Universidade de São Paulo
Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto (FFCLRP)
Departamento de Computação e Matemática (DCM)

Cartas de Amor: um Jogo Distribuído

Lucas Cardoso dos Santos (9865492)
Maya Monteiro Lima (13671942)
Miguel de Carvalho Oliveira (13672172)

Ribeirão Preto

2025

Sumário

1 Introdução

O desenvolvimento de jogos digitais envolve desafios que vão desde a modelagem de regras e dinâmicas até a implementação de sistemas robustos e seguros. Este relatório apresenta a criação de uma aplicação web multiplayer inspirada no jogo de cartas Love Letter, abordando desde os conceitos fundamentais do jogo, suas regras e mecânicas, até a materialização dessas ideias em uma solução tecnológica moderna.

A solução proposta explora aspectos de autenticação de usuários, gerenciamento de salas de jogo, comunicação em tempo real e sincronização de estados entre múltiplos participantes. Para garantir uma experiência fluida e segura, foram adotadas práticas de arquitetura em camadas, uso de protocolos adequados para cada tipo de interação e a definição de requisitos funcionais claros, que orientaram o desenvolvimento desde o início.

Além da implementação das funcionalidades essenciais, o projeto contempla o desenho de uma base de dados eficiente, capaz de armazenar informações relevantes sobre usuários, partidas e interações. O resultado é uma plataforma que alia entretenimento, tecnologia e boas práticas de engenharia de software, evidenciando o potencial de aplicações web modernas para proporcionar experiências interativas e seguras aos seus usuários.

2 Requisitos

1 Requisitos Funcionais

- **RF01**: O sistema deve permitir o registro de novos usuários com nome de usuário, e-mail e senha.
- **RF02**: O sistema deve permitir autenticação de usuários e manutenção de sessões seguras.
- **RF03**: Usuários autenticados devem poder criar novas salas de jogo.
- **RF04**: O sistema deve listar salas de jogo disponíveis para entrada.
- **RF05**: Cada sala de jogo deve suportar de 2 a 6 jogadores simultâneos.
- **RF06**: O estado do jogo deve ser sincronizado em tempo real entre todos os jogadores, utilizando WebSockets.
- **RF07**: O sistema deve implementar e aplicar as regras oficiais do jogo *Love Letter*, conforme especificado no Apêndice A.
- **RF08**: O sistema deve permitir que o usuário exclua sua própria conta.
- **RF09**: O sistema deve permitir que o usuário altere seu nome de usuário.
- **RF10**: O usuário deve poder sair de uma sala de jogo a qualquer momento.
- **RF11**: O usuário não pode participar de múltiplas salas simultaneamente.
- **RF12**: O sistema deve notificar os jogadores quando alguém sair da sala.
- **RF13**: O usuário deve poder acessar as regras do jogo a qualquer momento durante a partida.
- **RF14**: O sistema deve exibir logs de ações do jogo de forma compreensível.

2 Requisitos Não-Funcionais

2.1 Desempenho

- **RNF01**: A latência da comunicação em tempo real deve ser inferior a 1 segundo durante as partidas.

- **RNF02**: O sistema deve suportar pelo menos 10 usuários simultâneos sem degradação perceptível de desempenho.
- **RNF03**: O usuário deve poder desabilitar animações para melhorar a performance em dispositivos menos potentes.

2.2 Segurança

- **RNF04**: Senhas devem ser armazenadas utilizando algoritmos de hash seguros (ex: bcrypt, Argon2).
- **RNF05**: Toda comunicação entre cliente e servidor deve ser criptografada (HTTPS/WSS).
- **RNF06**: Sessões de usuários devem expirar após um período e utilizar tokens de autenticação seguros.

2.3 Usabilidade e Acessibilidade

- **RNF07**: O sistema deve ser acessível, suportando navegação por teclado e leitores de tela.
- **RNF08**: O sistema deve estar disponível em, pelo menos, português e inglês.

2.4 Portabilidade e Manutenção

- **RNF09**: O sistema deve funcionar nos principais navegadores modernos (Chrome, Firefox, Edge, Safari) e sistemas operacionais (Windows, Linux, macOS).
- **RNF10**: O código deve seguir princípios de design modular, com documentação e comentários adequados.
- **RNF11**: O sistema deve possuir mecanismos de logging e monitoramento para facilitar manutenção e auditoria.

2.5 Confiabilidade

- **RNF12**: A sincronização do estado do jogo deve ser testada quanto à consistência e tolerância a falhas de conexão.
- **RNF13**: O sistema deve permitir recuperação de sessões em caso de desconexão temporária.

3 Arquitetura do Sistema

O sistema desenvolvido adota uma arquitetura moderna e robusta, composta por dois principais componentes: um cliente web desenvolvido com Svelte e um servidor backend implementado em .NET. A comunicação entre cliente e servidor é realizada predominantemente via protocolo HTTP para operações tradicionais de gerenciamento (como criação e entrada em salas de jogo, gerenciamento de contas de usuário e obtenção de dados do jogo). Para operações em tempo real, essenciais para a dinâmica do jogo (como jogar cartas, comprar cartas e receber atualizações instantâneas), utiliza-se o protocolo WebSocket, garantindo baixa latência e interatividade.

1 Arquitetura em Camadas do Backend

O backend segue o padrão de arquitetura em camadas, promovendo separação de responsabilidades, facilidade de manutenção e escalabilidade. As principais camadas são:

- **Apresentação**: Responsável por expor as APIs HTTP e WebSocket, recebendo e respondendo às requisições dos clientes. Realiza validações iniciais e orquestra o fluxo das operações.
- **Aplicação**: Contém a lógica de orquestração dos casos de uso do sistema, coordenando as operações entre as demais camadas. Implementa os fluxos de negócio sem se preocupar com detalhes de infraestrutura ou regras de domínio.
- **Domínio**: Abriga as regras de negócio centrais e entidades do sistema. É independente de detalhes técnicos e representa o núcleo lógico da aplicação.
- **Infraestrutura**: Implementa detalhes técnicos como persistência de dados, integração com serviços externos e mecanismos de comunicação. Fornece implementações concretas para interfaces definidas nas camadas superiores.

