Relatório do trabalho de Arquitetura e Desenho de Software

Disciplina: Arquitetura e Desenho de Software

Docente: Alexandre Valente Conceição Pereira Sousa

Aluna: Ileana Karla Antunes de Souza, Nº A046152

Tema do trabalho: Sistema de Gestão de Avaliações Escolares

Ano Letivo: 2025/2026

Índice

1. Introdução

A gestão de avaliações escolares constitui um dos processos mais sensíveis e relevantes no contexto educacional. O cálculo de médias ponderadas, a consideração de diferentes dimensões avaliativas e a atribuição de menções qualitativas exigem rigor metodológico e precisão técnica. Ao longo da minha prática docente e acadêmica, percebi que muitos erros na consolidação de notas decorrem de cálculos manuais ou de planilhas pouco estruturadas, o que compromete a confiabilidade dos resultados finais e pode gerar inconsistências administrativas.

Diante desse cenário, desenvolvi um sistema informatizado voltado especificamente para a disciplina de Tecnologias da Informação e Comunicação (TIC), estruturado por períodos letivos (1.º, 2.º e 3.º períodos). O objetivo central foi automatizar o cálculo das notas cognitivas, integrar a avaliação das atitudes e gerar o boletim por período de forma automática e consistente. O sistema foi concebido para funcionar inicialmente por meio do **Django Admin**, permitindo rápida validação funcional e foco na lógica de negócio.

Além da automatização do cálculo, preocupei-me em garantir que os dados fossem validados segundo regras rígidas, tais como limites percentuais de atitudes, intervalo permitido para notas e integridade relacional entre aluno e turma. O sistema foi estruturado para que o boletim não seja inserido manualmente, mas sim **gerado automaticamente** sempre que o professor lança notas das avaliações e/ou atitudes, reduzindo a possibilidade de erro humano.

Este projeto não se limita a uma implementação técnica; ele representa a aplicação prática de princípios de arquitetura de *software*, integridade de dados e automação orientada a eventos. Ao longo do desenvolvimento, enfrentei desafios técnicos relacionados à modelagem, migrações, recálculo automático e sincronização de dados, os quais contribuíram significativamente para o amadurecimento da solução final. O presente relatório descreve detalhadamente todas as etapas, decisões arquiteturais, módulos desenvolvidos e perspectivas futuras do sistema.

2. Enquadramento do Projeto e Requisitos

O desenvolvimento deste sistema surgiu da necessidade de estruturar formalmente o processo avaliativo da disciplina de TIC, considerando as especificidades do ensino regular. No modelo adotado, a avaliação é composta por duas grandes dimensões: domínio cognitivo (avaliado por testes, trabalhos ou projetos com peso percentual) e

domínio atitudinal (avaliado por critérios comportamentais com tetos máximos definidos). A nota final por período é obtida pela soma da componente cognitiva (até 80 pontos) e da componente de atitudes (até 20 pontos), totalizando 100 pontos.

Do ponto de vista dos requisitos funcionais, o sistema deveria permitir:

- Registo de ano letivo;
- Registo de turmas e alunos;
- Registo de avaliações cognitivas por turma e por período, com definição de peso percentual para cada avaliação incluída;
- Lançamento/Registo de notas individuais por aluno em cada avaliação;
- Registo das atitudes por período, respeitando limites máximos por dimensão;
- Geração automática do boletim por período, que tem por objetivo permitir a visualização dos dados para o professor, permitindo selecionar o estado (aberto ou fechado) do boletim de avaliações, permite incluir o nível de autoavaliação feita pelo aluno naquele período selecionado, e incluir observações;
- Cálculo automático da média ponderada e da nota final;
- Geração de menção qualitativa e nível SGE (escala de 1 a 5).

Além disso, foi estabelecido que o boletim não poderia ser criado manualmente pelo utilizador, mas deveria surgir **automaticamente** quando fossem lançadas notas ou atitudes. Também foi definido que campos calculados (nota cognitiva, atitudes, nota final, menção e nível) deveriam ser somente leitura, impedindo alterações diretas pelo utilizador.

