



# Backend Gateway

Sistema de Recomendação de Games com ML Integration

Versão 2.0 | Documentação Completa



## Índice

[Visão Geral](#)

[Arquitetura do Sistema](#)

[Configuração](#)

[Estrutura do Projeto](#)

[Endpoints da API](#)

[Controllers](#)

[Models](#)

[Middleware](#)

[Exemplos de Uso](#)

[Troubleshooting](#)



## Visão Geral

O Backend Gateway é um servidor Express.js que atua como intermediário centralizado entre o Frontend/Mobile e o serviço de Machine Learning em Python (Flask). Ele gerencia autenticação, validação de dados, e proxying de requisições para o motor de recomendações.

## Principais Características

- **Gateway Pattern:** Centraliza todas as requisições da aplicação
- **Autenticação JWT:** Sistema seguro de autenticação de usuários
- **Validação de Dados:** Validação robusta de entrada com feedback claro
- **Proxy Flask:** Integração seamless com serviço de ML
- **Tratamento de Erros:** Middleware centralizado para erros
- **CORS Configurado:** Suporta requisições do Frontend/Mobile
- **In-Memory Storage:** Armazenamento rápido de usuários (desenvolvimento)

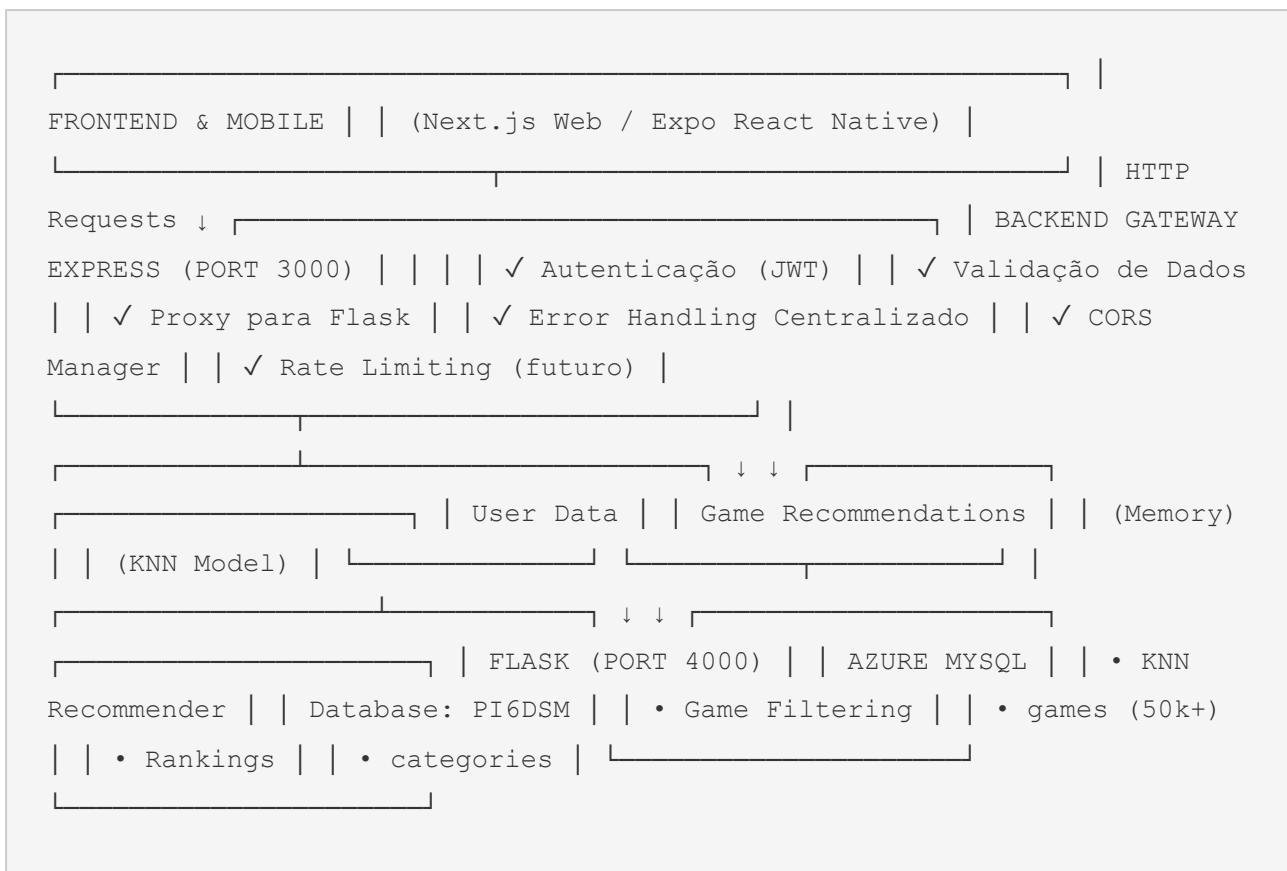
## Tecnologias Utilizadas

Tecnologia	Versão	Descrição
Express.js	4.19.2	Framework HTTP server
Axios	1.7.7	Client HTTP para requisições
CORS	2.8.5	Cross-Origin Resource Sharing
dotenv	16.4.7	Variáveis de ambiente
JWT	8.5.1	Tokens de autenticação
Nodemon	3.1.4	Auto-reload em desenvolvimento