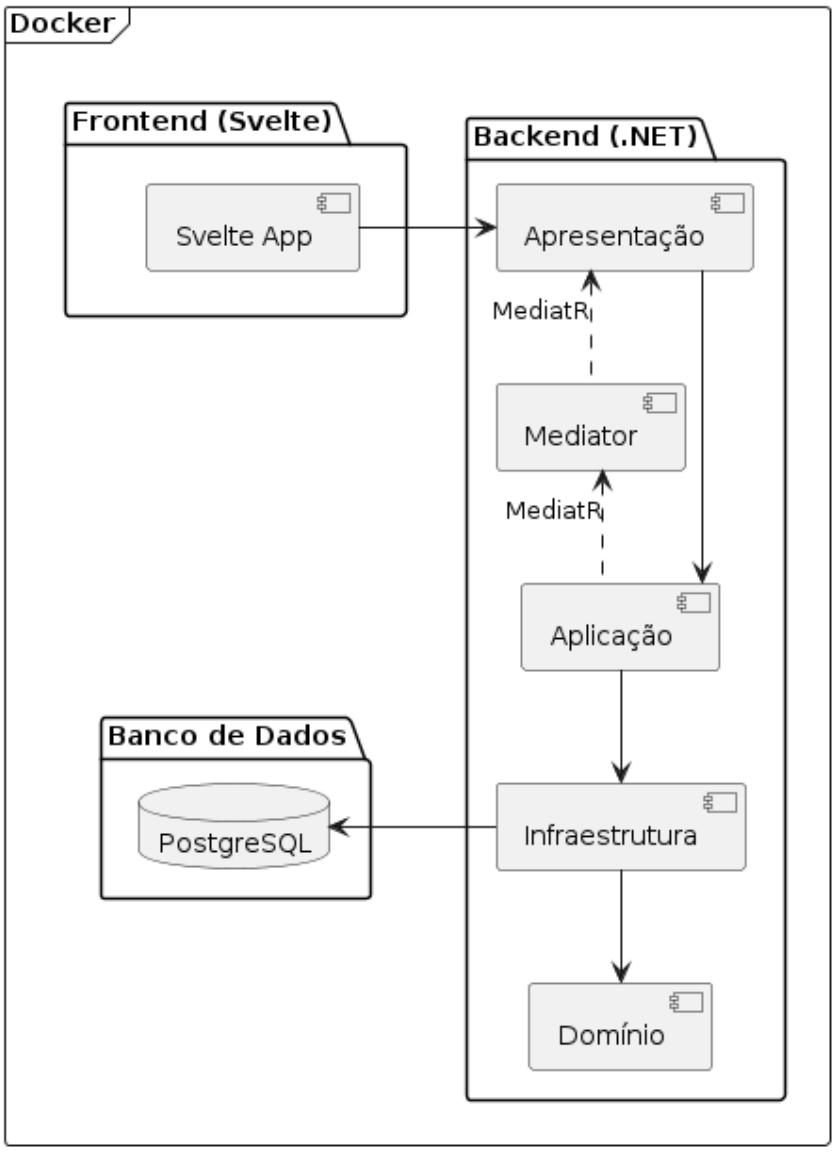

Figura 1 – Diagrama de arquitetura do sistema. Fonte: os autores

2 Padrões de Projeto Utilizados

O sistema faz uso de alguns padrões de projeto consagrados para promover flexibilidade e desacoplamento:

1. **Repository**: Abstrai o acesso a dados, permitindo que a lógica de negócio interaja com repositórios de forma independente da tecnologia de persistência.
2. **Mediator**: Centraliza a comunicação entre componentes, facilitando a implementação de casos de uso e reduzindo o acoplamento entre objetos.
3. **Factory**: Facilita a criação de objetos complexos, encapsulando a lógica de instanciação e promovendo flexibilidade na escolha de implementações.

Essa arquitetura garante que o sistema seja modular, testável e preparado para evoluções futuras.

3 Arquitetura do Frontend

O frontend do sistema foi desenvolvido utilizando o framework Svelte, que permite a criação de interfaces web reativas e eficientes. A estrutura do projeto segue boas práticas de modularização e separação de responsabilidades, facilitando a manutenção e a escalabilidade da aplicação.

O código-fonte do frontend está organizado em componentes Svelte, cada um responsável por uma parte específica da interface do usuário, como telas de login, gerenciamento de salas, lobby, mesa de jogo e exibição de cartas. Essa abordagem baseada em componentes permite o reuso de código e a fácil evolução da interface.

Além disso, o frontend faz uso de stores reativos do Svelte para gerenciar o estado global da aplicação, como informações do usuário autenticado, estado da sala e dados da partida em andamento. Isso garante que as atualizações de estado sejam refletidas automaticamente na interface, proporcionando uma experiência fluida ao usuário.

A arquitetura adotada no frontend, aliada ao uso de Svelte, resulta em uma aplicação leve, responsiva e de fácil manutenção, preparada para futuras expansões e melhorias.