Quanto aos requisitos não funcionais, priorizei a integridade dos dados, a consistência dos cálculos e a automatização por eventos. Para isso, utilizei validações no método **clean()** dos modelos, restrições de unicidade (**unique_together**), índices para otimização de consultas e atualização direta via ORM para evitar recursividade indesejada nos *signals*. A consistência transacional foi garantida com o uso de **transaction.atomic** e **transaction.on_commit**, assegurando que o recálculo do boletim ocorra somente após a confirmação da gravação dos dados.

Outro requisito essencial foi a clareza na visualização dos dados. No boletim, implementei tabelas detalhadas para exibição das atitudes e das avaliações cognitivas

com seus respectivos pesos e notas, incluindo o cálculo da média ponderada por período. Esse recurso permite que o professor consulte de forma transparente a composição da nota final, fortalecendo a rastreabilidade e a confiabilidade do sistema.

O projeto, portanto, foi concebido não apenas como uma aplicação técnica, mas como um instrumento de apoio pedagógico e administrativo, alinhado às práticas avaliativas da disciplina de TIC e fundamentado em princípios sólidos de engenharia de *software*.

3. Tecnologias Utilizadas e Justificativa das Escolhas

Para o desenvolvimento do sistema de gestão de avaliações, optei pela utilização da linguagem de programação **Python**, aliada ao framework **Django**, estruturado segundo o padrão arquitetural MVT (Model–View–Template). A escolha do Python fundamenta-se na sua legibilidade, clareza sintática e forte adoção no contexto educacional e acadêmico. Como docente da área de Informática, reconheço que Python oferece equilíbrio entre simplicidade e robustez, permitindo implementar regras de negócio complexas de forma organizada e compreensível.

O framework Django foi selecionado por oferecer uma arquitetura madura, baseada no princípio *"batteries included"*, ou seja, já fornece ferramentas prontas para autenticação, ORM, administração, validação de modelos e segurança. Esse conjunto de funcionalidades reduziu significativamente o tempo de desenvolvimento estrutural e permitiu concentrar esforços na lógica de negócio do sistema avaliativo. Além disso, o Django Admin foi utilizado como interface inicial de gestão, possibilitando validação rápida das funcionalidades antes da construção de uma interface personalizada.

A base de dados utilizada foi o **SQLite**, especialmente por se tratar de uma fase de desenvolvimento e prototipagem. O **SQLite** apresenta simplicidade de configuração e integração nativa com **Django**, sendo adequado para aplicações de pequeno e médio porte. Entretanto, o sistema foi estruturado de forma que possa futuramente migrar para um sistema de base de dados mais robusto, como **PostgreSQL**, sem necessidade de reformulação estrutural do código. A escolha pela base de dados no desenvolvimento deste trabalho está descrita no ponto "**3.1. Banco de Dados e Persistência**" do presente relatório.

Outro recurso tecnológico fundamental foi o uso de ***signals*** **(sinais do Django)** para automatização do recálculo do boletim. Sempre que uma nota ou atitude é inserida, alterada ou removida, o sistema dispara automaticamente uma função de recálculo. Para garantir consistência transacional e evitar recursividade, utilizei **transaction.atomic**

e **transaction.on_commit**, assegurando que o recálculo ocorra somente após a confirmação da gravação na base de dados.

Além disso, utilizei a biblioteca **decimal.Decimal** para garantir precisão matemática nos cálculos de médias ponderadas e conversões percentuais. A utilização de números decimais é essencial em sistemas avaliativos, pois evita erros de arredondamento típicos de números de ponto flutuante (float).

Por fim, durante o desenvolvimento, utilizei também Inteligência Artificial (IA) como ferramenta de apoio técnico, especialmente para análise de arquitetura, revisão de código e identificação de boas práticas. Entretanto, o uso da IA não substituiu o raciocínio crítico; diversas adaptações e correções foram necessárias para alinhar o código às regras específicas do sistema de Gestão de Avaliações Escolares implementado.

