MEC-SETEC
INSTITUTO FEDERAL MINAS GERAIS - *Campus* Formiga
Curso de Ciência da Computação

SISTEMAS WEB COLABORATIVOS: ANÁLISE DE ARQUITETURAS E DESENVOLVIMENTO DE PROTÓTIPO DE MODELADOR DER

Danielle Leal Silva

Orientador: Prof. Me Roger Santos Ferreira

Formiga - MG

2026

DANIELLE LEAL SILVA

SISTEMAS WEB COLABORATIVOS: ANÁLISE DE ARQUITETURAS E DESENVOLVIMENTO DE PROTÓTIPO DE MODELADOR DER

Monografia do trabalho de conclusão de curso apresentado ao Instituto Federal Minas Gerais - Campus Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Me Roger Santos Ferreira

Formiga - MG

2026

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS
Campus Formiga
Diretoria de Ensino
Docência Área Acadêmica de Computação
Rua São Luiz Gonzaga, s/n - Bairro São Luiz - CEP 35570-000 - Formiga - MG
- www.ifmg.edu.br

DANIELLE LEAL SILVA

SISTEMAS WEB COLABORATIVOS: ANÁLISE DE ARQUITETURAS E DESENVOLVIMENTO DE PROTÓTIPO DE MODELADOR DER

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Minas Gerais - Campus Formiga, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

APROVADA em 6 de outubro de 2026.

BANCA EXAMINADORA

Prof. Roger Santos Ferreira (Orientador)

Prof. Manoel Pereira Junior (IFMG)

Prof. Bruno Ferreira (IFMG)

Documento assinado eletronicamente por **Roger Santos Ferreira**, **Professor**, em 06/10/2026, às 11:29, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Bruno Ferreira**, **Professor**, em 06/10/2026, às 11:33, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Manoel Pereira Junior**, **Professor**, em 06/10/2026, às 11:35, conforme Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade do documento pode ser conferida no site https://sei.ifmg.edu.br/consultadocs informando o código verificador **2909852** e o código CRC **2B41EB6C**.

Silva, Danielle Leal
S586s Sistemas web colaborativos: análise de arquiteturas e desenvolvimento de protótipo de modelador DER /Danielle Leal Silva - Formiga : IFMG, 2026.
64 p. :il. color.

Orientador: Prof. Roger Santos Ferreira

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Sistemas web colaborativos. 2. Arquitetura de software. 3. Diagrama entidade-relacionamento. 4. CRDT. 5. I. Ferreira, Roger Santos. II. Título.

CDD 005.1

Ficha catalográfica elaborada pela Bibliotecária Lívia Renata Santos- CRB/6-2561

Dedico este trabalho a todos os professores que fizeram parte da minha trajetória e que, ao longo dos anos, me lembraram que eu era capaz. A cada professor que acreditou em mim, minha gratidão. Esta conquista também é de vocês.

Agradecimentos

Agradeço, primeiramente, a Deus, por ter me concedido saúde, força e sabedoria para superar as dificuldades e chegar até aqui.

À minha família, por todo o apoio, incentivo e compreensão ao longo dessa caminhada. Em especial, à minha irmã, Maria Eduarda Leal Silva, minha companheira de curso, com quem compartilhei tantos finais de semana de estudos. Ao meu namorado, Pedro Batista Mendonça, pelo apoio, incentivo e por estar sempre ao meu lado.

Ao meu orientador, Prof. Roger Santos Ferreira, pela orientação, apoio e incentivo durante minha formação e, especialmente, no desenvolvimento deste trabalho.

A todos os professores que fizeram parte da minha trajetória acadêmica, pelos conhecimentos compartilhados e por contribuírem para minha formação pessoal e profissional.

Aos meus amigos que compartilharam momentos bons e ruins durante o curso.

Por fim, agradeço a todos que, de alguma forma, contribuíram para que eu chegasse até aqui.

"Se eu vi mais longe, foi por estar sobre ombros de gigantes." (Isaac Newton)

Resumo

A crescente utilização de sistemas web colaborativos tem ampliado a necessidade de mecanismos capazes de garantir comunicação em tempo real, sincronização e consistência durante edições concorrentes. Nesse contexto, este trabalho teve como objetivo propor uma arquitetura para um editor web colaborativo de Diagramas Entidade-Relacionamento (DER), fundamentada em evidências da literatura e avaliada por meio do desenvolvimento do protótipo CoDER. Para isso, foi realizada uma Revisão Sistemática da Literatura sobre sistemas web colaborativos, seguida da definição da arquitetura e do desenvolvimento do protótipo. A solução adotou o modelo cliente-servidor, comunicação por API REST e WebSocket, e implementou um algoritmo CRDT baseado em operações para realizar a sincronização entre as réplicas. A avaliação foi realizada por meio de testes automatizados de convergência e testes de concorrência e carga com até 20 usuários simultâneos. Os resultados demonstraram convergência das réplicas nos cenários avaliados e ausência de perda de mensagens. Nos testes de concorrência e carga, a latência média permaneceu inferior a 36 ms nos cenários com até 15 usuários, enquanto, com 20 usuários, observou-se aumento da latência e redução do throughput em relação à carga oferecida. Os resultados estão restritos às condições do ambiente experimental, não permitindo generalizar o comportamento para cargas superiores. Conclui-se que a arquitetura proposta se mostrou viável para o desenvolvimento de um editor colaborativo de DER dentro do escopo e das condições avaliadas.

Palavras-chave: Sistemas web colaborativos; Arquitetura de software; Diagrama Entidade-Relacionamento; CRDT; WebSocket.

Abstract

The increasing use of collaborative web systems has increased the need for mechanisms capable of ensuring real-time communication, synchronization, and consistency during concurrent editing. In this context, this study aimed to propose an architecture for a collaborative web-based Entity-Relationship Diagram (ERD) editor, grounded in evidence from the literature and evaluated through the development of the CoDER prototype. To this end, a Systematic Literature Review (SLR) on collaborative web systems was conducted, followed by the definition of the architecture and the development of the prototype. The solution adopted the client-server model, communication through REST API and WebSocket, and implemented an operation-based CRDT algorithm to synchronize replicas. The evaluation was conducted through automated convergence tests and concurrency and load tests involving up to 20 simultaneous users. The results demonstrated replica convergence in the evaluated scenarios and no message loss. In the concurrency and load tests, average latency remained below 36 ms in scenarios with up to 15 users, whereas, with 20 users, an increase in latency and a reduction in throughput relative to the offered load were observed. The results are limited to the conditions of the experimental environment and do not allow generalization of the behavior under higher loads. It is concluded that the proposed architecture proved feasible for developing a collaborative ERD editor within the scope and conditions evaluated.

Keywords: Collaborative web systems; Software architecture; Entity-Relationship Diagram; CRDT; WebSocket.

Lista de ilustrações

Figura 1 – Exemplo de Diagrama Entidade-Relacionamento 24
Figura 2 – Arquitetura proposta para o editor colaborativo de DER 32
Figura 3 – Modelo de dados do CoDER 34
Figura 4 – Interface principal do CoDER após a autenticação do usuário 42
Figura 5 – Tela de autenticação do CoDER 43
Figura 6 – Tela de cadastro de novos usuários 43
Figura 7 – Tela de gerenciamento de projetos do CoDER 44
Figura 8 – Tela de projetos compartilhados com o usuário 44
Figura 9 – Editor de Diagramas Entidade-Relacionamento do CoDER 45
Figura 10 – Tela para salvamento de uma versão do diagrama 46
Figura 11 – Tela de compartilhamento de um projeto 47
Figura 12 – Organização dos principais componentes do código-fonte do CoDER . . 48
Figura 13 – Usuários conectados simultaneamente a um projeto no CoDER 51
Figura 14 – Fluxo de sincronização das operações no CoDER 53
Figura 15 – Fluxo de resolução de conflitos 55
Figura 16 – Resultado da execução dos testes de convergência 57

Lista de tabelas

Tabela 1 – Estudos relacionados 18
Tabela 2 – Resultados das etapas de seleção da RSL 29
Tabela 3 – Requisitos considerados na definição da arquitetura 30
Tabela 4 – Tecnologias utilizadas no desenvolvimento do protótipo 33
Tabela 5 – Cenários dos testes de convergência 36
Tabela 6 – Resultados dos testes de concorrência e carga 57

Lista de códigos

Código 4.1 – Envio de operações pelo serviço WebSocket 49
Código 4.2 – Recebimento e distribuição de operações pelo backend 49
Código 4.3 – Verificação de autorização durante a inscrição em um canal WebSocket 50
Código 4.4 – Registro de usuário conectado à sessão colaborativa 51
Código 4.5 – Aplicação de alterações utilizando LWW Register 53
Código 4.6 – Controle de idempotência das operações recebidas 55

Lista de abreviaturas e siglas

ACM	Association for Computing Machinery
API	Application Programming Interface
CRDT	Conflict-Free Replicated Data Type
DDL	Data Definition Language
DER	Diagrama Entidade-Relacionamento
ERD	Entity-Relationship Diagram
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LWW	Last-Write-Wins
MER	Modelo Entidade-Relacionamento
OT	Operational Transformation
P2P	Peer-to-Peer
PNG	Portable Network Graphics
RAM	Random Access Memory
REST	Representational State Transfer
RSL	Revisão Sistemática da Literatura
SQL	Structured Query Language
STOMP	Simple (Streaming) Text Oriented Messaging Protocol
WebRTC	Web Real-Time Communication

Sumário

1 INTRODUÇÃO 15
1.1 Justificativa 16
1.2 Objetivos 17
1.2.1 Objetivo geral 17
1.2.2 Objetivos específicos 17
1.3 Estrutura do Trabalho 17

2 FUNDAMENTAÇÃO TEÓRICA 18
2.1 Sistemas Web Colaborativos 20
2.2 Arquitetura de Software 22
2.3 Modelo Entidade-Relacionamento 23
2.4 Sincronização em Sistemas Colaborativos 25
2.4.1 WebSocket 25
2.4.2 Transformação Operacional (Operational Transformation) 26
2.4.3 CRDT (Conflict-Free Replicated Data Type) 26

3 MATERIAIS E MÉTODO 28
3.1 Revisão Sistemática da Literatura 28
3.2 Definição da Arquitetura 30
3.3 Desenvolvimento do Protótipo 33
3.4 Avaliação 35
3.4.1 Ambiente de execução 35
3.4.2 Testes de convergência 36
3.4.3 Testes de concorrência e carga 39

4 RESULTADOS E DISCUSSÕES 41
4.1 Visão Geral do Protótipo 41
4.2 Funcionalidades Implementadas 42
4.2.1 Autenticação de usuários 42
4.2.2 Gerenciamento de projetos 43
4.2.3 Editor de Diagramas Entidade-Relacionamento 45
4.3 Arquitetura Implementada 47
4.3.1 Comunicação entre cliente e servidor 48
4.3.2 Sincronização do estado compartilhado 52
4.3.3 Idempotência das operações 55
4.4 Resultados dos Testes 56

4.5 Discussão dos Resultados 58

5 CONSIDERAÇÕES FINAIS 60
5.1 Principais Contribuições 60
5.2 Trabalhos Futuros 61

REFERÊNCIAS 62

1 Introdução

A crescente adoção de aplicações web e de ambientes de trabalho distribuídos transformou a forma como equipes colaboram em contextos educacionais e corporativos, tornando a tecnologia um importante recurso para a comunicação, coordenação e execução de atividades em grupo. Esses aspectos estão diretamente relacionados aos fundamentos dos sistemas colaborativos, nos quais a comunicação, a coordenação e a cooperação constituem elementos centrais para a realização de atividades conjuntas (PIMENTEL; FUKS, 2011). Nesse cenário, organizações e instituições de ensino têm incorporado cada vez mais ferramentas digitais que favorecem a colaboração remota, buscando aumentar a produtividade, a eficiência e a interação entre seus participantes.

Entre essas ferramentas, destacam-se aplicações web colaborativas como Google Docs[1], Trello[2] e Figma[3]. Essas plataformas permitem que múltiplos usuários trabalhem simultaneamente sobre um mesmo recurso compartilhado, independentemente de sua localização geográfica. Além disso, documentos, projetos e demais artefatos permanecem disponíveis continuamente, facilitando o acompanhamento das atividades, o compartilhamento de informações e a gestão do trabalho colaborativo.

Diversas pesquisas têm investigado a utilização dessas ferramentas em contextos educacionais e corporativos, evidenciando benefícios relacionados à colaboração, à comunicação e à interação em tempo real. García (2024), por exemplo, destaca que recursos como Google Docs, Jamboard e Trello favorecem o engajamento entre alunos e professores, além de estimular a construção coletiva do conhecimento durante o processo de ensino-aprendizagem.

Embora essas aplicações ofereçam ao usuário uma experiência colaborativa integrada, sua implementação envolve desafios relacionados à comunicação em tempo real, à sincronização dos dados compartilhados e à resolução de conflitos decorrentes de edições concorrentes. Para lidar com esses desafios, a literatura apresenta diferentes abordagens arquiteturais, empregando arquiteturas cliente-servidor, mecanismos de comunicação baseados em WebSocket e estratégias de sincronização, como *Operational Transformation* (OT) e *Conflict-Free Replicated Data Types* (CRDT).

A Revisão Sistemática da Literatura (RSL) realizada nesta pesquisa evidenciou uma diversidade de soluções arquiteturais empregadas em sistemas web colaborativos. Entre os estudos analisados, apenas um abordou a edição colaborativa de diagramas, enquanto os demais se concentraram em outros tipos de artefatos e atividades colaborativas. Esse

[1] <https://docs.google.com>
[2] <https://trello.com>
[3] <https://www.figma.com>

cenário evidencia a possibilidade de aplicar os mecanismos e as arquiteturas identificadas na literatura ao domínio específico da modelagem de Diagramas Entidade-Relacionamento.

Os Diagramas Entidade-Relacionamento (DER) constituem uma das principais ferramentas utilizadas na modelagem conceitual de bancos de dados, sendo empregados tanto no ensino quanto no desenvolvimento de sistemas de informação. Atualmente, existem ferramentas que oferecem recursos de colaboração em tempo real para a criação e edição desses diagramas, como o dbdiagram.io, o drawDB e o ERDHub (dbdiagram.io, s.d.; drawDB, s.d.; ERDHub, s.d.). Essas ferramentas permitem que diferentes usuários colaborem na construção de modelos de dados. Entretanto, a colaboração nesse tipo de aplicação apresenta desafios relacionados à sincronização de elementos gráficos estruturados e ao tratamento de edições concorrentes. Essas características tornam a modelagem colaborativa de DER um contexto relevante para a investigação e aplicação de arquiteturas colaborativas.

Assim, este trabalho propõe uma arquitetura para um editor colaborativo de DER, fundamentada nas evidências obtidas por meio de uma Revisão Sistemática da Literatura e materializada pelo desenvolvimento de um protótipo funcional denominado CoDER. A solução integra uma arquitetura cliente-servidor, comunicação em tempo real por meio de WebSocket e um mecanismo de sincronização baseado em CRDT, permitindo que múltiplos usuários editem simultaneamente um mesmo diagrama de forma consistente.

1.1 Justificativa

O desenvolvimento de sistemas colaborativos representa um desafio relevante tanto para a pesquisa quanto para a indústria de software, uma vez que envolve requisitos não funcionais complexos, como consistência dos dados, comunicação em tempo real, tratamento de concorrência e escalabilidade. Embora existam diversas aplicações colaborativas consolidadas, os mecanismos arquiteturais empregados por essas soluções nem sempre são descritos na literatura de forma detalhada, dificultando sua reprodução e adaptação para novos domínios de aplicação.

Além disso, a RSL realizada nesta pesquisa evidenciou que, entre os sistemas web colaborativos selecionados, uma parcela menor dos estudos está voltada à edição colaborativa de diagramas. Esse resultado motivou a aplicação das abordagens arquiteturais identificadas na literatura ao contexto específico da modelagem de DER.

Sob a perspectiva científica, este trabalho contribui ao reunir, analisar e sintetizar evidências sobre arquiteturas de sistemas web colaborativos, utilizando essas informações como base para a proposição de uma arquitetura voltada ao domínio da modelagem de DER.

Sob a perspectiva tecnológica, a pesquisa demonstra a aplicação integrada de tecnologias amplamente utilizadas no desenvolvimento de aplicações modernas, como WebSocket, CRDT, Vue.js e Spring Boot, resultando em uma arquitetura capaz de suportar edição colaborativa em tempo real.

Por fim, sob a perspectiva prática, o protótipo desenvolvido pode servir como referência para pesquisas futuras e para o desenvolvimento de aplicações colaborativas semelhantes, além de poder contribuir para atividades de ensino e aprendizagem relacionadas à modelagem de bancos de dados.

1.2 Objetivos

1.2.1 Objetivo geral

Propor uma arquitetura para um editor web colaborativo de Diagramas Entidade-Relacionamento, fundamentada em uma Revisão Sistemática da Literatura e demonstrada por meio do desenvolvimento de um protótipo funcional.

1.2.2 Objetivos específicos

1. Realizar uma RSL sobre arquiteturas e tecnologias empregadas em sistemas web colaborativos, identificando e sintetizando as principais abordagens encontradas.
2. Propor uma arquitetura para um editor colaborativo de DER com base nas evidências obtidas na RSL.
3. Desenvolver um protótipo funcional de um editor colaborativo de DER, aplicando a arquitetura proposta.
4. Avaliar o funcionamento do protótipo desenvolvido em cenários de colaboração concorrente.

1.3 Estrutura do Trabalho

Este trabalho está organizado em cinco capítulos. O Capítulo 2 apresenta os principais fundamentos teóricos e os conceitos relacionados ao tema. O Capítulo 3 descreve a metodologia adotada para o desenvolvimento da pesquisa e do protótipo. O Capítulo 4 apresenta os resultados obtidos e as discussões decorrentes da implementação e avaliação da solução proposta. Por fim, o Capítulo 5 reúne as considerações finais do trabalho, suas limitações e sugestões para trabalhos futuros.

