Desenvolvimento de Módulos de Apoio ao Planejamento Acadêmico

Trabalho de Conclusão de Curso

André Lucas Coelho dos Santos

Flávia Maristela
Orientadora

Sandro Andrade
Coorientador

Instituto Federal da Bahia - IFBA
Curso de Análise e Desenvolvimento de Sistemas
Campus Salvador

Salvador, Bahia, Brasil
1 de fevereiro de 2026

Sumário

1 Visão geral

1.1 Declaração do Problema

O planejamento da oferta de disciplinas em instituições de ensino como o Instituto Federal da Bahia (IFBA) é um processo complexo e possui alto impacto na vida acadêmica. Atualmente, as coordenações dos cursos costumam se basear em estimativas para definir a quantidade de turmas e horários para o semestre letivo seguinte. Em muitos casos, os processos de pré-matrícula, quando ocorrem, são realizados de forma descentralizada, utilizando sistemas individuais e informais como formulários online, e sendo divulgados em canais de comunicação informais, como grupos de mensagens e e-mail. Essa abordagem fragmentada carece de uma plataforma unificada e integrada, gerando ineficiências significativas.

Para a gestão acadêmica, a ausência de um sistema centralizado para coletar a intenção de matrícula dos alunos pode acabar resultando em uma tomada de decisão com baixo embasamento em dados. As coordenações enfrentam dificuldades para determinar com precisão a demanda real por cada disciplina, a quantidade ideal de turmas a serem abertas e a preparação para alocação mais eficiente de salas. Como consequência, a instituição se depara com problemas recorrentes, como a falta de vagas em componentes curriculares obrigatórios e a alocação subótima de recursos.

Do ponto de vista discente, o problema se manifesta na falta de ferramentas centralizadas que apoiem seu planejamento acadêmico de forma integrada. A consulta à matriz curricular, a verificação de componentes pendentes e a manifestação de interesse em futuras disciplinas são, muitas vezes, processos desarticulados que dependem de múltiplas fontes de informação, ou essas informações estão mal disponibilizadas. Essa falta de clareza sobre a jornada acadêmica dificulta as escolhas do aluno. Adicionalmente, a ausência de um canal de feedback estruturado e integrado para a avaliação de disciplinas representa uma oportunidade perdida para a melhoria contínua da qualidade dos cursos.

A motivação para este trabalho surge da necessidade de solucionar essa lacuna dupla: otimizar a gestão acadêmica e, simultaneamente, enriquecer a experiência do estudante. A justificativa para o desenvolvimento deste projeto reside na criação de uma ferramenta estratégica que, ao ser integrada ao Emile, centralizará funcionalidades cruciais de planejamento. Para a gestão, o projeto fornecerá dados concretos para uma tomada de decisão mais assertiva e eficiente. Para os estudantes, oferecerá uma plataforma unificada que proporciona clareza sobre seu percurso formativo e lhes confere um papel mais ativo neste processo. Portanto, este projeto visa resolver o problema da ineficiência e da fragmentação no planejamento acadêmico por meio de uma solução tecnológica integrada.

1.1.1 Objetivos

O objetivo geral deste trabalho é desenvolver funcionalidades específicas para o aplicativo acadêmico Emile[1], fornecendo uma ferramenta robusta de apoio à gestão e acompanhamento acadêmica, sendo portanto relevante tanto no contexto docente, quanto no contexto discente. Através das novas funcionalidades apresentadas neste relatório, buscamos contribuir para

[1] O aplicativo Emile foi desenvolvido por professores do GSORT. HOje, o desenvolvimento de funcionalidades e a gestão do aplicativo é coordenada pelo Prof. Sandro Andrade.

uma melhor gestão dos cursos do IFBA através da implementação das funcionalidades de pré-matrícula, consulta curricular (componentes curriculares e a matriz curricular do curso) e avaliação de disciplinas.

Para que o objetivo geral seja atingido, foram delineados os seguintes objetivos específicos:

- Implementar a funcionalidade que permita aos estudantes demonstrar interesse em cursar disciplinas em semestre subsequente(processo de pré-matrícula);
- Implementar uma interface para que as coordenações de curso possam visualizar os dados agregados da pré-matrícula de forma consolidada.
- Disponibilizar a visualização da matriz curricular do curso do aluno diretamente no aplicativo;
- Implementar uma funcionalidade que permita aos alunos visualizar os componentes curriculares já cursados e pendentes;
- Desenvolver um sistema de avaliação de disciplinas que permita a coleta de dados qualitativos e quantitativos, garantindo o anonimato dos estudantes na exibição dos resultados para os docentes;
- Garantir a integração segura e eficiente das novas funcionalidades com a arquitetura e a base de dados do aplicativo existente.

1.2 Proposta de Solução de Software

A solução proposta para o problema apresentado consiste no desenvolvimento e integração de um conjunto de quatro novas funcionalidades no aplicativo acadêmico de código aberto Emile. Este projeto atuará como uma ferramenta de planejamento e apoio, complementando os sistemas oficiais de gestão acadêmica, como o SUAP, sem a intenção de substituí-los. A solução foi concebida para centralizar informações cruciais e criar novos canais de comunicação e coleta de dados, beneficiando tanto os estudantes quanto a gestão dos cursos.
As funcionalidades desenvolvidas são:

- Sistema de Pré-Matrícula: Esta é a funcionalidade central do projeto. Ela permitirá aos alunos manifestarem interesse nas disciplinas do próximo semestre, baseando-se na oferta de turmas do período anterior. A importância desta ferramenta é estratégica, pois fornecerá às coordenações de curso dados concretos sobre a demanda discente, permitindo um planejamento mais assertivo sobre a quantidade de turmas, horários e alocação de recursos.
- Visualização da Matriz Curricular: A solução integrará um módulo para acesso direto à estrutura curricular completa do curso do estudante, organizada por semestres e incluindo componentes optativos.
- Visualização de Componentes Cursados e Pendentes: Para auxiliar o planejamento do

aluno, o software exibirá de forma clara quais disciplinas da matriz já foram concluídas com aprovação e quais ainda precisam ser cursadas. Esta funcionalidade serve de base para o processo de pré-matrícula e oferece ao discente uma visão clara de sua jornada acadêmica.

- Sistema de Avaliação de Disciplinas: Módulo que permitirá aos discentes avaliar as disciplinas cursadas, fornecendo um feedback estruturado sobre aspectos relevantes. Este sistema cria um canal formal de comunicação que visa a melhoria contínua da qualidade dos cursos.

Em conjunto, estas funcionalidades transformam o aplicativo Emile em uma plataforma mais robusta, que não apenas informa, mas também engaja o aluno em seu percurso formativo e fornece inteligência de dados para uma gestão acadêmica mais eficiente.

1.3 Definições, Siglas e Abreviações

- API - Application Programming Interface (Interface de Programação de Aplicações): Conjunto de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na Web.
- QML - Qt Modeling Language: Uma linguagem declarativa utilizada para projetar interfaces de usuário, parte do framework Qt.
- RF - Requisito Funcional: Descreve o que o sistema deve fazer, segundo Sommerville (SOMMERVILLE, 2015), requisitos funcionais definem os serviços que o sistema deve fornecer.
- RNF - Requisito Não Funcional: Descreve como o sistema deve operar, definindo critérios de qualidade como desempenho, usabilidade e segurança.

1.4 Tecnologias da solução

A seleção das tecnologias para este projeto baseia-se na arquitetura preexistente do aplicativo Emile e nos requisitos de desempenho, segurança e manutenibilidade das novas funcionalidades. A seguir, são descritas as tecnologias utilizadas e as justificativas para suas escolhas.

Frontend (Aplicação Cliente): Qt com QML e C++ A interface do usuário e a lógica do lado do cliente serão desenvolvidas utilizando o framework Qt (THE QT COMPANY, 2025). A linguagem QML será empregada para a construção declarativa das interfaces, enquanto uma biblioteca externa em C++ (ISO/IEC, 2020), o Intermix, será utilizada para realizar tarefas como web scraping. A escolha por manter o ecossistema Qt se justifica pela necessidade de garantir a integração e a consistência visual e funcional com o aplicativo Emile existente, facilitando a manutenção futura e aproveitando a base de código já consolidada.

Backend: Python com FastAPI O backend, responsável por toda a lógica de negócio e comunicação

com o banco de dados, será desenvolvido em Python (ROSSUM; DRAKE, 2025), utilizando o framework FastAPI (RAMíREZ, 2025). Python foi escolhido para manter o ecossistema e garantir a manutenibilidade do código. O FastAPI destaca-se pela alta performance, suporte nativo à tipagem e geração automática de documentação interativa, sendo ideal para a criação de APIs RESTful modernas, rápidas e seguras, sem abrir mão da simplicidade e flexibilidade no desenvolvimento.

Banco de Dados: MariaDB Para a persistência dos dados, será utilizado um sistema de gerenciamento de banco de dados relacional, o MariaDB (MARIADB FOUNDATION, 2025). Trata-se de uma solução de código aberto, altamente compatível com o ecossistema MySQL, amplamente utilizada e com forte suporte da comunidade. O MariaDB é adequado para armazenar os dados estruturados do projeto, como as seleções de pré-matrícula e as avaliações de disciplinas, garantindo desempenho, integridade e segurança das informações, além de oferecer maior liberdade tecnológica por não estar vinculado a uma solução proprietária.

Comunicação: API RESTful A comunicação entre o aplicativo frontend e o servidor backend será realizada através de uma API RESTful (FIELDING, 2000). Este padrão de arquitetura foi escolhido por ser o padrão de mercado para a comunicação entre sistemas distribuídos. Ele permite um desacoplamento claro entre o cliente e o servidor, facilitando o desenvolvimento, a manutenção e a escalabilidade da solução.

Controle de Versão: Git e GitLab O controle de todo o código-fonte do projeto será gerenciado pelo Git (SOFTWARE FREEDOM CONSERVANCY, 2025), com os repositórios hospedados na plataforma GitLab (GITLAB INC., 2025). O Git é a ferramenta padrão para versionamento de código, essencial para rastrear alterações e permitir o trabalho colaborativo. O GitLab complementa o Git oferecendo um ambiente completo para gerenciamento do ciclo de vida do software, incluindo o armazenamento do repositório.

Análise Estática e Qualidade de Código: Ruff e Flake8 Para assegurar a manutenibilidade e a padronização do código desenvolvido no backend, foram integradas ferramentas de análise estática ao fluxo de integração contínua. O **Ruff** (MARSH, 2025) foi adotado por sua alta performance, atuando como linter e formatador para garantir consistência estilística e detecção rápida de erros. Complementarmente, o **Flake8** (PYCQA, 2016) é utilizado para validações rigorosas de conformidade com a PEP 8 (ROSSUM; WARSAW; COGHLAN, 2001) e controle de complexidade ciclomática, garantindo que apenas código que atenda aos critérios de qualidade definidos seja integrado ao repositório principal.

1.5 Trabalhos Relacionados

A análise de sistemas de gestão acadêmica consolidados é fundamental para compreender o cenário em que este projeto se insere. A seguir, são apresentados dois sistemas relevantes no contexto da educação pública federal brasileira: o SUAP e o SIGAA. O objetivo desta análise é diferenciar o escopo do aplicativo Emile das soluções existentes, destacando como as funcionalidades propostas buscam preencher lacunas específicas de planejamento,

engajamento e usabilidade.

Sistema Unificado de Administração Pública (SUAP)

O SUAP é o sistema oficial de gestão acadêmica e administrativa adotado pelo Instituto Federal da Bahia (IFBA) e por grande parte da Rede Federal de Educação Profissional, Científica e Tecnológica. É uma plataforma robusta que centraliza processos essenciais da instituição, incluindo a matrícula oficial, registro de notas, emissão de histórico escolar e gestão de pessoal.