3.1. Banco de Dados e Persistência

No desenvolvimento do sistema de Gestão de Avaliações Escolares utilizei o banco de dados **SQLite**, que é o banco relacional padrão configurado automaticamente pelo framework **Django**. O **SQLite** é um sistema de gerenciamento de banco de dados leve, baseado em arquivo único, que não requer instalação de servidor externo, sendo ideal para ambientes de desenvolvimento, testes e prototipagem acadêmica. No contexto deste projeto, o arquivo **db.sqlite3** foi gerado automaticamente após a execução das migrações, armazenando todas as tabelas correspondentes aos modelos definidos na aplicação.

A escolha do **SQLite** foi fundamentada na simplicidade de configuração, integração nativa com o **Django** e adequação ao escopo do projeto. Como o sistema está sendo desenvolvido com finalidade acadêmica e não está, neste momento, em ambiente de produção com múltiplos acessos simultâneos, o **SQLite** atende plenamente às necessidades de persistência de dados. A utilização desse banco permitiu concentrar esforços na modelagem das entidades e nas regras de negócio, sem a necessidade de configuração adicional de infraestrutura.

A integração entre o banco de dados e o sistema foi realizada por meio do ORM (Object-Relational Mapping) do Django. O ORM permite que as classes definidas em **models.py** sejam automaticamente convertidas em tabelas no banco de dados, enquanto as relações entre modelos são traduzidas em chaves estrangeiras e restrições de integridade. O uso de migrações (**makemigrations** e **migrate**) garantiu a criação

controlada da estrutura física das tabelas, assegurando consistência entre o código-fonte e a base de dados.

Embora o **SQLite** seja adequado para desenvolvimento, reconheço que, em um cenário de produção real com múltiplos professores acessando simultaneamente o sistema, seria recomendável migrar para um sistema de gerenciamento de banco de dados mais robusto, como **PostgreSQL**. Essa migração seria facilitada pelo próprio **Django**, que permite alteração do banco apenas na configuração, mantendo os modelos e regras de negócio inalterados. Dessa forma, o projeto apresenta potencial de escalabilidade e evolução futura.

4. Arquitetura e Organização do Projeto

O sistema foi estruturado de forma modular, respeitando princípios de separação de responsabilidades e organização em camadas. A aplicação principal está organizada dentro da **app tic**, integrada à **app nucleo**, responsável pelas entidades estruturais como **Turma** e **Aluno**.

A arquitetura do projeto pode ser compreendida em quatro camadas principais:

1. **Modelos (models.py)** – Responsáveis pela definição da estrutura dos dados e regras de validação.
2. **Serviços (services/tic_calculator.py)** – Responsáveis pela lógica de cálculo e regras de negócio.
3. ***Signals* (signals.py)** – Responsáveis pela automatização dos recálculos.
4. **Admin (admin.py)** – Responsável pela interface administrativa e apresentação dos dados.

Essa divisão permite alta coesão interna e baixo acoplamento entre componentes. O modelo não contém regras complexas de cálculo; essas regras estão concentradas na camada de serviços. Os ***signals*** apenas orquestram quando o cálculo deve ser executado. O **admin** limita-se à apresentação e controle de permissões.

4.1. Modelos de Dados

O modelo **BoletimPeriodoTIC** representa o núcleo do sistema. Ele associa turma, aluno e período, armazenando os resultados calculados (nota cognitiva, atitudes, nota final, menção e nível SGE). O modelo utiliza **unique_together** para impedir duplicidade de

boletins para o mesmo aluno na mesma turma e período, garantindo integridade referencial.

O modelo **AtitudesPeriodoTIC** possui relação **OneToOne** com o boletim, armazenando as cinco dimensões avaliativas com tetos máximos definidos. A validação é realizada no método **clean()**, assegurando que nenhum valor ultrapasse os limites estabelecidos.

O modelo **AvaliacaoCognitivaTIC** define as avaliações por turma e período, contendo nome e peso percentual. A restrição de unicidade impede que existam duas avaliações com o mesmo nome no mesmo período da mesma turma.

O modelo **NotaAvaliacaoCognitivaTIC** armazena a nota do aluno em determinada avaliação. A restrição **unique_together** impede que o mesmo aluno receba duas notas para a mesma avaliação.