4 Tecnologias Utilizadas

Durante o desenvolvimento do sistema, foram empregadas diversas tecnologias que contribuíram para a robustez, escalabilidade e facilidade de manutenção da solução:

- **SignalR**: Utilizado para facilitar a comunicação em tempo real via WebSocket entre servidor e clientes. O SignalR abstrai a complexidade do gerenciamento de conexões, transmissão de mensagens e reconexão automática, permitindo implementar funcionalidades interativas e síncronas de forma simples e eficiente.
- **MediatR**: Biblioteca empregada para implementar o padrão Mediator no backend, promovendo o desacoplamento entre as camadas de Apresentação e Aplicação. O MediatR centraliza o envio e o tratamento de comandos, queries e eventos, facilitando a manutenção e a evolução do código.
- **PostgreSQL e Entity Framework**: O sistema utiliza o banco de dados relacional PostgreSQL para persistência dos dados, integrado ao backend por meio do Entity Framework, que provê uma camada de abstração para o acesso e manipulação dos dados de forma orientada a objetos.

- **Docker**: Utilizado para empacotar e distribuir tanto o backend quanto o frontend em containers, garantindo portabilidade, reprodutibilidade e facilidade de deploy em diferentes ambientes.

Essas tecnologias, integradas à arquitetura do sistema, proporcionam uma base sólida para o desenvolvimento, operação e evolução contínua da aplicação.

4 Funcionamento do Sistema

O sistema Love Letter é uma aplicação web multiplayer que permite aos usuários jogar o jogo de cartas Love Letter em tempo real através de seus navegadores. O funcionamento do sistema pode ser compreendido através de diferentes perspectivas: o fluxo de interação do usuário, a arquitetura técnica, e os mecanismos de sincronização em tempo real.

1 Fluxo de Interação do Usuário

1.1 Autenticação e Gerenciamento de Conta

O usuário inicia sua experiência criando uma conta ou fazendo login no sistema. O processo de autenticação utiliza tokens JWT para manter a sessão do usuário de forma segura. Após a autenticação, o usuário tem acesso ao dashboard principal, onde pode gerenciar seu perfil, visualizar salas disponíveis ou criar novas salas de jogo.

1.2 Criação e Entrada em Salas

O usuário pode criar uma nova sala de jogo, definindo um nome e opcionalmente uma senha para acesso restrito. Alternativamente, pode visualizar todas as salas disponíveis e entrar em uma delas. O sistema suporta até seis jogadores por sala, e cada sala possui um host (criador) que tem permissões especiais, como a capacidade de deletar a sala.

1.3 Início da Partida

Uma vez na sala, os jogadores aguardam o início da partida. O host pode iniciar o jogo quando houver pelo menos dois jogadores presentes. O sistema automaticamente distribui as cartas iniciais e determina a ordem de jogo, iniciando com o primeiro jogador, conforme ilustrado na Figura 2.

A Figura 2 ilustra o processo de início do jogo. O host solicita o início da partida através da interface, o que dispara uma mensagem SignalR para o servidor. O sistema então valida se há jogadores suficientes (mínimo de 2) e se o solicitante é realmente o host da sala. Se todas as validações passarem, o sistema configura o jogo embaralhando o baralho, distribuindo as cartas iniciais e definindo o primeiro jogador. O estado do jogo é atualizado no banco de dados e todos os jogadores são notificados do início da rodada.

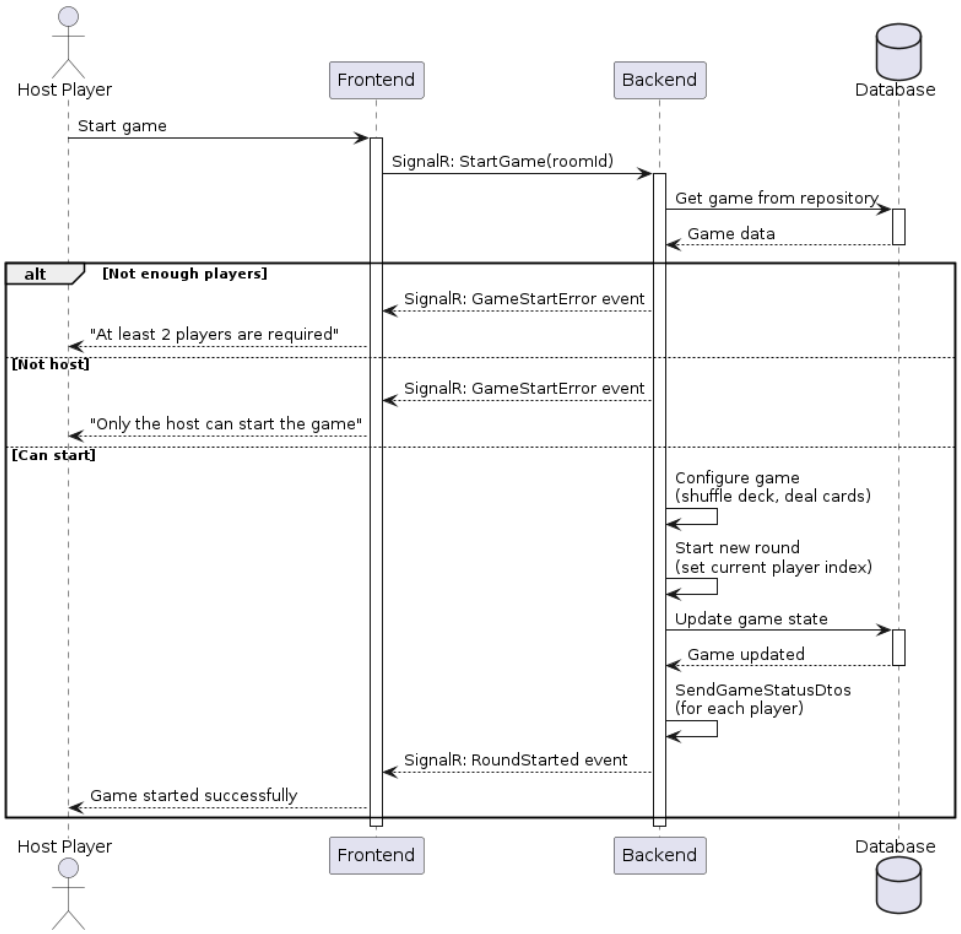

Figura 2 – Fluxo de início do jogo. Fonte: os autores

2 Mecanismos de Jogo

2.1 Ciclo de Turno

O jogo segue um ciclo de turnos bem definido:

1. **Compra de Carta**: O jogador da vez compra uma carta do baralho, adicionando-a à sua mão
2. **Jogada de Carta**: O jogador escolhe uma das duas cartas em sua mão para jogar, executando seu efeito específico
3. **Resolução de Efeitos**: O sistema processa o efeito da carta jogada, que pode incluir eliminação de jogadores, proteção, troca de cartas, entre outros
4. **Avanço de Turno**: O turno passa para o próximo jogador ativo

O diagrama da Figura 3 ilustra o fluxo geral dessas etapas durante um turno.

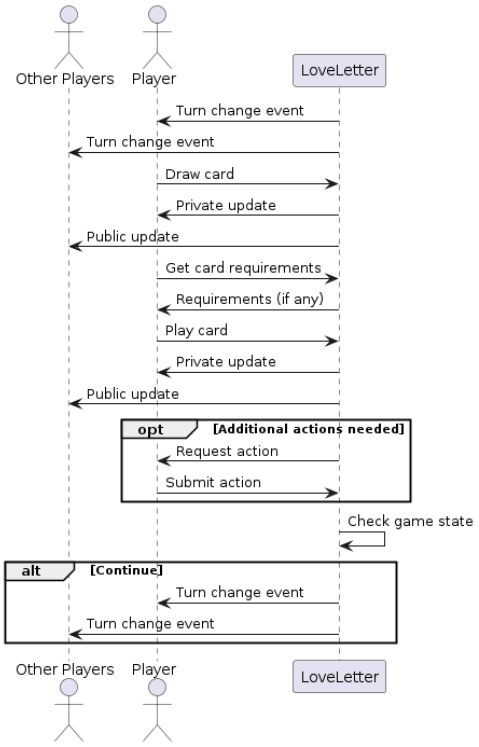

Figura 3 – Fluxo geral de um turno no jogo. Fonte: os autores

A Figura 4 ilustra o processo de verificação de finalização tanto de rodadas quanto do jogo completo. Após cada jogada, o sistema verifica se a rodada terminou (por eliminação de jogadores ou esgotamento do baralho). Se a rodada terminou, o sistema determina os vencedores e concede pontos bônus quando aplicável. Em seguida, verifica se o jogo completo terminou (quando um jogador atinge a pontuação necessária). Se o jogo não terminou, uma nova rodada é iniciada com a distribuição de novas cartas. Caso contrário, o turno simplesmente avança para o próximo jogador ativo.

2.2 Sistema de Cartas

O jogo utiliza 10 tipos diferentes de cartas, cada uma com valores e efeitos únicos, conforme ilustrado nas Figuras 5 a 14:

- **Espião (0)**: Concede ponto bônus se mantido até o final da rodada
- **Guarda (1)**: Permite adivinhar a carta de outro jogador
- **Padre (2)**: Permite olhar a carta de outro jogador
- **Barão (3)**: Compara cartas com outro jogador
- **Criada (4)**: Protege o jogador até seu próximo turno
- **Príncipe (5)**: Força outro jogador a descartar e comprar nova carta

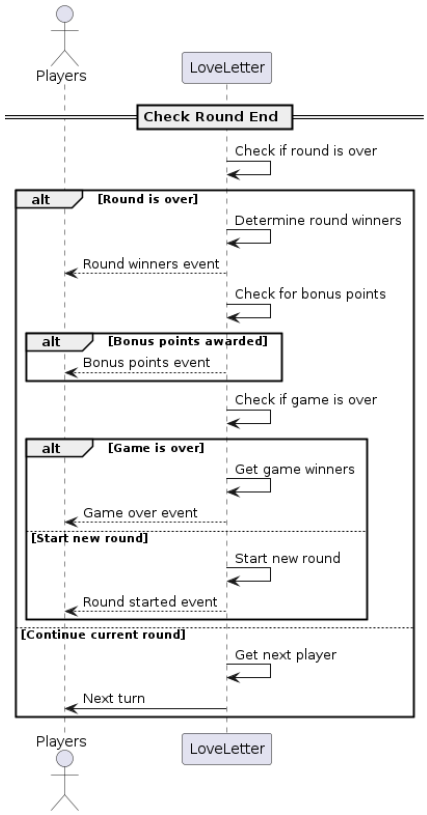

Figura 4 – Fluxo de avanço de turno e verificação de finalização. Fonte: os autores

- **Chanceler (6)**: Permite comprar duas cartas e escolher qual manter
- **Rei (7)**: Troca cartas com outro jogador
- **Condessa (8)**: Deve ser jogada quando o jogador possui Rei ou Príncipe
- **Princesa (9)**: Elimina o jogador se descartada

2.3 Estados do Jogo

O sistema gerencia diferentes estados do jogo para garantir que as ações sejam executadas na ordem correta:

- **WaitingForPlayers**: Aguardando jogadores entrarem na sala
- **WaitingForDraw**: Aguardando o jogador da vez comprar uma carta
- **WaitingForPlay**: Aguardando o jogador escolher e jogar uma carta
- **Finished**: Jogo finalizado