2 Fundamentação Teórica

A fundamentação teórica deste trabalho é baseada nos estudos identificados por meio da Revisão Sistemática da Literatura (RSL) realizada na etapa inicial da pesquisa. A RSL permitiu identificar as principais arquiteturas, tecnologias e técnicas empregadas em sistemas web colaborativos para edição compartilhada, fornecendo embasamento para a definição da arquitetura proposta neste trabalho.

Os estudos selecionados foram analisados considerando aspectos como arquitetura de software, mecanismos de comunicação em tempo real, técnicas de sincronização de dados, tecnologias utilizadas e estratégias para resolução de conflitos entre edições simultâneas. A síntese desses trabalhos é apresentada na Tabela 1.

Tabela 1 – Estudos relacionados

Trabalho	**Arquitetura**	**Comunicação**	**Sincronização**	**Banco de Dados**
CodePilot (WARNER; GUO, 2017)	Cliente-servidor	Tempo real (Firebase)	Firepad (OT)	Firebase
CodeR (KURNIAWAN; SOESANTO; WIJAYA, 2015)[1]	Cliente-servidor	WebSocket	OT	NoSQL
Diagramar (AURIA; BARZOLA; ECHEVERRIA, 2021)	Cliente-servidor	Firebase (listeners assíncronos)	Sincronização em tempo real	NoSQL
Differentially processed optimized collaborative rich text editor (JATANA et al., 2025)	Cliente-servidor	WebSocket	CRDT (Yjs)	Não informado
OnGraX (ZIMMER; KERREN, 2015)	Cliente-servidor	WebSocket	Bloqueios pessimistas	Neo4j e MySQL
CAE (LANGE et al., 2020)	Microsserviços / distribuída	P2P (las2peer)	CRDT (Yjs)	Não informado

Continua na próxima página

Trabalho	**Arquitetura**	**Comunicação**	**Sincronização**	**Banco de Dados**
HTML DOM Synchronization (GADEA; IONESCU; IONESCU, 2018)	Cliente-servidor	WebSocket	OT	Firebase
OntoEditor (HEMID et al., 2024)	Cliente-servidor	WebSocket (Socket.io)	OT (ShareDB)	MongoDB
Pyro (ZWEIHOFF; NAUJOKAT; STEFFEN, 2019)	Cliente-servidor	Não informado	CRDT	DyWA
SecCollab (STANCU; CHIROIU; RUGHINIS, 2017)	Cliente-servidor	Não informado	OT / sincronização diferencial	Não informado
SAGE2 + IAM (BHARADWAJ et al., 2021)	Cliente-servidor	Não informado	Sincronização de estado (SAGE2)	Não informado
Vclick (SINGH; YOAKUM, 2015)	Híbrida (cliente-servidor e P2P)	WebRTC e WebSocket	Não informado	Não informado

Fonte: Elaborada pela autora (2026).

A análise dos 12 trabalhos apresentados na Tabela 1 permite identificar algumas tendências quantitativas. Quanto à arquitetura, 10 trabalhos adotam o modelo cliente-servidor, enquanto 1 apresenta uma estrutura baseada em microsserviços/distribuída e outro emprega uma arquitetura híbrida. No que diz respeito à comunicação, WebSocket aparece em 6 trabalhos, mecanismos baseados no Firebase são encontrados em 2, e 1 recorre à comunicação P2P com las2peer. Os 3 trabalhos restantes não especificam o mecanismo empregado.

Em relação à sincronização, a Transformação Operacional (OT) é adotada em 5 trabalhos, enquanto 3 recorrem a CRDTs. Além disso, 1 trabalho emprega bloqueios pessimistas, 2 apresentam mecanismos de sincronização de estado ou de dados em tempo real e 1 não especifica a abordagem utilizada. Esses resultados evidenciam a recorrência de

[1] A denominação *CodeR* refere-se à aplicação de edição de código em tempo real proposta por Kurniawan et al. (KURNIAWAN; SOESANTO; WIJAYA, 2015). Já o nome *CoDER*, adotado para o protótipo desenvolvido neste trabalho, foi definido de forma independente e refere-se ao contexto de edição colaborativa de Diagramas Entidade-Relacionamento.

determinadas abordagens na literatura analisada e serviram de referência para a definição da arquitetura proposta neste trabalho. Com base nisso, as próximas seções apresentam os principais conceitos relacionados a sistemas web colaborativos, arquitetura de software, modelagem por meio do Modelo Entidade-Relacionamento (MER) e mecanismos de sincronização empregados em aplicações colaborativas em tempo real.

2.1 Sistemas Web Colaborativos

Sistemas web são aplicações de software acessadas por meio de navegadores ou interfaces de programação de aplicações (APIs), utilizando a infraestrutura da internet para disponibilizar funcionalidades aos usuários. Um sistema web colaborativo estende esse conceito ao permitir que múltiplos usuários interajam sobre recursos compartilhados, de forma síncrona ou assíncrona, com o objetivo de realizar atividades em conjunto, como a criação de documentos, o gerenciamento de projetos ou o compartilhamento de informações (PIMENTEL; FUKS, 2011).

Ao contrário de sistemas multiusuário simples, que apenas permitem o acesso simultâneo de diferentes pessoas, sistemas colaborativos possibilitam a comunicação, coordenação e cooperação direta entre os participantes. Algumas de suas características essenciais incluem:

- comunicação síncrona e assíncrona (como chats, notificações e comentários);
- atualizações em tempo real;
- gerenciamento de permissões e papéis de usuário;
- registro histórico e versionamento das alterações;
- resolução de conflitos quando múltiplos usuários editam o mesmo recurso simultaneamente.

Embora sistemas colaborativos possam oferecer suporte tanto à colaboração síncrona quanto à assíncrona, este trabalho concentra-se na colaboração síncrona, na qual múltiplos usuários interagem simultaneamente sobre um mesmo recurso compartilhado. Os sistemas colaborativos podem ser encontrados em diferentes domínios de aplicação, como edição de documentos, desenvolvimento de software, modelagem de diagramas e colaboração multimídia. A seguir são apresentados alguns exemplos representativos encontrados na literatura.

O Vclick, proposto por Singh e Yoakum (2015), apresenta uma arquitetura leve e escalável para colaboração multimídia via web, utilizando tecnologias como WebRTC e WebSocket. O Vclick se diferencia por manter toda a lógica da aplicação no navegador,

enquanto o servidor atua apenas no gerenciamento de recursos e sinalização. Essa abordagem resulta em uma plataforma flexível e modular, permitindo que diferentes aplicações, como videoconferência, chat, compartilhamento de tela e quadro branco, funcionem de forma desacoplada e integrada. Segundo os autores, essa estrutura facilita a escalabilidade do sistema, reduz a complexidade do servidor e melhora a experiência do usuário final, especialmente em ambientes corporativos distribuídos.

Outro exemplo é o Diagramar, proposto por Auria, Barzola e Echeverria (2021). Trata-se de uma aplicação web colaborativa voltada à realização e ao acompanhamento de atividades de design, permitindo a criação de modelos Entidade-Relacionamento (MER), mapas conceituais e outros tipos de diagramas. No contexto de modelos Entidade-Relacionamento, os estudantes podem adicionar, editar e remover elementos de forma simultânea, enquanto suas ações são registradas em tempo real. A ferramenta também permite que professores acompanhem a participação dos grupos por meio de visualizações que apresentam as contribuições individuais e coletivas realizadas durante as atividades.

Em relação à arquitetura, o Diagramar adota uma abordagem cliente-servidor, utilizando Vue.js no frontend e serviços do Firebase no backend. A aplicação utiliza o Firebase Realtime Database para armazenamento e sincronização dos dados, com mecanismos de atualização assíncrona para manter os diagramas atualizados entre os participantes (AURIA; BARZOLA; ECHEVERRIA, 2021). Diferentemente dos trabalhos que empregam explicitamente OT ou CRDTs, o artigo não especifica a utilização dessas técnicas para a sincronização das alterações.

O Diagramar apresenta uma relação direta com o presente trabalho por também disponibilizar a edição colaborativa de modelos Entidade-Relacionamento em ambiente web. Entretanto, seu objetivo principal está associado ao contexto educacional, especialmente ao acompanhamento de atividades realizadas por estudantes e à possibilidade de intervenção dos professores durante a colaboração. O CoDER, por sua vez, concentra-se na edição colaborativa de DER, tendo como foco a sincronização das operações realizadas pelos usuários durante a construção e alteração compartilhada dos diagramas.

Finalmente, o Google Docs é um exemplo consolidado de sistema web colaborativo de grande escala. Ele permite que múltiplos usuários editem o mesmo documento simultaneamente, visualizando as alterações uns dos outros em tempo real. O Google Docs utiliza a OT, um algoritmo que gerencia conflitos decorrentes de edições concorrentes, garantindo a consistência do documento final (DAY-RICHTER, 2010). A arquitetura cliente-servidor permite que cada cliente aplique alterações localmente de forma otimista, enquanto o servidor atua como fonte de verdade, processando e distribuindo atualizações para todos os clientes. Esse mecanismo possibilita colaboração contínua mesmo em situações de instabilidade de rede, mantendo a integridade dos dados.

2.2 Arquitetura de Software

De acordo com Bass, Clements e Kazman (2022), a arquitetura de software representa um elemento essencial para o desenvolvimento bem-sucedido de sistemas, atuando como uma ponte entre os objetivos de negócio, que geralmente são abstratos, e o sistema final, que é concreto. Esses autores definem arquitetura de software como o conjunto de estruturas necessárias para compreender um sistema, sendo essas estruturas compostas por elementos de software, pelas relações entre eles e pelas propriedades que ambos possuem.

A escolha do modelo arquitetural impacta diretamente características como desempenho, escalabilidade, manutenção, disponibilidade e facilidade de evolução do sistema. Em sistemas colaborativos, essa decisão torna-se ainda mais relevante devido à necessidade de sincronizar informações compartilhadas entre múltiplos usuários em tempo real.

Os sistemas colaborativos síncronos são, em sua maioria, sistemas distribuídos, pois permitem que diferentes usuários interajam simultaneamente sobre um mesmo espaço de trabalho compartilhado por meio de computadores conectados em rede (GOMES; WILLRICH; RIVERA, 2012). Nesse contexto, a arquitetura adotada deve garantir a propagação consistente das alterações realizadas pelos usuários, além de atender requisitos não funcionais como tempo de resposta, consistência dos dados, tolerância a falhas, segurança e escalabilidade.

Diferentes modelos arquiteturais podem ser empregados no desenvolvimento de sistemas colaborativos. A seguir, são apresentadas algumas abordagens identificadas na literatura e consideradas relevantes para o contexto deste trabalho.

- **Arquitetura Cliente-Servidor:** A arquitetura cliente-servidor organiza o sistema em componentes responsáveis por diferentes funções. O cliente normalmente concentra a interação com o usuário e o processamento relacionado à apresentação, enquanto o servidor disponibiliza serviços, processa requisições e centraliza recursos compartilhados. Entretanto, essa arquitetura apresenta limitações relacionadas à escalabilidade e à tolerância a falhas, pois o servidor constitui um ponto central de processamento e também um ponto único de falha (GOMES; WILLRICH; RIVERA, 2012).

- **Arquitetura de Microsserviços:** Nessa arquitetura, o sistema é dividido em serviços independentes que se comunicam por meio de APIs. Cada serviço é responsável por uma funcionalidade específica e pode ser desenvolvido, implantado e escalado de forma independente (GOMES; WILLRICH; RIVERA, 2012). Um exemplo é o sistema proposto por Lange et al. (2020), que integra edição colaborativa em tempo real e criação de wireframes em um ambiente baseado em *Model-Driven Web Engineering* (MDWE). Nesse sistema, funcionalidades como gerenciamento de modelos,

sincronização em tempo real, controle de versões e colaboração são implementadas como microsserviços, favorecendo a escalabilidade, a manutenção e a evolução da aplicação.

- **Arquitetura Baseada em Eventos (*Event-Driven*):** Nesse modelo arquitetural, a comunicação entre os componentes ocorre por meio da troca de eventos, reduzindo o acoplamento entre os serviços e permitindo maior flexibilidade na distribuição do processamento. Em aplicações colaborativas, essa abordagem favorece a atualização em tempo real das informações compartilhadas, permitindo que alterações realizadas por um usuário sejam propagadas imediatamente aos demais participantes da sessão (GOMES; WILLRICH; RIVERA, 2012).
- **Arquitetura Híbrida:** Segundo Gomes, Willrich e Rivera (2012), arquiteturas híbridas podem combinar características de abordagens centralizadas e descentralizadas, distribuindo responsabilidades entre clientes e servidor. Essa estratégia é particularmente relevante para sistemas colaborativos, nos quais parte do processamento pode ser realizada localmente, enquanto determinados serviços permanecem centralizados. Um exemplo é o Vclick (SINGH; YOAKUM, 2015), que utiliza sinalização no modelo cliente-servidor e conexões peer-to-peer (P2P) para a transmissão de mídia por meio do WebRTC. Dessa forma, o sistema combina mecanismos centralizados e descentralizados na comunicação entre os participantes.

2.3 Modelo Entidade-Relacionamento

O Modelo Entidade-Relacionamento (MER) é uma técnica de modelagem conceitual utilizada no projeto de bancos de dados para representar, de forma abstrata, os elementos de um domínio de aplicação e os relacionamentos existentes entre eles. Proposto por Chen (1976), esse modelo permite descrever a estrutura conceitual dos dados antes da implementação física do banco de dados, facilitando a análise dos requisitos e a comunicação entre analistas, desenvolvedores e demais envolvidos no projeto.

A representação gráfica do MER é realizada por meio do DER, que descreve visualmente as entidades, seus atributos e os relacionamentos existentes entre elas. Dessa forma, o diagrama possibilita compreender como as informações do domínio estão organizadas e como se relacionam, servindo como base para a construção do esquema lógico e, posteriormente, do banco de dados (ELMASRI; NAVATHE, 2011).

As entidades representam objetos ou conceitos do mundo real sobre os quais se deseja armazenar informações. Uma entidade pode representar pessoas, organizações, objetos, eventos ou qualquer outro elemento relevante para o domínio da aplicação (ELMASRI; NAVATHE, 2011). Exemplos de entidades incluem *Fornecedor*, *Pessoa*, *Curso* e *Imóvel*. Cada entidade é caracterizada por um conjunto de atributos, que descrevem suas

propriedades. Por exemplo, uma entidade *Pessoa* pode possuir atributos como nome, CPF e data de nascimento.

Os relacionamentos representam as associações existentes entre duas ou mais entidades, descrevendo como elas interagem no contexto do sistema (ELMASRI; NAVATHE, 2011). Por exemplo, uma entidade *Professor* pode estar associada a uma entidade *Departamento* por meio do relacionamento "lotação", indicando em qual departamento o professor atua.

Além dos relacionamentos, o MER utiliza cardinalidades para especificar as restrições de participação entre as entidades. Neste trabalho, utiliza-se a notação mínimo-máximo, em que cada par de valores indica as quantidades mínima e máxima de ocorrências que podem participar de um relacionamento. Por exemplo, (1,1) indica exatamente uma ocorrência, enquanto (1,N) indica uma ou mais ocorrências (ELMASRI; NAVATHE, 2011). Essas restrições são fundamentais para garantir a consistência do modelo e orientar sua implementação no banco de dados. A Figura 1 apresenta um exemplo de DER, ilustrando entidades, atributos, relacionamentos e suas respectivas cardinalidades.

A partir do modelo conceitual, é possível derivar o esquema lógico do banco de dados, no qual os relacionamentos entre as entidades são representados por meio de chaves estrangeiras. Essas chaves correspondem a atributos que referenciam as chaves primárias de outras tabelas, permitindo estabelecer associações entre os registros e preservar a integridade referencial do banco de dados.

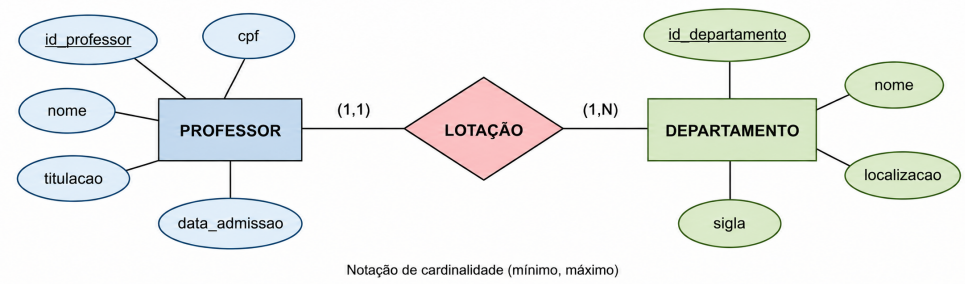


Figura 1 – Exemplo de Diagrama Entidade-Relacionamento

Fonte: Elaborada pela autora (2026).

Neste trabalho, o Diagrama Entidade-Relacionamento constitui o artefato manipulado pelos usuários durante a edição colaborativa. Dessa forma, compreender sua estrutura é essencial para o desenvolvimento do editor proposto, uma vez que todas as operações realizadas pelos usuários, como criação, edição, movimentação e remoção de elementos do diagrama, ocorrem sobre entidades, relacionamentos e seus respectivos atributos.

2.4 Sincronização em Sistemas Colaborativos

Um dos principais desafios no desenvolvimento de sistemas colaborativos em tempo real é manter a consistência do estado compartilhado entre todos os participantes da sessão. Como múltiplos usuários podem realizar alterações simultaneamente sobre um mesmo recurso, o sistema deve garantir que essas modificações sejam propagadas corretamente para todos os clientes, preservando um estado consistente entre os participantes, mesmo quando as operações são recebidas em ordens diferentes devido às características da comunicação em rede.