4.2. Camada de Serviços – Lógica de Cálculo

A lógica de cálculo foi isolada no módulo **tic_calculator.py**. Essa decisão arquitetural foi essencial para evitar acoplamento excessivo entre modelos e regras matemáticas. O serviço **calcular_resultado_boletim** executa:

- Cálculo da média ponderada das avaliações cognitivas (0–100);
- Conversão para escala 0–80;
- Soma das atitudes (0–20);
- Cálculo da nota final (0–100);
- Determinação da menção qualitativa;
- Determinação do nível SGE (1–5).

O método **recalcular_boletim** realiza atualização direta via ORM (**update()**), evitando chamada ao método **save()** e prevenindo recursão de *signals*.

4.3. Automatização com *Signals*

Os *signals* foram implementados para disparar automaticamente o recálculo do boletim sempre que:

- Uma nota é criada, alterada ou removida;

- Uma atitude é criada, alterada ou removida;
- O peso de uma avaliação é alterado.

Para evitar inconsistências transacionais, utilizei **transaction.on_commit**, garantindo que o cálculo ocorra apenas após confirmação da operação na base de dados.

4.4. Interface Administrativa

O Django Admin foi personalizado para:

- Tornar campos calculados somente leitura;
- Impedir criação manual de boletins;
- Impedir exclusão de boletins;
- Exibir tabelas detalhadas de atitudes e avaliações;
- Exibir média ponderada das avaliações cognitivas.

Essa abordagem garante que o professor visualize os resultados sem possibilidade de alterar diretamente campos que devem ser exclusivamente calculados pelo sistema.

5. Desenvolvimento dos Módulos e Análise Detalhada do Código-Fonte

Nesta seção apresento uma análise aprofundada dos módulos implementados no sistema, descrevendo não apenas sua função técnica, mas também as decisões arquiteturais envolvidas, os problemas enfrentados e as soluções adotadas. O objetivo é demonstrar que o desenvolvimento não se limitou à implementação funcional, mas envolveu reflexão crítica sobre organização, integridade de dados e coerência com a realidade escolar.

5.1. Módulo de Modelos (models.py)

O arquivo **models.py** constitui a base estrutural do sistema. Nele são definidas as entidades persistentes da aplicação, seus relacionamentos, restrições e validações.

5.1.1. Classe Periodo

A enumeração **Periodo** foi implementada utilizando **models.IntegerChoices**, permitindo restringir os valores possíveis do campo período para 1.º, 2.º e 3.º períodos. Essa escolha

elimina erros de inserção manual de valores inválidos (por exemplo, período 4 ou 0) e garante padronização.

Essa abordagem também facilita a utilização de caixas de seleção no Django Admin, melhorando a usabilidade e reduzindo inconsistências de dados.

5.1.2. Modelo BoletimPeriodoTIC

O modelo **BoletimPeriodoTIC** representa a entidade central do sistema avaliativo. Ele associa:

- Turma
- Aluno
- Período

E armazena os resultados calculados:

- Nota cognitiva (0–80)
- Pontuação de atitudes (0–20)
- Nota final (0–100)
- Menção qualitativa
- Nível SGE

A restrição: **unique_together = [("turma", "aluno", "periodo")]** garante que não exista mais de um boletim para o mesmo aluno, na mesma turma e período. Essa decisão foi essencial para evitar duplicidades que comprometeriam a integridade do sistema.

O método **clean()** implementa validações importantes:

- Confirma se o aluno pertence à turma selecionada;
- Garante que o período esteja entre 1 e 3;
- Valida a autoavaliação (nível entre 1 e 5).

Essa validação no nível do modelo garante consistência mesmo fora do Django Admin, como em execuções via scripts ou serviços.

5.1.3. Modelo AtitudesPeriodoTIC

Este modelo implementa as cinco dimensões do domínio "Saber Estar–Ser", cada uma com teto máximo específico:

- Responsabilidade e integridade (0–3)
- Excelência e exigência (0–6)
- Curiosidade, reflexão e inovação (0–2)
- Cidadania e participação (0–4)
- Liberdade (0–5)

A soma máxima totaliza 20 pontos.

A relação **OneToOne** com **BoletimPeriodoTIC** foi uma decisão arquitetural importante. Isso garante que exista apenas um conjunto de atitudes por boletim, reforçando integridade.

