UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vinícius Calixto Rocha

Uma aplicação Web para auxiliar o fluxo de estágios da FACOM: Detalhamento de sua arquitetura, dos fluxos e da máquina de estados

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Vinícius Calixto Rocha

Uma aplicação Web para auxiliar o fluxo de estágios da FACOM: Detalhamento de sua arquitetura, dos fluxos e da máquina de estados

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Alexsandro Santos Soares

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2023

Vinícius Calixto Rocha

Uma aplicação Web para auxiliar o fluxo de estágios da FACOM: Detalhamento de sua arquitetura, dos fluxos e da máquina de estados

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 01 de dezembro de 2023

Prof. Dr. Alexsandro Santos Soares
Orientador

Prof. Dr. Carlos Roberto Lopes

Profa. Dra. Christiane Regina Soares Brasil

Uberlândia, Brasil
2023

Dedico este trabalho à minha família, aos meus amigos e professores que, ao longo dos anos, apoiaram-me e não permitiram que eu desistisse deste caminho.

Agradecimentos

Primeiramente, agradeço a Deus pela capacidade e saúde para a realização deste trabalho.

Às pessoas da minha família adotiva que foram as principais a me incentivar a trilhar este caminho.

Às pessoas da minha família biológica, principalmente meu pai e minha avó, por estarem comigo e me apoiarem nos momentos mais críticos desta jornada.

A todo o corpo docente que tornou esta conquista possível, em especial, ao meu orientador, Alexsandro Santos Soares, que me ensinou e orientou com paciência, ânimo, empatia e amor pelo ensino.

Resumo

A vida cotidiana de um professor universitário frequentemente se revela complexa, visto que envolve múltiplas tarefas que ultrapassam as referências de ensino e de pesquisa. Isso inclui a administração de estágios, que, quando conduzida conforme os métodos tradicionais, pode se tornar repetitiva e trabalhosa. Diante desse cenário, neste trabalho, é detalhada uma aplicação Web responsiva, desenvolvida para centralizar, facilitar e automatizar as tarefas do corpo docente encarregado do acompanhamento dos estágios na Faculdade de Computação, o que envolve, também, os estagiários e seus orientadores. A aplicação é desacoplada considerando sua interface e suas regras de negócios, e parte de seu fluxo de funcionamento é estruturada como uma máquina de estados para proporcionar flexibilidade e automatizar várias etapas do processo de estágios, incluindo o envio de e-mails personalizados e a transição de estados por meio de eventos temporais que determinam o fluxo dos envolvidos nas etapas do estágio.

Palavras-chave: Aplicação Web, Arquitetura de software, Máquina de estados.

Lista de ilustrações

Lista de tabelas

Lista de Algoritmos

Lista de abreviaturas e siglas

API	*Application Programming Interface*
BCC	Bacharelado em Ciência da Computação
BSI	Bacharelado em Sistemas de Informação
CORS	*Cross-Origin Resource Sharing*
CRUD	*Create, Read, Update and Delete*
CSS	*Cascading Style Sheets*
FACOM	Faculdade de Computação
FIFO	*First In, First Out*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
MIME	*Multipurpose Internet Mail Extensions*
POO	Programação Orientada a Objetos
ORM	*Object–Relational Mapping*
OSI	*Open Systems Interconnection*
REST	*Representational State Transfer*
SMTP	*Simple Mail Transfer Protocol*
SPA	*Single-Page Application*
SSR	*Server-Side Rendering*
TCP	*Transmission Control Protocol*
UML	*Unified Modeling Language*

URI	*Uniform Resource Identifier*
URN	*Uniform Resource Name*
URL	*Uniform Resource Locator*
Web	*World Wide Web*

Sumário

1 Introdução

O cotidiano do coordenador de estágios, que adota métodos tradicionais e manuais em uma universidade, é desafiador, pois, muitas vezes, precisa conciliar suas atividades de pesquisa e de ensino com tarefas manuais constantes relacionadas às diversas fases do processo de estágio. Isso implica em responder repetidamente a perguntas dos alunos, por meio de e-mails, em lidar com assinaturas, autorizações e restrições de datas, todas elas baseadas em normas legais que são frequentemente alteradas, seguindo um fluxo padronizado e repetitivo, na maioria das vezes.

Conforme mencionado por Mansour (2019), a automação de procedimentos manuais apresenta diversos benefícios, como: a prevenção de erros humanos, a redução dos gastos operacionais, o aumento da satisfação dos participantes no processo, além da escalabilidade, entre outros. Portanto, desde que a implementação siga diretrizes sólidas de manutenção, flexibilidade e usabilidade, é altamente recomendável considerar a implementação de automação dos processos realizados pelo corpo docente.

Considerando os aspectos de flexibilidade e manutenibilidade de soluções, Emerson (2022) cita que o uso de máquinas de estados simplifica a criação e manutenção de fluxos complexos em aplicações. Essas aplicações podem ser modeladas em estados que são transitados, baseados em entradas de usuário e eventos, como pode ser definido o fluxo de estágios com os envolvidos. Além disso, é uma prática comum o uso do modelo orientado a eventos de máquinas de estados finitas para os diversos cenários de aplicações de software, principalmente quando existe a necessidade de realização de transições baseadas em eventos no tempo e nas entradas de usuário (WAGNER, 2006).

Outro aspecto relevante para aplicações é a presença para o acesso de todos os envolvidos de forma facilitada, alcançada por meio da implantação na Web. Dado o destaque da presença online, é fundamental que a interface seja responsiva, garantindo que a mesma seja acessível para uma variedade de telas diferentes, sem comprometer a experiência do usuário.

Desse modo, este trabalho tem o objetivo de descrever uma solução Web responsiva para a automatização das etapas de estágio a fim de tornar o fluxo de estágios flexível, centralizado e facilitado para todas as entidades envolvidas. A proposta para o fluxo de estágios a ser descrita neste texto utiliza máquinas de estados com transições baseadas nas entradas do usuário e também em eventos temporais para fornecer flexibilidade e manutenibilidade, a longo prazo, para futuras atualizações.

1.1 Objetivos

O foco principal deste projeto é apresentar uma solução Web responsiva, com destaque na sua arquitetura, nos componentes e fluxos, implementada seguindo práticas reconhecidas, com o intuito de automatizar procedimentos relacionados aos estágios da FACOM. Essa descrição utiliza abstrações, diagramas e conceitos definidos e discutidos na revisão bibliográfica.

1.1.1 Objetivos específicos

Para alcançar esse propósito, são definidos os seguintes objetivos específicos:

1. Detalhamento de requisitos, ferramentas e modelos para a contextualização do sistema alvo desta monografia.
2. Implementação do protótipo detalhado neste trabalho, fornecendo os requisitos apontados.
3. Descrição da arquitetura em diferentes camadas com ênfase em seus componentes.
4. Apresentação dos principais fluxos de execução.
5. Discussão por meio de análise crítica da arquitetura e fluxos tomando como base o detalhamento apresentado.

1.2 Justificativa

Um estudo sobre máquinas de estados, seguido por sua implementação, considerando um cenário prático real que envolve uma aplicação Web, é interessante para a comunidade de desenvolvedores em relação à perspectiva de estudo. Isso porque o uso de máquinas de estados fornece vários benefícios, incluindo: flexibilidade, manutenibilidade e simplificam a criação de fluxos complexos, além de permitir o seu uso em diversos cenários que envolvem diferentes tipos de soluções.

Considerando os benefícios da aplicação resultante, como apresentado por Mansour (2019), esses podem ir além do escopo de estágios e de seus processos. Ao fazer uso das conveniências oferecidas pela aplicação, o coordenador e outros participantes têm a oportunidade de direcionar as horas economizadas para realizar outras tarefas durante o expediente, como pesquisa, administração de aulas, entre outras. Isso não apenas melhora o processo de estágios, mas também contribui, de forma indireta, para o aprimoramento de diversas atividades acadêmicas.

Outro ponto essencial é que, ao levar em consideração a utilização do sistema, esta monografia atua como a documentação de referência do sistema, fornecendo abstrações divididas em camadas que podem ser estudadas por equipes distintas, cada uma desempenhando funções específicas.

1.3 Organização do trabalho

Este projeto está estruturado em 4 capítulos, cada um deles descreve os seguintes aspectos:

- No primeiro capítulo, é abordado o tema da monografia, iniciando uma discussão que engloba os objetivos, as justificativas e a estrutura da obra.
- No segundo, por sua vez, é realizado o levantamento de conceitos a partir de referências bibliográficas para auxiliar na descrição da aplicação realizada no desenvolvimento deste projeto.
- Adicionalmente, no terceiro, é apresentado o desenvolvimento, que descreve a aplicação resultante deste trabalho, separando-o em camadas e, para cada uma, é feita a apresentação de suas abstrações, diagramas, modelos de dados e demais componentes que atuam como referência para manutenção. Além de apresentar seus principais fluxos, disponibilização e discussão. A seguir são resumidas as seções do capítulo para um melhor direcionamento.
  - Na seção de visão geral do desenvolvimento, é contextualizada a solução alvo, apresentando-se suas tecnologias, motivação da divisão em camadas e os requisitos do sistema representados por diagramas de caso de uso, seguidos pelo respectivo detalhamento para dois contextos da aplicação: o uso por usuários convencionais e por administradores do sistema.
  - Já nas subseções de detalhamento das camadas da arquitetura, é realizada a divisão entre *Front-end*, *Back-end* e base de dados, seguido pelo aprofundamento nos detalhes sobre cada camada e seus componentes, incluindo linguagens, ferramentas, *frameworks* e modelos utilizados.
  - Além disso, os principais fluxos da aplicação são listados na seção de fluxos, na qual se incluem: cadastro, autenticação, solicitações gerais e máquina de estados.
  - Na última seção, é realizada a discussão sobre a aplicação por meio da análise crítica considerando características importantes de aplicações, além de serem ressaltados detalhes sobre cada uma das camadas.

- Por fim, é apresentada, no quarto capítulo, a conclusão deste projeto, referenciando trabalhos futuros.

2 Fundamentação teórica e revisão bibliográfica

Neste capítulo, há uma revisão da literatura, abordando os principais aspectos necessários para o detalhamento da arquitetura e dos fluxos da aplicação alvo deste projeto. Nas primeiras seções, são explorados conceitos relacionados a requisitos de software e à arquitetura e design de soluções Web, que incluem potenciais abstrações e diagramas, os quais são elementos fundamentais para a documentação efetiva da aplicação.

Posteriormente, em sua terceira e última seção, são explorados conceitos específicos utilizados no desenvolvimento, fundamentais para a compreensão deste trabalho. Isso inclui uma análise detalhada dos conceitos empregados no desenvolvimento, o que engloba modelos, ferramentas e tecnologias utilizadas.

2.1 Requisitos de software

Todo software possui o propósito de oferecer uma solução para um problema específico. Para compreender esta finalidade, a elaboração de requisitos é crucial, pois auxilia na compreensão da problemática em questão e até onde é proposta a solução. Essa fase é significativa e deve ser conduzida antes mesmo da definição dos aspectos da arquitetura, uma vez que os requisitos têm impacto em toda a aplicação, incluindo sua estrutura arquitetônica.

A definição do termo “requisitos de software” varia conforme a literatura. Considerando o escopo geral, é possível descrever que esses requisitos são uma especificação do que deve ser implementado, incluindo as descrições de funcionamento, de atributos ou até de restrições. Em todo caso, eles representam uma funcionalidade específica de um sistema alvo. Assim, percebe-se que o termo é bastante amplo, e é definido e caracterizado em tipos diferentes que podem ser relacionados entre si, como: especificação de requisitos de negócio, regras, restrições, interfaces, funcionalidades, requisitos funcionais e não funcionais, atributos de qualidade e requerimentos de sistema e de usuário (WIEGERS; BEATTY, 2013).

2.1.1 Requisitos e suas relações

Na Figura 1 são ilustradas as relações que consideram diferentes tipos de requisitos. Os retângulos indicam documentos resultantes da compilação dos diferentes tipos de requisitos, representados por figuras ovais. As setas contínuas indicam a presença dos

tipos de requisitos, enquanto as setas tracejadas representam o impacto dos requisitos. Outro aspecto demonstrado, é a divisão em camadas que indica os diferentes níveis de requisitos, envolvendo a produção de documentos.

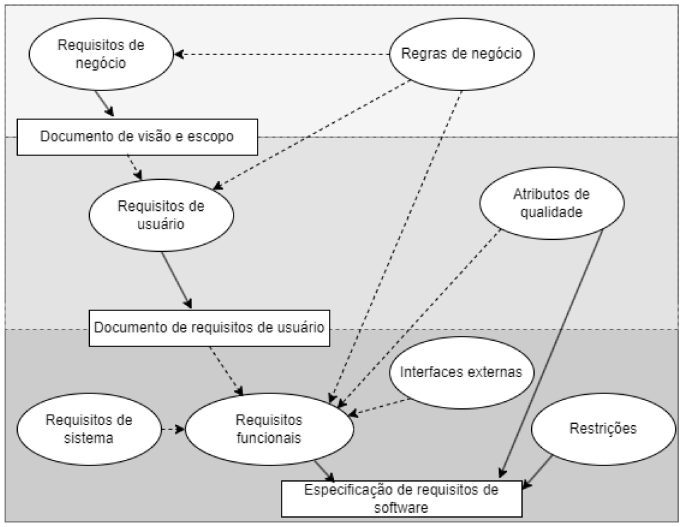


Figura 1 – Abstração dos diferentes tipos de requisitos de softwares e suas relações, adaptada de Wiegers e Beatty (2013).

Embora sejam definidos em níveis diferentes, os três documentos, apresentados na abstração, não precisam ser realizados separadamente ou em ordem. É necessário definir qual a melhor estratégia de levantamento de requisitos de software para a solução. Considerando aplicações de baixa escala, normalmente esses documentos são combinados, mas, para soluções de larga escala, é recomendada uma melhor profundidade nos documentos para o melhor entendimento de todo o seu escopo, visto que estas soluções envolvem muito mais funcionalidades do que as de baixa escala.

Além de apresentar seus tipos, Wiegers e Beatty (2013) também descrevem cada um dos tipos de requisitos, relacionando-os com as regras de negócio, restrições e interfaces. Bigelow (2020), por sua vez, apresenta outras variações de requisitos. Na Tabela 1, são apresentados os principais, considerando ambas as referências.

2.2 Arquitetura e design

Existe uma discussão sobre a diferença entre arquitetura e design, envolvendo software na comunidade. Embora estes conceitos possam ser relacionados, respectivamente, a modelagens de alto e de baixo nível sobre a abstração de um software, quando aplicadas na prática, possuem um mesmo objetivo: a descrição do formato do software e do seu comportamento, em relação aos seus diversos componentes para minimizar os recursos requeridos a fim de manter o sistema. Além disso, a arquitetura de um software consiste no formato definido para o sistema, que o organiza em componentes, posiciona-os e determina o método de comunicação entre eles (MARTIN, 2018).

Requisitos	Descrição	Dependência Direta
Requisitos de Negócio	Descrevem os porquês da implementação do sistema da organização. Possui foco principal nos objetivos da organização e entidades que solicitaram a solução	Regras de Negócio
Requisitos de Usuário	Listam os objetivos e tarefas que usuários poderão performar na solução incluindo características e atributos necessários.	Regras de Negócio, Documento de Visão e Escopo
Requisitos Funcionais	Especificam o comportamento que o produto irá exibir considerando condições específicas	Documento de Requisitos de Usuário, Interfaces Externas, Requisitos de Sistema
Requisitos Não Funcionais	Indicam detalhes a respeito da usabilidade da aplicação resultante considerando um escopo geral da solução	NA
Requisitos de Sistema	Abstraem os objetivos de sistema na qual o produto é definido considerando seus múltiplos componentes de subsistemas	NA

Tabela 1 – Requisitos, suas descrições e dependências.

Ford e Richards (2020) diferenciam a arquitetura e design ao considerar os aspectos do ciclo de vida da aplicação. Eles detalham que, enquanto a arquitetura é baseada nas etapas de modelagem, nos aspectos abstratos do software e em seus componentes, o design envolve as etapas de baixo nível que abordam as decisões de implementação, as regras de negócio e os protótipos de baixo nível. Eles citam, também, que, embora os conceitos sejam tratados como diferentes, eles devem ser trabalhados em conjunto para uma melhor qualidade de software resultante.

2.2.1 Características operacionais da arquitetura

De acordo com Ford e Richards (2020), as seguintes características principais fazem parte da finalidade da criação e da aplicação de padrões de arquiteturas, considerando a aplicação resultante:

- **Disponibilidade**: indica o quanto o sistema é disponível quanto a sua demanda e às eventuais atualizações. O desejado é uma aplicação que esteja ativa durante todo o tempo em que seus usuários desejarem utilizar.
- **Continuidade ou recuperação de desastres**: capacidade do sistema de se recuperar em caso de falhas.
- **Desempenho**: capacidade do sistema de desempenhar bem, conforme a sua demanda, incluindo as análises de testes, abordando seus picos de uso, testes de estresse, análise de requisições a APIs. A finalidade é a análise do desempenho do sistema para produção de métricas que podem ser utilizadas na tomada de decisões da aplicação.
- **Recuperabilidade**: inclui os requisitos de continuidade, isto é, detalhes e requisitos a respeito da recuperação de desastres da aplicação.

- **Confiabilidade ou segurança**: abrange os aspectos de segurança da aplicação e de dados dos seus usuários, considerando o nível de risco da aplicação.
- **Robustez**: capacidade do sistema de tolerância a falhas de aspectos externos, como falta de energia e falta de internet.
- **Escalabilidade**: capacidade do sistema de escalar seus recursos considerando o volume dos usuários e suas requisições, acrescendo ou decrescendo conforme a demanda.

Embora estas características sejam relevantes, existe uma compensação entre o uso de cada uma. Por exemplo, a definição de componentes, para melhorias de confiabilidade, pode prejudicar o desempenho do sistema por adicionar redundância, já a adição de componentes, para melhorar a escalabilidade, pode prejudicar a confiabilidade e desempenho por adicionar complexidade de orquestração dos sistemas em sua camada de comunicação entre componentes.

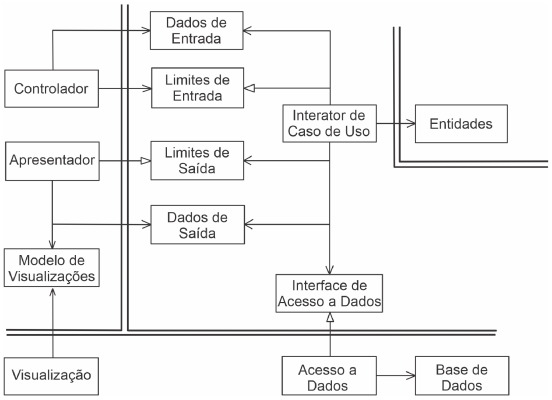
2.2.2 Arquiteturas Web

Apesar de terem sido discutidos aspectos teóricos da arquitetura em alto nível, esses conceitos também se aplicam às arquiteturas de aplicações Web. No estudo de Ford e Richards (2020), é apresentado um modelo genérico que divide a arquitetura em camadas, conforme é ilustrada na Figura 2. Esse modelo recomenda a comunicação entre camadas adjacentes, com a camada de apresentação sendo responsável pela interação com o usuário, a camada de negócios pelo controle das regras de negócio, a camada de persistência pela abstração dos modelos que gerenciam a conexão com o banco de dados e a própria base de dados.

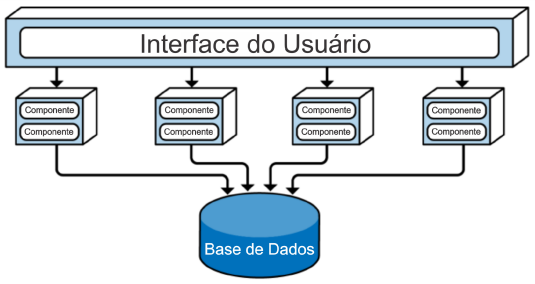
Figura 2 – Exemplo de separação de arquitetura em camadas, adaptada de Ford e Richards (2020).

A comunicação não adjacente de camadas oferece riscos às características da arquitetura, como a confiabilidade e desempenho, uma vez que se torna mais complexa a administração de segurança e testes de arquiteturas com enlaces de comunicação entre diversas camadas. Visto isso, é preciso definir desde o início como são posicionados os componentes e como funciona a comunicação entre esses elementos.

Martin (2018) e Ford e Richards (2020) apresentam arquiteturas Web, ilustradas na Figura 3, que, embora diferentes, quanto aos seus componentes, podem ser abstraídas, conforme o modelo em camadas anteriores.



(a) Arquitetura adaptada de Martin (2018).



(b) Arquitetura adaptada de Ford e Richards (2020).

Figura 3 – Exemplos de arquiteturas Web.

Enquanto a primeira apresenta uma solução Web em componentes, utilizando um *framework* de uma linguagem específica, a segunda apresenta um modelo genérico, envolvendo diversos componentes. É importante perceber que as duas arquiteturas são complementares, ao adaptar os componentes da segunda à primeira.

Em relação ao escopo mais específico de arquiteturas de aplicações Web, Luchaninov (2022) e IBM (2022) apresentam técnicas comuns utilizadas para arquitetar tais soluções. No geral, é considerada a arquitetura de três camadas, incluindo as camadas de apresentação, de aplicação e de dados, em que a camada de aplicação encapsula as camadas de negócio e persistência do modelo anterior.

O maior benefício de uso da arquitetura em três camadas é o desacoplamento das camadas, que podem ser implantadas utilizando infraestruturas independentes, permitindo o uso de tecnologias e times separados para a parte visual, de negócios e de banco de dados. Outros benefícios incluem a escalabilidade, a confiabilidade e a implantação eficiente.

Luchaninov (2022) apresenta alguns padrões de arquiteturas Web que podem ser implantadas seguindo o modelo de três camadas, detalhado anteriormente, utilizando componentes específicos para a sua configuração. Essas arquiteturas são bastante utilizadas no ambiente corporativo. Dentre elas se destaca a Aplicação de Página Única (SPA) que é importante para este trabalho.

Utilizada na camada de *Front-end* e adotada por grandes organizações, como Facebook e Google, o modelo de arquitetura SPA[1] oferece desempenho e flexibilidade, vi-

[1] Para mais detalhes sobre SPA, verifique: <https://en.wikipedia.org/wiki/Single-page_application>

abilizando o desenvolvimento de aplicações mais interativas em troca da introdução de complexidade e maior processamento. Essa abordagem possibilita a realização de renderizações dinâmicas de componentes em uma página, evitando a necessidade de recarregar a página por completo do lado da camada de apresentação, enquanto as demais camadas seguem a arquitetura convencional em três camadas.

Deacon (2009) descreve outra forma de arquitetar proposta pelo *framework* MVC[2] que é utilizada principalmente na camada do *Back-end* e abstrai a composição da arquitetura em 3 principais componentes: modelos, visões e controladores[3]. O autor ainda cita que o padrão foi proposto pelos criadores da linguagem SmallTalk, na década de 70, visando organizar a aplicação orientada a objetos com a finalidade principal de separar as camadas de apresentação das camadas de negócio e de persistência. A tripla do *framework* é detalhada a seguir:

- **Modelos**: é o conjunto de módulos que juntos modelam e representam um problema de uma camada inferior, como a camada de persistência. Nesse sentido, considerando o exemplo da base de dados, a camada pode atuar como uma abstração em níveis de objetos das tabelas da base de dados.
- **Visões**: representam a interface com o contexto externo da aplicação e podem ser desenvolvidas em telas visuais, interfaces de API e por comandos na interface da linha de comando.
- **Controladores**: se trata de um conjunto de objetos que são responsáveis por receber a entrada do usuário através de uma visão, realizar processamentos e controlar o fluxo utilizando os modelos para retornar uma resposta ao usuário.

ORM, que significa mapeador relacional de objetos[4], é uma técnica amplamente empregada na camada de modelos de software. Kriger (2023) detalha que seu propósito é mapear tabelas ou entidades do banco de dados em estruturas que podem ser utilizadas como objetos na aplicação. Essa técnica proporciona uma abordagem em que a manipulação do banco de dados e de seus componentes assemelha-se a interações em uma linguagem orientada a objetos, simplificando o desenvolvimento da aplicação.

Para a produção dos resultados baseada na entrada do usuário, existem alguns sentidos de comunicação mais utilizados, como os apresentados na Figura 4. No geral, o fluxo se inicia pela camada de interfaces, podendo ser definido pelo controlador ou não, e pode possuir os dados retornados pelo uso dos modelos diretamente pela interface ou manipulados pelos controladores.

[2] Para mais detalhes sobre MVC, verifique: <https://pt.wikipedia.org/wiki/MVC>
[3] Tradução livre da expressão em inglês Model-View-controller
[4] Tradução livre da expressão em inglês Object Relational Mapper

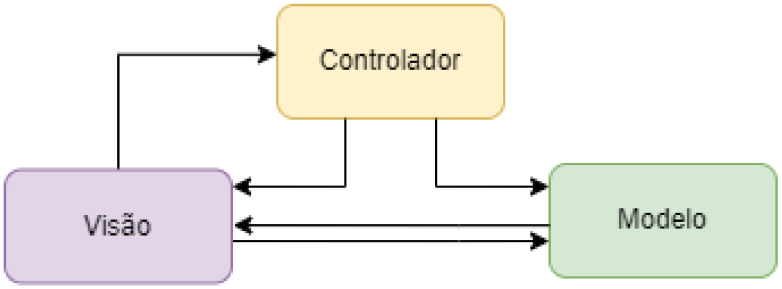


Figura 4 – Abstração das camadas do *framework* MVC e a comunicação entre seus componentes.

O uso do *framework* favorece o desacoplamento em funcionalidades da aplicação, permitindo o reuso eficiente dos modelos pelos controladores e dos controladores pelas visões, as quais podem utilizar parâmetros diferentes para personalizar os fluxos realizados por estes. Outro ponto positivo do uso do *framework* é a facilidade de criação de interfaces rapidamente com dados fictícios importantes para a documentação inicial do projeto devido à divisão dessas interfaces para os controladores e os modelos.

2.2.3 Abstrações da arquitetura

Uma das principais fases da construção do software é a definição da arquitetura e aprofundamento em detalhes do design. As etapas desempenham papéis fundamentais para a descrição dos componentes e todo o comportamento da solução da qual possui relevância, impactando nas tomadas de decisões do projeto em todo o seu ciclo de vida, incluindo seus resultados.

Normalmente, para aplicações de baixa escala, existe o procedimento informal de definição da arquitetura e design. Por outro lado, em cenários de aplicações de larga escala, envolvendo soluções críticas e complexas, as organizações costumam dedicar um maior esforço de recursos humanos e financeiros para a definição mais detalhada e padronizada sobre a arquitetura. Nestas, são consideradas diversas ferramentas de abstração de arquitetura e seus componentes, porém não existem ferramentas padrões, apenas as mais comuns (JONES, 2010).

A Linguagem de modelagem unificada ou UML é uma das ferramentas mais utilizadas para realizar a abstração de técnicas de modelagem de processos de negócios, incluindo o escopo de software. Sua principal finalidade é a definição de diagramas que abstraem pontos relevantes sobre a arquitetura, os quais são classificados entre duas categorias: diagramas de estrutura, que destacam detalhes a respeito do sistema, seus componentes e objetos, e diagramas comportamentais; que descrevem como o sistema deve se comportar em cenários específicos. Dentre os diagramas, os mais utilizados são os diagramas de casos de uso, diagramas de classes e diagramas de sequência (AMIT, 2018).

2.2.3.1 Diagrama de caso de uso

Fowler (2003) cita que os diagramas de caso de uso são frequentemente utilizados na etapa de descrição de requisitos funcionais em alto nível para descrever ações de usuários, considerando o funcionamento do sistema esperado, e eles fazem parte dos diagramas comportamentais.

Toda sua abstração utiliza três principais componentes: os atores, os casos de uso e as relações entre eles. Além disso, os diagramas de caso são posicionados e interligados para abstrair o comportamento dos requisitos com base nas interações de seus componentes. Outro elemento comumente utilizado são as tags de inclusão e extensão, as quais são utilizadas para, respectivamente, o reuso e a extensão de componentes. Na Figura 5 é apresentado um diagrama de caso de uso simplificado.

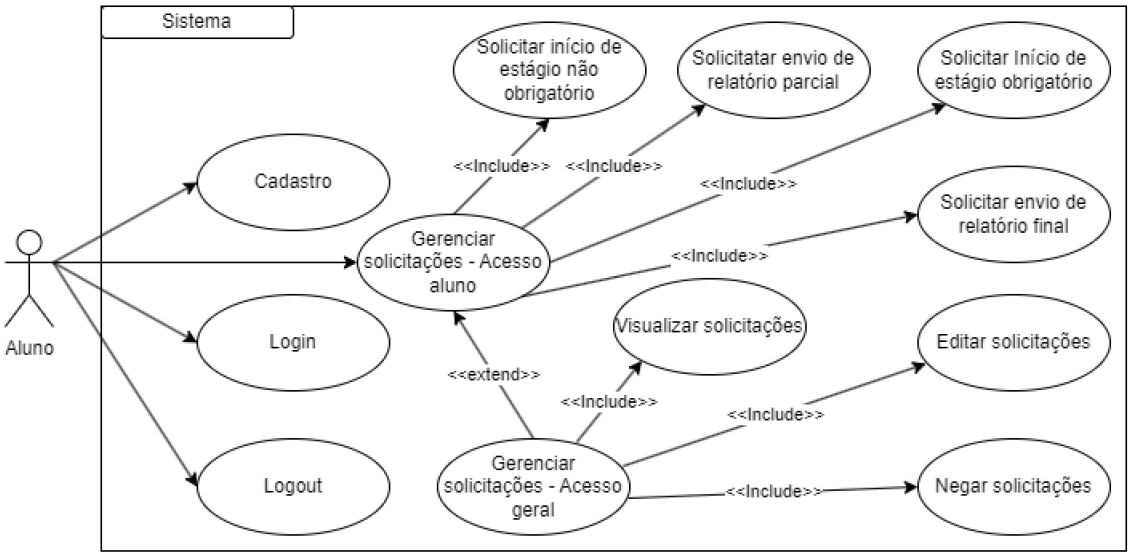


Figura 5 – Exemplo de um diagrama de caso de uso.

2.2.3.2 Diagrama de classes

Fowler (2003) descreve que o diagrama de classes é o mais utilizado, dentre os que envolvem UML, e possui a finalidade de descrever os tipos de objetos e as relações entre eles, apresentando suas características, com a inclusão de operações, restrições e como se comunicam.

As classes são representadas por caixas divididas em três seções, as quais são descritas de cima a baixo, como: o nome da classe, seus atributos e suas operações. Cada campo na classe é uma de suas propriedades, e os nomes são obrigatórios, indicando a classe respectiva na linguagem adotada e os atributos são estruturados da seguinte forma:

Visibilidade Nome: Tipo Multiplicidade = Default {String-de-Propriedade}

A visibilidade indica se o atributo é privado(-) ou público(+), o tipo de dados define o intervalo de valores possíveis ao atributo, a multiplicidade evidencia a quantidade de atributos que podem ou devem existir em paralelo aos objetos da classe e o valor pode ser assumido como não instanciado ou com valor atribuído utilizando, respectivamente, padrão ou o texto de propriedade.

