📚 Assistente Pedagógico VSCode

Documentação Técnica do Sistema - Especificação de Requisitos e Modelagem de Dados

📋 Índice

1. Visão Geral do Sistema

1.1 Descrição

O sistema é uma **extensão para VSCode** que funciona como um assistente pedagógico para alunos de programação. Diferente de ferramentas como GitHub Copilot que autocompletam código, este sistema foca em **explicar erros e ajudar o aluno a entender seus problemas**.

> **Propósito Principal:** Permitir que professores entendam as dificuldades reais de seus alunos através da análise das interações com a IA, possibilitando o planejamento de aulas mais direcionadas.

1.2 Componentes do Sistema

- **Extensão VSCode:** Interface do aluno para interação com o assistente
- **Backend:** API para processamento das interações e armazenamento de dados
- **Frontend Web:** Dashboard para professores acessarem dados consolidados
- **IA:** Motor de processamento que gera respostas, code reviews e consolidações

1.3 Objetivos

- Auxiliar alunos a entenderem seus erros de programação
- Gerar code reviews educativos (não apenas corrigir, mas explicar)
- Consolidar as dificuldades de cada sessão
- Permitir que professores identifiquem padrões de dificuldade na turma
- Fornecer dados para planejamento de aulas focadas

2. Requisitos do Sistema

2.1 Requisitos Funcionais

RF01 - Gestão de Usuários

- RF01.1 - O sistema deve permitir cadastro de professores
- RF01.2 - O sistema deve permitir cadastro de alunos
- RF01.3 - O sistema deve autenticar usuários via email e senha
- RF01.4 - Professor e Aluno são entidades separadas com dados distintos

RF02 - Gestão de Classes

- RF02.1 - Professor pode criar múltiplas classes
- RF02.2 - Professor pode adicionar alunos às suas classes
- RF02.3 - Um aluno pode participar de múltiplas classes
- RF02.4 - Cada classe possui seus problemas isolados

RF03 - Gestão de Problemas

- RF03.1 - Professor pode criar problemas em suas classes
- RF03.2 - Problemas podem conter: título, descrição, arquivos (PDF, etc.)
- RF03.3 - Cada problema pertence a uma única classe
- RF03.4 - Aluno pode criar problemas em casos especiais (sandbox)
- RF03.5 - Problemas criados por alunos ficam em uma área "sandbox" para triagem posterior

RF04 - Sessões de Chat

- RF04.1 - Aluno inicia sessão ao abrir VSCode e selecionar um problema
- RF04.2 - Cada aluno pode ter apenas UMA sessão aberta por vez
- RF04.3 - Sessão é fechada quando aluno sai da aba ou abre novo chat
- RF04.4 - Histórico do chat NÃO persiste entre sessões do mesmo problema
- RF04.5 - Aluno pode ter múltiplas sessões fechadas para o mesmo problema (histórico de tentativas)
- RF04.6 - Sessão sempre está vinculada a um problema

RF05 - Mensagens

- RF05.1 - Existem dois tipos de mensagens: do Aluno (User) e da IA (AI)
- RF05.2 - Mensagem do Aluno contém: código enviado e pergunta
- RF05.3 - Mensagem da IA contém: resposta explicativa e code review
- RF05.4 - Cada mensagem possui FK para o problema (desnormalização intencional para queries)
- RF05.5 - Mensagens são ordenadas cronologicamente

RF06 - Consolidação de Sessão

- RF06.1 - Ao fechar sessão, sistema dispara evento para gerar consolidação
- RF06.2 - Consolidação é gerada por IA analisando toda a conversa
- RF06.3 - Consolidação é estruturada (não texto livre) para facilitar queries
- RF06.4 - Cada sessão gera exatamente uma consolidação

RF07 - Dashboard do Professor

- RF07.1 - Professor acessa dados via frontend web
- RF07.2 - Professor pode visualizar consolidações de seus alunos
- RF07.3 - Professor pode buscar padrões de dificuldade ESTRUTURA A DEFINIR

2.2 Requisitos Não-Funcionais

RNF01 - Segurança

- RNF01.1 - Senhas devem ser armazenadas com hash seguro
- RNF01.2 - Professor só acessa dados de suas próprias classes
- RNF01.3 - Aluno só acessa seus próprios dados

