Gomoku - Projeto Web UFSC: Implementação de um Jogo Online com FastAPI, React e MongoDB

Alec Oliveira Coelho*, Luan Rodrigo da Silva Costa*
*Departamento de Informática e Estatística (INE), Universidade Federal de Santa Catarina (UFSC), Florianópolis, SC, Brasil

***Resumo*—Este artigo descreve a concepção e implementação de um projeto web completo para a disciplina INE5646 - Programação para Web da UFSC. O projeto consiste em um jogo Gomoku (Five in a Row) funcional, desenvolvido com uma arquitetura de aplicação web moderna. O *backend* foi implementado em Python utilizando o *framework* FastAPI, escolhido por sua alta performance e suporte assíncrono, essencial para comunicação em tempo real. O *frontend* foi desenvolvido em React com TypeScript, proporcionando uma interface de usuário reativa, responsiva e com tipagem segura. O armazenamento de dados, incluindo perfis de usuário, estados de jogo e gravações de partidas, é gerenciado pelo banco de dados NoSQL MongoDB, um requisito obrigatório do projeto. A comunicação em tempo real entre jogadores, como o *chat* e a sincronização de movimentos, é realizada via WebSockets. Funcionalidades avançadas, como *videochat* P2P com WebRTC e gravação de partidas usando FFMPEG, também foram implementadas, demonstrando a integração de múltiplas tecnologias complexas em um sistema coeso.**

***Palavras-chave*—Gomoku, Programação Web, FastAPI, React, TypeScript, MongoDB, WebSocket, WebRTC, FFMPEG, Arquitetura MVC.**

***Abstract*—This article describes the design and implementation of a complete web project for the INE5646 - Web Programming course at UFSC. The project consists of a functional Gomoku (Five in a Row) game, developed with a modern web application architecture. The backend was implemented in Python using the FastAPI framework, chosen for its high performance and asynchronous support, essential for real-time communication. The frontend was developed in React with TypeScript, providing a reactive, responsive, and type-safe user interface. Data storage, including user profiles, game states, and match recordings, is managed by the NoSQL database MongoDB, a mandatory requirement of the project. Real-time communication between players, such as chat and move synchronization, is handled via WebSockets. Advanced features, including P2P video chat with WebRTC and match recording using FFMPEG, were also implemented, demonstrating the integration of multiple complex technologies into a cohesive system.**

***Index Terms*—Gomoku, Web Programming, FastAPI, React, TypeScript, MongoDB, WebSocket, WebRTC, FFMPEG, MVC Architecture.**

I. Introdução

O projeto web desenvolvido para a disciplina INE5646 - Programação para Web da UFSC teve como objetivo a criação de uma aplicação web completa e funcional, implementando o jogo de estratégia Gomoku, também conhecido como "Cinco em Linha".

Universidade Federal de Santa Catarina (UFSC) (INE5646). Correspondência ao autor: Alec Coelho (email: aleccoelho50@gmail.com) e uanL ostaC (email: email.autor2@grad.ufsc.br).

A. Motivação

A principal motivação do projeto foi aplicar os conceitos e técnicas abordados na disciplina em um cenário prático e complexo. A escolha de um jogo *multiplayer* em tempo real permitiu explorar tecnologias avançadas de *backend* e *frontend*, indo além de uma aplicação CRUD (Create, Read, Update, Delete) padrão. O desafio consistia em construir um sistema robusto, escalável e que integrasse diversas ferramentas exigidas pelo plano de ensino, como o banco de dados MongoDB e o protocolo HTTPS.

B. Problema

O problema central do projeto foi desenvolver uma aplicação *full-stack* que suportasse múltiplos modos de jogo (PvP online, PvP local e PvE contra IA), autenticação de usuários, comunicação em tempo real (chat e jogadas) e funcionalidades complexas de mídia, como *videochat* P2P e gravação de partidas. A solução deveria seguir o padrão de projeto MVC (Model-View-Controller), ser responsiva (desktop e mobile) e atender a rigorosos requisitos de segurança e infraestrutura, incluindo o *deploy* obrigatório em um servidor VPS-UFSC.