As operações correspondem aos métodos que pertencem à classe e determinam as ações que a classe pode tomar. É importante ressaltar que alguns dos campos destacados anteriormente estão presentes na definição de operações, incluindo: visibilidade, nome, multiplicidade e textos de propriedade com a adição do tipo de retorno que o método retorna e a lista de parâmetros do método que possui nome seguido pelo tipo e valor. A estrutura das operações é destacada a seguir.

Visibilidade Nome (Lista-de-Parâmetros): Tipo-Retorno Multiplicidade
{String-de-Propriedade}

As associações são as setas que interligam classes indicando que as propriedades de uma são estendidas pela classe alvo da seta. Podem existir associações bidirecionais, que indicam que ambas as propriedades estão interligadas. Já as generalizações são utilizadas para a extensão de atributos da superclasse para as demais classes que compartilham de seus atributos. A superclasse é o alvo da seta completa.

Além dos componentes anteriores, existe a dependência indicada por setas incompletas tracejadas, que indicam dependências entre classes em que o alvo da seta aponta a classe que é a dependida, e também existem os comentários indicados por retângulos ligados com retas tracejadas a classes. Na Figura 6 é apresentado um diagrama de classes.

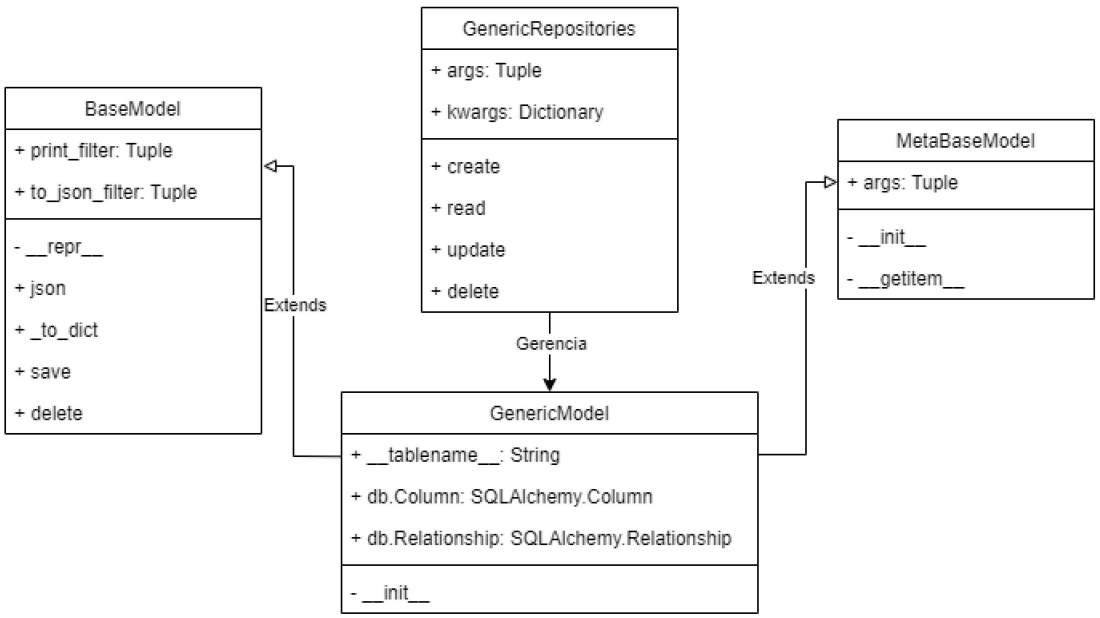


Figura 6 – Exemplo de um diagrama de classes.

2.2.3.3 Diagrama de sequência

O diagrama de sequência descreve, em um fluxo ordenado de operações, um cenário de interações envolvendo objetos de classes e mensagens trocadas. Isso inclui todos os objetos presentes no cenário e possui a ordem verticalmente decrescente, útil para descrever operações complexas (FOWLER, 2003).

O fluxo inicia por uma operação que não inclui um participante inicial, que são denominadas mensagens encontradas[5]. Os objetos do diagrama são nomeados participantes, enquanto as linhas verticais tracejadas indicam a ausência de fluxo, os retângulos verticais posicionados sob estas mostram processamento, esses possuem setas constantes ou tracejadas indicando, respectivamente, o envio de mensagens e retorno entre os participantes.

Outro aspecto relevante é a criação e a remoção de participantes de forma dinâmica no fluxo, onde são utilizadas as operações *new* e *close*, demarcadas por setas indicadoras de criação e marcadores *X*, indicando exclusão. Além disso, laços e condições podem ser aplicados ao utilizar quadros iterativos que encapsulam o fluxo de repetição ou de condição. É importante definir a expressão de condição de execução utilizada por estes controles de fluxo. Na Figura 7 é apresentado um diagrama de sequência.

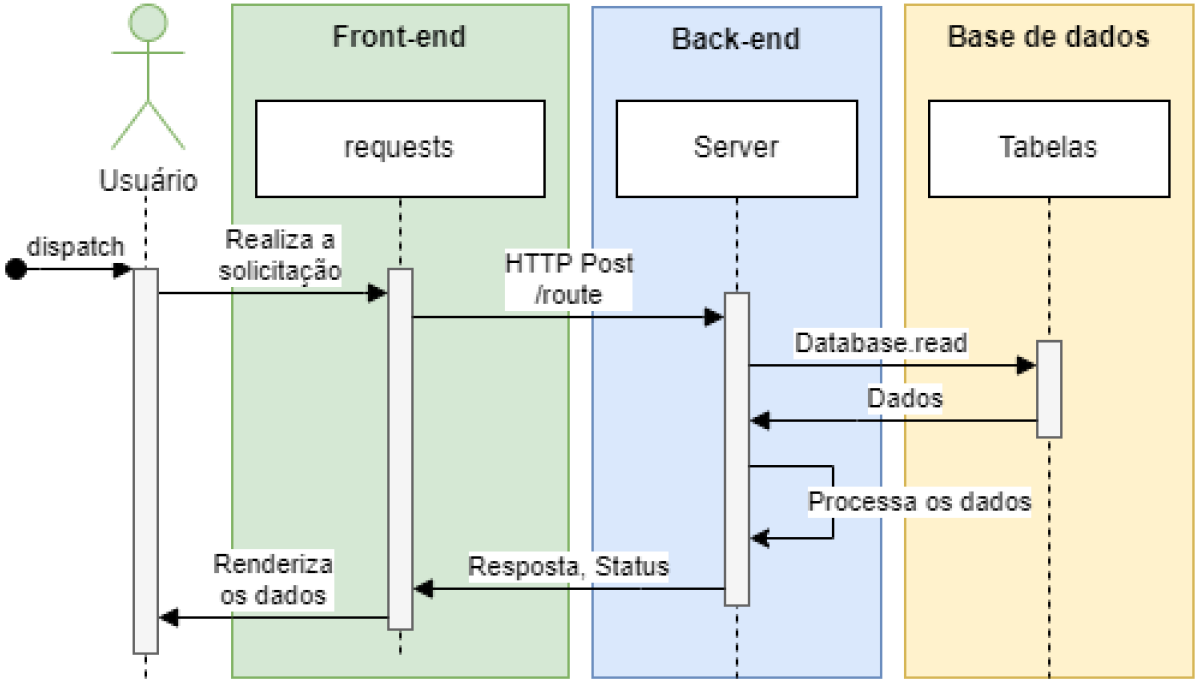


Figura 7 – Exemplo de um diagrama de sequência.

[5] Tradução livre da expressão em inglês found messages

2.3 Conceituação

Nesta seção, são apresentados conceitos, ferramentas, técnicas e modelos relevantes para este trabalho e utilizados no capítulo de desenvolvimento. As subseções são organizadas em ordem lexicográfica na sequência: API REST, JSON e JWT, Protocolos envolvendo HTTP e SMTP e Máquina de estados.

2.3.1 API REST

Massé (2012) descreve que, enquanto uma API é uma interface de comunicação que clientes utilizam para se comunicar com um servidor específico, REST é o estilo arquitetural de design moderno que pode ser adotado ao criar a API. Tornar um servidor que fornece uma API *Restful* é o mesmo que seguir o padrão REST e envolve adotar um conjunto de regras de design.

Presente em modelos de comunicação *cliente-servidor*, a API utiliza diversos componentes para o estabelecimento da comunicação, incluindo a URI do servidor, verbos, cabeçalhos, corpo e versão da requisição (HENDERSON, 2023). Enquanto a URI permite o mapeamento de um nome identificador de um recurso para o seu endereço físico, incluindo em seu escopo, as URLs convencionais da Web e também as URNs que servem de nomenclatura para recursos[6], os demais campos são específicos do protocolo HTTP.

Portanto, juntamente ao estilo arquitetural, é utilizado o protocolo HTTP, descrito futuramente nesta revisão, que são componentes da API resultante. O conjunto de regras de design é extenso e envolve a padronização de nomenclaturas para URI dos recursos do software que implementa a API, a semântica correta ao utilizar os verbos HTTP e os seus resultados, entre outros. A seguir, são listadas as normas relativas ao uso correto dos quatro principais verbos HTTP.

- ***GET***: deve retornar representações atualizadas dos objetos do servidor, a requisição pode conter cabeçalho e argumentos na URL, mas não pode conter corpo. Além disso, é recomendável que o estado do objeto do servidor se mantenha nestas requisições.
- ***PUT***: é utilizado para inserir e atualizar dados persistidos pelo uso de dados em cabeçalho e no corpo da URL, retornando um objeto como resposta e deve incluir as modificações ou uma mensagem que descreve o seu sucesso.
- ***POST***: permite a criação de recursos em uma coleção e a chamada de recursos que servem como controladores da API que tomam diversas ações. É necessária atenção

[6] Para mais informações sobre URI, URL e URN visite <https://auth0.com/blog/url-uri-urn-differences/>

para evitar o uso de outros escopos das requisições devido à abrangência que o POST possui em termos de uso.

- ***DELETE***: deve ser utilizada para remover recursos do servidor, envolvendo coleções e dados persistidos, e deve ser priorizado o envio de parâmetros de consulta para identificar o objeto em exclusão na URI da requisição.

Outro aspecto relevante são os códigos de retorno que a API REST deve fornecer. Estes são responsáveis por indicar ao cliente o estado final de uma solicitação realizada para que se possa tomar uma decisão baseada em seu retorno. A seguir, são detalhadas as categorias da API REST, separadas em intervalos:

- **1xx - Informativo**: permite a comunicação no nível da camada do protocolo de transferência.
- **2xx - Sucesso**: indica o sucesso de requisições do cliente. Nesta categoria, destacam-se os códigos 200 e 201, que são utilizados para indicar sucesso e sucesso ao criar um item respectivamente.
- **3xx - Redirecionamento**: descreve quais passos adicionais são necessários para completar a requisição.
- **4xx - Erro do cliente**: detalha erros de origem do cliente. Dentre os códigos desta categoria, destacam-se os 401, 403 e 404, os quais descrevem, respectivamente: solicitação não autorizada devido a credenciais inválidas, acesso não autorizado a um cliente autenticado e recursos não encontrados.
- **5xx - Erro do servidor**: detalha erros de origem do servidor. O código mais utilizado é o 500, que indica erro interno de servidor.

Embora toda a descrição seja indicada e detalhada por diversas referências, é de responsabilidade dos desenvolvedores o design correto da API. As diretrizes, para manter a arquitetura conforme as normas REST, devem ser seguidas ao utilizar os métodos e códigos de retorno corretos, considerando o seu escopo em termos de funcionalidade.

2.3.2 JSON e JWT

BasuMallick (2022) define que JSON, notação de objetos JavaScript[7], é um formato de arquivo usado em POO que utiliza uma linguagem simples, independente e inteligível para armazenar objetos que podem ser transmitidos entre aplicações, que foi desenvolvido

[7] Tradução livre da nomenclatura JavaScript Object Notation

pelos criadores da linguagem JavaScript. O formato é definido por um conjunto de representações em chave-valor, separadas por vírgula, em que cada representação possui um tipo entre 7 possíveis. Devido a sua independência, o formato é muito utilizado fora do contexto da linguagem JavaScript, e é utilizado por vários outros como o próprio JWT.

Lima (2021) descreve que o JWT, também conhecido como JSON Web Token, é um token padronizado pela RFC 7519[8], utilizado para a autenticidade e envio de informações em APIs. A autenticidade é garantida ao utilizar técnicas envolvendo assinatura digital com o algoritmo HMAC[9] ou pelo uso de pares de chaves públicas e privadas RSA[10] ou ECDSA[11]. O token é uma sequência de 3 campos JSON: cabeçalho, carga útil e assinatura[12], que são convertidos em caracteres em base 64[13], separadas em 3 partes por ponto. Esses campos são ilustrados na Figura 8 e detalhados em seguida:

Campos	Token
Cabeçalho	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
Carga útil	eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
Assinatura	SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Figura 8 – Exemplo de um token JWT, adaptada de Lima (2021).

- Cabeçalho: inclui o tipo do token, no caso JWT, e o algoritmo criptográfico utilizado.
- Carga útil: é o corpo do token onde é posicionado os dados específicos da API, aplicação ou do usuário.
- Assinatura: inclui a assinatura para a checagem de autenticidade do token. São utilizados o cabeçalho, a carga útil e a chave privada para a realização da assinatura que pode, posteriormente, ser verificada pela chave pública.

Uma aplicação amplamente utilizada, considerando o JWT, é a autenticação de usuários em um sistema. No processo inicial, os usuários realizam a autenticação enviando suas credenciais ao servidor, que, com base nessas informações, gera um JWT assinado com uma chave privada e o envia de volta ao usuário. Após receber o token, o usuário utiliza-o para enviar solicitações ao servidor. Este, por sua vez, verifica a validade do token, decidindo se permite ou restringe o acesso com base nessas informações (PEYROTT, 2018).

É importante ressaltar que a representação do JSON em base 64 não é segura em relação ao acesso por terceiros, visto que entidades que possuam o token resultante podem

[8] Verifique a RFC 7519 em<https://datatracker.ietf.org/doc/html/rfc7519>
[9] Para mais informações sobre HMAC acesse <https://en.wikipedia.org/wiki/HMAC>
[10] Verifique mais detalhes sobre RSA em <https://en.wikipedia.org/wiki/RSA_(cryptosystem)>
[11] Consulte informações sobre ECDSA em <https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm>
[12] Tradução livre das palavras em inglês Header, Payload e Signature
[13] Verifique detalhes formato base 64 no site <https://pt.wikipedia.org/wiki/Base64>

decodificar a representação e verificar os dados presentes. Entretanto, a autenticidade definida pela cifragem, utilizando a chave privada, garante que os dados não sejam alterados e recebidos pelo servidor ao fazer a checagem da assinatura com base na chave pública, sendo, portanto, uma ferramenta eficiente para estabelecer autenticação em aplicações.

2.3.3 Protocolos

Para o estudo de protocolos, é fundamental a discussão sobre os modelos em camadas de intercomunicação na rede. Tanenbaum (2011), além de descrever o modelo OSI de 7 camadas, também apresenta o modelo TCP/IP com alguns de seus principais protocolos detalhados na Figura 9. Os modelos constituem uma representação abrangente de todas as fases envolvidas no fluxo de comunicação pela internet.

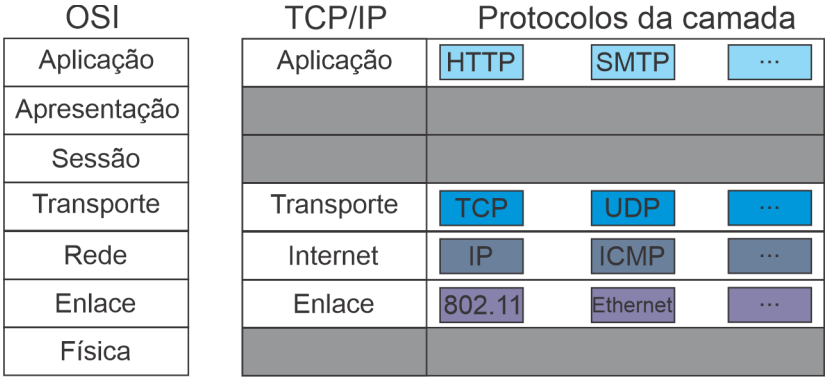


Figura 9 – Camadas do modelo OSI, TCP/IP e protocolos, adaptada de Tanenbaum (2011).

Na abstração, cada camada, desde o nível físico até o de aplicação, possui uma função específica na transmissão de dados, estabelecendo uma relação modular, hierárquica e transparente. Seguindo a ordem do modelo OSI, as camadas são organizadas da mais baixa para a mais alta, incluindo física, enlace de dados, rede, transporte, sessão, apresentação e aplicação. Já no TCP/IP, embora semelhante, existe uma sobreposição de funcionalidades de camadas adjacentes para simplificar o modelo em 4 camadas: enlace, internet, transporte e aplicação.

Protocolos, por sua vez, são um conjunto de regras e convenções que permitem a comunicação padronizada e eficiente entre dispositivos e são situados em uma das camadas específicas dos modelos apresentados anteriormente. Os protocolos da camada de aplicação, por exemplo, são responsáveis por abstrair detalhes e funcionalidades para aplicações finais e usuários, os quais são inclusos nos protocolos HTTP e SMTP.

2.3.3.1 HTTP

Gourley e Tott (2002) descrevem que o protocolo HTTP, que significa Protocolo de transferência de hipertexto, é atualmente um dos protocolos da camada de aplicação mais

utilizados para transferência de dados na internet, permitindo a transferência confiável de dados por utilizar TCP. TCP ou protocolo de controle de transmissão é um dos principais protocolos da camada de transporte, e permite o envio confiável de dados ao estabelecer uma conexão lógica entre pares de dispositivos na internet.

O funcionamento do protocolo é baseado no fluxo de requisição e resposta de recursos da Web gerenciado por servidores. Inicialmente, o recurso cliente realiza uma requisição a um recurso servidor, este retorna uma resposta ou não à solicitação. O protocolo é responsável pelo controle das regras da comunicação em alto nível e é disposto em diversos cabeçalhos, métodos que definem os tipos de requisições *Get*, *Put*, *Delete*, *Post* e *Head*, além dos códigos de resposta, conforme são exemplificados na Subseção 2.3.1.

Um dos principais cabeçalhos do protocolo é o de compartilhamento de recursos entre origens ou CORS. Este, ao ser configurado, permite que os navegadores verifiquem quais recursos de origens externas a outro recurso podem realizar o carregamento de dados da aplicação servidora. O dispositivo é importante porque permite definir restrições para dispositivos não autorizados de acesso malicioso.

2.3.3.2 SMTP

RedinSkala (2013) apresenta detalhes sobre o protocolo SMTP, que significa protocolo de envio de e-mail simples, e descreve que o protocolo da camada de aplicação, padronizado pela RFC 821[14], surgiu para estabelecer uma comunicação padronizada de mensagens pela internet. Os principais componentes, relacionados ao envio de suas mensagens, envolvem o endereço do emissor, endereço de destino, dados do cabeçalho, dados do corpo, anexos, entre outros.

O funcionamento do protocolo é baseado em servidores de envio e de recebimento de e-mails funcionando acima do protocolo TCP. Esses servidores possuem todas as diretrizes do protocolo implementadas, incluindo: filas de e-mail, sincronização, controle de erros, persistência das mensagens, entre outras. Estas funcionalidades permitem o envio e recebimento das mensagens ao serem enviadas ao endereço de e-mail, que são traduzidas para o servidor de recebimento alvo.

Por ser inicialmente padronizado o uso dos caracteres ASCII[15] para o envio de mensagens, foi definida a extensão MIME, que significa extensão multifuncional de correio da Internet. Ela passou a permitir o uso de outras representações de formatos para o envio de e-mails, como imagens, vídeos, entre outras, o que evidencia uma mudança bastante significativa.

[14] Acesse as informações da RFC 821 em <https://datatracker.ietf.org/doc/html/rfc821>
[15] Para mais informações sobre ASCII verifique <https://en.wikipedia.org/wiki/ASCII>

2.3.4 Máquina de estados

Wagner (2006) apresenta a máquina de estados finita como sendo um modelo abstrato que possui um número finito de estados, os quais são utilizados para representar informações e se alteram de acordo com entradas seguindo regras definidas nomeadas de transições. O autor descreve que o funcionamento é sequencial e ocorre com iterações conforme as entradas de usuário, as quais são checadas por uma unidade de processamento que verifica o estado atual e as regras das transições para decidir qual o próximo estado a ser escolhido como saída da operação. Além disso, mesmo que não exista uma constante definida, é recomendado o uso consciente de um limite de estados para que a máquina resultante não seja complexa dependendo do contexto da aplicação. Na Figura 10 é apresentado a abstração descrita.

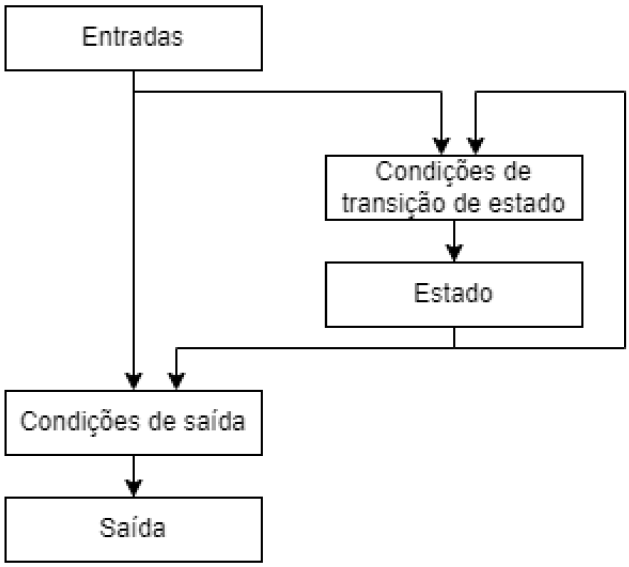


Figura 10 – Abstração de uma máquina de estados finita, adaptada de Wagner (2006).

Moore e Gupta (2023), além de descrever a máquina em alto nível, também apresenta o modelo formal matemático da máquina, denominado como um "Autômato Finito Determinístico", útil para realização de análises e descrição de diagramas resultantes da aplicação da técnica.

Um autômato finito determinístico é descrito por uma tupla de cinco elementos: $(Q, \Sigma, \delta, q_0, F)$, onde:

Q é o conjunto finito de estados

Σ é o alfabeto finito não vazio

δ é uma série de transições

q_0 é o estado inicial

F é o conjunto de estados de aceitação

Uma máquina capaz de executar o modelo considerando a sua quíntupla para uma entrada específica é denominada reconhecedora, e, a partir de uma entrada específica, pode decidir se a entrada é aceita ou é rejeitada após a sequência de operações em que o ato de aceitar ocorre caso a máquina pare em um estado de aceitação. Menezes (2009) utiliza a notação de conjuntos para a definição de alfabetos que podem ser reconhecidos pelos autômatos, como o exemplo a seguir extraído da referência.

$$\text{L} = \{\ \text{w} \mid \text{w possui número par de letras a e b}\}$$

Embora o modelo apresentado seja simples, inúmeras soluções de software utilizam das abstrações, em sua maioria com adições informais, para desenvolver processos complexos que podem ser modelados em estados com estruturas de dados e fluxos adicionais. O uso do processo, além de facilitar a modelagem da tarefa, torna a aplicação flexível devido à natureza do modelo de utilização de estados e transições, os quais permitem a adição de novos estados com esforço mínimo de configuração (EMERSON, 2022).

2.3.4.1 Abstrações

Wagner (2006) apresenta modelos e representações para as máquinas úteis para a análise de seus estados, transições e para documentação de resultados, pelas quais são descritas a Matriz de transição e o Diagrama de transição de estados. Nesta seção, essas abstrações são representadas por uma máquina de estados que possui um autômato que reconhece a linguagem L citada anteriormente.

2.3.4.1.1 Matriz de Transição

A matriz de transição no contexto de máquinas de estados é uma matriz onde são posicionados membros do alfabeto em suas células que permitem a relação de transição do estado indicado na linha para o estado indicado na coluna, utilizando o símbolo do alfabeto. Caso não existam transições para a célula, um marcador de nulo pode ser utilizado. Na tabela 2, é representada uma matriz de transição.

De \Para	Q_{0F}	Q_1	Q_2	Q_3
Q_{0F}	-	b	a	-
Q_1	b	-	-	a
Q_2	a	-	-	b
Q_3	-	a	b	-

Tabela 2 – Exemplo de uma matriz de transição.

Outra maneira de apresentar a tabela é a representação de estados listados pelas linhas e símbolos do alfabeto nas colunas, nas quais as células são preenchidas com os

estados resultantes. Esta alternativa é mais prática e deve ser utilizada em cenários onde existe alto volume de estados e/ou símbolos em seu modelo. Na Tabela 3, é representada a matriz de transição alternativa.

De \Entrada	a	b
Q_{0F}	Q_2	Q_1
Q_1	Q_3	Q_{0F}
Q_2	Q_{0F}	Q_3
Q_3	Q_1	Q_2

Tabela 3 – Exemplo de uma matriz de transição alternativa.

2.3.4.1.2 Diagrama de transição de estados

O Diagrama de transição de estados permite uma visualização da abstração da máquina em grafos. Os círculos representam os estados, as setas indicam as transições e apontam no estado de saída, além de possuir descrito em seu corpo o conjunto de símbolos de entrada que fazem parte da transição. Menezes (2009) utiliza, em seus diagramas, uma seta sem estado de partida para indicar o estado inicial e um círculo com duas circunferências para indicar estados de aceitação. Na Figura 11 é apresentado um exemplo de diagrama de transição de estados.

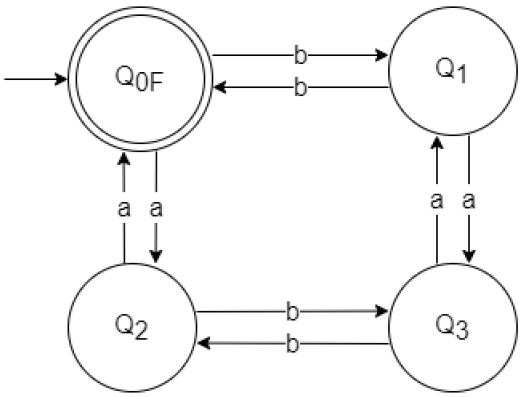


Figura 11 – Exemplo de um diagrama de transição de estados, adaptada de Menezes (2009).

2.3.4.2 Implementações

Carmely (2009) apresenta uma implementação simples para cenários de soluções que dependem de uma modelagem única e específica descrita no Algoritmo 1, considerando a solução para a linguagem L. Apesar de sua facilidade, a implementação não é flexível a mudanças, sendo necessária a alteração manual do código caso a máquina de estados sofra alterações além do crescimento linear da quantidade de linhas de código com relação à quantidade de estados da máquina.

Algoritmo 1: Pseudocódigo do algoritmo de máquinas de estados apresentado por Carmely.

```
1: estadoatual = Q_0F
2: faça
3:    simboloatual = Leia(entrada)
4:    se (estadoatual = Q_0F)
5:       se (simboloatual = a)
6:          estadoatual = Q_2
7:       senão se (simboloatual = b)
8:          estadoatual = Q_1
9:       fim se
10:   senão se (estadoatual = Q_1)
11:      se (simboloatual = a)
12:         estadoatual = Q_3
13:      senão se (simboloatual = b)
14:         estadoatual = Q_0F
15:      fim se
16:   senão se (estadoatual = Q_2)
17:      se (simboloatual = a)
18:         estadoatual = Q_0F
19:      senão se (simboloatual = b)
20:         estadoatual = Q_3
21:      fim se
22:   senão se (estadoatual = Q_3)
23:      se (simboloatual = a)
24:         estadoatual = Q_2
25:      senão se (simboloatual = b)
26:         estadoatual = Q_1
27:      fim se
28:   fim-se
29: enquanto (simboloatual ≠ ∅)
30: retorna estadoatual = Q_0F
```

Uma alternativa simples para os problemas apresentados no algoritmo anterior é o uso de uma linguagem pré-definida para descrever os componentes da máquina de estados, seguida pela adoção de estruturas de dados e controles de fluxo que definem a lógica das transições realizadas dada uma entrada. AWS (2023) implementa, no seu serviço Step Functions, uma máquina de estados na nuvem que descreve os seus componentes, utilizando JSON, que é posteriormente interpretado para realizar transições baseadas nas entradas de usuário e eventos.

Contudo, dada uma descrição e utilizando uma estrutura de dados que suporte a abstração, é possível realizar uma implementação independente de diferentes descrições de máquina de estados, permitindo que mudanças de funcionamento do fluxo sejam realizadas com alterações apenas do lado da descrição, possibilitando alta flexibilidade e

simplicidade. No Algorítmo 2, é apresentado o algorítmo baseado em descrições de máquinas de estados.

Algoritmo 2: Pseudocódigo do algoritmo de máquinas de estados abstraídas em uma estrutura de dados.

```
1: estadoatual = Q_0F
2: conjuntodeestadosfinais = LeiaDescrição(estadosfinais)
3: conjuntodetransições = LeiaDescrição(transições)
4: faça
5:    simboloatual = Leia(entrada)
6:    para cada transição ∈ conjuntodetransições faça
7:       se PossuiEstadoSimbolo(transição, estadoatual, simboloatual)
8:          estadoatual = EstadoSaida(transição)
9:       senão se UltimaTransicao(conjuntodetransições, transição)
10:         retorna Falso
11:      fim se
12:   fim-para
13: enquanto (simboloatual ≠ ∅)
14: retorna estadoatual ∈ conjuntodeestadosfinais
```

2.3.5 Relação da fundamentação e revisão com o trabalho

As Seções 2.1 e 2.2 são fundamentais para a documentação e implementação do protótipo alvo deste trabalho. Enquanto a primeira, apresenta os tipos de requisitos que são abstraídos e implementados no sistema, a segunda estuda a arquitetura da solução e seus principais padrões, envolvendo abstrações, utilizadas para o detalhamento da aplicação.

Os conceitos apresentados na seção de conceituação, são utilizados na implementação do protótipo e também no detalhamento da aplicação feita neste trabalho. A seção serve de referência para a identificação de técnicas específicas, utilizadas na implementação.

Por fim, o estudo da última seção, apresenta o algoritmo base utilizado para a implementação da máquina de estados no sistema, além de fornecer os diagramas que são fundamentais para o detalhamento realizado sobre as máquinas neste trabalho.

3 Desenvolvimento

Este capítulo descreve o sistema alvo ao considerar suas principais tecnologias e componentes, utilizando os conceitos abordados na seção de revisão bibliográfica, especialmente as abstrações utilizadas para representar o sistema. Na primeira seção, é apresentada uma visão geral da solução resultante. Na segunda, as subseções detalham, com maior profundidade, o sistema, dividindo-o em *Front-end*, *Back-end* e Base de dados. A terceira seção aborda os fluxos mais cruciais da aplicação, abrangendo todas as camadas. A quarta, por sua vez, descreve como a aplicação é disponibilizada e, por fim, a última apresenta uma discussão considerando aspectos negativos e positivos da aplicação.