No contexto deste trabalho, o SUAP atua como o sistema de registro oficial (o "sistema da verdade"). A solução desenvolvida não visa substituí-lo, mas sim complementá-lo. Enquanto o SUAP gerencia o vínculo formal e os registros definitivos, o módulo do aplicativo Emile foca na etapa anterior: o planejamento. A funcionalidade de pré-matrícula gera dados de intenção que subsidiam as coordenações na configuração da oferta de turmas no SUAP com maior assertividade. Simultaneamente, a visualização de componentes pendentes e da matriz curricular consome dados do SUAP para apresentá-los de forma mais amigável, facilitando a organização da trajetória discente. Adicionalmente, o projeto introduz um Sistema de Avaliação de Disciplinas, criando um canal de feedback qualitativo e anônimo que inexiste na implementação atual do sistema institucional.

Sistema Integrado de Gestão de Atividades Acadêmicas (SIGAA)

O SIGAA é amplamente utilizado por diversas universidades federais no Brasil, destacando-se por integrar, em uma única plataforma, a gestão de graduação, pós-graduação, pesquisa e extensão. O sistema oferece um vasto conjunto de funcionalidades para alunos, docentes e gestores, incluindo a visualização de estruturas curriculares e matrículas online.

Para este projeto, o SIGAA serve como referência de funcionalidades maduras. Embora ele já contemple um módulo de avaliação, o diferencial deste trabalho reside na implementação dessas ferramentas em uma plataforma móvel de código aberto, com foco na experiência do usuário (UX). Diferente da abordagem tradicional, que por vezes torna a avaliação uma obrigação burocrática, o projeto busca reduzir a fricção e garantir o anonimato na exibição dos comentários para incentivar a participação real dos estudantes. Além disso, a proposta de uma pré-matrícula voltada à inteligência de dados — permitindo prever demandas antes do período oficial — explora um nicho de apoio à decisão estratégica que muitas vezes não é o foco principal dos grandes sistemas monolíticos de registro acadêmico.

2 Requisitos

2.1 Requisitos Funcionais

RF1 Como um aluno, eu quero visualizar uma lista de todos os componentes curriculares nos quais ainda não fui aprovado, tanto obrigatórios quanto optativos, para que eu possa planejar minha trajetória acadêmica e me preparar para a pré-matrícula.

RF2 Como um aluno, eu quero visualizar uma lista de todos os componentes curriculares nos

quais já fui aprovado, para que eu possa acompanhar meu progresso no curso e ter um registro claro do meu histórico acadêmico.

RF3 Como um aluno, eu quero iniciar o processo de pré-matrícula, visualizando as disciplinas pendentes e as turmas disponíveis para elas, para que eu possa manifestar meu interesse em cursá-las no próximo semestre.

RF4 Como um aluno, eu quero selecionar as turmas nas quais desejo me matricular e ser informado sobre conflitos de horário, para que eu possa montar uma grade viável e submeter uma pré-matrícula válida.

RF5 Como um aluno, eu quero salvar e submeter as disciplinas nas quais demonstrei interesse durante a pré-matrícula, para que eu possa formalizar minhas intenções de matrícula e contribuir com os dados para o planejamento da oferta de turmas.

RF6 Como um aluno, eu quero visualizar a matriz curricular completa do meu curso, organizada por semestres, e as optativas disponíveis para que eu possa entender a estrutura e o fluxo recomendado de disciplinas.

RF7 Como um aluno, eu quero acessar um formulário de avaliação para as disciplinas que já cursei, para que eu possa fornecer meu feedback sobre a qualidade do curso.

RF8 Como um aluno, eu quero submeter um formulário de avaliação com notas e comentários, para que eu possa contribuir para a melhoria contínua das disciplinas e do curso.

RF9 Como um aluno, eu quero poder visualizar o formulário de avaliação com notas e comentários que realizei.

RF10 Como um coordenador de curso, eu quero exportar/baixar um arquivo de relatório consolidado com o número de alunos interessados por disciplina e turma, para que eu possa analisar os dados externamente e planejar a oferta do próximo semestre.

RF11 Como um professor, eu quero consultar a média de avaliações de cada disciplina que eu ministro e ver comentários individuais dos alunos anônimos, para que eu possa absorver os feedbacks e identificar oportunidades de melhoria.

2.2 Requisitos Não-Funcionais

RNF01 Usabilidade: A interface das novas funcionalidades deve ser intuitiva, clara e de fácil navegação, seguindo a identidade visual do aplicativo Emile existente.

RNF02 Desempenho: As consultas de matriz curricular, histórico e dados de pré-matrícula devem ser executadas em tempo hábil. O sistema deve suportar o acesso simultâneo de múltiplos usuários durante o período de pré-matrícula sem degradação de performance.

RNF03 Segurança: Os dados dos alunos devem ser acessados apenas por eles mesmos, mediante autenticação no SUAP. As avaliações de disciplinas devem ser disponibilizadas aos docentes sem qualquer vínculo com o aluno que as enviou.

RNF04 Confiabilidade O sistema deve garantir que os dados de pré-matrícula e avaliação sejam armazenados de forma íntegra e sem perdas.

RNF05 Compatibilidade: As novas funcionalidades devem ser compatíveis com as versões dos sistemas operacionais (Android/iOS) suportadas pelo aplicativo Emile.]

RNF06 Manutenibilidade : O código-fonte desenvolvido deve ser bem documentado, modularizado e seguir os padrões de projeto do aplicativo existente para facilitar futuras manutenções e evoluções.

3 Design

3.1 Projeto UML

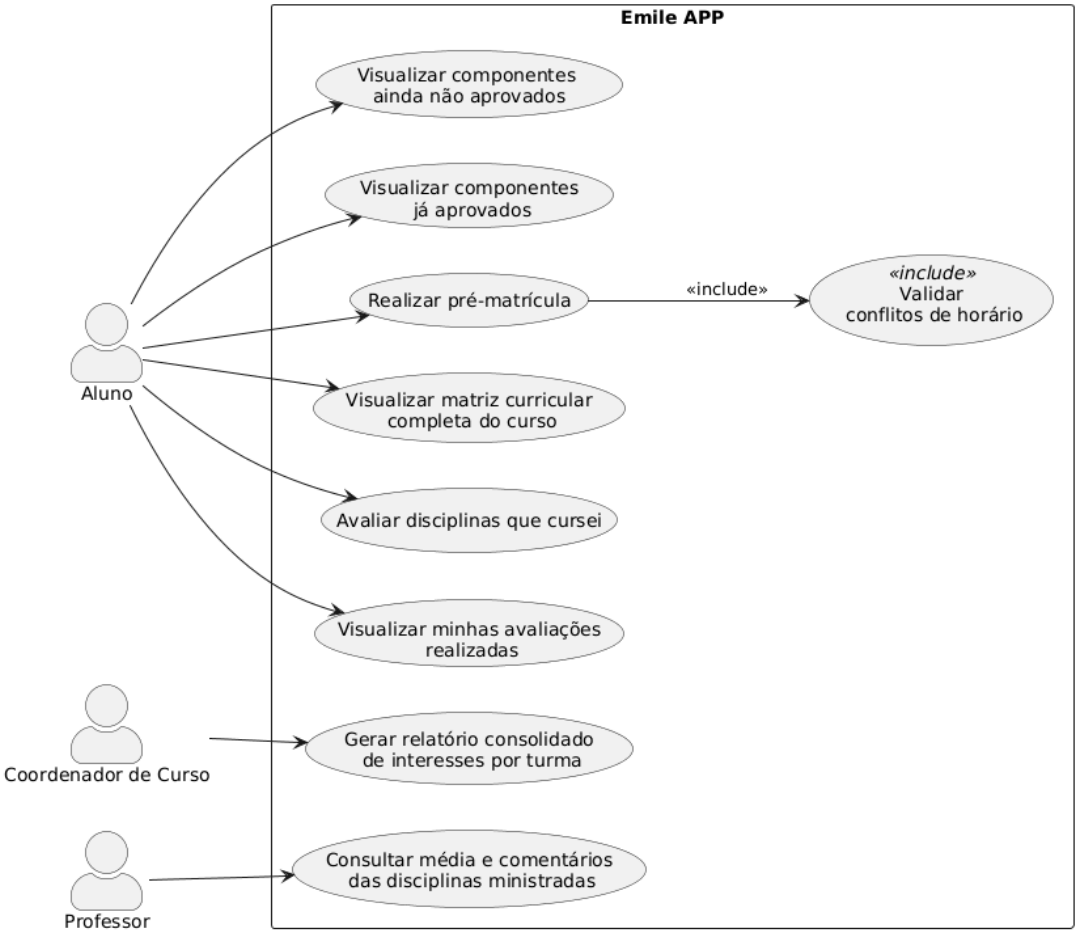

Figura 1: Diagrama de Caso de Uso

3.2 Visão arquitetural

Para a documentação e visualização da arquitetura do sistema, foi adotado o *C4 Model* (Context, Containers, Components, and Code). Esta abordagem, proposta por Brown (2025), permite descrever a estrutura do software em diferentes níveis de abstração, facilitando a

comunicação tanto com *stakeholders* técnicos quanto não técnicos. O modelo foi escolhido por sua capacidade de ilustrar claramente as fronteiras do sistema e suas integrações externas (BROWN, 2025).

A arquitetura da solução desenvolvida para o aplicativo Emile foi estruturada focando nos dois primeiros níveis do modelo: Contexto e Contêineres.

3.2.1 Nível de Contexto

O diagrama de contexto, apresentado na Figura 2, situa o sistema Emile no ecossistema do IFBA. Ele ilustra as interações de alto nível entre os usuários (alunos e professores) e o sistema, bem como as dependências de sistemas externos essenciais, como o SUAP (para autenticação e dados acadêmicos) e serviços de terceiros (Google, OpenAI e Firebase).

Figura 2: Diagrama de contexto do Emile

3.2.2 Nível de Contêineres

Aprofundando a visão arquitetural, o diagrama de contêineres (Figura 3) detalha as unidades implantáveis que compõem o sistema Emile. A solução foi segmentada em três contêineres principais:

- **Aplicativo Móvel (Frontend):** Interface desenvolvida em Qt/QML, responsável pela interação direta com o usuário;
- **API Backend:** Desenvolvida em Python com FastAPI, centraliza a lógica de negócios das novas funcionalidades;
- **Banco de Dados:** Instância MariaDB isolada para persistência dos dados de pré-matrícula e avaliações, garantindo a integridade dos dados.