C. Trabalhos Relacionados

(Esta seção deve ser preenchida com a pesquisa de outros projetos ou artigos que implementaram soluções similares, *e.g.*, outros jogos web *multiplayer*, e comparar suas arquiteturas com a escolhida neste trabalho.)

D. Contribuição do Trabalho

A principal contribuição deste trabalho é a integração bem-sucedida de um ecossistema de tecnologias modernas para criar uma experiência de usuário complexa e em tempo real. Demonstra-se uma arquitetura desacoplada onde o *backend* (FastAPI) atua como uma API RESTful e um servidor WebSocket, o *frontend* (React) consome esses serviços de forma reativa, e o MongoDB gerencia a persistência de dados. Além disso, o projeto serve como um estudo de caso prático na implementação de funcionalidades avançadas como WebRTC para *videochat* e FFMPEG para gravação e *streaming* de vídeo, ambas gerenciadas através do MongoDB GridFS.

E. Organização do Trabalho

Este artigo está organizado da seguinte forma: A Seção II apresenta a fundamentação teórica sobre as principais tecnologias utilizadas. A Seção III detalha os materiais e métodos, incluindo a arquitetura do sistema e o roteiro de instalação. A Seção IV discute os resultados obtidos, a estrutura do projeto, os problemas encontrados e as soluções de segurança. Finalmente, a Seção V apresenta as conclusões do trabalho.

II. Fundamentação Teórica

Esta seção aborda os conceitos teóricos das principais tecnologias que formam a *stack* do projeto Gomoku.

A. FastAPI

FastAPI é um *framework* web Python moderno de alta performance, baseado em Starlette (para a parte assíncrona ASGI) e Pydantic (para validação de dados) [8]. Sua arquitetura permite o desenvolvimento rápido de APIs RESTful e o gerenciamento eficiente de conexões WebSocket, sendo ideal para aplicações que exigem baixa latência e I/O intensivo, como o *backend* deste projeto.

B. React e TypeScript

React é uma biblioteca JavaScript para a construção de interfaces de usuário (UI) baseada em componentes [9]. Ele utiliza um *Virtual DOM* para otimizar as atualizações da UI, resultando em uma experiência de usuário fluida. TypeScript é um *superset* do JavaScript que adiciona tipagem estática opcional [10], utilizado no projeto para garantir a manutenibilidade, robustez e escalabilidade do código do *frontend*.

C. MongoDB

MongoDB é um banco de dados NoSQL orientado a documentos, que armazena dados em estruturas flexíveis similares a JSON (chamadas BSON) [11]. Foi o banco de dados obrigatório para o projeto e é utilizado para persistir dados de usuários, jogos e placares. O projeto também utiliza o MongoDB GridFS, um mecanismo para armazenar arquivos grandes, como as gravações de vídeo das partidas.

D. WebSocket

O protocolo WebSocket fornece um canal de comunicação bidirecional e *full-duplex* sobre uma única conexão TCP. Diferente do ciclo de requisição-resposta do HTTP, o WebSocket mantém uma conexão persistente, permitindo que o servidor envie dados ao cliente proativamente. Esta tecnologia é a base para as funcionalidades em tempo real do projeto, como o chat, o *lobby* e a sincronização dos movimentos no tabuleiro.

E. WebRTC (Web Real-Time Communication)

WebRTC é um projeto de código aberto e API que permite a comunicação de áudio, vídeo e dados em tempo real (P2P) diretamente entre navegadores, sem a necessidade de *plugins* intermediários [12]. No projeto, é utilizado para implementar a funcionalidade de *videochat* entre os dois jogadores de uma partida.

F. FFMPEG

FFMPEG é uma suíte de *software* livre para manipulação, gravação, conversão e *streaming* de áudio e vídeo [13]. Conforme os requisitos do projeto, o FFMPEG é utilizado no *backend* para processar e gravar as partidas no formato WebM, que são subsequentemente armazenadas no MongoDB GridFS.

III. Materiais e Métodos

Esta seção descreve a arquitetura do projeto, as ferramentas utilizadas e o processo de configuração e instalação para replicação do ambiente.

