental no desenvolvimento de sistemas de software eficientes e confiáveis. Uma parte crucial desse processo é a utilização de metodologias de planejamento de atividades para garantir uma abordagem sistemática e organizada. No contexto do trabalho em questão, a metodologia de planejamento de atividades escolhida foi o Trello.

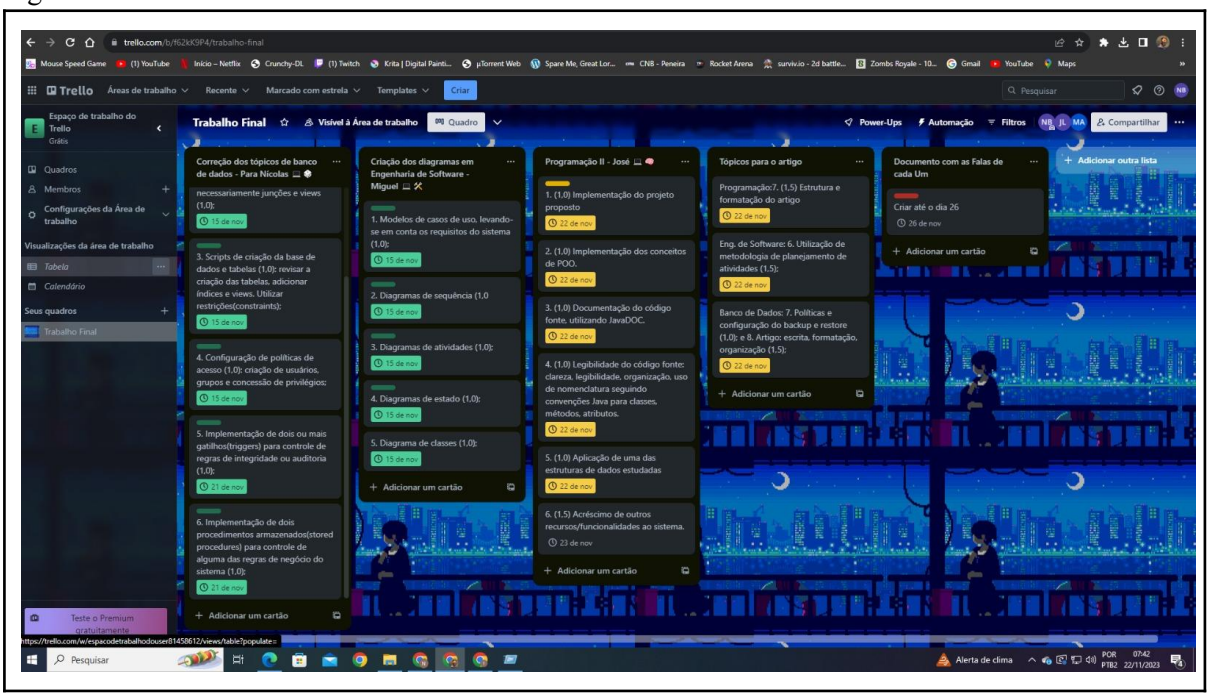
Trello é uma ferramenta eficaz de gerenciamento de projetos que faz uso dos conceitos de quadros, listas e cartões para organizar tarefas e atividades. Nosso processo de planejamento pode ser visualizado nas Figuras 1, 2 e 3. Essas imagens fornecem uma representação visual do desenvolvimento e organização das atividades ao longo do tempo, facilitando uma compreensão do progresso do projeto.

Figura 1



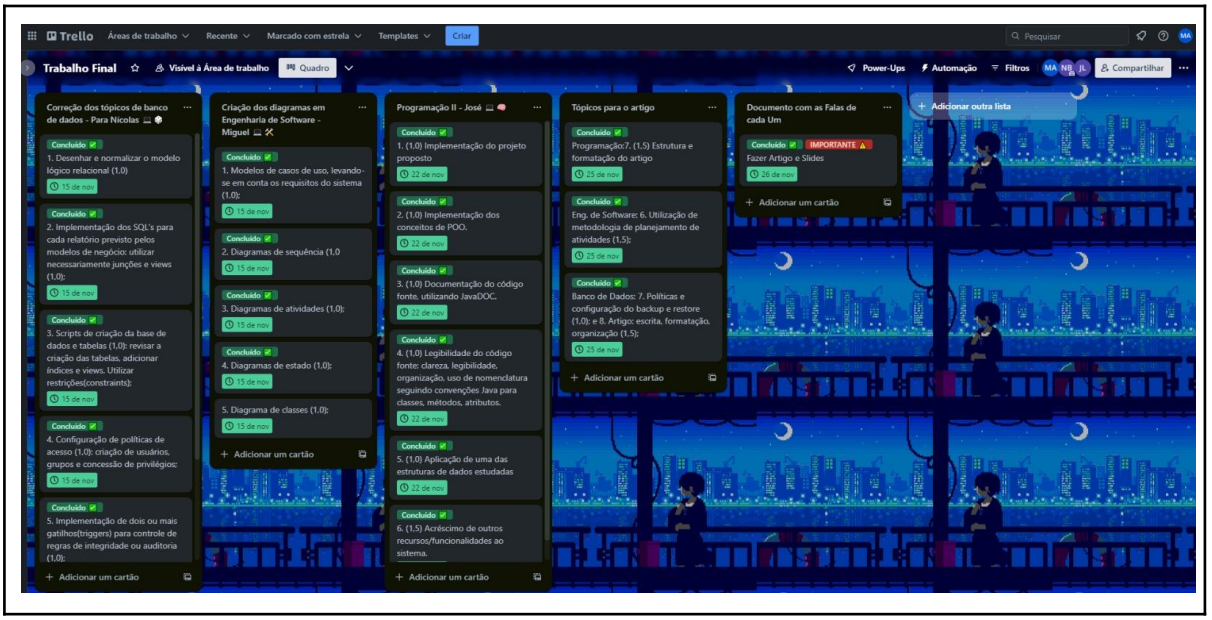
Fonte: Os autores (2023).