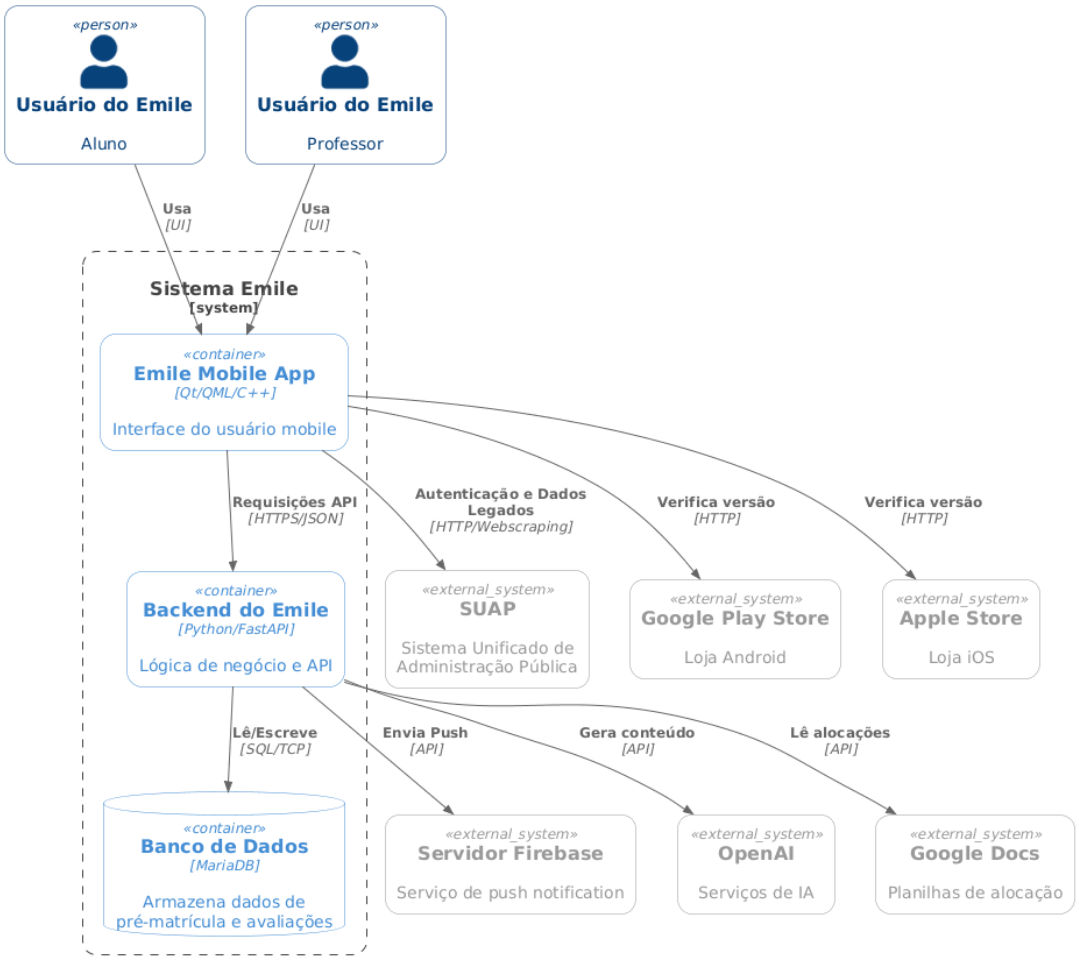

Figura 3: Diagrama de containers do Emile

3.3 Modelo de Banco de Dados

O **MariaDB** foi selecionado como o Sistema de Gerenciamento de Banco de Dados Relacional (SGBDR) para garantir a **persistência, integridade e segurança** dos dados essenciais introduzidos pelas novas funcionalidades.

Embora o Emile se integre ao SUAP para consulta de dados cadastrais, o banco de dados próprio do projeto centraliza as **novas interações e dados estratégicos**. Dentre as funcionalidades desenvolvidas, o banco atende especificamente ao **Sistema de Pré-Matrícula** e ao **Sistema de Avaliação de Disciplinas**.

A estrutura de dados foi modelada para atender aos seguintes requisitos:

- **Pré-Matrícula:** O banco armazena as turmas disponíveis para o semestre subsequente, as quais são inseridas através de um script inicial de população. Dessa forma, as intenções de matrícula registradas pelos alunos possuem integridade referencial direta com essas turmas locais, garantindo maior performance e independência do sistema legado, conforme ilustrado na Figura 4.

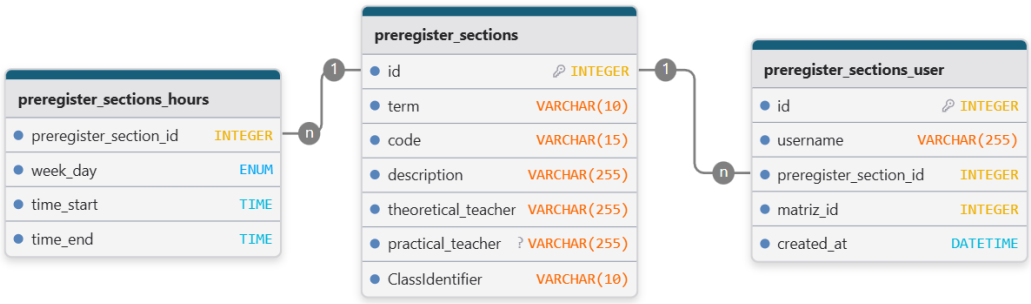

Figura 4: Modelo de dados para pré-matrícula

- **Avaliação de Disciplinas:** O sistema armazena as **avaliações quantitativas e qualitativas** (notas e comentários). A modelagem foi concebida para dissociar a identidade do aluno dos resultados exibidos publicamente, garantindo o **anonimato** necessário para o processo avaliativo (Figura 5).

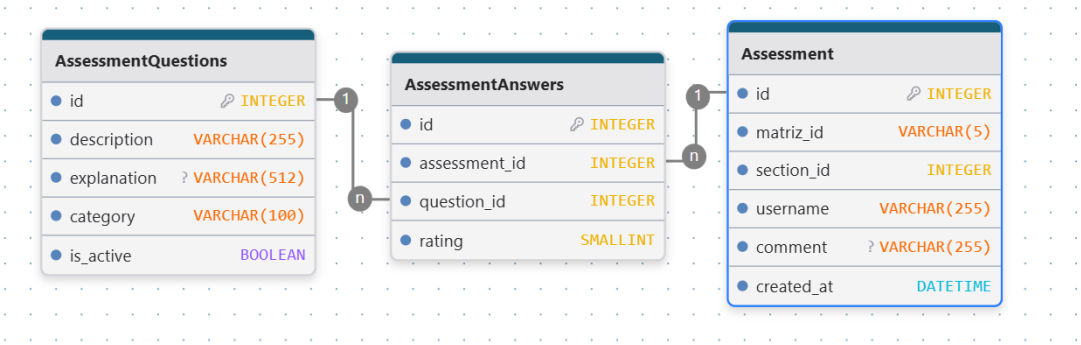

Figura 5: Modelo de dados para avaliação de disciplina

3.4 Detalhes de Implementação

Esta seção apresenta a materialização da arquitetura e dos requisitos definidos anteriormente por meio de trechos de código críticos para o funcionamento do sistema. A implementação priorizou a integridade dos dados e o processamento eficiente no *backend* (API), garantindo que as regras de negócio fossem centralizadas e desacopladas da interface do usuário.

3.4.1 Processamento de Dados Analíticos

Um dos objetivos principais do trabalho é fornecer inteligência de dados à coordenação do curso. Para isso, foi desenvolvido um algoritmo de agregação que transforma as avaliações individuais dos alunos em relatórios estatísticos consolidados.

A Figura 6 exibe a função `get_assessment_report`, que itera sobre os registros armazenados no banco de dados, calculando a média aritmética e gerando histogramas de frequência para cada critério avaliado. O código separa as métricas quantitativas (notas de 1 a 5)

dos *feedbacks* qualitativos (comentários), retornando uma estrutura JSON otimizada para renderização de gráficos no *frontend*.

```python
@router.get("/assessments/report", response_model=AssessmentReport)
def get_assessment_report(
    section_id: int, token: str = Depends(validate_token), session: Session = Depends(get_session)
):
    # Busca as avaliações realizadas para a disciplina
    assessments_query = select(SectionAssessment).where(SectionAssessment.section_id == section_id)
    assessments = session.exec(assessments_query).all()

    total_assessments = len(assessments)
    assessment_ids = [a.id for a in assessments]

    # Separa em uma lista todos os comentários realizados durante as avaliações
    # protegendo a identidade dos usuários
    comments_list = [a.comment for a in assessments if a.comment and a.comment.strip()]

    # Se não houver avaliações, retorna um relatório vazio
    if not assessment_ids:
        return AssessmentReport(section_id=section_id, total_assessments=0, questions_stats=[], comments=[])

    # Consulta as perguntas para organizar os dados na tela de relatório
    stmt = (
        select(SectionAssessmentAnswer, SectionAssessmentQuestion)
        .join(SectionAssessmentQuestion, SectionAssessmentAnswer.question_id == SectionAssessmentQuestion.id)
        .where(SectionAssessmentAnswer.assessment_id.in_(assessment_ids))
        .order_by(
            SectionAssessmentQuestion.category,
            SectionAssessmentQuestion.description,
        )
    )
    results = session.exec(stmt).all()

    stats_map = {}

    # Agrupa os dados por pergunta para calcular as estatísticas
    for answer, question in results:
        q_id = question.id
        if q_id not in stats_map:
            stats_map[q_id] = {
                "sum": 0,
                "count": 0,
                "ratings": {},
                "description": question.description,
                "explanation": question.explanation,
                "category": question.category,
            }

        stats_map[q_id]["sum"] += answer.rating
        stats_map[q_id]["count"] += 1

        ratings_map = stats_map[q_id]["ratings"]
        ratings_map[answer.rating] = ratings_map.get(answer.rating, 0) + 1

    # Monta a lista de estatísticas por pergunta
    questions_stats = []
    for q_id, data in stats_map.items():
        avg = data["sum"] / data["count"] if data["count"] > 0 else 0
        evaluations_per_rate = sorted(
            [(rating, count) for rating, count in data["ratings"].items()],
            key=lambda x: x[0],
        )
        questions_stats.append(
            QuestionStats(
                question_id=q_id,
                description=data["description"],
                explanation=data["explanation"],
                category=data["category"],
                average_rating=round(avg, 1),
                total_responses=data["count"],
                evaluations_per_rate=evaluations_per_rate,
            )
        )

    # Ordena as perguntas por categoria
    questions_stats.sort(key=lambda x: x.category)

    return AssessmentReport(
        section_id=section_id,
        total_assessments=total_assessments,
        questions_stats=questions_stats,
        comments=comments_list,
    )
```

Figura 6: Trecho de código do endpoint que processa os dados das avaliações realizadas

3.4.2 Ingestão e Normalização de Dados (ETL)

A viabilização do módulo de pré-matrícula depende da existência prévia dos dados das turmas e dos horários no banco de dados da solução. Visto que não é disponibilizada uma API pública para consulta direta destas ofertas, foi desenvolvida uma rotina de Extração, Transformação e Carga (ETL).

O processo de **ingestão** dos dados inicia-se com a obtenção do relatório de turmas ofertadas no SUAP. Estes dados brutos são submetidos a um endpoint no backend, responsável por povoar as tabelas do banco de dados do Emile. O fluxo completo desta integração é detalhado na Figura 7.