Sem mecanismos adequados de sincronização, podem ocorrer situações como perda de atualizações, sobrescrita de alterações realizadas por outros usuários e divergência entre as diferentes cópias do recurso compartilhado. Esses problemas tornam-se mais evidentes em ambientes distribuídos, nos quais atrasos de rede, mensagens fora de ordem ou desconexões temporárias podem fazer com que cada cliente possua uma visão diferente do estado da aplicação. Em um editor colaborativo de DER, por exemplo, essas situações podem ocorrer quando diferentes usuários alteram simultaneamente uma mesma entidade, seus atributos ou sua existência no diagrama.

Para minimizar esses problemas, sistemas colaborativos empregam mecanismos de comunicação em tempo real para distribuir rapidamente as alterações realizadas pelos usuários e algoritmos de sincronização responsáveis por garantir a consistência do estado compartilhado. Entre as tecnologias utilizadas neste trabalho destacam-se o protocolo WebSocket, responsável pela comunicação bidirecional entre cliente e servidor com baixa latência, e os *Conflict-Free Replicated Data Types* (CRDTs), empregados para garantir a convergência do estado compartilhado mesmo diante de edições concorrentes. As subseções a seguir apresentam os conceitos fundamentais dessas tecnologias.

2.4.1 WebSocket

O WebSocket é uma tecnologia de comunicação bidirecional que permite a transmissão de dados entre um navegador web e um servidor de maneira eficiente e em tempo real, criando uma conexão persistente entre o cliente e o servidor. Dessa forma, possibilita uma comunicação de baixa latência, permitindo que as alterações realizadas por um usuário sejam enviadas ao servidor e propagadas aos demais participantes quase instantaneamente.

Em sistemas colaborativos, o WebSocket é amplamente utilizado para implementar recursos como edição simultânea de documentos, notificações em tempo real e sincronização instantânea do estado compartilhado entre múltiplos usuários.

Entre suas principais vantagens estão a baixa latência, o baixo consumo de banda e a simplicidade de implementação. No entanto, o WebSocket é responsável apenas pelo transporte das mensagens, não oferecendo mecanismos para resolver conflitos decorrentes

de edições concorrentes (WANG; SALIM; MOSKOVITS, 2013).

2.4.2 Transformação Operacional (Operational Transformation)

A Transformação Operacional é uma técnica de sincronização desenvolvida para permitir a edição colaborativa em tempo real de documentos compartilhados. Proposta inicialmente por Ellis e Gibbs (1989), essa abordagem busca garantir que diferentes usuários possam realizar alterações simultaneamente sobre um mesmo recurso, mantendo a consistência entre todas as réplicas do documento.

O princípio de funcionamento da OT consiste na transformação de operações concorrentes antes de sua aplicação. Quando dois ou mais usuários modificam simultaneamente um mesmo documento, as operações recebidas são ajustadas de acordo com as alterações já executadas, preservando a intenção original de cada usuário e evitando inconsistências decorrentes da ordem de chegada das mensagens.

Essa técnica foi amplamente adotada em sistemas colaborativos de grande escala, sendo o Google Docs um dos exemplos de sua aplicação. Nesse tipo de sistema, as alterações realizadas pelos usuários são enviadas ao servidor, que transforma as operações concorrentes e as redistribui aos demais participantes, garantindo que todos os clientes mantenham versões equivalentes do documento compartilhado.

Embora a OT tenha se consolidado como uma solução eficiente para edição colaborativa, sua implementação torna-se mais complexa à medida que aumenta a variedade de operações suportadas pela aplicação, pois é necessário definir regras de transformação para cada combinação possível de operações concorrentes.

Como alternativa à OT, os CRDTs adotam uma abordagem distinta para a sincronização de dados compartilhados. Em vez de transformar operações concorrentes, os CRDTs utilizam estruturas de dados projetadas para garantir convergência de forma determinística, reduzindo a complexidade da sincronização em aplicações distribuídas.

2.4.3 CRDT (Conflict-Free Replicated Data Type)

Os CRDTs são estruturas de dados utilizadas em sistemas distribuídos que permitem que múltiplas réplicas sejam atualizadas de forma independente e simultânea. Essas estruturas garantem que todas as réplicas convirjam, ao final, para um mesmo estado consistente, mesmo que as atualizações ocorram fora de ordem ou que alguns nós sofram interrupções temporárias na comunicação (SHAPIRO et al., 2011).

Diferentemente da OT, que preserva a consistência por meio da transformação de operações concorrentes, os CRDTs são projetados para que operações ou estados possam ser combinados de forma determinística, garantindo convergência sem a necessidade de transformar operações previamente executadas.

Os CRDTs podem ser classificados em duas categorias principais: os baseados em operações e os baseados em estados (SHAPIRO et al., 2011). Nos primeiros, as atualizações são representadas por operações que, sob as condições adequadas de entrega e comutatividade, permitem garantir a convergência das réplicas. Nos CRDTs baseados em estados, cada réplica mantém uma cópia do estado local e periodicamente realiza a fusão desse estado com o das demais réplicas por meio de funções determinísticas (SHAPIRO et al., 2011).

Os CRDTs têm sido aplicados em diversas áreas. Em sistemas colaborativos, por exemplo, são utilizados para permitir que múltiplos usuários realizem edições simultâneas em documentos, tabelas ou diagramas sem que ocorram conflitos ou perda de informações. Um exemplo é o Collabs, um framework desenvolvido para criação de aplicações colaborativas em tempo real que utiliza CRDTs para gerenciar dados distribuídos e garantir consistência eventual mesmo em cenários de conectividade instável ou funcionamento offline. O framework possibilita a implementação de editores de texto, sistemas de desenho colaborativo e tabelas dinâmicas, demonstrando robustez em experimentos com mais de 100 usuários simultâneos (WEIDNER et al., 2023).

Neste trabalho, adotou-se uma abordagem de sincronização com CRDTs baseados em operações, na qual cada modificação realizada pelos usuários é representada por uma operação e propagada aos demais participantes da sessão.

3 Materiais e Método

Este capítulo apresenta os materiais utilizados e os procedimentos metodológicos adotados para a realização desta pesquisa. O desenvolvimento do trabalho foi estruturado em quatro etapas principais: (i) execução de uma Revisão Sistemática da Literatura (RSL); (ii) definição da arquitetura do protótipo proposto, baseada nos resultados obtidos na revisão e nos requisitos identificados; (iii) desenvolvimento de um protótipo de um editor colaborativo de DER; e (iv) avaliação do protótipo, por meio de testes de convergência do mecanismo de sincronização e de testes de concorrência e carga, com o objetivo de verificar o comportamento da solução em cenários de colaboração simultânea.

3.1 Revisão Sistemática da Literatura

A primeira etapa da pesquisa consistiu na realização de uma Revisão Sistemática da Literatura (RSL), conduzida conforme o protocolo proposto por Biolchini et al. (2005). O objetivo dessa etapa foi identificar, analisar e sintetizar as principais arquiteturas, tecnologias e mecanismos de sincronização empregados no desenvolvimento de sistemas web colaborativos, fornecendo embasamento científico para a definição da arquitetura proposta neste trabalho.

A questão de pesquisa definida para a revisão foi: *Quais arquiteturas e tecnologias têm sido utilizadas no desenvolvimento de sistemas web colaborativos, segundo a literatura?* A partir dessa questão, foram estabelecidos os critérios de seleção dos estudos e a estratégia de busca utilizada nas bases de dados científicas.

Foram definidos como critérios de inclusão: a publicação do estudo entre 2015 e 2025; a redação em português ou inglês; a disponibilidade do texto completo na web, por meio das bases consultadas; e a abordagem de arquiteturas ou tecnologias utilizadas no desenvolvimento de sistemas web colaborativos. Foram excluídos os estudos publicados fora do período estabelecido, redigidos em outros idiomas, sem acesso ao texto completo ou que não apresentassem relação com o objetivo da revisão.

A estratégia de busca foi construída a partir de palavras-chave relacionadas aos conceitos de colaboração, comunicação em tempo real, tecnologias, ferramentas, arquitetura de software e aplicações web. As palavras-chave utilizadas em inglês foram *Collaboration*, *real-time*, *technology*, *tool*, *software architecture*, *web system* e *web application*. Em português, foram utilizados os termos *Colaboração*, *tempo real*, *tecnologia*, *ferramentas*, *arquitetura de software*, *sistemas web* e *aplicações web*.

A partir dessas palavras-chave, foram definidas as seguintes strings de busca. Para

os trabalhos em inglês, utilizou-se:

```
((collaboration OR realtime) AND (technology OR tool OR "software
architecture") AND ("web system" OR "web application"))
```

Para os trabalhos em português, foi utilizada a seguinte string:

```
((colaboração OR tempo real) AND (tecnologia OR ferramenta OR arquitetura
de software) AND (sistemas web OR aplicação web))
```

As buscas foram realizadas nas bases IEEE Xplore, ACM Digital Library, Scopus, Elsevier ScienceDirect e SpringerLink. A execução da revisão ocorreu entre junho e setembro de 2025.

Inicialmente, foram recuperados 900 estudos a partir das strings de busca. Na seleção primária, foram analisados os títulos e as palavras-chave dos trabalhos, resultando em 65 estudos potencialmente relevantes. Em seguida, foi realizada a seleção secundária, na qual foram analisados os resumos e as conclusões dos estudos selecionados na etapa anterior. Nessa etapa, os trabalhos foram classificados de acordo com sua relevância, ocorrência de duplicidade e disponibilidade das informações necessárias à revisão.

Ao final do processo de seleção, 12 estudos foram considerados relevantes e incluídos na análise detalhada. A distribuição dos resultados obtidos nas diferentes bases de dados e etapas da seleção é apresentada na Tabela 2.

Tabela 2 – Resultados das etapas de seleção da RSL

Base	**Resultados**	**Primária**	**Irrelevantes**	**Repetidos**	**Incompletos**	**Final**
IEEE Xplore	287	19	11	0	3	5
ACM	72	5	4	0	0	1
SpringerLink	234	8	3	0	2	3
Scopus	250	31	11	5	12	3
ScienceDirect	57	2	0	0	2	0
Total	**900**	**65**	**29**	**5**	**19**	**12**

Fonte: Elaborada pela autora (2026).

Durante a execução da revisão, os estudos recuperados foram tratados com o auxílio do software Zotero, utilizado para o gerenciamento das referências bibliográficas, remoção de duplicatas e organização dos metadados dos trabalhos selecionados.

Após a seleção dos estudos primários, os trabalhos selecionados foram analisados considerando aspectos relacionados à arquitetura de software, mecanismos de comunicação em tempo real, técnicas de sincronização, tecnologias empregadas e estratégias utilizadas para o tratamento de alterações concorrentes.

A análise realizada na RSL permitiu identificar padrões recorrentes entre os estudos selecionados, especialmente em relação à utilização de arquiteturas cliente-servidor, mecanismos de comunicação em tempo real e técnicas de sincronização para o tratamento de alterações concorrentes. Essas evidências foram utilizadas como referência para a definição da arquitetura do protótipo desenvolvido neste trabalho.

3.2 Definição da Arquitetura

A segunda etapa da pesquisa consistiu na definição da arquitetura do protótipo proposto. Para isso, foram levantados os requisitos funcionais e não funcionais que orientaram o desenvolvimento do editor colaborativo de DER. Os requisitos funcionais foram definidos a partir da análise das funcionalidades oferecidas por ferramentas de modelagem de diagramas, como o MySQL Workbench e o drawDB, considerando os recursos pertinentes ao escopo do protótipo. Os requisitos não funcionais, por sua vez, foram estabelecidos com base nas evidências obtidas na RSL, considerando as características arquiteturais e os mecanismos necessários à colaboração e à sincronização das alterações. Os requisitos definidos são apresentados na Tabela 3.

Tabela 3 – Requisitos considerados na definição da arquitetura

Código	**Tipo**	**Requisito**
RF01	Funcional	Propagar as alterações realizadas por um usuário aos demais participantes em tempo real.
RF02	Funcional	Permitir a edição colaborativa de um diagrama por múltiplos usuários simultaneamente.
RF03	Funcional	Permitir desfazer e refazer operações realizadas durante a edição.
RF04	Funcional	Permitir gerenciamento de projetos e participantes.
RF05	Funcional	Permitir o controle de acesso dos usuários aos projetos.
RNF01	Não funcional	Manter a consistência do estado compartilhado durante edições concorrentes.
RNF02	Não funcional	Garantir a convergência das réplicas após o processamento das operações.
RNF03	Não funcional	Oferecer comunicação em tempo real entre os participantes.

Código	Tipo	Requisito
RNF04	Não funcional	Manter separação de responsabilidades entre apresentação, negócio, comunicação e persistência.
RNF05	Não funcional	Permitir manutenção e evolução da aplicação.

Fonte: Elaborada pela autora (2026).

Inicialmente, foram analisadas as arquiteturas e tecnologias identificadas na literatura. Os estudos selecionados evidenciaram a predominância de arquiteturas cliente-servidor em sistemas colaborativos, frequentemente associadas ao uso de WebSocket para comunicação em tempo real e de mecanismos de sincronização, como OT e CRDT. Essas evidências serviram como ponto de partida para a definição da arquitetura do protótipo.

Embora um dos trabalhos analisados tenha adotado uma arquitetura baseada em microsserviços, optou-se, neste trabalho, pela utilização de um monólito organizado em camadas. Essa decisão considerou o escopo e as necessidades do protótipo, para os quais a separação da aplicação em serviços independentes acrescentaria complexidade arquitetural sem benefícios proporcionais. A literatura aponta que arquiteturas baseadas em microsserviços podem envolver desafios relacionados à distribuição dos componentes, à comunicação entre serviços e à infraestrutura necessária para sua operação (DRAGONI et al., 2017). Dessa forma, a organização da aplicação em camadas mostrou-se adequada para separar as responsabilidades e facilitar a manutenção, preservando a possibilidade de evolução futura da solução.

A comunicação entre os clientes e o servidor foi implementada por meio de WebSocket, tecnologia muito utilizada nos estudos analisados e adequada para aplicações que exigem troca contínua de mensagens com baixa latência. Sua utilização possibilita que as alterações realizadas por um usuário sejam imediatamente propagadas aos demais participantes da sessão colaborativa.

Para garantir a consistência dos diagramas durante edições simultâneas, optou-se por um mecanismo de replicação baseado em operações, fundamentado nos princípios de CRDT. Embora os mecanismos baseados em OT tenham sido mais recorrentes nos estudos da RSL, a escolha do CRDT considerou sua adequação à arquitetura proposta, especialmente pela possibilidade de processar operações independentemente em cada réplica, sem depender de um componente central para transformar operações concorrentes (SUN et al., 2020). No CoDER, as alterações realizadas pelos usuários são representadas como operações independentes, propagadas entre os clientes e processadas localmente segundo critérios determinísticos. Dessa forma, espera-se que as réplicas alcancem estados equivalentes após receberem o mesmo conjunto de operações. A convergência do mecanismo implementado foi posteriormente verificada por meio dos testes descritos na etapa de

avaliação.

Embora frameworks como Yjs e Automerge disponibilizem implementações consolidadas de mecanismos CRDT, optou-se, neste trabalho, pela implementação de um mecanismo próprio. Essa decisão teve como objetivo permitir maior controle sobre as operações específicas do domínio de modelagem de DER e possibilitar a análise direta dos mecanismos envolvidos na resolução de alterações concorrentes. Dessa forma, a implementação não teve como objetivo competir com frameworks CRDT de propósito geral, mas atender às necessidades específicas do protótipo e permitir a avaliação do mecanismo desenvolvido.

No fluxo de edição, as interações realizadas sobre o diagrama são capturadas como eventos pela biblioteca Vue Flow. Ações como criação, remoção, movimentação e edição de elementos são convertidas em operações independentes, que representam as alterações a serem aplicadas ao estado do diagrama. Essas operações são então processadas pelo mecanismo de sincronização e transmitidas aos demais clientes por meio do WebSocket.

Outra decisão arquitetural importante consistiu em concentrar toda a lógica de sincronização nas réplicas mantidas pelos clientes. Assim, cada cliente é responsável por aplicar localmente, por meio do algoritmo CRDT, as operações recebidas, enquanto o servidor atua apenas como intermediador das mensagens recebidas via WebSocket e como responsável pela persistência dos dados do projeto. Essa abordagem reduz a complexidade do servidor, descentraliza o tratamento de conflitos e mantém a responsabilidade pela convergência do estado compartilhado nos próprios clientes.

Ao final dessa etapa, foram definidos os principais componentes da arquitetura, suas responsabilidades e os mecanismos de comunicação entre cliente, servidor e banco de dados. A arquitetura resultante é apresentada na Figura 2, estabelecendo a base necessária para o desenvolvimento do protótipo apresentado na etapa seguinte.

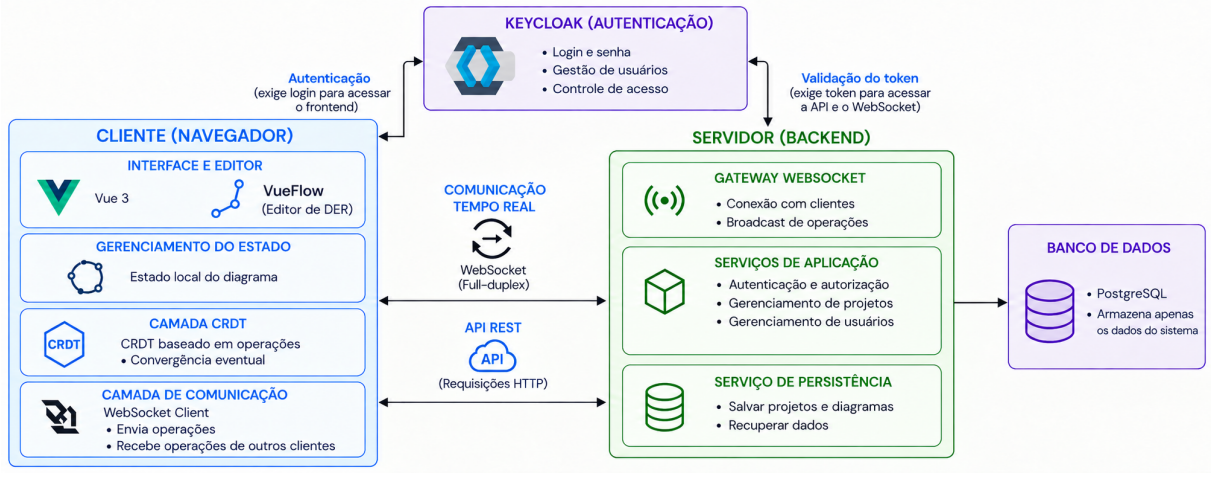


Figura 2 – Arquitetura proposta para o editor colaborativo de DER

Fonte: Elaborada pela autora (2026).