RNF02 - Performance

- RNF02.1 - Mensagens devem ter FK para problema para otimizar queries
- RNF02.2 - Consolidações devem ser estruturadas para facilitar buscas

RNF03 - Usabilidade

- RNF03.1 - Interface do VSCode deve ser intuitiva
- RNF03.2 - Aluno deve conseguir iniciar sessão com poucos cliques

3. Atores do Sistema

Ator	Descrição	Principais Ações
Professor	Responsável por criar classes, definir problemas e analisar o desempenho dos alunos	• Criar e gerenciar classes • Criar problemas (PDFs, enunciados) • Adicionar alunos às classes • Visualizar consolidações • Buscar padrões de dificuldade
Aluno	Utiliza a extensão VSCode para obter ajuda com problemas de programação	• Selecionar problema • Iniciar sessão de chat • Enviar código e perguntas • Receber explicações e code reviews • Criar problemas especiais (sandbox)
Sistema (IA)	Processa interações e gera conteúdo pedagógico	• Analisar código do aluno • Gerar explicações sobre erros • Produzir code reviews educativos • Consolidar sessões

4. Fluxos Principais

4.1 Fluxo do Aluno - Sessão de Estudo

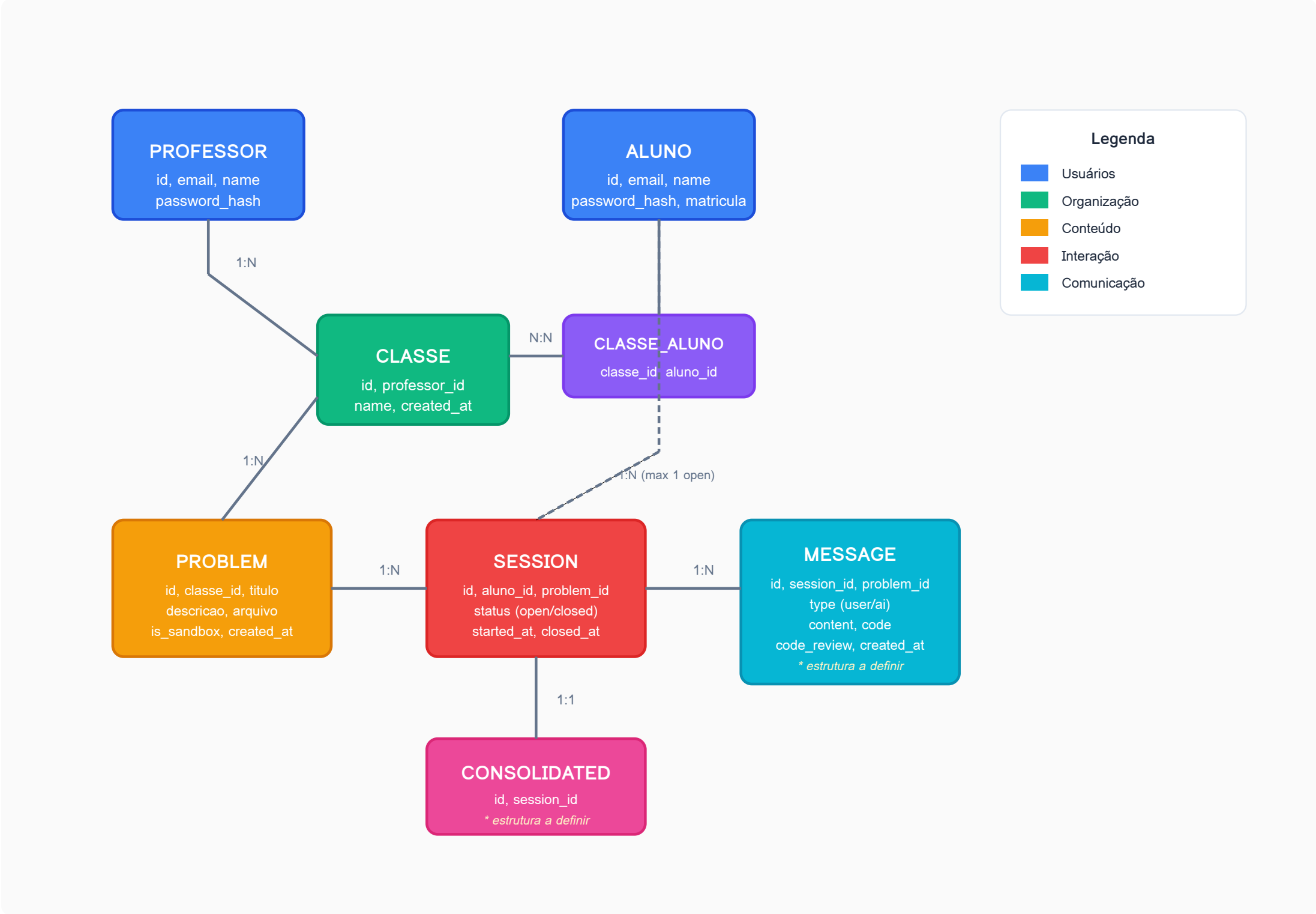
1. **Abertura do VSCode**
   Aluno abre o VSCode com a extensão instalada e faz login
2. **Seleção do Problema**
   Aluno navega pelas suas classes e seleciona o problema que deseja trabalhar
3. **Início da Sessão**
   Sistema cria uma nova SESSION vinculada ao aluno e ao problema selecionado
4. **Interação**
   Aluno envia mensagens contendo código e perguntas. IA responde com explicações e code reviews. Mensagens são salvas na sessão.
5. **Fechamento da Sessão**
   Aluno sai da aba ou abre novo chat. Sessão é marcada como fechada.
6. **Geração da Consolidação**
   Sistema dispara evento assíncrono. IA analisa toda a conversa e gera SESSION_CONSOLIDATED com estrutura de dados sobre as dificuldades identificadas.