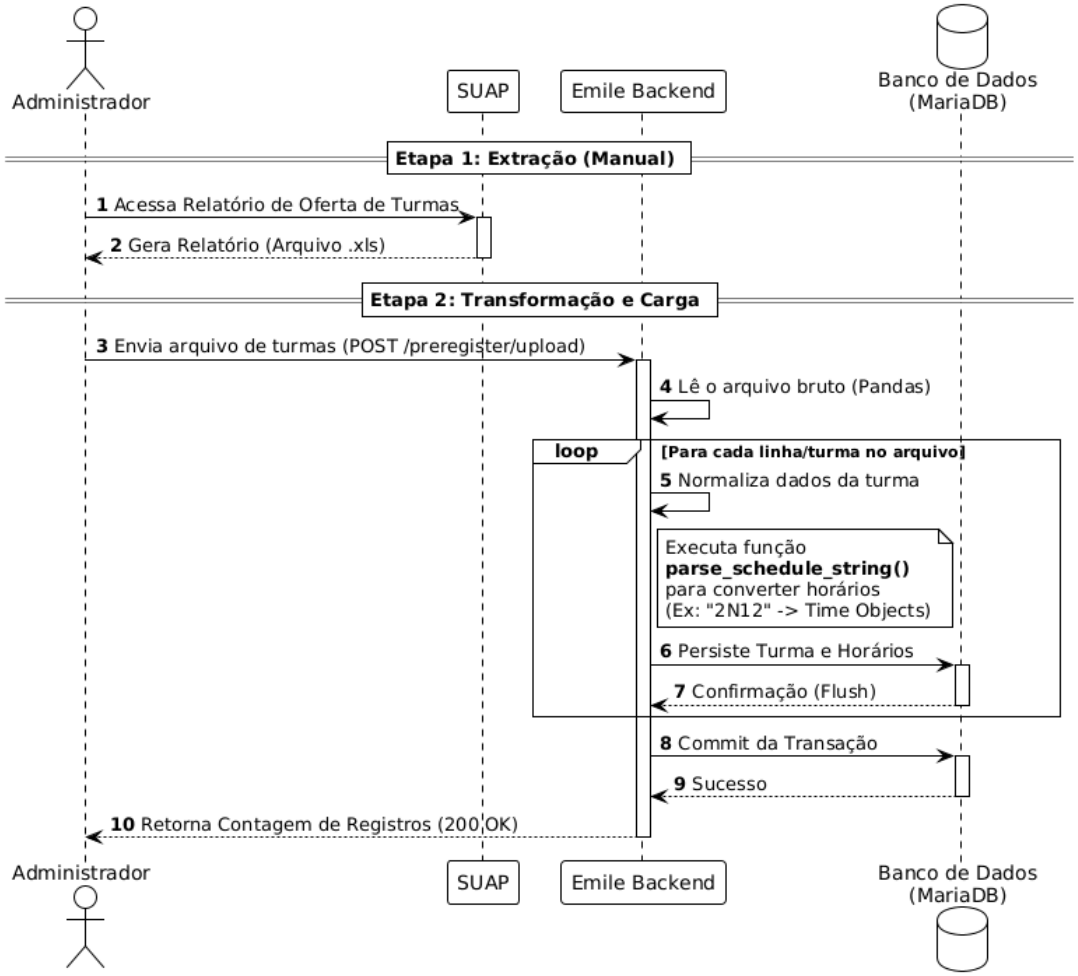

Figura 7: Fluxo do processo de ETL para importação de turmas do SUAP

Durante a etapa de **normalização** das informações, para garantir a consistência com a modelagem proposta, um dos principais desafios é tratar a heterogeneidade da notação de horários. Destaca-se a função `parse_schedule_string`, responsável por interpretar a notação compacta do SUAP (ex: “2N1516”) e convertê-la em estruturas de tempo padronizadas (`datetime.time`), viabilizando cálculos precisos de duração e intervalos das aulas, conforme ilustrado na Figura 8.

```python
TIME_SLOTS = {
    "M1": (time(7, 0), time(7, 50)),
    "M2": (time(7, 50), time(8, 40)),
    "M3": (time(8, 40), time(9, 30)),
    "M4": (time(9, 30), time(10, 20)),
    "M5": (time(10, 40), time(11, 30)),
    "M6": (time(11, 30), time(12, 20)),
    "V7": (time(13, 20), time(14, 10)),
    "V8": (time(14, 10), time(15, 0)),
    "V9": (time(15, 20), time(16, 10)),
    "V10": (time(16, 10), time(17, 0)),
    "V11": (time(17, 0), time(17, 50)),
    "V12": (time(17, 50), time(18, 40)),
    "N13": (time(18, 40), time(19, 30)),
    "N14": (time(19, 30), time(20, 20)),
    "N15": (time(20, 20), time(21, 10)),
    "N16": (time(21, 10), time(22, 0)),
}


def parse_schedule_string(schedule_str: str) -> list[dict]:
    if not isinstance(schedule_str, str) or not schedule_str.strip():
        return []

    entries = []
    # Regex para identificar padrões como '2N1516' (Segunda Noite 20:20-22:00)
    matches = re.findall(r"([2-7]+)\s*([MVN])\s*(\d+)", schedule_str.upper())

    for days, shift, slots_str in matches:
        # Função auxiliar para mapear os slots de tempo corretamente (no exemplo, '15' e '16' para Noite)
        slot_keys = parse_slots_for_shift(shift, slots_str)
        # Verifica se os slots são válidos
        valid_keys = [k for k in slot_keys if k in TIME_SLOTS]
        if not valid_keys:
            continue

        # Obtém o horário de início e fim com base nos slots identificados
        start_time = TIME_SLOTS[valid_keys[0]][0]
        end_time = TIME_SLOTS[valid_keys[-1]][1]

        # Cria uma entrada para cada dia identificado
        for day_char in days:
            if day_char in DAY_MAP:
                entries.append({"week_day": DAY_MAP[day_char], "time_start": start_time, "time_end": end_time})

    # Exemplo: '2N1516' vira [{'week_day': WeekDay.Segunda, 'time_start': time(20, 20), 'time_end': time(22, 0)}]
    return entries


def parse_slots_for_shift(shift: str, num_str: str) -> list[str]:
    keys = []
    i = 0
    length = len(num_str)
    while i < length:
        if shift == "M":
            # Se for M (Manhã), cada número é um slot individual
            keys.append(f"M{num_str[i]}")
            i += 1
        elif shift == "N":
            # Se for N (Noite), os slots são de dois em dois
            if i + 1 < length:
                keys.append(f"N{num_str[i : i + 2]}")
                i += 2
            else:
                break
        elif shift == "V":
            # Se for V (Tarde), pode ser um ou dois dígitos
            char = num_str[i]
            # Se for 7, 8 ou 9, é um slot individual
            if char in ["7", "8", "9"]:
                keys.append(f"V{char}")
                i += 1
            else:
                # Caso contrário, tenta pegar dois dígitos
                if i + 1 < length:
                    keys.append(f"V{num_str[i : i + 2]}")
                    i += 2
                else:
                    break

    # Exemplo: '1516' com shift 'N' vira ['N15', 'N16']
    return keys
```

Figura 8: Código responsável por processar os horários no formato do SUAP

3.4.3 Validação de Regras de Negócio e Conflitos

A integridade da pré-matrícula depende estritamente da validação temporal, impedindo que um aluno manifeste interesse em disciplinas que ocorrem simultaneamente. A Figura 9 apresenta a função `validate_schedule_conflicts`, responsável por essa lógica.

O algoritmo realiza uma verificação de intersecção entre intervalos de tempo. Ao receber uma lista de disciplinas desejadas, o sistema compara os horários de início e fim de cada

aula. Caso seja detectada uma sobreposição temporal (onde o início da nova aula é anterior ao término de uma aula já selecionada no mesmo dia), o sistema interrompe a transação e retorna um erro detalhado, informando ao usuário exatamente quais componentes curriculares estão gerando o conflito.

```python
def validate_schedule_conflicts(sections: list[PreregisterSection]):
    """
    Checks for time overlaps in a list of sections.
    Raises HTTPException(409) if a conflict is found.
    """
    daily_schedule = {}

    for section in sections:
        # Garante que os horários estejam carregados
        if not section.hours:
            continue

        for hour in section.hours:
            day = hour.week_day

            if day not in daily_schedule:
                daily_schedule[day] = []

            new_start = hour.time_start
            new_end = hour.time_end

            # Verifica sobreposição com horários já processados
            for existing_start, existing_end, existing_code in daily_schedule[day]:
                # Fórmula de Interseção: (StartA < EndB) and (StartB < EndA)
                if new_start < existing_end and existing_start < new_end:
                    raise HTTPException(
                        status_code=409,
                        detail=(
                            f"Conflito de horário detectado: {section.code} "
                            f"coincide com {existing_code} na {day} "
                            f"entre {new_start} e {new_end}."
                        ),
                    )

            # Adiciona o horário atual na lista de verificação
            daily_schedule[day].append((new_start, new_end, section.code))
```

Figura 9: Trecho de código responsável por garantir verificação de conflito de horários entre as disciplinas escolhidas

4 Qualidade

4.1 Estratégia de Garantia da Qualidade

A estratégia de garantia da qualidade (Quality Assurance - QA) adotada para o desenvolvimento das novas funcionalidades do aplicativo Emile combinou técnicas de análise estática automatizada de código com testes funcionais manuais. Essa abordagem híbrida visou assegurar tanto a manutenibilidade e padronização do código-fonte no backend, quanto a usabili-

dade e a corretude das regras de negócio na interface do usuário (frontend).

4.1.1 Análise Estática e Integração Contínua (Backend)

O backend do Emile, segue um política rigorosa de garantia de qualidade de código. Dado o caráter colaborativo do projeto, foi estabelecida uma esteira de integração contínua(Continuous Integration - CI) no GitLab, prática que visa integrar o trabalho de todos os desenvolvedores em um repositório compartilhado (FOWLER, 2024), a fim de automatizar a verificação de padrões de desenvolvimento antes da incorporação de novas funcionalidades.
A estratégia de testes automatizados para o backend concentra-se na análise estática do código-fonte, atuando como um ”portão de qualidade”(Quality Gate). As ferramentas utilizadas foram:

- Ruff: Adotado por sua alta performance, o Ruff é utilizado no projeto com dupla função:
  - Formatador (Formatter): Garante a consistência estilística do código (espaçamento, quebras de linha, citações), eliminando debates subjetivos sobre estilo em revisões de código.
  - Linter Rápido: Identifica erros de sintaxe e violações básicas de boas práticas em tempo de execução reduzido, otimizando o tempo total do pipeline de CI.
- Flake8: Utilizado em conjunto com o Ruff para uma análise estática complementar, focando na detecção de:
  - Complexidade Ciclomática: Identificação de trechos de código com lógica condicional excessiva, que dificultam a manutenção e testes futuros.
  - Conformidade com a PEP 8: Validação estrita das convenções de estilo oficiais da linguagem Python que podem não ser cobertas apenas pela formatação automática.

4.1.2 Testes Funcionais

Dada a natureza visual e interativa das novas funcionalidades desenvolvidas em Qt/QML (Frontend), a validação do comportamento do sistema foi realizada através de testes funcionais de caixa preta.
Para cada Requisito Funcional (RF) especificado na Seção 2.1, foram executados cenários de teste manuais para validar:

1. **Fluxo de Sucesso:** Verificação se a funcionalidade (ex: realizar pré-matrícula, baixar relatório) produz o resultado esperado com dados válidos.
2. **Tratamento de Erros:** Verificação do comportamento do aplicativo diante de falhas de conexão ou ausência de dados no Backend.
3. **Usabilidade:** Validação da navegação e feedback visual ao usuário, garantindo a consistência com a identidade do aplicativo Emile.

5 Implantação

5.1 Projeto de implantação

A implantação da solução segue a estratégia de Integração Contínua (CI) já estabelecida no ciclo de vida do aplicativo Emile. As funcionalidades são integradas e disponibilizadas incrementalmente, à medida que são desenvolvidas e validadas. Essa abordagem permite a coleta constante de feedback dos usuários e promove a percepção de evolução contínua do software.

Do ponto de vista técnico, o processo de implantação varia conforme a natureza da funcionalidade:

- Funcionalidades de Web Scraping: Como a feature de Componentes Curriculares, seguem a lógica de extração de dados do SUAP reside na camada cliente, a atualização desses recursos demanda apenas a distribuição de uma nova versão do aplicativo móvel (frontend).
- Funcionalidades com Persistência (Pré-matrícula e Avaliação): Para estes módulos, a implantação é mais abrangente, exigindo a atualização simultânea do frontend e do backend, além da aplicação de migrações no banco de dados para a criação das estruturas necessárias ao armazenamento das novas informações.

6 Manual do Usuário

O manual está estruturado de forma a cobrir os fluxos de trabalho essenciais. Ele aborda desde o acesso às informações acadêmicas até a execução de processos complexos, como a realização da pré-matrícula e a submissão de avaliações de disciplinas. Cada seção a seguir descreve o passo a passo da interação, acompanhado de capturas de tela (screenshots) do sistema em funcionamento, validando a usabilidade e a interface.

6.1 Visualização de Componentes Curriculares

Para acessar as informações acadêmicas, o usuário deve estar autenticado com um perfil de **aluno**. Na tela inicial do aplicativo, deve-se selecionar a opção “Meus componentes curriculares”, conforme destacado na Figura 10.