3.3 Desenvolvimento do Protótipo

A terceira etapa da pesquisa consistiu no desenvolvimento do protótipo funcional do editor colaborativo de DER, denominado CoDER. Essa etapa teve como objetivo implementar a arquitetura proposta e materializar as decisões arquiteturais definidas a partir da RSL.

O desenvolvimento foi conduzido de forma incremental, permitindo que cada funcionalidade fosse implementada, integrada e validada antes do início da etapa seguinte. Para o gerenciamento do projeto foi utilizado o sistema de controle de versão Git, tendo o GitLab como plataforma para hospedagem do repositório e acompanhamento da evolução do código-fonte.

Inicialmente foram definidas as tecnologias que comporiam a solução. Essa escolha considerou principalmente a compatibilidade com a arquitetura proposta, a disponibilidade de ferramentas para desenvolvimento de aplicações colaborativas e a experiência prévia da pesquisadora, permitindo concentrar os esforços nos desafios específicos da pesquisa.

As principais tecnologias utilizadas são apresentadas na Tabela 4.

Tabela 4 – Tecnologias utilizadas no desenvolvimento do protótipo

Tecnologia	Versão	Finalidade
Java	17	Backend
Spring Boot	3.3.5	Framework backend
Vue.js	3.5.24	Frontend
Vue Flow	1.48.1	Editor gráfico
Vuetify	3.11.8	Componentes visuais
Vitest	4.1.8	Testes automatizados do frontend
Pinia	3.0.4	Gerenciamento de estado
PostgreSQL	17	Banco de dados
Liquibase	4.27.0	Controle de versão do banco de dados
JavaScript	ES2022	Linguagem de programação do frontend
Node.js	22.21.0	Ambiente de execução JavaScript
Keycloak	26.2.5	Autenticação e autorização
Docker	28	Containerização
Git	2.x	Controle de versão
GitLab	-	Hospedagem do repositório

Fonte: Elaborada pela autora (2026).

Após a definição das tecnologias, foi realizada a prototipação da interface da aplicação, utilizando o ChatGPT como ferramenta de apoio. A ferramenta foi utilizada para auxiliar na elaboração do layout inicial das telas, sugerindo formas de organizar os elementos visuais e os componentes da interface. As sugestões geradas serviram como ponto de partida para a implementação, sendo avaliadas e adaptadas conforme as necessidades do protótipo.

Em seguida, foi realizada a modelagem do banco de dados, com a definição da estrutura necessária para armazenar as informações dos usuários, projetos, permissões de acesso e diagramas. Para o armazenamento dos diagramas, optou-se por persistir o documento JSON gerado pela biblioteca Vue Flow, preservando a estrutura gráfica utilizada pelo editor. Essa abordagem simplificou a representação dos diagramas na camada de persistência e evitou a necessidade de decompor cada elemento gráfico em registros separados no banco de dados. A estrutura definida para a persistência dos dados da aplicação é apresentada na Figura 3, que representa o modelo de dados do CoDER e suas principais entidades e relacionamentos.

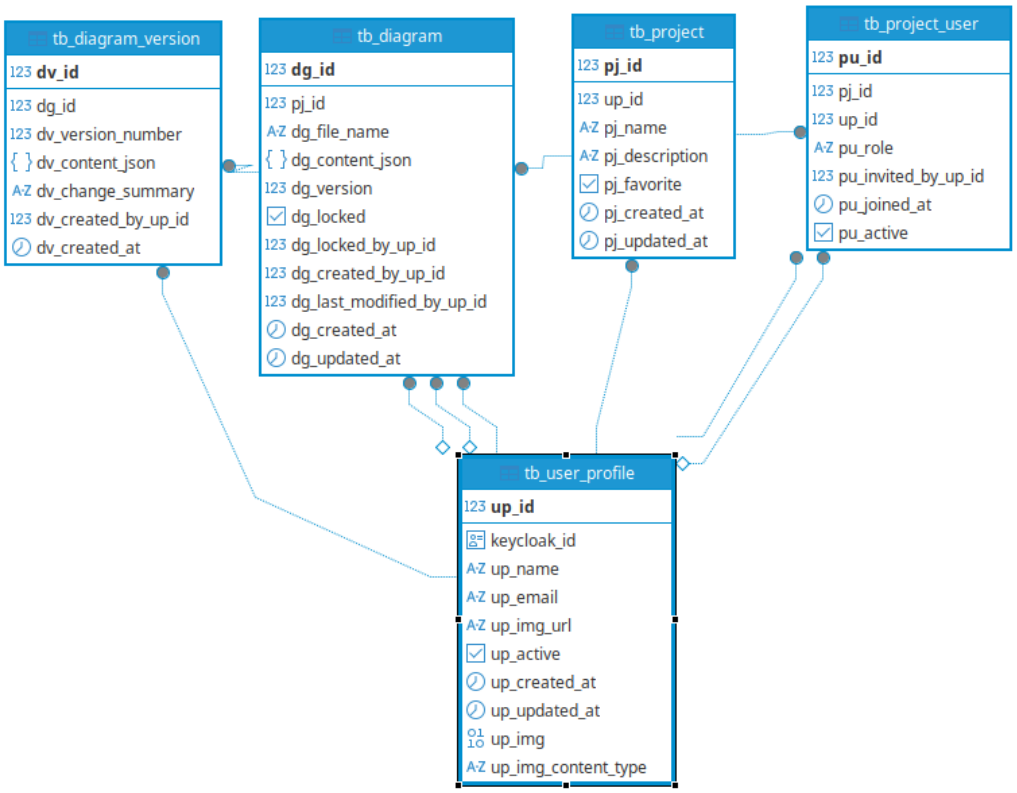


Figura 3 – Modelo de dados do CoDER

Fonte: Elaborada pela autora (2026).

A implementação teve início pelo frontend da aplicação utilizando Vue.js. Para construção do editor gráfico foi empregada a biblioteca Vue Flow, escolhida por disponibilizar recursos específicos para manipulação de grafos, geração de eventos para as interações realizadas pelo usuário e facilidade de personalização da interface, características importantes para o desenvolvimento do editor colaborativo.

Na sequência, o backend foi desenvolvido com Java e Spring Boot. A integração com a interface previamente implementada ocorreu de forma incremental, à medida que as funcionalidades eram desenvolvidas. Essa etapa contemplou o gerenciamento de usuários, projetos e diagramas, além dos recursos de autenticação e persistência dos dados.

A autenticação e autorização da aplicação foram implementadas utilizando o Keycloak. A escolha dessa solução permitiu desacoplar o gerenciamento de identidades da lógica da aplicação, utilizando um servidor dedicado para autenticação, além de proporcionar à pesquisadora experiência prática com uma tecnologia adotada no ambiente profissional em que atua.

Após a conclusão das funcionalidades básicas, foi implementado o mecanismo de colaboração em tempo real utilizando WebSocket e um algoritmo CRDT baseado em operações. Também foram desenvolvidas as funcionalidades de desfazer e refazer operações, permitindo ao usuário reverter e reaplicar alterações realizadas durante a edição. Esses mecanismos são mantidos no histórico local do cliente, não sendo propagados como operações colaborativas para os demais participantes.

Posteriormente, o protótipo foi implantado na plataforma Render utilizando contêineres Docker. A utilização do Docker permitiu padronizar o ambiente de execução da aplicação e simplificar sua implantação, enquanto a plataforma Render possibilitou disponibilizar o sistema gratuitamente em um ambiente acessível pela internet para realização de testes exploratórios.

Por fim, foram realizados ajustes finais relacionados à colaboração em tempo real, estabilidade da aplicação e experiência do usuário, resultando em um protótipo funcional capaz de validar a arquitetura proposta nesta pesquisa.

3.4 Avaliação

A avaliação do protótipo teve como objetivo verificar o comportamento da solução em relação à sincronização do estado compartilhado e à comunicação colaborativa em tempo real. Para isso, foram realizados testes de convergência do mecanismo de sincronização e testes de concorrência e carga, descritos nas subseções a seguir.

3.4.1 Ambiente de execução

Os experimentos foram realizados em ambiente local, utilizando um único computador para a execução dos componentes envolvidos nos testes. A máquina utilizada possui processador Intel Core i7-10510U, com quatro núcleos físicos e suporte a oito threads, e 32 GB de memória RAM. O sistema operacional utilizado foi o Linux Mint 22.2, baseado no Ubuntu 24.04, em arquitetura 64 bits.

O backend da aplicação, o servidor de autenticação e os demais serviços necessários ao funcionamento do protótipo foram executados no ambiente local, assim como o script responsável pela geração da carga. Dessa forma, os clientes utilizados nos testes de concorrência foram simulados por processos executados no mesmo host do servidor.

O backend foi executado utilizando a JVM correspondente à versão Java 17 definida para o projeto, enquanto os testes de concorrência foram executados com Node.js. Os serviços executados em contêineres utilizaram Docker. Sendo assim, os experimentos foram conduzidos em um ambiente controlado, sem a introdução de diferentes condições de rede entre os clientes e o servidor.

Essa configuração deve ser considerada na interpretação dos resultados, pois os testes avaliam o comportamento da comunicação e do protótipo sob diferentes cargas de usuários simulados, mas não reproduzem as condições de uma implantação distribuída em diferentes máquinas ou redes.

3.4.2 Testes de convergência

Os testes de convergência tiveram como objetivo verificar se diferentes réplicas do estado compartilhado alcançam o mesmo estado final após o processamento de operações, incluindo situações em que as operações são recebidas em ordens diferentes. Essa avaliação foi realizada diretamente sobre o mecanismo de sincronização implementado, sem a utilização da camada de comunicação WebSocket, permitindo isolar o comportamento do algoritmo CRDT.

Os testes foram implementados com o framework Vitest e executados em memória. Foram definidos 20 cenários de teste, contemplando operações relacionadas às diferentes alterações que podem ser realizadas sobre os elementos do diagrama, bem como situações de concorrência, duplicação e processamento de operações em diferentes ordens. Para cada cenário, foi definida uma propriedade a ser verificada após a aplicação das operações. A Tabela 5 apresenta os cenários considerados na avaliação, as operações envolvidas e as respectivas propriedades verificadas.

Considerou-se que um teste foi bem-sucedido quando o comportamento observado atendeu à propriedade definida para o respectivo cenário. Nos cenários que avaliam convergência, isso corresponde à obtenção de estados equivalentes entre as réplicas, independentemente da ordem em que as operações foram processadas.

Tabela 5 – Cenários dos testes de convergência

ID	**Cenário**	**Operações envolvidas**	**Propriedade verificada**
01	Diferentes ordens de aplicação	Adicionar entidade; alterar nome da entidade	Convergência das réplicas após a aplicação das mesmas operações em ordens distintas.

ID	Cenário	Operações envolvidas	Propriedade verificada
02	Conflito concorrente de nome de entidade	Alterar nome da mesma entidade por usuários diferentes	Resolução determinística de alterações concorrentes.
03	Aplicação duplicada de operação	Aplicar a mesma operação de adição de entidade duas vezes	Idempotência da aplicação e prevenção da duplicação da entidade.
04	Conflito concorrente de movimentação de entidade	Mover a mesma entidade por usuários diferentes	Resolução determinística de posições concorrentes e convergência das réplicas.
05	Renomeação e exclusão concorrentes de entidade	Alterar nome e excluir a mesma entidade por usuários diferentes	Convergência diante de operações concorrentes de alteração e exclusão.
06	Movimentação e exclusão concorrentes de entidade	Mover entidade; excluir entidade	Convergência diante de operações concorrentes de alteração e exclusão.
07	Entrega de operações fora de ordem	Adicionar entidade; alterar nome; mover entidade	Convergência independentemente da ordem de entrega das operações.
08	Operações geradas em réplicas distintas	Adicionar uma entidade em cada réplica	Convergência após a aplicação das operações geradas separadamente nas réplicas.
09	Cem operações com ordem de aplicação aleatória	Adicionar 100 entidades	Convergência entre duas réplicas após a aplicação das mesmas operações em ordens distintas.
10	Dez réplicas com ordem de aplicação aleatória	Adicionar 50 entidades	Convergência entre dez réplicas após a aplicação das mesmas operações em ordens distintas.
11	Conflito concorrente de nome de atributo	Alterar o nome do mesmo atributo por usuários diferentes	Resolução determinística de alterações concorrentes e convergência das réplicas.

ID	Cenário	Operações envolvidas	Propriedade verificada
12	Renomeação e remoção concorrentes de atributo	Alterar o nome e remover o mesmo atributo por usuários diferentes	Prevalência da remoção e convergência das réplicas independentemente da ordem de aplicação.
13	Conflito concorrente de tipo de atributo	Alterar o tipo do mesmo atributo por usuários diferentes	Resolução determinística de alterações concorrentes e convergência das réplicas.
14	Conflito concorrente de chave primária	Definir a chave primária do mesmo atributo por usuários diferentes	Resolução determinística de valores concorrentes e convergência das réplicas.
15	Conflito concorrente de chave estrangeira	Definir a chave estrangeira do mesmo atributo por usuários diferentes	Resolução determinística de valores concorrentes e convergência das réplicas.
16	Adição de relacionamento	Adicionar duas entidades e um relacionamento entre elas	Convergência das réplicas após a aplicação das mesmas operações em ordens distintas.
17	Conflito concorrente de cardinalidade	Alterar cardinalidade do mesmo relacionamento por usuários diferentes	Resolução determinística de alterações concorrentes e convergência das réplicas.
18	Atualização e exclusão concorrentes de relacionamento	Alterar a relação e excluir o mesmo relacionamento por usuários diferentes	Convergência das réplicas diante de operações concorrentes de alteração e exclusão.
19	Desfazer adição de entidade	Adicionar entidade; desfazer operação	Remoção do efeito da operação de adição por meio do mecanismo de desfazer.
20	Refazer adição de entidade	Adicionar entidade; desfazer operação e refazer operação	Restauração do efeito da operação de adição por meio do mecanismo de refazer.

Fonte: Elaborada pela autora (2026).

3.4.3 Testes de concorrência e carga

Os testes de concorrência tiveram como objetivo avaliar o comportamento da comunicação em tempo real do protótipo diante do aumento do número de usuários e da carga de operações. Diferentemente dos testes de convergência, esses testes não tiveram como objetivo verificar diretamente o algoritmo CRDT, mas avaliar o funcionamento da comunicação entre os clientes e o servidor em condições de concorrência.

Os testes foram implementados em Node.js utilizando as bibliotecas `@stomp/stompjs` e `ws`. Inicialmente, foram autenticados 20 usuários no Keycloak, que ficaram disponíveis para composição dos diferentes cenários de teste. Para cada cenário, os usuários selecionados foram conectados simultaneamente ao canal WebSocket correspondente ao projeto utilizado no experimento. Após a confirmação da conexão de todos os participantes, a carga foi iniciada de forma concorrente.

Foram definidos cinco cenários, com 2, 5, 10, 15 e 20 usuários. Em cada cenário, cada usuário enviou 20 operações de adição de entidades (`ADD_NODE`), totalizando, respectivamente, 40, 100, 200, 300 e 400 operações por execução. O intervalo entre o envio das operações foi fixado em 100 ms. Cada cenário foi executado cinco vezes, mantendo-se as mesmas configurações entre as repetições.

A cada execução, o teste registrou a quantidade de operações enviadas e de entregas esperadas e realizadas. Como cada operação deveria ser recebida por todos os usuários conectados ao projeto, a quantidade de entregas esperadas foi determinada pela multiplicação do número de operações enviadas pelo número de participantes. A partir desses valores, foram identificadas eventuais mensagens perdidas e calculada a taxa de entrega.

A latência de cada mensagem foi obtida pela diferença entre o instante em que a operação foi enviada e o instante em que sua mensagem correspondente foi recebida pelo cliente. A partir das latências observadas em cada execução, foram calculadas as medidas mínima, máxima, média, mediana, desvio padrão e os percentis P95 e P99. Ao final das cinco repetições de cada cenário, o script consolidou os resultados por meio da média e do desvio padrão das métricas selecionadas, apresentando os valores de latência média, P95, P99 e throughput.

A carga oferecida foi determinada em função do número de usuários e do intervalo entre operações, conforme a expressão:

$$C = U \times \frac{1000}{I}$$

em que C representa a carga oferecida em operações por segundo, U representa o número de usuários participantes e I representa o intervalo entre operações, em milis-

segundos. Com o intervalo fixado em 100 ms, os cinco cenários correspondem a cargas oferecidas de 20, 50, 100, 150 e 200 operações por segundo.

O throughput foi obtido a partir da quantidade de operações processadas e do tempo total observado em cada execução. Essa métrica foi utilizada como informação complementar à carga oferecida, uma vez que a taxa de geração das operações foi previamente controlada pelo intervalo definido no experimento.

Os resultados consolidados dos testes são apresentados e discutidos no capítulo seguinte.