4.2 Fluxo do Professor - Análise de Desempenho

1. **Acesso ao Dashboard**
   Professor faz login no frontend web
2. **Seleção da Classe**
   Professor escolhe qual classe deseja analisar
3. **Visualização de Dados**
   Professor consulta consolidações, identifica padrões de dificuldade e planeja próximas aulas

5. Modelo de Dados (MER)

5.1 Diagrama de Entidades e Relacionamentos



6. Especificação das Entidades

6.1 PROFESSOR

PROFESSOR

```
id - UUID / SERIAL (PK)
email - VARCHAR(255) UNIQUE NOT NULL
password_hash - VARCHAR(255) NOT NULL
name - VARCHAR(255) NOT NULL
created_at - TIMESTAMP DEFAULT NOW()
```

6.2 ALUNO

ALUNO

```
id - UUID / SERIAL (PK)
email - VARCHAR(255) UNIQUE NOT NULL
password_hash - VARCHAR(255) NOT NULL
name - VARCHAR(255) NOT NULL
matricula - VARCHAR(50) (opcional)
created_at - TIMESTAMP DEFAULT NOW()
```

6.3 CLASSE

CLASSE

```
id - UUID / SERIAL (PK)
professor_id - FK → PROFESSOR(id) NOT NULL
name - VARCHAR(255) NOT NULL
created_at - TIMESTAMP DEFAULT NOW()
```

6.4 CLASSE_ALUNO (Tabela Associativa N:N)

CLASSE_ALUNO

```
classe_id - FK → CLASSE(id) (PK)
aluno_id - FK → ALUNO(id) (PK)
joined_at - TIMESTAMP DEFAULT NOW()
```

6.5 PROBLEM

PROBLEM

```
id - UUID / SERIAL (PK)
classe_id - FK → CLASSE(id) NOT NULL
titulo - VARCHAR(255) NOT NULL
descricao - TEXT
arquivo - VARCHAR(500) (path para PDF, etc.)
is_sandbox - BOOLEAN DEFAULT FALSE
created_by_aluno_id - FK → ALUNO(id) NULL (se sandbox)
created_at - TIMESTAMP DEFAULT NOW()
```

Nota: Problemas com `is_sandbox = TRUE` são criados por alunos e ficam em área especial para triagem posterior pelo professor.

6.6 SESSION

SESSION

```
id - UUID / SERIAL (PK)
aluno_id - FK → ALUNO(id) NOT NULL
problem_id - FK → PROBLEM(id) NOT NULL
status - ENUM('open', 'closed') DEFAULT 'open'
started_at - TIMESTAMP DEFAULT NOW()
closed_at - TIMESTAMP NULL
```

> **Regra de Negócio:** Cada aluno pode ter no máximo UMA sessão com status 'open' por vez. Ao abrir nova sessão, a anterior deve ser fechada automaticamente.