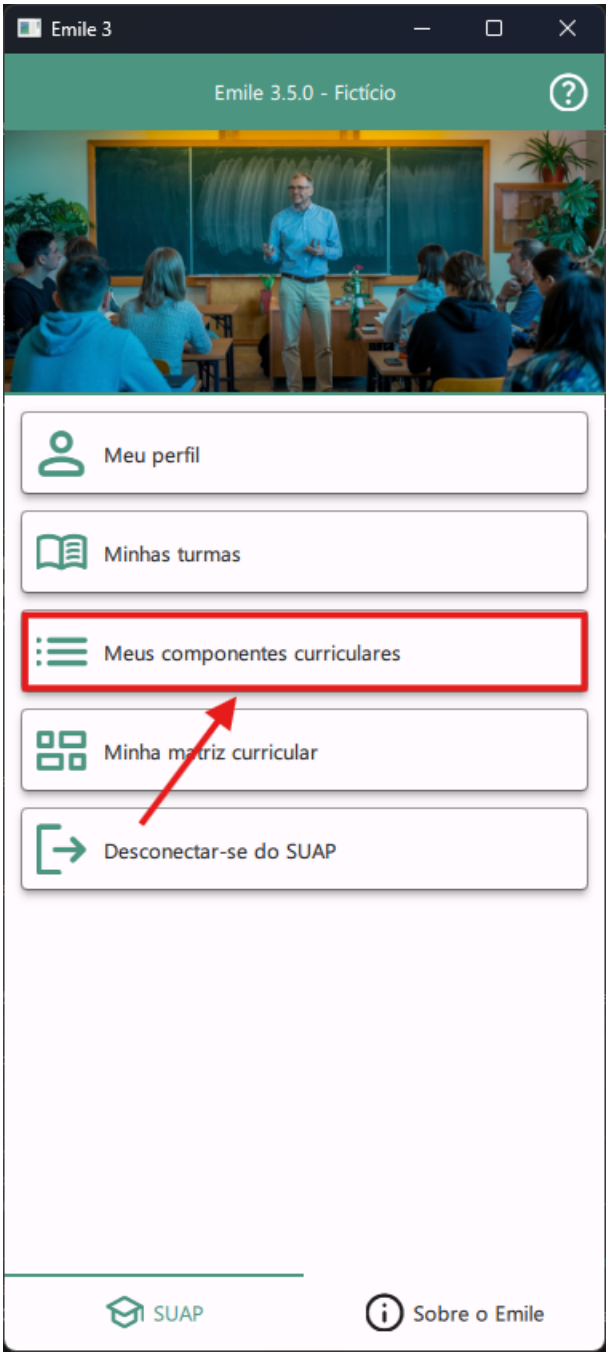

Figura 10: Tela inicial com destaque para o acesso aos componentes

Ao selecionar esta opção, a interface de listagem de disciplinas será carregada (Figura 11).

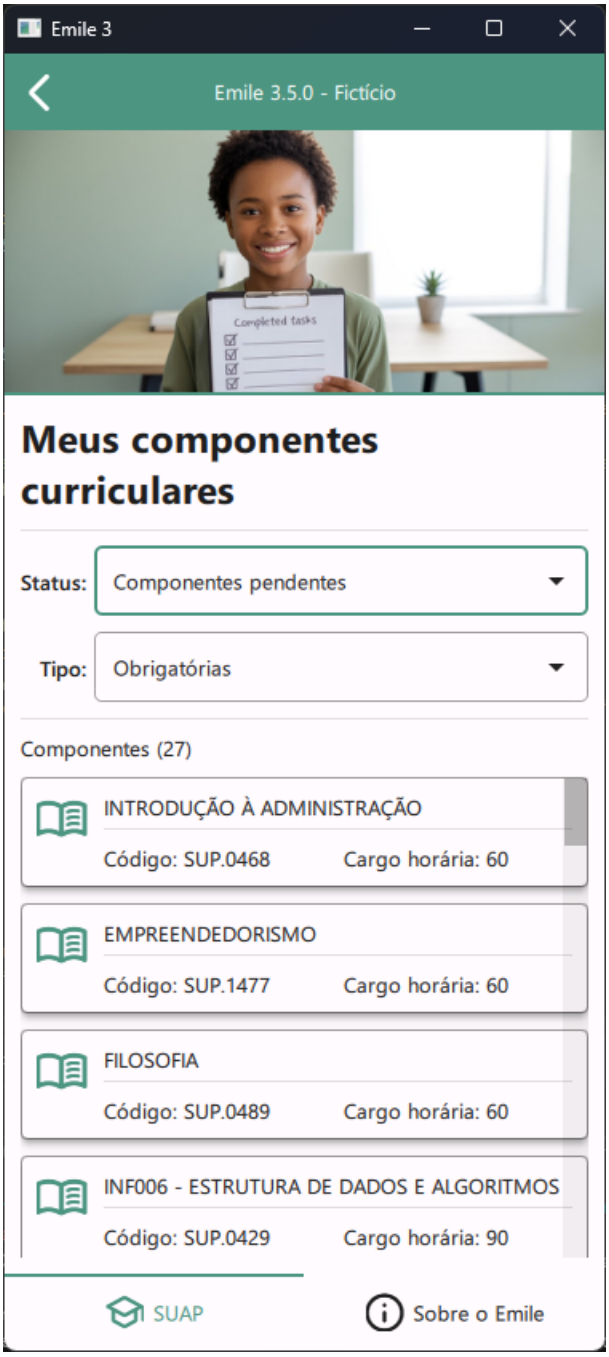

Figura 11: Interface de visualização de componentes curriculares

Nesta tela, é possível alterar a visualização dos dados exibidos conforme a necessidade do aluno. Por padrão, o sistema lista os componentes **obrigatórios** nos quais o aluno já obteve aprovação. Contudo, utilizando os controles superiores, é possível alternar a visualização para exibir componentes **pendentes**, bem como consultar a listagem de disciplinas **optativas**.

6.2 Visualização da Matriz Curricular do Curso

Para acessar a estrutura completa do curso, o usuário deve estar autenticado com um perfil de **aluno**. Na tela inicial do aplicativo, deve-se selecionar a opção “Minha matriz curricular”,

conforme destacado na Figura 12.

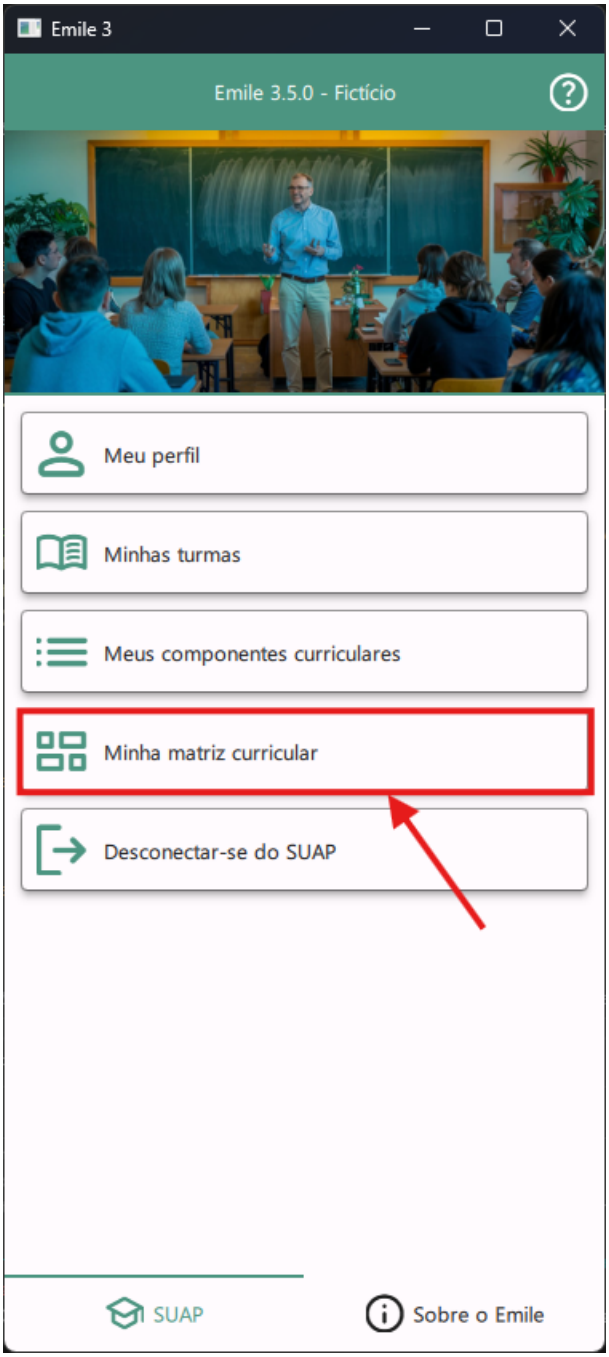

Figura 12: Tela inicial com destaque para o acesso à matriz curricular

Ao selecionar esta opção, a interface de listagem de disciplinas será carregada (Figura 13).

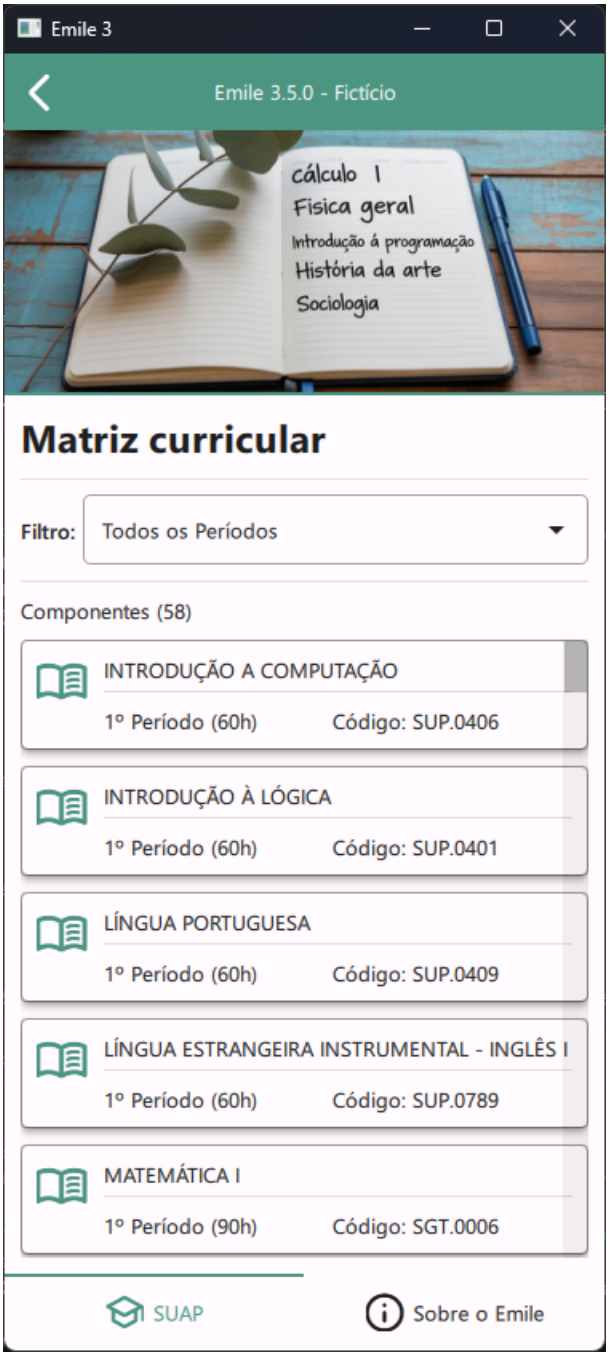

Figura 13: Interface de visualização da matriz curricular

Nesta tela, é possível filtrar os dados exibidos conforme a necessidade do aluno. Por padrão, o sistema lista os componentes de **Todos os Períodos** planejados para o curso. Contudo, utilizando o controle superior, é possível alternar a visualização para exibir apenas os componentes de um período específico, ou apenas os componentes optativos vinculados à matriz do curso.

6.3 Avaliação de Disciplinas

O módulo de avaliação de disciplinas compreende dois fluxos distintos: o envio do feedback por parte do aluno e a consulta consolidada dos resultados pelo docente. A seguir, detalha-se o processo de submissão e análise de uma avaliação.

