

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
DEPARTAMENTO DE CIÊNCIA DA INFORMAÇÃO
CURSO CIÊNCIA DA INFORMAÇÃO

Luiz Gustavo Erthal

Sistema de Recomendação para Matrícula Baseado em Satisfação de Restrições para o Curso
de Ciência da Informação da UFSC

Florianópolis
2026

Luiz Gustavo Erthal

Sistema de Recomendação para Matrícula Baseado em Satisfação de Restrições para o Curso
de Ciência da Informação da UFSC

Trabalho Conclusão do Curso de Graduação em Ciência
da Informação do Centro de Ciências da Educação da
Universidade Federal de Santa Catarina como requisito
para a obtenção do título de Bacharel em Ciência da
Informação
Orientador: Prof. Dr. Ricardo Alexandre Reinaldo de
Moraes

Florianópolis

2026

Ficha de identificação da obra

Erthal, Luiz Gustavo

Sistema de Recomendação para Matrícula Baseado em Satisfação de Restrições para o Curso de Ciência da Informação da UFSC / Luiz Gustavo Erthal ; orientador, Ricardo Alexandre Reinaldo de Moraes, 2026.

85 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro de Ciências da Educação, Graduação em Ciência da Informação, Florianópolis, 2026.

Inclui referências.

1. Ciência da Informação. 2. sistemas de recomendação. 3. problemas de satisfação de restrição. I. Moraes, Ricardo Alexandre Reinaldo de . II. Universidade Federal de Santa Catarina. Graduação em Ciência da Informação. III. Título.

Luiz Gustavo Erthal

**Sistema de Recomendação para Matrícula Baseado em Satisfação de Restrições para o
Curso de Ciência da Informação da UFSC**

Florianópolis, 09 de Julho de 2026.

O presente Trabalho de Conclusão de Curso foi avaliado e aprovado pela banca examinadora composta pelos seguintes membros:

Prof. Ricardo Alexandre Reinaldo de Moraes, Dr.

Documento assinado digitalmente
Douglas Dyllon Jeronimo de Macedo
Data: 27/07/2026 19:54:18-0300
CPF: ***.987.372-**
Verifique as assinaturas em <https://v.ufsc.br>

Prof. Douglas Dyllon Jeronimo de Macedo, Dr.

Universidade Federal de Santa Catarina

Prof. Márcio Matias, Dr.

Universidade Federal de Santa Catarina

Documento assinado digitalmente
Marcio Matias
Data: 24/07/2026 12:02:19-0300
CPF: ***.755.779-**
Verifique as assinaturas em <https://v.ufsc.br>

Certifico que esta é a **versão original e final** do Trabalho de Conclusão de Curso que foi julgado adequado para obtenção do título de Bacharel em Ciência da Informação por mim e pelos demais membros da banca examinadora.

Documento assinado digitalmente
Ricardo Alexandre Reinaldo de Moraes
Data: 24/07/2026 10:33:26-0300
CPF: ***.812.939-**
Verifique as assinaturas em <https://v.ufsc.br>

Prof. Ricardo Alexandre Reinaldo de Moraes, Dr.

Orientador

Florianópolis, 2026

RESUMO

Este trabalho apresenta o desenvolvimento de um modelo computacional para recomendação automática de grades horárias para estudantes do curso de Ciência da Informação da Universidade Federal de Santa Catarina. O problema abordado surge da crescente complexidade na organização das disciplinas ao longo da trajetória acadêmica, especialmente em função de reprovações, atividades extracurriculares e restrições individuais dos estudantes. O sistema proposto utiliza conceitos de sistemas de recomendação aliados ao paradigma de Problemas de Satisfação de Restrições, permitindo representar formalmente as variáveis, domínios e restrições envolvidas na construção das grades horárias. Como método, adota-se uma abordagem aplicada, com desenvolvimento de software baseado em processo ágil utilizando Kanban, contemplando etapas de levantamento de requisitos, modelagem, implementação e validação. O modelo considera dados institucionais, como horários e pré-requisitos das disciplinas, bem como informações individuais dos estudantes, incluindo histórico acadêmico, preferências e limitações. A partir desses dados, o sistema gera sugestões de grades que respeitam restrições como conflitos de horário, carga horária e requisitos curriculares. Como resultado esperado, espera-se obter-se uma ferramenta capaz de auxiliar na tomada de decisão dos estudantes, promovendo maior eficiência no planejamento acadêmico e contribuindo para a redução do tempo de conclusão do curso.

Palavras-chave: sistemas de recomendação; Problemas de Satisfação de Restrição; grade horária; planejamento acadêmico.

ABSTRACT

This work presents the development of a computational model for the automatic recommendation of course schedules for students of the Information Science program at the Federal University of Santa Catarina. The addressed problem arises from the increasing complexity of organizing academic schedules throughout the students' trajectory, especially due to course failures, extracurricular activities, and individual constraints. The proposed system combines recommender system concepts with the Constraint Satisfaction Problem paradigm, allowing the formal representation of variables, domains, and constraints involved in schedule construction. The method follows an applied approach, involving software development based on an agile process using Kanban, including requirements analysis, modeling, implementation, and validation stages. The model considers institutional data, such as course schedules and prerequisites, as well as individual student information, including academic history, preferences, and constraints. Based on these inputs, the system generates schedule suggestions that satisfy constraints such as time conflicts, workload limits, and curricular requirements. As an expected result, the system provides a tool to support student decision-making, improving academic planning efficiency and contributing to reducing the time required to complete the degree.

Keywords: recommender systems; constraint satisfaction problems; course scheduling; academic planning; systems.

LISTA DE FIGURAS

Figura 1 - Diagrama esquemático do banco de dados do sistema	25
Figura 2 - Diagrama de arquitetura do sistema	28
Figura 3 - Grade curricular recomendada renderizada	32

LISTA DE TABELAS

Tabela 1 - Escore por participante	34
Tabela 2 - Hit Ratio por participante	36

LISTA DE ABREVIATURAS E SIGLAS

CSP *Constraints Satisfaction Problems* (Problemas de Satisfação de Restrição)

SUMÁRIO

1 INTRODUÇÃO	12
1.1 OBJETIVOS	13
1.2 OBJETIVO GERAL	13
1.3 OBJETIVOS ESPECÍFICOS	13
1.4 JUSTIFICATIVA	13
1.5 ORGANIZAÇÃO DO TRABALHO	14
2 FUNDAMENTAÇÃO TEÓRICA	15
2.2 MÉTRICAS DE AVALIAÇÃO DE SISTEMAS DE RECOMENDAÇÃO	17
2.3 ESCALA DE USABILIDADE DE SISTEMAS (SYSTEM USABILITY SCALE - SUS)	17
2.3 PROBLEMAS DE SATISFAÇÃO DE RESTRIÇÃO	18
2.4 O PROBLEMA DE ELABORAÇÃO DE GRADES HORÁRIAS	20
2.5 CONSIDERAÇÕES FINAIS DA FUNDAMENTAÇÃO TEÓRICA	21
3 PROCESSO DE DESENVOLVIMENTO DE SISTEMAS	23
3.1 ABORDAGEM METODOLÓGICA	23
3.2 PROCESSO DE DESENVOLVIMENTO DO SISTEMA	23
3.3 LEVANTAMENTO DE REQUISITOS	24
3.3.1 Requisitos funcionais	25
3.3.2 Requisitos não funcionais	25
3.4 MODELAGEM DO PROBLEMA COMO CSP	25
3.6 LIMITAÇÕES DO SISTEMA	26
3.7 DIAGRAMA ESQUEMÁTICO DE BANCO DE DADOS	27
3.8 CONSIDERAÇÕES FINAIS DO PROCESSO DE DESENVOLVIMENTO DE SISTEMAS	27
4 ARQUITETURA E IMPLEMENTAÇÃO DO SISTEMA	29
4.1 VISÃO GERAL DA ARQUITETURA	29
4.2 MÓDULO DE EXTRAÇÃO DE DADOS (PARSING)	30
4.3 CAMADA DE PERSISTÊNCIA E INTEGRAÇÃO COM BANCO DE DADOS	31
4.4 MOTOR DE RECOMENDAÇÃO E RESOLUÇÃO DE RESTRIÇÕES (CSP)	31
4.4.1 Modelagem e Processamento de Restrições Rígidas	31
4.4.2 Modelagem de Restrições Flexíveis e Algoritmo de Seleção	32
4.5 INTERFACE DO USUÁRIO E VISUALIZAÇÃO DA GRADE	33
4.6 CONSIDERAÇÕES FINAIS DA ARQUITETURA E IMPLEMENTAÇÃO DO SISTEMA	34
5 TESTES E VALIDAÇÃO	35
5.1 CÁLCULO DA ESCALA DE USABILIDADE DE SISTEMAS	35
5.2 AVALIAÇÃO DA QUALIDADE DAS RECOMENDAÇÕES POR MEIO DO HIT RATIO	36

5.3 DISCUSSÃO DOS RESULTADOS	37
6 CONCLUSÃO	39
6.1 SUGESTÕES PARA TRABALHOS FUTUROS	40
REFERÊNCIAS	41
APÊNDICE A – Como utilizar o sistema	43
APÊNDICE B – Código fonte historico_pdf_parser.py	46
APÊNDICE C – Código fonte pdf_parser.py	50
APÊNDICE D – Código fonte supabase_conn.py	61
APÊNDICE E – Código fonte rec_sys.py	63
APÊNDICE F – Código fonte disciplinas_handler.py	70
APÊNDICE G – Código fonte timetable_handler.py	74
APÊNDICE H – Código fonte app.py	78

1 INTRODUÇÃO

Ao longo da trajetória acadêmica, a organização da grade horária dos estudantes de Ciência da Informação tende a se tornar mais complexa. Enquanto alguns estudantes conseguem seguir a sequência de disciplinas proposta pelo curso, outros enfrentam dificuldades para manter esse fluxo.

Fatores como reprovações, participação em atividades extracurriculares (como projetos de pesquisa e extensão), estágios e demandas pessoais contribuem para a necessidade de uma organização mais flexível da grade curricular. Como consequência, muitos estudantes constroem trajetórias acadêmicas individualizadas, exigindo maior esforço no planejamento de suas disciplinas a cada semestre.

No cenário acadêmico, alguns trabalhos relacionados fizeram o uso de sistemas de recomendação para apoiar o estudante tem sido explorado sob diversas perspectivas. Queiroz (2023) propôs um sistema para auxiliar na escolha de disciplinas optativas para o curso de Sistemas de Informação, utilizando o conceito de "trilhas de carreira" para direcionar o aluno. Outras abordagens focam em elementos periféricos ao currículo, como o modelo de Marques (2022), que desenvolveu um sistema híbrido para recomendação de atividades complementares e capacitação profissional.

Há também aplicações voltadas ao desempenho em Ambientes Virtuais de Aprendizagem (AVA), como o trabalho de Damasceno (2024), que utilizou recomendações para filtrar recursos digitais educativos, e soluções para etapas específicas da graduação, como o modelo de Nascimento, Gondim e Costa Filho (2023) para a recomendação de orientadores de TCC.

Entretanto, observa-se que a maioria das soluções existentes foca em recomendações de itens isolados ou atividades complementares. Diante desse cenário, este trabalho diferencia-se ao propor o desenvolvimento de um sistema de recomendação focado na montagem da grade horária integral. Através do paradigma de Satisfação de Restrições (CSP), o sistema busca auxiliar os estudantes na tomada de decisão durante o período de matrícula, conciliando obrigаторiedades curriculares, disciplinas optativas e preferências individuais em uma solução otimizada para o planejamento acadêmico.

1.1 OBJETIVOS

Esta seção apresenta os objetivos que orientam o desenvolvimento deste trabalho, delimitando seu escopo e direcionando as etapas de modelagem e implementação da solução proposta. A seguir, são descritos o objetivo geral e os objetivos específicos do trabalho.

1.2 OBJETIVO GERAL

Desenvolver um aplicativo baseado em um modelo computacional de recomendação capaz de sugerir disciplinas para matrícula a estudantes do curso de Ciência da Informação da Universidade Federal de Santa Catarina, considerando o histórico acadêmico, os pré-requisitos curriculares, a oferta de disciplinas e as restrições de horários informadas pelo usuário.

1.3 OBJETIVOS ESPECÍFICOS

- a) Modelar o problema de geração de grades horárias com base em variáveis, domínios e restrições;
- b) Implementar o modelo computacional para recomendação de grades horárias personalizadas;
- c) Testar e validar o aplicativo proposto;
- d) Documentar o processo de desenvolvimento e de uso do sistema.

1.4 JUSTIFICATIVA

A realização deste trabalho justifica-se pela necessidade de apoiar estudantes do curso de Ciência da Informação na organização de suas grades horárias, especialmente diante de diferentes trajetórias acadêmicas, restrições pessoais e ofertas semestrais de disciplinas. A proposta busca contribuir com uma solução prática que auxilie no processo de matrícula, reduzindo dificuldades relacionadas ao planejamento acadêmico.

Além disso, o desenvolvimento deste sistema está alinhado ao interesse pessoal do autor pela área de desenvolvimento de software, bancos de dados e organização de infraestrutura computacional. Dessa forma, o trabalho permite aplicar conhecimentos adquiridos ao longo da graduação em uma solução voltada ao próprio curso, retornando parte da formação recebida por meio de uma ferramenta com potencial de uso acadêmico.

1.5 ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em seis seções. A seção 1 apresenta a introdução, a contextualização do problema, os objetivos do trabalho, a justificativa e a organização deste documento.

A seção 2 reúne a fundamentação teórica, abordando os conceitos de sistemas de recomendação, Problemas de Satisfação de Restrições (*Constraint Satisfaction Problems - CSP*), elaboração de grades horárias acadêmicas e as métricas utilizadas para avaliação do sistema proposto.

A seção 3 descreve os procedimentos metodológicos adotados, incluindo a abordagem do trabalho, o processo de desenvolvimento do sistema, o levantamento de requisitos, a modelagem do problema como um CSP, as limitações da solução e a estrutura do banco de dados.

A seção 4 apresenta a arquitetura e a implementação do sistema, detalhando seus principais módulos, o funcionamento do motor de recomendação, a persistência dos dados e a interface desenvolvida para interação com o usuário.