4 Resultados e discussões

Este capítulo apresenta os resultados obtidos com o desenvolvimento e a avaliação do protótipo CoDER. Inicialmente, é apresentada uma visão geral do protótipo desenvolvido, seguida pela descrição das principais funcionalidades implementadas. Essa descrição tem como objetivo contextualizar os recursos disponibilizados pelo protótipo e demonstrar sua aplicação no cenário de edição colaborativa de DER. Em seguida, é apresentada a arquitetura implementada, detalhando seus principais componentes e as tecnologias utilizadas. Por fim, são apresentados os resultados da avaliação do protótipo, contemplando os testes de convergência e os testes de concorrência e carga, seguidos da discussão dos resultados obtidos.

O código-fonte do projeto está disponível em um repositório Git, e o acesso pode ser realizado por meio do seguinte endereço: <https://gitlab.com/DanielleLeal/dereditor.git>.

4.1 Visão Geral do Protótipo

O CoDER (*Collaborative Entity-Relationship Diagram*) é um protótipo de aplicação web para criação e edição colaborativa de DER. A aplicação permite que usuários criem projetos, elaborem seus diagramas e compartilhem o acesso com outros participantes, possibilitando a realização de alterações de forma colaborativa.

Entre as funcionalidades disponibilizadas estão o gerenciamento de projetos e participantes, o controle de permissões de acesso, a criação e edição de elementos dos diagramas, a movimentação e remoção de elementos, além dos recursos de desfazer e refazer operações. A aplicação também permite acompanhar as alterações realizadas pelos demais participantes durante uma sessão de edição colaborativa.

A Figura 4 apresenta a interface principal do CoDER após a autenticação do usuário, evidenciando o ponto de entrada para as funcionalidades disponibilizadas pelo protótipo.

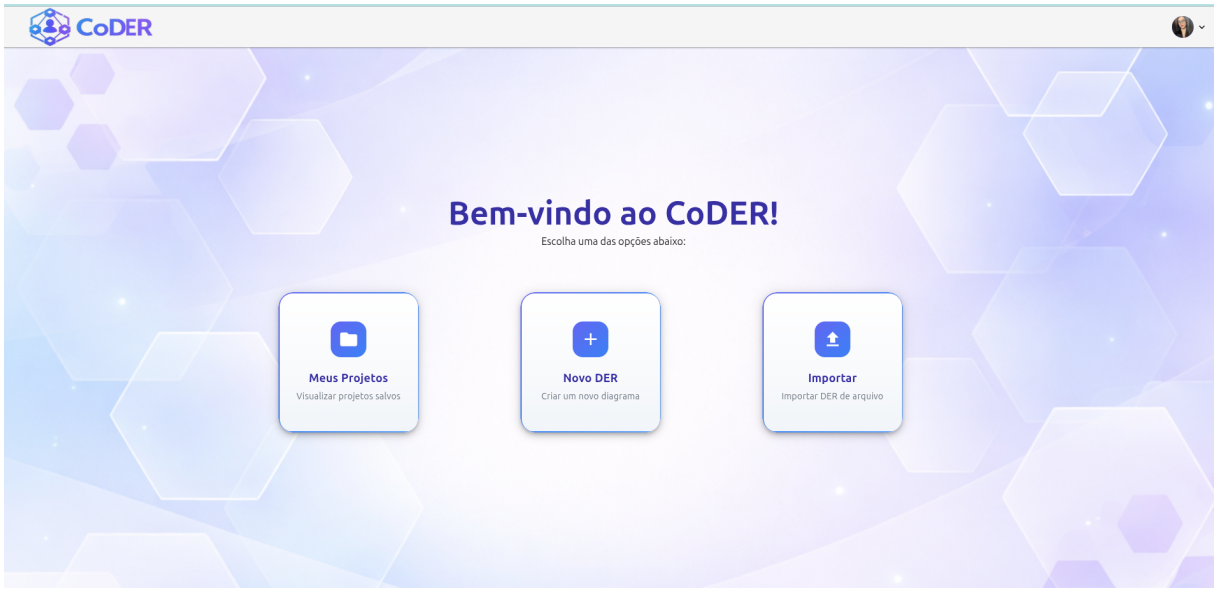


Figura 4 – Interface principal do CoDER após a autenticação do usuário

Fonte: Elaborada pela autora (2026).

4.2 Funcionalidades Implementadas

4.2.1 Autenticação de usuários

A autenticação e o gerenciamento das identidades dos usuários foram realizados por meio do *Keycloak*, utilizado como componente externo responsável por essas atividades. Entre as funcionalidades disponibilizadas estão o login, o cadastro de novos usuários, a verificação do endereço de e-mail, a recuperação de senha e a edição do perfil. Dessa forma, essas responsabilidades são delegadas ao provedor de identidade, reduzindo a necessidade de implementação e manutenção desses mecanismos no núcleo do CoDER.

Para armazenar as informações de perfil utilizadas pela aplicação, foi criada a tabela `tb_user_profile`, contendo dados como nome, e-mail, foto e o identificador do usuário no *Keycloak* (`keycloak_id`). Após a autenticação, o CoDER verifica a existência de um perfil associado ao identificador recebido. Caso não exista um registro correspondente, um novo perfil é criado na base de dados da aplicação. Nos acessos seguintes, o perfil previamente cadastrado é recuperado e utilizado pela aplicação.

Além da configuração do processo de autenticação, foi desenvolvido um tema personalizado para o *Keycloak*, adaptado à identidade visual do CoDER. A personalização foi aplicada às principais telas relacionadas ao gerenciamento da conta, incluindo autenticação, cadastro de usuários e recuperação de senha. Dessa forma, embora essas funcionalidades sejam executadas por um componente externo ao núcleo da aplicação, sua apresentação visual permanece integrada à interface do protótipo.

A Figura 5 apresenta a tela de autenticação personalizada do CoDER, enquanto a

Figura 6 apresenta a tela de cadastro de novos usuários.

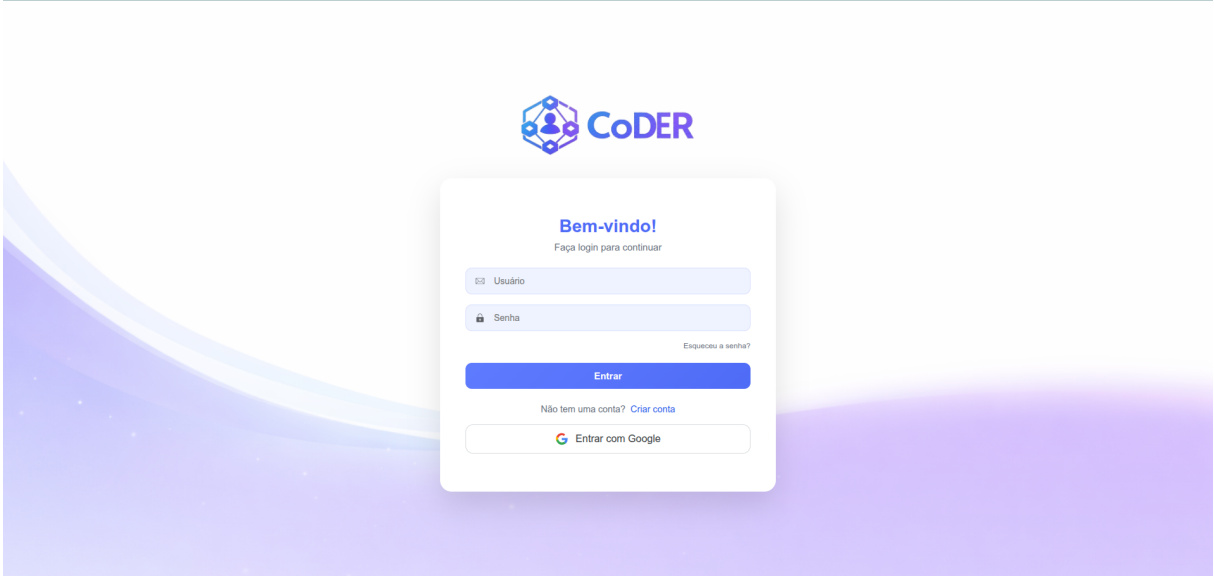


Figura 5 – Tela de autenticação do CoDER

Fonte: Elaborada pela autora (2026).

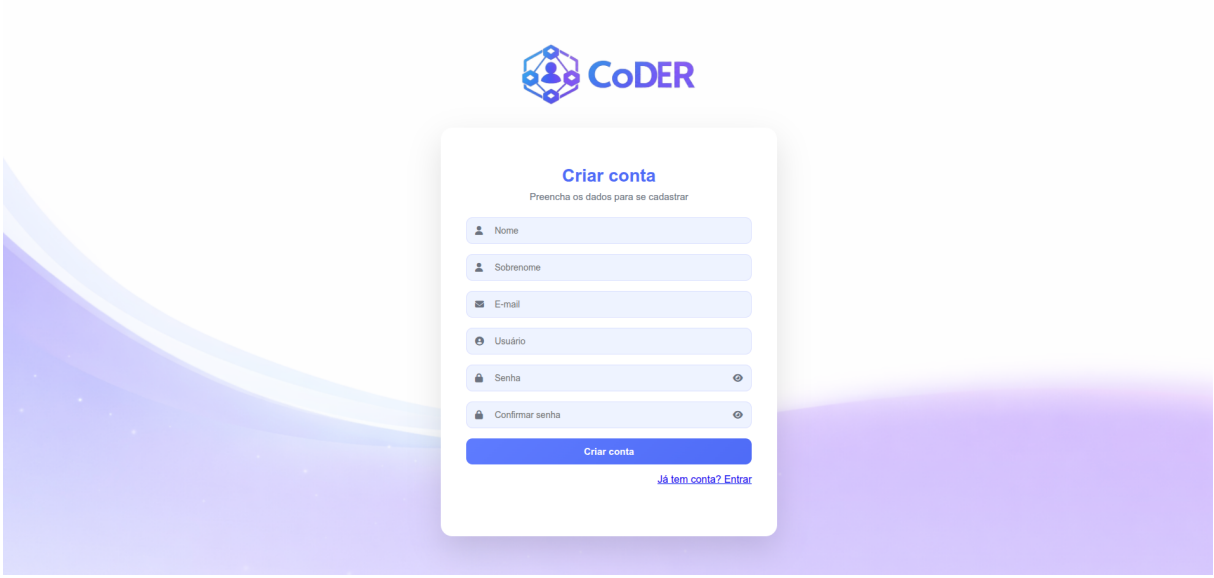


Figura 6 – Tela de cadastro de novos usuários

Fonte: Elaborada pela autora (2026).

4.2.2 Gerenciamento de projetos

Após a autenticação, o usuário pode acessar seus projetos a partir da tela inicial do CoDER. Um novo projeto pode ser criado diretamente pela página inicial ou por meio da opção *Meus Projetos*, que apresenta os projetos criados pelo usuário.

A tela de gerenciamento de projetos, apresentada na Figura 7, disponibiliza recursos para organização e localização dos projetos. A listagem é apresentada de forma paginada e

possui mecanismos de pesquisa por nome e de ordenação, permitindo organizar os projetos em ordem alfabética crescente ou por data de criação, possibilitando a visualização dos projetos mais recentes ou mais antigos.

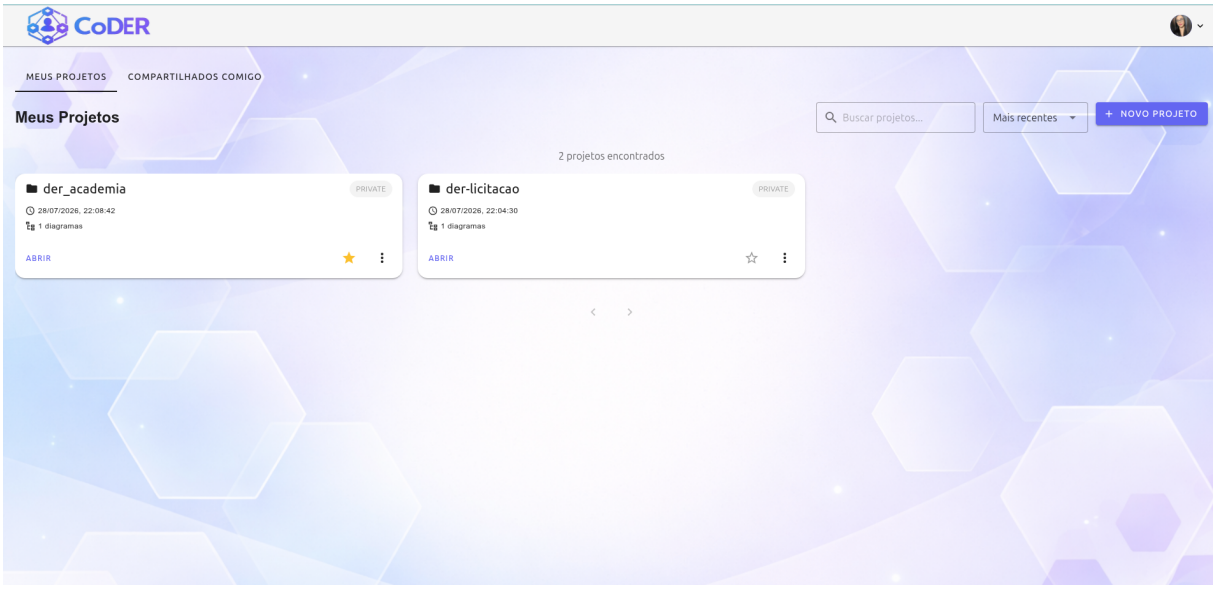


Figura 7 – Tela de gerenciamento de projetos do CoDER

Fonte: Elaborada pela autora (2026).

Além dos projetos criados pelo próprio usuário, a tela disponibiliza uma segunda aba denominada *Compartilhados comigo*, apresentada na Figura 8, destinada à visualização dos projetos aos quais o usuário recebeu acesso por meio do compartilhamento realizado por outro usuário.

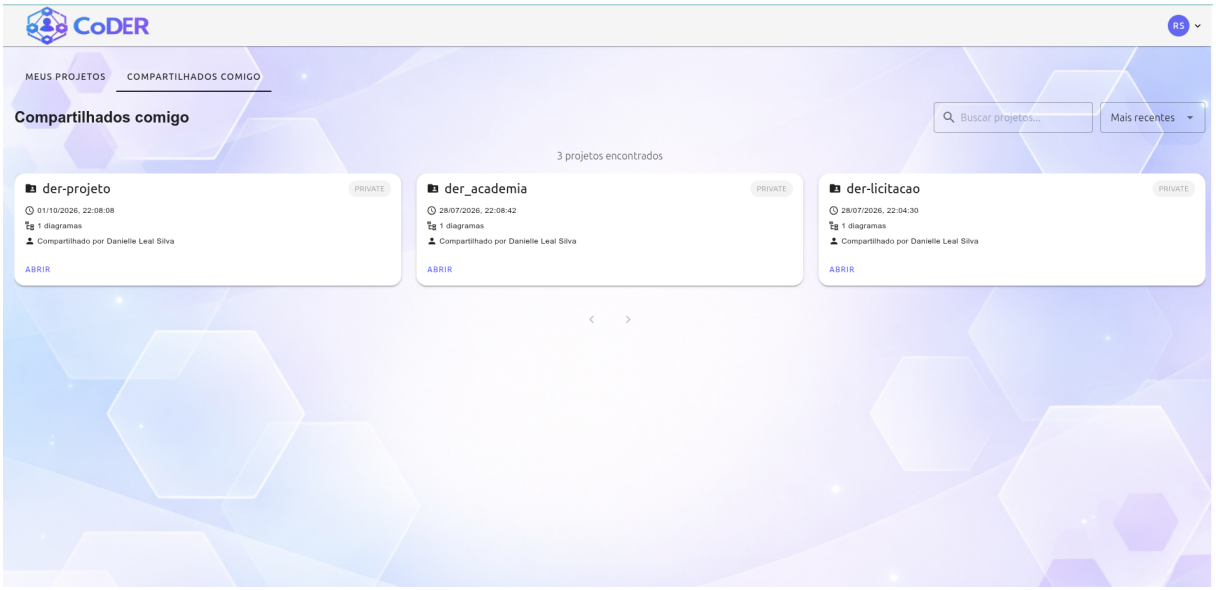


Figura 8 – Tela de projetos compartilhados com o usuário

Fonte: Elaborada pela autora (2026).

Para cada projeto, são disponibilizadas operações de gerenciamento por meio de um menu de opções, incluindo renomeação, exclusão e exportação. A exportação gera um arquivo com extensão *.coder*, que consiste em uma representação do projeto no formato JSON, estruturada segundo o modelo de dados utilizado pelo CoDER. Esse arquivo pode ser posteriormente utilizado pela funcionalidade de importação, permitindo recuperar o projeto no protótipo e facilitando o transporte e a reutilização dos diagramas.

4.2.3 Editor de Diagramas Entidade-Relacionamento

A principal funcionalidade do CoDER é a criação e edição de DER. Após acessar um projeto, o usuário é direcionado ao editor, apresentado na Figura 9, no qual pode construir e modificar o diagrama de forma visual.