6.7 MESSAGE ESTRUTURA A DEFINIR

MESSAGE (Proposta - Tabela Única com Discriminator)

```
id - UUID / SERIAL (PK)
session_id - FK → SESSION(id) NOT NULL
problem_id - FK → PROBLEM(id) NOT NULL (desnormalizado)
type - ENUM('user', 'ai') NOT NULL
content - TEXT (pergunta do aluno OU resposta da IA)
code - TEXT NULL (código enviado pelo aluno)
code_review - TEXT NULL (review gerado pela IA)
created_at - TIMESTAMP DEFAULT NOW()
```

> **Decisão Pendente:** Foram discutidas 3 opções para estrutura de mensagens:
> - **Opção A:** Tabela única com type discriminator (mostrada acima)
> - **Opção B:** Tabelas separadas UserMessage e AiMessage
> - **Opção C:** Tabela MESSAGE container + tabelas específicas de conteúdo
>
> A decisão será tomada posteriormente considerando trade-offs de N+1 queries vs. semântica.

6.8 CONSOLIDATED ESTRUTURA A DEFINIR

CONSOLIDATED

```
id - UUID / SERIAL (PK)
session_id - FK → SESSION(id) UNIQUE NOT NULL
??? - estrutura a ser definida
created_at - TIMESTAMP DEFAULT NOW()
```

> **Decisão Pendente:** A estrutura interna do CONSOLIDATED depende de:
> - Que tipo de busca o professor vai fazer
> - Se será por tags, categorias, conceitos, scores, etc.
> - Definição virá após estudo das necessidades do professor

7. Relacionamentos

Origem	Cardinalidade	Destino	Descrição
PROFESSOR	1 : N	CLASSE	Um professor cria várias classes
CLASSE	N : N	ALUNO	Via tabela CLASSE_ALUNO. Aluno pode estar em várias classes.
CLASSE	1 : N	PROBLEM	Cada classe tem seus problemas isolados
PROBLEM	1 : N	SESSION	Um problema pode ter várias sessões (de diferentes alunos ou tentativas)
ALUNO	1 : N	SESSION	Aluno pode ter várias sessões (fechadas). Apenas 1 aberta por vez.
SESSION	1 : N	MESSAGE	Uma sessão contém várias mensagens ordenadas
SESSION	1 : 1	CONSOLIDATED	Cada sessão fechada gera exatamente uma consolidação

8. Decisões Pendentes

8.1 Estrutura de MESSAGE

Status: A DEFINIR

Três opções foram discutidas:

- **Opção A - Tabela única:** Simples, mas com campos nullable dependendo do type
- **Opção B - Tabelas separadas:** Semântica limpa, mas pode causar N+1 ao reconstruir chat
- **Opção C - Container + específicas:** Balance entre as duas, resolve N+1 com JOIN

Decisão será tomada considerando padrões de acesso e performance.

8.2 Estrutura de CONSOLIDATED

Status: A DEFINIR

Depende de entender como o professor vai buscar as informações:

- "Alunos que erraram conceito X"?
- "Dificuldades da turma no problema Y"?
- "Evolução do aluno Z ao longo do tempo"?

A estrutura será definida após estudo das necessidades do professor.

8.3 Problemas Sandbox

Status: PROCESSO A DEFINIR

Definido que problemas criados por alunos vão para área "sandbox". Falta definir:

- Como professor faz triagem desses problemas
- Se podem ser promovidos a problemas oficiais da classe
- Regras de criação (quando aluno pode criar)

9. Glossário

Termo	Definição
Session	Período de interação do aluno com o assistente para um problema específico. Inicia ao selecionar problema, termina ao sair ou abrir novo chat.
Consolidated	Análise estruturada gerada por IA ao final de cada sessão, resumindo as dificuldades identificadas.
Code Review	Análise do código do aluno feita pela IA, com foco educativo (explicar erros, não apenas corrigir).
Problem	Exercício ou desafio de programação definido pelo professor, contendo enunciado e possivelmente arquivos anexos.
Sandbox	Área especial para problemas criados por alunos, aguardando triagem do professor.
Classe	Agrupamento de alunos sob responsabilidade de um professor, com problemas próprios.

Documento gerado em Dezembro de 2024

Versão 1.0 - Especificação Inicial