A seção 5 descreve os testes realizados e a validação da solução, contemplando a avaliação da usabilidade por meio da Escala de Usabilidade de Sistemas (*System Usability Scale - SUS*), a análise da qualidade das recomendações utilizando a métrica *Hit Ratio* e a discussão dos resultados obtidos.

Por fim, a seção 6 apresenta as conclusões do trabalho, destacando as principais contribuições do trabalho, suas limitações e sugestões para trabalhos futuros. Ao final do documento são apresentadas as referências utilizadas e os apêndices, contendo instruções de uso do sistema e os principais códigos-fonte desenvolvidos.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, apresenta-se o conceito de sistemas de recomendação e sistemas de satisfação de restrição.

2.1 SISTEMAS DE RECOMENDAÇÃO

Um sistema de recomendação representa uma evolução das técnicas tradicionais de recuperação e filtragem de informações. Seu objetivo primordial é selecionar e apresentar itens personalizados que possuam alta probabilidade de interesse para um usuário específico, mitigando o fenômeno da sobrecarga de informação (Wen, 2008). Sob a ótica da Ciência da Informação, essas ferramentas atuam como mediadores algorítmicos que transformam dados de comportamento em conhecimento útil para a tomada de decisão.

Segundo a taxonomia clássica proposta por Adomavicius e Tuzhilin (2005), os sistemas de recomendação operam fundamentalmente a partir de três abordagens estratégicas:

- a) Filtragem Baseada em Conteúdo (*Content-Based Filtering*): sugere itens que compartilham características ou metadados semelhantes com aqueles pelos quais o usuário demonstrou interesse prévio;
- b) Filtragem Colaborativa (*Collaborative Filtering*): fundamenta-se na premissa de que usuários que compartilharam preferências semelhantes no passado tenderão a concordar no futuro. O sistema mapeia vizinhanças de usuários (*peers*) para recomendar itens consumidos positivamente pelo grupo semelhante;
- c) Abordagens Híbridas (*Hybrid Systems*): combinam as duas técnicas precedentes visando potencializar as virtudes de cada uma e mitigar limitações intrínsecas, tais como o problema de início a frio (*cold start*), que ocorre quando o volume de dados sobre novos usuários ou itens é insuficiente para gerar inferências precisas.

Do ponto de vista formal e algorítmico, o problema central de recomendação pode ser modelado a partir de uma estrutura de dados de preferências. Conforme exposto por Aggarwal (2016), a modelagem matemática desse cenário assume a existência de uma matriz de utilidade incompleta de dimensões $\mathbf{m} \times \mathbf{n}$, onde $\mathbf{m}$ representa o conjunto de usuários e $\mathbf{n}$ delimita o universo de itens disponíveis. A partir dessa estrutura, o autor divide a resolução do problema de recomendação em duas formulações principais:

- a) Versão de Predição do Problema (*Prediction Version*): foca em estimar numericamente o valor de avaliação (*rating*) para uma combinação específica entre um usuário e um item ainda não interagido. Utilizando os dados históricos observados como conjunto de treinamento, o algoritmo de aprendizado realiza a tarefa de complementação de matriz (*matrix completion problem*) para inferir as células vazias da matriz de utilidade;
- b) Versão de Ranqueamento do Problema (*Ranking Version*): nesta abordagem, o foco deixa de ser o valor absoluto da avaliação e passa a ser a relevância relativa dos itens. O objetivo comercial ou prático passa a ser a determinação dos *top-k* itens mais relevantes para um usuário específico.

Aggarwal (2016) ressalta que a formulação baseada em predição é mais genérica, uma vez que, ao estimar todas as avaliações ausentes, basta ordenar os resultados para obter a solução do problema de ranqueamento *top-k*. No entanto, em aplicações práticas, a precisão absoluta da nota predita é secundária se comparada à ordenação correta dos itens que serão visualizados pelo usuário.

No contexto do planejamento acadêmico e da montagem de grades horárias, contudo, as preferências subjetivas e o "gosto" do estudante constituem apenas uma parte das variáveis do problema. Diferente de domínios comerciais como o de entretenimento ou e-commerce, onde o consumo de um item inadequado gera apenas insatisfação, no domínio acadêmico o sistema deve garantir a consistência e a validade lógica das sugestões geradas - afinal, há um projeto pedagógico a ser seguido.

Para atender a esse requisito, as abordagens puramente estatísticas ou colaborativas revelam-se insuficientes, tornando inevitável a utilização para o recurso aos Sistemas Baseados em Conhecimento (*Knowledge-Based Recommender Systems*). Reforçando essa perspectiva, Uta et al. (2024, p. 1, tradução livre) apontam que:

Em contraste com as abordagens de recomendação tradicionais de filtragem colaborativa e filtragem baseada em conteúdo, os recomendadores baseados em conhecimento exploram o conhecimento semântico das preferências do usuário, o conhecimento dos itens e o conhecimento de recomendação para identificar itens relevantes para o usuário [...]

Dessa forma, conforme explicitado por Burke, Felfernig e Göker (2011), esse modelo de recomendação se distancia da necessidade de históricos massivos de avaliações anteriores (comuns na filtragem colaborativa), estruturando-se a partir de fontes explícitas de

informação: o "conhecimento de conteúdo" sobre os itens e o "conhecimento individual" sobre as demandas específicas do usuário. No escopo deste trabalho, esse corpo de conhecimento é composto pelas diretrizes do Projeto Pedagógico do Curso (PPC) - o que engloba a árvore de pré-requisitos e equivalências disciplinares (conhecimento de conteúdo) - associadas às restrições de horários e à disponibilidade física e temporal da instituição e do próprio discente (conhecimento individual).

2.2 MÉTRICAS DE AVALIAÇÃO DE SISTEMAS DE RECOMENDAÇÃO

A avaliação de sistemas de recomendação busca mensurar a capacidade do modelo em sugerir itens relevantes aos usuários. Em problemas de recomendação top-k, nos quais o sistema apresenta apenas os itens considerados mais adequados, métricas de acurácia e qualidade de ordenação são amplamente utilizadas (Aggarwal, 2016).

Neste trabalho, será empregada a métrica Hit Ratio, proposta por Alsini, Huynh e Datta (2020), por sua adequação a cenários em que a quantidade de itens recomendados e de itens relevantes pode variar entre diferentes usuários. A métrica é definida como:

$$Hit\ Ratio = \frac{m}{\min(nr, ng)}$$

em que **m** representa o número de disciplinas corretamente recomendadas, **nR** o número de disciplinas recomendadas e **nG** o número de disciplinas efetivamente selecionadas pelo estudante. Valores próximos de 1 indicam maior aderência entre as recomendações e as escolhas reais do usuário.

2.3 ESCALA DE USABILIDADE DE SISTEMAS (SYSTEM USABILITY SCALE - SUS)

Além da avaliação da qualidade das recomendações geradas, é importante mensurar a percepção dos usuários quanto à facilidade de uso do sistema. Para esse propósito, uma das métricas mais utilizadas na área de Interação Humano-Computador é a Escala de Usabilidade de Sistemas (System Usability Scale – SUS), proposta por Brooke (1996).

O SUS consiste em um questionário padronizado composto por dez afirmações respondidas em uma escala Likert de cinco pontos, variando de "discordo totalmente" a "concordo totalmente". As questões alternam entre afirmações positivas e negativas sobre o

sistema avaliado, com o objetivo de reduzir vieses de resposta e proporcionar uma avaliação mais confiável da usabilidade.

A pontuação do SUS varia de 0 a 100 pontos. Para o cálculo do escore, as respostas das questões ímpares são ajustadas subtraindo-se 1 do valor selecionado pelo participante, enquanto nas questões pares o valor é obtido pela diferença entre 5 e a resposta fornecida. A soma dos valores obtidos é multiplicada por 2,5, resultando na pontuação final de usabilidade.

Segundo Sauro e Lewis (2016), um escore SUS de 68 representa aproximadamente a média observada em sistemas avaliados, sendo valores superiores indicativos de melhor usabilidade percebida. Escores acima de 78 pontos costumam ser associados a sistemas com excelente experiência de uso, enquanto valores inferiores a 52 pontos indicam problemas significativos de usabilidade.

Neste trabalho, o SUS será utilizado como instrumento complementar de avaliação, permitindo verificar a percepção dos estudantes quanto à facilidade de utilização do sistema de recomendação de disciplinas. Dessa forma, além da análise da aderência das recomendações por meio do *Hit Ratio*, será possível avaliar a aceitação da solução sob a perspectiva da experiência do usuário.

2.3 PROBLEMAS DE SATISFAÇÃO DE RESTRIÇÃO

Os Problemas de Satisfação de Restrição (*Constraint Satisfaction Problems - CSPs*) representam um paradigma consolidado no campo da Inteligência Artificial para a resolução de problemas combinatórios. Diferente dos métodos de busca tradicionais em espaços de estados, onde os estados são tratados como estruturas atômicas ou "caixas pretas" indivisíveis, a abordagem baseada em CSP decompõe o estado por meio de uma representação fatorada, expressa por um conjunto de variáveis que recebem valores específicos até que um objetivo seja alcançado (Russell; Norvig, 2021).

Sob a ótica de sistemas de recomendação aplicados a domínios complexos, Burke, Felfernig e Göker (2011) destacam que a incorporação de técnicas de satisfação de restrições revolucionou o desenvolvimento de recomendadores baseados em conhecimento. Isso ocorre porque o modelo permite cruzar os requisitos do usuário com as regras rígidas do domínio sem a necessidade de históricos prévios de consumo. Formalmente, Russell e Norvig (2021) definem um CSP como uma estrutura matemática composta por uma tupla tridimensional $\langle V, D, C \rangle$:

- a) variáveis (V): consiste no conjunto de elementos que demandam uma atribuição formal de valores, denotado por $V=\{V1,V2,...,Vn\}$. No contexto do planejamento acadêmico, as variáveis correspondem às disciplinas que o estudante deseja ou precisa cursar no semestre letivo corrente;
- b) domínios (D): delimita o conjunto de valores permitidos para cada variável individual, sendo representado por $D=\{D1,D2,...,Dn\}$. Para a montagem da grade horária, o domínio de uma disciplina V_i engloba as turmas ofertadas, seus respectivos professores, códigos e horários de funcionamento disponibilizados pela instituição;
- c) restrições (C): especifica o conjunto de regras rígidas que limitam e coordenam as combinações de valores permitidos entre as variáveis, definido por $C=\{C1,C2,...,Cm\}$. Cada restrição consiste em um par $\langle \text{escopo}, \text{relação} \rangle$, onde o escopo é uma tupla de variáveis envolvidas e a relação determina os valores que elas podem assumir simultaneamente. No cenário universitário, isso se traduz na impossibilidade física de choques de horário (restrições de exclusão mútua) e em respeito aos pré-requisitos curriculares obrigatórios.

Conforme a taxonomia estabelecida por Uta et al. (2024), a recomendação baseada em restrições atua mapeando explicitamente esses três componentes para filtrar e construir soluções sob medida. A resolução de um modelo CSP consiste em obter uma atribuição completa - na qual todas as variáveis possuem um valor associado - e consistente, garantindo que nenhuma das restrições impostas pelo escopo do problema seja violada (Russell; Norvig, 2021).

Contudo, problemas voltados ao agendamento de horários (timetabling acadêmico) possuem natureza combinatória complexa de alta dimensão. À medida que o número de variáveis e tamanhos de domínios cresce, o espaço de busca sofre uma expansão exponencial. Para contornar essa barreira e localizar soluções viáveis de forma computacionalmente eficiente, recorre-se a algoritmos de busca com retrocesso (backtracking search), que operam de forma sistemática realizando atribuições parciais e retrocedendo na árvore de decisão ao detectar caminhos inconsistentes (Russell; Norvig, 2021).

Para otimizar esse processo e reduzir precocemente as dimensões do espaço de estados antes e durante a execução da busca, utilizam-se técnicas de propagação de restrições. No sistema de recomendação de matrículas, essas abordagens atuam reduzindo o impacto de uma

escolha: ao selecionar uma disciplina matutina específica, o algoritmo propaga essa restrição e elimina automaticamente do domínio das demais variáveis todas as turmas que gerem conflito de horário, eliminando ramos inteiros do espaço amostral inútil e acelerando a entrega de uma grade horária otimizada e personalizada ao discente.

2.4 O PROBLEMA DE ELABORAÇÃO DE GRADES HORÁRIAS

A elaboração de grades horárias acadêmicas, reconhecida internacionalmente na literatura científica pelo termo *timetabling*, constitui um desafio logístico e administrativo crítico para as instituições de ensino. O *timetabling* transcende a mera organização de horários, configurando-se como um problema de otimização combinatória de alta complexidade. De modo abrangente, Burke e Petrovic (2002) definem essa atividade como a tarefa de alocar um conjunto de eventos (como disciplinas e turmas) a um número limitado de períodos de tempo e espaços físicos, assegurando a conformidade com um conjunto predefinido de regras e objetivos institucionais.

Complementando essa visão estrutural, Wren (1996, p. 53, tradução livre) conceitua o problema sob a ótica da gestão informacional de recursos:

[...] timetabling é a alocação, sujeita a restrições, de determinados recursos a objetos colocados em matrizes de espaço-tempo, de modo a satisfazer, tão proximamente quanto possível, um conjunto de objetivos desejáveis filtrados por meio de um conjunto de restrições.

No ambiente universitário, essa distribuição ganha contornos ainda mais complexos devido à fragmentação curricular e à flexibilidade dos fluxos de matrícula dos estudantes. Conforme apontam Babaei, Karimpour e Hadidi (2015), o chamado Problema de Grades Horárias de Cursos Universitários (University Course Timetabling Problem - UCTP) diferencia-se de agendas escolares rígidas porque precisa lidar com uma matriz de restrições heterogêneas, as quais a literatura classicamente divide em duas categorias fundamentais: restrições rígidas (*hard constraints*) e restrições flexíveis (*soft constraints*).