Figura 2



Fonte: Os autores (2023).

Figura 3



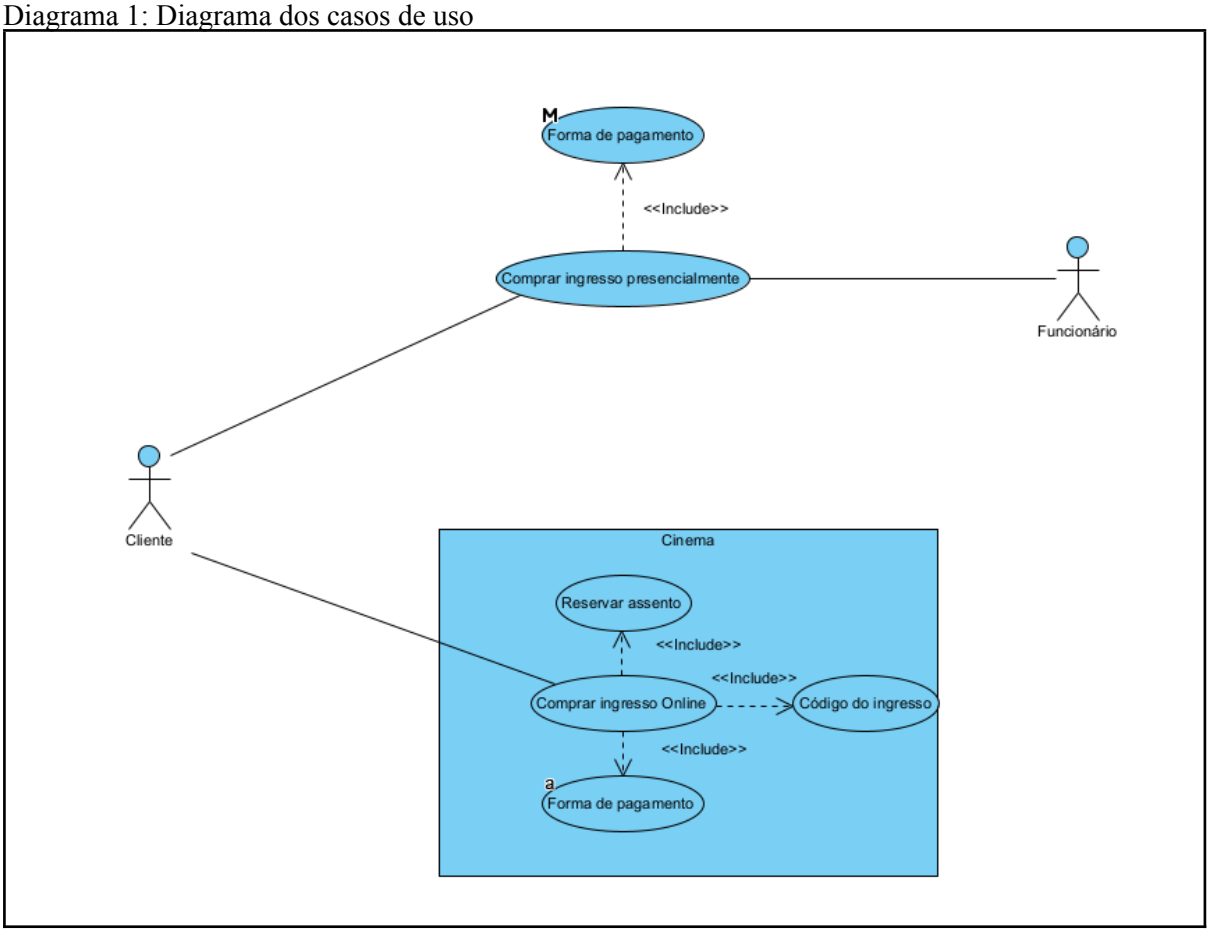
Fonte: Os autores (2023).

2.2 MODELOS UML

A modelagem utilizando a Linguagem de Modelagem Unificada (UML) é uma prática comum no desenvolvimento de software para representar visualmente a estrutura e o comportamento de um sistema.

Com os projetos e objetivos estabelecidos, foi desenvolvida a modelagem que ilustra o processo de venda de ingressos. O Diagrama 1 apresenta o caso de uso.