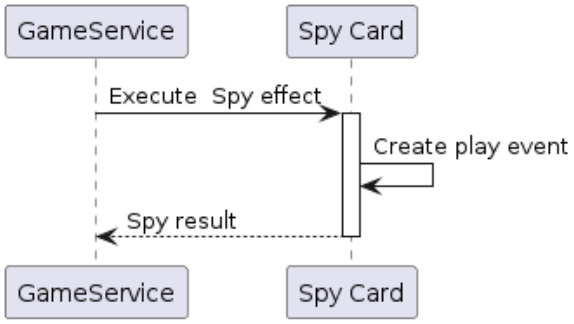

Figura 5 – Jogada da carta Espião. Fonte: os autores

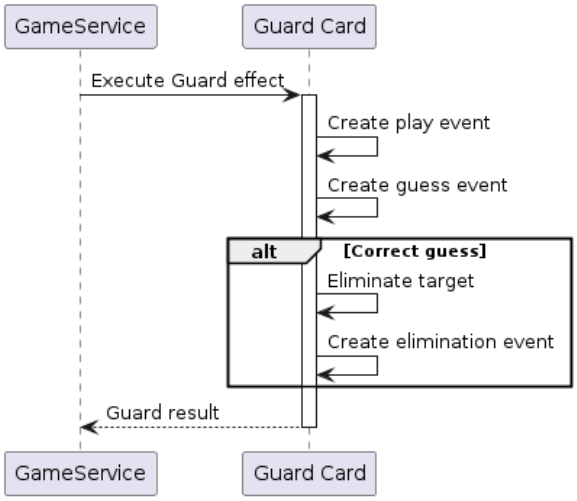

Figura 6 – Jogada da carta Guarda. Fonte: os autores

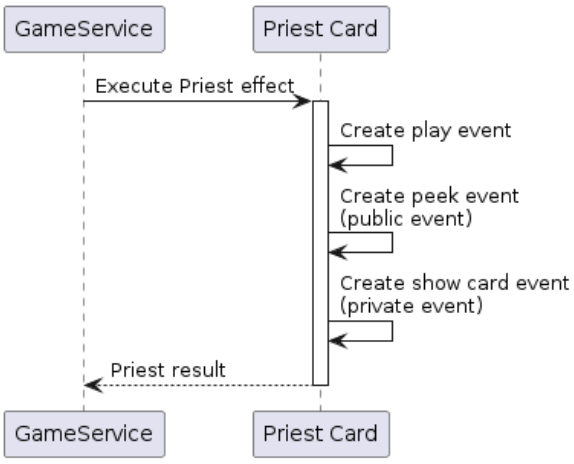

Figura 7 – Jogada da carta Sacerdote. Fonte: os autores

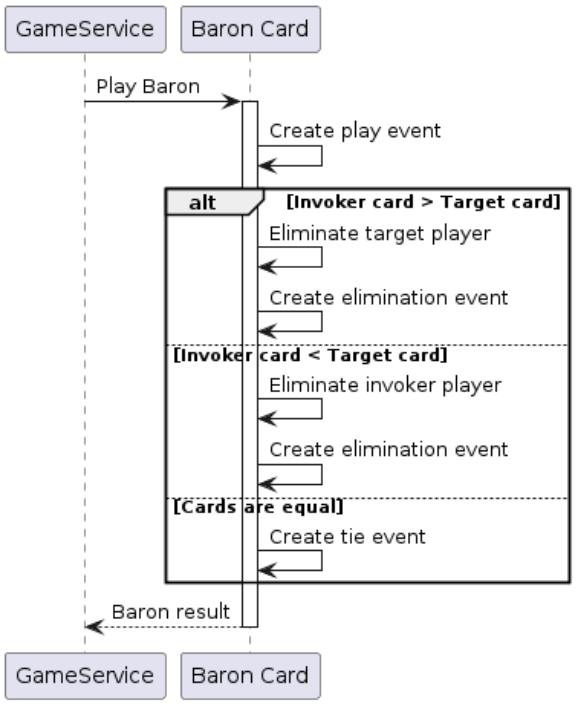

Figura 8 – Jogada da carta Barão. Fonte: os autores

3 Validação de Ações

O sistema implementa múltiplas camadas de validação para garantir que apenas ações válidas sejam executadas:

- **Validação de Turno**: Apenas o jogador da vez pode executar ações

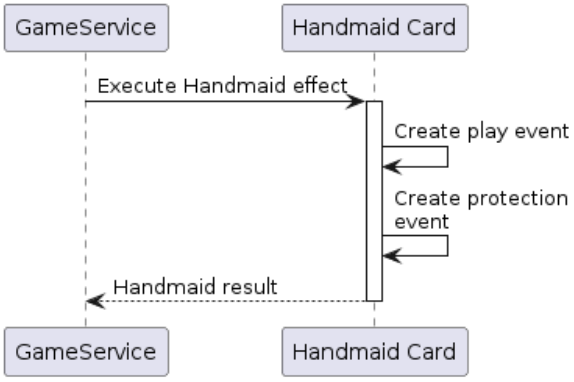

Figura 9 – Jogada da carta Criada. Fonte: os autores

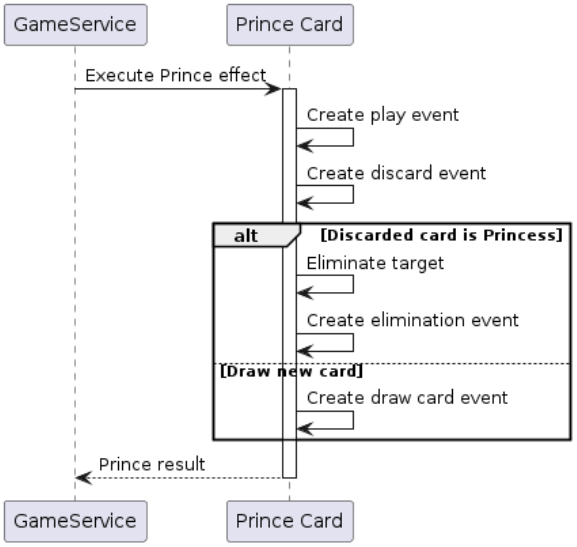

Figura 10 – Jogada da carta Príncipe. Fonte: os autores

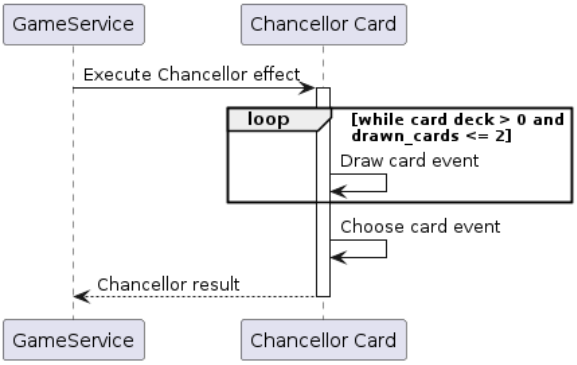

Figura 11 – Jogada da carta Chanceler. Fonte: os autores

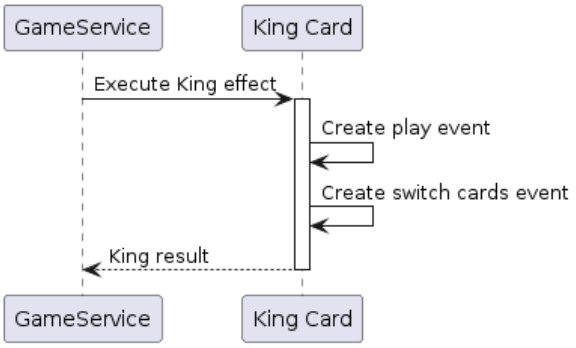

Figura 12 – Jogada da carta Rei. Fonte: os autores

- **Validação de Estado**: Ações só são permitidas nos estados apropriados do jogo
- **Validação de Cartas**: O jogador deve possuir a carta que deseja jogar
- **Validação de Requisitos**: Cartas com efeitos específicos requerem parâmetros adicionais

4 Experiência do Usuário

4.1 Interface Responsiva

O frontend desenvolvido em Svelte oferece uma interface moderna e responsiva, com animações suaves que podem ser desabilitadas para melhorar a performance em

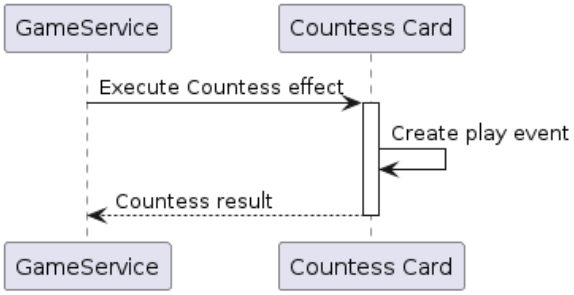

Figura 13 – Jogada da carta Condessa. Fonte: os autores

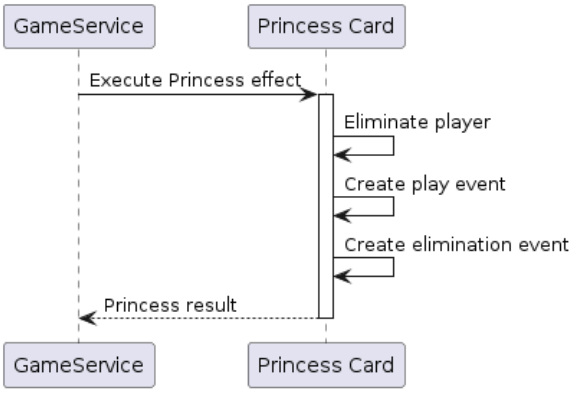

Figura 14 – Jogada da carta Princesa. Fonte: os autores

dispositivos menos potentes.

4.2 Feedback Visual

O sistema fornece feedback visual claro para todas as ações:

- **Indicadores de Turno**: Mostra claramente de quem é a vez
- **Log de Jogo**: Mantém histórico de todas as ações executadas
- **Animações de Cartas**: Visualiza movimentação e efeitos das cartas
- **Notificações de Estado**: Informa mudanças importantes no jogo

4.3 Acessibilidade

O sistema inclui recursos de acessibilidade como:

- **Suporte a Múltiplos Idiomas**: Interface disponível em português, inglês e espanhol
- **Controles Intuitivos**: Interface clara e fácil de navegar

Esta arquitetura garante que o sistema Love Letter ofereça uma experiência de jogo fluida, segura e envolvente para todos os participantes, mantendo a fidelidade às regras originais do jogo enquanto aproveita as capacidades da tecnologia moderna para criar uma experiência multiplayer online robusta.

5 Endpoints da Aplicação

1 Endpoints HTTP

Abaixo estão listados os principais endpoints HTTP disponíveis na API do backend, organizados por funcionalidade:

1.1 Gerenciamento de Conta

- **POST /api/Account/create**
  - Cria uma nova conta de usuário.
  - **Requer autenticação?** Não
  - **Body:** username, email, password
  - **Retorno:** 200 (token de acesso e mensagem de sucesso), 409 (usuário já existe)
- **POST /api/Account/login**
  - Realiza login do usuário.
  - **Requer autenticação?** Não
  - **Body:** email, password
  - **Retorno:** 200 (token de acesso), 401 (credenciais inválidas)
- **PUT /api/Account/email**
  - Atualiza os dados da conta do usuário autenticado.
  - **Requer autenticação?** Sim (JWT Bearer Token)
  - **Body:** username
  - **Retorno:** 200 (sucesso), 401 (não autenticado), 403 (proibido), 404 (usuário não encontrado)
- **DELETE /api/Account/email**
  - Deleta a conta do usuário autenticado.
  - **Requer autenticação?** Sim (JWT Bearer Token)
  - **Retorno:** 200 (sucesso), 401 (não autenticado), 403 (proibido), 404 (usuário não encontrado)