As restrições rígidas (*hard constraints*) compreendem as condições físicas, lógicas e regulatórias que devem ser obrigatoriamente satisfeitas para que o cronograma resultante possua validade operacional básica (Burke; Petrovic, 2002). No escopo do planejamento de matrículas, essas restrições funcionam como os filtros absolutos de consistência local e de arcos mapeados no modelo CSP. São exemplos mandatórios de *hard constraints*: a

impossibilidade de o discente frequentar duas turmas simultaneamente (choques de horário), o respeito à árvore de pré-requisitos estabelecida no Projeto Pedagógico do Curso (PPC) e a adequação aos bloqueios de turnos ou dias da semana informados pelo próprio usuário. Se uma única restrição rígida for violada, a solução torna-se computacionalmente inconsistente, inviabilizando a grade horária.

Por outro lado, as restrições flexíveis (*soft constraints*) representam preferências, conveniências e critérios de qualidade que, embora desejáveis, não impedem a validação técnica da solução caso venham a ser infringidos (Wren, 1996). Em um sistema de recomendação acadêmica, as *soft constraints* definem o nível de personalização e otimização da grade sugerida ao estudante. Elas englobam variáveis subjetivas como a afinidade por determinadas áreas de conhecimento (por exemplo, a predileção por optativas voltadas à Tecnologia da Informação ou à Gestão da Informação), o balanceamento da carga horária semestral para evitar sobrecargas cognitivas e o respeito ao teto máximo de horas de disciplinas optativas solicitadas.

Em termos práticos, enquanto o cumprimento das restrições rígidas delimita o espaço de soluções logicamente viáveis através do algoritmo de busca por retrocesso (*backtracking*), a avaliação das restrições flexíveis orienta as heurísticas de ordenação e pontuação (priorização) do recomendador. Portanto, a qualidade e a utilidade da solução gerada pelo sistema estão diretamente atreladas à sua capacidade de blindar as *hard constraints* institucionais ao mesmo tempo em que maximiza a satisfação das *soft constraints* individuais do discente, resultando em uma trilha formativa personalizada e viável.

2.5 CONSIDERAÇÕES FINAIS DA FUNDAMENTAÇÃO TEÓRICA

Esta seção apresentou os fundamentos teóricos que embasam a construção de sistemas de recomendação aplicados ao contexto acadêmico. Foram discutidas as principais abordagens de recomendação e evidenciada a adequação dos sistemas baseados em conhecimento para cenários que demandam o atendimento de regras e restrições específicas.

Também foram abordadas as métricas de avaliação empregadas em problemas de recomendação *top-k*, com destaque para o *Hit Ratio*, adotado para mensurar a aderência entre as recomendações produzidas e as escolhas realizadas pelos estudantes.

Adicionalmente, foram apresentados os Problemas de Satisfação de Restrições e sua aplicação ao problema de elaboração de grades horárias universitárias, caracterizado pela necessidade de atender simultaneamente a restrições rígidas e preferências individuais. Nesse contexto, verificou-se que a combinação entre técnicas de recomendação baseadas em conhecimento e modelagem por restrições constitui uma abordagem adequada para a geração de grades horárias personalizadas e consistentes com as exigências institucionais e as necessidades dos discentes.

3 PROCESSO DE DESENVOLVIMENTO DE SISTEMAS

Esta seção apresenta os métodos, técnicas e procedimentos adotados para o desenvolvimento do sistema de recomendação de grades horárias.

3.1 ABORDAGEM METODOLÓGICA

Do ponto de vista dos procedimentos, o trabalho envolve modelagem computacional, desenvolvimento de software e validação do sistema por meio de testes. Conforme Sommerville (2019), o desenvolvimento de sistemas computacionais abrange etapas como especificação, implementação e validação, sendo esta última essencial para assegurar que o sistema atenda aos requisitos estabelecidos.

Adicionalmente, segundo Gil (2010), a pesquisa experimental caracteriza-se pela realização de testes com o objetivo de avaliar soluções para problemas específicos. Nesse sentido, o presente trabalho pode ser classificado como de natureza experimental e tecnológica, uma vez que envolve a construção e avaliação de uma solução computacional aplicada a um contexto real.

Como ferramenta de apoio ao desenvolvimento deste trabalho, foi utilizada inteligência artificial generativa para auxiliar na revisão gramatical e na melhoria da redação dos textos, bem como na elaboração de elementos gráficos e ilustrações empregados ao longo do documento. Todo o conteúdo produzido com esse suporte foi revisado, validado e adaptado pelo autor, que permaneceu responsável pela elaboração, interpretação e veracidade das informações apresentadas.

3.2 PROCESSO DE DESENVOLVIMENTO DO SISTEMA

Para o desenvolvimento do sistema, foi adotado um processo ágil, com foco na flexibilidade e adaptação contínua, utilizando o método Kanban.

Traduzindo para o português, Kanban significa cartão. Pode ser entendido como um método de gestão, utilizado para controlar a produção de forma puxada, ou seja, de acordo com a demanda real. Nesse sistema, a empresa organiza seus processos considerando a necessidade de consumo de produtos, utilizando cartões que contêm informações importantes sobre os materiais. Esses cartões orientam a movimentação, o abastecimento e a reposição, tornando o fluxo de trabalho mais eficiente e evitando excessos ou faltas na produção (Gonçalves; Reis; Santos, 2016).

O Kanban permite a visualização do fluxo de trabalho, facilitando o gerenciamento das tarefas e a identificação de gargalos no desenvolvimento. As atividades foram organizadas em um quadro dividido em colunas, como: Tarefas a serem realizadas (*backlog*), A fazer (*to-do*), Em andamento (*doing*) e concluído (*done*).

Cada tarefa foi representada por um cartão, contendo sua descrição e prioridade. O fluxo de desenvolvimento foi contínuo, sem iterações fixas, permitindo ajustes conforme a evolução do projeto.

O processo foi dividido nas seguintes etapas:

1. Levantamento de requisitos
2. Identificação das necessidades do sistema com base no problema proposto.
3. Modelagem do problema (CSP)
4. Definição das variáveis, domínios e restrições para representar a grade horária.
5. Projeto do sistema
6. Definição da arquitetura, estrutura de dados e tecnologias utilizadas.
7. Implementação
8. Desenvolvimento do sistema e dos algoritmos de recomendação.
9. Testes e validação
10. Verificação do funcionamento do sistema e análise dos resultados gerados.
11. Ajustes e refinamento

3.3 LEVANTAMENTO DE REQUISITOS

O levantamento de requisitos foi conduzido a partir do problema identificado e dos objetivos propostos para o sistema, buscando compreender de forma aprofundada as necessidades reais dos usuários. Para isso, foram realizadas entrevistas com o coordenador do curso e com estudantes do curso de Ciência da Informação.

As entrevistas tiveram como objetivo identificar dificuldades recorrentes no processo de matrícula, estratégias utilizadas pelos alunos para montagem da grade horária e expectativas em relação a possíveis soluções automatizadas.

A partir dessas contribuições, foi possível estruturar os requisitos do sistema de forma alinhada à realidade acadêmica, garantindo que as funcionalidades propostas atendam tanto às exigências institucionais quanto às necessidades práticas dos estudantes. Os requisitos funcionais a seguir descrevem as principais funcionalidades que o sistema deve oferecer para apoiar o planejamento acadêmico de forma eficiente e automatizada.

3.3.1 Requisitos funcionais

Os requisitos funcionais descrevem as funcionalidades que o sistema deve oferecer:

- RF01: Permitir o cadastro de disciplinas via arquivo gerado pelo CAGR
- RF02: Receber o histórico acadêmico do aluno
- RF03: Identificar disciplinas já cursadas e aprovadas
- RF04: Permitir a definição de restrições do usuário (ex: horários indisponíveis)
- RF05: Gerar automaticamente sugestões de grade horária
- RF06: Garantir que não haja conflito de horários entre disciplinas
- RF07: Priorizar disciplinas obrigatórias ainda não concluídas
- RF08: Considerar pré-requisitos das disciplinas

3.3.2 Requisitos não funcionais

Os requisitos não funcionais definem características de qualidade do sistema:

- RNF01: O sistema deve apresentar tempo de resposta adequado
- RNF02: O sistema deve ser de fácil utilização
- RNF03: O sistema deve ser modular e de fácil manutenção
- RNF04: O sistema deve ser implementado com tecnologias acessíveis
- RNF05: O sistema deve garantir consistência nos dados processados
- RNF06: O sistema deve ser capaz de lidar com diferentes perfis de alunos

3.4 MODELAGEM DO PROBLEMA COMO CSP

O problema de geração de grade horária será modelado como um Problema de Satisfação de Restrições, conforme apresentado na fundamentação teórica.

- Variáveis (V): disciplinas a serem cursadas
- Domínios (D): horários disponíveis para cada disciplina
- Restrições (C):
 - Conflito de horários;
 - Pré-requisitos;
 - Disponibilidade do aluno;
 - Limite de carga horário.

A solução do problema corresponde a uma combinação de disciplinas que satisfaça todas as restrições impostas.

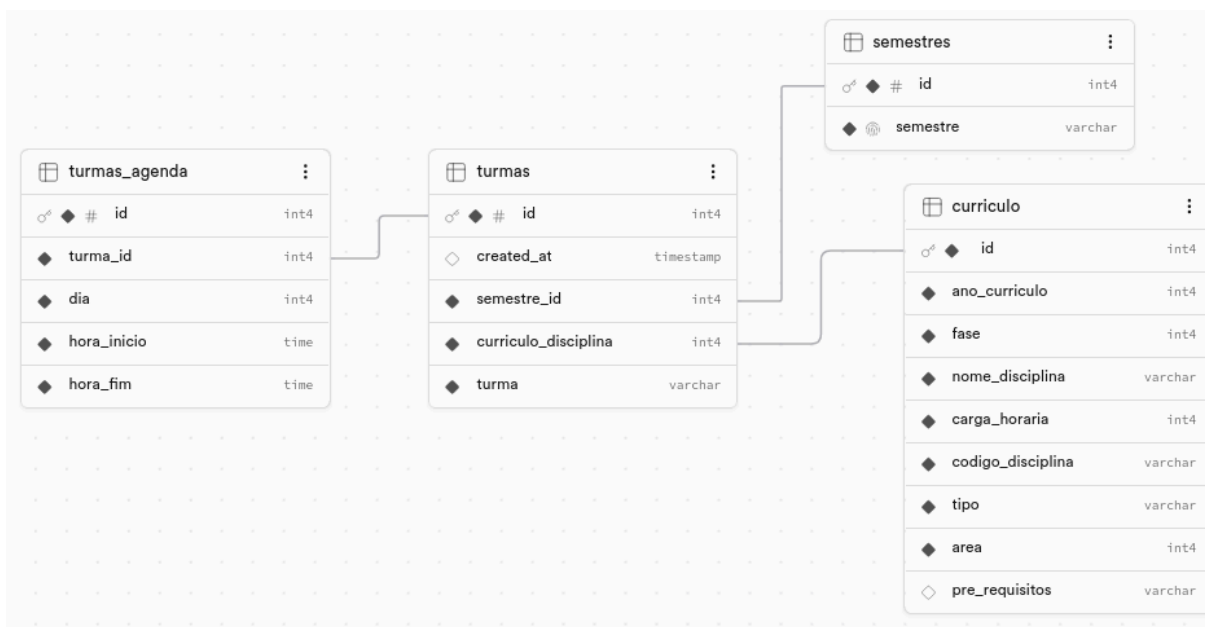
3.6 LIMITAÇÕES DO SISTEMA

O sistema de recomendação proposto apresenta limitações técnicas e de escopo que devem ser consideradas para a correta compreensão de sua aplicabilidade:

- a) Ausência de Acesso em Tempo Real: O modelo não possui integração direta ou sincronização automática via API com a base de dados do Cadastro de Graduação (CAGR). Consequentemente, a atualização das turmas e horários depende de um processo manual de *upload* de arquivos pelo administrador do sistema. A fidedignidade das recomendações está, portanto, condicionada à frequência dessas atualizações.
- b) Escopo das Recomendações: Conforme estabelecido no Projeto Pedagógico do Curso (PPC) de Ciência da Informação, os estudantes possuem flexibilidade para cursar disciplinas em outros departamentos para a integralização de horas de atividades complementares ou eletivas. Entretanto, o motor de recomendação deste trabalho limita-se estritamente às disciplinas ofertadas pelos departamentos vinculados ao curso de Ciência da Informação da UFSC. Portanto, componentes curriculares externos ou itinerários formativos transversais não são contemplados pelo algoritmo nas sugestões de grade horária.

3.7 DIAGRAMA ESQUEMÁTICO DE BANCO DE DADOS

Figura 1 - Diagrama esquemático do banco de dados do sistema



Fonte: Elaborado pelo autor (2026).

O banco de dados é composto por quatro entidades principais, as quais representam os processos de estrutura curricular e de oferta de disciplinas. A entidade **currículo** armazena as informações relativas às disciplinas obrigatórias e optativas de cada ano curricular. As entidades **semestres**, **turmas** e **turmas_agenda**, por sua vez, armazenam os dados referentes às ofertas realizadas em cada período letivo, incluindo as turmas disponíveis e seus respectivos horários.

3.8 CONSIDERAÇÕES FINAIS DO PROCESSO DE DESENVOLVIMENTO DE SISTEMAS

Esta seção apresentou os procedimentos metodológicos adotados para o desenvolvimento do sistema de recomendação de grades horárias. Foram descritas a abordagem experimental e tecnológica empregada, o processo de desenvolvimento baseado no método Kanban e as etapas de levantamento de requisitos, modelagem do problema e implementação da solução.

Também foram definidos os requisitos funcionais e não funcionais do sistema, bem como a modelagem do problema como um Problema de Satisfação de Restrições, permitindo

representar formalmente as variáveis, domínios e restrições envolvidos no planejamento acadêmico. Adicionalmente, foram apresentadas as limitações do sistema e a estrutura do banco de dados responsável pelo armazenamento das informações curriculares e das ofertas de disciplinas.

Dessa forma, os procedimentos e modelos descritos nesta seção estabelecem a base necessária para a implementação e a avaliação do sistema de recomendação proposto, assegurando sua aderência às necessidades dos usuários e às particularidades do contexto acadêmico.