3.1 Visão geral

A aplicação resultante é uma aplicação arquitetada em três camadas desacopladas *Front-end*, *Back-end* e base de dados. Ela segue as abstrações feitas por Ford e Richards (2020) e Martin (2018) que servem como protótipo a uma solução de longo prazo e uma solução disponível a curto prazo que possui um conjunto de funcionalidades já implementado a ela. Esse conjunto permite o gerenciamento flexível, personalizável e evolutivo do fluxo básico de estágios da FACOM considerando usuários da coordenação de estágios, orientadores e alunos.

A arquitetura da solução e as definições de módulos de cada uma das suas camadas são realizadas utilizando como base modelos e estruturas de projetos modernos que permitem o reuso de componentes e simplificação da manutenção e evolução da aplicação a longo prazo, além de conter pontos mais importantes de segurança implementados na comunicação entre as camadas. As principais ferramentas, a divisão entre camadas e divisão entre módulos da aplicação são exibidas na Figura 12.

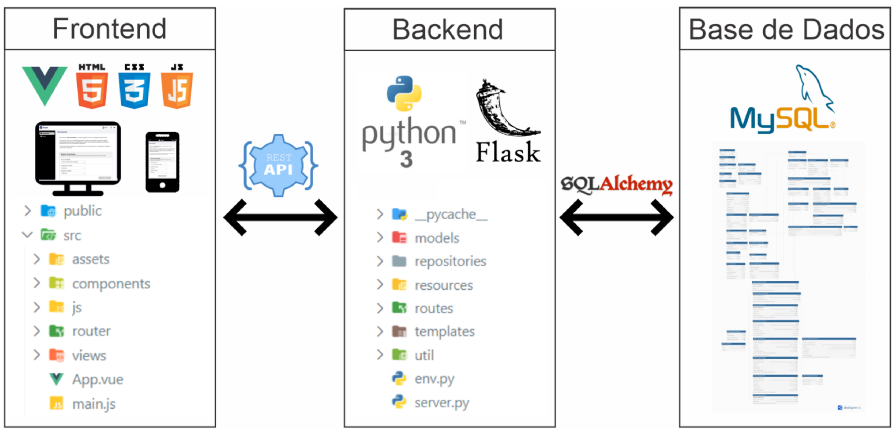


Figura 12 – Camadas da arquitetura base, tecnologias e divisão da aplicação.

Cada uma das camadas apresentadas, suas tecnologias, módulos e motivações são discutidas nos capítulos seguintes com maior profundidade e, no geral, o sistema possui as telas implementadas na camada de *Front-end*, que utiliza uma API REST para a comunicação com o *Back-end*. Este implementa as rotas da API, as regras de negócio e modelos ORM para a comunicação com o Banco de Dados relacional que, por sua vez, possui as tabelas que armazenam informações de configurações, usuários e de funcionalidades as quais o sistema utiliza.

O sistema é projetado para fornecer flexibilidade na configuração do fluxo de estágios e também seus principais componentes. Há alterações simplificadas pelos seus administradores por meio de ajustes em sua base de dados, como a alteração de fluxos existentes, criação de novos fluxos de solicitações. Esses fluxos são implementados utilizando uma abstração de máquina de estados, personalização e criação de e-mails e até definição de páginas dinâmicas simples nos fluxos que podem ser definidos na base de dados.

Além disso, a aplicação é mantida em dois repositórios diferentes[1,2], os quais separam as partes visuais das regras de negócio e banco de dados. O motivo para isso é o desacoplamento dos diferentes grupos de tecnologias, considerando seu escopo de desenvolvimento em *Front-end* e *Back-end*. Isso permite que, em eventuais atualizações, exista a possibilidade de times trabalharem em paralelo, além de manter uma maior consistência com base nas ferramentas de disponibilização de aplicações, que podem utilizar apenas recursos necessários para implantar a parte visual e a parte de negócios do sistema.

Nas subseções a seguir, são apresentados a contextualização do sistema considerando dois tipos diferentes de uso do sistema, o uso por usuários do fluxo e o uso por administradores que desejam realizar manutenção e evolução envolvendo o banco de dados para melhor clareza do escopo da solução.

3.1.1 Diagrama de caso de uso e contextualização para usuários do sistema

O estudo do contexto, baseando-se no fluxo de estágios e suas normas, é de suma importância para o entendimento do funcionamento da aplicação e suas funcionalidades. Esse funcionamento e aplicações podem ser estudados ao consultar a documentação oficial da FACOM[3,4], que é constantemente alterada e este é o principal motivo para a definição de funcionalidades flexíveis no sistema.

Existem três principais usuários, considerando o fluxo, são eles: i) os alunos, que

[1] *Front-end* disponível em <https://github.com/VCalixtoR/sisflow-frontend>
[2] *Back-end* e código da base de dados disponível em <https://github.com/VCalixtoR/sisflow-backend>
[3] Documentação oficial dos estágios do BCC: <https://facom.ufu.br/graduacao/bcc/estagio-supervisionado>
[4] Documentação oficial dos estágios do BSI: <https://facom.ufu.br/graduacao/bsi-santamonica/estagio-supervisionado>

realizam solicitações a respeito do estágio, ii) os orientadores, que orientam os discentes e iii) a coordenação, que realiza os procedimentos formais a respeito do estágio. Devido à existência desses usuários, o sistema os abstrai e eles definem mudanças impactantes nas funcionalidades da aplicação, considerando acessos a componentes de interface, o uso e gerenciamento de diferentes funcionalidades. Na Figura 13, é apresentado o diagrama de caso de uso do sistema que auxilia no entendimento desses perfis e seus impactos no sistema.

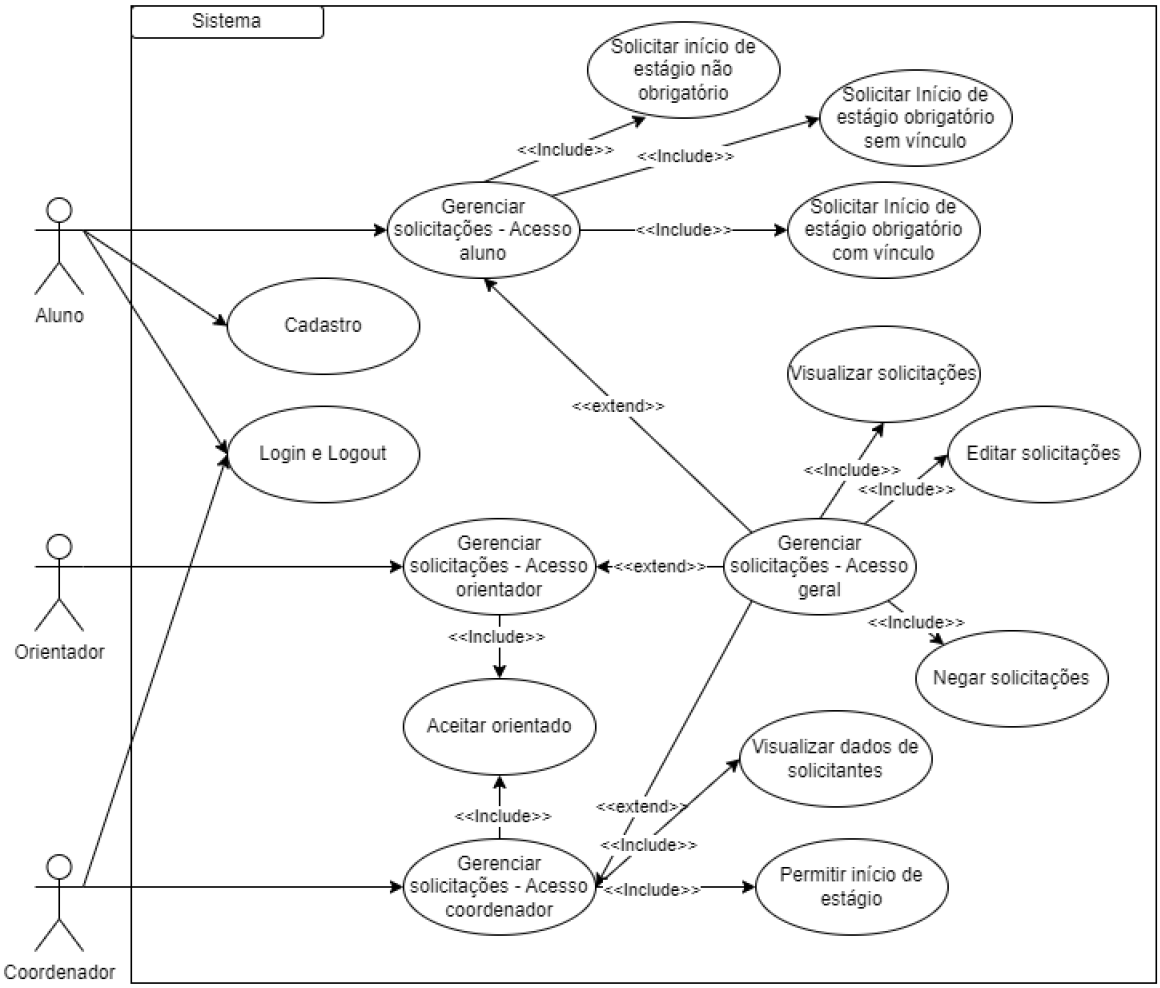


Figura 13 – Diagrama de caso de uso do sistema.

Além de apresentar o impacto dos perfis, ao observar o diagrama presente na Figura 13, é possível visualizar as principais funcionalidades implementadas no sistema. Em primeiro caso, o aluno possui, além da possibilidade de cadastro e login, três diferentes tipos de solicitações que podem ser realizadas envolvendo o início de estágio. Essas solicitações utilizam diferentes máquinas de estados implementadas no sistema, sendo que, uma solicitação realizada pelo discente resulta em uma instância de uma máquina de estados. Essa máquina inicia um fluxo de solicitações envolvendo demais perfis de usuários que possuem estados alterados baseado em ações de usuário e por eventos baseados no tempo.

Todos os perfis podem responder às solicitações iniciadas pelo aluno, considerando o estado atual de seu fluxo. Ele é atribuído a um ou mais perfis e possui as funcionalidades básicas de gerência de solicitações, que incluem visualizar, editar e negar solicitações, desde que o perfil seja autorizado a responder àquela etapa do fluxo da solicitação. Este ponto é relevante para a garantia de consistência do andamento da solicitação com relação ao estágio.

Já os perfis do corpo docente possuem mais funcionalidades, sendo que o perfil de orientadores pode aceitar a orientação de um aluno. O perfil do coordenador, por sua vez, além de ser possível aceitar a orientação, por ser elegível, também pode aceitar o início de estágio e visualizar mais dados a respeito dos envolvidos devido à necessidade de verificar se as informações dos envolvidos estão de acordo com as normas estabelecidas pelo setor jurídico.

3.1.2 Diagrama de casos de uso e contextualização para administradores

Como apresentado na subseção anterior, as normas e o fluxo envolvendo o processo de estágios são dinâmicas. Devido a isso, é importante a definição de funcionalidades que permitam a alteração pelo perfil administrador que adaptem o sistema às mudanças, de forma facilitada.

Para atingir essa necessidade, as principais funcionalidades que envolvem a máquina de estados, definição de páginas, perfis e agendador de eventos, são populadas e configuradas, considerando abstrações preenchidas no banco de dados. Isso permite que alterações na base de dados afetem o funcionamento do fluxo, permitindo a adaptação de mudanças às normas exigidas ao adicionar uma camada maior de complexidade na aplicação. O diagrama de caso de uso, apresentado na Figura 14, ilustra as possíveis modificações que um perfil administrador pode fazer ao realizar alterações na base de dados.

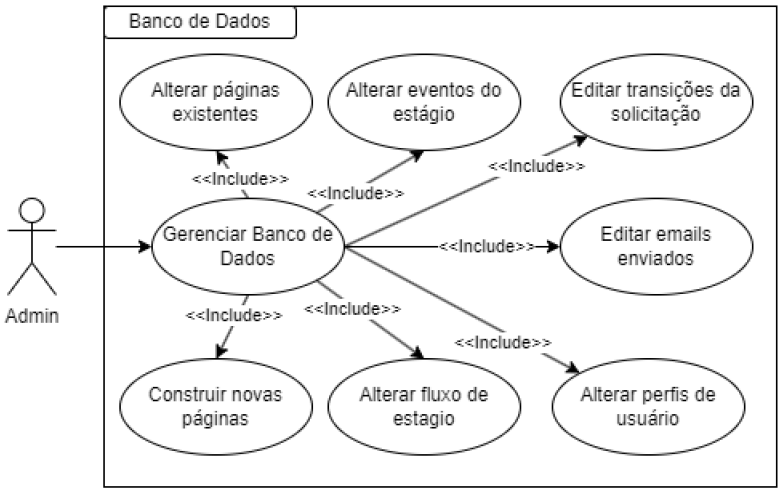


Figura 14 – Diagrama de caso de uso para administradores.

São diversos os requisitos que representam as alterações que podem ser realizadas. A seguir, é fornecido um descritivo de cada um deles:

- **Alterar eventos do estágio**: possibilita a alteração de eventos de transição de estados do fluxo para um estado específico, considerando um intervalo de tempo.
- **Alterar fluxo de estágio**: permite alterar a abstração da máquina de estados que modifica como é o fluxo realizado no sistema para solicitações de alunos.
- **Alterar páginas dinâmicas existentes**: proporciona a alteração de páginas dinâmicas criadas.
- **Alterar perfis de usuário**: viabiliza a criação de novos perfis de usuário com base em dados personalizados e armazenados em um JSON na base de dados.
- **Construir novas páginas dinâmicas**: permite a criação de páginas dinâmicas que podem ser utilizadas no fluxo.
- **Editar e-mails enviados**: possibilita a edição das mensagens e dos e-mails que são enviados ao ocorrer uma transição.
- **Editar transições de solicitação**: concede a edição de transições que ocorrem na mudança dos estados do fluxo, possibilitando a edição e seleção dos vários tipos de transições.

3.2 Camadas da arquitetura

Nesta seção, são descritos os aspectos relevantes de cada uma das camadas da aplicação, conforme apresentadas na Figura 12. Neles, são considerados o funcionamento geral, as tecnologias empregadas, as ferramentas utilizadas, a divisão em categorias ou módulos e as motivações que fundamentaram a tomada de decisões. Para isso, serão utilizados os diagramas estudados anteriormente como apoio, proporcionando uma melhor análise das camadas da aplicação.

3.2.1 Front-end

Na camada visual ou o *Front-end* da aplicação, estão implementadas as interfaces que os usuários interagem com o sistema. Essa camada é desenvolvida utilizando o *framework* Vue.js[5], pois ele permite usufruir dos benefícios de desenvolvimento e reuso que uma SPA oferece, conforme mencionado por Luchaninov (2022). O *framework* utiliza,

[5] O *framework* Vue.js na versão 3.2 pode ser verificado em <https://blog.vuejs.org/>

em seu conjunto de ferramentas, as tecnologias mais convencionais de soluções *Front-end*, como HTML, CSS e JavaScript, organizadas pelo gerenciador de pacotes npm[6].

A estrutura de módulos e componentes utilizada é aquela disponibilizada oficialmente pelo *framework*, que possui a maioria das dependências e módulos necessários para o desenvolvimento e facilita a adição de novos módulos. Contudo, a aplicação é construída a partir da junção de componentes, localizados no módulo respectivo, em um conjunto denominado "visão"[7]. As visões, por sua vez, são renderizadas pelo navegador e selecionadas com base na rota atual do site que o usuário acessa, conforme definido nos conceitos de SPA. É apresentado na Figura 15 uma abstração geral da camada e um detalhamento, de cada um dos componentes apresentados, é realizado no Apêndice A.

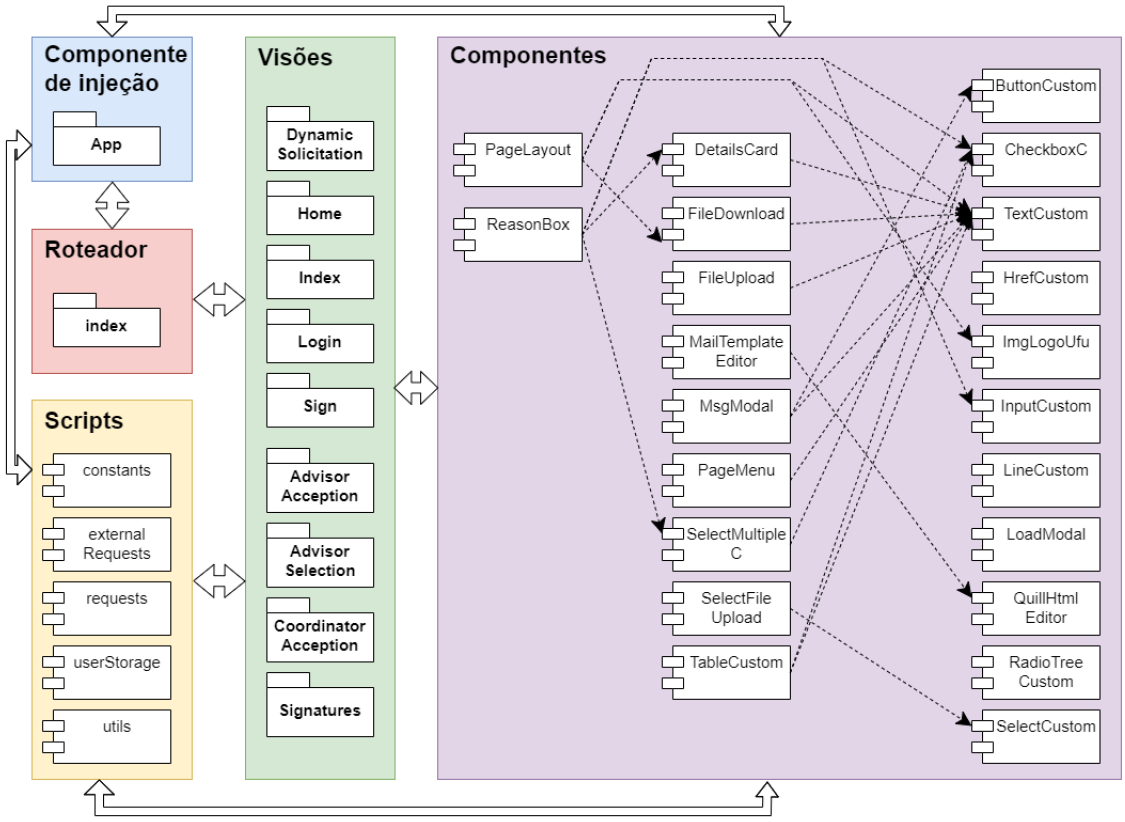


Figura 15 – Abstração do *Front-end*, seus componentes e comunicação.

A construção da aplicação inicia a partir de um modelo simples HTML que chama o componente de injeção *App*, responsável por aplicar todo o gerenciamento base do SPA. Nesse processo, inclui-se o uso do *index* de roteamento para a montagem de visões que possuem seus componentes inseridos. Portanto, esse é o ponto focal do *framework* e da camada que abstrai, dos arquitetos de software juntamente ao roteador e o montador de pacotes npm, toda a complexidade das configurações para permitir os benefícios do SPA.

Após a montagem dos recursos necessários para o uso do SPA, o roteador pode decidir, com base na lógica implementada, qual das visões será renderizada na página

[6] O gerenciador de pacotes npm na versão 8.19 pode ser consultado em <https://www.npmjs.com/>
[7] Tradução da palavra em inglês view

única com base nas interações com os usuários. Por exemplo, após o usuário clicar em um componente da página, é possível solicitar ao roteador que renderize uma nova visão que, então, é posicionada substituindo a antiga na página.

Do lado dos usuários, quando a visão é alterada, existe a percepção de que foi renderizada uma nova página, porém a mudança é uma simulação feita pelo Vue.js. Ele renderiza novos componentes, reposicionando os antigos da mesma página pelo uso de scripts da biblioteca para abstrair e simular a mudança de visões sem a necessidade do recarregamento de toda a página. Isso evita requisições novas ao servidor de *Back-end*, o que melhora o desempenho e facilita o reuso de componentes.

Além da construção apresentada anteriormente, em todos os componentes são incluídas seções de HTML, CSS e JavaScript centralizadas. Elas fazem parte do escopo do objeto ou das visões para a definição do seu funcionamento, incluindo a descrição de sua interface e a implementação de sua lógica. Esse é um padrão do *framework*, e é utilizado na solução por facilitar a construção de novos componentes e por favorecer o desacoplamento de funcionalidades de uso específico. Por outro lado, funcionalidades gerais são posicionadas nos scripts para permitir um melhor reuso como é o caso do script *requests* que possui implementadas as rotas de comunicação com o *Back-end*.

3.2.1.1 Visões

Para obter um melhor detalhamento do *Front-end*, é crucial descrever a função de cada visão levando em consideração o fluxo de estágios. É importante destacar que existem dois tipos de visões: as dinâmicas, geradas com base nos dados do banco de dados, e as estáticas, que são complexas demais para serem criadas dinamicamente. A seguir, é fornecida uma descrição destas.

- ***DynamicSolicitation***: possui a estrutura para a renderização das páginas dinâmicas com base em dados do Banco de Dados fornecidos pelo *Back-end*. É realizada a requisição ao *Back-end* sobre uma solicitação específica, se o tipo de página da solicitação for dinâmica, é retornado um objeto que descreve os componentes da visão. Esta possui um loop que itera sobre a estrutura fornecida, renderizando cada componente na ordem que foi enviado formando a página dinâmica.
- ***Home***: implementa a página principal do usuário com as solicitações e seus estados. É apresentada uma tabela contendo as etapas de cada solicitação e as ações que podem ser tomadas, sendo possível visualizar ou editar as etapas da solicitação dependendo de seu estado atual. Para o perfil de alunos, são apresentados componentes que permitem a realização de solicitações, iniciando seus respectivos fluxos.
- ***Index, Login e Sign***: possui a página inicial do sistema, login e cadastro. Essas páginas desempenham o papel de disponibilizar componentes que permitem aos

usuários fornecer suas credenciais para se cadastrar ou efetuar login. Isso, por sua vez, cria uma sessão autenticada que possibilita o acesso às rotas da API que exigem um token de autenticação para serem acessadas.

- ***AdvisorAcception***: é uma visão estática que pertence ao fluxo de início de estágio. Possui os componentes para a aceitação do discente pelo orientador em um processo de estágio que o discente tenha solicitado a orientação ao orientador.
- ***AdvisorSelection***: assim como a anterior, é uma visão estática do fluxo de início de estágios que implementa a página que o aluno solicita a orientação de estágio ao orientador. Este, posteriormente, pode aceitar na página *AdvisorAcception*.
- ***CoordinatorAcception***: também é como as duas anteriores, é uma visão que a coordenação de estágios utiliza para decidir se o aluno é apto a iniciar o processo de início de estágio. Possui componentes de informações do aluno e um modelo de e-mails que pode ser alterado para o envio deste personalizado no caso do indeferimento.
- ***Signature***: é uma visão estática do fluxo de início de estágios que permite assinaturas e envio de documentos pelos três perfis do sistema para a finalização de processos que precisem de assinaturas. Utilizada para finalização dos fluxos iniciais.

3.2.1.2 Scripts

Com base na descrição anterior sobre os scripts, é perceptível que eles são fundamentais para a aplicação de boas práticas na arquitetura envolvendo reuso e desacoplamento. No geral, eles são utilizados pelas visões e componentes, incluindo o componente de injeção para realizar diversas funcionalidades como o envio de requisições ao servidor de *Back-end* e para reutilização de variáveis de configurações padrões que define cores e tamanhos de fontes para a camada. A seguir, são apresentados os scripts e seus usos:

- ***constants***: são presentes em seu escopo constantes que são injetadas como variáveis CSS e são utilizadas por diversos componentes. As constantes definem cores, fontes e tamanho de quebra de página para responsividade. Elas são padrões à aplicação e, ao serem alteradas, impactam em todo o sistema. Este script é crítico e permite o reuso eficiente das configurações.
- ***externalRequests***: utilizado para o posicionamento de requisições a aplicações terceiras à solução que não utilizam da mesma URL de endereço do *Back-end*.
- ***requests***: implementa a estrutura de comunicação HTTP utilizando a interface Fetch API[8] para que o *Front-end* faça requisições ao *Back-end*, além de possuir

[8] Para mais detalhes sobre Fetch API verifique: <https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API>

todas as suas rotas de requisições. A estrutura definida possui tratamento de status HTTP convencionais, bem como problemas de rede e é utilizada pela maioria das visões da aplicação.

- ***userStorage***: dispõe de funções para manipulação de variáveis de sessão do usuário para armazenamento de tokens para autenticação. Todas as variáveis são salvas em sessão, as quais são eliminadas pelo navegador com o seu término por segurança.
- ***utils***: possui utilitários de uso geral como, por exemplo, funções que formatam datas. O script permite o reuso de funções que podem ser utilizadas em mais de um componente.

3.2.1.3 Exemplo de funcionamento da visão Home

Para melhor detalhamento dos pontos apresentados anteriormente nesta subseção, é descrito o funcionamento da página Home, cuja seleção acontece por ela ser considerada a página mais importante para o início dos fluxos de solicitação do aluno. Ela é apresentada na Figura 16, considerando o posicionamento de componentes implementados na camada, os padrões da linguagem HTML não são destacados.

Como já mencionado, o componente de injeção *App* desempenha diversas funções, que incluem a definição de componentes utilizados antes da renderização das visualizações e da gestão das rotas. No contexto da página inicial, o componente *App* define as cores e tamanhos de fonte padrão, utilizando o script constants. Em seguida, ele renderiza o componente *PageLayout*, no qual os componentes da *Home* são exibidos quando a rota correspondente é acessada. Este acesso é permitido pelo script de roteamento, que especifica que a visão utilizada é a *Home* através do mapeamento entre rotas e caminho dos módulos das visões.

A razão pela qual o componente *App* renderiza o *PageLayout* antes da renderização interna da visão *Home* é permitir a reutilização deste layout para outras visões. Diversas visões aproveitam os componentes *PageMenu* e a barra superior, que inclui o nome do sistema e o logotipo da UFU. Essa abordagem possibilita a adição de novas visões ao fluxo sem a necessidade de modificar o layout da página. Ao invés disso, basta definir os componentes específicos da nova visão, mantendo a consistência e a eficiência na estrutura da página.

Prosseguindo com o fluxo, uma vez que o modelo base da página, incluindo o menu e a barra superior, é renderizado por meio do *PageLayout*, a parte restante da página é gerada. Para esse propósito, na seção de scripts da visão *Home*, é definida a lógica em JavaScript responsável por fazer solicitações ao *Back-end* utilizando o script *requests* para obter os dados específicos de solicitações do usuário. Outros dados do usuário, como o

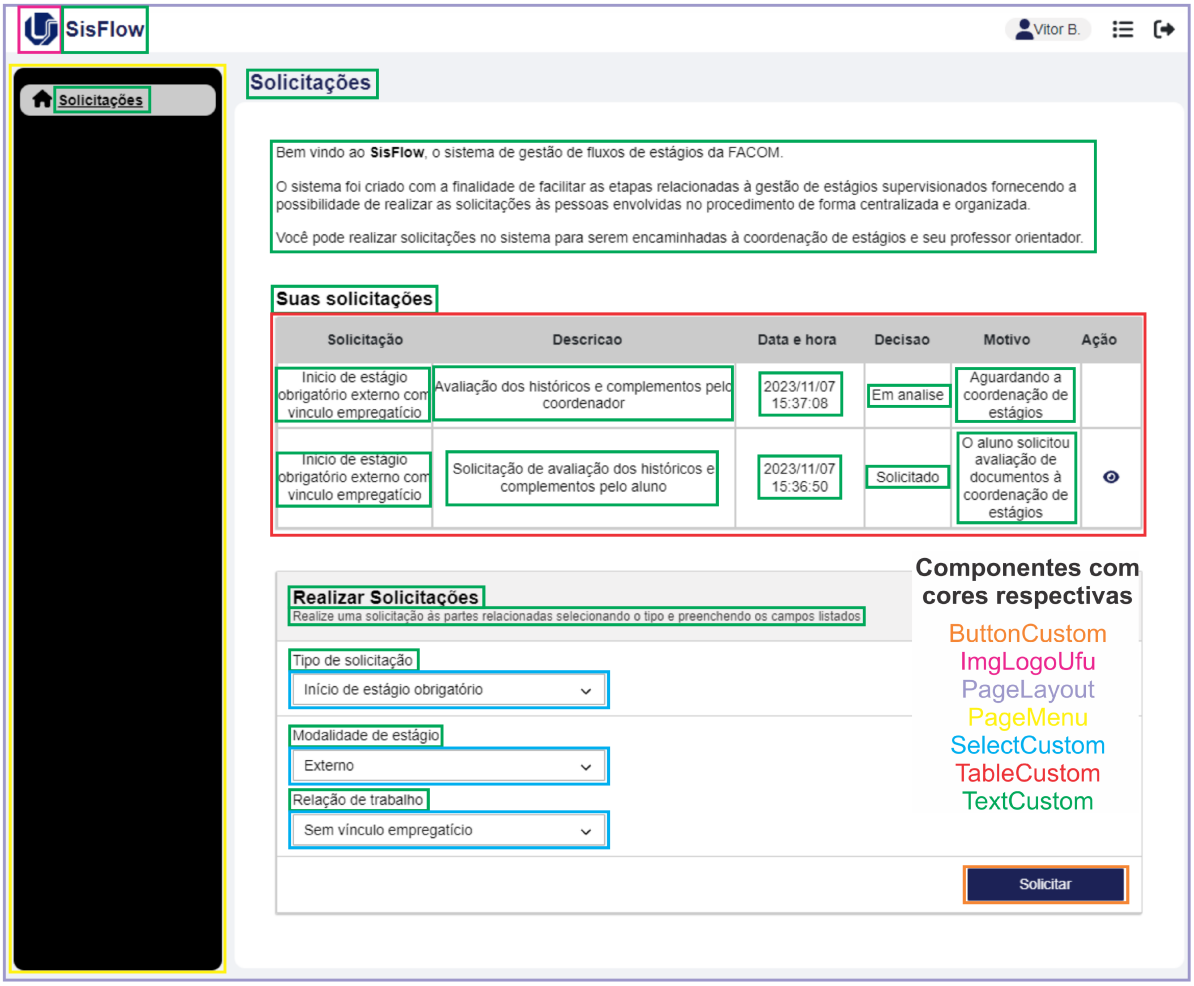


Figura 16 – Página *Home* com componentes destacados.

nome e o perfil, são obtidos a partir do token de sessão armazenado quando o usuário faz o login utilizando o script *userStorage*.