6.3.1 Realizando a Avaliação de uma Disciplina

Para avaliar um componente curricular, o usuário deve estar autenticado com um perfil de **aluno**. A partir da tela de detalhes da disciplina desejada, deve-se selecionar a opção “Avaliar esta disciplina”, conforme destacado na Figura 14.

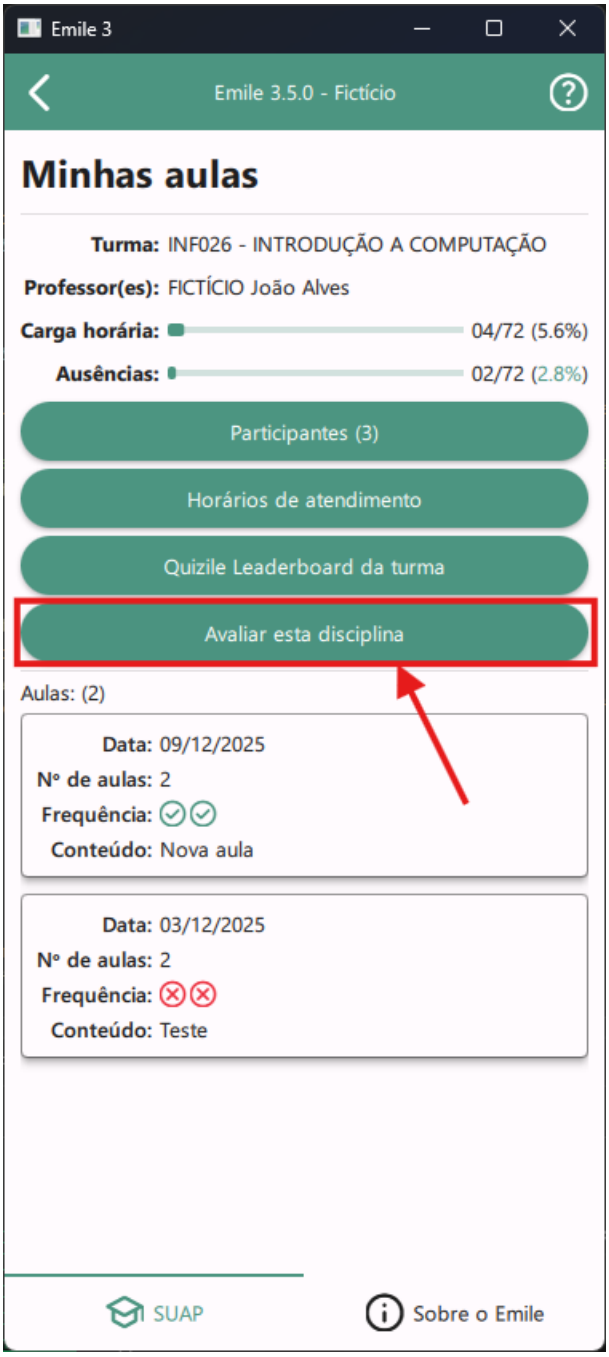

Figura 14: Tela de detalhes da disciplina com acesso ao formulário de avaliação

Ao iniciar o processo, o sistema apresenta um alerta informativo sobre a política de anonimato da avaliação, detalhando como os dados serão tratados, conforme ilustrado na Figura 15.

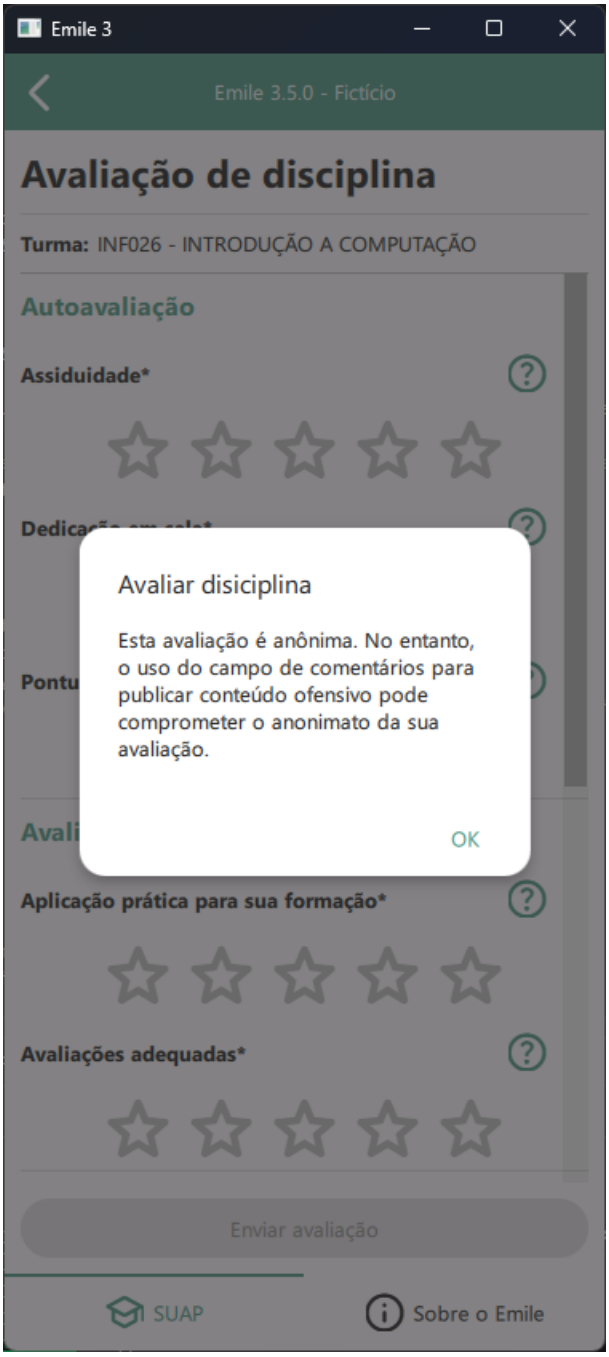

Figura 15: Alerta sobre a política de anonimato da avaliação

Após a confirmação da leitura da mensagem, o formulário é liberado. Por medida de consistência, o botão de “Enviar avaliação” permanece bloqueado até que todos os critérios quantitativos (perguntas obrigatórias) sejam respondidos. Caso o usuário possua alguma dúvida sobre determinado item, é possível selecionar o ícone de ajuda (?) para visualizar uma explicação detalhada, como mostra a Figura 16.

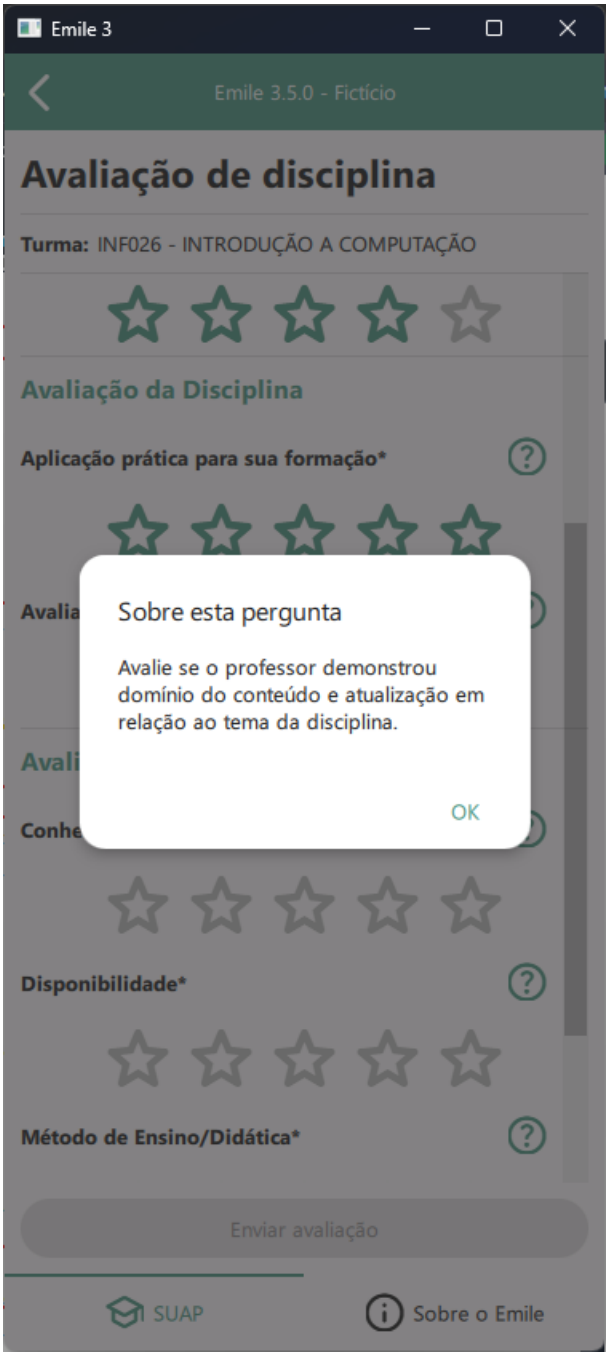

Figura 16: Formulário de avaliação com preenchimento pendente e destaque para ajuda

Com o preenchimento integral dos critérios, a função de envio é habilitada (Figura 17). O aluno dispõe ainda de um campo opcional para comentários textuais. Ressalta-se que este espaço é livre para elogios ou críticas construtivas, devendo-se observar as normas de conduta que vetam conteúdo ofensivo.

Figura 17: Formulário de avaliação devidamente preenchido

Ao solicitar o envio, o sistema exibe uma mensagem de confirmação final. Esta etapa é necessária para garantir a integridade dos dados, uma vez que não é possível editar o formulário após a submissão. Confirmada a ação, a avaliação é salva no banco de dados.

6.3.2 Consulta de Avaliações Realizadas

Após a submissão bem-sucedida da avaliação, o sistema atualiza a interface da disciplina. O botão, anteriormente rotulado como “Avaliar esta disciplina”, passará a exibir “Visualizar minha avaliação”, conforme destacado na Figura 18.

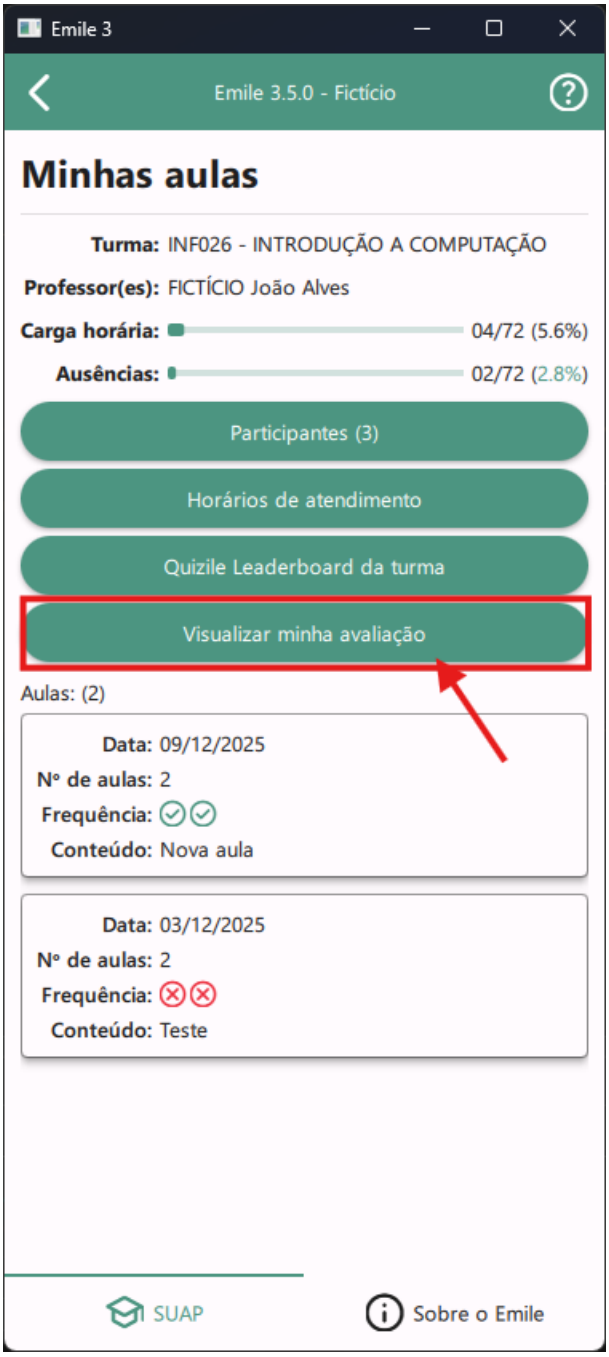

Figura 18: Acesso à visualização do formulário de avaliação enviado anteriormente

Ao selecionar esta opção, o usuário é redirecionado para a visualização do formulário em modo de leitura, contendo as respostas previamente enviadas (Figura 19), permitindo a conferência do feedback registrado.