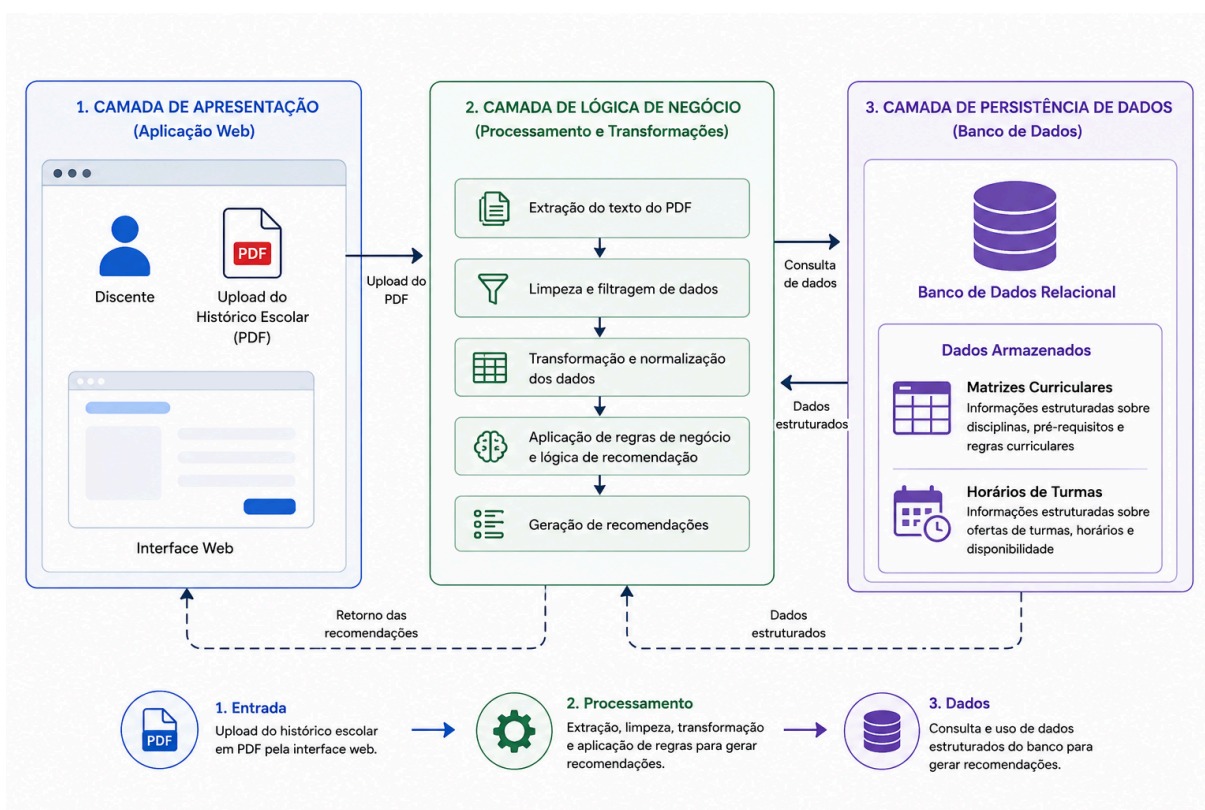
4 ARQUITETURA E IMPLEMENTAÇÃO DO SISTEMA

Nesta seção, são apresentadas a arquitetura e a implementação computacional do sistema de recomendação de grade curricular proposto. A descrição detalha a divisão em camadas da aplicação, os módulos de processamento e extração de dados (*parsers*), o esquema de persistência, e a codificação do motor de recomendação baseado no paradigma de Satisfação de Restrições (CSP).

4.1 VISÃO GERAL DA ARQUITETURA

Para garantir modularidade, facilidade de manutenção e consistência dos dados processados (conforme estabelecido nos requisitos RNF03 e RNF05), o sistema foi desenvolvido utilizando uma arquitetura em três camadas fundamentais: Camada de Apresentação (página *web*), Camada de Lógica de Negócio (processamento de dados e transformações) e Camada de Persistência de Dados (banco de dados).

Figura 2 - Diagrama de arquitetura do sistema



Fonte: Elaborado pelo autor (2026)

A separação lógica permite que o motor de recomendação opere de maneira independente da interface de usuário e do banco de dados relacional. O fluxo geral de dados inicia-se com a entrada de dados não estruturados (o histórico escolar em formato PDF) fornecidos pelo discente na Camada de Apresentação, passa pelas transformações e filtrações lógicas na Camada de Lógica de Negócio e consome os dados estruturados de matrizes curriculares e horários de turmas armazenadas na Camada de Persistência.

4.2 MÓDULO DE EXTRAÇÃO DE DADOS (*PARSING*)

Uma das etapas críticas para o funcionamento do recomendador baseado em conhecimento é a ingestão de dados individuais e de conteúdo. Essa atribuição é dividida entre dois subcomponentes especializados que utilizam a biblioteca *pdfplumber* e expressões regulares para converter documentos PDF em estruturas de dados manipuláveis (*list* e *dict* em Python):

1. **Módulo de Histórico Escolar (*historico_pdf_parser.py*):** Este módulo realiza a análise do arquivo de Controle Curricular fornecido pelo estudante (atendendo aos requisitos RF01 e RF02). O algoritmo mapeia o texto linha por linha, isolando os códigos de disciplinas (padrões como **CIN***d*+ ou três letras maiúsculas seguidas de quatro dígitos). A lógica do fluxo valida o *status* da disciplina: caso identifique o termo "Cursando", o componente é classificado com o *status* CURSANDO; caso encontre um registro de nota, valida se o valor flutuante é igual ou superior a 6.0, incluindo a disciplina no conjunto APROVADO. Disciplinas marcadas com "Não Cursou" são sumariamente descartadas. O código-fonte completo deste módulo encontra-se no Apêndice B.
2. **Módulo de Grade Horária Institucional (*pdf_parser.py*):** Utilizado primordialmente para a carga administrativa de dados das turmas ofertadas em um determinado semestre letivo. O *parser* decodifica a notação de horários padrão da Universidade Federal de Santa Catarina (UFSC), expressa no formato **d.hhmm-n** (onde **d** representa o dia da semana de 2 a 6, **hhmm** o horário de início e **n** o número de blocos de 50 minutos). O algoritmo realiza o cálculo exato do horário de término por meio da função *calcular_fim_com_intervalo()*, computando os intervalos regulamentares de 10 minutos vigentes na instituição (10:00–10:10 e 20:10–20:20) para evitar distorções

temporais na validação de choques de horário. Ao final, o módulo gera dinamicamente *scripts* SQL para popular a base de dados. O código-fonte completo deste módulo encontra-se no Apêndice C.

4.3 CAMADA DE PERSISTÊNCIA E INTEGRAÇÃO COM BANCO DE DADOS

A persistência de dados foi implementada utilizando a plataforma *Backend-as-a-Service* (BaaS) **Supabase**, que fornece um banco de dados relacional **PostgreSQL**. A comunicação com o ecossistema Python ocorre por meio do cliente oficial da API Supabase, encapsulado no módulo *supabase_conn.py*. O código-fonte completo deste módulo encontra-se no Apêndice D.

A integração realiza consultas relacionais complexas de forma otimizada. A função *get_curriculo()* recupera a matriz pedagógica oficial filtrada pelo ano de ingresso selecionado pelo usuário (currículos 2016 ou 2026), ordenando os registros para garantir que disciplinas obrigatórias precedam logicamente as optativas.

Para alimentar o motor de recomendação, a função *get_horarios()* executa uma junção relacional interna (*inner join*) que recupera simultaneamente os dados da disciplina e os metadados de suas respectivas turmas e agendas de funcionamento. Essa estratégia minimiza a sobrecarga de rede ao trazer em uma única requisição a árvore aninhada de horários necessários para o cálculo de conflitos.

4.4 MOTOR DE RECOMENDAÇÃO E RESOLUÇÃO DE RESTRIÇÕES (CSP)

O núcleo de inteligência do sistema reside no módulo *rec_sys.py*, o qual traduz as definições formais de variáveis, domínios e restrições estabelecidas na seção 3.4 para um algoritmo computacional de filtragem e seleção gulosa orientada por prioridades. O código-fonte completo deste módulo encontra-se no Apêndice E.

4.4.1 Modelagem e Processamento de Restrições Rígidas

As restrições rígidas atuam como filtros absolutos para podar o espaço de busca. O tempo é tratado de forma contínua através da conversão de horários (HH:MM:SS) em minutos inteiros transcorridos desde a meia-noite (função *parse_time()*). A detecção de sobreposição de intervalos temporais é dada pela função matemática:

$$Overlap(A, B) = \neg(A_{FIM} \leq B_{INICIO} \vee A_{INICIO} \geq B_{FIM})$$

A validação de consistência do CSP executa três filtragens concorrentes na função *filtrar_disciplinas()*, que utilizam opções selecionadas pelo usuário na utilização do sistema:

- **Bloqueio de Turnos (turno_bloqueado):** Elimina do domínio da disciplina quaisquer turmas cujas aulas interceptem os intervalos de minutos definidos para as preferências do usuário.
- **Bloqueio de Horários Específicos (horario_bloqueado):** Remove turmas que coincidam com dias inteiros ou *slots* de horários específicos marcados como indisponíveis pelo estudante via interface.
- **Regra Especial de Consistência Pedagógica:** O sistema implementa uma regra de negócio específica do curso: a disciplina de Estágio Obrigatório (CIN7505) é removida do domínio de variáveis disponíveis se o algoritmo detectar a existência de qualquer disciplina pendente pertencente ao núcleo comum inicial (fases 1, 2, 3 ou 4).

4.4.2 Modelagem de Restrições Flexíveis e Algoritmo de Seleção

Uma vez reduzido o domínio das variáveis para turmas estritamente viáveis e sem conflitos locais, o sistema realiza a ordenação das disciplinas para a montagem da grade otimizada através da função *selecionar_grade()*. A ordenação baseia-se em uma tupla de prioridades calculada pela função *prioridade_disciplina()*, estruturada da seguinte forma:

$$PRIORIDADE = \{Obrigatoriedade, Fase Pedagógica, Alinhamento de Área\}$$

Em termos de codificação, a ordenação avalia os elementos em ordem de relevância decrescente:

1. **Obrigatoriedade:** Disciplinas obrigatórias recebem peso **0** (maior prioridade de alocação) e optativas recebem peso **1**. No Python, quando a função *sorted()* é utilizada para ordenar uma lista (como é feito na função *selecionar_grade()*) e o critério de ordenação é uma tupla, a ordenação padrão ocorre de forma crescente (do menor para o maior) elemento por elemento.
2. **Fase Pedagógica:** Para obrigatórias, priorizam-se as fases mais iniciais para evitar a retenção de fluxo do aluno.

3. **Alinhamento de Área:** Para optativas, o sistema pontua melhor aquelas que pertencem à área de preferência declarada pelo usuário (Tecnologia da Informação ou Gestão da Informação), configurando a personalização da recomendação baseada em conhecimento.

Após a ordenação, o algoritmo executa uma busca sequencial iterativa. Uma disciplina e sua melhor turma associada são inseridas na grade final se, e somente se:

1. A inclusão de sua carga horária não violar o limite teto máximo estabelecido para o semestre letivo (468 horas).
2. Não houver conflito de horário com nenhuma outra turma previamente selecionada na mesma execução (*possui_conflito_horario()*).
3. Para optativas, a carga acumulada não estourar o limite de horas de optativas restantes calculadas dinamicamente pelo módulo *disciplinas_handler.py* com base nas equivalências dos currículos 2016 e 2026. O código fonte pode ser visto no apêndice F e G.

4.5 INTERFACE DO USUÁRIO E VISUALIZAÇÃO DA GRADE

A interface gráfica com o usuário foi construída utilizando o *framework Streamlit* (*app.py*), provendo uma aplicação web responsiva e centralizada em uma única página (*Single Page Application*). O leiaute organiza-se em blocos lógicos sequenciais. O código fonte pode ser encontrado no apêndice H.

O usuário anexa seu arquivo PDF de controle curricular e indica seu ano de matriz. O sistema imediatamente processa o arquivo e renderiza em tela três tabelas dinâmicas utilizando o componente *st.dataframe()* em colunas paralelas: disciplinas **Aprovadas**, disciplinas em curso (**Cursando**) e disciplinas pendentes (**Para Cursar**), conferindo transparência aos dados extraídos.

Na sequência, o discente interage com os seletores de restrições (*st.multiselect* e *st.radio*), definindo os turnos bloqueados, horários específicos de indisponibilidade e preferências de direcionamento de disciplinas optativas.

Ao acionar o botão de comando "Gerar!", a lógica de negócio é disparada. Após o retorno do motor de recomendação, o módulo *timetable_handler.py* entra em ação para

estruturar os dados. A função ***build_timetable()*** mapeia as turmas selecionadas contra uma matriz estática que representa os quatorze *slots* de horários oficiais da instituição (***SLOTS_UFSC***).

Figura 3 - Grade curricular recomendada renderizada

Gere sua recomendação de grade curricular

Gerar!

Sua Grade de Horários

slot	Segunda	Terça	Quarta	Quinta	Sexta	Sábado
11:50						
13:30						
14:20						
15:10						
16:20						
17:10 1						
18:30	CIN7501	MTM3110	CIN7601	MTM3110	CIN7201	
19:20	CIN7501	MTM3110	CIN7601	MTM3110	CIN7201	
20:20	CIN7406	CIN7301	CIN7943	CIN7412	CIN7201	
22:00	CIN7406	CIN7301	CIN7943	CIN7412	CIN7201	

Fonte: Elaborado pelo autor (2026).

Por fim, a função ***render_timetable()*** aplica uma operação de pivotamento de matriz, reorganizando os *slots* como linhas e os dias da semana como colunas, preenchendo as células vazias com *strings* nulas. O resultado final é renderizado na tela do usuário como um cronograma semanal visual intuitivo de fácil leitura, acompanhado da listagem detalhada de metadados das disciplinas recomendadas.

4.6 CONSIDERAÇÕES FINAIS DA ARQUITETURA E IMPLEMENTAÇÃO DO SISTEMA

Esta seção apresentou a arquitetura e a implementação do sistema de recomendação de grades horárias, detalhando seus principais componentes e o fluxo de processamento dos dados. Foram descritos os módulos responsáveis pela extração de informações, persistência dos dados, aplicação das regras de negócio e geração das recomendações.