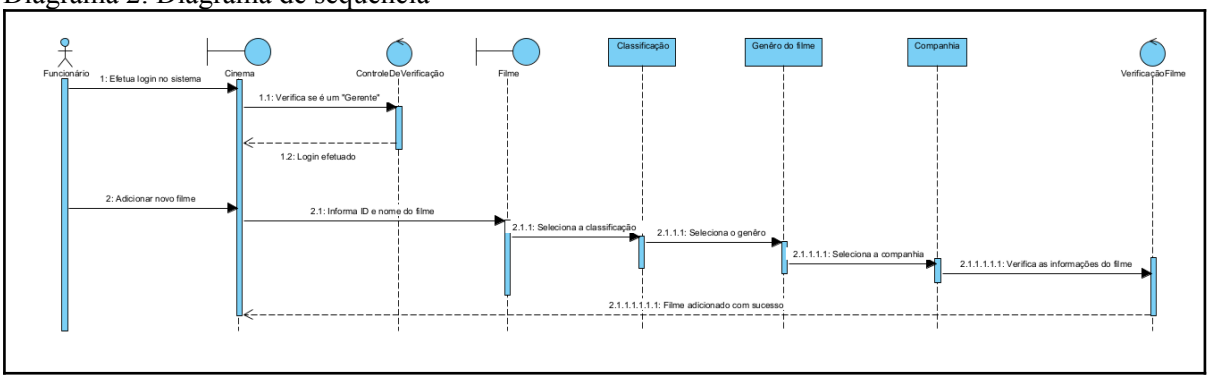
Diagrama 1: Diagrama dos casos de uso



Fonte: Os autores (2023).

O Diagrama 2 representa um diagrama de sequência que ilustra como os objetos operam um com os outros e em que ordem. Este diagrama específico foca no processo de adição de um filme.

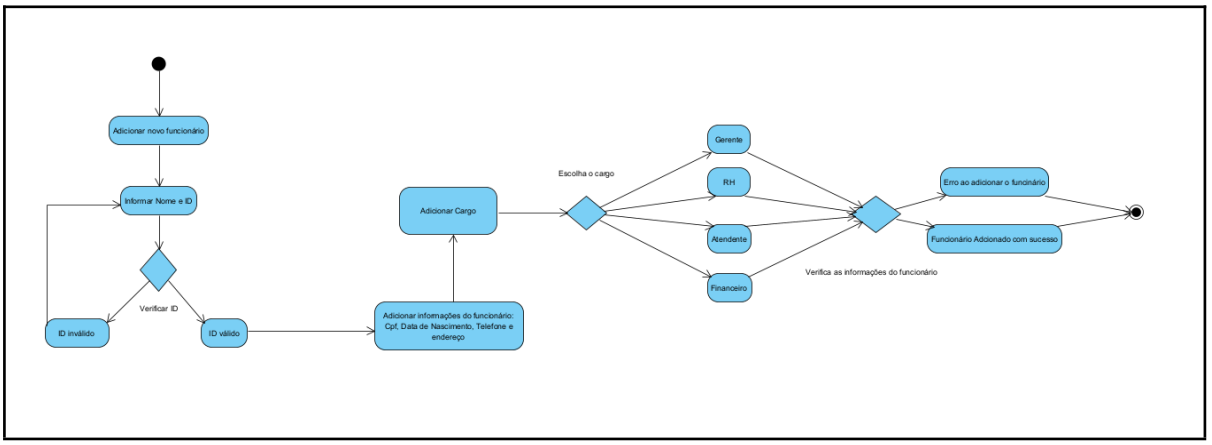
Diagrama 2: Diagrama de sequência



Fonte: Os autores (2023).

O Diagrama 3 representa um diagrama de atividade, uma ferramenta útil para modelar o fluxo em um sistema. Ele é eficaz na visualização da lógica sequencial e concorrente das atividades dentro de um processo. Este diagrama específico foca no processo de adição de um novo funcionário, ilustrando como as atividades ocorrem e se relacionam ao longo do tempo.