Uma vez que a visão tenha sido completamente renderizada, ela é exibida ao usuário, permitindo que ele visualize e responda às suas solicitações. No caso de um aluno, a opção de realizar novas solicitações ao sistema é disponibilizada no campo *Realizar Solicitações*. É importante ressaltar que a visão Home possui uma lógica que ajusta seus componentes com base no perfil do usuário, personalizando o conteúdo apresentado conforme as necessidades específicas de cada perfil.

3.2.2 Back-end

A camada *Back-end* da aplicação desempenha um papel fundamental no sistema, encarregando-se da gestão dos dados da aplicação. Ela considera todo o fluxo de solicitações e dados de usuário, da aplicação de regras de segurança, da configuração da API REST e da modelagem de objetos ORM para facilitar a interação com o banco de dados, bem como a comunicação tanto com o banco quanto com o *Front-end*.

Foi selecionada a linguagem Python[9] em conjunto com o *framework* Flask[10] para a criação do servidor que implementa a API REST, que possibilita a comunicação entre o *Back-end* e o *Front-end*. Além disso, foi adotado, para estabelecer a conexão entre o *Back-end* e a base de dados, o conjunto de ferramentas SqlAlchemy[11] que é uma biblioteca que permite o mapeamento entre objetos da linguagem e instâncias das tabelas do banco de dados utilizando ORM.

Já para a definição da estrutura de módulos, foi utilizado um modelo bastante reconhecido pela comunidade[12]. Além de ser uma estrutura de uso organizacional, o modelo de módulos promove o desacoplamento de rotas ao utilizar pequenos trechos de código nomeados de *Blueprints* que podem ser ou não anexados a API em um script. Isso facilita a manutenção e evolução da aplicação que permite até a adição dinâmica de rotas. Outro ponto relevante sobre seu uso é a divisão entre rotas, recursos, repositórios e modelos ORM que facilita o reuso devido à definição de pedaços de códigos desacoplados e de funcionalidade única. A estrutura de módulos resultantes pode ser visualizada na Figura 12.

3.2.2.1 Classes

A camada é caracterizada por várias funcionalidades, que englobam abstrações de máquinas de estados, segurança da API baseada na autenticação por pares de chaves públicas e privadas, bem como *threads* para servidores de e-mail e agendadores de eventos. Essas funcionalidades são encapsuladas em classes dentro da aplicação e são representadas no diagrama de classes, conforme ilustrado na Figura 17.

É importante ressaltar que as abstrações de classes GenericResources, GenericRepositories e GenericModels representam um conjunto de classes que possuem atributos, métodos e funcionalidades similares, e são agrupadas para simplificação devido a elas, em conjunto, envolverem 76 classes utilizadas de forma semelhante com pequenas diferenças considerando seus atributos e métodos. Por exemplo, as classes de modelo ORM(agrupadas em *GenericModel*), as quais herdam as classes *BaseModel* e *MetaBaseModel*, diferenciam-se apenas pelas colunas da base de dados e suas relações que representam, mas mantêm a mesma funcionalidade de abstrair uma tabela da base de dados em um objeto.

A principal classe apresentada no diagrama é a do servidor Flask implementada pela biblioteca e serve de base para a montagem do diagrama considerando as demais

[9] Para mais detalhes sobre a linguagem Python na versão 3.10 verifique o site <https://www.python.org/>

[10] Verifique mais detalhes sobre o *framework* Flask 2.2.2 em <https://flask.palletsprojects.com/>

[11] Verifique mais detalhes sobre o conjunto de ferramentas SQL SqlAlchemy 2.0 em <https://www.sqlalchemy.org/>

[12] Consulte o boilerplate de projetos python em seu repositório <https://github.com/antkahn/flask-api-starter-kit>

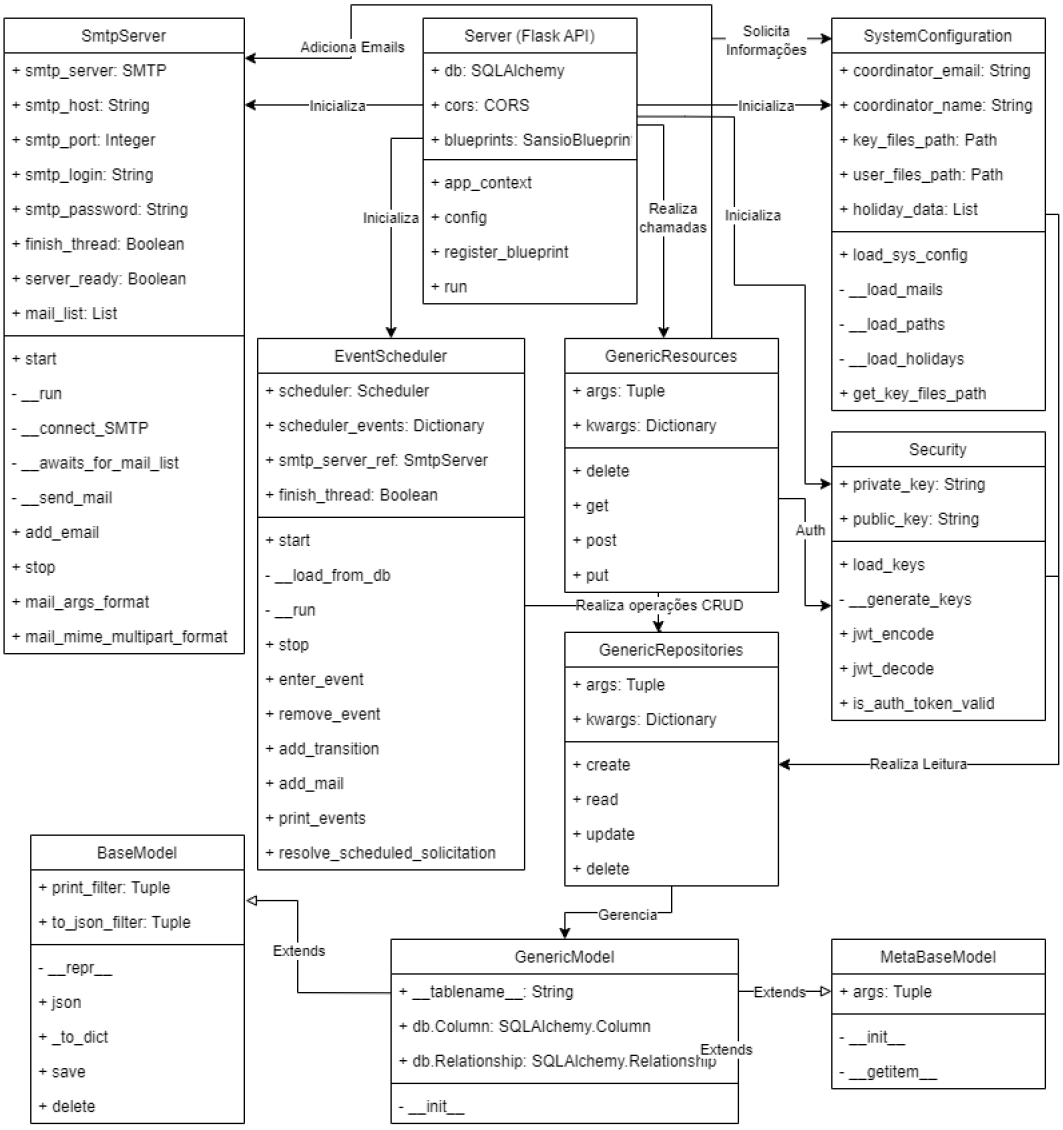


Figura 17 – Diagrama de classes do *Back-end*.

classes. Esta classe é a primeira importante a ser instanciada no fluxo e é responsável pela inicialização de outras. Essas outras, por sua vez, são responsáveis pelas funcionalidades de envio de e-mails, agendamento de eventos, configurações de sistema, segurança entre outras. Cada uma das classes é detalhada a seguir:

- ***Server(Flask API)***: classe responsável por abstrair detalhes gerais do servidor para o desenvolvedor, incluindo a API REST, gerenciando todo o fluxo de *threads* de conexão com clientes, uso do protocolo HTTP, seus verbos e configuração de CORS, além de ser responsável por iniciar a conexão com a base de dados utilizando o SQLAlchemy. A classe utiliza modelos chamados de *Blueprints* dos quais representam rotas desacopladas que são adicionadas à classe para que esta disponibilize a rota da API para aplicações externas como o *Front-end*.

- ***SystemConfiguration***: é a primeira classe inicializada pelo servidor e possui atri-

butos e métodos de uso geral utilizados pelas classes principais apontadas no diagrama e classes de recursos. A principal finalidade da criação da classe é inicializar a sua instância que coleta dados do banco de dados para favorecer o reuso e evitar consultas repetidas à base de dados. Por exemplo, o nome do coordenador é bastante solicitado pelas classes de recursos, e com a presença do atributo na instância, é possível evitar consultas desnecessárias na base de dados.

- ***SmtpServer***: é uma classe que implementa um segundo fluxo ao servidor que representa um servidor SMTP para gerência e envio de e-mails. Pode receber solicitações de envio de e-mails por outros módulos ao adicionar os dados para o envio em sua lista de e-mail. A classe formata os dados e envia o e-mail de forma assíncrona para não impedir o fluxo da API.

- ***EventScheduler***: semelhante ao item anterior, a classe implementa uma *thread* responsável pelo fluxo de eventos da máquina de estados. Ao ser iniciada pelo servidor, são carregados, da base de dados, os eventos pendentes se existirem à classe, e eles são anexados em sua fila de eventos. Após isso, é possível adicionar, em tempo real, eventos com base em solicitações realizadas pelas classes de recursos. Todos os eventos são salvos no banco de dados com um estado que indica se é pendente, realizado, cancelado, ou etc. Isso é importante, pois, se o sistema falhar, é possível recuperar os eventos em andamento.

- ***GenericResources***: são classes que implementam a lógica das requisições recebidas pelo servidor, aplicando mudanças nos dados e realizando operações na base de dados por meio das classes de repositórios. Elas são definidas conforme os verbos HTTP, e, por terem a natureza de tratamento da lógica do fluxo, possuem a máquina de estados implementada em parte de suas operações.

- ***GenericRepositories***: é um conjunto de classes destinadas a realizar operações CRUD da base de dados. Por serem as únicas interfaces para a realização de operações com a base de dados, elas são utilizadas pela maioria dos outros módulos e possuem operações desacopladas em métodos estáticos, favorecendo o reuso.

- ***GenericModel***: são classes que definem modelos de representação das entidades da base de dados, em outras palavras, são as especificações resultantes da aplicação de ORM.

- ***BaseModel* e *MetaBaseModel***: são utilizadas pelas classes de modelos a partir de herança para que os modelos herdem métodos utilitários como o método de representação para impressão no console e conversão dos objetos em JSON. São classes com implementação reutilizada do modelo de módulos base.

3.2.2.2 Rotas da API

Um ponto importante a respeito da camada são as rotas da API REST fornecidas, que cada uma processa e/ou retorna dados com uma finalidade específica. A finalidade principal delas pode ser resumida em realizar funcionalidades para os usuários, atuando no processamento e persistência de dados, considerando operações de escrita, leitura, atualização e remoção.

Nesta subseção, elas são apresentadas a seguir, na Tabela 4, incluindo a categoria, operação HTTP, rota e qual componente a utiliza na camada de *Front-end*, sendo posteriormente detalhada cada categoria.

- **Arquivos**: contem as rotas *Get* e *Post* que permitem respectivamente baixar e enviar arquivos do sistema. Utilizado por diversas visões do *Front-end*.
- ***Autenticação***: são rotas relacionadas ao login e cadastro dos usuários. A rota de login é a primeira a ser executada pelos usuários cadastrados para o início de execução de demais rotas. As rotas de cadastro envolvem solicitações de credenciais aos usuários e códigos de confirmação enviados ao e-mail institucional. Mais detalhes sobre estes fluxos são apresentados na seção posterior de fluxos do sistema.
- ***Envio de E-mails***: permite o envio de e-mails por uma requisição *Post*, na versão deste trabalho apenas a visão de aceitação pelo coordenador a utiliza no escopo do *Front-end*, os demais usos são relacionados ao *Back-end* no fluxo de solicitações.
- ***Orientadores***: permite a leitura de dados por meio do método *Get*, utilizado pelos alunos para a seleção do orientador participante do estágio.
- ***Página Dinâmica***: requisição criada e mantida para uso futuro, retorna dados sobre uma página dinâmica dado o seu identificador. Não é utilizado no *Front-end* da versão deste projeto devido aos dados da página dinâmica serem anexados aos resultados da rota de solicitação que possui a página dinâmica respectiva.
- ***Solicitação***: possui os métodos para criação, edição e leitura de solicitações de usuários. Possui lógica de restrição de perfis que podem realizar as solicitações para manter o fluxo de solicitações consistente.
- ***Solicitação de Orientadores***: rotas que permitem a seleção de orientadores por alunos pelo método *Put* e a aceitação do discente pelo orientador através do método *Patch* que atualiza o status da solicitação para aceito.
- ***Solicitações***: rotas de leitura de múltiplas solicitações, cada uma é respectiva a um perfil definido, pois os resultados e campos são específicos a estes.

Categoria	Verbo HTTP	Rota	Componentes do *Front-end* que utilizam
Arquivos	Post	/file	FileUpload, SelectFileUpload(Indiretamente), DynamicSolicitation(Indiretamente), Signatures(Indiretamente)
	Get	/file	FileDownload, DynamicSolicitation(Indiretamente), CoordinatorAcception(Indiretamente), Signatures(Indiretamente)
Autenticação	Get	/sign	Sign
	Post	/login	App, Login(Indiretamente)
	Post	/sign	Sign
	Put	/sign	Sign
Envio de E-mails	Post	/sendmail	CoordinatorAcception
Orientadores	Get	/advisors	AdvisorSelection
Página Dinâmica	Get	/dynamicpage	
Solicitação	Get	/solicitation	AdvisorAcception, AdvisorSelection, CoordinatorAcception, DynamicSolicitation, Signatures
	Post	/solicitation	AdvisorAcception, AdvisorSelection, CoordinatorAcception, DynamicSolicitation, Signatures
	Put	/solicitation	Home
Solicitação de Orientadores	Patch	/solicitation/advisor	AdvisorAcception
	Put	/solicitation/advisor	AdvisorSelection
Solicitações	Get	/solicitations/coordinator	Home
	Get	/solicitations/advisor	Home
	Get	/solicitations/student	Home
Tabela de Motivos	Get	/reasons	ReasonBox, CoordinatorAcception(Indiretamente)
Transições de Solicitações	Get	/solicitation/transitions	

Tabela 4 – Tabela com as rotas do *Back-end*.

- ***Tabela de Motivos***: retorna frases classificadas que representam motivos para a não aceitação de alunos pela coordenação. Utilizada na tabela de motivos na aceitação de alunos pela coordenação.

- ***Transições de Solicitações***: rota que permite a visualização de transições de um estado específico de uma solicitação. Assim como a rota de página dinâmica, seus dados são retornados juntamente à solicitação e é uma rota feita para uso futuro.

3.2.2.3 Detalhamento

Nesta subseção, são detalhadas as partes importantes da camada utilizando as classes apresentadas anteriormente para um aprofundamento de seu funcionamento. É incluída a descrição de etapas iniciais do sistema, da estrutura das rotas da API, do servidor de e-mails e do agendador de eventos. Detalhes sobre a máquina de estados são incluídos na seção de fluxos deste capítulo por envolver todas as camadas do sistema.

3.2.2.3.1 Início do servidor

O início do processo é importante por ser o responsável por inicializar todas as funcionalidades da camada. Este é detalhado no diagrama de sequência 18 e seus passos são descritos a seguir.

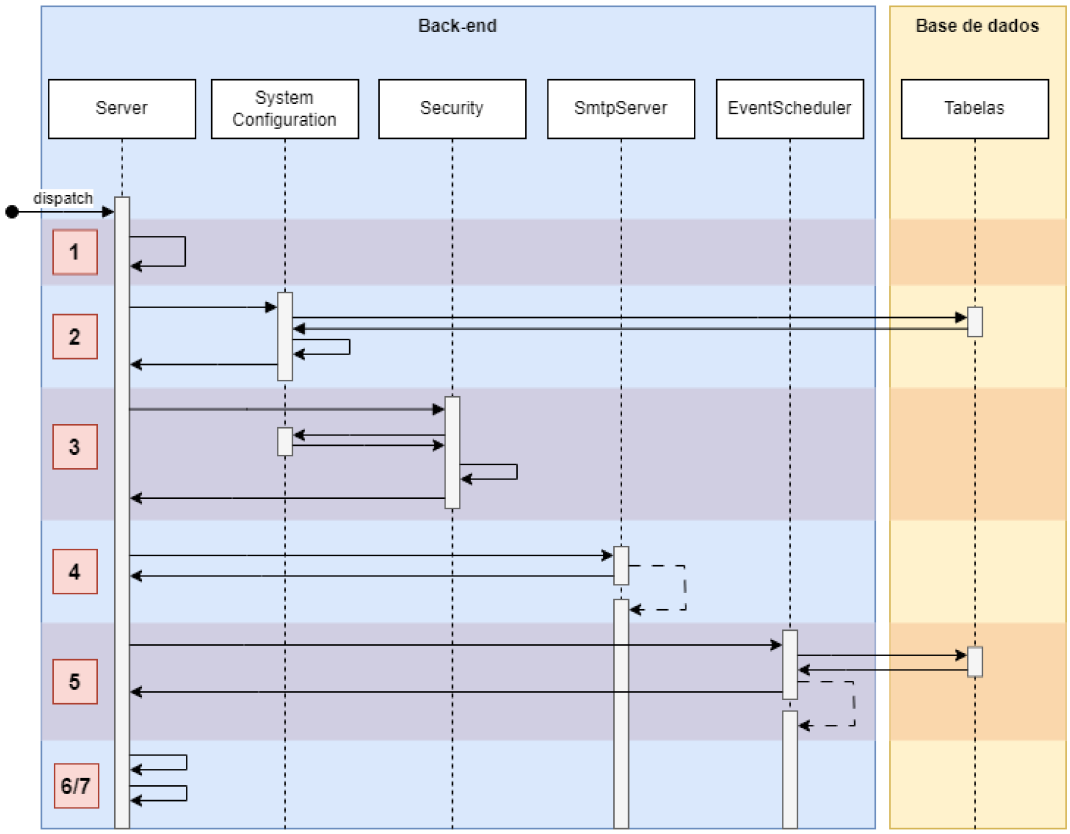


Figura 18 – Diagrama de sequência do fluxo inicial do servidor da camada de *Back-end*.

1. **Definições iniciais**: é criada a instância do servidor Flask e configurações iniciais, incluindo a inicialização do módulo de controle de variáveis de ambiente e configuração de CORS HTTP, se está habilitado o modo testes e configurações adicionais relacionadas ao banco de dados abstraídas pelo SQLAlchemy para sua respectiva inicialização.

2. **Carregamento das configurações do sistema**: após a inicialização do conector com o banco de dados, são carregadas as configurações de sistema em uma instância

da classe *SystemConfiguration*, que utiliza o banco de dados para carregar dados para reuso conforme mencionado na subseção anterior.

3. **Carregamento dos recursos de segurança do sistema**: com as configurações carregadas, é instanciada uma variável da classe *Security* que lhe são atribuídos detalhes de segurança do sistema para a geração de tokens JWT, baseados em pares de chaves e outros detalhes, como a cifragem de senhas de usuário com uso de *salga de senha*. Todos esses detalhes são explicados, posteriormente, na seção de fluxo de autenticação. Além disso, a instância utiliza dados das configurações da etapa anterior para acessar o local de armazenamento do par de chaves.

4. **Início da *thread* servidor SMTP**: após os detalhes de segurança definidos, é inicializada a thread que fica responsável por detalhes sobre o envio de e-mails. Esta é posicionada em uma instância que pode ser importada e acessada por outros módulos do sistema.

5. **Início da *thread* agendadora de eventos**: com o servidor SMTP iniciado, é realizada a inicialização da *thread* responsável pelo agendamento de eventos. Seus eventos podem ocasionar o envio de e-mails ou a realização de transições de estados da máquina de estados.

6. **Carregamento dos *Blueprints* e suas rotas**: após todos os componentes iniciados, são anexadas rotas da API ao servidor por meio do uso dos *Blueprints*, conforme mencionado anteriormente. Os *Blueprints* representam rotas desacopladas que funcionam como descritivos do funcionamento das rotas ao servidor que podem ser importados dos módulos respectivos e anexados ao servidor para a ativação de rotas.

7. **Inicialização do servidor**: É iniciada a instância do servidor Flask, utilizando um host e porta especificados nas variáveis de ambiente. A partir deste ponto, o servidor está pronto para aceitar requisições.

3.2.2.3.2 Estrutura das rotas da API

Embora cada rota possua funcionalidades e retornos específicos, todas seguem uma estrutura básica para sua definição considerando os módulos do sistema e componentes. Após a inicialização descrita anteriormente, os usuários podem realizar requisições que são respondidas pelo fluxo definido para as rotas, os quais são apresentados na Figura 19.

Inicialmente, a requisição chega pelo servidor e é direcionada ao *Blueprint* configurado para a rota em específico, esse trabalho é feito automaticamente pela classe Flask após a estrutura ser anexada ao servidor na inicialização. Após isso, a solicitação é encaminhada ao recurso específico com o verbo HTTP enviado pelo usuário.

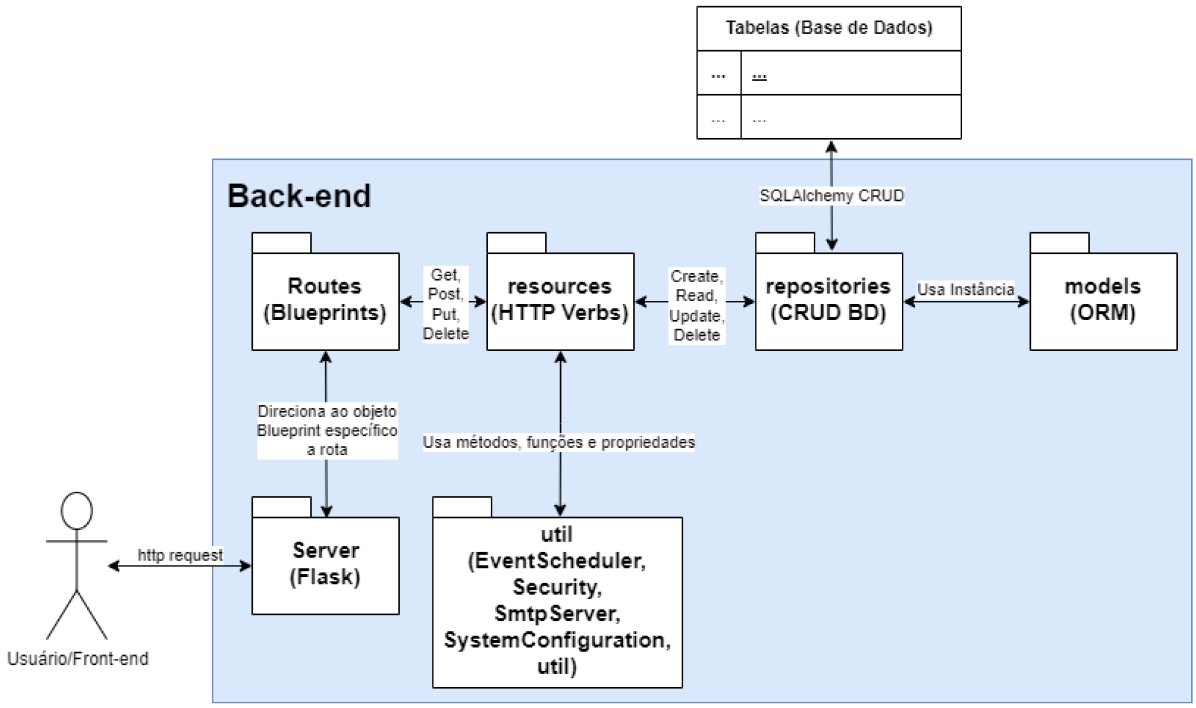


Figura 19 – Abstração do funcionamento das rotas da API do *Back-end*.

Os recursos são classes com métodos nomeados com os verbos HTTP responsáveis por aplicar as regras de negócio e acessar utilitários, incluindo a chamada a envio de e-mails para o servidor SMTP, agendamento de eventos, checagem de segurança, realização de chamadas aos repositórios, entre outras. Os recursos não realizam operações fora de seu escopo para manter a consistência da arquitetura de módulos, evitando a chamada direta ao banco de dados ou a definição de modelos em seu escopo.

Conforme mencionado, os recursos utilizam os utilitários e as operações dos repositórios para aplicar as regras de negócio. Os utilitários envolvem as classes apresentadas anteriormente, presentes no pacote útil na figura anterior. Já os repositórios são classes com métodos que implementam operações CRUD de comunicação com o banco de dados utilizando dos modelos como abstração. São desacopladas as regras de negócio e acesso ao banco de dados, separando estes em recursos e repositórios, respectivamente, para evitar inconsistências.

Os modelos são gerados por meio da aplicação do ORM com o uso do SQLAlchemy. Eles são criados como instâncias pelos repositórios e, posteriormente, empregados pelos recursos para efetuar as operações da rota. Cada modelo representa uma abstração direta de um objeto para a tabela do banco de dados, incluindo suas relações.

Um exemplo de como a solicitação funciona é o processo de login. Esse processo é acionado por uma solicitação HTTP Post do cliente, que é direcionada ao recurso representado pela classe Login com o método Post. Esse método da classe é responsável por verificar as credenciais do usuário usando a instância da classe utilitária *Security* e

por acessar os dados do usuário por meio do repositório e modelo correspondente. Se o usuário for autenticado com sucesso, o token de autenticação é retornado ao cliente.

3.2.2.3.3 Servidor SMTP

Conforme descrito no fluxo inicial do *Back-end*, ao iniciar o servidor Flask, é inicializada uma instância da classe *SmtpServer* que implementa o servidor. Ao carregar a instância com o método respectivo, é criada uma lista que permite receber e-mails e uma *thread* que atua como o servidor SMTP

A principal motivação para o uso de *threads* é devido ao fato de o envio das mensagens ser feito de forma assíncrona para evitar que o usuário necessite esperar que ele seja realizado para finalizar a requisição. Além disso, permite associar uma lista de e-mails ao objeto para que estes possam ser encaminhados em sequência.

Contudo, com a *thread* iniciada, os recursos podem solicitar a adição de e-mails encaminhando os dados do corpo da mensagem, assunto e destino, e estes serão formatados do lado do *SmtpServer* no formato padrão MIME. Esses mesmos dados serão inseridos em um modelo padrão utilizando técnicas de raspagem da Web[13] e serão encaminhados para o respectivo destinatário em sequência seguindo uma fila FIFO.

3.2.2.3.4 Agendador de eventos

Um recurso importante para o sistema e o fluxo de estágios é o agendador de eventos, com ele é possível tornar o sistema reativo a intervalos de tempo. O processo é iniciado de forma similar ao servidor SMTP, iniciando pela definição de uma instância que pode ser carregada para criar uma *thread* que permite a gerência dos eventos.

Ao contrário do servidor, na inicialização do agendador, são consultadas as tabelas referentes a eventos, isso é motivado por ser necessário recuperar eventos que não foram concluídos se o sistema falhar. Contudo, dada a inicialização e o carregamento, a *thread* responsável por aguardar até o próximo evento é inicializada iterando a cada segundo para a verificação de novos eventos adicionados e para a emissão de eventos com o intervalo do tempo atingido.

Após inicializada, os demais componentes do sistema podem acessar a *thread* ao utilizar seus métodos para adicionar eventos em sua lista de eventos ordenados no tempo, funcionando de forma similar ao servidor SMTP. E, cada evento possui um status associado indicando se este está em andamento, cancelado ou enviado, que podem ser atualizados pelos recursos.

[13] Tradução livre da expressão Web scraping

É possível alterar a frequência de atualização para tornar a reação mais precisa ou evitar gastos de recursos, as atualizações frequentes são necessárias devido à incerteza de quando serão adicionados novos eventos em conjunto com o comportamento das bibliotecas utilizadas.

3.2.3 Banco de dados

Conforme citado nas seções anteriores, é utilizado o banco de dados relacional MySQL na camada de persistência de dados, projetado de forma flexível para ser possível realizar mudanças no fluxo do sistema ao aplicar mudanças e criações de instâncias em suas tabelas. MySQL foi escolhido devido à necessidade da aplicação de armazenar dados mais sensíveis que devem seguir restrições e podem ser modeladas por relacionamentos, além de ser uma das bases de dados mais utilizadas.

Foram utilizadas 43 tabelas para a persistência de dados, pois grande parte da complexidade para tornar a aplicação flexível foi abstraída na camada. Na abstração presente na Figura 20, pode ser visualizada uma lista com os nomes das tabelas presentes no esquema de banco de dados e suas relações com outras tabelas.

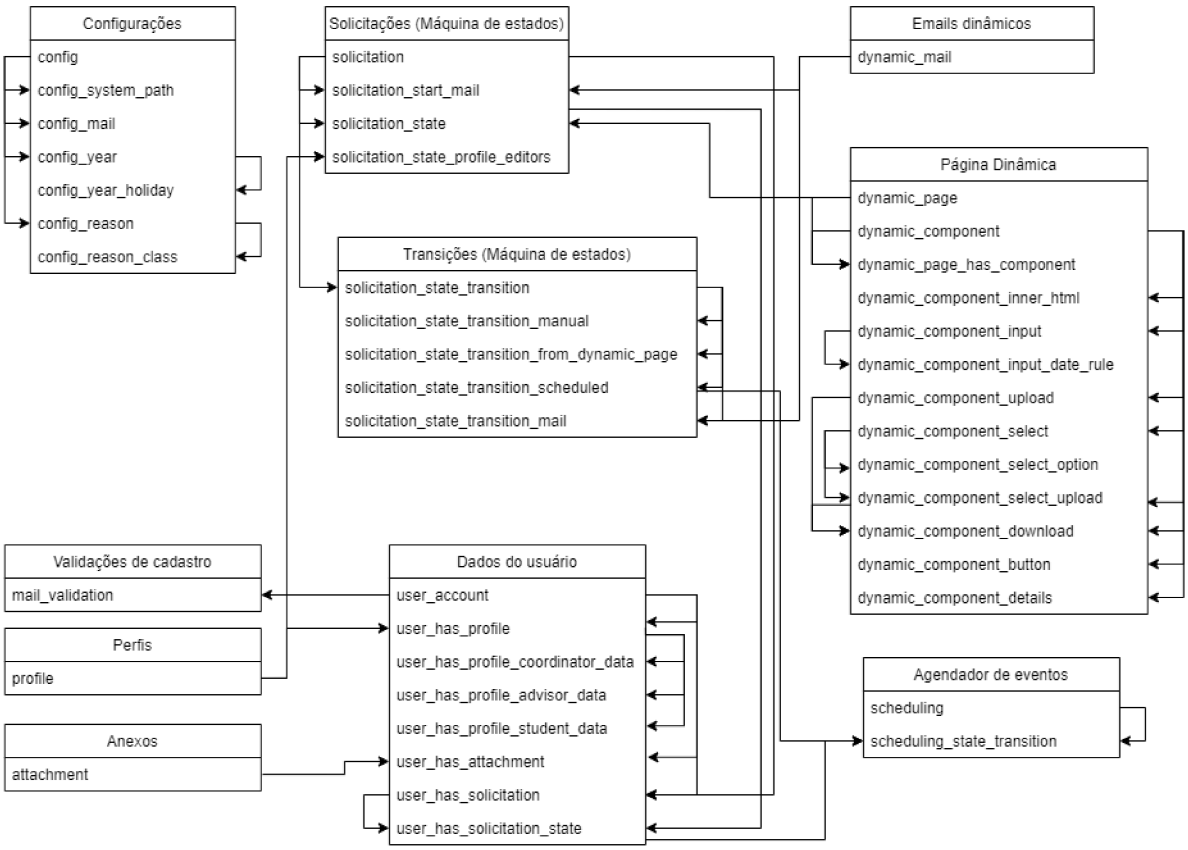


Figura 20 – Abstração simplificada do Banco de dados, apresenta lista de tabelas agrupadas em categorias e seus relacionamentos.