O método **clean()** valida:

- Limites individuais de cada dimensão;
- Soma total entre 0 e 20.

Além disso, o método **save()** chama **full_clean()** antes de salvar, garantindo validação automática em qualquer contexto de gravação.

5.1.4. Modelo AvaliacaoCognitivaTIC

Este modelo representa os instrumentos avaliativos do domínio cognitivo. Ele define:

- Turma
- Período
- Nome da avaliação
- Peso percentual

A restrição: **_together = [("turma", "periodo", "nome")]** impede duplicidade de avaliações com o mesmo nome no mesmo contexto.

A validação do peso percentual assegura que:

- O peso seja maior que 0;
- O peso não ultrapasse 100%.

A decisão de não obrigar que a soma dos pesos seja exatamente 100% permite maior flexibilidade ao professor, sendo a média ponderada calculada dinamicamente no momento do processamento.

5.1.5. Modelo NotaAvaliacaoCognitivaTIC

Este modelo associa um aluno a uma avaliação específica.

A restrição: **unique_together = [("avaliacao", "aluno")]** garante que o aluno não receba duas notas para a mesma avaliação.

A validação do campo nota_0a100 assegura que o valor esteja entre 0 e 100.

Essa estrutura permite cálculo da média ponderada de forma flexível e dinâmica.

5.2. Módulo de Serviços (tic_calculator.py)

A camada de serviços foi criada para centralizar toda a lógica de cálculo, separando regra de negócio da persistência.

Essa decisão foi fundamental para:

- Melhor organização arquitetural;
- Facilidade de manutenção;
- Possibilidade de testes unitários;
- Evitar lógica complexa dentro dos modelos.

5.2.1 Estrutura do Resultado

A classe **ResultadoTIC** foi implementada como @dataclass, tornando o retorno do cálculo estruturado e imutável. Essa abordagem aumenta clareza semântica e organização do código.

Ela encapsula:

- Nota cognitiva (0–80)
- Pontuação de atitudes (0–20)

- Nota final (0–100)
- Menção qualitativa
- Nível SGE

5.2.2 Lógica do Cálculo Cognitivo

O cálculo cognitivo segue as seguintes etapas:

1. Busca todas as notas do aluno para aquele período.
2. Calcula soma ponderada (nota × peso).
3. Divide pela soma dos pesos.
4. Converte resultado para escala 0–80.
5. Aplica arredondamento com Decimal.

O uso de Decimal foi essencial para evitar imprecisões matemáticas típicas de floats.

5.2.3 Lógica do Cálculo das Atitudes

A soma das atitudes respeita os tetos máximos definidos no modelo. Mesmo que o professor tente inserir valores maiores, a validação impede inconsistência.

O sistema calcula:

nota_final = cognitivo (0–80) + atitudes (0–20)

resultando em escala 0–100.

5.2.4 Persistência sem Recursão

O método **recalcular_boletim()** utiliza:

BoletimPeriodoTIC.objects.filter(pk=...).update(...)

e não save().

Essa decisão evita que *signals* sejam disparados novamente, prevenindo recursividade infinita.

5.3. Módulo de Signals

Os *signals* automatizam o recálculo do boletim sempre que há:

- Inclusão de nota
- Alteração de nota
- Exclusão de nota
- Alteração de atitudes
- Alteração de peso de avaliação

O uso de:

transaction.on_commit()

foi uma melhoria importante. Ele garante que o cálculo só ocorra após confirmação da gravação na base de dados.

Essa abordagem resolve problemas comuns de inconsistência transacional.

5.4. Módulo Admin

O **Django Admin** foi personalizado para:

- Impedir edição manual de campos calculados;
- Impedir criação manual de boletins;
- Impedir exclusão de boletins;
- Exibir tabelas detalhadas com médias.

A exibição da média ponderada diretamente na tabela administrativa melhora transparência e acompanhamento pedagógico.