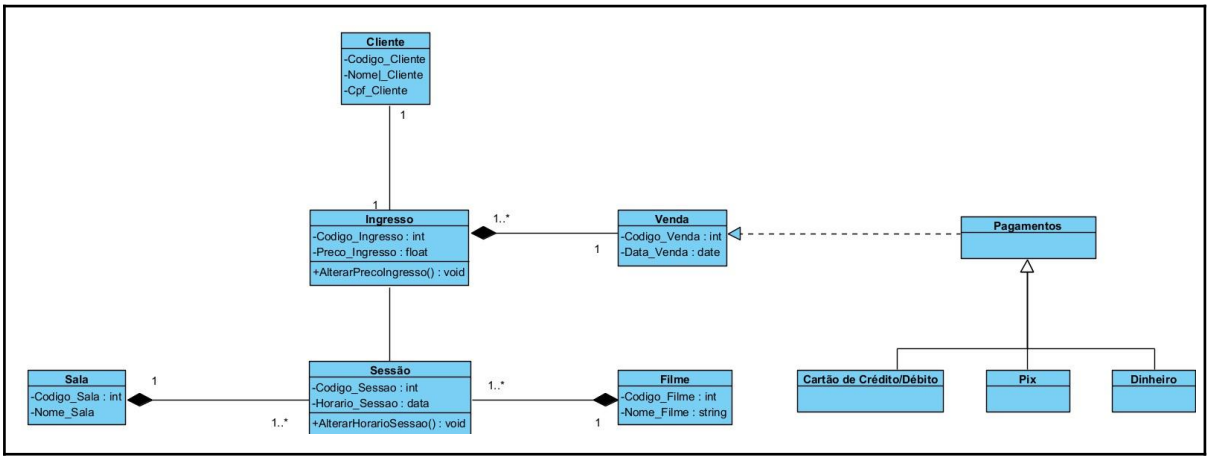
Diagrama 3: Diagrama de atividade



Fonte: Os autores (2023).

Diagrama 4 apresenta um diagrama de classes, proporcionando uma visão geral do sistema de software ao exibir classes, atributos, operações e seus relacionamentos. Esse diagrama específico destaca o processo de venda de ingressos, oferecendo uma representação visual das classes envolvidas, seus atributos e como elas interagem no contexto da venda de ingressos.

Diagrama 4: Diagrama de classes

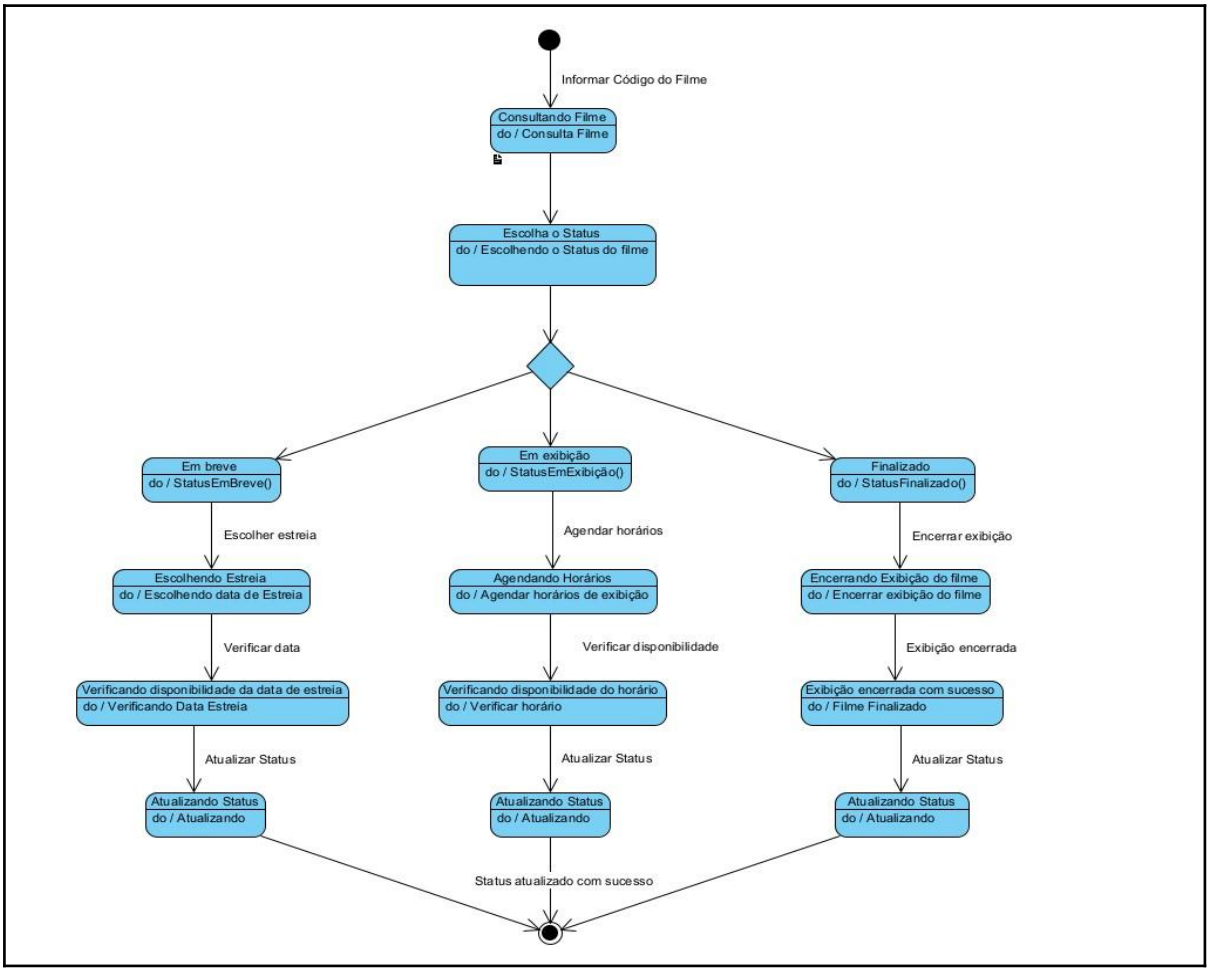


Fonte: Os autores (2023).