Devido à quantidade de tabelas, estas foram agrupadas em categorias na abstração anterior, as quais são descritas a seguir com o enfoque de fornecer um descritivo de como

é possível realizar alterações de funcionalidades do sistema, como a mudança de fluxos das máquinas de estados considerando mudanças em suas tabelas. Para mais informações sobre cada uma das tabelas, verifique o Apêndice B.

- ***Configurações***: é um conjunto de tabelas que possui configurações gerais da aplicação, como diretórios para armazenar arquivos de usuários, par de chaves, configurações de e-mails e classes de motivos utilizadas como pontos de auditoria a serem seguidos pela coordenação para indeferir o início de estágio de alunos. É possível alterar a localização dos arquivos externos da aplicação e alterar as opções de auditoria exibidas na tela de aceitação de início de estágios pela coordenação.
- ***Validação de cadastro***: possui apenas uma tabela e serve para armazenar dados a respeito de permissões de cadastros de novos usuários no sistema.
- ***Perfis***: funciona em conjunto com os dados do usuário para a definição de perfis de usuários. Sua população em conjunto com os dados de usuários fornece a possibilidade de definição de perfis personalizados com outros dados.
- ***Anexos***: assim como a categoria anterior, possui apenas uma tabela associada e funciona juntamente aos dados de usuário para definir restrições de acesso a anexos de usuários diferentes.
- ***Solicitações (Máquina de estados)***: é o conjunto de tabelas que abstrai o comportamento de uma máquina de estados possibilitando criar um fluxo que um usuário pode seguir. Os dados do usuário com relação à máquina de estados são populados nas tabelas de dados de usuário para que a única funcionalidade deste grupo seja a abstração da máquina de estados. Ao alterar ou adicionar novas instâncias às tabelas deste grupo juntamente ao grupo de transições, é possível criar novos fluxos, alterar ou definir comportamentos de estados e suas transições.
- ***Transições (Máquina de estados)***: funciona juntamente com as solicitações, mas são agrupadas em uma categoria diferente por organização e para destacar os diferentes tipos de transições que existem, estas são: transições manuais, por páginas dinâmicas, por e-mails e por eventos agendados.
- ***Dados do usuário***: representa as tabelas que armazenam dados do usuário como os dados de seu cadastro e dados relacionados a outras categorias como a categoria de perfis e de solicitações. Embora seja possível, não é recomendada a alteração de dados de usuários por serem críticos uma vez definidos e utilizados.
- ***E-mails dinâmicos***: possui definido em sua única tabela campos que representam um e-mail que pode ser utilizado em conjunto com as tabelas das demais categorias para o envio personalizado de e-mails. Ao alterar sua tabela, é possível modificar

e-mails existentes ou criar novos, os quais podem ser associados posteriormente em partes do fluxo de solicitações.

- ***Página Dinâmica***: é um conjunto de várias tabelas que permite descrever páginas utilizadas pelo fluxo, as páginas resultante são simples e possuem os componentes que possuem a tabela respectiva. É possível a criação e mudança de páginas dinâmicas ao alterar suas instâncias controlando suas relações com as tabelas de solicitações.
- ***Agendador de eventos***: funciona juntamente com a categoria de transições populada por tabelas que persistem os dados de eventos que ativam transições da máquina de estados com base em um intervalo de tempo. É possível alterar a relação de transições que são agendadas e também o intervalo de tempo que a transição leva para ser realizada pela *thread* de agendamento de eventos

3.2.3.1 Criação do esquema e suas tabelas

Para tornar a aplicação simples de iniciar em novos servidores, os scripts e dados iniciais para criação das tabelas estão presentes no repositório do *Back-end* para serem criados automaticamente na primeira vez que a aplicação é executada, desde que as ferramentas estejam instaladas e configuradas.

É verificado pelo sistema se o esquema existe e se não o sistema inicia uma transação e cria todas as tabelas necessárias, incluindo os dados iniciais que geram os fluxos das máquinas de estados detalhados nesta monografia. O nome do esquema utilizado pode ser facilmente alterado ao modificar o campo respectivo nas variáveis de ambiente.

3.3 Fluxos da aplicação

Após estudada cada uma das camadas, é importante analisar os principais fluxos da aplicação para o entendimento das funcionalidades das camadas em conjunto. Esta seção é dedicada a apresentar os quatro fluxos mais importantes da aplicação, que são: o fluxo de cadastro, o fluxo de autenticação, o fluxo utilizado para processar as solicitações dos usuários da aplicação e o fluxo principal envolvendo a máquina de estados.

3.3.1 Fluxo de cadastro

O fluxo de cadastro é o primeiro realizado pelos usuários e de relevante importância para a definição inicial de aspectos de segurança envolvendo os dados do usuário e possibilitando o acesso autenticado das rotas da API via tokens.

Esta aplicação é utilizada como base para o cadastro de seus usuários no e-mail institucional da universidade, devido ao fato de ela ser contextualizada no cenário da

Universidade Federal de Uberlândia. É considerado o preenchimento prévio das bases de dados com os dados básicos do aluno para reaproveitar as informações disponíveis em outras bases de dados da FACOM, não incluindo senhas, telefones e e-mails secundários, pois estes são preenchidos durante o cadastro. É apresentado, na Figura 21, o fluxo de cadastro considerando o e-mail institucional como uma de suas entidades para facilitar o detalhamento. Além disso, na Figura 22, são detalhadas as telas, incluindo o e-mail de confirmação enviado pelo usuário.

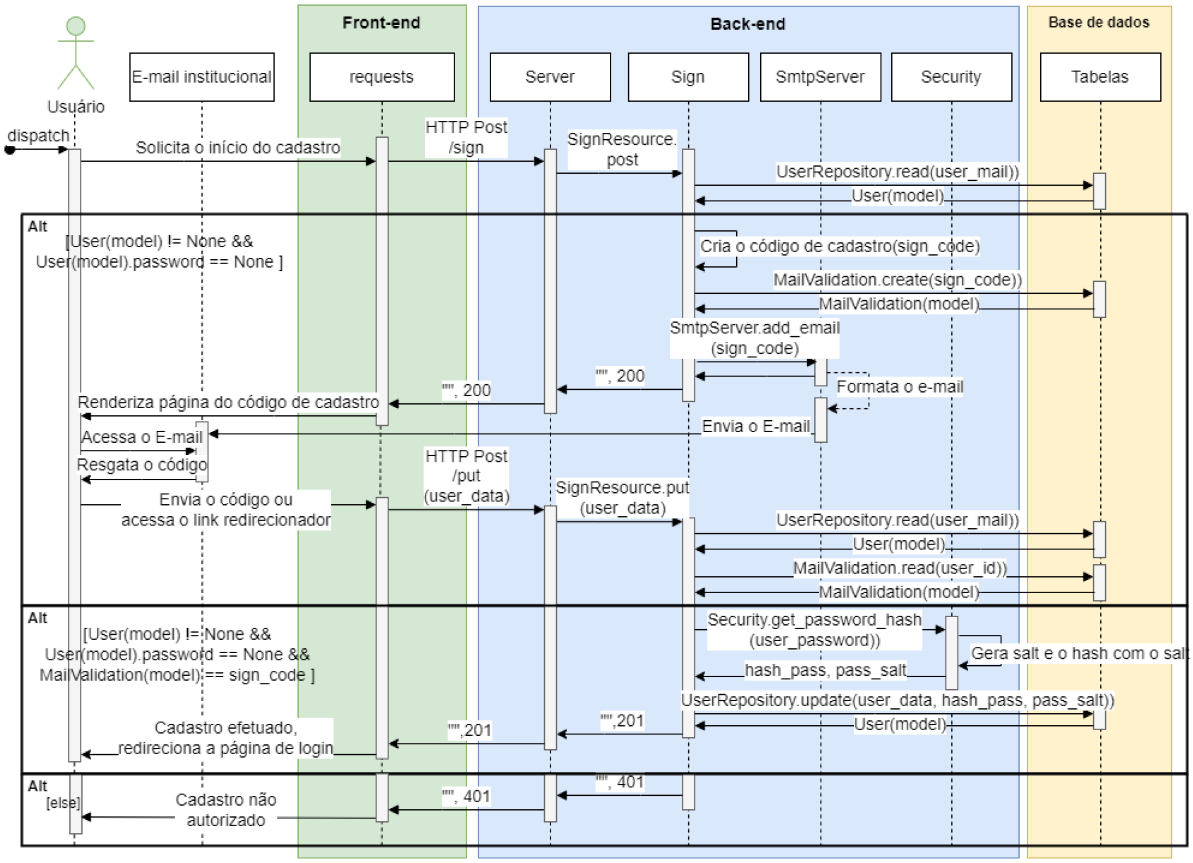


Figura 21 – Diagrama de sequência do cadastro do usuário.

Para realizar o cadastro, inicialmente o cliente informa o seu e-mail institucional solicitando o cadastro em sua página, o *Front-end* encaminha uma chamada ao *Back-end* utilizando seu script *requests*, que faz uma requisição *post* e chega ao recurso da classe *Sign*. Depois disso, ela atende a solicitações de cadastro através do direcionamento do servidor Flask para a sua rota em específico.

Em seguida, o recurso é responsável por verificar dados do usuário previamente preenchidos no sistema por administradores que inclui o e-mail institucional. Se os dados forem encontrados e for verificado que um cadastro não tenha sido realizado, isto é, não exista senha definida, o cadastro pode prosseguir e o recurso então gera e salva, na base de dados, um código de validação de e-mail. Esse e-mail é encaminhando ao servidor SMTP para o envio assíncrono e a chamada é retornada ao usuário com sucesso para a geração do código.

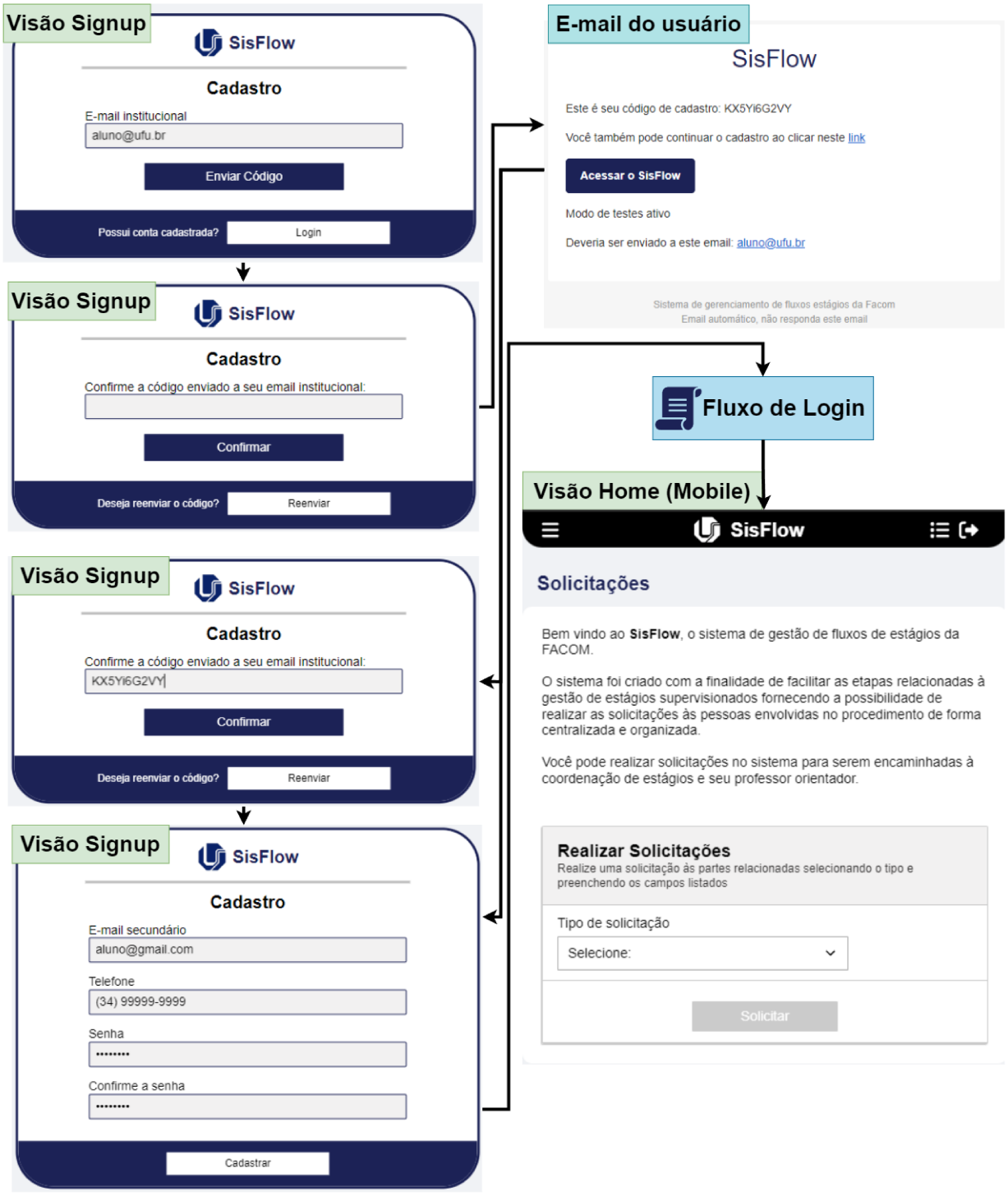


Figura 22 – Fluxo em telas do cadastro do usuário.

Com o código gerado e a chamada retornada, a visão de cadastros do *Front-end* redireciona a página de solicitação do código de cadastro, o qual o usuário pode preencher manualmente ao verificar a mensagem de e-mail ou pode clicar em um link presente no e-mail que redireciona para a próxima tela do fluxo de cadastro. Em ambos os casos, o usuário é direcionado à tela de preenchimento de dados para cadastro.

Nesta tela, o usuário insere dados que incluem o e-mail secundário, telefone, senha

e confirmação de senha e, então, envia uma segunda requisição do *Front-end* ao *Back-end* para a persistência dos dados cadastrais. Ao receber a requisição *Put* com os dados do usuário incluindo o código de cadastro, o servidor redireciona-os ao recurso específico responsável por realizar a leitura dos dados armazenados do usuário e da validação de e-mails para, então, realizar a verificação se os dados estão corretos.

Após a validação, o recurso utiliza a classe *Security* para gerar um hash da senha do usuário, incluindo o uso da técnica de concatenação da senha com um código aleatório, nomeado *salga de senha*[14] para evitar problemas de segurança envolvendo, principalmente, tentativas de acesso utilizando tabelas *arco-íris*[15]. Com o hash gerado, o usuário é atualizado com os novos dados incluindo a nova senha e seu código aleatório, e o cadastro então é concluído e o cliente direcionado à tela para efetuar a autenticação por login.

É importante ressaltar que existe uma terceira requisição não apresentada no fluxo para a simplificação do mesmo, esta é uma requisição *Get* realizada para validação do código de cadastro utilizada quando o usuário usa o campo do código de cadastro ao invés do link de redirecionamento para a tela de dados para cadastro. Isso é necessário para ocorrer a troca de telas somente após o usuário informar o código correto, assim como ocorre nos fluxos de cadastro de sites reconhecidos.

3.3.2 Fluxo de autenticação

Devido à solução ser disposta em camadas que se comunicam via APIs, é necessária a definição de recursos de segurança que protejam o acesso de terceiros não autorizados às rotas da API. Uma das formas mais simples e utilizadas é o uso de tokens que garantem a autenticidade de usuários após estes se autenticarem no sistema como, por exemplo, o uso de tokens JWT para a geração de assinaturas para verificação de autenticidade. Enquanto é apresentado na Figura 23 o diagrama de sequência, que detalha o fluxo utilizado para realizar a autenticação de usuários, na Figura 24 é apresentado as telas que o usuário percorre na camada de *Front-end*.

O fluxo inicia após o usuário realizar a solicitação de login enviando as suas credenciais. A partir disso, é criada uma solicitação HTTP post e enviada a rota de login para o *Back-end*. Após receber a requisição, o servidor Flask encaminha a solicitação ao recurso de login que, por sua vez, realiza a verificação e formatação dos parâmetros enviados pelo usuário.

Com os parâmetros verificados, são solicitadas informações da base de dados utilizando o repositório de usuários que retorna o modelo respectivo. Com o modelo, é possível verificar se o hash gerado, considerando a aplicação de um código aleatório também pre-

[14] Tradução livre da expressão password salting. Verifique detalhes da técnica em <https://en.wikipedia.org/wiki/Salt_(cryptography)>

[15] Tradução livre da expressão rainbow tables

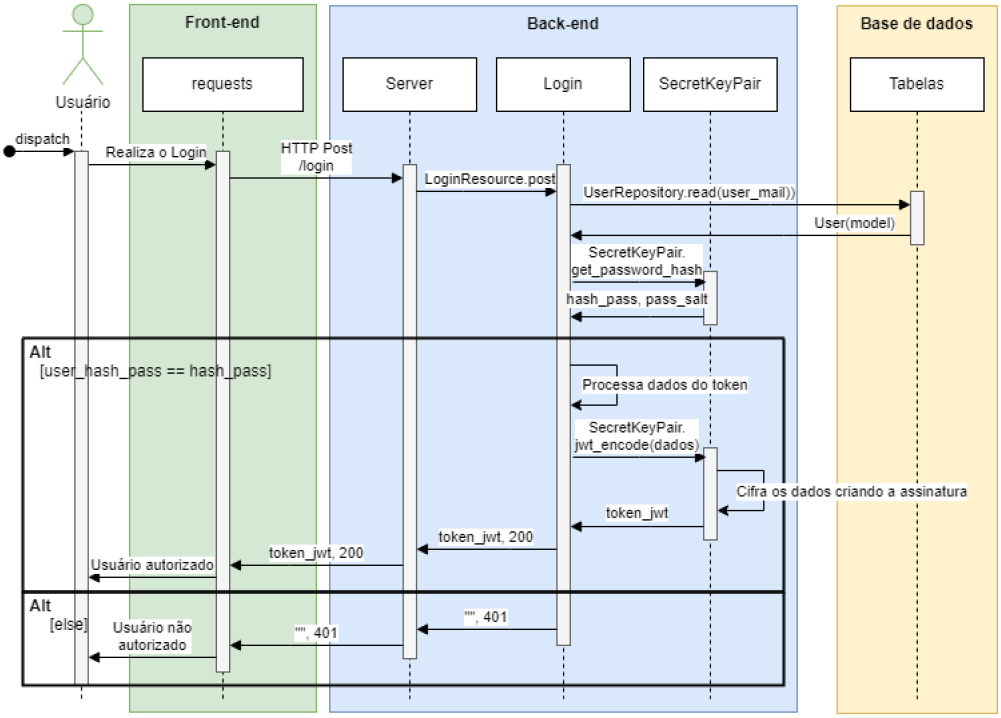


Figura 23 – Diagrama de sequência da autenticação do usuário.

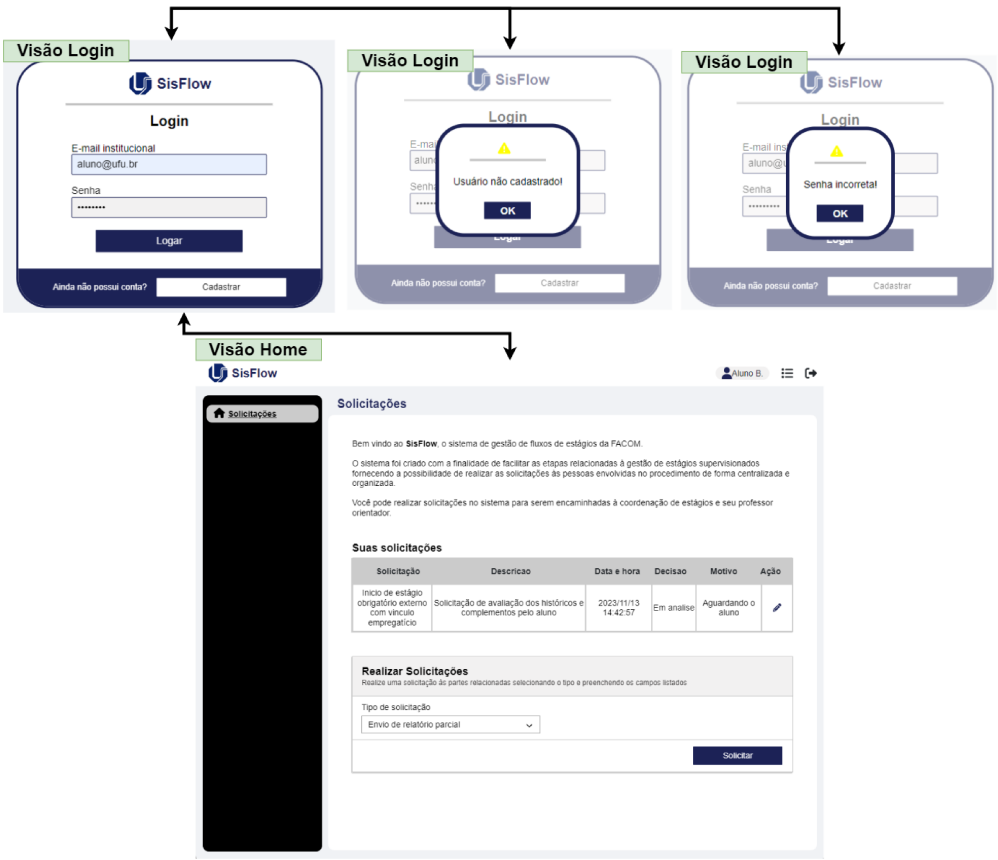


Figura 24 – Fluxo em telas da autenticação do usuário.

sente no modelo, é equivalente ao hash persistido na base de dados. Caso seja, os dados do token são gerados e cifrados criando o JWT e sua assinatura. Após criado o JWT, este

é retornado à aplicação do usuário, que passa a utilizar o JWT para a autenticação das demais rotas que possuem uso obrigatório do JWT. Caso as credenciais sejam inválidas, o token não é retornado e o usuário não tem acesso às demais rotas da API.

3.3.3 Fluxo geral de solicitações

As operações de solicitações são as mais críticas da solução devido à utilização de todas as funcionalidades mais importantes do sistema, incluindo máquina de estados, o agendador de eventos e o servidor de e-mails. É de suma importância que haja o seu estudo para o entendimento da solução. Na Figura 25, é apresentado um diagrama de sequência que auxilia na descrição do processo.

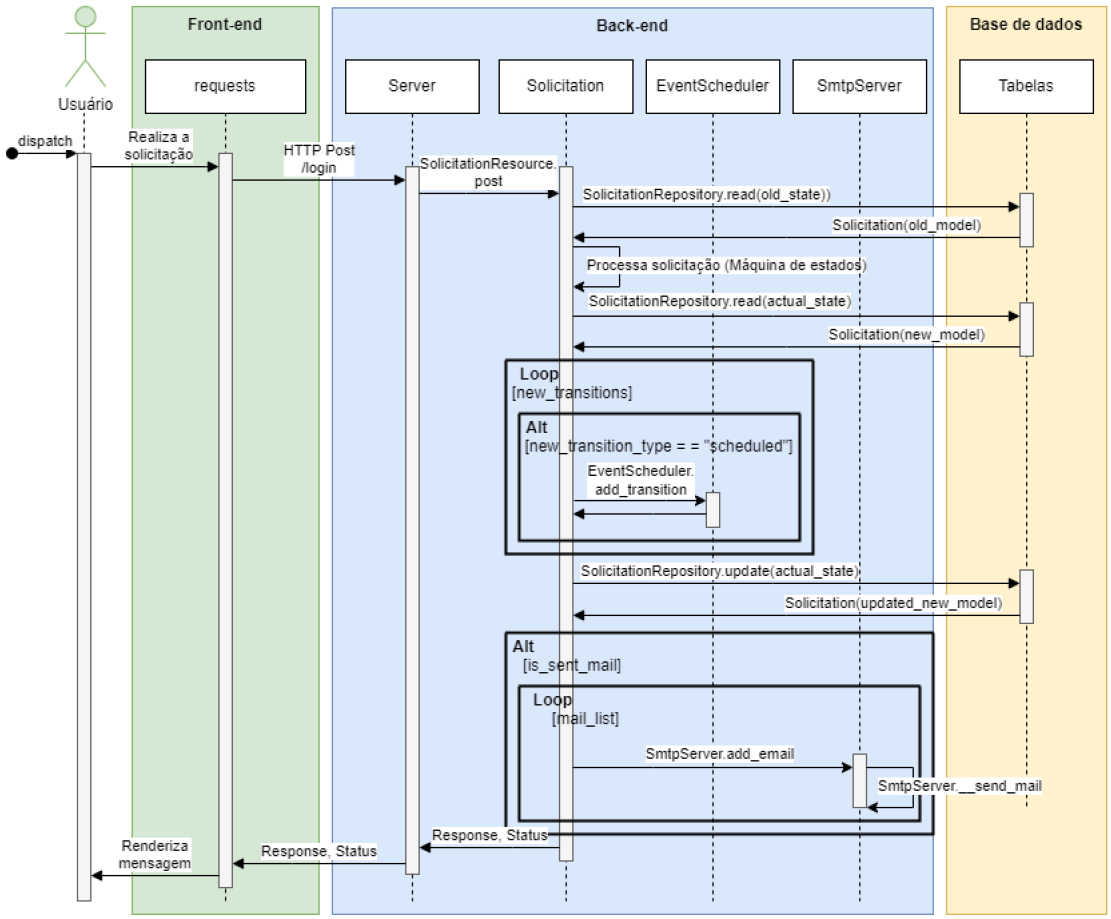


Figura 25 – Diagrama de sequência do processamento básico de solicitações.

Iniciando pela solicitação realizada pelo usuário, a solicitação é repassada do *Front-end* ao *Back-end* pelo seu script de envio de requisições. Em relação ao servidor, a instância da classe *Server* do Flask recebe a solicitação e então encaminha ao recurso de solicitações. Este, por sua vez, é responsável por fazer a formatação dos argumentos recebidos e a validação de cada argumento considerando o seu tipo.

Com os parâmetros formatados, o recurso realiza uma chamada ao repositório de solicitações para realizar a leitura de pontos relevantes para continuar o processamento da

solicitação. Com os dados prontos, é aplicado o fluxo da máquina de estágios que altera os estados da solicitação com base no tipo de transição que foi realizada e, então é feita a leitura dos dados do novo estado.

Após os dados do novo estado carregados em memória, é realizado o processamento de agendamento de eventos da solicitação do usuário, considerando as suas transições que são agendadas. Após isso, persistem os dados da solicitação com o novo estado e também de possíveis novos eventos agendados. Por fim, caso a transição que ocorreu possua o envio de e-mails associados, estes são enviados de forma assíncrona para não impedir o retorno da solicitação ao usuário.

3.3.4 Fluxo principal da máquina de estados

Como descrito na seção anterior, os fluxos de solicitação são os pontos focais da aplicação e desses, a etapa de execução da máquina de estados é a mais relevante, sendo, portanto, uma das etapas mais cruciais de toda a aplicação. Conforme também exemplificado anteriormente, a máquina de estados é posicionada no fluxo de solicitações e é baseada no Algoritmo 2 apresentado, e é relativa às entradas do usuário e eventos temporais. A máquina é abstraída nas tabelas do banco de dados categorizadas na categoria de *Solicitações* e *Transições* conforme mostra a abstração 20 e, a partir de suas instâncias, é possível criar um fluxo no qual os usuários poderão interagir.

Outro ponto relevante, considerando os aspectos de contexto alvo da aplicação, é o fluxo de início de estágio. Este é o mais dispendioso para os usuários envolvidos no processo de estágios. Isso porque envolve diversas etapas para deferimentos e validações documentais, além do aceite de orientação pelos professores orientadores e do processo de assinaturas. Visto isso, juntamente às afirmações sobre a máquina de estados anteriores, é importante o estudo de um fluxo da máquina de estados considerando o fluxo inicial de estágios. É apresentado, na Figura 26, o diagrama de transição de estados baseado no fluxo final criado na base de dados do sistema. Para a verificação das telas que compõem cada um dos estados, verifique o Apêndice C.

O processo se inicia a partir da solicitação do aluno, evento que posiciona a máquina em seu primeiro estado, responsável por renderizar uma página dinâmica que permite a coleta de documentação do aluno. Posteriormente, essa documentação passa por validação pela coordenação de estágio no segundo estado, por meio de uma página manual que permite verificar a aptidão do aluno para o procedimento. Após a aprovação da documentação, a máquina avança para o terceiro estado, permitindo ao discente solicitar orientação de um dos professores cadastrados na base de dados.