Figura 19: Visualização do formulário de avaliação preenchido

6.3.3 Relatório de Avaliação de Disciplina

Para visualizar o relatório consolidado de uma turma, o usuário deve estar autenticado com um perfil de **professor**. A partir da tela de detalhes da disciplina desejada, deve-se selecionar a opção “Relatório de avaliação de disciplina”, conforme destacado na Figura 20.

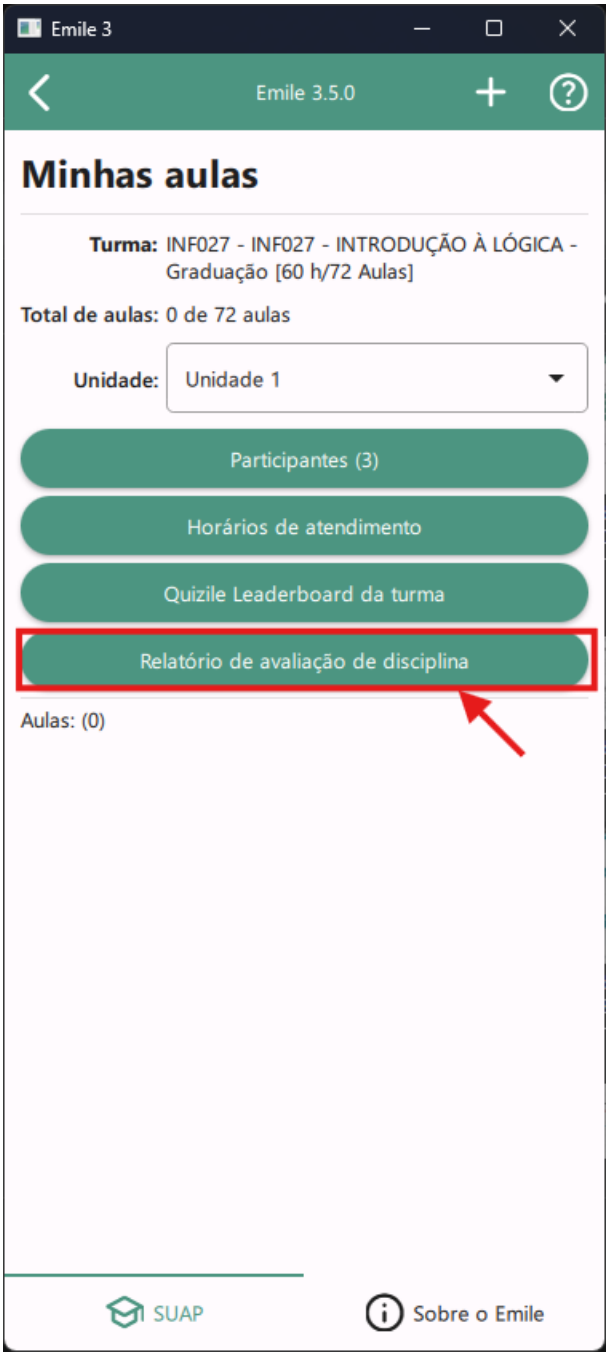

Figura 20: Acesso ao relatório de avaliações da disciplina (Visão do Professor)

Ao selecionar esta opção, o sistema exibe o painel de resultados contendo o total de alunos participantes e a média das respostas para cada pergunta. Adicionalmente, é apresentado um gráfico detalhado com a distribuição das avaliações por nível (estrelas), conforme a Figura 21. Ao final da listagem de critérios, encontra-se uma seção contendo todos os comentários qualitativos submetidos pelos alunos (Figura 22).

Figura 21: Relatório de avaliações: visualização dos resultados quantitativos

Figura 22: Relatório de avaliações: visualização dos comentários qualitativos

6.4 Pré-matrícula de Disciplinas

O módulo de pré-matrícula permite ao discente sinalizar interesse nas turmas ofertadas para o semestre subsequente. Este processo é fundamental para o planejamento da oferta de vagas e organização da grade horária. A seguir, detalha-se o fluxo de seleção de turmas e validação de horários.

6.4.1 Seleção de Turmas

Para realizar a pré-matrícula, o usuário deve estar autenticado com um perfil de **aluno**. No menu principal, deve-se selecionar a opção “Pré-matrícula”. O sistema apresentará a lista de componentes curriculares disponíveis, agrupados por período, conforme ilustrado na Figura 23.

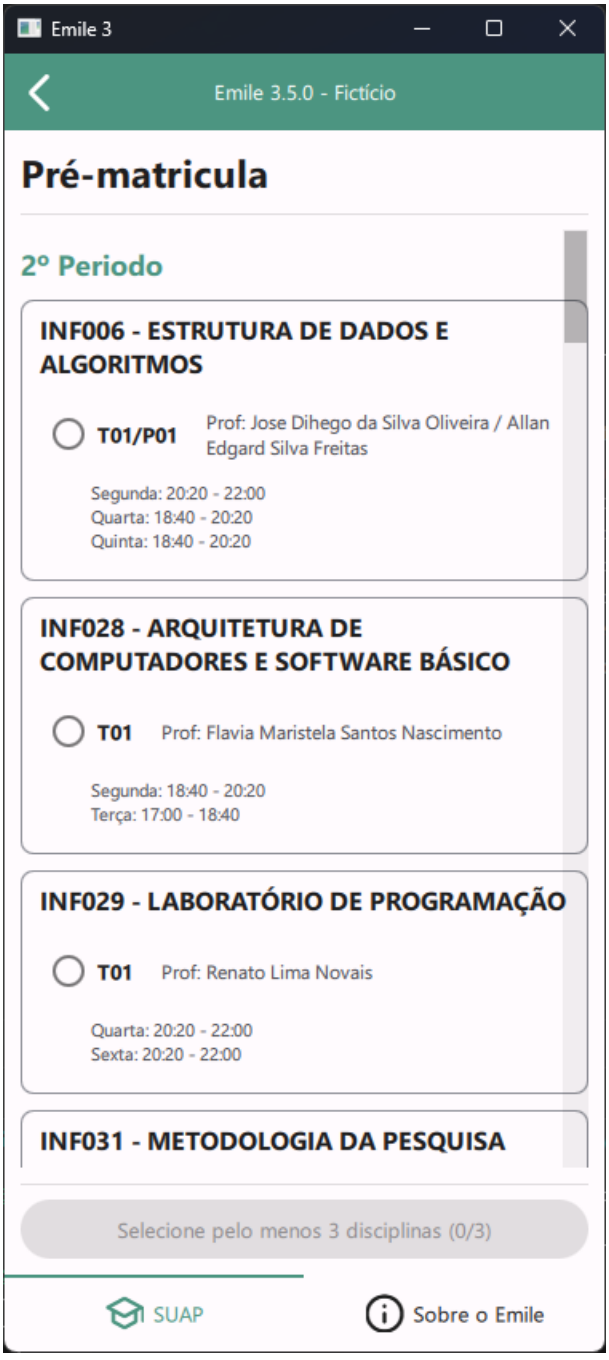

Figura 23: Tela de seleção de disciplinas agrupadas por período

Para cada disciplina, são exibidas as turmas disponíveis, indicando o código da turma, o docente responsável e os horários das aulas. O aluno deve selecionar a turma desejada clicando na opção correspondente (botão de seleção única). Caso deseje remover uma seleção

prévia, basta acionar a opção “Limpar” localizada ao lado do nome da disciplina, conforme a figura 24.

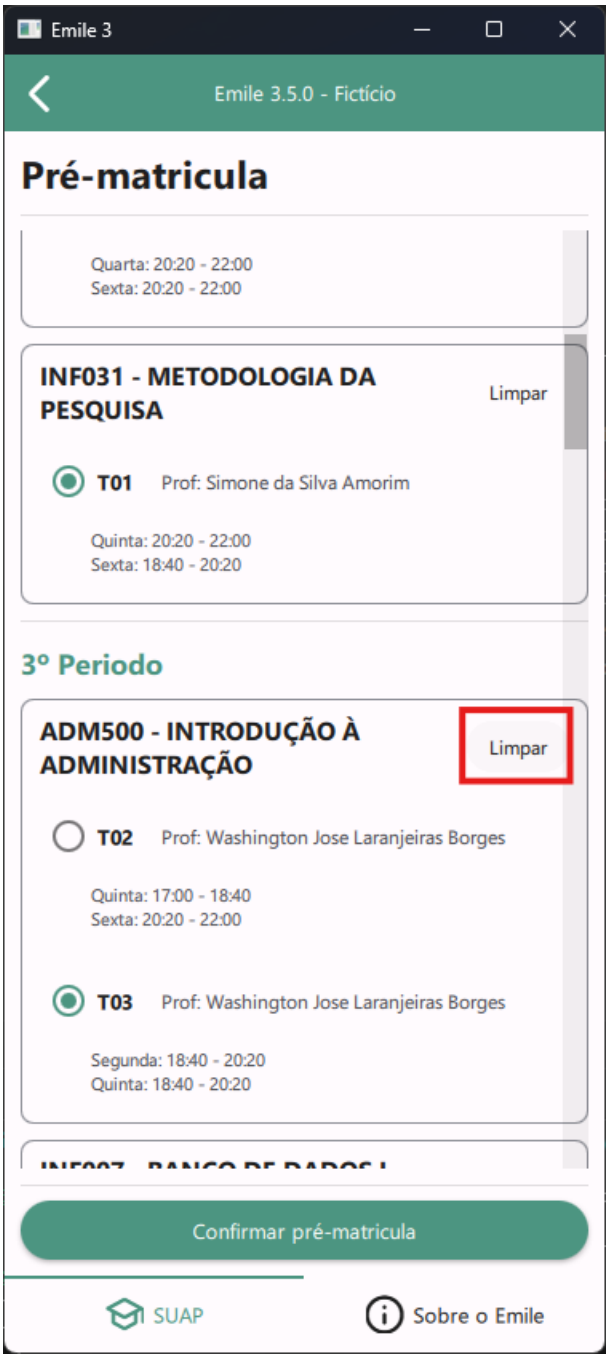

Figura 24: Tela de pré matricula com destaque no botão de limpar seleção

6.4.2 Validação e Conflito de Horários

Após selecionar as turmas de interesse, o aluno deve clicar no botão “Confirmar” ao final da página. Neste momento, o sistema executa uma validação automática para verificar a compatibilidade da grade horária escolhida.

Caso exista sobreposição de horários entre duas ou mais turmas selecionadas, o sistema

impedirá a conclusão do processo e exibirá um alerta de “Conflito de horários”, detalhando quais disciplinas e dias estão colidindo, conforme a Figura 25.