Também foi demonstrado como a modelagem baseada em Problemas de Satisfação de Restrições foi traduzida para uma solução computacional capaz de considerar simultaneamente restrições acadêmicas e preferências dos estudantes. Por fim, foram apresentados os mecanismos de interação com o usuário e de visualização das grades geradas, consolidando a implementação da solução proposta e fornecendo a base para a análise dos resultados apresentada na seção seguinte.

5 TESTES E VALIDAÇÃO

Esta seção apresenta os resultados obtidos na etapa de validação do sistema de recomendação de grades horárias. A avaliação foi conduzida com estudantes do curso de Ciência da Informação da Universidade Federal de Santa Catarina (UFSC), buscando analisar tanto a qualidade das recomendações geradas quanto a usabilidade da aplicação.

5.1 CÁLCULO DA ESCALA DE USABILIDADE DE SISTEMAS

Com o objetivo de avaliar a usabilidade do sistema proposto, foi aplicada a Escala de Usabilidade de Sistemas (*System Usability Scale* - SUS), composta por dez afirmações respondidas em escala Likert de cinco pontos, variando de 1 (discordo totalmente) a 5 (concordo totalmente). Ao todo, 24 estudantes participaram da validação do sistema.

Tabela 1 - Escore por participante

Participante	Escore SUS
1	90
2	100
3	97,5
4	72,5
5	82,5
6	85
7	95
8	100
9	95
10	97,5
11	95
12	95
13	95
14	97,5
15	82,5
16	95
17	100
18	70
19	72,5
20	87,5
21	92,5
22	80
23	95
24	97,5

Fonte: Elaborado pelo autor (2026).

A soma dos escores individuais foi igual a 2145 pontos. Dividindo-se esse valor pelo número de participantes (n=24), obteve-se um escore médio de 89,38.

Segundo Brooke (1996), um escore superior a 68 pontos já representa uma usabilidade acima da média. Complementarmente, Sauro e Lewis (2016) classificam sistemas com pontuações superiores a 80 pontos como de **excelente usabilidade**, indicando elevado grau de satisfação dos usuários e forte probabilidade de recomendação da aplicação. Dessa forma, o resultado obtido demonstra que os participantes perceberam o sistema como intuitivo, de fácil aprendizagem e adequado para apoiar o processo de planejamento da matrícula.

5.2 AVALIAÇÃO DA QUALIDADE DAS RECOMENDAÇÕES POR MEIO DO *HIT RATIO*

Além da avaliação de usabilidade, foi analisada a qualidade das recomendações produzidas pelo sistema utilizando a métrica *Hit Ratio*, apresentada na seção 2.2. Essa métrica mede a proporção de itens recomendados que foram considerados relevantes pelos usuários.

A soma dos valores individuais foi igual a **20,99**. Assim, considerando os 24 participantes, obteve-se um *Hit Ratio* médio de **0,875**, indicando que aproximadamente 87,5% das disciplinas recomendadas foram consideradas adequadas pelos participantes.

Esse resultado demonstra elevada aderência entre as recomendações produzidas pelo sistema e o planejamento acadêmico dos estudantes. Observa-se que 13 dos 24 participantes (54,2%) atribuíram *Hit Ratio* igual a 1,0, indicando concordância total com as recomendações geradas.

Os menores valores observados decorreram, principalmente, de situações particulares relatadas pelos próprios participantes, como intenção de mudança de curso, interesse em cursar disciplinas de outros departamentos, validação futura de componentes curriculares ou preferências pessoais que não fazem parte das regras atualmente modeladas pelo sistema. Esses casos não representam falhas do algoritmo de recomendação, mas limitações inerentes ao escopo adotado neste trabalho.

Tabela 2 - Hit Ratio por participante

Participante	Hit Ratio
1	1
2	1
3	0,6
4	0,8
5	0,71
6	0,56
7	0,78
8	0,89
9	1
10	0,86
11	1
12	1
13	1
14	1
15	0,75
16	0,71
17	1
18	1
19	0,33
20	1
21	1
22	1
23	1
24	1

Fonte: Elaborado pelo autor (2026).

5.3 DISCUSSÃO DOS RESULTADOS

Além da avaliação quantitativa, os participantes registraram comentários e sugestões que contribuíram para uma análise qualitativa do sistema. De maneira geral, observou-se elevada aceitação da proposta. Diversos estudantes relataram que o sistema reduz significativamente o tempo necessário para montar a grade horária e facilita a identificação das disciplinas mais adequadas ao semestre, principalmente por considerar automaticamente pré-requisitos, conflitos de horários e preferências do usuário. Entre as principais sugestões de melhoria, destacam-se:

- a) permitir que o usuário defina disciplinas prioritárias ou exclua disciplinas que não deseja cursar naquele semestre;

- b) apresentar mais alternativas de disciplinas optativas compatíveis com os horários disponíveis, e não apenas a recomendação considerada ideal pelo algoritmo;
- c) exibir diretamente a turma selecionada ao lado de cada disciplina recomendada;
- d) aprimorar a visualização da grade horária, possibilitando a exibição completa dos períodos da manhã, tarde e noite sem necessidade de rolagem;

Também foram observados alguns casos particulares em que as recomendações diferiram do planejamento pessoal do estudante. Em todos esses casos, verificou-se que a divergência estava relacionada a fatores externos ao escopo do sistema, como intenção de mudança de curso, validação futura de disciplinas, preferências individuais por determinadas matérias ou alterações de currículo ainda não refletidas nas regras de recomendação.

Esses resultados indicam que o motor de recomendação desenvolvido atende adequadamente às restrições acadêmicas previstas no Projeto Pedagógico do Curso, ao mesmo tempo em que apresenta elevado nível de usabilidade. As sugestões coletadas durante a validação representam oportunidades para o aprimoramento da aplicação, especialmente no que se refere à personalização das recomendações e à evolução da interface do usuário, aspectos já contemplados nas propostas de trabalhos futuros apresentadas neste estudo.

6 CONCLUSÃO

O presente trabalho alcançou com êxito o seu objetivo geral ao propor, modelar e implementar um sistema computacional funcional focado na recomendação e montagem automatizada de grades horárias integrais para o curso de Ciência da Informação da Universidade Federal de Santa Catarina. O trabalho responde de forma positiva à problemática central levantada: a viabilidade de mitigar a complexidade no planejamento de matrículas de estudantes com trajetórias acadêmicas irregulares por meio de abordagens algorítmicas baseadas em conhecimento.

No que tange aos objetivos específicos estabelecidos, as seguintes contribuições foram consolidadas:

- **Modelagem do Problema como CSP:** O cenário de *timetabling* acadêmico sob a perspectiva do discente foi formalizado com precisão, mapeando componentes curriculares pendentes como variáveis, turmas ofertadas como domínios e conflitos físicos e pedagógicos como restrições.
- **Desenvolvimento do Modelo Computacional:** A arquitetura do sistema foi materializada utilizando a linguagem Python, o framework *Streamlit* para interface com o usuário e a plataforma *Supabase (PostgreSQL)* para a camada de persistência, resultando em uma aplicação modular e de baixo acoplamento.
- **Testes e Validação:** O motor de recomendação demonstrou capacidade de processar dados não estruturados extraídos de PDFs oficiais, podar o espaço de busca eliminando ramos inconsistentes e gerar cronogramas validados que eliminam choques de horários (*hard constraints*) ao mesmo tempo em que priorizam as preferências subjetivas de turnos e áreas de optativas (*soft constraints*).
- **Documentação do Processo:** As etapas de levantamento de requisitos, modelagem matemática, arquitetura de software e lógica dos algoritmos foram integralmente registradas ao longo deste documento.

Em suma, a união do paradigma de Satisfação de Restrições (CSP) com os Sistemas de Recomendação Baseados em Conhecimento provou-se uma abordagem robusta para o domínio acadêmico.

6.1 SUGESTÕES PARA TRABALHOS FUTUROS

Apesar dos resultados satisfatórios, a natureza experimental e tecnológica deste trabalho abre margem para expansão e refinamentos. Deste modo, recomendam-se os seguintes desdobramentos para trabalhos futuros:

- **Automação na Ingestão de Dados (API/Web Scrapping):** Desenvolver um módulo de integração ou raspagem de dados automatizada junto ao Cadastro de Graduação (CAGR) da UFSC para eliminar o processo de upload manual dos horários de turmas pelo administrador, permitindo o monitoramento do saldo de vagas em tempo real durante os períodos de ajuste de matrícula.
- **Ampliação para outros Departamentos:** adaptar o sistema para contemplar disciplinas ofertadas por outros departamentos da UFSC, bem como permitir sua aplicação em diferentes cursos de graduação, tornando a solução mais abrangente e flexível.
- **Aprimoramento da interface e da experiência do usuário:** implementar melhorias de usabilidade e acessibilidade, com o objetivo de tornar a interação com o sistema mais intuitiva e facilitar o processo de planejamento acadêmico.

REFERÊNCIAS

- ADOMAVICIUS, G.; TUZHILIN, A. *Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions*. IEEE Transactions on Knowledge and Data Engineering, v. 17, n. 6, p. 734-749, 2005.
- AGGARWAL, C. C. *Recommender Systems: The Textbook*. Cham: Springer, 2016.
- ALSINI, Areej; HUYNH, Du Q.; DATTA, Amitava. Hit ratio: an evaluation metric for hashtag recommendation. arXiv, arXiv:2010.01258v1 [cs.IR], 2020. Disponível em: <https://arxiv.org/abs/2010.01258>. Acesso em: 20 jun. 2026.
- BABAEI, Heidar; KARIMPOUR, Javad; HADIDI, Ali. *A survey of automated university timetabling*. Computers & Industrial Engineering, v. 86, p. 43-59, 2015.
- BROOKE, John. *SUS: a quick and dirty usability scale*. In: JORDAN, P. W.; THOMAS, B.; WEERDMEESTER, B.; MCCLELLAND, I. (org.). *Usability Evaluation in Industry*. London: Taylor & Francis, 1996. p. 189-194.
- BURKE, Robin; Felfernig, Alexander; GÖKER, Mehmet H. *Recommender Systems: An Overview*. AI Magazine, [s. l.], v. 32, n. 3, p. 13-18, 2011.
- BURKE, Edmund K.; PETROVIC, Sanja. *Recent research directions in automated timetabling*. European Journal of Operational Research, v. 140, n. 2, p. 266-280, 2002.
- DAMASCENO, Leandro Andrade. *Aplicação de um sistema de recomendação para ambientes virtuais de aprendizagem*. 2024. 67 f. Monografia (Graduação em Ciência da Computação) – Centro de Informática, Universidade Federal de Pernambuco, Recife, 2024.
- GIL, Antonio Carlos. *Métodos e técnicas de pesquisa social*. 6. ed. São Paulo: Atlas, 2010.
- GONÇALVES, L. C. A.; REIS, L. P.; SANTOS, J. M. F. *Aplicação do sistema Kanban no almoxarifado de uma indústria produtora de álcool combustível*. In: ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO, 36., 2016, João Pessoa. *Anais...* João Pessoa: ENEGE, 2016. 17 p. Disponível em: https://www.abepro.org.br/biblioteca/TN_STO_226_323_28985.pdf. Acesso em: 30 mar. 2026.
- JANNACH, Dietmar *et al.* *Recommender systems: an introduction*. Cambridge: Cambridge University Press, 2011.
- MARQUES, Gerson Adriano. *Um modelo de sistema de recomendação de atividades complementares para capacitação profissional do aluno de graduação*. 2022. 121 f. Dissertação (Mestrado em Computação Aplicada) – Universidade do Vale do Rio dos Sinos, São Leopoldo, 2022.
- NASCIMENTO, Mairon Santana do; GONDIM, Ruan dos Santos; COSTA FILHO, Raimundo Valter. *Desenvolvimento de um sistema de recomendação de orientadores para TCCs utilizando um modelo preditivo baseado em Naive Bayes*. Aracati: IFCE, 2023.

QUEIROZ, Rodrigo Modesto Câmara. *Protótipo de sistema de recomendação para auxiliar na escolha de disciplinas optativas do curso de Sistemas de Informação da Universidade Federal do Ceará - Campus Quixadá*. 2023. 54 f. Monografia (Bacharelado em Sistemas de Informação) – Universidade Federal do Ceará, Quixadá, 2023.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 4. ed. Pearson, 2021.

SAURO, Jeff; LEWIS, James R. *Quantifying the user experience: practical statistics for user research*. 2. ed. Cambridge: Morgan Kaufmann, 2016.

SCHMIDT, Douglas C.; WHITE, Jules; BENAVIDES, David; SALIDO, Miguel A. *Constraint Satisfaction Problems*. ScienceDirect. Disponível em: <https://www.sciencedirect.com/topics/computer-science/constraint-satisfaction-problems>. Acesso em: 29 mar. 2026.

SOMMERVILLE, Ian. *Engenharia de software*. 10. ed. São Paulo: Pearson, 2019.

UTA, Mathias *et al.* *Knowledge-based recommender systems: overview and research directions*. *Frontiers in Big Data*, [s. l.], v. 7, art. 1304439, p. 1-18, 26 fev. 2024. DOI: 10.3389/fdata.2024.1304439. Disponível em: <https://doi.org/10.3389/fdata.2024.1304439>. Acesso em: 13 jun. 2026.

WEN, Zheng. *Recommendation system based on collaborative filtering*. 2008. Disponível em: <http://www.zheng-wen.com/WenRecommendation.pdf>. Acesso em: 29 mar. 2026.

WREN, Anthony. *Scheduling, timetabling and rostering: a special relationship?* In: BURKE, E. K.; ROSS, P. (org.). *Practice and Theory of Automated Timetabling I*. Berlin: Springer, 1996. p. 46–75.

APÊNDICE A – Como utilizar o sistema

Para iniciar, o usuário deve ter acesso ao seu controle curricular, que é facilmente obtido acessando ao Controle Acadêmico da Graduação - CAGR. Ao fazer *login* com seu cadastro UFSC, o aluno tem acesso ao clicar em ‘Controle Curricular’.