6. Uso de Inteligência Artificial no Desenvolvimento do Projeto

O desenvolvimento deste sistema contou com o apoio de ferramentas de Inteligência Artificial (IA) como suporte técnico e estrutural. Contudo, é importante destacar que a IA foi utilizada como instrumento auxiliar, e não como substituta do processo intelectual e decisório do desenvolvimento. Todas as decisões arquiteturais, validações, regras de

negócio e ajustes estruturais foram analisados criticamente por mim antes de serem implementados.

A IA foi utilizada principalmente para:

- Estruturar versões iniciais de modelos Django;
- Sugerir organização de camadas (*models*, *services*, *signals*, *admin*);
- Auxiliar na correção de erros sintáticos;
- Apoiar na refatoração de código;
- Explicar conceitos técnicos como recursividade em *signals* e uso de **transaction.on_commit**.

Entretanto, diversas sugestões iniciais geradas pela IA apresentaram inconsistências, especialmente relacionadas a:

- Uso incorreto de nomes de campos (por exemplo, nota vs. nota_0a100);
- Inconsistência entre **related_name** em modelos e referências em **services**;
- Geração de lógica com risco de recursão infinita em *signals*;
- Falta de validações coerentes com as regras reais do sistema educativo;
- Estruturação incompleta da camada administrativa.

Em diversas situações, o código sugerido precisou ser ajustado manualmente, reorganizado ou mesmo reescrito integralmente para se adequar à realidade do projeto. Isso evidencia um ponto crucial: a IA pode acelerar o desenvolvimento, mas não substitui o raciocínio arquitetural, a compreensão das regras de negócio e a validação técnica rigorosa.

O uso da IA trouxe vantagens significativas:

- Redução de tempo na escrita inicial de estruturas repetitivas;
- Apoio na identificação de boas práticas;
- Sugestões de refatoração e modularização;
- Apoio didático na compreensão de conceitos avançados do Django.

Por outro lado, as desvantagens observadas incluem:

- Geração de código aparentemente correto, mas semanticamente incoerente;
- Necessidade constante de revisão crítica;
- Ajustes frequentes para adequação ao contexto real do projeto.

Portanto, neste trabalho, a IA foi utilizada como ferramenta de apoio estratégico, mas todo o desenvolvimento foi conduzido sob análise técnica, validação manual e decisão consciente de arquitetura.

7. Problemas Encontrados e Ajustes Realizados

Durante o desenvolvimento do sistema, diversos problemas técnicos e conceituais surgiram, exigindo ajustes estruturais importantes. A seguir apresento os principais desafios enfrentados e as soluções adotadas.

7.1. Problema de Recursão em Signals

Inicialmente, o recálculo do boletim utilizava **save()** dentro do próprio modelo, o que fazia com que os *signals* fossem disparados novamente, gerando um ciclo recursivo.

Esse problema poderia causar:

- Sobrecarga do sistema;
- Execução infinita de cálculos;
- Travamento da aplicação.

A solução adotada foi:

- Utilizar **update()** ao invés de **save()** na função **recalcular_boletim**;
- Implementar **transaction.on_commit()** para garantir consistência transacional.

Essa alteração representou uma melhoria arquitetural relevante, separando persistência de eventos automáticos.

7.2. Inconsistência entre Campos e Services

Em diversas ocasiões, nomes de campos no modelo não estavam sincronizados com a camada de serviços. Exemplos:

- nota vs. nota_0a100;
- peso vs. peso_percentual;
- boletim_periodo vs. boletim.

Essas inconsistências geravam erros como:

- AttributeError;
- Falhas de recálculo;
- Registros criados com erro aparente, mas persistidos na base.

A solução foi padronizar nomenclaturas e revisar todos os pontos de integração entre:

- models.py
- tic_calculator.py
- signals.py
- admin.py

Esse processo evidenciou a importância da coerência semântica em projetos de *software*.

7.3. Problemas de Integridade de Dados

Foram identificadas falhas como:

- Permissão de inserir períodos acima de 3;
- Possibilidade de duplicar boletins;
- Permissão de editar campos calculados manualmente;
- Exclusão indevida de boletins.

As soluções incluíram:

- Uso de **IntegerChoices** para o período;
- **unique_together** para boletins;
- **readonly_fields** no Django Admin;

- Bloqueio de **has_delete_permission**.

Esses ajustes tornaram o sistema mais robusto e alinhado com as regras do contexto escolar.