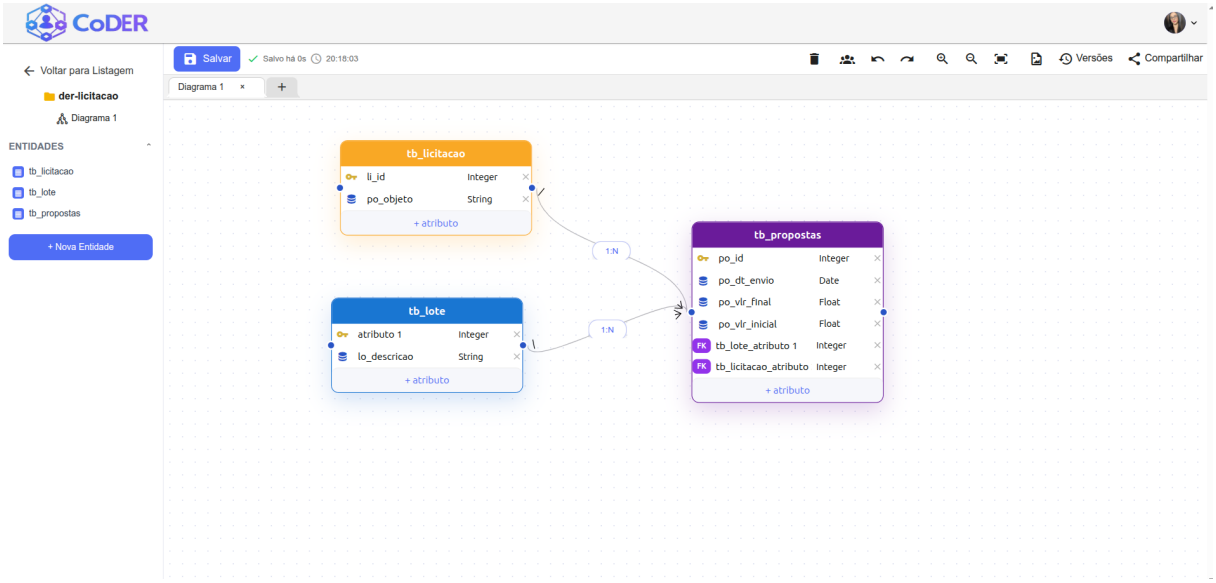


Figura 9 – Editor de Diagramas Entidade-Relacionamento do CoDER

Fonte: Elaborada pela autora (2026).

O editor permite a criação de entidades, atributos e relacionamentos, além da definição de chaves primárias, incluindo chaves primárias compostas, e da configuração das cardinalidades dos relacionamentos. Também permite a especificação de tipos de dados para os atributos e a representação de chaves estrangeiras. Os elementos do diagrama podem ser renomeados e modificados conforme a necessidade da modelagem. O editor disponibiliza ainda operações de edição, como exclusão de elementos, desfazer e refazer ações, além de recursos de aproximação e afastamento da área de trabalho (*zoom in* e *zoom out*). O diagrama pode ser exportado no formato PNG.

Embora seja denominado modelador de DER, o editor reúne elementos associados à modelagem conceitual e à modelagem lógica relacional. A modelagem conceitual representa entidades, atributos e relacionamentos em um nível mais abstrato, enquanto a modelagem lógica relacional define a estrutura das tabelas e aspectos como tipos de dados, chaves

primárias e chaves estrangeiras. Dessa forma, o editor do CoDER incorpora características das duas etapas da modelagem de dados.

O CoDER possui mecanismos de persistência das alterações realizadas. O salvamento automático verifica, a cada 60 segundos, a existência de alterações pendentes e realiza o salvamento quando necessário. Além disso, o usuário pode registrar manualmente uma nova versão do diagrama e consultar as versões previamente salvas, podendo restaurá-las quando necessário, conforme apresentado na Figura 10.

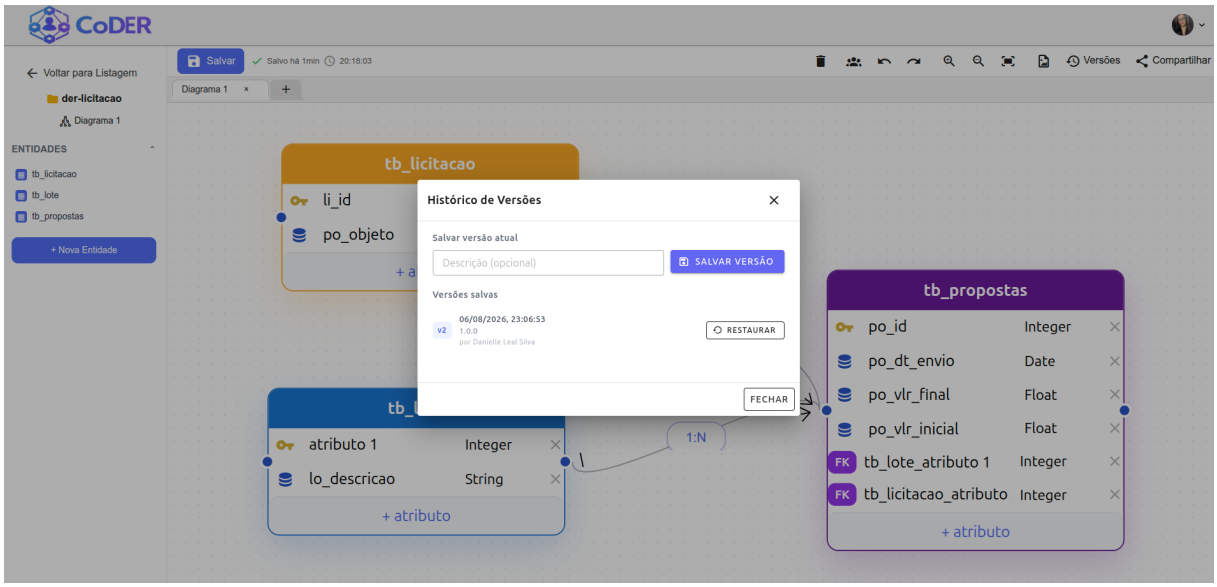


Figura 10 – Tela para salvamento de uma versão do diagrama

Fonte: Elaborada pela autora (2026).

Por fim, o projeto pode ser compartilhado com outros usuários, permitindo ao proprietário definir diferentes níveis de permissão de acesso. O CoDER disponibiliza os perfis *Editor* e *Visualizador*. O usuário com permissão de edição pode modificar o diagrama, enquanto o visualizador possui acesso ao projeto sem permissão para realizar alterações. A interface para realização do compartilhamento é apresentada na Figura 11.

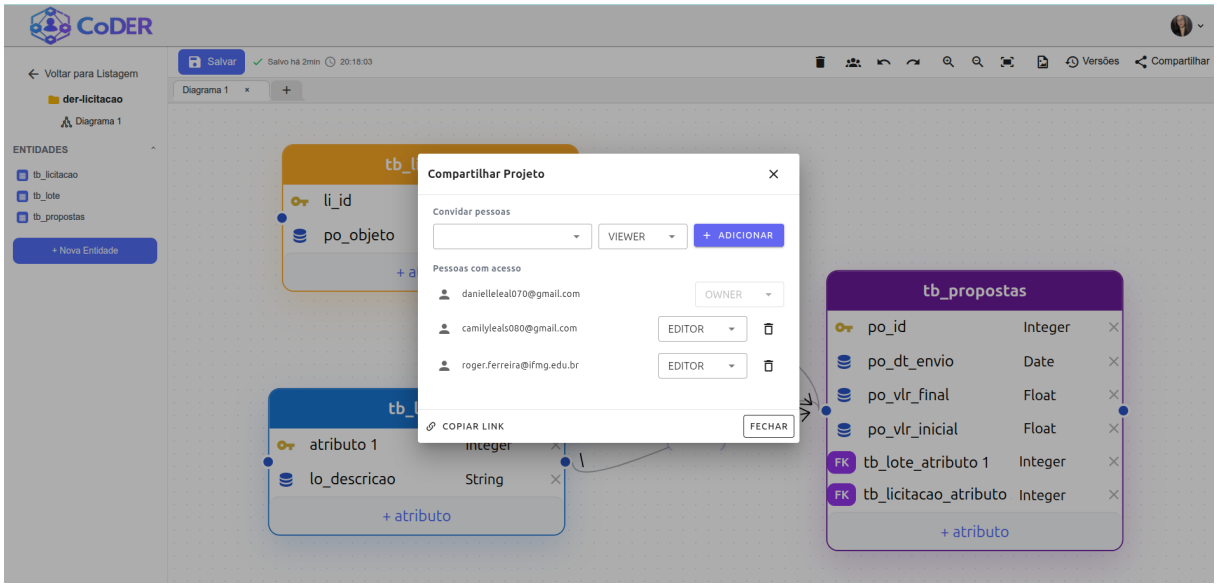


Figura 11 – Tela de compartilhamento de um projeto

Fonte: Elaborada pela autora (2026).

4.3 Arquitetura Implementada

A arquitetura proposta foi materializada no protótipo CoDER mantendo a separação entre apresentação, gerenciamento do estado, regras de negócio, persistência e comunicação em tempo real. A implementação foi organizada em duas partes principais: frontend e backend. O frontend concentra a interação com o usuário e o gerenciamento do estado local do diagrama, enquanto o backend concentra as regras de negócio, o acesso aos dados persistidos e a comunicação entre os clientes conectados. Essa organização mantém a decisão arquitetural de utilizar uma aplicação monolítica organizada em camadas.

O frontend foi desenvolvido utilizando Vue.js e Vue Flow. O Vue Flow é responsável pela representação e manipulação visual dos elementos do DER, enquanto o Pinia é utilizado para o gerenciamento do estado da aplicação. Também foram organizadas camadas específicas para autenticação, navegação e comunicação com os serviços do backend.

O backend foi desenvolvido em Java 17 utilizando Spring Boot. Sua organização separa controladores, serviços, modelos e repositórios. Os controladores disponibilizam os endpoints da API REST e os pontos de comunicação via WebSocket; os serviços concentram as regras de negócio; os repositórios são responsáveis pelo acesso aos dados persistidos; e os modelos representam as estruturas utilizadas pela aplicação.

A persistência foi implementada utilizando PostgreSQL, executado em um contêiner Docker, enquanto o Liquibase é utilizado para controlar as alterações do esquema do banco de dados. O acesso ao banco de dados é realizado exclusivamente pelo backend, mantendo o frontend desacoplado da camada de persistência.

A Figura 12 apresenta uma visão simplificada da organização dos principais componentes do código-fonte do CoDER.

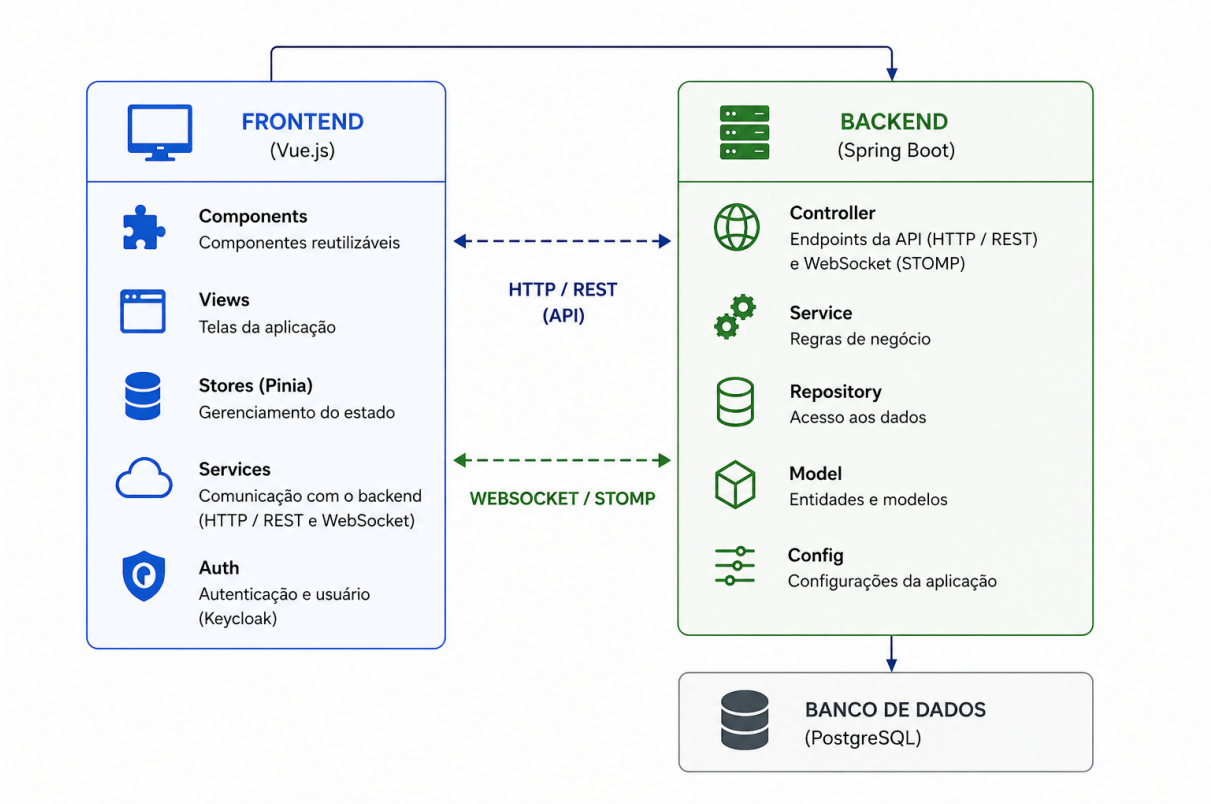


Figura 12 – Organização dos principais componentes do código-fonte do CoDER

Fonte: Elaborada pela autora (2026).

4.3.1 Comunicação entre cliente e servidor

A comunicação entre o cliente e o servidor ocorre por meio de mecanismos distintos, de acordo com a finalidade da operação. As funcionalidades convencionais da aplicação, como o gerenciamento de usuários e projetos e a persistência dos dados, utilizam uma API REST por meio do protocolo HTTP. Para a colaboração em tempo real, foi implementado um canal de comunicação bidirecional utilizando WebSocket com o protocolo STOMP, que define um formato de mensagens e regras para seu envio, recebimento e direcionamento entre os clientes e o servidor.

Essa separação permite utilizar o mecanismo mais adequado para cada finalidade: a API REST atende às operações convencionais de consulta e alteração dos recursos da aplicação, enquanto o WebSocket mantém um canal persistente para a troca contínua de eventos durante uma sessão colaborativa.

No frontend, a comunicação WebSocket foi encapsulada em um serviço específico, responsável por estabelecer a conexão, realizar as inscrições nos canais do projeto e enviar as operações produzidas pelo usuário. Cada projeto possui um canal destinado

às operações colaborativas e outro destinado às informações de presença dos usuários conectados. O envio de uma operação é realizado para o destino associado ao projeto, conforme apresentado no Código 4.1.

Código 4.1 – Envio de operações pelo serviço WebSocket

```
send(projectId, op) {
    if (!this.connected) {
        console.warn("WS ainda não conectado")
        return
    }

    stompClient.publish({
        destination: `/app/project/${projectId}/event`,
        body: JSON.stringify(op)
    })
}
```

No código apresentado, a operação é serializada para JSON e publicada pelo cliente no destino associado ao projeto. No backend, o prefixo `/app` identifica os destinos das mensagens direcionadas ao processamento da aplicação, enquanto o prefixo `/topic` é utilizado pelo broker simples do Spring para encaminhar mensagens aos clientes inscritos nos respectivos destinos. O endpoint `/ws`, por sua vez, é utilizado para estabelecer a conexão WebSocket utilizada pela comunicação STOMP.

No backend, o recebimento das operações é realizado por um controlador WebSocket. Após receber uma mensagem direcionada ao projeto, o servidor a encaminha aos clientes inscritos no respectivo canal, conforme apresentado no Código 4.2.

Código 4.2 – Recebimento e distribuição de operações pelo backend

```
@MessageMapping("/project/{projectId}/event")
    public void handleEvent(@DestinationVariable Long projectId, DiagramEvent event, Principal principal) {
        JwtAuthenticationToken auth = (JwtAuthenticationToken) principal;
        UserProfile user = userSyncService.syncUser(auth.getToken());
        event.setUserId(auth.getToken().getSubject());

        if (PRESENCE_EVENTS.contains(event.getType())) {
            projectAccessService.requireMember(projectId, user.getId());
        } else {
            projectAccessService.requireEditor(projectId, user.getId());
        }

        if (event.getType() == EnumEventType.CURSOR_MOVE) {
            messagingTemplate.convertAndSend("/topic/project/" + projectId + "/presence", event);
        } else {
            messagingTemplate.convertAndSend("/topic/project/" + projectId, event);
        }
    }
```

Dessa forma, o backend atua como intermediador da comunicação entre os clientes conectados ao mesmo projeto. A mensagem recebida é publicada no canal correspondente ao projeto, permitindo que os clientes inscritos recebam a operação.

A comunicação WebSocket também é integrada ao mecanismo de autenticação. O cliente utiliza o token de acesso fornecido pelo Keycloak e o envia durante o estabelecimento da conexão STOMP. No backend, o token é validado e associado à sessão WebSocket, permitindo identificar o usuário responsável pela conexão.

Além da autenticação, o backend realiza uma verificação de autorização durante a inscrição do cliente no canal de um projeto. Conforme apresentado no Código 4.3, o identificador do projeto é obtido a partir do destino da inscrição e, em seguida, é verificado se o usuário autenticado possui acesso ao projeto.

Código 4.3 – Verificação de autorização durante a inscrição em um canal WebSocket

```
if (StompCommand.SUBSCRIBE.equals(accessor.getCommand())) {

    JwtAuthenticationToken auth =
            (JwtAuthenticationToken) accessor.getUser();

    Jwt jwt = auth.getToken();
    String destination = accessor.getDestination();

    if (destination != null &&
            destination.startsWith("/topic/project/")) {

        Long projectId =
                Long.valueOf(destination.split("/")[3]);

        UserProfile user = userService.syncUser(jwt);

        projectAccessService.requireMember(
                projectId, user.getId());
    }
}
```

Assim, estabelecer uma conexão WebSocket não basta para acessar os dados de um projeto. O usuário precisa estar autenticado e possuir acesso ao projeto correspondente ao canal no qual pretende se inscrever.

O WebSocket também é utilizado para controlar a presença dos participantes de uma sessão colaborativa. Quando um usuário realiza uma inscrição no canal do projeto, sua sessão é registrada pelo serviço de presença. Quando a conexão é encerrada, a sessão é removida e a lista de participantes conectados é atualizada.

O registro da presença associa o projeto, a sessão WebSocket e as informações de

identificação do usuário, conforme apresentado no Código 4.4.