A. Arquitetura da Aplicação

A aplicação segue um padrão de projeto próximo ao MVC (Model-View-Controller), desacoplado em dois serviços principais:

- **Backend (API):** Implementado em FastAPI, serve como o *Controller* e o *Model*. Ele expõe uma API RESTful para gerenciamento de usuários e jogos, e um *endpoint* WebSocket para comunicação em tempo real (chat, lobby, jogadas). Ele se comunica com o banco de dados MongoDB (usando o *driver* assíncrono Motor) para persistir os dados.
- **Frontend (View):** Implementado em React com TypeScript, é a camada de visualização. Consome a API RESTful do *backend* para operações de dados e se conecta ao WebSocket para atualizações em tempo real. O estado da aplicação é gerenciado localmente nos componentes e através de Contexts do React.
- **Banco de Dados:** O MongoDB atua como a camada de persistência.
- **Infraestrutura:** A aplicação é orquestrada utilizando Docker e Docker Compose, facilitando os ambientes de desenvolvimento e produção.

B. Tecnologias, Frameworks e APIs

A Tabela I e a Tabela II resumem as principais tecnologias usadas no *backend* e *frontend*, respectivamente.

Tabela I
Stack Tecnológica do Backend

Tecnologia	Descrição
FastAPI	Framework web assíncrono (ASGI)
Motor	Driver MongoDB assíncrono
Uvicorn	Servidor ASGI
WebSockets	Comunicação em tempo real
python-jose	Autenticação JWT
ffmpeg-python	Wrapper para gravação de vídeo

C. Softwares e Roteiro de Instalação

O projeto é desenhado para ser executado com Docker e Docker Compose, simplificando a configuração do ambiente.

Tabela II
STACK TECNOLÓGICA DO FRONTEND

Tecnologia	Descrição
React	Biblioteca de UI
TypeScript	Tipagem estática para JavaScript
Axios	Cliente HTTP (REST)
Socket.io-client	Cliente WebSocket
React Router	Roteamento de páginas

1) Pré-requisitos:

- Docker
- Docker Compose
- Git

2) Roteiro de Instalação (Produção): O processo de *deploy* em modo de produção é feito com o Docker Compose, que constrói e orquestra os contêineres do *backend*, *frontend* (servido via Nginx) e MongoDB.

```
# Clone o repositorio #erro
git clone https://github.com/Coelho50/Gomoku.git
cd Gomoku

# Execute em modo producao (com Nginx) #erro
docker-compose -f docker-compose.yml up -d --profile
    production
```

Listing 1. Comandos para execução em modo produção

3) Roteiro de Instalação (Desenvolvimento): O modo de desenvolvimento utiliza o Docker Compose para subir os serviços com *hot-reloading* no *backend* (Uvicorn) e *frontend* (React Scripts), além de expor um painel de administração do MongoDB (Mongo Express).

```
# Execute com MongoDB Admin Interface
docker-compose --profile debug up

# Portas em Modo Debug:
# Frontend: http://localhost:9001
# Backend API: http://localhost:9000
# MongoDB Admin: http://localhost:8081
```

Listing 2. Comandos para execução em modo desenvolvimento

D. Links do Projeto

- **Repositório GitHub:** https://github.com/Coelho50/Gomoku
- **Link da Aplicação (VPS-UFSC):** https://pw.alec.coelho.vms.ufsc.br

IV. RESULTADOS

Esta seção apresenta os resultados da implementação, a estrutura final do projeto, os problemas encontrados durante o desenvolvimento e as soluções de segurança aplicadas.

A. Estrutura do Projeto (MVC)

O *backend* foi estruturado seguindo o padrão MVC, adaptado para o FastAPI:

- **Models:** Definidos em ‘backend/models/‘, usando Pydantic para validação de dados de API e classes para os modelos do MongoDB.
- **Views (Templates):** O *backend* é *headless* (sem *view*), sendo o React a camada de visualização desacoplada.
- **Controllers (Rotas):** Definidos em ‘backend/routers/‘. Cada arquivo (ex: ‘auth.py‘, ‘games.py‘, ‘websocket.py‘) agrupa a lógica de negócios para um conjunto de *endpoints*, recebendo requisições, interagindo com os serviços e retornando respostas JSON.
- **Services:** A lógica de negócios complexa (ex: cálculo de ELO, gravação FFMPEG) foi abstraída em `backend/services/`, como `ranking_service.py` e `ffmpeg_service.py`.