O Diagrama 5 é um diagrama de estado que visualiza como ocorrem as transições entre diferentes estados para a escolha de status de um filme. Esse diagrama oferece uma representação gráfica do ciclo de vida do status de um filme, destacando os estados de "Em Breve", "Em Exibição" e "Finalizado".

- **Em Breve:**
    - Representa o estado inicial, indicando que o filme está programado para ser exibido em breve.
- **Em Exibição:**
    - Indica o estado atual do filme, sinalizando que ele está sendo atualmente exibido nos cinemas.
- **Finalizado:**
    - Representa o estado final, indicando que o filme encerrou seu período de exibição e não está mais em cartaz.

Diagrama 5: Diagrama de estado



Fonte: Os autores (2023).

2.3 REQUISITOS

No desenvolvimento da Saphira, é essencial estabelecer requisitos claros e abrangentes que moldaram a estrutura e funcionalidade do banco de dados associado. A seguir, apresentamos os Requisitos Funcionais e Não Funcionais.

- **Requisitos funcionais**
    - Gerenciamento de vendas de ingressos.
    - Adição de filmes de forma simples.
    - Administração de funcionários e sessões de trabalho.
    - Controle de Estoque.
- **Requisitos não funcionais**
    - Desempenho.
    - Escalabilidade.
    - Usabilidade.
    - Segurança robusta.
    - Manutenção simples.

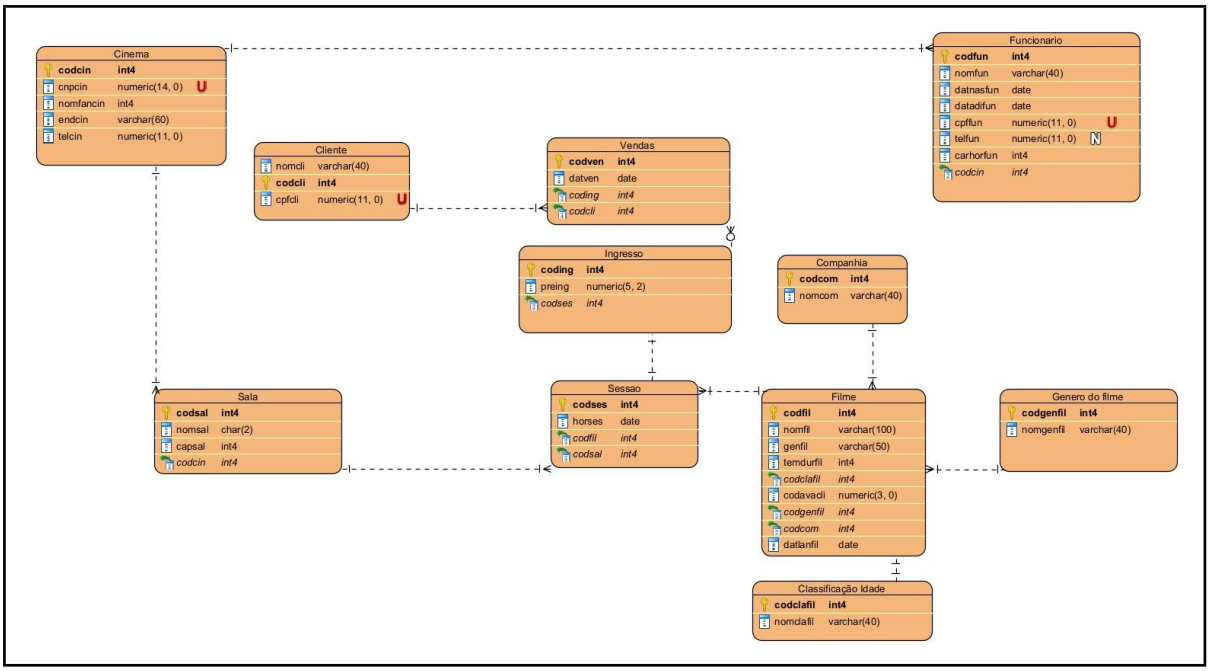
2.3 MODELO LÓGICO RELACIONAL E DICIONÁRIO DE DADOS

Com os requisitos prontos, iniciamos a criação do banco de dados para nosso sistema de gerenciamento de cinema com dois componentes essenciais: o Modelo Lógico Relacional e o Dicionário de Dados.

O Modelo Lógico Relacional (Figura 4: Modelo Lógico Relacional) é como um mapa que mostra as partes importantes do nosso sistema, como filmes, clientes e ingressos, e como elas se conectam. Isso nos ajuda a organizar e entender melhor como as informações estão relacionadas, tornando mais fácil encontrar e usar os dados quando precisamos.

Paralelamente, desenvolvemos o Dicionário de Dados, apresentado na Figura 5: Dicionário de Dados. Este dicionário é como um guia detalhado que explica o significado de cada parte do nosso sistema. Por exemplo, diz que tipo de informação está guardada em cada tabela e quais regras precisamos seguir. É uma referência importante para quem vai trabalhar no sistema, garantindo que todos estejam na mesma página sobre como as coisas funcionam.