Código 4.4 – Registro de usuário conectado à sessão colaborativa

```
presenceService.addUser(
        projectId,
        sessionId,
        userId,
        name,
        email
);

broadcastUsers(projectId);
```

A utilização do WebSocket para o controle de presença permite que os participantes visualizem quais usuários estão conectados ao projeto sem a necessidade de consultas periódicas ao servidor. A Figura 13 apresenta usuários conectados simultaneamente ao mesmo projeto no CoDER.

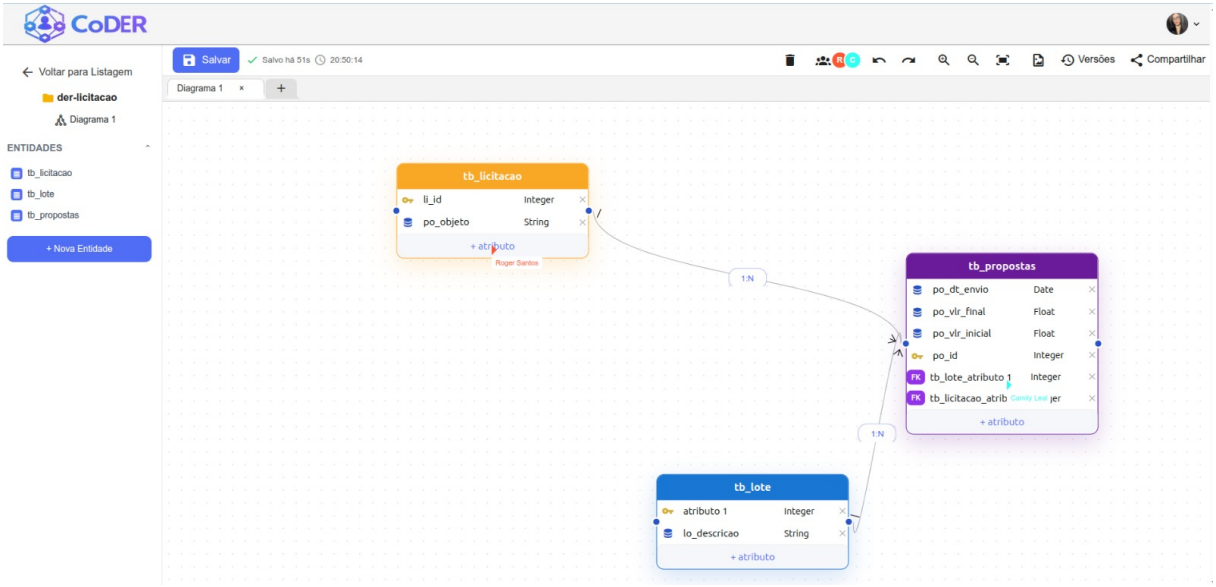


Figura 13 – Usuários conectados simultaneamente a um projeto no CoDER

Fonte: Elaborada pela autora (2026).

Na interface, a presença dos participantes é representada pelos indicadores associados aos usuários conectados, permitindo identificar visualmente que mais de um participante está acessando o mesmo projeto durante a sessão colaborativa. A interface também apresenta a identificação visual da posição dos participantes no editor, evidenciando a atuação simultânea sobre o mesmo espaço de trabalho.

4.3.2 Sincronização do estado compartilhado

A comunicação em tempo real é utilizada como mecanismo de transporte das operações, enquanto a consistência do estado compartilhado é responsabilidade da camada de sincronização baseada em CRDT. Dessa forma, a comunicação e a resolução de alterações concorrentes são tratadas como responsabilidades distintas: o WebSocket propaga as operações entre os clientes, enquanto o mecanismo CRDT determina como essas operações devem ser incorporadas às réplicas locais.

A implementação utiliza um CRDT baseado em operações. Cada alteração realizada no editor é representada por uma operação contendo, entre outras informações, um identificador único, o tipo da alteração, os dados modificados e metadados utilizados pelo mecanismo de sincronização. Após ser processada localmente, a operação é transmitida por meio do WebSocket aos demais clientes, que também a processam em suas respectivas réplicas.

Para determinar qual alteração deve prevalecer quando operações concorrentes modificam o mesmo estado, todas as operações de alteração são submetidas a um mecanismo LWW (*Last-Write-Wins*). Nesse modelo, cada valor possui metadados que permitem estabelecer uma ordem entre as alterações. Quando uma nova operação é recebida, seus metadados são comparados com os metadados associados ao valor atualmente armazenado. Caso a nova operação seja considerada posterior, seu valor substitui o anterior; caso contrário, a alteração é descartada.

No CoDER, a ordenação utilizada pelo LWW é baseada em relógios lógicos de Lamport (LAMPORT, 1978). Diferentemente de um relógio físico, o relógio lógico não representa uma hora do dia, mas uma posição relativa dos eventos observados por uma réplica. Cada operação recebe um valor de Lamport que permite estabelecer uma ordem lógica entre as alterações, sem depender do horário configurado nos computadores dos usuários.

Para facilitar a compreensão do processo de sincronização, a Figura 14 apresenta o fluxo de processamento das operações no CoDER, desde sua criação no cliente de origem até sua aplicação nas demais réplicas. O fluxo contempla a aplicação local da operação, a verificação de idempotência e a transmissão aos demais clientes, que processam a operação em suas respectivas réplicas.

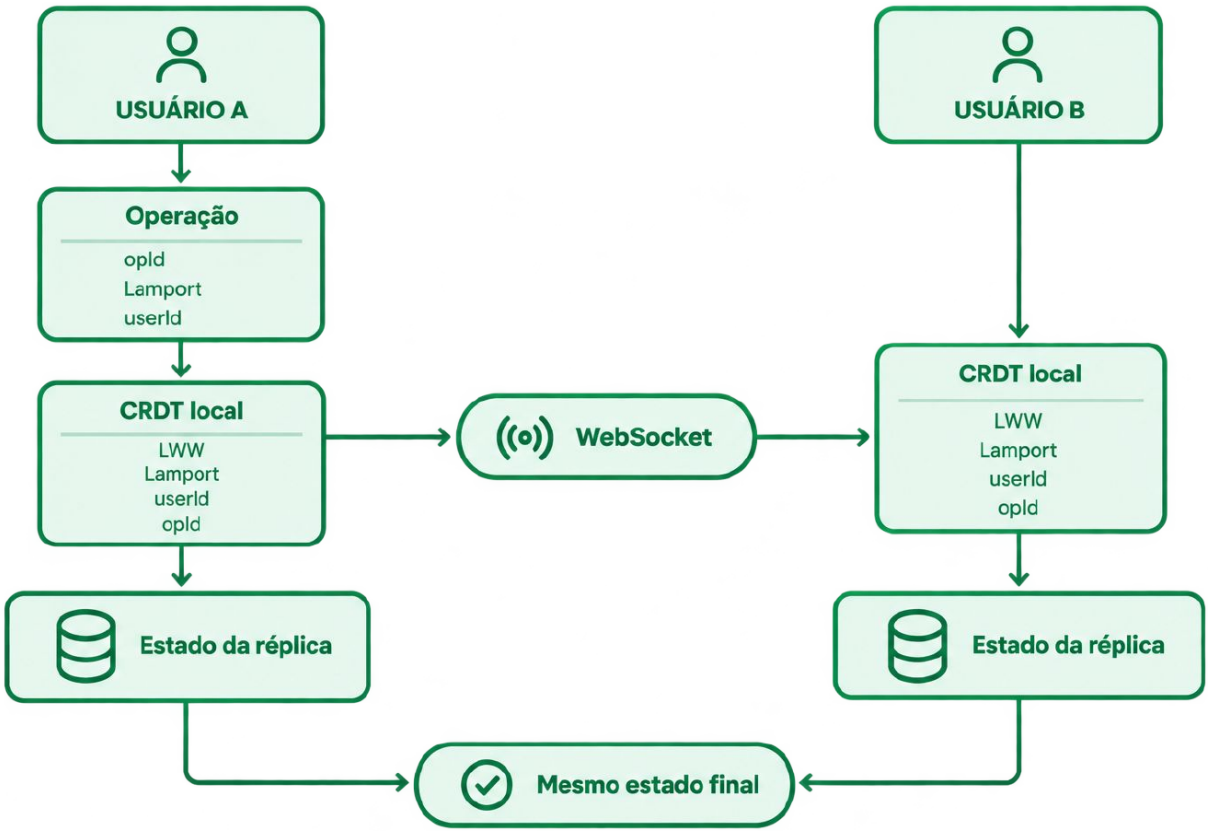


Figura 14 – Fluxo de sincronização das operações no CoDER

Fonte: Elaborada pela autora (2026).

Quando duas operações apresentam valores distintos de Lamport, aquela associada ao maior valor é considerada posterior. Quando os valores são iguais, o relógio lógico não é suficiente para estabelecer uma ordem entre as operações. Nesse caso, o identificador do usuário é utilizado como segundo critério de ordenação, sendo considerado vencedor o maior identificador. Como esse critério é igual em todas as réplicas, a decisão é determinística, independentemente da ordem em que as mensagens foram recebidas.

A decisão de aplicar uma alteração é tomada pelo método apresentado no Código 4.5.

Código 4.5 – Aplicação de alterações utilizando LWW Register

```
applyLWWRegister({ object, field, value, lamport, userId }) {
    if (!object._crdt) {
        object._crdt = {}
    }

    const current = object._crdt[field] || {
        lamport: 0,
        userId: ''
    }

    const incoming = {
        lamport,
```

```
        userId
    }

    const shouldApply =
        incoming.lamport > current.lamport ||
        (
            incoming.lamport === current.lamport &&
            incoming.userId > current.userId
        )

    if (!shouldApply) {
        return false
    }

    object[field] = value
    object._crdt[field] = incoming

    return true
}
```

O Código 4.5 evidencia que a decisão é realizada exclusivamente a partir dos metadados associados à operação. Assim, a ordem de chegada das mensagens não determina diretamente o estado final. Duas réplicas que processem o mesmo conjunto de operações, ainda que em ordens diferentes, utilizam os mesmos critérios de comparação e alcançam o mesmo valor prevalente.

Para complementar a compreensão desse mecanismo, a Figura 15 apresenta o fluxo de resolução de conflitos utilizado no CoDER. O processo contempla a verificação de idempotência, a comparação dos valores de Lamport e, quando necessário, a utilização do identificador do usuário como critério de desempate, determinando qual alteração será aplicada à réplica.

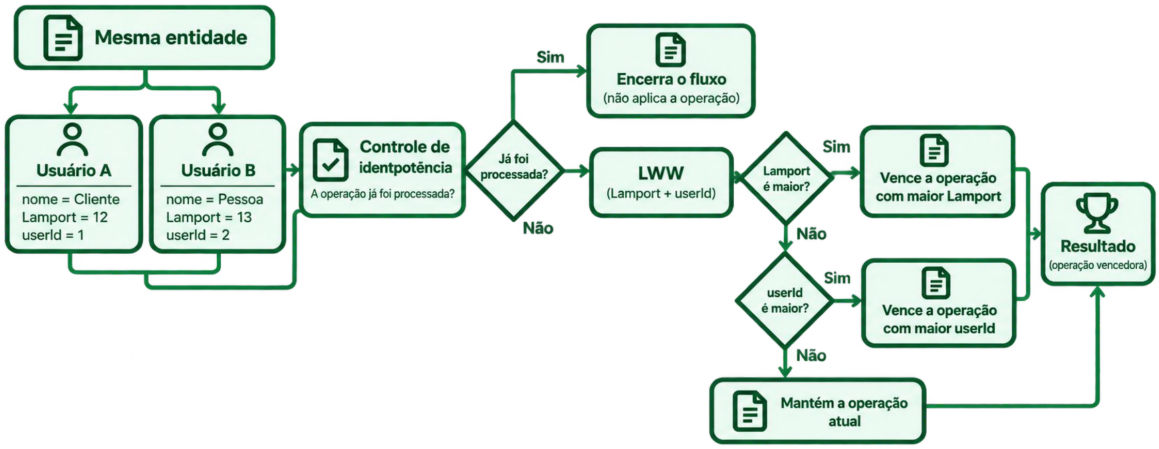


Figura 15 – Fluxo de resolução de conflitos

Fonte: Elaborada pela autora (2026).

Diante dessas características, o mecanismo implementado é definido neste trabalho como um CRDT baseado em operações, pois as alterações são representadas por operações processadas nas diferentes réplicas. A resolução determinística de conflitos por meio do LWW, combinada à ordenação baseada em relógios de Lamport e ao controle de operações duplicadas, permite que réplicas que processem o mesmo conjunto de operações alcancem o mesmo estado final. Essa propriedade é verificada experimentalmente por meio dos testes de convergência apresentados na Subseção 4.4.

4.3.3 Idempotência das operações

Além da ordenação determinística, o mecanismo de sincronização implementa idempotência para evitar que uma mesma operação seja aplicada mais de uma vez. Essa propriedade é necessária em um sistema distribuído, no qual uma operação pode ser recebida novamente por uma réplica devido ao fluxo de comunicação ou ao processamento de uma mesma mensagem em mais de um ponto.

Cada operação possui um identificador único (`opId`). Antes de executar uma operação, o cliente verifica se seu identificador já foi registrado como processado. Caso a operação já tenha sido aplicada, seu processamento é interrompido. Caso contrário, o identificador é registrado e a operação é executada normalmente.

O mecanismo é apresentado no Código 4.6.

Código 4.6 – Controle de idempotência das operações recebidas

```
applyOperation(op, saveHistory = true) {
    if (!op || !op.type || !op.opId) return

    if (this.appliedOps.has(op.opId)) {
        return
```

```
    }

    this.appliedOps.set(op.opId, Date.now())

    this.cleanupAppliedOpsIfNeeded()

    const opCopy = deepClone(op)

    this.execute(opCopy)

    if (saveHistory) {

        this.undoStack.push(deepClone(opCopy))

        if (this.undoStack.length > MAX_HISTORY) {
            this.undoStack.shift()
        }

        this.redoStack = []
    }
}
```

Dessa forma, o mecanismo de sincronização reúne dois recursos complementares: o identificador da operação impede seu processamento duplicado, enquanto o LWW determina qual alteração deve prevalecer quando operações diferentes disputam o mesmo estado. Essa combinação contribui para que as operações possam ser processadas localmente sem depender da ordem em que foram recebidas.

4.4 Resultados dos Testes

Os 20 cenários de teste definidos na Seção 3.4.2 foram implementados e executados utilizando o framework Vitest. Todos os 20 cenários apresentaram o comportamento esperado para as propriedades avaliadas. Nos cenários relacionados à convergência, as réplicas alcançaram estados equivalentes após a aplicação das operações. Também foram observados os comportamentos esperados nos cenários envolvendo operações concorrentes, aplicação de operações em diferentes ordens, aplicação duplicada de operações e operações de desfazer e refazer. A Figura 16 apresenta o resultado da execução dos testes, evidenciando a aprovação dos 20 cenários implementados.

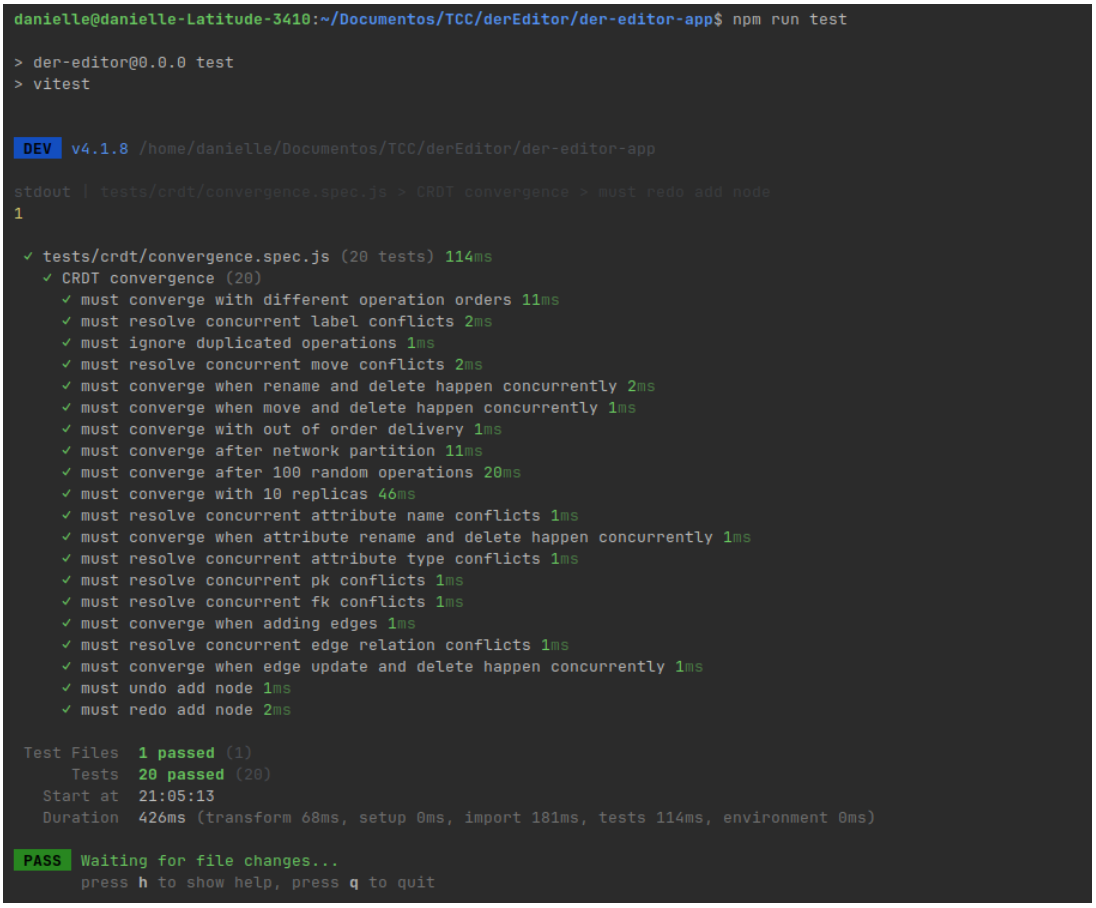


Figura 16 – Resultado da execução dos testes de convergência

Fonte: Elaborada pela autora (2026).