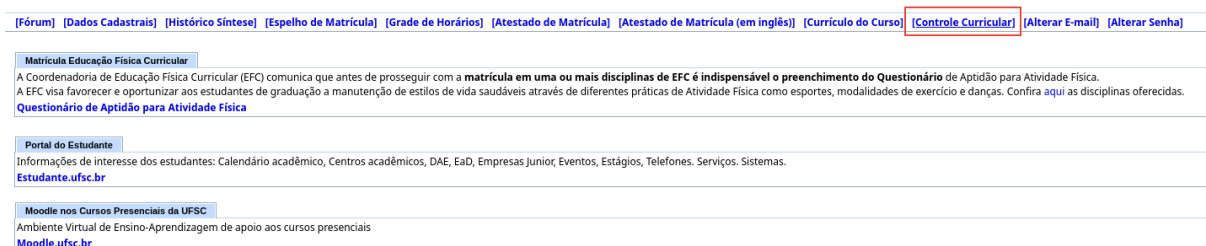


Figura 2 - Como obter o arquivo do Controle Curricular

Fonte: CAGR (2026)

Ao clicar na opção, o controle curricular é automaticamente baixado em formato PDF. Este deve ser o documento a ser utilizado pelo usuário no sistema de recomendação de grade curricular.

Em seguida, acesse: <https://ufsc-cinfo.streamlit.app/>.



Figura 3 - Página inicial do Sistema de Recomendação de Grade Curricular

Fonte: <https://ufsc-cinfo.streamlit.app/> (2026)

Clique em 'Upload', selecione o arquivo que contém o controle curricular anteriormente baixado. Selecione também a qual ano que seu currículo pertence.

O sistema irá retornar três tabelas: Aprovadas, cursando e Para cursar. Caso seja um aluno do primeiro semestre que ainda não foi aprovado em nenhuma disciplina, a tabela "Aprovadas" estará vazia.

APROVADAS					CURSANDO				PARA CURSAR					
codigo_disciplina	nome_disciplina	nota	carga_horaria	ti	codigo_disciplina	nome_disciplina	carga_horaria	tipo	codigo_disciplina	nome_disciplina	fase	tipo	area	car
CIN7141	Lógica Instrumental I	8.5	36	C	CAD5103	Administração I	72	Ob	OPT	OPTATIVAS	0	Op/EX	0	
CIN7143	Empreendedorismo I	9	36	C	MTM3110	Cálculo	72	Ob	CIN7943	Experiência do Usuário (UX) User Experience	1	Ob	1	
CIN7144	Tutoria Acadêmica I	10	18	C	CIN7201	Sistemas de Organização do Conhecimento	72	Ob	CIN7145	Gestão da Informação	1	Ob	2	
CIN7925	Introdução a Algoritmos	7.5	36	C	CIN7412	Marketing da Informação	36	Ob	CIN7309	Gestão de Processos Organizacionais	2	Ob	2	
LLV7802	Leitura e Produção do Texto	10	72	C					CIN7907	Lógica Aplicada I	2	Ob	1	
CIN7140	Pesquisa Bibliográfica	10	72	C					INE5111	Estatística Aplicada I	2	Ob	1	
CIN7142	Evolução do Pensamento Filosófico e Científico	10	36	C					CIN7204	Tutoria Acadêmica II	2	Ob	0	
CIN7306	Competência Informacional	9.5	36	C					CIN7301	Introdução à Representação Temática	3	Ob	2	
CIN7905	Teoria da Decisão	9	36	C					CIN7302	Introdução à Representação Descritiva	3	Ob	2	
									CIN7304	Introdução à Bancos de Dados	3	Ob	1	

Figura 4 - Tabelas de disciplinas

Fonte: Elaborado pelo autor (2026)

Logo após, o usuário poderá escolher suas restrições e prioridades. Restrições de Turnos se referem a períodos completos, como Todas as Manhãs, por exemplo.

As restrições de horários se referem a horários ou dias específicos, por exemplo, todas às Segundas-Feiras ou Quarta-Feira às 18h30.

Quanto às optativas, o aluno pode escolher se deseja gerar uma grade curricular com ou sem optativas, além de escolher qual área deseja focar.

Restrições de Turnos

Turnos que você não pode ter aula

Caso não haja restrição, pule esta etapa

Restrições de Horários

Horários que você não pode ter aula

Caso não haja restrição, pule esta etapa

Optativas

Gerar grade com optativas?

☒ Sim

☐ Não, apenas disciplinas obrigatórias

Preferência por...

☒ Tecnologia da Informação

☐ Gestão da Informação

☐ Indiferente

Figura 5 - Restrições

Fonte: Elaborado pelo autor (2026)

Ao clicar em "Gerar" na seção "Gerar Controle Curricular", o sistema irá retornar uma grade curricular que obedeça às restrições e escolhas selecionadas.



Figura 6 - Grade Curricular sugerida

Fonte: Elaborado pelo autor (2026)

APÊNDICE B – Código fonte *historico_pdf_parser.py*

```
import pdfplumber
import re
from typing import List, Dict

# Variável para testar local
PDF_PATH = "INSERIR_PATH_PARA_TESTE_LOCAL.pdf"

def extrair_nome(bloco: str) -> str:
    """Remove código, carga horária e metadados"""

    # remove códigos duplicados no início
    bloco = re.sub(r"^(?:[A-Z]{3}\d{4})+", "", bloco).strip()

    # remove código restante
    bloco = re.sub(r"^(CIN\d+|[A-Z]{3}\d{4})\s+", "", bloco)

    # corta antes da carga horária (número isolado)
    partes = re.split(r"\s+\d+\b", bloco)
    nome = partes[0]

    # remove informação final (status etc)
    nome = re.sub(r"\b(Cursando|Ob|Op|EX|FS)\b.*", "", nome)

    return nome.strip()

def extrair_carga_horaria(bloco: str) -> int | None:
    """
    Extraí a carga horária da disciplina.
    Exemplo:
    CIN7140 Pesquisa Bibliográfica 72 ...
    """

    match = re.search(r"\s(\d{2,3})\s", bloco)

    if match:
        return int(match.group(1))

    return None
```

```

def extrair_tipo(bloco: str) -> str | None:
    """
    Extrai o tipo da disciplina:
    Ob = Obrigatória
    Op = Optativa
    Ex = Extensão
    """

    match = re.search(r"\b(Ob|Op|EX)\b", bloco)

    if match:
        return match.group(1)

    return None


def extrair_disciplinas(pdf_upload) -> List[Dict]:
    """
    Extrai disciplinas do PDF do histórico.
    """
    with pdfplumber.open(pdf_upload) as pdf:
        texto = ""
        for page in pdf.pages:
            texto += page.extract_text() + "\n"

    linhas = texto.split("\n")

    disciplinas = []
    codigos_vistos = set()

    for linha in linhas:
        partes = re.split(r"(?=(CIN\d+| [A-Z]{3}\d{4}))", linha)

        blocos = []
        i = 1
        while i < len(partes):
            bloco = partes[i] + partes[i + 1]
            blocos.append(bloco.strip())
            i += 2

        for bloco in blocos:

```

```

codigo_match = re.match(r"(CIN\d+|[A-Z]{3}\d{4})", bloco)

if not codigo_match:
    continue

codigo = codigo_match.group(1)

if codigo in codigos_vistos:
    continue

if "Não Cursou" in bloco:
    continue

if "Cursando" in bloco:
    disciplinas.append({
        "codigo_disciplina": codigo,
        "nome_disciplina": extrair_nome(bloco),
        "carga_horaria": extrair_carga_horaria(bloco),
        "tipo": extrair_tipo(bloco),
        "nota": None,
        "status": "CURSANDO"
    })
    codigos_vistos.add(codigo)
    continue

match_nota = re.search(r"(\d{4}/\d)\s+(\d+\.\d)", bloco)

if match_nota:
    nota = float(match_nota.group(2))

    if nota >= 6:
        disciplinas.append({
            "codigo_disciplina": codigo,
            "nome_disciplina": extrair_nome(bloco),
            "carga_horaria": extrair_carga_horaria(bloco),
            "tipo": extrair_tipo(bloco),
            "nota": nota,
            "status": "APROVADO"
        })
        codigos_vistos.add(codigo)

return disciplinas

```

```
if __name__ == "__main__":  
    disciplinas = extrair_disciplinas(PDF_PATH)  
  
    for d in disciplinas:  
        print(d)
```

APÊNDICE C – Código fonte *pdf_parser.py*

```
import pdfplumber
import re
from datetime import datetime, timedelta
from dataclasses import dataclass, field
from typing import List, Optional

# CONFIG
PDF_PATH = "./data/input/20262_biblio.pdf"
OUTPUT_SQL = "grade_biblio_20262.sql"

SEMESTRE = "20262"

BLOCO_MINUTOS = 50

# INTERVALOS
INTERVALOS = [
    ("10:00", "10:10"),
    ("20:10", "20:20"),
]

# DATACLASSES
@dataclass
class Agenda:
    dia: int
    hora_inicio: str
    hora_fim: str

@dataclass
class Turma:
    semestre: str

    codigo_disciplina: str
    turma: str
    nome_disciplina: str

    agendas: List[Agenda] = field(default_factory=list)

# HELPERS
```

```

def escape_sql(value: str) -> str:

    if value is None:
        return ""

    return value.replace("'", "'")

def calcular_fim_com_intervalo(
    inicio: datetime,
    duracao_blocos: int
) -> datetime:

    minutos_total = duracao_blocos * BLOCO_MINUTOS

    fim = inicio + timedelta(minutes=minutos_total)

    for inicio_int, fim_int in INTERVALOS:

        ini_int = inicio.replace(
            hour=int(inicio_int.split(":")[0]),
            minute=int(inicio_int.split(":")[1])
        )

        fim_intervalo = inicio.replace(
            hour=int(fim_int.split(":")[0]),
            minute=int(fim_int.split(":")[1])
        )

        if inicio < ini_int and fim > ini_int:
            fim += (fim_intervalo - ini_int)

    return fim

def parse_horario(horario_str: str) -> Optional[Agenda]:

    match = re.match(r"(\d)\.(\d{4})-(\d)", horario_str)

    if not match:
        return None

```

```

    dia = int(match.group(1))
    hora = match.group(2)
    duracao = int(match.group(3))

    inicio = datetime.strptime(hora, "%H%M")

    fim = calcular_fim_com_intervalo(inicio, duracao)

    return Agenda(
        dia=dia,
        hora_inicio=inicio.strftime("%H:%M:%S"),
        hora_fim=fim.strftime("%H:%M:%S")
    )

def extrair_horarios(texto: str) -> List[Agenda]:

    horarios = re.findall(r"\d\.\d{4}-\d", texto)

    resultado = []

    for horario in horarios:

        parsed = parse_horario(horario)

        if parsed:
            resultado.append(parsed)

    return resultado

# PDF PARSER
def extrair_disciplinas_do_pdf(pdf_path: str) -> List[Turma]:

    turmas = []

    with pdfplumber.open(pdf_path) as pdf:

        texto_completo = ""

        for page in pdf.pages:

```

```

        texto = page.extract_text()

        if texto:
            texto_completo += texto + "\n"

linhas = texto_completo.split("\n")

linhas_processadas = []

# JUNTA LINHAS QUEBRADAS
i = 0

while i < len(linhas):

    linha = linhas[i].strip()

    if not linha:
        i += 1
        continue

    # Detecta início disciplina
    if re.match(r"^[A-Z]{3}\d{4}", linha):

        linha_completa = line = linha

        j = i + 1

        while j < len(linhas):

            prox = linhas[j].strip()

            # Nova disciplina
            if re.match(r"^[A-Z]{3}\d{4}", prox):
                break

            # Cabeçalhos
            if prox.startswith("CADASTRO"):
                break

            if prox.startswith("SeTIC"):
                break

```

```

        linha_completa += " " + prox

        j += 1

    linhas_processadas.append(linha_completa)

    i = j

else:
    i += 1

# PARSE DISCIPLINAS
for linha in linhas_processadas:

    try:

        partes = linha.split()

        codigo_disciplina = partes[0]
        turma = partes[1]

        # ENCONTRA CARGA HORÁRIA (Adicionado 0 na lista)
        workload_index = None

        for idx, parte in enumerate(partes):

            if parte.isdigit():

                possible = int(parte)

                if possible in [0, 18, 36, 72, 74, 108, 126, 432]:
                    workload_index = idx
                    break

        if workload_index is None:
            print(f"ERRO workload: {linha}")
            continue

        # NOME DISCIPLINA
        nome_disciplina = " ".join(
            partes[2:workload_index]

```

```

    )

    # HORÁRIOS
    agendas = extrair_horarios(linha)

    turma_obj = Turma(
        semestre=SEMESTRE,

        codigo_disciplina=codigo_disciplina,
        turma=turma,
        nome_disciplina=nome_disciplina,

        agendas=agendas
    )

    turmas.append(turma_obj)

except Exception as e:

    print("\nERRO AO PROCESSAR:")
    print(linha)
    print(e)

return turmas

def gerar_sql(
    turmas: List[Turma],
    output_sql: str
):

    sql = []

    # TABELAS
    sql.append("""
CREATE TABLE IF NOT EXISTS semestres (
    id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    semestre VARCHAR(10) UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS turmas (
    id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

```

```

        created_at TIMESTAMP DEFAULT NOW(),

        semestre_id INTEGER NOT NULL
            REFERENCES semestres(id),

        curriculo_disciplina INTEGER NOT NULL
            REFERENCES curriculo(id),

        turma VARCHAR(20) NOT NULL
    );

CREATE TABLE IF NOT EXISTS turmas_agenda (
    id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    turma_id INTEGER NOT NULL
        REFERENCES turmas(id),

    dia INTEGER NOT NULL,

    hora_inicio TIME NOT NULL,
    hora_fim TIME NOT NULL
);

CREATE UNIQUE INDEX IF NOT EXISTS unique_turma
ON turmas (
    semestre_id,
    curriculo_disciplina,
    turma
);

CREATE INDEX IF NOT EXISTS idx_curriculo_codigo
ON curriculo(codigo_disciplina);

CREATE INDEX IF NOT EXISTS idx_turmas_semestre
ON turmas(semestre_id);

CREATE INDEX IF NOT EXISTS idx_turmas_agenda_turma
ON turmas_agenda(turma_id);

CREATE INDEX IF NOT EXISTS idx_turmas_agenda_dia
ON turmas_agenda(dia);