Figura 4: Modelo Lógico Relacional



Fonte: Os autores (2023).

Figura 5: Dicionário de Dado

1. Data Dictionary

Entity Name	Entity Description					
Column Name	**Column Description**	**Data Type**	**Length**	**Primary Key**	**Nullable**	**Unique**
Cinema						
cnpcin	Cnpj da empresa	numeric	14	false	false	true
codcin	Código do cinema	int4	10	true	false	false
endcin	Endereço do cinema	varchar	60	false	false	false
nomfancin	Nome fantasia do cinema	int4	40	false	false	false
telcin	Telefone do cinema com DDD: exemplo - (49) 9****-****	numeric	11	false	false	false
Classificação Idade						
codclafil	Código da classificação do filme	int4	10	true	false	false
nomclafil	Nome da classificação do filme	varchar	40	false	false	false
Cliente						
codcli	Código do cliente	int4	10	true	false	false
cpfcli	CPF do cliente	numeric	11	false	false	true
nomcli	Nome cliente	varchar	40	false	false	false
Companhia						
codcom	Código da Companhia do Filme	int4	10	true	false	false
nomcom	Nome da Companhia do Filme	varchar	40	false	false	false
Filme						
codavacli		numeric	3	false	false	false
codclafil		int4	10	false	false	false
codcom		int4	10	false	false	false
codfil	Código do Filme	int4	10	true	false	false
codgenfil		int4	10	false	false	false
datlanfil	Data de lançamento do filme	date	0	false	false	false
genfil	Gênero do Filme	varchar	50	false	false	false
nomfil	Nome do filme	varchar	100	false	false	false
temdurfil	Tempo de duração do filme	int4	10	false	false	false
Funcionario						

Entity Name	Entity Description					
carhorfun	Carga horária do funcionário	int4	10	false	false	false
codcin		int4	10	false	false	false
codfun	Código do Funcionário	int4	10	true	false	false
cpffun	CPF do funcionário	numeric	11	false	false	true
datadifun	Data de adimissão do funcionário	date	0	false	false	false
datnasfun	Data de nascimento do funcionário	date	0	false	false	false
nomfun	Nome do Funcionário	varchar	40	false	false	false
telfun	Telefone do funcionário	numeric	11	false	true	false
Genero do filme						
codgenfil	Código do gênero do filme	int4	10	true	false	false
nomgenfil		varchar	40	false	false	false
Ingresso						
coding	Código do ingresso	int4	10	true	false	false
codses		int4	10	false	false	false
preing	Preço do ingresso	numeric	5.2	false	false	false
Sala						
capsal	Capacidade da Sala	int4	10	false	false	false
codcin		int4	10	false	false	false
codsal	Código da Sala do Cinema	int4	10	true	false	false
nomsal	Nome da Sala. Exemplo: A1, A2..B1,B2..	char	2	false	false	false
Sessao						
codfil		int4	10	false	false	false
codsal		int4	10	false	false	false
codses	Código da sessão	int4	10	true	false	false
horses	Horário da sessão	date	0	false	false	false
Vendas						
codcli		int4	10	false	false	false
coding		int4	10	false	false	false
codven	Código da venda	int4	10	true	false	false
datven	Data da venda do ingresso	date	0	false	false	false

Fonte: Os autores (2023).

2.4 ÍNDICES E VIEWS

No desenvolvimento do nosso sistema, incorporamos views e índices para aprimorar a eficiência das consultas e otimizar o desempenho do banco de dados, utilizando o PostgreSQL. Esses elementos são essenciais para tornar as operações de busca e manipulação de dados mais rápidas e diretas. A Figura 6: View e a Figura 7: Índices fornece uma visualização desses componentes em nosso projeto.

Figura 6: View

```
--1) Relação do nome do filme, tempo de
--duração e o gênero de todos os filmes.
--Ordene o relatório do filme mais
--longo(tempo) para o filme mais curto;
create view vw_listaFilmes as
select
        f.nomfil "Nome do filme",
        f.temdurfil "Tempo de duração",
        g.nomgenfil "Genero"
from
        filme f
inner join  GeneroDoFilme g on
        f.codgenfil = g.codgenfil
order by
        f.temdurfil desc;
```

Fonte: Os autores (2023).

Figura 7: Índices

```
-- Index para chaves secundárias
create index idx_nomeClassicacao_sk
        on ClassificacaoIdade(nomclafil);

create index idx_nomeCompanhia_sk
        on Companhia(nomcom);

create index idx_nomeCliente_sk
on Cliente(nomcli);

create index idx_nomeFilme_sk
on Filme(nomfil);

create index idx_nomeFuncionario_sk
on Funcionario(nomfun);

create index idx_nomeGenero
on GeneroDoFilme(nomgenfil);
```

Fonte: Os autores (2023).

2.5 POLÍTICAS DE ACESSO

Na configuração de políticas de acesso, criamos grupos de usuários e concedemos privilégios específicos para assegurar a segurança e o controle de acesso ao banco de dados. A Figura 8: Grupos e Permissões ilustra esse processo.