## Arquitetura do Sistema

O Backend implementa o padrão Gateway, funcionando como ponto único de entrada para todas as requisições:



## Fluxo de Requisições

- Cliente envia requisição:** Frontend/Mobile → Backend (<http://localhost:3000>)
- Backend processa:** Validação, CORS check, autenticação
- Roteamento:** Identifica qual controller deve processar
- Proxy Flask (se necessário):** Backend → Flask (<http://localhost:4000>)
- Resposta:** Backend retorna dados ao cliente



## Configuração

### Variáveis de Ambiente (.env)

O arquivo `.env` deve estar na raiz da pasta `back/`:

```
NODE_ENV=development
PORT=3000
JWT_SECRET=seu_secret_jwt_com_min_32_caracteres

# Frontend URLs
FRONTEND_URL=http://localhost:3001

# Flask Service
FLASK_HOST=localhost
FLASK_PORT=4000

# Azure MySQL
AZURE_MYSQL_HOST=13.68.75.61
AZURE_MYSQL_DATABASE=PI6DSM
AZURE_MYSQL_USER=claudio
AZURE_MYSQL_PASSWORD=FatecFranca123#
AZURE_MYSQL_PORT=3306

# URLs
MOBILE_URL=exp://localhost:8081
```

## Instalação de Dependências

```
cd back
npm install
```

## Executar em Desenvolvimento

```
npm run dev
# Ou com Flask integrado:
npm run server
```

## Executar em Produção

```
npm start
```



# Estrutura do Projeto

```

back/
├── src/
│   ├── index.js          # Entry point principal
│   ├── controllers/
│   │   ├── userController.js    # Autenticação e usuários
│   │   ├── gameController.js    # Gerenciamento de jogos
│   │   └── recommendationController.js  # Recomendações
│   ├── models/
│   │   ├── userModel.js        # Modelo de usuário
│   │   └── entryModel.js       # Modelo de entrada (não usado)
│   ├── routes/
│   │   ├── userRoutes.js      # Rotas de usuários
│   │   ├── gameRoutes.js      # Rotas de jogos
│   │   └── recommendationRoutes.js # Rotas de recomendações
│   └── middleware/
│       ├── errorHandler.js    # Tratamento centralizado de erros
│       └── flaskProxy.js       # Cliente HTTP para Flask
├── .env                  # Variáveis de ambiente
└── package.json          # Dependências do projeto
└── README.md             # Documentação básica

```

## Descrição dos Arquivos Principais

### index.js

Arquivo principal que inicializa o servidor Express. Configura middleware global, CORS, rotas e trata erros globais.

### controllers/\*.js

Controladores contêm a lógica de negócio de cada funcionalidade. Recebem requisições e retornam respostas.

### models/\*.js

Models definem estruturas de dados e operações em memória. Atualmente usam armazenamento em memória.

### routes/\*.js

Rotas definem os endpoints HTTP e mapeiam para controladores apropriados.

### middleware/\*.js

Middleware processam requisições antes de chegarem aos controladores (validação, erro, proxy).



## Endpoints da API

### Autenticação e Usuários (/api/users)

**GET** /api/users

**Descrição:** Lista todos os usuários cadastrados

**Parâmetros:** Nenhum

**Resposta:**

```
{ "dados": [ { "id": 1, "nome": "João Silva", "email": "joao@example.com", "categorias": ["Action", "Adventure"], "criadoEm": "2025-11-14T..." } ] }
```

**GET** /api/users/categories**Descrição:** Lista todas as categorias válidas de jogos**Parâmetros:** Nenhum**Resposta:**

{ "categorias": [ "Action", "Adventure", "Indie", "RPG", ... ] }

**POST** /api/users**Descrição:** Cadastra novo usuário**Body:**

{ "nome": "João Silva", "email": "joao@example.com", "senha": "senha123", "confirmarSenha": "senha123", "categorias": ["Action", "Adventure", "Indie"] }

**Resposta (201):**

{ "mensagem": "Usuário criado com sucesso.", "dados": { "id": 1, "nome": "João Silva", "email": "joao@example.com", "categorias": ["Action", "Adventure", "Indie"], "criadoEm": "2025-11-14T..." } }