Uma vez solicitada a orientação, a máquina transita para seu quarto estado, onde aguarda o aceite do discente pelo orientador. Com o discente aceito, a máquina prosse-

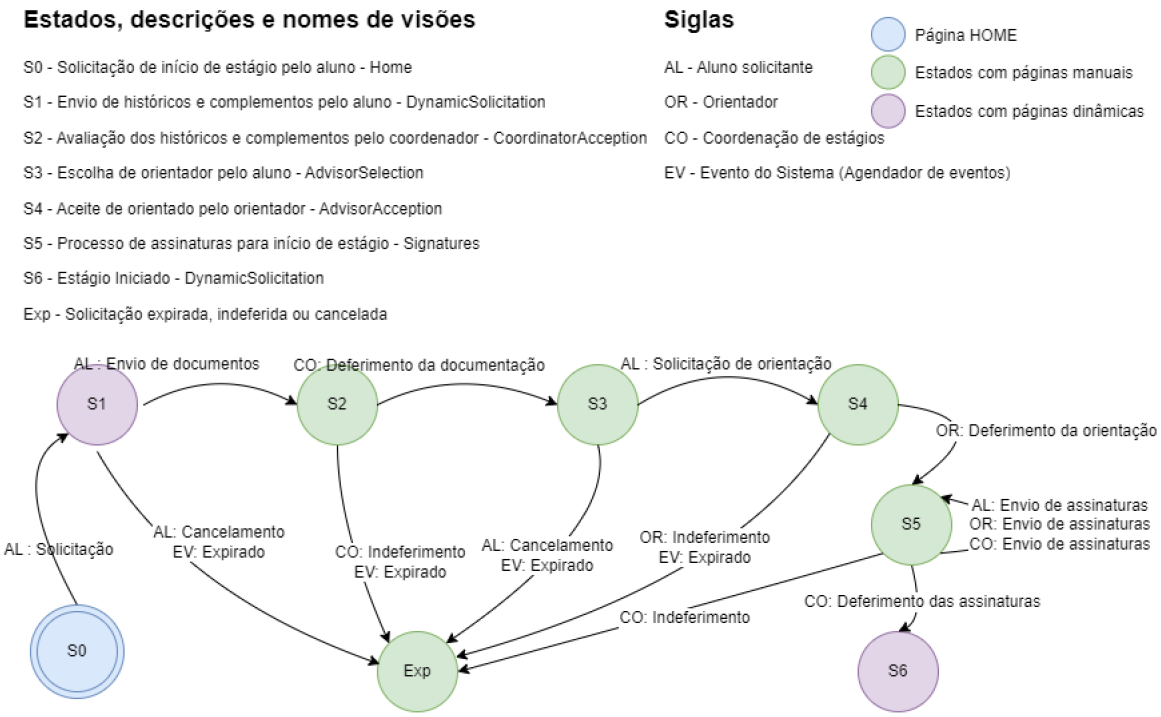


Figura 26 – Diagrama de transição de estados da solicitação de início de estágio.

gue para o próximo estado, onde ocorre o envio e assinatura dos documentos essenciais para iniciar o estágio. Após a conclusão desse processo de assinatura, o aluno se torna apto a iniciar o estágio e caso ocorra algum problema em qualquer estado, é possível realizar cancelamentos ou indeferimentos da solicitação para o perfil de usuário específico, direcionando a máquina a um estado de exceção.

3.4 Disponibilização

Conforme descrito no início do desenvolvimento, a solução é disponibilizada em dois repositórios para o desacoplamento, considerando os diferentes escopos das camadas da arquitetura e suas tecnologias, sendo que, a camada de *Back-end* inclui os descritivos para criação das tabelas da base de dados. Visto isso, a disponibilização das três camadas é desacoplada em três processos diferentes incluindo os servidores de *Front-end*, *Back-end*, e base de dados.

A aplicação pode ser disponibilizada localmente utilizando apenas um dispositivo, ou ela pode ser separada em computadores diferentes. No cenário deste trabalho, a solução foi disposta utilizando três servidores na nuvem, incluindo dois de seus provedores, sendo eles: Railway, para a base de dados e o *Back-end*, e o Netlify, para o *Front-end*. Decidiu-se optar pelos servidores na nuvem devido a diversas qualidades que estes fornecem, principalmente pela simplicidade, baixo preço e pela disponibilização automática realizada quando é versionado novas alterações nos ramos principais dos repositórios. É apresentado,

na Figura 27, a arquitetura resultante da disponibilização envolvendo as plataformas.

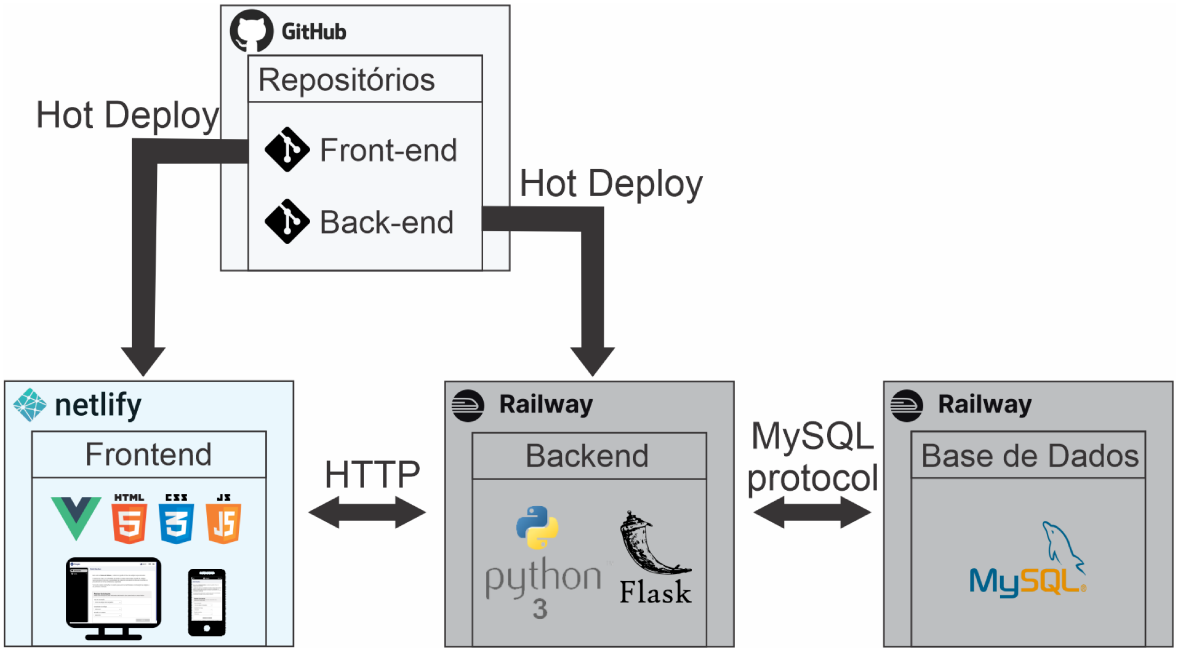


Figura 27 – Abstração da disponibilização da solução do projeto.

É importante ressaltar que embora a disponibilização da versão final foi realizada no ambiente da nuvem, é possível a disponibilização em servidores locais, entretanto, é importante utilizar recursos que tornem a aplicação escalável e possibilite a recuperação de desastres de forma automática das quais são detalhadas por Ford e Richards (2020) e são abstraídas para a solução deste trabalho pelas plataformas da nuvem. Para mais detalhes sobre os comandos para inicialização dos repositórios e um funcionamento breve é possível consultar a seção de início rápido dos repositórios desta solução além de todo o detalhamento realizado neste trabalho.

3.5 Discussão

Nesta seção é realizada a discussão por meio de uma análise crítica do sistema ao relacionar o resultado definido pela sua arquitetura, camadas e componentes com as características operacionais apontadas por Ford e Richards (2020) além de outras relevantes para ressaltar as qualidades e pontos de melhoria para a solução resultante.

- **Confiabilidade ou segurança**: Diversos mecanismos básicos de segurança foram adotados na aplicação que permitem, além de restringir acesso não autorizado de recursos externos e perfis de usuários inválidos às requisições da API, a proteção básica dos dados do usuário ao utilizar a autenticação por meio do uso de tokens JWT assinados por chaves privadas. Entretanto, em cenários de produção, é necessária a verificação de outros pontos relevantes, como a restrição de injeção de código malicioso, uso de ferramentas de análise de vulnerabilidades, entre outros.

- **Continuidade ou recuperação de desastres**: A recuperação de desastres é totalmente abstraída pela plataforma na nuvem. Estas são responsáveis por distribuir nós de redundância geometricamente separados para maior garantia de recuperação dos recursos. Entretanto, no cenário desta aplicação, não foram utilizados tais recursos devido aos altos custos associados. No cenário de disponibilização local, é importante a disponibilização distribuída da aplicação considerando diferentes nós separados geograficamente.

- **Desempenho**: No cenário em que esta aplicação é disponibilizada, o desempenho é garantido através da escalabilidade horizontal que a plataforma da nuvem fornece. À medida que existe um acréscimo ou decréscimo de requisições, o desempenho é ajustado. É importante apontar que, no caso de disponibilização local, medidas para a verificação de desempenho devem ser analisadas para o ajuste de recursos de acordo. Outro ponto positivo para a característica é a configuração de processos que demandam bastante processamento de forma assíncrona configurados na aplicação, como o envio de e-mails e a checagem de eventos que permitem o desempenho do ponto de vista das requisições e retorno aos usuários.

- **Disponibilidade**: A disponibilidade da aplicação é bastante relacionada com o ambiente em que a aplicação é disponibilizada. No cenário da divulgação em plataformas na nuvem, geralmente todos os recursos para a orquestração de diferentes nós distribuídos para a aplicação podem ser abstraídos do usuário, tornando a característica simples de se alcançar. Nesta solução, considerando a sua disponibilização final, as plataformas abstraem os recursos permitindo alta disponibilidade. Entretanto, caso a aplicação seja divulgada localmente, recursos devem ser configurados de acordo para distribuir a aplicação horizontalmente de acordo com métricas para a garantia da característica.

- **Escalabilidade**: A escalabilidade é atingida horizontalmente ao utilizar a plataforma na nuvem que adapta os seus recursos conforme a demanda, como mencionado anteriormente.

- **Manutenibilidade**: Considerando aspectos como a segurança e custos de longo prazo, como esta aplicação não é uma solução de longo prazo e envolve recursos de segurança básicos, a característica é um ponto de melhoria no sistema. Entretanto, toda a estrutura de módulos e a arquitetura utilizada envolve modelos reconhecidos pela comunidade, que permitem o reuso devido ao desacoplamento ao nível de funcionalidade, sendo pontos positivos para a característica.

- **Recuperabilidade**: Não são apontados ou documentados detalhes a respeito do funcionamento da recuperação de desastres por serem abstraídas pela plataforma da nuvem.

- **Robustez**: Diversos componentes são pensados para a robustez da aplicação no caso de recuperação de dados em fluxos interrompidos, como, por exemplo, a persistência de dados da máquina de estados e agendador de eventos para que, caso ocorra falha no sistema, estes sejam recuperados. Além disso, as camadas de *Front-end* e *Back-end* não mantêm o estado de seus dados salvos em memória volátil entre fluxos, conforme deve ser a interface e a API REST. Apesar dos pontos apresentados, não são realizados backups ou réplicas do banco de dados, sendo um ponto crítico de melhoria a ser considerado.

- **Usabilidade**: No escopo dos usuários, a aplicação abstrai e simplifica todo o fluxo envolvendo os processos de estágios em interfaces simples e responsivas, sendo uma qualidade presente na aplicação. Considerando o escopo de administradores, a usabilidade possui pontos de melhoria, conforme os aspectos de configuração dos fluxos por serem feitos diretamente no banco de dados e não em uma interface simplificada.

- **Versionamento**: O versionamento da aplicação desacopla em 2 camadas isoladas que podem ser trabalhadas por equipes de responsabilidades diferentes, além de possibilitar a disponibilização eficiente por evitar camadas não utilizadas no mesmo, sendo um ponto de qualidade da aplicação.

Além dos pontos apresentados anteriormente, é relevante a discussão crítica a respeito de cada uma das camadas que o sistema foi organizado, sendo realizada a seguir:

- **Base de dados** - A camada envolve a persistência de informações que abstraem objetos complexos como a máquina de estados e o agendador de eventos e por isso pode ser difícil o seu entendimento, embora este trabalho se baseie em fornecer uma boa documentação. Por esse motivo, essa camada pode possuir uma curva de aprendizado complexa para novos administradores, porém, após o seu entendimento, a camada fornece funcionalidades úteis para a alteração dos fluxos da aplicação de forma simplificada.

  Além disso, existe a tentativa de utilizar corretamente os relacionamentos que a base de dados fornece para tornar a imposição de regras de persistência de forma simplificada, como o uso de diversas chaves estrangeiras em algumas tabelas que permitem uma maior consistência dos dados armazenados. Outro ponto relevante é a criação automática da tabela ao se executar a aplicação pela primeira vez, funcionalidade que pode facilitar o cotidiano de novos administradores.

- **Back-end** - A camada mais trabalhada em todo o projeto, e é arquitetada de forma profissional, utilizando o *framework* MVC, sendo, portanto, uma solução interessante a longo prazo. Existem alguns recursos, como a máquina de estados e o agendador de

eventos, que podem necessitar de configurações adicionais para um melhor controle de fluxo geral da aplicação.

O uso de *blueprints*, das *threads* para desacoplamento de fluxos demorados, do *framework* MVC e técnica ORM são qualidades da camada ao mesmo tempo que adiciona complexidade. Entretanto, a curva de aprendizado desta camada é mais simples, por ela possuir todo o fluxo codificado acessível incluindo diversos comentários.

- **Front-end** - A camada visual é responsiva, simples e eficiente e, por esta não ser uma aplicação para produção, pode ser necessária a adição de mais informações que ajudem o usuário a se posicionar em relação aos fluxos nos quais está inserido e de seu funcionamento, além de detalhar melhor as informações importantes para os orientadores e coordenadores.

  Considerando os detalhes de sua arquitetura, essa camada é composta de módulos definidos por um modelo oficial do seu framework, o que facilita a sua evolução no longo prazo. Por outro lado, observando os módulos mais internos como os de componentes, pode ser necessária a maior modularização para evitar que a pasta possua uma vasta quantidade de componentes com funcionalidades diferentes.

Em linhas gerais, a solução é arquitetada utilizando tecnologias, ferramentas e modelos reconhecidos. Além disso, a sua implementação se baseia em flexibilidade, reuso, manutenção e evolução. Grande parte das qualidades da solução é baseada em como ela é disponibilizada, principalmente em características que necessitam da disponibilização distribuída e horizontalmente escalável. Existem muitas melhorias a serem realizadas que são necessárias para atender o fluxo completo alvo desta solução, porém as funcionalidades mais complexas foram implementadas, incluindo a máquina de estados, o envio de e-mails e o agendamento de eventos.

4 Conclusão

Portanto, o presente trabalho descreve uma solução Web responsiva que serve como base para a automatização de alguns dos principais fluxos de estágios para a FACOM. A divisão e detalhamento da arquitetura em suas três principais camadas, a apresentação de seus fluxos, a disponibilização e, por fim, a análise crítica, contribuem para a documentação do projeto e para a realização do objetivo principal da monografia.

O primeiro objetivo específico, que é listado juntamente aos outros na Seção 1.1.1, é alcançado por meio da apresentação realizada em todo o desenvolvimento, principalmente, nas seções de visão geral do desenvolvimento. O conteúdo desta seção é fundamental para ilustrar os requisitos e sua interação com os usuários por meio de diagramas, possibilitando a contextualização do sistema. Isso facilita a compreensão dos leitores em relação ao cenário para o qual este projeto é destinado, ao mesmo tempo em que os introduz à estrutura da arquitetura da solução e suas motivações.

O segundo objetivo é conquistado pela apresentação de cada uma das camadas divididas entre: *Front-end*, *Back-end* e base de dados, nas subseções de detalhamento das camadas. A divisão ajuda a compreensão isolada, de cada uma das camadas, que podem ser focalizadas por times que atuam na camada respectiva para manutenção em seus componentes. Adicionalmente, nos apêndices desta monografia, encontram-se detalhes mais aprofundados sobre os principais componentes das camadas, fornecendo uma apresentação mais detalhada de cada um deles.

Além disso, o terceiro objetivo específico é alcançado por meio do descritivo realizado na seção de fluxos do sistema. Essa seção é crucial para a compreensão integral do sistema, uma vez que, enquanto as descrições nas seções de camada apresentam funcionalidades isoladas, a seção de fluxos destaca processos que atravessam todas as camadas do sistema, possibilitando uma visão abrangente do sistema em sua totalidade. Adicionalmente, os fluxos de solicitações e de máquina de estados contribuem para a análise de fluxos mais complexos da aplicação, facilitando a sua compreensão.

A seção de discussão está alinhada com o último objetivo específico. As análises apresentadas nessa seção têm como base características arquiteturais relevantes para sistemas modernos, conforme a revisão realizada. Elas funcionam como um relato significativo dos principais pontos positivos e áreas de aprimoramento da solução. A análise contribui para o destaque de pontos positivos e de melhoria dos quais o sistema implementa.

No geral, a aplicação apresenta qualidades com enfoque em características específicas, ressaltando principalmente a manutenibilidade, a evolução e reuso, além de possuir outras incluídas pela disponibilização na plataforma na nuvem.

Contudo, a solução incorpora os principais e mais complexos componentes, constituindo um sólido protótipo que implementa os requisitos apontados na contextualização e que permite a continuidade e o aprimoramento da solução. Isso se deve ao uso de ferramentas, linguagens e modelos que favorecem o reuso e a evolução efetiva da aplicação. Adicionalmente, a solução teve um impacto significativo no desenvolvimento intelectual do seu programador. Seu processo de criação proporcionou uma ampla aprendizagem sobre a arquitetura de soluções, fazendo uso de práticas consolidadas na literatura, estudadas na universidade e aplicadas no contexto organizacional.

Adicionalmente, devido à evolução da solução ao longo do desenvolvimento, que envolveu a utilização de diferentes tecnologias e ferramentas em várias versões, foram possíveis a avaliação e decisão de quais são mais adequadas para cenários específicos. Isso não apenas contribui para o aprimoramento da solução final, mas também influenciará as escolhas em outros projetos ao longo da carreira do desenvolvedor desta aplicação.

4.1 Trabalhos futuros

Embora a aplicação resultante possua parte dos principais requisitos de seu contexto alvo, esta não possui toda a complexidade que envolve o fluxo de estágios dada a diversidade de perfis, fatores e funcionalidades relacionadas, servindo de base à evolução para uma versão mais completa. Nesta seção, destacam-se as principais evoluções possíveis para o sistema, apresentando uma lista de melhorias que podem ser implementadas para alcançar os complementos desejados:

- **Adicionar mais componentes à página dinâmica**: na versão deste trabalho, a página conta com 8 componentes diferentes e pode ser necessária a adição de mais componentes para que, em conjunto com a gestão do banco de dados em interfaces, seja possível a criação de páginas dinâmicas altamente personalizáveis diretamente em interfaces aos administradores.
- **Aplicar restrições para a definição de perfis dinâmicos**: é essencial impor restrições para garantir o preenchimento adequado das tabelas de perfis do usuário que possibilitam a criação de perfis dinâmicos. Na versão atual desta monografia, os campos necessários para esses perfis foram implementados, no entanto, a lógica de restrição de campos específicos ainda não foi incorporada.
- **Codificação de retornos das *threads* ao fluxo principal**: é implementada a comunicação apenas no sentido do fluxo principal às *threads* de envios de mensagens de e-mail SMTP e ao agendador de eventos, sendo importante a implementação da comunicação no sentido contrário possibilitando maior controle das respostas desses fluxos.

- **Criação de backups contínuos da base de dados**: uma qualidade desta aplicação é não manter o estado armazenado nas camadas de *Front-end* e *Back-end*, focalizando toda a persistência do lado do banco de dados. Devido a isso, é de grande relevância o uso de ferramentas de backup do banco de dados realizadas em intervalos de tempo específicos para a possibilidade de recuperação no caso de falhas.

- **Definição de Relatórios**: podendo compor uma parte fundamental do sistema, a criação de relatórios, através da utilização dos dados persistidos, é de grande relevância para a coordenação de estágios e para orientadores que podem solicitar, por exemplo, a sua lista de orientados em um intervalo de tempo específico.

- **Disponibilização local e distribuída**: como discutido no capítulo correspondente, a implementação desta solução ocorre em uma plataforma de nuvem. No contexto da FACOM, a disponibilização local pode ser significativa. Para otimizar essa abordagem, é crucial incorporar características como escalabilidade e disponibilidade, entre outras que são obtidas ao distribuir a solução.

- **Edição de dados cadastrais dos usuários**: a funcionalidade não foi definida para esta versão e é de importância para versões futuras por possibilitar a alteração de senhas, endereços de e-mail secundários, telefones, entre outros.

- **Edição dos fluxos implementados**: embora os fluxos presentes na versão deste trabalho atuem em suas partes essenciais, é necessário adicionar maior complexidade para o funcionamento no caso de desvios do fluxo e atuação de novos perfis, como, por exemplo, a adição de estados realizados pelo supervisor de estágio, permitindo a assinatura destes no processo.

- **Gestão da complexidade do banco de dados em interfaces**: embora toda a complexidade implementada permite a definição de fluxos configuráveis e dinâmicos, estes ainda são feitos diretamente no banco de dados. Assim, seria relevante criar interfaces que possibilitem realizar operações CRUD ao se interagir com o *Front-end* para o perfil de administradores envolvendo a criação das máquinas de estados, mudanças de configurações, edições em mensagens de e-mail dinâmicas, entre outros.

- **Implementação de testes automatizados**: essa funcionalidade é de extrema importância para qualquer sistema e estava originalmente planejada no escopo inicial desta solução. No entanto, devido a restrições de prazo, não foi incluída nesta versão e pode ser incorporada em iterações subsequentes.

- **Melhorias na documentação**: outro aspecto relevante é a documentação do sistema, embora muito trabalho foi realizado para a escrita desta monografia e criação

de documentos inclusos no repositório do projeto, existem pontos bastante específicos que podem ser melhor detalhados, por exemplo, a criação de documentação da API, utilizando outras ferramentas das utilizadas neste projeto e a criação de um modelo oficial de documentação seguindo um protótipo reconhecido.

- **Modularização de componentes da arquitetura**: um desafio potencial na estrutura, à medida que a aplicação se expande, é o acúmulo de vários componentes em uma única pasta. Pode ser importante considerar a modularização desses componentes em subpastas para uma organização mais eficiente, especialmente na estrutura de componentes do *Front-end*.

- **Novas solicitações**: outra possibilidade é a adição de novas solicitações que podem ser realizadas por outros perfis além dos alunos presentes no contexto, como, por exemplo, a adição de solicitação de encerramento de estágio.

- **Prevenir e-mails de serem marcados como SPAM**: embora o envio de mensagens de e-mail através do SMTP tenha sido configurado, elas são classificadas como SPAM ao serem enviadas para endereços institucionais da universidade. É crucial colaborar com a equipe técnica para evitar esse problema.

Referências

AMIT. **All You Need to Know About UML Diagrams: Types and 5+ Examples**. 2018. Acesso em: 10 out. 2023. Citado na página 25.

AWS. **AWS Step Functions**. 2023. Disponível em: <https://docs.aws.amazon.com/step-functions/latest/dg/welcome.html>. Acesso em: 10 out. 2023. Citado na página 37.

BASUMALLICK, C. **What Is JSON? Meaning, Types, Uses, and Examples**. 2022. Disponível em: <https://www.spiceworks.com/tech/devops/articles/what-is-json/>. Citado na página 30.

BIGELOW, S. J. **What are the types of requirements in software engineering?** 2020. Disponível em: <https://www.techtarget.com/searchsoftwarequality/answer/What-are-requirements-types>. Acesso em: 10 out. 2023. Citado na página 20.

CARMELY, T. **Using finite state machines to design software**. 2009. Disponível em: <https://www.embedded.com/using-finite-state-machines-to-design-software/>. Acesso em: 10 out. 2023. Citado na página 36.

DEACON, J. Model-view-controller (mvc) architecture. 2009. Disponível em: <http://www.johndeacon.net/john-deacon/articles/model-view-controller-architecture/>. Citado na página 24.

EMERSON. **Application Design Patterns: State Machines**. 2022. Disponível em: <https://www.ni.com/en/support/documentation/supplemental/16/simple-state-machine-template-documentation.html>. Acesso em: 10 out. 2023. Citado 2 vezes nas páginas 15 e 35.

FORD, N.; RICHARDS, M. **Fundamentals of Software Architecture: An Engineering Approach**. [S.l.]: O'Reilly Media, Inc., 2020. ISBN 9781492043454. Citado 6 vezes nas páginas 6, 21, 22, 23, 39 e 68.

FOWLER, M. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. [S.l.: s.n.], 2003. ISBN 978-0-321-19368-1. Citado 2 vezes nas páginas 26 e 28.

GOURLEY, D.; TOTT, B. **Book - HTTP - The Definitive Guide**. [S.l.: s.n.], 2002. Citado na página 32.

HENDERSON, T. **Understanding HTTP Requests: Structure, Methods Examples**. 2023. Disponível em: <https://www.linode.com/docs/guides/http-get-request/>. Citado na página 29.

IBM. **What is three-tier architecture?** 2022. Disponível em: <https://www.ibm.com/topics/three-tier-architecture>. Acesso em: 10 out. 2023. Citado na página 23.

JONES, C. **Software Engineering Best Practices: Lessons from Successful Projects in the Top Companies**. [S.l.: s.n.], 2010. ISBN 978-0-07-162162-5. Citado na página 25.

KRIGER, B. **ORM (OBJECT RELATIONAL MAPPER) – SAIBA O QUE É E IMPORTÂNCIA NA PROGRAMAÇÃO**. 2023. Disponível em: <https://kenzie.com.br/blog/orm/>. Citado na página 24.

LIMA, C. **O que é JWT?** 2021. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-jwt>. Citado 2 vezes nas páginas 6 e 31.

LUCHANINOV, Y. **Web Application Architecture in 2023: Moving in the Right Direction**. 2022. Disponível em: <https://mobidev.biz/blog/web-application-architecture-types>. Acesso em: 10 out. 2023. Citado 2 vezes nas páginas 23 e 43.

MANSOUR, A. **7 Reasons Why Automation Is Good For Your Business**. 2019. Disponível em: <view-source:https://b5digital.dk/blog/7-reasons-why-automation-is-good-for-your-business/>. Acesso em: 10 out. 2023. Citado 2 vezes nas páginas 15 e 16.

MARTIN, R. C. **Clean Architecture - A Craftsman's Guide To Software Structure and Design**. [S.l.: s.n.], 2018. ISBN 978-0-13-449416-6. Citado 3 vezes nas páginas 20, 23 e 39.

MASSé, M. **REST API Design Rulebook**. [S.l.: s.n.], 2012. ISBN 978-1-449-31050-9. Citado na página 29.

MENEZES, P. B. **Linguagens Formais e Autômatos: Volume 3 da Série Livros Didáticos Informática UFRGS**. [S.l.]: Bookman Editora, 2009. ISBN 9788577807994. Citado 3 vezes nas páginas 6, 35 e 36.

MOORE, K.; GUPTA, D. **Finite State Machines**. 2023. Disponível em: <https://brilliant.org/wiki/finite-state-machines/>. Acesso em: 10 out. 2023. Citado na página 34.

PEYROTT, S. E. **The JWT Handbook**. [S.l.: s.n.], 2018. Citado na página 31.

REDINSKALA. **The SMTP Protocol Fundamentals**. 1st. ed. [S.l.: s.n.], 2013. Citado na página 33.

TANENBAUM, A. S. **Computer Networks**. 5th. ed. [S.l.: s.n.], 2011. ISBN 13: 978-0-13-212695-3. Citado 2 vezes nas páginas 6 e 32.

WAGNER, F. **Modeling software with finite state machines : a practical approach**. [S.l.]: Auerbach, 2006. ISBN 0849380863. Acesso em: 10 out. 2023. Citado 4 vezes nas páginas 6, 15, 34 e 35.

WIEGERS, K.; BEATTY, J. **Software Requirements**. [S.l.: s.n.], 2013. ISBN 978-0-7356-7966-5. Citado 3 vezes nas páginas 6, 19 e 20.

Apêndices

APÊNDICE A – Componentes das visões do Front-end

Este apêndice é dedicado a detalhar os componentes do *Front-end* para estender o detalhamento da camada. Na Figura 28, é apresentado os componentes detalhados a seguir.

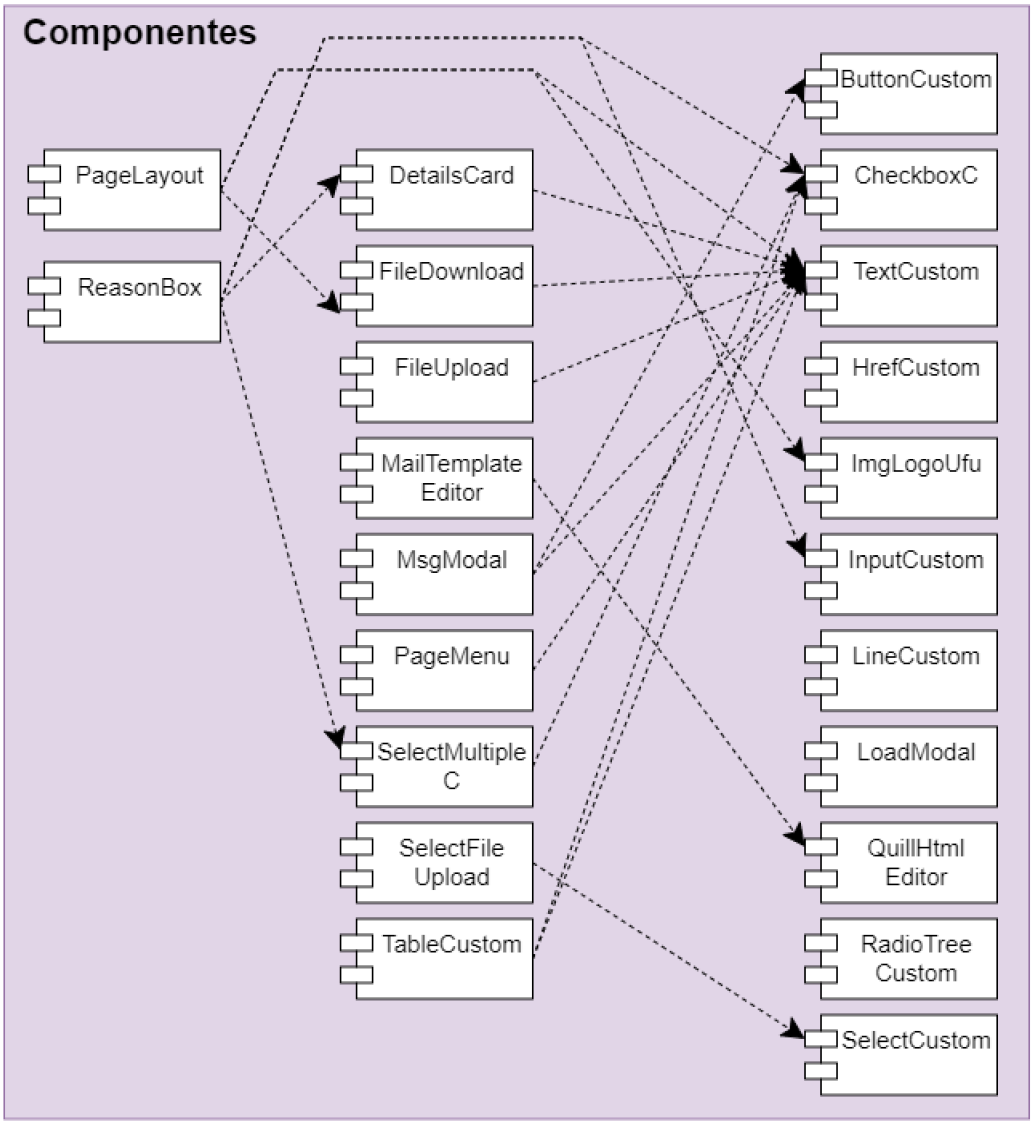


Figura 28 – Abstração dos componentes das visões do *Front-end*.

- ***ButtonCustom***: implementa o botão customizado utilizado em várias visões, incluindo a formatação automática das cores dado o nome atribuído a estas.