Figura 8: Grupos e Permissões

```
create group gr_Gerente;

create group gr_Rh;

create group gr_Bilheteria;

create group gr_Financeiro;

Grant insert, select, delete on filme, sessao to gr_Gerente;

grant insert, select, delete on funcionario to gr_Rh;

grant select on filme, ingresso, sessao to gr_Bilheteria;

grant insert, select, delete on cliente, ingresso, vendas to gr_Financeiro;
```

Fonte: Os autores (2023).

2.6 FUNCTIONS

Em nossa implementação do sistema, introduzimos duas funções fundamentais para calcular a bilheteria, abordando perspectivas mensais e anuais. A função mensal desempenha um papel essencial ao oferecer uma análise detalhada do desempenho a cada mês, proporcionando uma percepção importante sobre os períodos de maior movimento. Por outro lado, a função anual oferece uma visão abrangente do desempenho financeiro ao longo do tempo. As Figuras 9 e 10: "Function Mês" e "Function Ano" ilustram essas implementações.

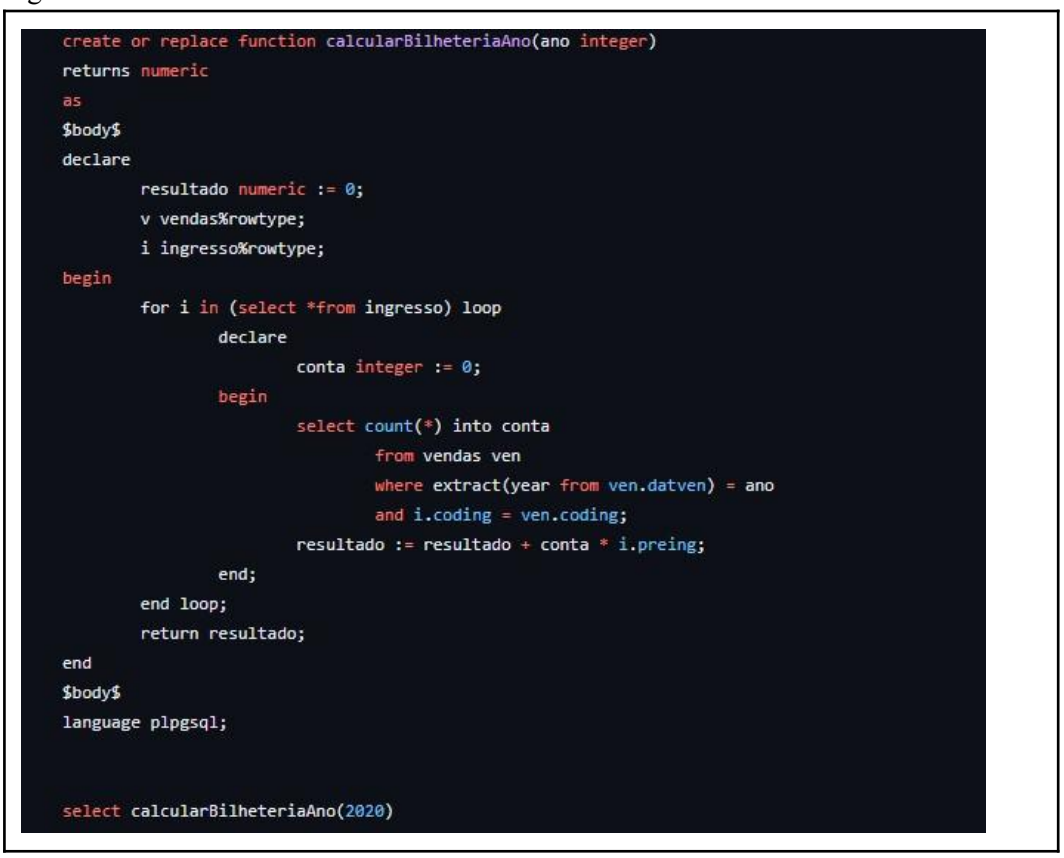
Figura 9: Function Mês

```
create or replace function calcularBilheteriaMes(mes integer, ano integer)
returns numeric
as
$body$
declare
        resultado numeric := 0;
        v vendas%rowtype;
        i ingresso%rowtype;
begin
        for i in (select *from ingresso) loop
                declare
                        conta integer := 0;
                begin
                        select count(*) into conta
                                from vendas ven
                                where extract(month from ven.datven) = mes
                                and extract(year from ven.datven) = ano
                                and i.coding = ven.coding;
                        resultado := resultado + conta * i.preing;
                end;
        end loop;
        return resultado;
end
$body$
language plpgsql;

select calcularBilheteriaMes(7,2023);
```

Fonte: Os autores (2023).

Figura 10: Function Ano



```
create or replace function calcularBilheteriaAno(ano integer)
returns numeric
as
$body$
declare
        resultado numeric := 0;
        v vendas%rowtype;
        i ingresso%rowtype;
begin
        for i in (select *from ingresso) loop
                declare
                        conta integer := 0;
                begin
                        select count(*) into conta
                                from vendas ven
                                where extract(year from ven.datven) = ano
                                and i.coding = ven.coding;
                        resultado := resultado + conta * i.preing;
                end;
        end loop;
        return resultado;
end
$body$
language plpgsql;


select calcularBilheteriaAno(2020)
```

Fonte: Os autores (2023).