**POST** /api/users/login**Descrição:** Autentica usuário**Body:**

```
{ "email": "joao@example.com", "senha": "senha123" }
```

**Resposta (200):**

```
{ "mensagem": "Login realizado com sucesso.", "dados": { "id": 1, "nome": "João Silva", "email": "joao@example.com", "categorias": ["Action", "Adventure"], "criadoEm": "2025-11-14T..." } }
```

## Jogos (/api/games)

**GET** /api/games**Descrição:** Lista jogos com paginação**Parâmetros Query:**

- `page` (int): Página (padrão: 1)
- `limit` (int): Itens por página (padrão: 50)

**Exemplo:** GET /api/games?page=1&limit=10**GET** /api/games/:id**Descrição:** Busca jogo por ID**Parâmetros:** `id` (int)**Exemplo:** GET /api/games/42

**GET** /api/games/search

**Descrição:** Busca jogo por nome

**Parâmetros Query:**

- `q` (string): Termo de busca (obrigatório)

**Exemplo:** GET /api/games/search?q=minecraft

**GET** /api/games/categories

**Descrição:** Filtra jogos por até 4 categorias

**Parâmetros Query:**

- `cat1, cat2, cat3, cat4` (string): Categorias
- `limit` (int): Quantidade de resultados (padrão: 10)

**Exemplo:** GET /api/games/categories?

`cat1=Action&cat2=Adventure&limit=20`

**GET** /api/games/aleatorio

**Descrição:** Retorna um jogo aleatório

**Parâmetros:** Nenhum

**POST** /api/games/:id/rate**Descrição:** Registra avaliação de um jogo**Body:**

{ "positiva": true }

**Parâmetros:** `id` (int), `positiva` (boolean)

## Recomendações (/api/recommendations)

**GET** /api/recommendations/users/:userId**Descrição:** Recomendações personalizadas por categorias**Parâmetros:**

- `userId` (int): ID do usuário
- `limit` (int): Quantidade de recomendações (padrão: 10)

**GET** /api/recommendations/ranking/popular**Descrição:** Jogos mais populares**Parâmetros Query:**

- `limit` (int): Quantidade (padrão: 10)

**GET** /api/recommendations/ranking/best

**Descrição:** Jogos melhor avaliados

#### Parâmetros Query:

- `limit` (int): Quantidade (padrão: 10)
- `minRatings` (int): Avaliações mínimas (padrão: 5)

**GET** /api/recommendations/games/:id/similar

**Descrição:** Jogos similares a um jogo específico

#### Parâmetros:

- `id` (int): ID do jogo
- `limit` (int): Quantidade (padrão: 5)

**GET** /api/recommendations/system/health

**Descrição:** Status da saúde do sistema (Backend + Flask)

**Parâmetros:** Nenhum

#### Resposta:

```
{ "sucesso": true, "backend": "online", "flask": { "status": "healthy", "jogos_carregados": 50000, "modelo_treinado": true } }
```

## Controllers

### UserController

Gerencia autenticação e dados de usuários.

## Métodos Principais:

- **create(req, res):** Cadastra novo usuário com validação completa
- **login(req, res):** Autentica usuário por email/senha
- **getAll(req, res):** Lista todos os usuários
- **getCategories(req, res):** Retorna categorias válidas do CSV

## Validações:

- Nome: obrigatório, string
- Email: obrigatório, formato válido, único
- Senha: mínimo 6 caracteres, confirmação obrigatória
- Categorias: array não vazio, validação contra CSV

## GameController

Gerencia operações de jogos através do proxy Flask.

## Métodos Principais:

- **getAll(req, res, next):** Lista jogos com paginação
- **getById(req, res, next):** Busca jogo por ID
- **search(req, res, next):** Busca por nome com termo de busca
- **getByCategories(req, res, next):** Filtra até 4 categorias
- **getRandom(req, res, next):** Retorna jogo aleatório
- **rate(req, res, next):** Registra avaliação positiva/negativa

## RecommendationController

Gerencia recomendações e rankings.

## Métodos Principais:

- **getForUser(req, res, next):** Recomendações por categorias do usuário
- **getPopular(req, res, next):** Jogos mais populares

- **getBestRated(req, res, next):** Jogos melhor avaliados
- **getSimilar(req, res, next):** Jogos similares
- **getSystemHealth(req, res, next):** Status do sistema completo



## Models

### UserModel

Modelo de dados para usuários com armazenamento em memória.