- ***CheckboxC***: codifica um checkbox próprio que permite a atribuição de CSS mais eficiente do que o checkbox implementado oficialmente pela linguagem HTML.
- ***DetailsCard***: apresenta uma caixa de detalhes, cada um em uma linha. Os detalhes são recebidos via uma lista de itens pelo componente que o chama.
- ***FileDownload***: recebe o link de download, nome e outros atributos, apresentando o nome e um botão que permite baixar um arquivo. Utilizado para baixar arquivos dos usuários salvos no servidor.
- ***FileUpload***: funciona similar ao componente anterior, exceto por realizar o envio do arquivo do usuário do computador local ao servidor.
- ***HrefCustom***: responsável por renderizar na tela um link de acesso customizável com as cores padrões definidas no sistema.
- ***ImgLogoUfu***: renderiza o logo da UFU em duas variações selecionáveis.
- ***InputCustom***: permite a entrada de dados do usuário em um input customizável que aceita diversas cores, máscaras, formatos, entre outras características, além de facilitar a definição de ações baseadas em eventos do *framework* Vue.js.
- ***LineCustom***: desenha uma linha simples na tela útil para demarcar mudanças de componentes nas visões.
- ***LoadModal***: é uma caixa suspensa que mostra o círculo de carregamento, utilizado em praticamente todas as requisições do sistema para indicar carregamento ao usuário.
- ***MailTemplateEditor***: modelo de e-mails utilizado como modelo para o envio de e-mails personalizáveis diretamente das interfaces do *Front-end*.
- ***MsgModal***: similar à caixa de carregamento, exceto por renderizar uma mensagem ao usuário, podendo apresentar botões de confirmação e rejeição.
- ***PageLayout***: componente utilizado por quase todas as visões, descreve o layout padrão das páginas de acesso autenticado, renderizando o menu, barra do topo e a estrutura envolvendo borda e fundo do conteúdo de cada página, facilitando o desenvolvimento de novas visões. Além disso, é responsável por alterar automaticamente o layout entre tamanhos de telas de dispositivos diferentes, permitindo a responsividade.
- ***PageMenu***: utilizada pelo componente de layout das visões para renderizar o menu da página. Possui as versões mobile e desktop.

- ***QuillHtmlEditor***: incorpora o editor de HTML Quill utilizado para editar as mensagens de e-mail dinâmicas. Utilizada apenas pela visão *CoordinatorAcception*.
- ***RadioTreeCustom***: cria uma árvore de botões de rádio ao receber uma lista que pode conter outras listas de itens.
- ***ReasonBox***: configura a interface que inclui a tabela de motivos utilizada pelo coordenador para a escolha de motivos de indeferimento do início de estágio dos alunos. A tabela é preenchida com dados do banco de dados e permite a seleção e leitura de seus itens por componentes externos.
- ***SelectCustom***: descreve uma caixa de seleção personalizada com as configurações do sistema que permite a seleção de um item. Assim como o *CheckboxC*, é um componente implementado para uma maior customização do que o disponível pela linguagem HTML.
- ***SelectFileUpload***: semelhante ao componente *FileUpload* exceto por utilizar o componente *SelectCustom* para a seleção de um rótulo para atribuir a um arquivo enviado ao servidor, permitindo um maior detalhamento deste.
- ***SelectMultipleC***: possui o funcionamento similar ao *SelectCustom* exceto por permitir a seleção de vários itens simultaneamente.
- ***TableCustom***: sendo um dos componentes mais complexos, implementa uma tabela que é estruturada de acordo com um objeto de descrição, incorporando os títulos, campos e tipos definidos no objeto. Assim como outros componentes, possui lógica de responsividade implementada para a representação em diversos formatos de tela.
- ***TextCustom***: recebe um texto e outras configurações como entrada e renderiza uma frase aplicando as configurações de cores e fontes globais definidas no arquivo de configuração.

APÊNDICE B – Tabelas do banco de dados

Neste capítulo, é apresentada cada uma das categorias definidas anteriormente para o banco de dados com uma maior profundidade em suas tabelas e campos. Na Figura 29, são posicionadas todas as categorias, destacando o nome de cada uma. Esta figura pode ser melhor visualizada no repositório que inclui os dados do *Back-end* e da base de dados. Em sequência, nas seções deste capítulo, são separadas cada uma das categorias destacando as suas figuras, seguidas pela descrição de suas tabelas, incluindo detalhes e motivações de uso, e dicionário de dados dos campos.

Figura 29 – Diagrama completo da base de dados.

B.1 Configurações

Sendo uma categoria não relacionada às outras, persistem, em suas tabelas, detalhes sobre configurações do sistema envolvendo diversos tipos. Todos herdam uma tabela

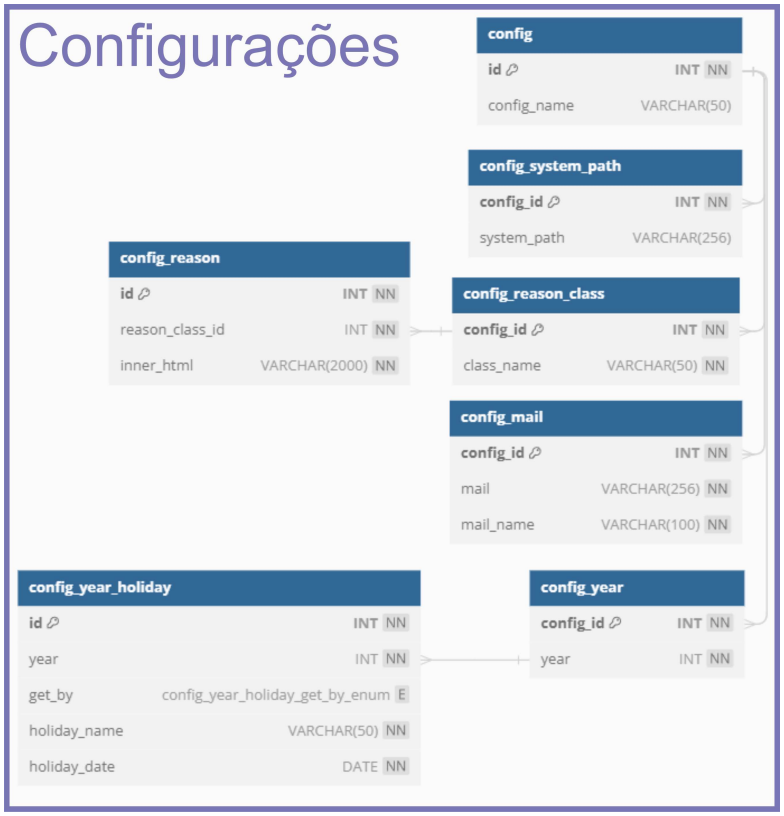


Figura 30 – Diagrama da categoria configurações.

de configuração base que pode ser utilizada para carregar todas as configurações de forma eficiente. A seguir, são listadas as tabelas:

- ***config***: permite a listagem e atribuição de campos padrões a todas as configurações que recebem seus campos como herança.
- ***config_system_path***: possui caminhos do sistema utilizados para armazenar arquivos, armazenar pares de chaves, entre outros.
- ***config_reason_class e config_reason***: juntas, as tabelas permitem a criação de uma classe de motivos seguido por suas instâncias. Motivos são códigos HTML anexados à tabela de motivos que o orientador utiliza para verificação da validade de documentos do aluno, estes podem ser classificados como, por exemplo, motivos do BCC, do BSI, entre outros.
- ***config_mail***: responsável por armazenar e-mails utilizados pelo sistema para reuso eficiente. Na versão desta monografia, é armazenado o e-mail da coordenação de estágios.
- ***config_year e config_year_holiday***: enquanto a primeira representa anos dos quais o sistema fica disponível, a segunda associa a cada um deles instâncias de feriados. Na versão deste trabalho, os feriados não são usados, porém como foram construídos, a sua lógica foi mantida para uso futuro.

Nome da Tabela	Nome da Coluna	Tipo de Dados	Restrições	Descrição
config	*id*	INT	PK, NOT NULL	Id da tabela
	config_name	VARCHAR(50)		Nome das configurações
config_system_path, config_reason_class, config_mail, config_year, config_year_holiday	*config_id*	INT	PK, FK, NOT NULL	Id da tabela, herdado da tabela configs
config_system_path	*system_path*	VARCHAR(256)		Caminho do sistema. Permite Linux e Windows
config_reason_class	*class_name*	VARCHAR(50)	NOT NULL	Nome da classe de motivos
config_reason	*id*	INT	NOT NULL	Id da tabela
	reason_class_id	INT	FK, NOT NULL	Id da classe que a instância é inserida
	ineer_html	VARCHAR(2000)	NOT NULL	HTML utilizado para renderizar o motivo
config_mail	*mail*	VARCHAR(256)	NOT NULL	Endereço de e-mail a ser salvo
	mail_name	VARCHAR(156)	NOT NULL	Nome do usuário do e-mail
config_year	*year*	INT	NOT NULL	Número que representa um ano para anexar feriados.
config_year_holiday	*year*	INT	FK, NOT NULL	Responsável pelo relacionamento entre anos e feriados
	id	INT	PK, NOT NULL	chave única da tabela
	get_by	ENUM		Armazena como é realizada a requisiçã de datas de feriado
	holiday_name		NOT NULL	Nome do feriado
	holiday_date		NOT NULL	Dia do feriado

Tabela 5 – Dicionário das tabelas da categoria configurações.

B.2 Dados do usuário

Figura 31 – Diagrama da categoria dados do usuário.

A categoria lista tabelas envolvendo os dados da conta do usuário, dados específicos de perfis, além de dados que servem como relação com outras categorias como a tabela de solicitações do usuário relacionada com a máquina de estados. É detalhada cada uma das tabelas a seguir:

- ***user_account***: possui os campos relativos ao usuário envolvendo seus e-mails, nome, gênero, telefone, senhas e data de criação. Um aspecto relevante é a presença do campo *password_salt* que armazena um código anexado à senha do usuário antes de ser codificada e persistida para uma maior segurança dos dados do usuário.

- ***user_has_profile***: atua juntamente com a tabela da categoria de perfis e com as tabelas de dados respectivas para a definição de múltiplos perfis para o usuário. A tabela armazena campos personalizáveis para a possibilidade de criação de perfis dinâmicos, além de incluir data de início e fim para a criação de perfis temporários.

- ***user_has_profile_coordinator_data*, *user_has_profile_advisor_data* e *user_has_profile_student_data***: são tabelas que atuam como descrito na tabela anterior para a criação de perfis de usuário. Como o fluxo básico da aplicação envolve sempre estes três perfis, elas foram definidas para manter a consistência do sistema evitando a definição dinâmica.

- ***user_has_attachment***: permite relacionar um usuário ao seu anexo presente na categoria respectiva para a possibilidade de evitar acessos não autorizados a usuários que não possuam o relacionamento ao anexo.

- ***user_has_solicitation* e *user_has_solicitation_state***: enquanto a primeira tabela é utilizada por alunos solicitantes que iniciam uma solicitação para a representação de um fluxo de solicitação da máquina de estados, a segunda representa os dados de um estado da máquina específica.

  A tabela de solicitação do usuário possui o campo anexado, que indica o orientador do fluxo o qual é preenchido quando o docente aceita o aluno no fluxo, alterando o valor do campo de aceitação pelo orientador para verdadeiro. Além disso, a tabela mantém o estado atual e os dados do fluxo, incluindo o caminho para anexos do usuário, entradas e outras informações relevantes do fluxo que são preenchidas e reutilizadas à medida que o fluxo prossegue.

  Já a segunda tabela é relacionada com o estado da máquina de estado e persistem informações indicando se a decisão é o deferimento, envio, cancelamento, entre outras, além de datas de término e início e a motivação que leva a decisão do estado.

Nome da Tabela	Nome da Coluna	Tipo de Dados	Restrições	Descrição
user_account	*id*	INT	PK, NOT NULL	Id da tabela
	institutional_email	VARCHAR(256)	NOT NULL	E-mail institucional do usuário
	secondary_email	VARCHAR(256)		E-mail secundário do usuário. Pode ser utilizado para o envio em paralelo com o institucional
	user_name	VARCHAR(100)	NOT NULL	Nome do usuário
	gender	ENUM	NOT NULL	Gênero do usuário
	phone	VARCHAR(15)		Telefone do usuário
	pasword_hash	CHAR(65)		Hash da senha do usuário incluindo senha e salt. A senha não é armazenada diretamente para a segurança do usuário
	password_salt	CHAR(16)		Caracteres aleatórios para a salga de senha do usuário por segurança
	creation_datetime	DATETIME		Data e hora de criação do usuário
user_has_profile, user_has_solicitation, user_has_attachment	*user_id*	INT	FK, NOT NULL	Id do usuário usado como chave estrangeira para estabelecer as relações das tabelas
user_has_profile	*id*	INT	PK, NOT NULL	Id da tabela
	profile_id	INT	FK, NOT NULL	Chave estrangeira da tabela de perfis
	user_dynamic_profile _fields_data	JSON		JSON que indica os campos variáveis de um perfil dinâmico do usuário. Este pode ser comparado com os meta-dados do perfil dinâmico para validação
	start_datetime	DATETIME	NOT NULL	Data e hora de início da atribuição do perfil ao usuário
	end_datetime	DATETIME		Data e hora que é finalizado os acessos do perfil do usuário
user_has_solicitation	*id*	INT	PK, NOT NULL	Id da tabela
	advisor_siape	VARCHAR(15)		Código Siape do orientador do aluno. Preenchido na solicitação de início de estágio
	is_accepted_by _advisor	BOOL		Se o aluno foi aceito pelo orientador
	solicitation_id	INT	FK, NOT NULL	Id da solicitação do usuário
	actual_solicitation _state_id	INT	FK, NOT NULL	Id do estado atual da solicitação
	solicitation_user_data	JSON		Dados gerais do usuário da solicitação
user_has_attachment	*id*	INT	PK, NOT NULL	Id da tabela
	attachment_id	INT	FK, NOT NULL	Id do anexo que o usuário possui acesso
user_has_profile _student_data, user_has_profile _coordinator_data, user_has_profile _advisor_data	*user_has_profile_id*	INT	PK, FK, NOT NULL	Chave estrangeira que relaciona os perfis que o usuário possui com os seus dados de usuário
user_has_profile _student_data	*matricula*	VARCHAR(15)	NOT NULL	Matricula do aluno
	course	ENUM	NOT NULL	Enumerador que permite escolha entre BCC e BSI
user_has_profile _coordinator_data, user_has_profile _advisor_data	*siape*	VARCHAR(15)	NOT NULL	Siape do perfil respectivo. Coordenadores também possuem Siape
user_has_solicitation _state	*id*	INT	PK, NOT NULL	Id da tabela
	user_has_solicitation _id	INT	FK, NOT NULL	Chave estrangeira dos dados do usuário da solicitação
	solicitation_state_id	INT	FK, NOT NULL	Chave estrangeira com o estado que possui os dados desta tabela
	decision	ENUM	NOT NULL	Decisão final no estado do usuário
	reason	VARCHAR(100)		Motivo final no estado do usuário
	start_datetime	DATETIME	NOT NULL	Inicio do estado para o usuário
	end_datetime	DATETIME		Fim do estado para o usuário

Tabela 6 – Dicionário das tabelas da categoria dados do usuário.

B.3 Validação de Cadastros

Figura 32 – Diagrama da categoria validação de cadastros.

A categoria envolve apenas a tabela de validação de e-mails e esta é utilizada no fluxo de cadastro para a persistência de códigos de cadastros. Estes, por sua vez, são encaminhados ao e-mail do usuário para a sua confirmação. Além disso, é configurado o campo de expiração do token automaticamente preenchido para oito horas após a criação por um gatilho do banco de dados para invalidar os tokens.

Nome da Tabela	Nome da Coluna	Tipo de Dados	Restrições	Descrição
mail_validation	*institutional_email*	VARCHAR(256)	PK, FK, NOT NULL	E-mail institucional do usuário. Permite a associação do código de validação ao usuário
	validation_code	CHAR(10)		Código de validação de cadastro
	validation_code _expires_in	TIMESTAMP		Intervalo de tempo que o código será expirado a partir da criação

Tabela 7 – Dicionário das tabelas da categoria validação de cadastros.

B.4 Anexos

Figura 33 – Diagrama da categoria anexos.

Esta inclui a tabela de anexos que atua juntamente com a tabela de anexos do usuário para a persistência de anexos no sistema. O campo *hash_name* armazena o nome do arquivo que é gerado, utilizando uma codificação hash, que é persistente em um diretório do sistema, indicado por uma configuração de caminhos apresentada na seção de configurações.

Nome da Tabela	Nome da Coluna	Tipo de Dados	Restrições	Descrição
attachment	*id*	INT	PK, NOT NULL	Id da tabela
	hash_name	VARCHAR(150)	NOT NULL	Hash do nome do anexo. Possui em seu corpo o nome do anexo, possibilitando a sua localização

Tabela 8 – Dicionário das tabelas da categoria anexos.

B.5 Perfis

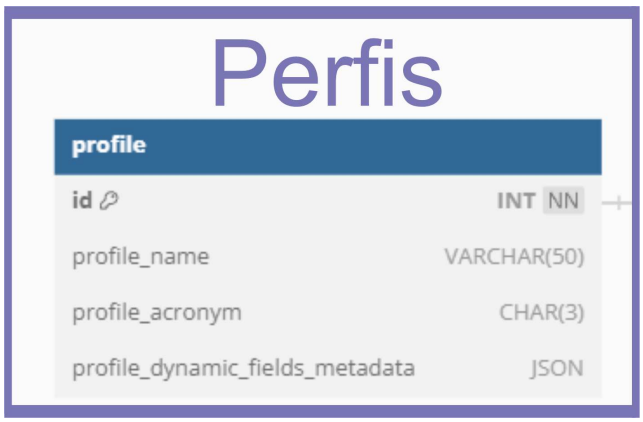


Figura 34 – Diagrama da categoria perfis.

A tabela de perfis opera em conjunto com as tabelas de perfis de usuário para sua definição. Os campos de nome e acrônimo são configurados para identificar os perfis-base, permitindo também a criação de perfis adicionais. Além disso, há um campo de metadados que pode ser utilizado para restringir os campos personalizados na tabela de perfis do usuário. Essa restrição pode ser aplicada ao realizar a comparação entre os campos de metadados e os dados dinâmicos do usuário quando o perfil é incluído. É importante ressaltar que, embora foram definidos os campos, a comparação não foi aplicada por não serem definidos perfis dinâmicos nesta versão.

Nome da Tabela	Nome da Coluna	Tipo de Dados	Restrições	Descrição
profile	*id*	INT	PK, NOT NULL	Id da tabela
	profile_name	VARCHAR(50)		Nome do perfil
	profile_acronym	CHAR(3)		Acrônimo do perfil, composto por 3 letras
	profile_dynamic _fields_metadata	JSON		Campos dinâmicos permitidos no perfil. Pode ser utilizado para validar os campos dinâmicos do usuário na tabela user_has_profile

Tabela 9 – Dicionário das tabelas da categoria perfis.

B.6 Solicitações (Máquina de estados)

Figura 35 – Diagrama da categoria solicitações.

Categoria responsável, juntamente com a de transições, por persistir a lógica dos fluxos da máquina de estados. Enquanto as categorias definem o fluxo que os usuários seguem, envolvendo a definição do fluxo, os estados e transições, as tabelas de solicitações do usuário armazenam os dados internos do fluxo. Em outras palavras, enquanto a definição da estrutura da máquina é abstraída pelas categorias de solicitações e transições, os dados específicos de uma instância de execução são persistidos na categoria dos usuários. A seguir, são descritas cada uma das tabelas da categoria.

- **solicitation**: tabela que define um fluxo, permite a relação de uma solicitação ter vários estados além da possibilidade da atribuição do nome.
- **solicitation_state**: abstrai os estados da solicitação, possui campos de descrição, de duração, da página dinâmica, caso esta seja atribuída ao estado, e se é inicial e período máximo de duração da solicitação no estado antes da mesma ser expirada.
- **solicitation_start_mail**: atua juntamente às mensagens de e-mail dinâmicas para permitir o envio de um e-mail ao realizar o início de uma solicitação sem a necessidade de realizar uma transição.
- **solicitation_state_profile_editors**: permite definir a lista de perfis que podem editar o estado de uma solicitação, possibilitando a restrição da edição do estado a perfis não autorizados.

Nome da Tabela	Nome da Coluna	Tipo de Dados	Restrições	Descrição
solicitation	*id*	INT	PK, NOT NULL	Id da tabela
	solicitation_name	VARCHAR(256)		Nome da solicitação
solicitation_state, solicitation_start_mail	*solicitation_id*	INT	FK, NOT NULL	Id da solicitação, herdado da tabela solicitations
solicitation_state	*id*	INT	PK, NOT NULL	Id da tabela
	state_description	VARCHAR(256)		Descrição do estado da solicitação, útil para administradores
	state_max_duration _days	INT		Dias máximos de duração da solicitação no estado, a qual é cancelada após os dias
	state_dynamic _page_id	INT	FK	Id da tabela de páginas dinâmicas relacionada ao estado
	state_static _page_name	INT	FK	Nome da página estática relacionada ao estado. O nome deve refletir a Visão no Front-end
	is_initial_state	BOOL	NOT NULL	Indica se é o estado inicial da solicitação
solicitation_start_mail	*id*	INT	NOT NULL	Id da tabela
	dynamic_mail_id	INT	NOT NULL	Id de um e-mail da tabela de e-mails dinâmicos que é enviado ao iniciar a solicitação
solicitation_state _profile_editors	*id*	INT	NOT NULL	Id da tabela
	solicitation_state_id	INT	NOT NULL	Id do estado da solicitação associado ao perfil editor da tabela
	state_profile_editor	INT	NOT NULL	Id do perfil editor da tabela de perfis

Tabela 10 – Dicionário das tabelas da categoria solicitações.

B.7 E-mails Dinâmicos

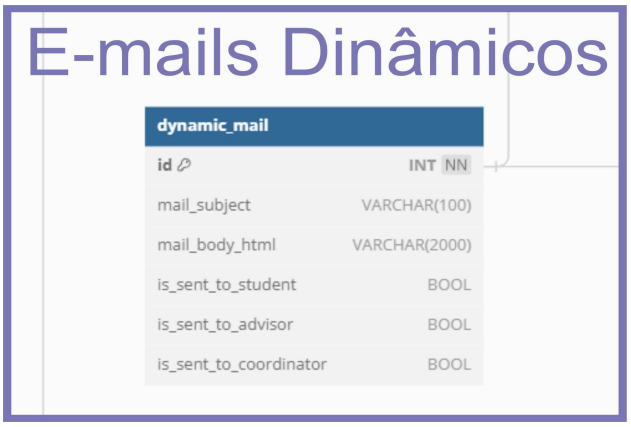


Figura 36 – Diagrama da categoria e-mails dinâmicos.

Inclui a tabela que descreve mensagens de e-mail atribuíveis ao início de solicitações, transições de estados e configurações do sistema para o envio dessas mensagens. Inclui os campos de assunto, corpo em HTML e os perfis de envio, que são utilizados em conjunto para a criação de um formato MIME, incluindo o HTML, que é enviado a cada um dos perfis descritos.

Nome da Tabela	Nome da Coluna	Tipo de Dados	Restrições	Descrição
dynamic__mail	*id*	INT	PK, NOT NULL	Id da tabela
	mail__subject	VARCHAR(100)		Assunto da mensagem
	mail__body__html	VARCHAR(2000)		Corpo em HTML da mensagem
	is__sent__to__student	BOOL		Indica se a mensagem é enviada a estudantes
	is__sent__to__advisor	BOOL		Indica se a mensagem é enviada a orientadores
	is__sent__to__coordinator	BOOL		Indica se a mensagem é enviada a coordenadores

Tabela 11 – Dicionário das tabelas da categoria e-mails dinâmicos.

B.8 Transições (Máquina de estados)

Figura 37 – Diagrama da categoria transições.

Possui tabelas que operam em conjunto com as tabelas da categoria de solicitações para estabelecer o fluxo geral da máquina de estados. Enquanto a outra categoria descreve o fluxo e seus estados, esta se concentra nas transições dos estados, definindo diversos tipos de transições, como as agendadas, as de envio de e-mails, as realizadas por páginas dinâmicas e as manuais. A seguir, estão listadas as tabelas correspondentes a essa categoria.

- ***solicitation__state__transition***: tabela responsável por definir os campos herdados por todos os tipos de transições, incluindo a indicação do estado de origem e destino, juntamente com o nome da transição.
- ***solicitation__state__transition__manual***: transições realizadas manualmente via interações com usuários durante o fluxo da aplicação em páginas estáticas. São ne-

cessárias em fluxos que envolvem configurações adicionais além dos gerais implementados na solicitação.

- ***solicitation__state__transition__scheduled***: atua juntamente com a categoria de agendamento de eventos para a definição de uma transição baseada no tempo. Enquanto esta tabela descreve que deve ocorrer o agendamento, a outra categoria armazena as instâncias dos eventos.
- ***solicitation__state__transition__mail***: transição que resulta no envio de um e-mail. Pode ser utilizada para o envio de e-mails de lembrete e detalhamento das etapas do fluxo.
- ***solicitation__state__transition__from__dynamic__page***: atua juntamente à categoria de páginas dinâmicas por meio de uma relação entre as tabelas para a definição de transições ao clicar nos botões dinâmicos criados pela página.

Nome da Tabela	Nome da Coluna	Tipo de Dados	Restrições	Descrição
solicitation__state__transition	*id*	INT	PK, NOT NULL	Id da tabela
	solicitation_state_id__from	INT	FK, NOT NULL	Id do estado de origem da transição
	solicitation_state_id__to	INT	FK	Id do estado de destino ao aplicar a transição
	transition_name	VARCHAR(30)	NOT NULL	Nome da transição, útil para encontrar uma transição
solicitation__state__transition__manual, solicitation__state__transition__scheduled, solicitation__state__transition__mail, solicitation__state__transition__from__dynamic__page	*solicitation_state__transition_id*	INT	PK, FK, NOT NULL	Id da transição base, permite a organização de tipos diferentes de transições com base em uma super classe
	transition_decision	ENUM	NOT NULL	Enumerador que indica a decisão que é tomada ao aplicar a transição na solicitação do usuário. É visualizado pelo usuário
	transition_reason	VARCHAR(100)	NOT NULL	Frase personalizável que descreve o motivo de uma transição ter ocorrido e é visualizada por usuários
solicitation__state__transition__scheduled	*transition_delay__seconds*	INT	NOT NULL	Indica a quantidade de segundos que a transição será executada após ser agendada
solicitation__state__transition__mail	*id*	INT	PK, NOT NULL	Id da tabela. Não utiliza o próprio id da transição para permitir o envio de múltiplos e-mails em uma transição
	dynamic_mail_id	INT	FK, NOT NULL	Id do e-mail a ser enviado
solicitation__state__transition__from__dynamic__page	*dynamic_page__component*	ENUM	NOT NULL	Enumerador que indica qual o componente em uma página dinâmica aplica a transição

Tabela 12 – Dicionário das tabelas da categoria transições.

B.9 Agendador de Eventos

É configurada com duas tabelas que permitem a persistência de instâncias de eventos agendados para fluxos de solicitação de usuários, útil para a recuperação destas

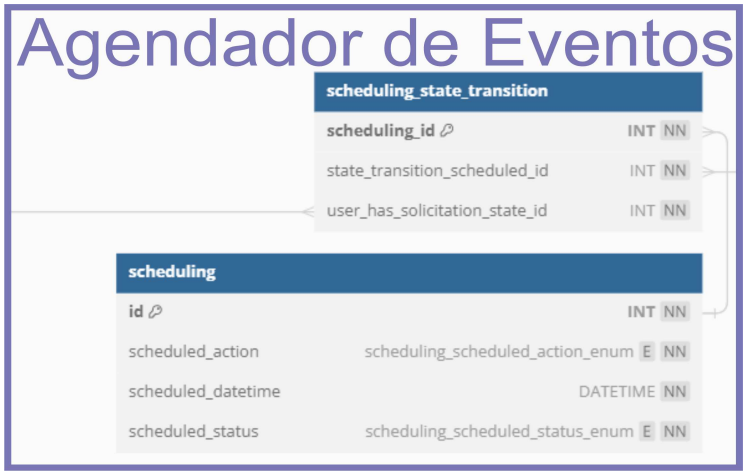


Figura 38 – Diagrama da categoria agendador de eventos.

em caso de falhas no sistema. São utilizadas ao iniciar o carregamento do sistema para preencher o agendador com as pendentes e são persistidas ou alteradas quando um estado que as possuem é, respectivamente, iniciado ou alterado. Enquanto a tabela de *scheduling* possui a ação, dados e status que permitem a execução do evento em específico, a tabela *scheduling_state_transition* é responsável por aplicar os relacionamentos que interligam o usuário ao evento e o evento à transição agendada.

Nome da Tabela	Nome da Coluna	Tipo de Dados	Restrições	Descrição
scheduling	*id*	INT	PK, NOT NULL	Id da tabela
	scheduled_action	ENUM	NOT NULL	Enumerador de ações do evento, como: transitar em estados ou enviar mensagens
	scheduled_datetime	DATETIME	NOT NULL	Data e hora que o evento é emitido
	scheduled_status	ENUM	NOT NULL	Status do evento, indica se foi enviado, se foi cancelado ou se está pendente
scheduling_state_transition	*scheduling_id*	INT	PK, FK, NOT NULL	Id recebido da tabela de agendamentos, necessário, pois, enquanto a anterior possui os dados do evento, a atual atua com as restrições
	state_transition_scheduled_id	INT	FK, NOT NULL	Id da transição que aplica o agendamento de eventos ao ocorrer a transição de estados
	user_has_solicitation_state_id	INT	FK, NOT NULL	Id do estado da solicitação do usuário. Permite a utilização dos dados da solicitação de forma assíncrona.

Tabela 13 – Dicionário das tabelas da categoria agendador de eventos.