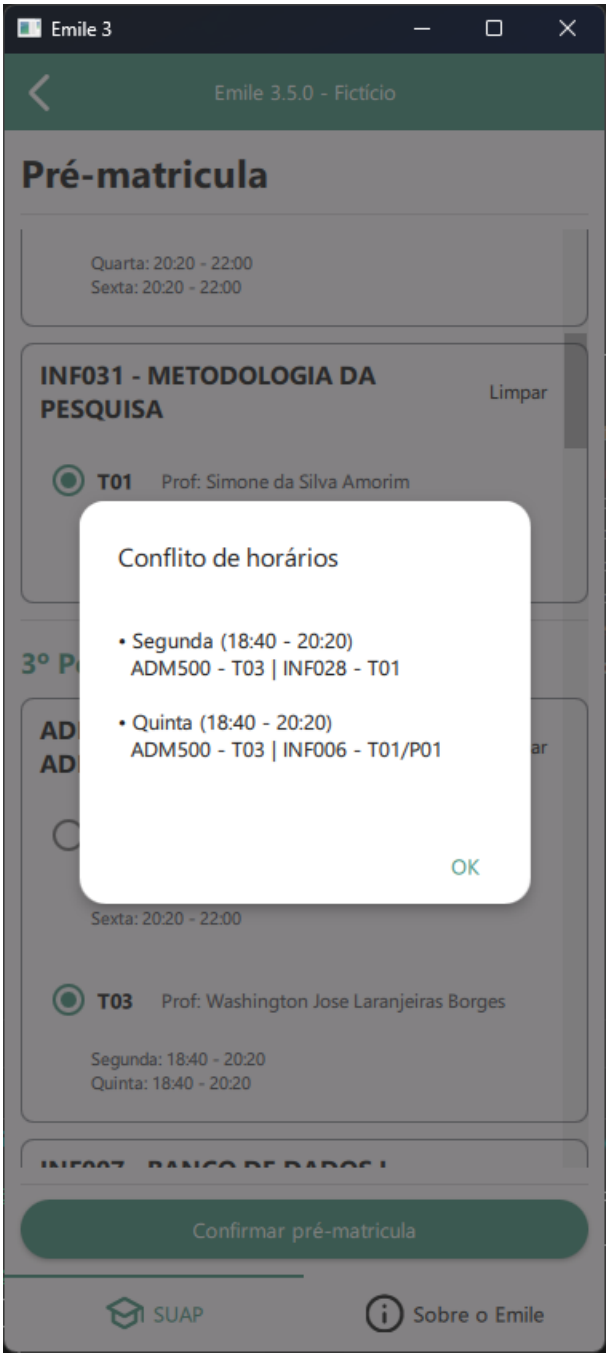

Figura 25: Alerta de conflito de horários impedindo a submissão

Para prosseguir, o usuário deve fechar o alerta e ajustar sua seleção, escolhendo outra turma para um dos componentes conflitantes ou removendo a seleção de uma das disciplinas.

6.4.3 Confirmação da Pré-matrícula

Uma vez resolvidas as pendências de horário, ao clicar novamente em “Confirmar”, o sistema solicitará uma confirmação final da ação através de uma caixa de diálogo (Figura 26).

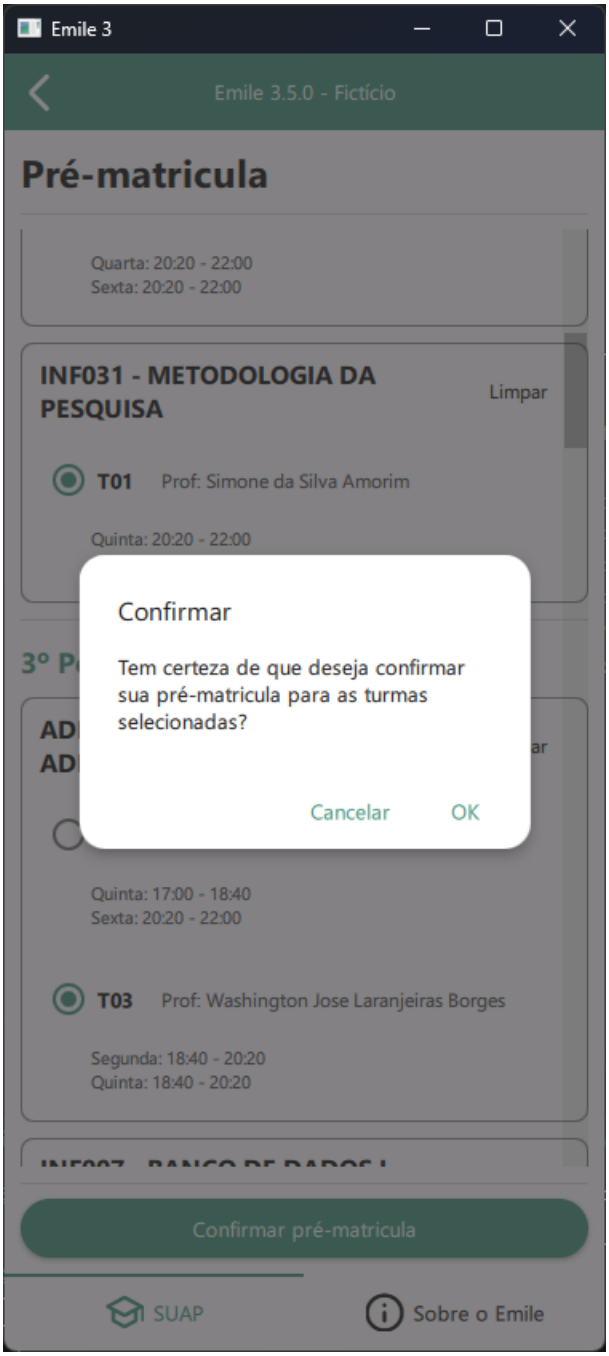

Figura 26: Caixa de diálogo para confirmação da pré-matrícula

Ao confirmar, os dados são processados e uma mensagem de sucesso é exibida, redirecionando o usuário para a tela inicial do sistema.

7 Conclusão

Este trabalho atingiu apresentou a implementação e integração de três novas funcionalidades no Emile - a versão móvel do SUAP, desenvolvido para dar apoio à gerência de atividades acadêmicas.

A implementação da funcionalidade de **Visualização Curricular** representou a integração das interfaces de “Componentes Curriculares”e “Matriz Curricular”. Isto permitiu centralizar o acesso à informação, possibilitando que o aluno compreenda sua trajetória acadêmica e identifique pendências curriculares com clareza e autonomia. A disponibilização de um **Sistema de Avaliação**, através do módulo de avaliação de disciplinas, estabeleceu um canal seguro e anônimo para *feedback* qualitativo e quantitativo. Como consequência, o Emile incorporou relatórios estatísticos importantes para a autoavaliação docente e melhoria do curso.

A funcionalidade da **Pré-matrícula** foi implementada para permitir aos estudantes manifestar interesse em disciplinas futuras, antes do início da matrícula web. A validação automática de conflitos de horários minimiza eventuais erros no processo de matrícula e ainda garante a integridade dos dados coletados, oferecendo insumos confiáveis para o planejamento da oferta de turmas. Por fim, este trabalho atingiu **Qualidade e Integração**, através da adoção do *C4 Model* para a arquitetura, aliada a uma esteira de Integração Contínua (CI) e ferramentas de análise estática (Ruff/Flake8).

Desta forma conseguimos incorporar novas funcionalidades à base de código do Emile de forma sustentável, segura e performática. Além disto, este trabalho contribuiu para melhorar a fragmentação de informações e suprir a carência de dados concretos para a tomada de decisão nas coordenações.

Importante destacar a necessidade de uma validação da ferramenta em larga escala, abrangendo todo o corpo discente Instituto Federal da Bahia, mais especificamene do Campus Salvador para viabilizar a coleta massiva de dados e testes de carga em produção.

Trabalhos Futuros

Visando a evolução contínua da solução, sugerem-se os seguintes desdobramentos:

- **Painel Administrativo Web:** O desenvolvimento de uma interface web (*Dashboard*) dedicada aos coordenadores de curso. Essa ferramenta permitiria a visualização dinâmica e a interação com os dados consolidados das avaliações e da pré-matrícula, facilitando a análise estratégica sem a necessidade de processamento manual de arquivos.
- **Integração Institucional Automatizada:** Recomenda-se a viabilização de uma integração direta (via API ou banco de dados) com o sistema acadêmico institucional (SUAP). Tal medida simplificaria drasticamente o processo de preparação dos dados para a funcionalidade de pré-matrícula, eliminando a necessidade de rotinas manuais de extração e tratamento de dados (ETL).

Por fim, o projeto entrega uma contribuição tecnológica funcional, de código aberto, que moderniza a experiência acadêmica no instituto e fornece alicerces sólidos para uma gestão baseada em dados.

Agradecimentos

Agradeço a todo o corpo docente do IFBA pela excelência no ensino e pela base sólida proporcionada. Em especial, registro minha admiração aos professores do curso de ADS, que são exímios no que fazem e verdadeiras referências profissionais que levarei para minha carreira.

Expresso minha profunda gratidão à minha orientadora, **Prof.ª Flávia Maristela**, cuja orientação foi o pilar fundamental para a realização deste trabalho. Sua disponibilidade, paciência e apoio constante durante todo esse árduo processo foram imprescindíveis; sem o seu incentivo, este TCC não teria sido possível.

Agradeço imensamente ao **Prof. Sandro Andrade**, não apenas por ter desenvolvido o aplicativo Emile — alicerce de toda esta pesquisa — mas por sua mentoria ativa. Seu apoio transcendeu a parte técnica, sendo crucial em cada etapa do desenvolvimento através de dicas valiosas e feedbacks precisos que elevaram a qualidade final desta solução.

Por fim, estendo meus agradecimentos à minha família, que sempre incentivou meu crescimento e foi o suporte crucial para mais esta conquista. Agradeço também aos meus amigos da turma 2023.1 de ADS, pelo companheirismo e apoio mútuo durante toda a nossa jornada acadêmica.

Referências

BROWN, S. *The C4 Model for Visualising Software Architecture*. [S.l.], 2025. Disponível em: ⟨https://c4model.com/introduction⟩.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado) — University of California, Irvine, 2000. Disponível em: ⟨https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf⟩.

FOWLER, M. *Continuous Integration*. 2024. Original article describing CI practices. Disponível em: ⟨https://martinfowler.com/articles/continuousIntegration.html⟩.

GITLAB INC. *The One DevOps Platform*. 2025. Disponível em: ⟨https://about.gitlab.com/⟩.

ISO/IEC. *ISO/IEC 14882:2020 Programming languages — C++*. 2020. Disponível em: ⟨https://www.iso.org/standard/79358.html⟩.

MARIADB FOUNDATION. *MariaDB Server Documentation*. [S.l.], 2025. Disponível em: ⟨https://mariadb.com/kb/en/documentation/⟩.

MARSH, C. *Ruff: An extremely fast Python linter and code formatter*. 2025. Disponível em: ⟨https://docs.astral.sh/ruff/⟩.

PYCQA. *Flake8: Your Tool For Style Guide Enforcement*. 2016. Disponível em: ⟨https://flake8.pycqa.org/en/latest/⟩.

RAMíREZ, S. *FastAPI: High performance, easy to learn, fast to code, ready for production*. 2025. Disponível em: ⟨https://fastapi.tiangolo.com/⟩.

ROSSUM, G. V.; DRAKE, F. L. *The Python Language Reference*. [S.l.], 2025. Disponível em: ⟨https://docs.python.org/3/reference/⟩.

ROSSUM, G. V.; WARSAW, B.; COGHLAN, N. *PEP 8 – Style Guide for Python Code*. 2001. Disponível em: ⟨https://peps.python.org/pep-0008/⟩.

SOFTWARE FREEDOM CONSERVANCY. *Git: Fast Version Control System*. 2025. Disponível em: ⟨https://git-scm.com/⟩.

SOMMERVILLE, I. *Software Engineering*. 10. ed. London: Pearson, 2015.

THE QT COMPANY. *Qt Documentation: Qt QML*. 2025. Framework de desenvolvimento multiplataforma. Disponível em: ⟨https://doc.qt.io/qt-6/qtqml-index.html⟩.