Estrutura de Dados:

```
{
  id: Number,
  nome: String,
  email: String,
  categorias: Array<String>,
  passwordHash: String,
  salt: String,
  criadoEm: DateTime
}
```

Métodos Estáticos:

- **findAll():** Retorna todos os usuários
- **findByEmail(email):** Busca por email único
- **findByName(nome):** Busca por nome
- **create(dados):** Cria novo usuário com ID auto-incrementado

### Segurança de Senha

Senhas são hasheadas com PBKDF2:

- **Algoritmo:** PBKDF2 com SHA-512
- **Iterações:** 100.000

- **Salt:** 16 bytes aleatórios



## Middleware

### FlaskProxy Middleware

Cliente HTTP centralizado para comunicação com Flask usando Axios.

#### Configuração:

- Base URL: `http://FLASK_HOST:FLASK_PORT`
- Timeout: 10 segundos
- Header padrão: `Content-Type: application/json`

#### Interceptores:

- **Request:** Log de requisições saindo
- **Response:** Log de respostas recebidas
- **Error:** Log de erros de conexão

### Error Handler Middleware

Padroniza tratamento de erros em toda aplicação.

#### Tipos de Erro Tratados:

- Erros do Flask (ECONNREFUSED)
- Erros HTTP com status code
- Erros genéricos de servidor

#### Resposta Padronizada:

```
{  
  "erro": "Descrição do erro",  
  "detalhes": { ... }  
}
```



## Exemplos de Uso

### 1. Criar Usuário

```
curl -X POST http://localhost:3000/api/users \  
-H "Content-Type: application/json" \  
-d '{  
    "nome": "João Silva",  
    "email": "joao@example.com",  
    "senha": "senha123",  
    "confirmarSenha": "senha123",  
    "categorias": ["Action", "Adventure", "Indie"]  
}'
```

### 2. Login

```
curl -X POST http://localhost:3000/api/users/login \  
-H "Content-Type: application/json" \  
-d '{  
    "email": "joao@example.com",  
    "senha": "senha123"  
}'
```

### 3. Listar Categorias

```
curl http://localhost:3000/api/users/categories
```

### 4. Listar Jogos

```
curl "http://localhost:3000/api/games?page=1&limit=10"
```

## 5. Buscar Jogo por Nome

```
curl "http://localhost:3000/api/games/search?q=minecraft"
```

## 6. Filtrar por Categorias

```
curl "http://localhost:3000/api/games/categories?cat1=Action&cat2=Adventure&limit=10"
```

## 7. Avaliar Jogo

```
curl -X POST http://localhost:3000/api/games/42/rate \  
-H "Content-Type: application/json" \  
-d '{"positiva": true}'
```

## 8. Obter Recomendações Personalizadas

```
curl "http://localhost:3000/api/recommendations/users/1?limit=10"
```

## 9. Jogos Mais Populares

```
curl "http://localhost:3000/api/recommendations/ranking/popular?limit=10"
```

## 10. Verificar Saúde do Sistema

```
curl http://localhost:3000/api/recommendations/system/health
```



## Troubleshooting

### Erro: ECONNREFUSED 127.0.0.1:4000

**Problema:** Backend não consegue conectar ao Flask

### Solução:

- Verificar se Flask está rodando em `http://localhost:4000`
- Aguardar 10-15 segundos para modelo de ML treinar
- Confirmar `FLASK_HOST` e `FLASK_PORT` em `.env`
- Verificar se porta 4000 não está bloqueada por firewall

## Erro: EADDRINUSE ::3000

**Problema:** Porta 3000 já está em uso

### Solução (Windows):

```
netstat -ano | findstr :3000  
taskkill /PID <PID> /F
```

## Erro: 404 em requisições da API

**Problema:** Endpoints não encontrados

### Solução:

- Verificar URL exata do endpoint
- Confirmar método HTTP (GET/POST)
- Validar parâmetros e query strings
- Consultar lista completa em "Endpoints da API"

## Erro: 400 "Corpo da requisição inválido"

**Problema:** Dados enviados não passaram na validação

### Solução:

- Verificar se `Content-Type` é `application/json`
- Validar formato JSON do body
- Conferir campos obrigatórios

- Ler mensagem de erro específica

## Erro: CORS blocked

**Problema:** Frontend/Mobile bloqueado por CORS

**Solução:**

- Adicionar porta do cliente em CORS whitelist no index.js
- Confirmar que Origin é http://localhost:XXXX
- Verificar credenciais: true em CORS options

## Erro: "Categorias inválidas"

**Problema:** Categoria enviada não existe no CSV

**Solução:**

- Chamar GET /api/users/categories para ver categorias válidas
- Usar apenas categorias da lista retornada
- Verificar case-sensitive (Action ≠ action)

---

 Documentação Completa - Backend Gateway v2.0

Sistema de Recomendação de Games com Machine Learning

Última atualização: 14 de Novembro de 2025

© 2025 DSM-P6-G05 - Todos os direitos reservados