7.4. Reestruturação do Django Admin

Inicialmente, o **Admin** permitia edição indevida de campos calculados. Isso comprometia a integridade dos resultados.

Foi necessário:

- Tornar campos calculados somente leitura;
- Impedir criação manual de boletins;
- Criar tabelas HTML customizadas para exibição de avaliações;
- Inserir cálculo de média ponderada no rodapé das avaliações.

Esse processo exigiu reescrita integral da classe **BoletimPeriodoTICAdmin**.

8. Análise Crítica do Sistema Desenvolvido

O sistema desenvolvido apresenta uma arquitetura modular adequada para projetos de média complexidade. A separação clara entre:

- Modelos (estrutura de dados),
- Serviços (regras de negócio),
- *Signals* (automatização),
- Admin (interface administrativa),

demonstra alinhamento com princípios de organização e coesão.

Do ponto de vista pedagógico, o sistema resolve um problema real enfrentado por docentes: a dependência excessiva de planilhas Excel e cálculos manuais, frequentemente suscetíveis a erro humano.

Além disso, a implementação de cálculo automático e recálculo dinâmico reduz inconsistências e aumenta confiabilidade.

Entretanto, o sistema ainda possui limitações:

- Interface limitada ao Django Admin;
- Ausência de controle de permissões diferenciadas por perfil;
- Falta de relatórios gráficos;
- Ausência de exportação em PDF ou Excel.

Essas limitações não comprometem a funcionalidade central, mas indicam caminhos de evolução.

9. Implementações Futuras

O sistema pode evoluir significativamente com as seguintes melhorias:

9.1. Interface Própria

Desenvolvimento de interface com:

- *Templates* customizados;
- Visualização consolidada anual por aluno;
- Painel de desempenho com gráficos.

9.2. Gráficos de Desempenho

Integração com bibliotecas como:

- Chart.js
- Django REST + frontend React
- Relatórios comparativos por período

9.3. Exportação de Relatórios

Implementação de:

- Geração de boletim em PDF;
- Exportação CSV/Excel;

- Relatórios estatísticos da turma.

9.4. Módulo Ensino Profissional (UFCD)

Extensão do sistema para:

- Escala 0–20;
- Peso fixo 70/30;
- Associação de UFCD a múltiplas turmas;
- Autoavaliação específica por UFCD.

9.5. Controle de Acesso

Implementação de:

- Perfis (Professor, Coordenador, Direção);
- Restrição de acesso por turma;
- Logs de auditoria.

10. Conclusão

O desenvolvimento deste sistema de Gestão de Avaliações Escolares permitiu-me aplicar, de forma prática, conceitos fundamentais de Arquitetura e Desenho de *Software*.

O projeto envolveu:

- Modelagem de dados consistente;
- Implementação de regras de negócio complexas;
- Automatização segura com *signals*;
- Separação clara de responsabilidades;
- Validações robustas;
- Ajustes estruturais baseados em análise crítica.

A escolha do Python e do Django mostrou-se adequada devido à produtividade, clareza sintática e robustez do framework.

O uso de IA foi estratégico e crítico, contribuindo para agilidade, mas exigindo constante validação humana.

Concluo que o sistema atende aos objetivos inicialmente propostos e representa uma solução funcional, extensível e pedagogicamente relevante. Além disso, o projeto evidencia maturidade arquitetural, reflexão técnica e capacidade de evolução futura.

11. Referências

Ambler, S. W. (2003). *Mapping objects to relational databases: O/R mapping in detail*. AmbySoft Inc.

Bird, C., et al. (2023). *Taking flight with Copilot: Early insights and opportunities of AI-powered code generation tools*. Microsoft Research. https://www.microsoft.com/en-us/research/

Django Software Foundation. (2024). *Django documentation*. https://docs.djangoproject.com/

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Holovaty, A., & Kaplan-Moss, J. (2009). *The definitive guide to Django: Web development done right*. Apress.

McKinsey & Company. (2023). *The state of AI in 2023: Generative AI's breakout year*. https://www.mckinsey.com/

Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database system concepts* (7th ed.). McGraw-Hill Education.

Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.