A Figura 1 (placeholder) ilustra a árvore de diretórios principal do *backend*.

Placeholder para Figura: Árvore de Diretórios do Backend

Figura 1. Estrutura de diretórios do serviço de backend.

B. Funcionalidades Implementadas

O projeto implementou com sucesso todas as funcionalidades avançadas requisitadas:

- **Gravação com FFMPEG:** O ‘`ffmpeg_service.py`‘ gerencia a gravação de partidas, com APIs para iniciar, parar e listar gravações. Os vídeos são armazenados no MongoDB GridFS e podem ser acessados via *streaming* por uma URL.
- **Videochat com WebRTC:** O ‘`webrtc_service.py`‘ atua como servidor de sinalização (via WebSocket) para estabelecer conexões P2P entre os jogadores, incluindo configuração de servidores STUN.
- **Sistema de Ranking ELO:** O ‘`ranking_service.py`‘ calcula o ELO dos jogadores após cada partida online (Fator `K=32`), armazena estatísticas e fornece *endpoints* para um *leaderboard* global.
- **Administração CRUD:** O ‘routers/admin.py‘ implementa um conjunto completo de rotas protegidas para gerenciamento de usuários, jogos e configurações do sistema.
- **Design Responsivo:** Conforme detalhado em ‘`DESIGN_IMPLEMENTATION_SUMMARY.md`‘, foi implementado um sistema de design responsivo completo, garantindo a usabilidade em dispositivos *mobile*, *tablet* e *desktop*.

As Figuras 2 e 3 (placeholders) mostram *screenshots* da aplicação final.

Placeholder para Screenshot: Tela do Lobby

Figura 2. Interface do lobby de jogos, mostrando a seleção de modo de jogo e a lista de jogadores online.

C. Problemas Encontrados e Soluções

Durante o desenvolvimento, diversos desafios técnicos surgiram:

Placeholder para Screenshot: Tela de Jogo

Figura 3. Interface da partida, exibindo o tabuleiro, o chat em tempo real e o componente de videochat.

- **Instabilidade de Conexão WebSocket:** O *lobby* desconectava imediatamente após a conexão. A causa raiz era que o *backend* aceitava a nova conexão *antes* de fechar a conexão antiga do mesmo usuário. A correção envolveu reordenar o fluxo para: 1) Autenticar token, 2) Fechar conexão antiga, 3) Aceitar nova conexão.
- **Matchmaking com Reconexão:** Jogadores desapareciam da fila ao reconectar. A função `'disconnect_from_lobby()'` removia o usuário de `'online_players'`. A solução foi modificar a lógica para, durante uma reconexão, apenas remover o *socket* antigo, mantendo o usuário na lista de jogadores online.
- **Modal de Vitória Exibido como Erro:** Um bug visual crítico fazia com que a tela de vitória fosse renderizada como um modal de erro. A correção foi feita no ‘Game.tsx‘, garantindo que o estado ‘isGameOver‘ com um vencedor disparasse o componente ‘GameSuccessModal‘ ao invés de um modal de erro genérico.

D. Vulnerabilidades e Soluções de Segurança

A segurança foi um pilar do projeto, conforme os requisitos da disciplina:

- **Autenticação:** Implementada com JSON Web Tokens (JWT) (RFC 7519), utilizando *tokens* de acesso (curta duração) e *refresh tokens* (longa duração) armazenados de forma segura.
- **HTTPS:** O *deploy* em produção no VPS-UFSC utilizará Nginx como *proxy* reverso com certificados SSL/TLS (Let's Encrypt), garantindo a criptografia do tráfego.
- **CORS:** O *backend* FastAPI é configurado para permitir requisições apenas das origens do *frontend* definidas em variáveis de ambiente.
- **Prevenção de Injeção:** O uso de um *driver* moderno (Motor) para o MongoDB mitiga riscos de NoSQL Injection, pois as consultas são parametrizadas. A validação de entrada é feita pelo Pydantic.
- **XSS/CSRF:** Medidas de sanitização de *inputs* no *frontend* (especialmente em campos de chat) e *headers* de segurança (via Nginx/Helmet.js) foram planejadas para prevenir XSS e CSRF.