1.2 Salas de Jogo

- **POST /api/GameRoom**
  - Cria uma nova sala de jogo.
  - **Requer autenticação?** Sim (JWT Bearer Token)
  - **Body:** nome da sala, senha (opcional)
  - **Retorno:** 200 (ID da sala), 400 (erro de validação), 500 (erro interno)
- **GET /api/GameRoom**
  - Lista todas as salas de jogo disponíveis.
  - **Requer autenticação?** Sim (JWT Bearer Token)
  - **Retorno:** 200 (lista de salas), 500 (erro interno)
- **GET /api/GameRoom/user**
  - Lista as salas em que o usuário autenticado está participando.
  - **Requer autenticação?** Sim (JWT Bearer Token)
  - **Retorno:** 200 (lista de salas do usuário), 500 (erro interno)
- **DELETE /api/GameRoom/roomId**
  - Deleta uma sala de jogo (se o usuário for o host).
  - **Requer autenticação?** Sim (JWT Bearer Token)
  - **Retorno:** 204 (sucesso), 404 (sala não encontrada ou não autorizado), 500 (erro interno)

1.3 Saúde da Aplicação

- **GET /Health**
  - Verifica se a aplicação está funcionando corretamente.
  - **Requer autenticação?** Não
  - **Retorno:** 200 (status e timestamp)

2 Mensagens WebSocket (SignalR)

A aplicação utiliza WebSocket via SignalR para comunicação em tempo real durante as partidas. Abaixo estão listadas as principais mensagens (eventos) trocadas entre cliente e servidor, organizadas por funcionalidade:

2.1 Mensagens Recebidas do Servidor

- **JoinedRoom**: Confirmação de entrada em uma sala de jogo.
- **UserJoined**: Notifica que um novo jogador entrou na sala.
- **UserLeft**: Notifica que um jogador saiu da sala.
- **UsernameChanged**: Informa alteração de nome de usuário de um participante.
- **RoundStarted**: Início de uma nova rodada.
- **CurrentGameStatus**: Atualização do estado atual da partida.
- **NextTurn**: Indica de quem é a vez de jogar.
- **PlayerDrewCard**: Notifica que um jogador comprou uma carta.
- **GameStartError**: Erro ao tentar iniciar a partida.
- **PlayerUpdatePrivate**: Atualização privada do estado do jogador (cartas na mão, etc).
- **DrawCardError**: Erro ao tentar comprar carta.
- **CardRequirements**: Informa requisitos para jogar determinada carta.
- **PlayCardError**: Erro ao tentar jogar uma carta.
- **PlayCard**: Alguém jogou uma carta (tipo e jogador).
- **GuessCard**: Alguém fez uma tentativa de adivinhar carta de outro jogador.
- **PeekCard**: Alguém olhou a carta de outro jogador.
- **ShowCard**: Uma carta foi revelada.
- **CompareCards**: Comparação de cartas entre jogadores.
- **ComparisonTie**: Empate na comparação de cartas.
- **DiscardCard**: Um jogador descartou uma carta.
- **DrawCard**: Um jogador comprou uma carta.
- **CardReturnedToDeck**: Cartas devolvidas ao baralho.
- **PlayerEliminated**: Um jogador foi eliminado.
- **SwitchCards**: Troca de cartas entre jogadores.

- **PlayerProtected**: Jogador protegido por efeito de carta.
- **ChooseCard**: Jogador deve escolher uma carta.
- **PublicPlayerUpdate**: Atualização pública do estado de um jogador.
- **CardChoiceSubmitted**: Jogador submeteu escolha de carta.
- **CardChoiceError**: Erro ao submeter escolha de carta.
- **MandatoryCardPlay**: Jogada obrigatória de carta específica.
- **RoundWinners**: Fim de rodada, vencedores.
- **BonusPoints**: Jogadores que receberam pontos bônus.
- **GameOver**: Fim de jogo, vencedores.

2.2 Mensagens Enviadas ao Servidor

- **JoinRoom**: Entrar em uma sala de jogo (parâmetros: roomId, senha).
- **LeaveRoom**: Sair da sala de jogo.
- **ReconnectToRoom**: Reconectar-se a uma sala.
- **StartGame**: Iniciar a partida.
- **DrawCard**: Comprar carta.
- **GetCardRequirements**: Solicitar requisitos para jogar uma carta.
- **PlayCard**: Jogar uma carta (parâmetros: tipo, alvo, etc).
- **SubmitCardChoice**: Submeter escolha de carta após efeitos especiais.
- **GetCurrentGameStatus**: Solicitar status atual da partida.

Cada mensagem pode conter parâmetros específicos, conforme a ação desejada. Essas mensagens permitem a sincronização em tempo real do estado do jogo entre todos os participantes.

6 Tabelas da Base de Dados

1 Visão Geral das Tabelas

A base de dados do sistema é composta por três tabelas principais: **Users**, **Games** e **Players**. A seguir, detalhamos cada uma delas:

1.1 Tabela Users

- **Email**: identificador único do usuário (chave primária).
- **Username**: nome de exibição do usuário.
- **PasswordHash**: senha do usuário armazenada de forma segura (hash).

1.2 Tabela Games

- **Id**: identificador único da partida (UUID, chave primária).
- **Name**: nome da partida.
- **HostEmail**: email do usuário que criou a partida (chave estrangeira para Users).
- **Password**: senha da sala (opcional).
- **MaxTokens**: número máximo de tokens para vencer.
- **GameState**: estado atual da partida (ex: aguardando, em andamento, finalizada).
- **CurrentPlayerIndex**: índice do jogador da vez.
- **CardsDeck**: baralho de cartas da partida (array de inteiros).
- **ReservedCard**: carta reservada.
- **CreatedAt** / **UpdatedAt**: datas de criação e atualização.