```

```

ALTER TABLE semestres DISABLE ROW LEVEL SECURITY;
ALTER TABLE turmas DISABLE ROW LEVEL SECURITY;
ALTER TABLE turmas_agenda DISABLE ROW LEVEL SECURITY;
""")

# SEMESTRE
sql.append(f"""
INSERT INTO semestres (semestre)
VALUES ('{SEMESTRE}')
ON CONFLICT (semestre) DO NOTHING;
""")

# INSERTS
sql.append(f"""
DO $$
DECLARE
    v_semestre_id INTEGER;
    v_turma_id INTEGER;
    r_curr RECORD;
BEGIN

SELECT id
INTO v_semestre_id
FROM semestres
WHERE semestre = '{SEMESTRE}';
""")

for turma in turmas:

    codigo_disciplina = escape_sql(
        turma.codigo_disciplina
    )

    nome_turma = escape_sql(turma.turma)

    nome_disciplina = escape_sql(
        turma.nome_disciplina
    )

```

```

        # Alterado para criar a disciplina padrão (2026) caso não exista
nenhuma referência no banco

        sql.append(f"""
        -- Garantir que a disciplina exista em pelo menos um currículo
padrão (2026)

        IF NOT EXISTS (SELECT 1 FROM curriculo WHERE codigo_disciplina =
'{codigo_disciplina}') THEN
            INSERT INTO curriculo (ano_curriculo, fase, nome_disciplina,
carga_horaria, codigo_disciplina, tipo, area)
            VALUES (2026, 0, '{nome_disciplina}', 0, '{codigo_disciplina}',
'Op', 0);
        END IF;

        -- PERMANENTE: Loop para iterar em TODOS os currículos associados a
este código (Ex: 2016 e 2026)
        FOR r_curr IN (SELECT id FROM curriculo WHERE codigo_disciplina =
'{codigo_disciplina}') LOOP

            v_turma_id := NULL;

            -- Busca se a turma já existe para este currículo específico
            SELECT id INTO v_turma_id
            FROM turmas
            WHERE semestre_id = v_semestre_id
            AND curriculo_disciplina = r_curr.id
            AND turma = '{nome_turma}';

            -- Se a turma não existir para este currículo, insere
            IF v_turma_id IS NULL THEN
                INSERT INTO turmas (semestre_id, curriculo_disciplina,
turma)
                VALUES (v_semestre_id, r_curr.id, '{nome_turma}')
                RETURNING id INTO v_turma_id;
            END IF;

            -- Insere as agendas correspondentes ligadas à turma deste
currículo (Evitando duplicados na agenda)
            """)

            for agenda in turma.agendas:
                sql.append(f"""
                IF NOT EXISTS (

```

```

        SELECT 1 FROM turmas_agenda
        WHERE turma_id = v_turma_id AND dia = {agenda.dia} AND
hora_inicio = '{agenda.hora_inicio}'
    ) THEN
        INSERT INTO turmas_agenda (turma_id, dia, hora_inicio,
hora_fim)
        VALUES (v_turma_id, {agenda.dia}, '{agenda.hora_inicio}',
'{agenda.hora_fim}');
    END IF;
"""

    sql.append("""
END LOOP;
""")

    sql.append("""
END $$;
""")

    with open(output_sql, "w", encoding="utf-8") as f:
        f.write("\n".join(sql))

    print(f"\nSQL gerado com sucesso: {output_sql}")

if __name__ == "__main__":

    turmas = extrair_disciplinas_do_pdf(PDF_PATH)

    print(f"\nTotal de seções encontradas: {len(turmas)}\n")

    for s in turmas[:90]:

        print("=" * 60)
        print(f"DISCIPLINA: {s.codigo_disciplina}")
        print(f"TURMA: {s.turma}")
        print(f"NOME: {s.nome_disciplina}")

        for agenda in s.agendas:

            print(
                f"    DIA {agenda.dia} | "

```

```
        f"{agenda.hora_inicio} -> "  
        f"{agenda.hora_fim}"  
    )  
  
    gerar_sql(turmas, OUTPUT_SQL)
```

APÊNDICE D – Código fonte *supabase_conn.py*

```
import os
import streamlit as st
from supabase import create_client, Client

SUPABASE_URL = st.secrets["SUPABASE_URL"]
SUPABASE_KEY = st.secrets["API_KEY"]

supabase: Client = create_client(
    SUPABASE_URL,
    SUPABASE_KEY
)

def get_curriculo(
    ano_curriculo: int
):
    """
    Seleciona todas as colunas da tabela CURRICULO, filtrando pelo ANO
    CURRICULO,
    ordenando primeiro por TIPO (Ob antes de Op) e depois por FASE.
    """

    response = (
        supabase
        .table("curriculo")
        .select("*")
        .eq("ano_curriculo", ano_curriculo)
        .order("tipo", desc=False)
        .order("fase", desc=False)
        .execute()
    )

    return response.data

def get_horarios(codigos_disciplinas: list, ano_curriculo):

    response = (
        supabase
```

```
.table("curriculo")
.select("""
    codigo_disciplina,
    nome_disciplina,
    fase,
    tipo,
    area,
    carga_horaria,
    turmas:turmas!inner(
        id,
        semestre_id,
        turmas_agenda!inner(
            dia,
            hora_inicio,
            hora_fim
        )
    )
""")
.eq("ano_curriculo", ano_curriculo)
.in_("codigo_disciplina", codigos_disciplinas)
.execute()
)

return response.data
```

APÊNDICE E – Código fonte *rec_sys.py*

```

from datetime import datetime
from functools import partial

DIA_MAP = {
    2: "Segunda",
    3: "Terça",
    4: "Quarta",
    5: "Quinta",
    6: "Sexta",
}

TURNOS = {
    "Todas as manhãs": (7 * 60 + 30, 12 * 60),
    "Todas as tardes": (13 * 60 + 30, 17 * 60 + 30),
    "Todas as noites": (18 * 60 + 30, 22 * 60),
}

AREA_PREFERENCIA = {
    "Tecnologia da Informação": 1,
    "Gestão da Informação": 2,
}

def int_ou_zero(valor):
    try:
        return int(valor)
    except (TypeError, ValueError):
        return 0

def eh_obrigatoria(disc):
    tipo = str(disc.get("tipo", "")).strip().lower()
    fase = int_ou_zero(disc.get("fase"))

    return fase != 0 or tipo in {"ob", "obrigatoria", "obrigatória"}

def parse_time(t: str) -> int:
    """
    Converte '18:30:00' -> minutes since 00:00
    """

```

```

h, m, _ = t.split(":")
return int(h) * 60 + int(m)

def interval_overlap(a_start, a_end, b_start, b_end) -> bool:
    """
    Returns True if two time intervals overlap
    """
    return not (a_end <= b_start or a_start >= b_end)

def turno_bloqueado(turma, options_turnos) -> bool:
    if not options_turnos:
        return False

    for turno in options_turnos:
        if turno not in TURNOS:
            continue

        block_start, block_end = TURNOS[turno]

        for aula in turma.get("turmas_agenda", []):
            start = parse_time(aula["hora_inicio"])
            end = parse_time(aula["hora_fim"])

            if interval_overlap(start, end, block_start, block_end):
                return True

    return False

def horario_bloqueado(turma, horarios_bloqueados) -> bool:
    if not horarios_bloqueados:
        return False

    for h in horarios_bloqueados:
        try:
            parts = [x.strip() for x in h.split("|")]

            dia_txt = parts[0].strip().lower()

            # CASO 1: DIA BLOQUEADO

```

```

        if len(parts) == 1:
            for aula in turma.get("turmas_agenda", []):
                print(aula)
                dia = DIA_MAP.get(aula["dia"], "").lower()

                if dia == dia_txt:
                    return True

        # CASO 2: DIA E HORÁRIO BLOQUEADO
        else:
            horas = parts[1]
            inicio_str, fim_str = [x.strip() for x in
horas.split("-")]

            inicio = parse_time(inicio_str + ":00")
            fim = parse_time(fim_str + ":00")

            for aula in turma.get("turmas_agenda", []):
                dia = DIA_MAP.get(aula["dia"], "").lower()

                if dia != dia_txt:
                    continue

                start = parse_time(aula["hora_inicio"])
                end = parse_time(aula["hora_fim"])

                if interval_overlap(start, end, inicio, fim):
                    return True

    except Exception:
        continue

    return False

def filtrar_disciplinas(horarios, options_turnos=None,
horarios_bloqueados=None):
    """
    Retorna apenas disciplinas + turmas válidas
    após restrições.

    Regra especial:

```

```

- CIN7505 (Est. Obrig.) só pode aparecer caso NÃO exista nenhuma
  disciplina das fases 1, 2, 3 ou 4 disponível.
"""

resultado = []

for disciplina in horarios:
    disciplinas_validas = {
        "codigo_disciplina": disciplina["codigo_disciplina"],
        "nome_disciplina": disciplina["nome_disciplina"],
        "fase": disciplina["fase"],
        "tipo": disciplina["tipo"],
        "area": disciplina.get("area"),
        "carga_horaria": disciplina.get("carga_horaria", 0),
        "turmas": []
    }

    for turma in disciplina.get("turmas", []):
        if turno_bloqueado(turma, options_turnos):
            continue

        if horario_bloqueado(turma, horarios_bloqueados):
            continue

        disciplinas_validas["turmas"].append(turma)

    if disciplinas_validas["turmas"]:
        resultado.append(disciplinas_validas)

# REGRA ESPECIAL DA CIN7505
existe_nucleo_comum = any(
    d["fase"] in [1, 2, 3, 4]
    for d in resultado
    if d["codigo_disciplina"] != "CIN7505"
)

if existe_nucleo_comum:
    resultado = [
        d for d in resultado
        if d["codigo_disciplina"] != "CIN7505"
    ]

```

```

print(resultado)
return resultado

def possui_conflito_horario(turma1, turma2):
    for a1 in turma1["turmas_agenda"]:
        a1_start = parse_time(a1["hora_inicio"])
        a1_end = parse_time(a1["hora_fim"])

        for a2 in turma2["turmas_agenda"]:
            a2_start = parse_time(a2["hora_inicio"])
            a2_end = parse_time(a2["hora_fim"])

            if interval_overlap(a1_start, a1_end, a2_start, a2_end):
                if a1["dia"] == a2["dia"]:
                    return True

    return False

def prioridade_disciplina(disc, area_preferida=None):
    obrigatoria = eh_obrigatoria(disc)
    area = int_ou_zero(disc.get("area"))
    area_match = 0 if area_preferida is None or area == area_preferida
    else 1

    fase = int_ou_zero(disc.get("fase"))

    return (
        0 if obrigatoria else 1,
        fase if obrigatoria else area_match,
        0 if obrigatoria else fase,
        disc.get("nome_disciplina", "")
    )

def selecionar_grade(disciplinas, tipo_opt="Indiferente",
max_optativas_horas=0):
    """
    Seleção:
    - Obrigatórias têm prioridade
    - Fases obrigatórias mais iniciais têm prioridade

```

```

- Optativas respeitam a preferência de área
- Sem conflito de horários
- Limite máximo geral de 468 horas
"""

MAX_HORAS_GERAL = 468 # Configuração do limite máximo geral

area_preferida = AREA_PREFERENCIA.get(tipo_opt)

disciplinas = sorted(
    disciplinas,
    key=partial(prioridade_disciplina,
area_preferida=area_preferida)
)

selecionadas = []
turmas_escolhidas = []
optativas_horas = 0
total_horas = 0 # Controla o total de horas da grade selecionada

for disc in disciplinas:
    obrigatoria = eh_obrigatoria(disc)
    area = int_ou_zero(disc.get("area"))
    carga_horaria = int_ou_zero(disc.get("carga_horaria"))

    # 1. VALIDAÇÃO DO LIMITE GERAL (468h)
    # Se adicionar esta disciplina estourar o limite geral, passamos
para a próxima (ou encerramos)
    if total_horas + carga_horaria > MAX_HORAS_GERAL:
        continue

    # 2. VALIDAÇÃO DAS OPTATIVAS
    if not obrigatoria:
        if area == 0:
            continue

        if max_optativas_horas <= 0:
            continue

        if optativas_horas + carga_horaria > max_optativas_horas:
            continue

```

```
melhor_turma = None

for turma in disc["turmas"]:
    conflito = False

    for t in turmas_escolhidas:
        if possui_conflito_horario(turma, t):
            conflito = True
            break

    if not conflito:
        melhor_turma = turma
        break

if melhor_turma:
    selecionadas.append({
        "codigo_disciplina": disc["codigo_disciplina"],
        "nome_disciplina": disc["nome_disciplina"],
        "fase": disc["fase"],
        "tipo": disc["tipo"],
        "carga_horaria": carga_horaria,
        "area": disc.get("area"),
        "turma_escolhida": melhor_turma
    })

    turmas_escolhidas.append(melhor_turma)

    # Atualiza os acumuladores de horas
    total_horas += carga_horaria
    if not obrigatoria:
        optativas_horas += carga_horaria

return selecionadas
```