V. Conclusão

O desenvolvimento do projeto Gomoku Web atingiu seus objetivos principais, entregando uma aplicação *full-stack* robusta que não apenas cumpre os requisitos básicos do jogo, mas também implementa um conjunto de funcionalidades avançadas e complexas.

A arquitetura escolhida, baseada em FastAPI para o *backend* e React para o *frontend*, provou ser altamente eficaz. O FastAPI gerenciou com excelência as operações assíncronas de WebSockets e as requisições da API, enquanto o React forneceu uma interface de usuário fluida e responsiva. A integração de WebRTC, FFMPEG e um sistema de ranking ELO demonstrou a capacidade de estender a aplicação com serviços de nível profissional.

Os desafios encontrados, especialmente na estabilidade da comunicação em tempo real e no *design* responsivo, foram superados e resultaram em um sistema mais resiliente e polido. Conforme o relatório `'FUNCIONALIDADES_IMPLEMENTADAS.md'`, o *backend* encontra-se 100% completo em termos de APIs, enquanto o *frontend* está em fase avançada de integração.

Como próximos passos, restam o *deploy* final no servidor VPS-UFSC com configuração HTTPS e a conclusão da integração das páginas de *frontend* pendentes (Ranking, Admin). O projeto conclui-se como um sucesso, servindo como um portfólio abrangente das técnicas de programação web modernas.

Referências

[1] W. B. da Silva, "Instruções para o PW,"*Moodle UFSC - INE5646*, 2025. Acessado em: 05/11/2025. [Disponível em: Gomoku/docs/projeto.md]

[2] A. O. Coelho e L. R. da S. Costa, "FUNCIONALIDADES IMPLEMENTADAS - Gomoku Project,"*Documentação do Projeto*, 2025. [Disponível em: `Gomoku/FUNCIONALIDADES_IMPLEMENTADAS.md`]

[3] A. O. Coelho e L. R. da S. Costa, "Gomoku - Projeto Web UFSC,"*Documentação do Projeto (README.md)*, 2025. [Disponível em: Gomoku/README.md]

[4] A. O. Coelho e L. R. da S. Costa, "docker-compose.yml,"*Configuração do Projeto*, 2025. [Disponível em: Gomoku/docker-compose.yml]

[5] A. O. Coelho e L. R. da S. Costa, "requirements.txt,"*Dependências do Backend*, 2025. [Disponível em: Gomoku/backend/requirements.txt]

[6] A. O. Coelho e L. R. da S. Costa, "package.json,"*Dependências do Frontend*, 2025. [Disponível em: Gomoku/frontend/package.json]

[7] A. O. Coelho e L. R. da S. Costa, "Resumo de Implementação - Design System Responsivo,"*Documentação do Projeto*, 2025. [Disponível em: `Gomoku/DESIGN_IMPLEMENTATION_SUMMARY.md`]

[8] *FastAPI*, Tiangolo. [Online]. Disponível: https://fastapi.tiangolo.com/

[9] *React*, Meta. [Online]. Disponível: https://react.dev/

[10] *TypeScript*, Microsoft. [Online]. Disponível: https://www.typescriptlang.org/

[11] *MongoDB Documentation*, MongoDB Inc. [Online]. Disponível: https://www.mongodb.com/docs/

[12] *WebRTC*, Google. [Online]. Disponível: https://webrtc.org/

[13] *FFMPEG*, FFMPEG.org. [Online]. Disponível: https://ffmpeg.org/

Apêndice A
Código Fonte de Configuração

Apresentam-se a seguir os principais arquivos de configuração do projeto, que definem a infraestrutura (Docker), as

dependências do *backend* e as dependências do *frontend*.

A. docker-compose.yml

B. package.json (Frontend)