B.10 Página Dinâmica

Sendo a categoria com o maior número de tabelas, esta permite a construção de páginas dinâmicas através da sua montagem ao posicionar componentes configurados em suas tabelas. Inicialmente, é definida a página dinâmica relacionada a um estado da solicitação, e depois, são anexados componentes por meio de relacionamentos nas tabelas

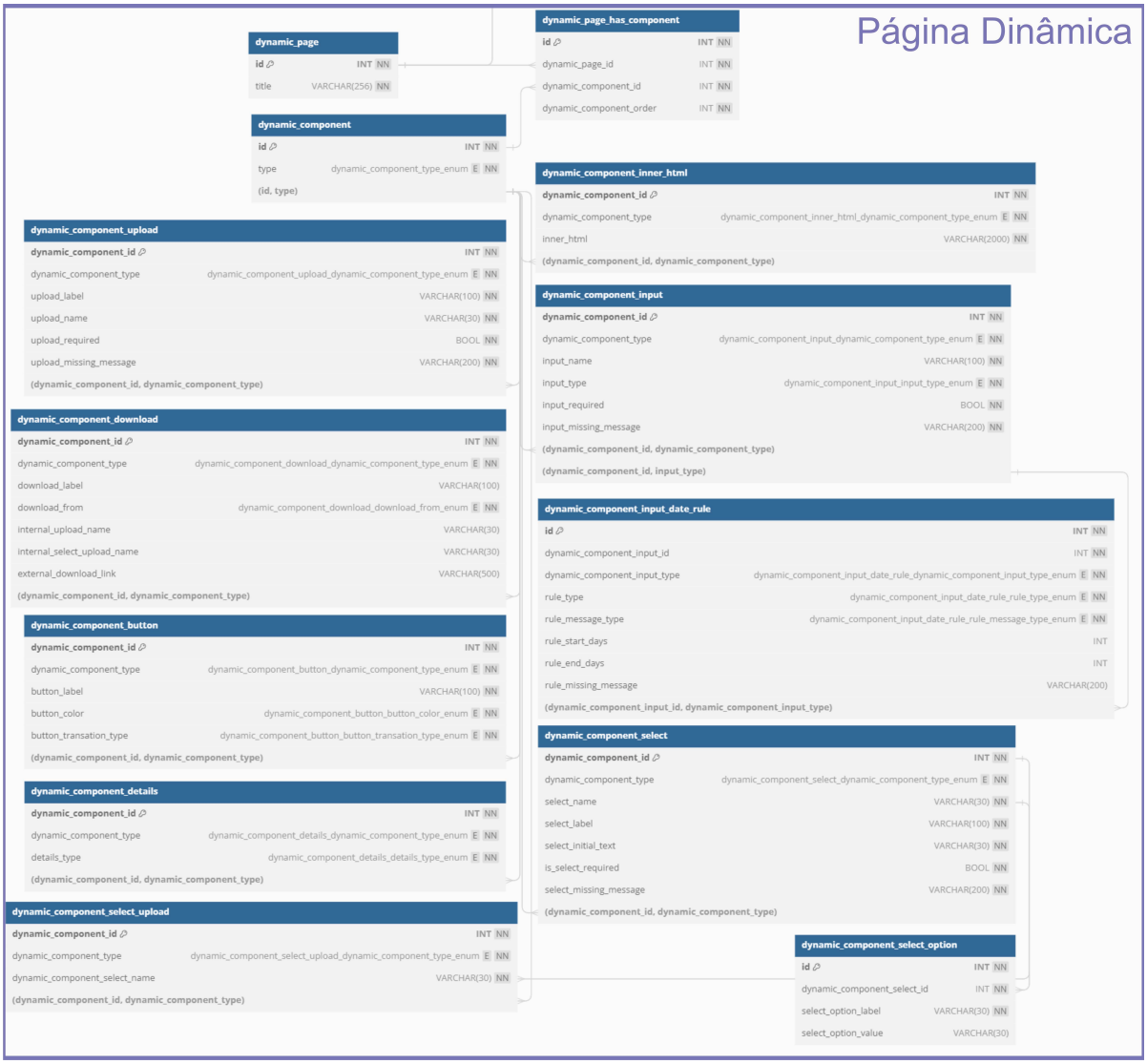


Figura 39 – Diagrama da categoria página dinâmica.

e transições ao relacionar objetos com as transições, permitindo que a página dinâmica faça parte do fluxo da máquina de estados. A seguir, são listadas as suas tabelas:

- ***dynamic_page***: representa a página dinâmica incluindo seu título e se relaciona com as tabelas *dynamic_component* e *dynamic_page_has_component* para definir a sua estrutura envolvendo os componentes.
- ***dynamic_page_has_component***: atua como a intermediadora entre a página e os componentes, permitindo que a página possua uma lista de componente além de definir a ordem destes.
- ***dynamic_component***: tabela herdada por todos os componentes permitindo a associação do tipo do componente, importante para a renderização adequada do lado do *Front-end*.

- ***dynamic_component_inner_html***: permite a definição de uma estrutura em HTML que pode ser injetada na página, permitindo a adição de parágrafos estilizados com CSS na página dinâmica.

- ***dynamic_component_upload***: possui os campos necessários para a definição de um componente de envio de arquivos pelo usuário, como o rótulo, o nome resultante, se este é requerido e uma mensagem que será renderizada caso este não seja anexado e for requerido.

- ***dynamic_component_download***: similar ao anterior, exceto por ser utilizado para baixar arquivos de uma fonte específica indicada pelo campo enumerado textitdownload_from com o endereço de acesso listado nos demais três campos: *internal_upload_name* que indica o download de um envio realizado anteriormente por um componente de envio, *internal_select_upload_name* que indica o download de um envio realizado anteriormente por um componente de seleção e envio ou *external_download_link* que permite baixar arquivos dado uma URL específica.

- ***dynamic_component_input*** e ***dynamic_component_input_date_rule***: enquanto o primeiro possui campos para renderizar um input na página incluindo o seu nome, tipo, se é obrigatório e a mensagem de renderização caso este seja obrigatório e não tenha sido enviado, o segundo é restringido pela relação do banco de dados ao tipo data pelo campo *input_type* e permite a definição de restrição de intervalos de datas ao usuário.

- ***dynamic_component_select*** e ***dynamic_component_select_option***: são responsáveis por criar uma caixa de seleção com opções. A primeira tabela define atributos da caixa, incluindo nome, rótulo, texto inicial, se é obrigatório e a mensagem renderizada no caso de falta. Já a segunda configura as opções da caixa, incluindo os rótulos e valores para a seleção do usuário.

- ***dynamic_component_button***: configura os botões da página. Todos os botões possuem rótulos, cor e o tipo de transição da qual é comparada às transições do estado para ser aplicada ao botão ser clicado. A cor é relevante para diferenciar diferentes resultados de ações ao se clicar no botão.

- ***dynamic_component_details***: permite a renderização de uma caixa de detalhes que pode incluir dados de usuários para a verificação por outros no fluxo, o campo com tipo de detalhes indica quais valores serão renderizados no *Front-end*.

- ***dynamic_component_select_upload***: junta as funcionalidades do componente de caixa de seleção e do de envio para possibilitar a caracterização do arquivo a ser envio pelas opções da caixa de seleção.

Nome da Tabela	Nome da Coluna	Tipo de Dados	Restrições	Descrição
dynamic_page	*id*	INT	PK, NOT NULL	Id da tabela
	title	VARCHAR(256)	NOT NULL	Título que aparece no topo da página e descreve seu conteúdo
dynamic_component	*id*	INT	PK, NOT NULL	Id da tabela
	type	ENUM	NOT NULL	Enumerado que indica o tipo do componente como: input, upload, download, entre outros.
dynamic_page_has_component	*id*	INT	PK, NOT NULL	Id da tabela
	dynamic_page_id	INT	FK, NOT NULL	Id da página dinâmica que possui o componente associado
	dynamic_component_id	INT	FK, NOT NULL	Id do componente que a página dinâmica possui associado
	dynamic_component_order	INT	NOT NULL	Ordem do componente, iniciando em 1
dynamic_component_inner_html, dynamic_component_upload, dynamic_component_input, dynamic_component_download, dynamic_component_button, dynamic_component_select, dynamic_component_details, dynamic_component_select_upload	*dynamic_component_id*	INT	PK, FK, NOT NULL	Chave estrangeira que faz a relação da tabela com dados específicos do componente com a sua superclasse
	dynamic_component_type	INT	FK, NOT NULL	Juntamente ao campo anterior, estabelece a restrição das instâncias da tabela serem componentes do tipo correto com a restrição da superclasse pelo id e tipo
dynamic_component_inner_html	*inner_html*	VARCHAR(2000)	NOT NULL	Código HTML renderizado no navegador
dynamic_component_upload	*upload_label*	VARCHAR(100)	NOT NULL	Rótulo que aparece ao lado da seleção de arquivo para upload
	upload_name	VARCHAR(30)	NOT NULL	Nome associado ao anexo realizado. Deve ser associado ao download para a realização do download deste anexo
	upload_required	BOOL	NOT NULL	Se o envio é obrigatório na página
	upload_missing_message	VARCHAR(200)	NOT NULL	Mensagem caso o envio não tenha sido realizado
dynamic_component_input	*input_name*	VARCHAR(100)	NOT NULL	Nome associado ao input realizado
	input_type	ENUM	NOT NULL	Enumerador que indica o tipo do input, como: date, text, entre outros
	input_required	BOOL	NOT NULL	Se o envio é obrigatório na página
	input_missing_message	VARCHAR(200)	NOT NULL	Mensagem caso o envio não tenha sido realizado
dynamic_component_button	*button_label*	VARCHAR(100)	NOT NULL	Rótulo que aparece no botão
	button_color	ENUM	NOT NULL	Enumerador que permite a seleção de cores do botão
	button_transation_type	ENUM	NOT NULL	Enumerador que indica o tipo de transição a ser realizada ao clicar no botão. Funciona juntamente ao tipo de transição de páginas dinâmicas para transicionar os estados
dynamic_component_select	*select_name*	VARCHAR(30)	NOT NULL	Nome associado a seleção realizada
	select_label	VARCHAR(100)	NOT NULL	Rótulo que aparece ao lado da seleção
	select_initial_text	VARCHAR(30)	NOT NULL	Texto inicial que aparece antes da seleção
	is_select_required	BOOL	NOT NULL	Se a seleção é obrigatória na página
	select_missing_message	VARCHAR(200)	NOT NULL	Mensagem caso a seleção não tenha sido realizada
dynamic_component_details	*details_type*	ENUM	NOT NULL	Enumerador que indica o perfil que possui os detalhes exibidos na página
dynamic_component_select_upload	*dynamic_component_select_name*	VARCHAR(30)	NOT NULL	Nome do componente de seleção utilizado antes do envio de um anexo

Tabela 14 – Primeira parte do dicionário das tabelas da categoria página dinâmica.

Nome da Tabela	**Nome da Coluna**	**Tipo de Dados**	**Restrições**	**Descrição**
dynamic_component_input_date_rule	*id*	INT	PK, NOT NULL	Id da tabela. Não usa o próprio id do input para possibilitar a associação de múltiplas regras
	dynamic_component_input_id	INT	FK, NOT NULL	Id do input que a restrição é aplicada
	dynamic_component_input_type	ENUM	FK, NOT NULL	Chave estrangeira recebida do input que permite a verificação se o tipo do input é de datas
	rule_type	ENUM	NOT NULL	Tipo da regra: restringir, permitir, entre outros
	rule_message_type	ENUM	NOT NULL	Tipo de mensagem da regra: aviso ou erro.
	rule_start_days	INT		Inicio do intervalo da regra em dias
	rule_end_days	INT		Término do intervalo da regra em dias
	rule_missing_message	VARCHAR(200)		Mensagem caso a regra não tenha sido satisfeita
dynamic_component_select_option	*id*	INT	PK, NOT NULL	Id da tabela. Não usa o próprio id do select para possibilitar a associação de múltiplas opções
	dynamic_component_select_id	INT	FK, NOT NULL	Chave estrangeira que relaciona o select às suas opções
	select_option_label	VARCHAR(30)	NOT NULL	Rótulo que é visível da opção
	select_option_value	VARCHAR(30)		Valor da opção que é utilizado ao armazenar

Tabela 15 – Segunda parte do dicionário das tabelas da categoria página dinâmica.

APÊNDICE C – Telas do fluxo de início de estágio

Neste apêndice, são listadas todas as telas do fluxo de início de estágios apresentadas na Subseção 3.3.4, além da tabela de solicitações do aluno resultante após o procedimento.

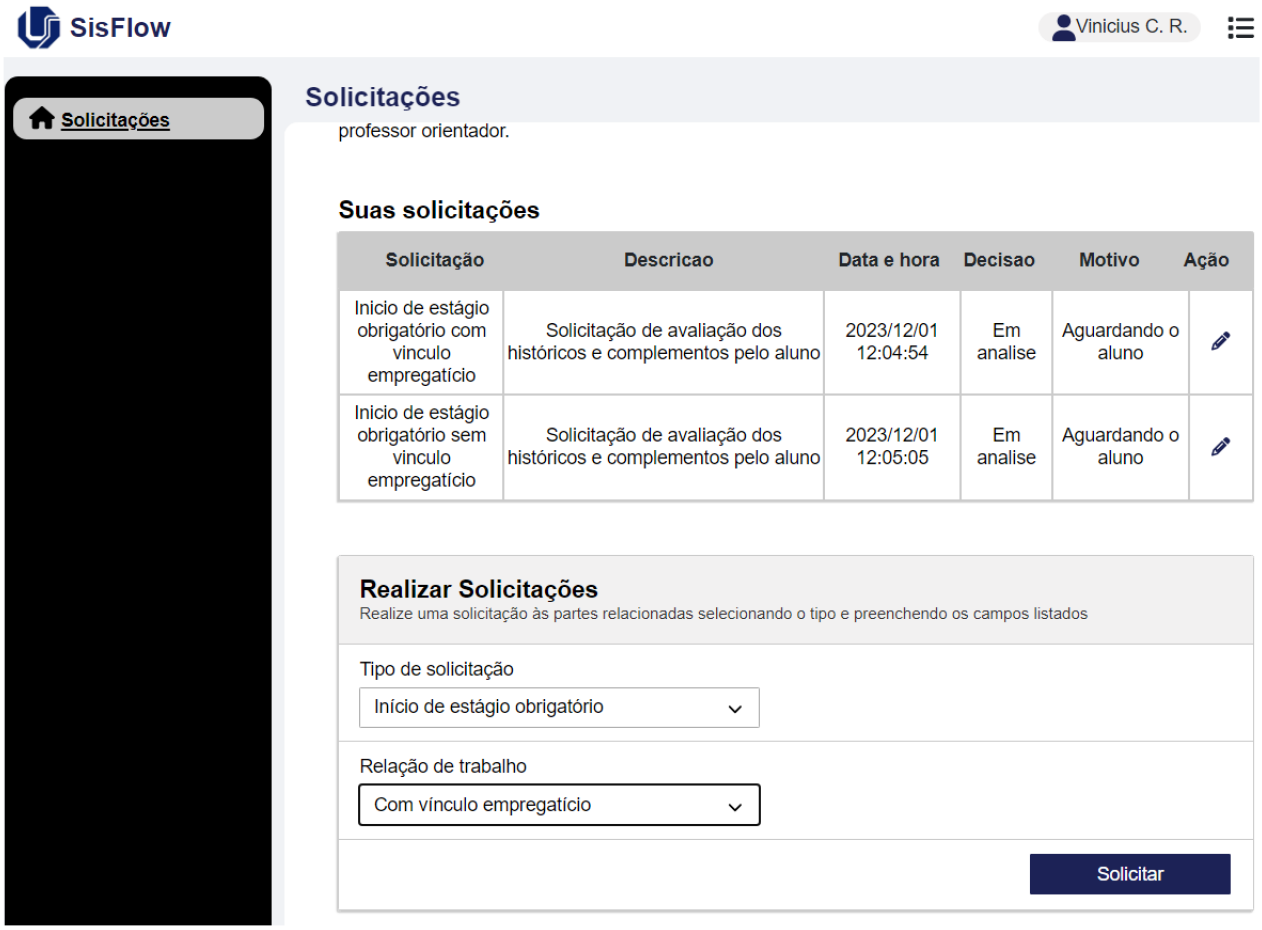


Figura 40 – Tela *Home* com a solicitação de início de estágio pelo aluno (estado S0).

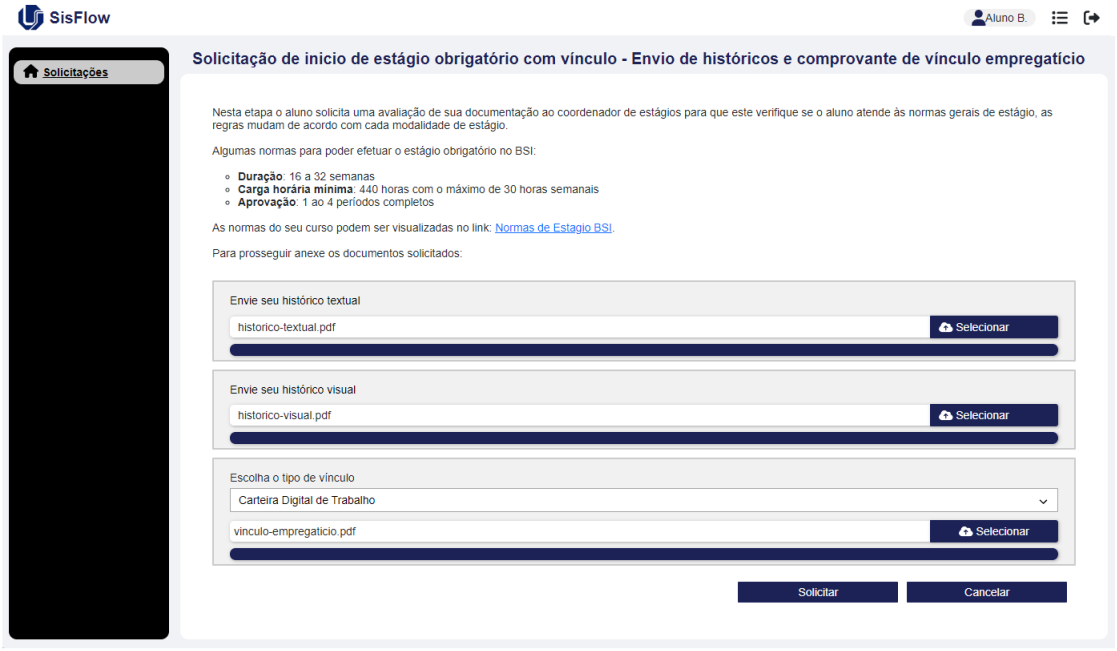


Figura 41 – Tela *DynamicSolicitation* com o envio de históricos e complementos pelo aluno (estado S1).

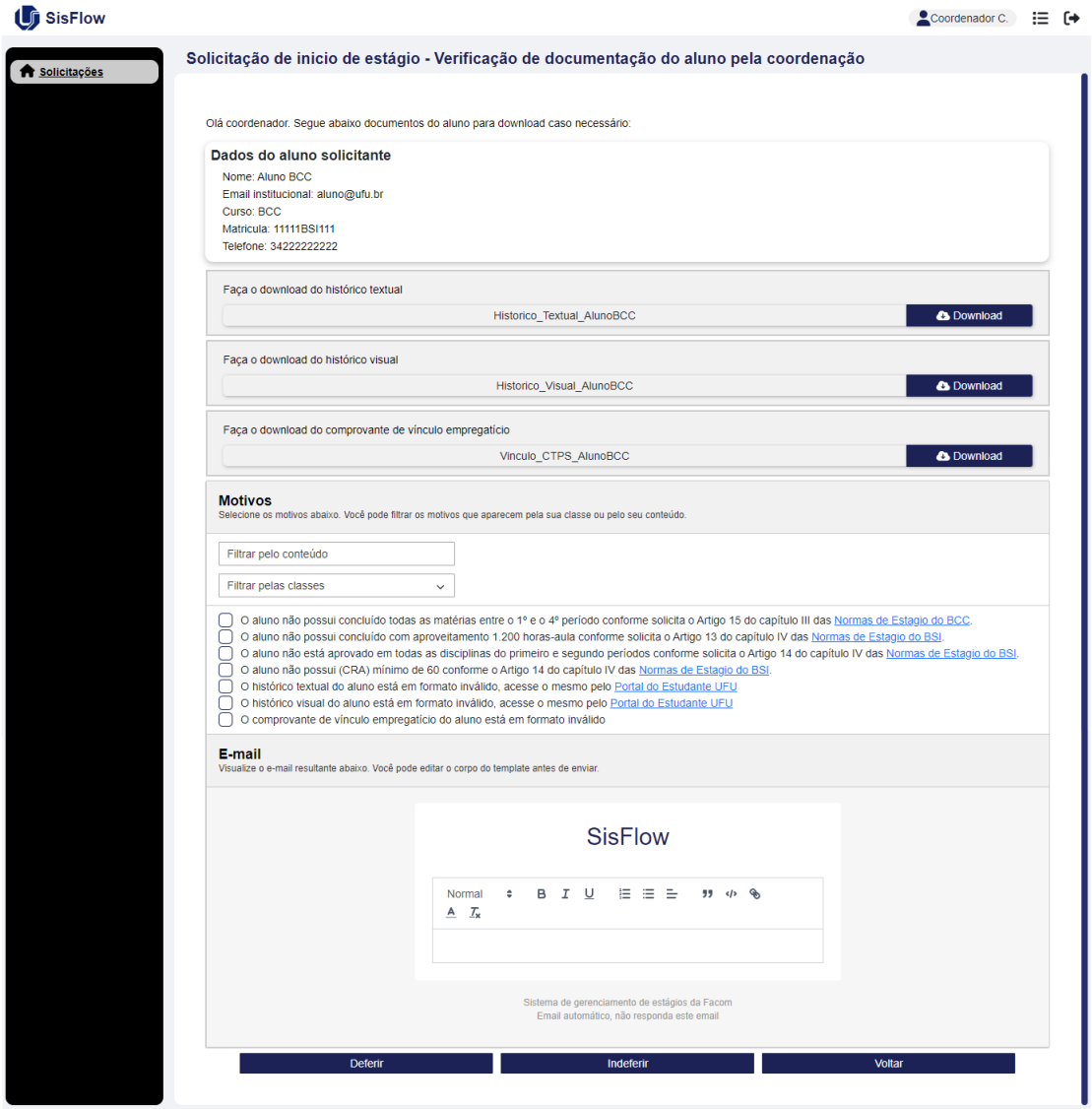


Figura 42 – Tela *CoordinatorAcception* com a avaliação dos históricos e complementos pelo coordenador (estado S2).

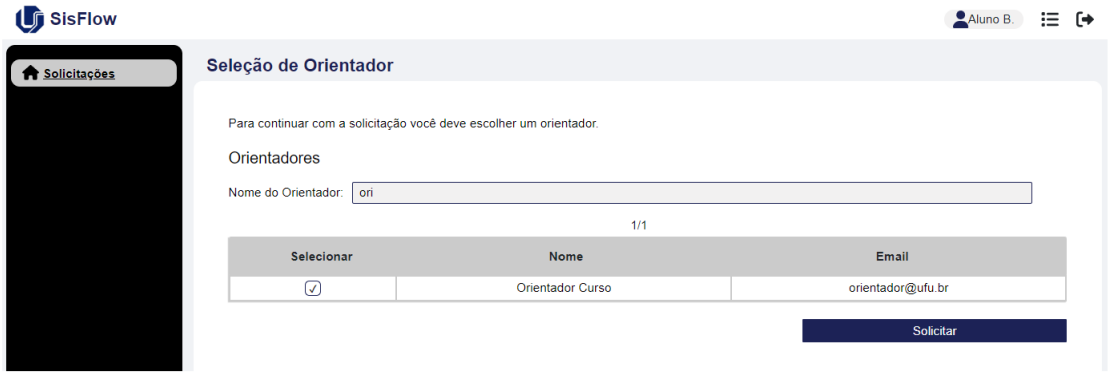


Figura 43 – Tela *AdvisorSelection* com a escolha de orientador pelo aluno (estado S3).

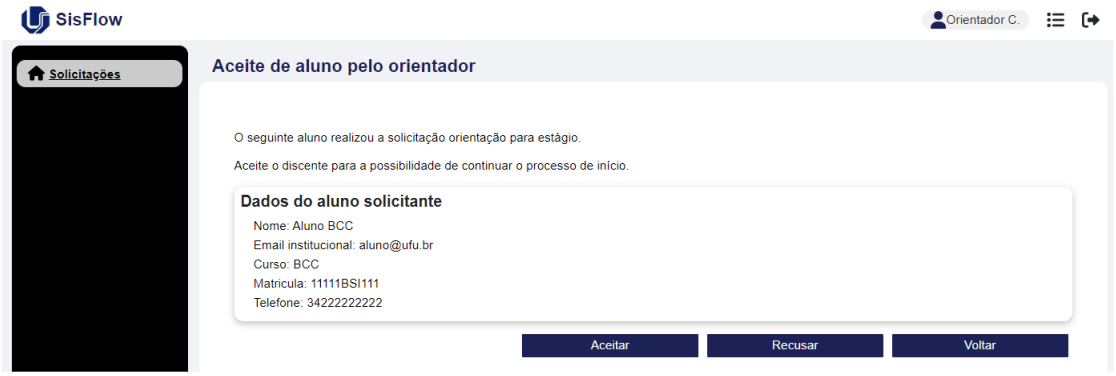


Figura 44 – Tela *AdvisorAcception* com o aceite de orientado pelo orientador (estado S4).

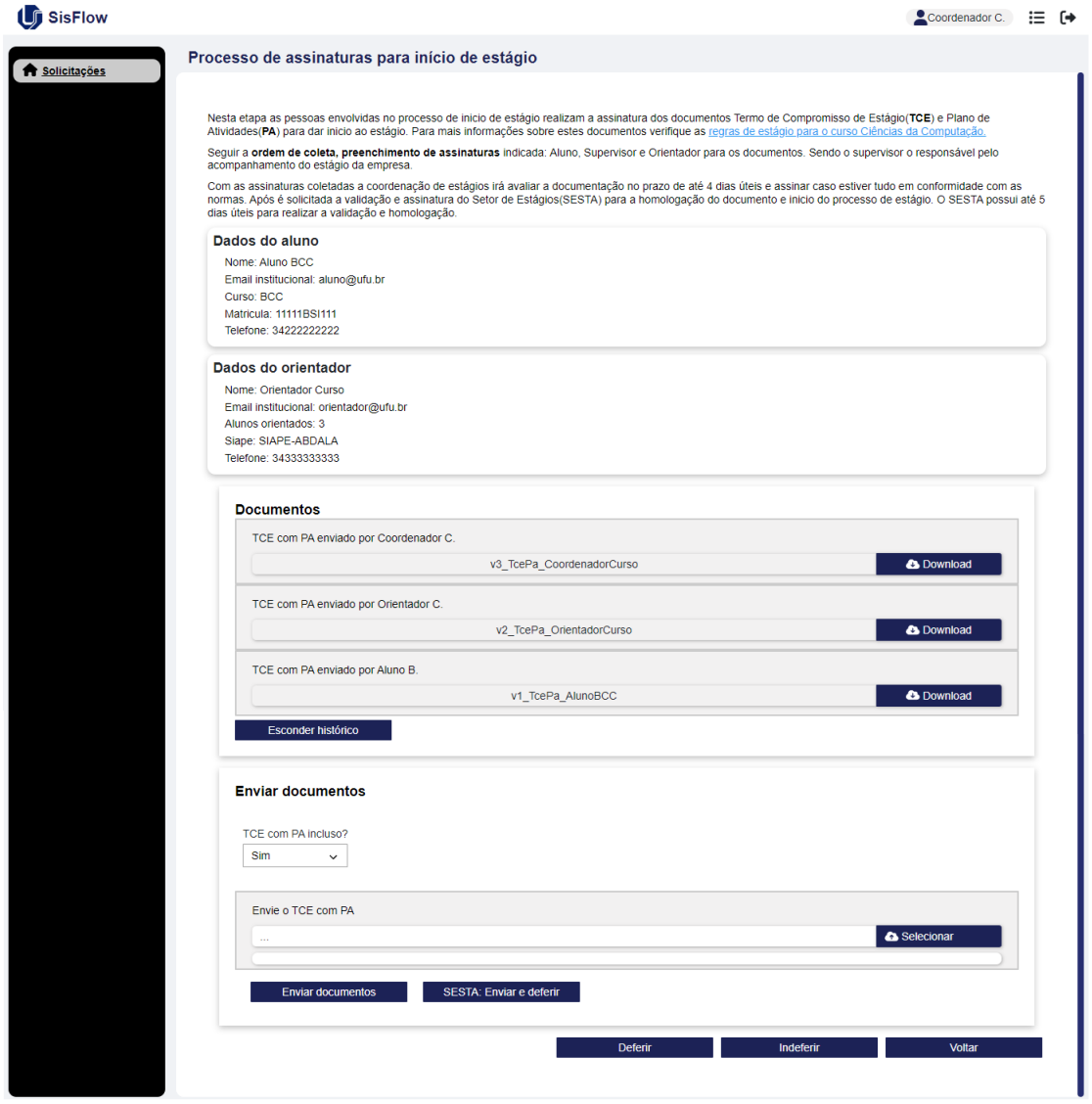


Figura 45 – Tela *Signatures* com o processo de assinaturas para início de estágio (estado S5).

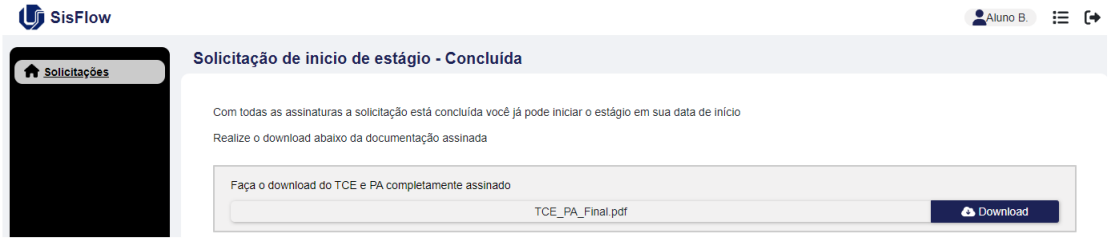


Figura 46 – Tela *DynamicSolicitation* com o estágio iniciado (estado S6).

Suas solicitações

Solicitação	Descricao	Data e hora	Decisao	Motivo	Ação
Inicio de estágio obrigatório externo com vinculo empregatício	Estágio iniciado	2023/11/14 14:38:04	Deferido	Finalizado	
Inicio de estágio obrigatório externo com vinculo empregatício	Processo de assinaturas para início de estágio	2023/11/14 14:35:27	Deferido	Documentos homologados	
Inicio de estágio obrigatório externo com vinculo empregatício	Processo de assinaturas para início de estágio	2023/11/14 14:34:33	Enviado	Atualização de documentos pela coordenação	
Inicio de estágio obrigatório externo com vinculo empregatício	Processo de assinaturas para início de estágio	2023/11/14 14:26:22	Enviado	Atualização de documentos pelo orientador	
Inicio de estágio obrigatório externo com vinculo empregatício	Processo de assinaturas para início de estágio	2023/11/14 14:25:06	Enviado	Atualização de documentos pelo aluno	
Inicio de estágio obrigatório externo com vinculo empregatício	Aceite de orientado pelo orientador	2023/11/14 14:24:28	Deferido	O orientador aceitou a solicitação	
Inicio de estágio obrigatório externo com vinculo empregatício	Escolha de orientador pelo aluno	2023/11/14 14:23:02	Solicitado	O aluno solicitou a orientação ao orientador	
Inicio de estágio obrigatório externo com vinculo empregatício	Avaliação dos históricos e complementos pelo coordenador	2023/11/14 14:22:31	Deferido	A documentação do aluno está aprovada	
Inicio de estágio obrigatório externo com vinculo empregatício	Solicitação de avaliação dos históricos e complementos pelo aluno	2023/11/14 14:22:07	Solicitado	O aluno solicitou avaliação de documentos à coordenação de estágios	

Figura 47 – Parte da tela *Home* com tabela de solicitações do aluno resultante após o fim do processo de início do estágio.