2.7 TRIGGERS

Na implementação do nosso sistema, incorporamos triggers para simplificar operações específicas. Criamos dois triggers distintos: um para verificar a capacidade, garantindo que não seja ultrapassado um limite predeterminado, e outro para a data de lançamento, assegurando que os registros cumpram critérios temporais específicos. A Figuras 11 e 12: “Capacidade sala” e “Data de lançamento” fornece uma visualização desses triggers em nosso projeto.

Figura 11: Capacidade sala

```
create or replace function verifica_capacidade()
returns trigger
as
'
declare
        capacidadeAtual integer;
        capacidadeMaxima integer;
begin
        select capsal into capacidadeAtual from sala where codsal = new.codsal;
        select count(*) into capacidadeMaxima from ingresso where codses = new.codses;

        if capacidadeMaxima = capacidadeAtual then
        raise exception ''capacidade da sala atingida!'';
    elsif capacidadeMaxima > capacidadeAtual then
        raise exception ''capacidade da sala excedida!'';
    end if;
   return new;
end;
'
language plpgsql;


create trigger controle_capacidade
before insert on vendas
for each row
execute function verifica_capacidade();
```

Fonte: Os autores (2023).

Figura 12: Data de lançamento

```
CREATE OR REPLACE FUNCTION verifica_data_lancamento()
RETURNS TRIGGER AS
'
BEGIN
    IF NEW.horses < (SELECT datlanfil FROM Filme WHERE codfil = NEW.codfil) THEN
        RAISE EXCEPTION ''A data da sessão é anterior à data de lançamento do filme!'';
    END IF;

    RETURN new;
end;
'
LANGUAGE plpgsql;

create trigger controle_data_lancamento
before insert on sessao
for each row
execute function verifica_data_lancamento();
```

Fonte: Os autores (2023).

2.8 SISTEMA

Na etapa final do desenvolvimento do sistema, escolhemos Java como linguagem principal, aplicando os princípios da Programação Orientada a Objetos (POO) para estruturar e organizar nosso código de maneira coesa. Essa decisão resultou em uma implementação mais flexível e de fácil compreensão. Além disso, para garantir a clareza e documentação eficaz do código-fonte, utilizamos JavaDOC, facilitando a compreensão do sistema e permitindo uma manutenção mais eficiente no futuro.

2.9 CLASSES

No desenvolvimento das classes do sistema, escolhemos incorporar a biblioteca Lombok para simplificar e agilizar o código. Ao optar pelo Lombok, conseguimos reduzir substancialmente a quantidade de código, eliminando a necessidade de escrever getters, setters e métodos de toString, entre outros. Essa escolha não apenas simplificou a implementação das classes, mas também resultou em um código mais claro e fácil de ler. A Figura 13: Lombok ilustra visualmente como essa abordagem contribui para a simplicidade e eficiência no desenvolvimento do sistema.

A Figura 13: Lombok

```java
package br.com.cinema.saphira.model;

import jakarta.persistence.Column;

@Entity
@Data
public class Ingresso {
    @Id
    @Column(name = "coding")
    private int codigoIngresso;

    @Column(name = "preing", nullable = false)
    private double precoIngresso;

    @OneToOne
    @JoinColumn(name = "codses", nullable = false)
    private int codigoSessao;
}

```

Fonte: Os autores (2023).

2.10 CONTROLLERS

Desenvolvemos repositórios dedicados para cada classe, e, com base nesses repositórios, criamos controladores que mapeiam e coordenam suas funções específicas. Essa abordagem proporcionou uma organização eficiente do sistema, permitindo o acesso e a manipulação de dados de forma estruturada e flexível. A Figura 14: Repositório e a Figura 15: Controllers ilustra visualmente como esse processo completo contribui para a gestão eficaz das operações no sistema.

Figura 14: Repositório

```java
package br.com.cinema.saphira.repository;

import org.springframework.data.jpa.repository.JpaRepository;

public interface CinemaRepository extends JpaRepository<Cinema,Integer> {

}

```

Fonte: Os autores (2023).

Figura 15: Controllers

```java
@RestController
@RequestMapping("/Cinema")
public class CinemaControllers {
    @Autowired
    private CinemaRepository cinemaRepository;

    @PostMapping(value = "salvar")
    @ResponseBody
    public ResponseEntity<Cinema> salvar(@RequestBody Cinema cinema) {
        Cinema cine = cinemaRepository.save(cinema);
        return new ResponseEntity<Cinema>(cine, HttpStatus.CREATED);
    }
```

Fonte: Os autores (2023).

3 CONCLUSÃO

Em conclusão, o Saphira é uma solução completa para aprimorar a gestão de cinemas. A integração de engenharia de software, banco de dados e programação oferece ferramentas eficientes para automatizar tarefas, economizar tempo e promover uma administração mais eficiente. A clareza na apresentação técnica ao longo do artigo reflete nosso compromisso em destacar as funcionalidades do sistema de maneira acessível. O Saphira é uma resposta eficaz para os desafios da gestão cinematográfica.