APÊNDICE F – Código fonte *disciplinas_handler.py*

```
# utils/disciplinas_handler.py

import pandas as pd

EQUIVALENCIAS_2016 = {
    "CIN7410": {
        "equivale": ["CIN7408", "CIN7307"],
        "conta_optativa": False
    },
    "CIN7936": {
        "equivale": ["CIN7305"],
        # só conta como optativa se NÃO estiver sendo usada como
        # equivalência obrigatória
        "conta_optativa_condicional": True
    },
    # Caso as disciplinas estejam com o código novo ao invés do antigo
    # no cadastro de turmas
    "CIN7111": {
        "equivale": ["CIN7445"],
        "conta_optativa_condicional": False
    },
    "CIN7112": {
        "equivale": ["CIN7603"],
        "conta_optativa_condicional": False
    },
    "CIN7000": {
        "equivale": ["CIN7408", "CIN7307", 'CIN7410'],
        "conta_optativa": False
    }
}

EQUIVALENCIAS_2026 = {
    "CIN7410": {
        "equivale": ["CIN7000"],
        "conta_optativa": False
    },
    "CIN7445": {
        "equivale": ["CIN7111"],
        "conta_optativa": False
    },
    "CIN7603": {
```

```

        "equivale": ["CIN7112"],
        "conta_optativa": False
    }
}

def equivalencias_2016(
    df: pd.DataFrame,
    codigos_concluidos: set
) -> pd.DataFrame:
    """
    Remove disciplinas equivalentes
    caso o aluno já tenha concluído
    a disciplina principal.
    """

    codigos_remove = set()

    for principal, regra in EQUIVALENCIAS_2016.items():

        equivalentes = regra["equivale"]

        if principal in codigos_concluidos:
            codigos_remove.update(equivalentes)

    df_filtrado = df[
        ~df["codigo_disciplina"].isin(codigos_remove)
    ].copy()

    return df_filtrado

def equivalencias_2026(
    df: pd.DataFrame,
    codigos_concluidos: set
) -> pd.DataFrame:
    """
    Remove disciplinas equivalentes
    caso o aluno já tenha concluído
    a disciplina principal.
    """

    codigos_remove = set()

```

```

for principal, regra in EQUIVALENCIAS_2026.items():

    equivalentes = regra["equivale"]

    if principal in codigos_concluidos:
        codigos_remove.update(equivalentes)

df_filtrado = df[
    ~df["codigo_disciplina"].isin(codigos_remove)
].copy()

return df_filtrado

def calcular_optativas(
    df_aprovadas,
    df_cursando,
    ano_curriculo
):

    df_disciplinas = pd.concat(
        [df_aprovadas, df_cursando],
        ignore_index=True
    )

    df_disciplinas = df_disciplinas.drop_duplicates(
        subset=["codigo_disciplina"]
    )

    disciplinas = df_disciplinas["codigo_disciplina"].tolist()

    if ano_curriculo == "2016":

        # CIN7410 conta como obrigatória
        df_disciplinas = df_disciplinas[
            df_disciplinas["codigo_disciplina"] != "CIN7410"
        ]

        # CIN7936 só conta se CIN7305 tiver sido cursada
        if "CIN7936" in disciplinas and "CIN7305" not in disciplinas:
            df_disciplinas = df_disciplinas[

```

```
        df_disciplinas["codigo_disciplina"] != "CIN7936"
    ]

    # somente disciplinas não obrigatórias
    df_optativas = df_disciplinas[
        df_disciplinas["tipo"] != "Ob"
    ]

    total_horas = df_optativas["carga_horaria"].sum()

    if ano_curriculo == "2016":
        horas_restantes = max(0, 576 - total_horas)

    elif ano_curriculo == "2026":
        horas_restantes = max(0, 540 - total_horas)

    else:
        horas_restantes = 0

    return {
        "codigo_disciplina": "OPT",
        "nome_disciplina": "OPTATIVAS",
        "fase": 0,
        "tipo": "Op/EX",
        "area": 0,
        "carga_horaria": horas_restantes,
    }
```

APÊNDICE G – Código fonte *timetable_handler.py*

```

import pandas as pd
from datetime import time

SLOTS_UFSC = [
    # Manhã
    (time(7, 30), time(8, 20)),
    (time(8, 20), time(9, 10)),
    (time(9, 10), time(10, 0)),
    (time(10, 10), time(11, 0)),
    (time(11, 0), time(11, 50)),
    # Tarde
    (time(13, 30), time(14, 20)),
    (time(14, 20), time(15, 10)),
    (time(15, 10), time(16, 0)),
    (time(16, 20), time(17, 10)),
    (time(17, 10), time(18, 0)),
    # Noite
    (time(18, 30), time(19, 20)),
    (time(19, 20), time(20, 10)),
    (time(20, 20), time(21, 10)),
    (time(21, 10), time(22, 0)),
]

DIA_MAP_INV = {
    2: "Segunda", 3: "Terça", 4: "Quarta",
    5: "Quinta", 6: "Sexta"
}

def str_para_time(txt):
    return time(int(txt.split(":")[0]), int(txt.split(":")[1]))

def formatar_rotulo(ini_str):
    if ini_str == "11:00":
        return "11:50"
    if ini_str == "17:10":
        return "17:10\n18:00"
    if ini_str == "21:10":
        return "22:00"
    return ini_str

def build_timetable(final_grade):

```

```

rows = []
for disc in final_grade:
    turma = disc["turma_escolhida"]
    for aula in turma["turmas_agenda"]:
        aula_inicio = str_para_time(aula["hora_inicio"])
        aula_fim = str_para_time(aula["hora_fim"])
        dia_semana = DIA_MAP_INV[aula["dia"]]

        for slot_ini, slot_fim in SLOTS_UFSC:
            if aula_inicio <= slot_ini and aula_fim >= slot_fim:
                ini_str = slot_ini.strftime("%H:%M")
                rows.append({
                    "dia": dia_semana,
                    "slot": formatar_rotulo(ini_str),
                    "disciplina": disc["codigo_disciplina"]
                })
    return pd.DataFrame(rows)

def render_timetable(df):
    if df.empty:
        return pd.DataFrame()

    timetable = df.pivot_table(
        index="slot",
        columns="dia",
        values="disciplina",
        aggfunc="first"
    )

    # Gera a estrutura de ordenação aplicando a nova regra de rótulos
    ordem_linhas = [formatar_rotulo(ini.strftime("%H:%M")) for ini, _ in SLOTS_UFSC]
    ordem_colunas = ["Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado"]

    return timetable.reindex(index=ordem_linhas,
                             columns=ordem_colunas).fillna("")

def int_ou_zero(valor):
    try:
        return int(valor)
    
```

```

except (TypeError, ValueError):
    return 0

def montar_recomendacao(df_para_cursar, gerar_opt):
    fases = df_para_cursar["fase"].apply(int_ou_zero)
    if "area" in df_para_cursar.columns:
        areas = df_para_cursar["area"].apply(int_ou_zero)
    else:
        areas = df_para_cursar["fase"].apply(lambda _: 0)
    obrigatorias = df_para_cursar[
        (fases != 0) &
        (df_para_cursar["codigo_disciplina"] != "OPT")
    ]

    codigos = obrigatorias["codigo_disciplina"].tolist()
    max_optativas_horas = 0

    optativas_restantes = df_para_cursar[
        df_para_cursar["codigo_disciplina"] == "OPT"
    ]

    horas_optativas_restantes = 0
    if not optativas_restantes.empty:
        horas_optativas_restantes = int_ou_zero(
            optativas_restantes.iloc[0]["carga_horaria"]
        )

    tem_optativas_restantes = horas_optativas_restantes > 0

    if gerar_opt == "Sim" and tem_optativas_restantes:
        optativas = df_para_cursar[
            (fases == 0) &
            (areas != 0) &
            (df_para_cursar["codigo_disciplina"] != "OPT") &
            (df_para_cursar["tipo"] != "Ob")
        ]

        codigos += optativas["codigo_disciplina"].tolist()
        max_optativas_horas = horas_optativas_restantes

```

```
return list(dict.fromkeys(codigos)), max_optativas_horas
```

APÊNDICE H – Código fonte *app.py*

```

import sys
import time
from pathlib import Path

ROOT_DIR = Path(__file__).resolve().parent.parent
sys.path.append(str(ROOT_DIR))

import streamlit as st
import pandas as pd

from pdf_parser.historico_pdf_parser import extrair_disciplinas
from db_conn.supabase_conn import get_curriculo, get_horarios
from recsys.rec_sys import filtrar_disciplinas, selecionar_grade
from utils.disciplinas_handler import equivalencias_2016,
equivalencias_2026, calcular_optativas
from utils.timetable_handler import build_timetable,
render_timetable, montar_recomendacao

st.set_page_config(
    page_title="Sistema de Recomendação de Grade Curricular",
    page_icon="📖",
    layout="wide"
)

st.image("data/capa_cinfo.png")

st.title("Sistema de Recomendação de Grade Curricular")

st.subheader("Faça o upload do Controle Curricular")
pdf_curriculo = st.file_uploader(
    "Anexe o PDF",
    type=["pdf"]
)

st.subheader("Curriculo")
ano_curriculo = st.radio("Selecione o ano do seu curriculo",
    ["2016", "2026"],
)

if pdf_curriculo:

```

```

disciplinas_df = extrair_disciplinas(pdf_curriculo)

# Separação por status
aprovadas = [
    d for d in disciplinas_df
    if d["status"] == "APROVADO"
]

cursando = [
    d for d in disciplinas_df
    if d["status"] == "CURSANDO"
]

df_aprovadas_raw = pd.DataFrame(aprovadas)
df_cursando_raw = pd.DataFrame(cursando)

# APROVADAS
try:
    df_aprovadas = df_aprovadas_raw[["codigo_disciplina",
"nome_disciplina", "nota", "carga_horaria", "tipo" ]]
except:
    df_aprovadas = pd.DataFrame(columns=["codigo_disciplina",
"nome_disciplina", "nota", "carga_horaria", "tipo" ])

# CURSANDO
df_cursando = df_cursando_raw[["codigo_disciplina",
"nome_disciplina", "carga_horaria", "tipo"]]

# união dos códigos APROVADAS + CURSANDO
codigos_excluir = set(
    df_aprovadas["codigo_disciplina"].tolist() +
    df_cursando["codigo_disciplina"].tolist()
)

curriculo = get_curriculo(ano_curriculo=ano_curriculo)

# PARA CURSAR
df_cursar = pd.DataFrame([
    {
        "codigo_disciplina": d["codigo_disciplina"],
        "nome_disciplina": d["nome_disciplina"],
        "fase": d["fase"],

```

```

        "tipo": d["tipo"],
        "area": d.get("area", 0),
        "carga_horaria": d["carga_horaria"],
    }
    for d in curriculo
    if (
        d["codigo_disciplina"] not in codigos_excluir
    )
])

if ano_curriculo == '2016':
    df_cursar = equivalencias_2016(
        df_cursar,
        codigos_excluir
    )

if ano_curriculo == '2016':
    df_cursar = equivalencias_2026(
        df_cursar,
        codigos_excluir
    )

# linha de optativas
row_optativas = calcular_optativas(
    df_aprovadas,
    df_cursando,
    ano_curriculo
)

# adiciona ao dataframe
df_cursar = pd.concat(
    [df_cursar, pd.DataFrame([row_optativas])],
    ignore_index=True
)

# COLUNAS DAS DISCIPLINAS
disc_col1, disc_col2, disc_col3 = st.columns(3)

with disc_col1:
    st.subheader("APROVADAS")

```

```

        st.dataframe(
            df_aprovadas,
            hide_index=True,
            width="stretch"
        )

    with disc_col2:
        st.subheader("CURSANDO")

        st.dataframe(
            df_cursando,
            hide_index=True,
            width="stretch"
        )

    with disc_col3:
        st.subheader("PARA CURSAR")
        df_para_cursar = pd.DataFrame(
            df_cursar
        ).sort_values("fase")

        st.dataframe(
            df_para_cursar[(df_para_cursar["fase"] != 0) |
(df_para_cursar["codigo_disciplina"] == "OPT")],
            hide_index=True,
            width="stretch"
        )

    st.divider()

    # COLUNAS DAS RESTRIÇÕES
    rest_col1, rest_col2, rest_col3 = st.columns(3)

    with rest_col1:

        st.subheader("Restrições de Turnos")

        options_turnos = st.multiselect(
            "Turnos que você não pode ter aula",
            [
                "Todas as manhãs",

```

```

        "Todas as tardes",
        "Todas as noites"
    ],
    placeholder="Caso não haja restrição, pule esta etapa"
)

with rest_col2:

    st.subheader("Restrições de Horários")

    horarios_ufsc = [
        "07:30 - 08:20",
        "08:20 - 09:10",
        "09:10 - 10:00",
        "10:10 - 11:00",
        "11:00 - 11:50",
        "11:50 - 12:40",

        "13:30 - 14:20",
        "14:20 - 15:10",
        "15:10 - 16:00",
        "16:20 - 17:10",
        "17:10 - 18:00",
        "18:00 - 18:50",

        "18:30 - 19:20",
        "19:20 - 20:10",
        "20:20 - 21:10",
        "21:10 - 22:00",
    ]

    dias_semana = [
        "Segunda",
        "Terça",
        "Quarta",
        "Quinta",
        "Sexta",
    ]

    opcoes_horarios = [
        f"{dia} | {horario}"
        for dia in dias_semana

```

```

        for horario in horarios_ufsc
    ]

    horarios_bloqueados = st.multiselect(
        "Horários que você não pode ter aula",
        options=dias_semana + opcoes_horarios,
        placeholder="Caso não haja restrição, pule esta etapa"
    )

    with rest_col3:
        st.subheader("Optativas")
        gerar_opt = st.radio("Gerar grade com optativas?",
            ["Sim", "Não, apenas disciplinas obrigatórias"],
        )

        if gerar_opt == "Sim":
            tipo_opt = st.radio("Preferência por...",
                ["Tecnologia da Informação", "Gestão da Informação",
                "Indiferente"],
            )
        else:
            tipo_opt = "Indiferente"

    st.divider()

    st.header("Gere sua recomendação de grade curricular")
    botao = st.button("Gerar!", type="primary")

    if botao:
        codigos, max_optativas_horas =
montar_recomendacao(df_para_cursar, gerar_opt)
        print('Disciplinas Para Cursar:', codigos)
        horarios = get_horarios(codigos, ano_curriculo)

        print('\n Horarios:', horarios)

        filtered = filtrar_disciplinas(
            horarios,
            options_turnos=options_turnos,
            horarios_bloqueados=horarios_bloqueados
        )

```

```
final_grade = selecionar_grade(
    filtered,
    tipo_opt=tipo_opt,
    max_optativas_horas=max_optativas_horas
)

df_tt = build_timetable(final_grade)
tt_view = render_timetable(df_tt)

st.subheader("Sua Grade de Horários")
st.dataframe(tt_view, use_container_width=True)

st.subheader("Disciplinas recomendadas")
for disc in final_grade:
    st.markdown(
        f"{disc['codigo_disciplina']}, "
        f"{disc['nome_disciplina']}, "
        f"{disc['tipo']}, "
        f"{disc['carga_horaria']}"
    )
```