Relações:

- **HostEmail** referencia Users.
- Relação 1:N com Players (uma partida possui vários jogadores).

1.3 Tabela Players

- **GameId**: identificador da partida (chave primária composta).
- **UserEmail**: email do usuário (chave primária composta).
- **Id**: identificador interno do jogador na partida.
- **HoldingCards**: cartas na mão do jogador (array de inteiros).
- **PlayedCards**: cartas já jogadas pelo jogador (array de inteiros).
- **Score**: pontuação do jogador.
- **Status**: status do jogador na partida (ex: ativo, eliminado).

Relações:

- **GameId** referencia Games.
- **UserEmail** referencia Users.

Essas tabelas e seus relacionamentos garantem o controle de usuários, partidas e jogadores, permitindo o gerenciamento eficiente do jogo e suas regras.

7 Resultados

O projeto está hospedado em um repositório público no GitHub, disponível em: `https://github.com/cardoso42/cartas-de-amor`

Foi possível gerar certificados digitais para o estabelecimento de conexão HTTPS com o site `https://cardoso42.site`, que está hospedado em uma instância da AWS. Isso garante a segurança na comunicação entre cliente e servidor, utilizando criptografia TLS.

A seguir, apresentamos uma captura de tela exemplificando o funcionamento do sistema:

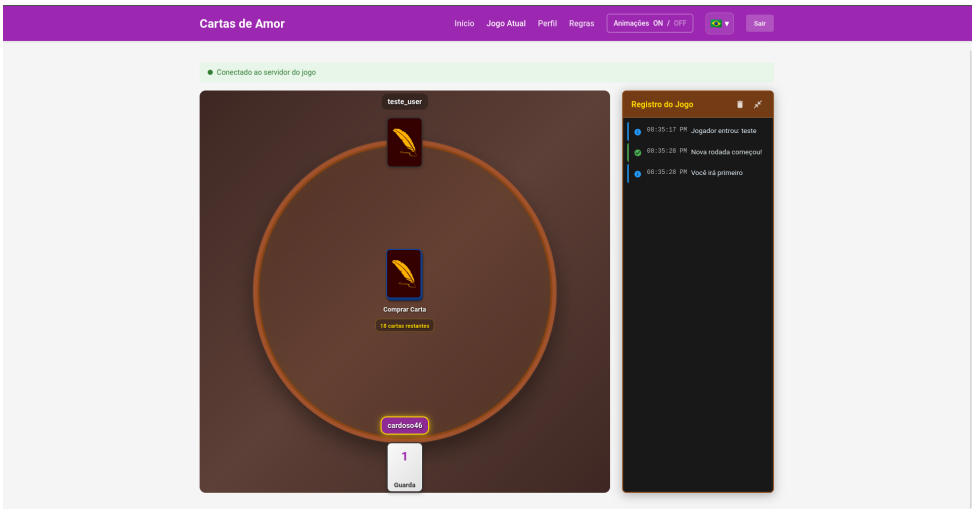

Figura 15 – Exemplo de tela do sistema em funcionamento. Fonte: os autores

A Regras do Jogo

1 Visão Geral

Love Letter é um jogo de dedução, risco e eliminação onde os jogadores competem para entregar uma carta de amor à princesa. Cada carta tem um valor diferente e habilidades especiais que podem ajudar ou atrapalhar os jogadores. É um jogo de cartas jogado com 2 a 6 jogadores. O objetivo é vencer um certo número de rodadas (tokens de afeição) sendo o último jogador restante ou segurando a carta de maior valor ao final de uma rodada.

2 Fluxo do Jogo

- Cada jogador começa com uma carta.
- Em seu turno, o jogador compra uma carta e escolhe uma das duas cartas para jogar.
- O efeito da carta jogada é aplicado imediatamente.
- Jogadores podem ser eliminados durante a rodada com base nos efeitos das cartas.
- A rodada termina quando resta apenas um jogador ou quando o baralho acaba.
- O jogador com a carta de maior valor vence a rodada.

3 Lista de Cartas e Efeitos

- **Spy (0)**: Ao final da rodada, se o jogador for o *único* restante que jogou ou descartou um Spy durante aquela rodada, recebe 1 token de afeição, mesmo que não tenha vencido a rodada.
- **Guard (1)**: Adivinhe a carta na mão de outro jogador (exceto Guard). Se acertar, o jogador é eliminado.
- **Priest (2)**: Veja a carta na mão de outro jogador.
- **Baron (3)**: Compare a mão com outro jogador; quem tiver a carta de menor valor é eliminado.
- **Handmaid (4)**: Proteção contra todos os efeitos até seu próximo turno.

- **Prince (5)**: Escolha qualquer jogador (inclusive você) para descartar a mão (revelando a carta, mas sem aplicar seus efeitos) e comprar uma nova carta.
- **Chancellor (6)**: Compre 2 cartas e escolha apenas uma das três cartas na mão para manter. As outras duas cartas retornam ao final do baralho.
- **King (7)**: Troque de mão com outro jogador.
- **Countess (8)**: Deve ser jogada se estiver na mão junto com Prince ou King.
- **Princess (9)**: Se descartada ou jogada, o jogador é eliminado.

4 Como Vencer o Jogo

Um jogador ganha um token de afeição a cada rodada vencida. O número de tokens necessários para vencer depende do número de jogadores, conforme a tabela abaixo:

Jogadores	2	3	4	5	6
Tokens para Vencer	6	5	4	3	3

Por exemplo, em um jogo para 2 jogadores, o primeiro a conquistar 6 tokens vence a partida. Em jogos com 5 ou 6 jogadores, apenas 3 tokens são necessários.