Os testes de concorrência e carga foram executados conforme o procedimento descrito na Seção 3.4.3. A Tabela 6 apresenta os resultados obtidos nos cinco cenários avaliados, considerando as médias e os desvios padrão calculados a partir das cinco execuções de cada cenário.

Tabela 6 – Resultados dos testes de concorrência e carga

Usu.	**Carga (op./s)**	**Lat. (ms)**	**P95 (ms)**	**P99 (ms)**	**Throughput (op./s)**
2	20,00	8,47 ± 1,86	17,80 ± 7,70	26,80 ± 2,99	19,93 ± 0,02
5	50,00	5,28 ± 1,25	9,80 ± 5,49	20,40 ± 12,34	49,80 ± 0,04
10	100,00	35,17 ± 44,27	86,80 ± 130,58	106,60 ± 146,14	98,82 ± 1,02
15	150,00	29,29 ± 11,37	50,60 ± 22,67	64,40 ± 31,10	148,07 ± 0,94
20	200,00	321,24 ± 291,94	467,20 ± 425,21	490,20 ± 434,59	175,56 ± 26,29

Fonte: Elaborada pela autora (2026).

Em todos os cenários avaliados, não foram observadas perdas de operações. Até

15 usuários, a latência média permaneceu abaixo de 36 ms e o throughput observado permaneceu próximo à carga oferecida, indicando que a comunicação acompanhou a taxa de geração das operações nesses cenários.

Observa-se, entretanto, que o cenário com 10 usuários apresentou latência média de $35{,}17 \pm 44{,}27$ ms, superior à registrada no cenário com 15 usuários, de $29{,}29 \pm 11{,}37$ ms. O desvio padrão elevado no cenário com 10 usuários indica uma variação expressiva entre as execuções, de modo que a latência não apresentou crescimento progressivo entre esses dois cenários.

No cenário com 20 usuários, observou-se um aumento expressivo na latência, que atingiu média de $321{,}24 \pm 291{,}94$ ms, enquanto o P95 e o P99 alcançaram, respectivamente, $467{,}20 \pm 425{,}21$ ms e $490{,}20 \pm 434{,}59$ ms. Nesse cenário, os valores também apresentaram maior variação, indicando menor estabilidade das medições sob condições de maior carga. O throughput médio foi de $175{,}56 \pm 26{,}29$ operações por segundo, abaixo da carga oferecida de 200 operações por segundo.

Apesar da redução do throughput no maior cenário, não foram registradas perdas de operações. Os resultados indicam, portanto, que o aumento da concorrência afetou principalmente o tempo de entrega e a capacidade de acompanhar a taxa de geração das operações, com maior impacto observado no cenário de 20 usuários.

Além dos testes automatizados, foram realizados testes manuais no protótipo implantado na plataforma Render. Foram verificadas as funcionalidades de cadastro e login de usuários, criação de projetos, compartilhamento entre usuários, criação de diagramas e controle das permissões de acesso para edição e visualização. Durante os testes de colaboração em tempo real, também foi verificada a sincronização das alterações realizadas no diagrama entre os usuários conectados. De modo geral, as funcionalidades verificadas apresentaram o comportamento esperado, incluindo o controle de acesso conforme as permissões atribuídas e a atualização colaborativa dos diagramas.

4.5 Discussão dos Resultados

Os resultados da avaliação devem ser interpretados considerando as características da arquitetura implementada e o comportamento observado nos experimentos realizados. Os testes de convergência demonstraram que o mecanismo de sincronização produziu estados equivalentes entre as réplicas nos cenários avaliados, incluindo diferentes ordens de processamento, operações concorrentes, operações duplicadas e múltiplas réplicas. Esses resultados são compatíveis com a utilização de um mecanismo CRDT orientado a operações, associado ao controle de idempotência e aos relógios lógicos utilizados na implementação.

A ausência de perdas de mensagens nos cenários de concorrência e carga indica que,

nas condições avaliadas, todas as operações esperadas foram entregues aos participantes conectados ao projeto. Esse resultado, contudo, não implica ausência de degradação no tempo de entrega, uma vez que foram observadas variações nas métricas de latência conforme a carga aplicada.

A taxa de operações foi definida previamente a partir do número de usuários e do intervalo de 100 ms entre operações. Dessa forma, o aumento da carga oferecida entre os cenários foi determinado pelo aumento do número de participantes. O throughput observado permaneceu próximo à carga oferecida nos cenários com até 15 usuários. No cenário com 20 usuários, entretanto, o throughput observado foi inferior à carga oferecida, acompanhado por um aumento expressivo da latência. Esses resultados representam o comportamento do sistema diante das cargas aplicadas, não devendo ser interpretados como uma medida da capacidade máxima do sistema.

As variações observadas nas métricas de latência também devem ser interpretadas com cautela, uma vez que o número de participantes e a carga oferecida foram modificados simultaneamente entre os cenários. Ainda assim, no cenário com 20 usuários, observou-se uma diferença expressiva em relação aos demais cenários, com latência média de 321,24 ms e desvio padrão de 291,94 ms, enquanto nos cenários com até 15 usuários a latência média permaneceu abaixo de 36 ms. Esse comportamento indica uma maior variabilidade no processamento e na entrega das operações sob a maior carga avaliada.

Por fim, a solução implementada apresenta relação com as evidências identificadas na RSL, especialmente quanto à adoção de arquitetura cliente-servidor, comunicação por WebSocket e utilização de mecanismos de sincronização baseados em CRDT. No contexto deste trabalho, essas referências fundamentaram a composição da arquitetura e a escolha do mecanismo de sincronização utilizado no protótipo.

5 Considerações Finais

Este trabalho teve como objetivo propor uma arquitetura para um editor web colaborativo de DER, fundamentada em uma RSL e demonstrada por meio do desenvolvimento de um protótipo funcional. O objetivo foi alcançado com a definição e implementação da arquitetura no protótipo CoDER, que possibilitou a edição colaborativa de diagramas em tempo real.

Os requisitos funcionais RF01 a RF05 foram implementados no protótipo, contemplando a edição colaborativa, comunicação em tempo real, gerenciamento de projetos e participantes, desfazer e refazer operações e controle de acesso. Os requisitos não funcionais RNF01 a RNF05 foram considerados na definição e implementação da arquitetura, com os aspectos de consistência e convergência avaliados por meio dos testes realizados. Os resultados obtidos fornecem evidências de que a arquitetura implementada atende ao funcionamento proposto para o protótipo, especialmente quanto à edição colaborativa, à comunicação em tempo real e à sincronização das operações.

Apesar dos resultados obtidos, é importante considerar as limitações do experimento. O número de usuários e a carga oferecida foram aumentados simultaneamente, de modo que não foi possível isolar o efeito do número de participantes sobre as métricas de desempenho. Além disso, os testes foram executados em uma única máquina, com os componentes do sistema e os clientes simulados no mesmo ambiente, não contemplando uma implantação distribuída em múltiplas máquinas, diferentes condições de rede ou múltiplas instâncias do servidor. A avaliação também foi limitada aos cenários e cargas definidos no experimento. Por fim, não foi realizada uma avaliação com usuários, o que impossibilitou analisar aspectos relacionados à usabilidade e à experiência de uso do protótipo.

5.1 Principais Contribuições

Como principais contribuições, destacam-se a sistematização de evidências sobre arquiteturas e tecnologias utilizadas em sistemas web colaborativos por meio da RSL; a proposição de uma arquitetura aplicada ao domínio de modelagem de DER; e o desenvolvimento do protótipo CoDER, que integra autenticação, gerenciamento de projetos, controle de permissões, comunicação em tempo real e sincronização de operações.

Também constitui contribuição do trabalho a implementação e avaliação de um mecanismo de sincronização baseado em CRDT no contexto específico de um editor colaborativo de DER, considerando os cenários de concorrência definidos na avaliação.

5.2 Trabalhos Futuros

Como trabalhos futuros, sugere-se ampliar a avaliação do mecanismo de sincronização e investigar a evolução da arquitetura para cenários de maior escala, incluindo múltiplas instâncias do servidor, uso de broker externo, maior número de usuários e diferentes condições de rede.

No âmbito da consistência, podem ser investigados mecanismos de entrega causal, diferentes semânticas para operações concorrentes e estratégias para sincronização de clientes que ingressam após o início de uma sessão. Também pode ser explorado o suporte à edição offline, com sincronização das alterações após a reconexão.

Quanto à avaliação, podem ser realizados estudos com usuários, especialmente estudantes de disciplinas relacionadas a Banco de Dados, utilizando métricas de usabilidade. Também pode ser realizada uma comparação experimental com bibliotecas de sincronização, como Yjs e Automerge, considerando critérios equivalentes.

Por fim, podem ser incorporadas novas funcionalidades ao protótipo, como a extensão dos mecanismos de desfazer e refazer para o contexto colaborativo, de modo que as operações inversas e reaplicadas por um usuário também sejam sincronizadas com os demais participantes da sessão, e a geração de código DDL SQL a partir dos modelos elaborados.

Referências

AURIA, B.; BARZOLA, V.; ECHEVERRIA, V. A collaborative design tool to support learning environments during emergent remote learning. In: GIANNATTASIO, G. (Ed.). *2021 IEEE URUCON*. Piscataway: IEEE, 2021. p. 541–546. Acesso em: 16 set. 2025. Disponível em: <https://doi.org/10.1109/URUCON53396.2021.9647346>. Citado 2 vezes nas páginas 18 e 21.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 4. ed. Boston: Addison-Wesley, 2022. ISBN 9780136886099. Citado na página 22.

BHARADWAJ, K. et al. Securing collaborative work in wide-band display environments. In: *2021 IEEE 7th International Conference on Collaboration and Internet Computing (CIC)*. Piscataway, NJ, USA: IEEE, 2021. p. 26–34. Acesso em: 21 de setembro de 2026. Disponível em: <https://doi.org/10.1109/CIC52973.2021.00014>. Citado na página 19.

BIOLCHINI, J. et al. *Systematic Review in Software Engineering*. Rio de Janeiro, 2005. 31 p. Acesso em: 22 set. 2026. Disponível em: <https://www.cos.ufrj.br/uploadfile/es67905.pdf>. Citado na página 28.

CHEN, P. P.-S. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*, v. 1, n. 1, p. 9–36, 1976. Citado na página 23.

DAY-RICHTER, J. *What's different about the new Google Docs: Making collaboration fast*. 2010. Acesso em: 22 set. 2025. Disponível em: <https://cloud.googleblog.com/2010/09/whats-different-about-new-google-docs.html>. Citado na página 21.

dbdiagram.io. *Real-time Collaboration*. s.d. Acesso em: 23 out. 2026. Disponível em: <https://docs.dbdiagram.io/collaborations/>. Citado na página 16.

DRAGONI, N. et al. Microservices: Yesterday, today, and tomorrow. In: MAZZARA, M.; MEYER, B. (Ed.). *Present and Ulterior Software Engineering*. Cham: Springer, 2017. p. 195–216. Citado na página 31.

drawDB. *Online Database Schema Design and ER Diagram Tool*. s.d. Acesso em: 23 out. 2026. Disponível em: <https://www.drawdb.app/>. Citado na página 16.

ELLIS, C. A.; GIBBS, S. J. Concurrency control in groupware systems. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ACM, p. 399–407, 1989. Citado na página 26.

ELMASRI, R.; NAVATHE, S. B. *Sistemas de Banco de Dados*. 6. ed. São Paulo: Pearson Addison Wesley, 2011. Citado 2 vezes nas páginas 23 e 24.

ERDHub. *Real-Time Editing*. s.d. Acesso em: 23 out. 2026. Disponível em: <https://docs.erdhub.com/collaboration/realtime>. Citado na página 16.

GADEA, C.; IONESCU, B.; IONESCU, D. New algorithms and methods for collaborative co-editing using HTML DOM synchronization. In: *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*. Philadelphia, PA,

USA: IEEE, 2018. p. 217–226. Acesso em: 21 de setembro de 2026. Disponível em: <https://doi.org/10.1109/CIC.2018.00038>. Citado na página 19.

GARCÍA, J. O. O uso de ferramentas colaborativas no processo de ensino-aprendizagem. *Revista Educação Contemporânea*, v. 1, n. 2, p. 635–644, 2024. ISSN 2966-4705. Acesso em: 16 set. 2026. Disponível em: <https://www.editoraverde.org/portal/revistas/index.php/reca/article/view/643/803>. Citado na página 15.

GOMES, R. L.; WILLRICH, R.; RIVERA, G. de J. H. Arquiteturas distribuídas para sistemas colaborativos. In: PIMENTEL, M.; FUKS, H. (Ed.). *Sistemas Colaborativos*. Rio de Janeiro: Elsevier, 2012. cap. 20, p. 327–347. Citado 2 vezes nas páginas 22 e 23.

HEMID, A. et al. Ontoeditor: Real-time collaboration via distributed version control for ontology development. In: PEÑUELA, A. M. et al. (Ed.). *The Semantic Web*. Cham: Springer Nature Switzerland, 2024. p. 326–341. Acesso em: 16 set. 2025. Disponível em: <https://doi.org/10.1007/978-3-031-60626-7_18>. Citado na página 19.

JATANA, N. et al. Differentially processed optimized collaborative rich text editor. *Multimedia Tools and Applications*, v. 84, n. 17, p. 17189–17204, 2025. Acesso em: 16 set. 2025. Disponível em: <https://doi.org/10.1007/s11042-024-19734-3>. Citado na página 18.

KURNIAWAN, A.; SOESANTO, C.; WIJAYA, J. E. C. Coder: Real-time code editor application for collaborative programming. *Procedia Computer Science*, v. 59, p. 510–519, 2015. Acesso em: 16 set. 2025. Disponível em: <https://doi.org/10.1016/j.procs.2015.07.531>. Citado 2 vezes nas páginas 18 e 19.

LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, Association for Computing Machinery, v. 21, n. 7, p. 558–565, 1978. Citado na página 52.

LANGE, P. de et al. Integrating web-based collaborative live editing and wireframing into a model-driven web engineering process. *Data Science and Engineering*, v. 5, n. 3, p. 240–260, 2020. Acesso em: 23 set. 2026. Disponível em: <https://doi.org/10.1007/s41019-020-00131-3>. Citado 2 vezes nas páginas 18 e 22.

PIMENTEL, M.; FUKS, H. *Sistemas colaborativos*. Rio de Janeiro: Elsevier, 2011. ISBN 978-85-352-4669-8. Citado 2 vezes nas páginas 15 e 20.

SHAPIRO, M. et al. Conflict-free replicated data types. In: *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems (SSS 2011)*. Berlin, Heidelberg: Springer, 2011. (Lecture Notes in Computer Science, v. 6976), p. 386–400. Acesso em: 21 de setembro de 2026. Disponível em: <https://doi.org/10.1007/978-3-642-24550-3_29>. Citado 2 vezes nas páginas 26 e 27.

SINGH, K.; YOAKUM, J. Vclick: Endpoint driven enterprise webrtc. In: *2015 IEEE International Symposium on Multimedia (ISM)*. Miami, FL, USA: IEEE Computer Society, 2015. p. 441–444. Acesso em: 21 de setembro de 2026. Disponível em: <https://doi.org/10.1109/ISM.2015.92>. Citado 3 vezes nas páginas 19, 20 e 23.

STANCU, F.-A.; CHIROIU, M.; RUGHINIS, R. Seccollab - improving confidentiality for existing cloud-based collaborative editors. In: *2017 21st International Conference on Control Systems and Computer Science (CSCS)*. Bucharest, Romania: IEEE,

2017. p. 324–331. Acesso em: 21 de setembro de 2026. Disponível em: <https://doi.org/10.1109/CSCS.2017.51>. Citado na página 19.

SUN, D. et al. Real differences between ot and crdt in correctness and complexity for consistency maintenance in co-editors. *Proceedings of the ACM on Human-Computer Interaction*, ACM, v. 4, n. CSCW1, p. 1–30, 2020. Acesso em: 21 de setembro de 2026. Disponível em: <https://doi.org/10.1145/3392825>. Citado na página 31.

WANG, V.; SALIM, F.; MOSKOVITS, P. *The Definitive Guide to HTML5 WebSocket*. [S.l.]: Apress, 2013. ISBN 9781430247401. Citado na página 26.

WARNER, J.; GUO, P. J. Codepilot: Scaffolding end-to-end collaborative software development for novice programmers. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2017. (CHI '17), p. 1136–1141. ISBN 9781450346559. Acesso em: 16 set. 2026. Disponível em: <https://doi.org/10.1145/3025453.3025876>. Citado na página 18.

WEIDNER, M. et al. *Collabs: A Flexible and Performant CRDT Collaboration Framework*. 2023. Acesso em: 16 set. 2025. Disponível em: <https://arxiv.org/abs/2212.02618>. Citado na página 27.

ZIMMER, B.; KERREN, A. Harnessing webgl and websockets for a web-based collaborative graph exploration tool. In: FRASINCAR, F. et al. (Ed.). *Engineering the Web in the Big Data Era*. Cham: Springer, 2015. (Lecture Notes in Computer Science, v. 9114), p. 583–598. Acesso em: 16 set. 2026. Disponível em: <https://doi.org/10.1007/978-3-319-19890-3_37>. Citado na página 18.

ZWEIHOFF, P.; NAUJOKAT, S.; STEFFEN, B. Pyro: Generating domain-specific collaborative online modeling environments. In: HÄHNLE, R.; AALST, W. van der (Ed.). *Fundamental Approaches to Software Engineering*. Cham: Springer International Publishing, 2019. p. 101–115. Citado na página 19.