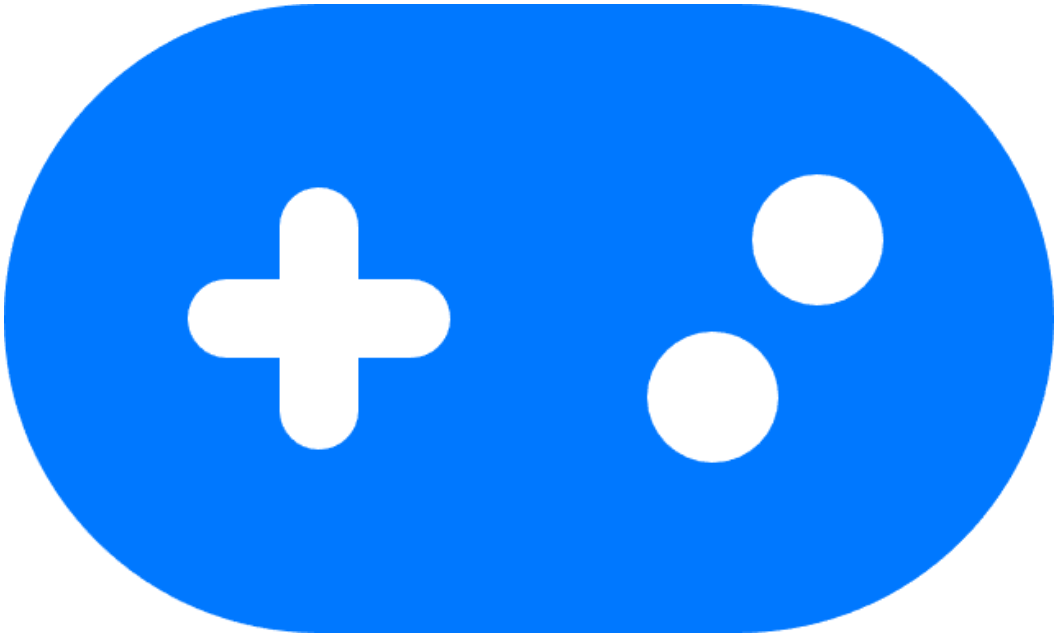


ARCADE GAMES

PMUD’s final project

Alex Matilla Santos
2025 - 26
Universitat Politècnica de Catalunya

Índex

1. Introducció

1.1. Presentació del Projecte

Arcade Games és una aplicació web que ofereix una col·lecció de **quatre jocs clàssics** amb funcionalitats de competència en línia. El projecte combina una interfície d'usuari intuïtiva amb un sistema de backend robust que permet als usuaris **registrar-se, jugar i competir** per les primeres posicions en els rànquings globals.

1.2. Tecnologies Utilitzades

Frontend

- **HTML5**: Estructura semàntica de les pàgines.
- **CSS3**: Estilització amb suport per a tema clar/fosc i disseny responsiu.
- **JavaScript / jQuery**: Lògica de client i manipulació del DOM.
- **AJAX**: Comunicació asíncrona amb l'API.

Backend

- **Node.js**: Entorn d'execució JavaScript al servidor.
- **Express.js**: Framework web per al desenvolupament de l'API REST.
- **SQLite3**: Sistema de gestió de base de dades lleuger.
- **JWT (JSON Web Tokens)**: Autenticació segura sense estat.
- **bcryptjs**: Encriptació segura de contrasenyes.

Eines de Desenvolupament

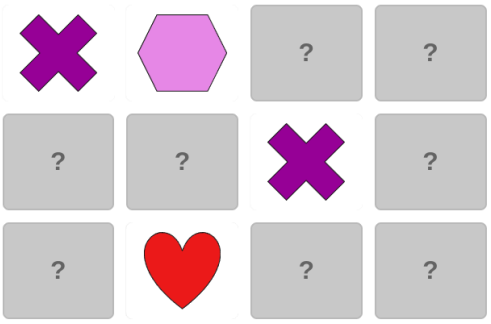
- **Git**: Control de versions.
- **Python HTTP Server**: Servidor de desenvolupament per al frontend.
- **Nodemon**: Reinici automàtic del servidor durant el desenvolupament.

2. Característiques Principals

2.1. Catàleg de Jocs

La plataforma ofereix **quatre jocs clàssics**, cadascun dissenyat per proporcionar una experiència única i atractiva:

- **Memory Match**: posa a prova la memòria dels jugadors mitjançant cartes que s'han d'emparellar en el menor nombre de moviments i temps possible.



- **Minesweeper**: desafia el raonament lògic, obligant el jugador a descobrir totes les caselles segures sense activar cap mina.

- **Sliding Puzzle**: requereix planificació i paciència per ordenar correctament les fitxes numèriques mitjançant moviments estratègics.

- **2048**: combina estratègia i càlcul matemàtic per fusionar fitxes numèriques fins a assolir el valor més alt possible.

Aquesta varietat de jocs permet oferir diferents tipus de reptes, mantenint l'interès i la motivació dels usuaris.

2.2. Sistema d'Autenticació JWT

La seguretat dels usuaris està garantida mitjançant **JSON Web Tokens (JWT)**. Quan un usuari es registra o inicia sessió, el servidor genera un token únic signat digitalment que conté la seva informació d'identificació.

Aquest token:

- S'envia al client després de l'autenticació.
- S'emmagatzema localment al navegador.
- S'inclou automàticament a totes les peticions futures a l'API mitjançant el header **Authorization**.

Aquest mecanisme assegura que **només els usuaris autenticats** puguin guardar puntuacions, accedir als rànquings i utilitzar les funcionalitats avançades de la plataforma.

Codi d'autenticació JWT

```javascript
Client → POST login/register → Server valida → Genera JWT → Client guarda token
Client (token) → Request autenticada → Server verifica JWT → Permet accés
```

2.3. Sistema de Puntuacions i Leaderboards

Cada joc disposa d'un **algorisme de puntuació específic**, adaptat a les seves mecàniques i dificultat. Els factors principals que es tenen en compte són:

- Temps de finalització
- Nombre de moviments
- Nivell de dificultat
- Eficiència del jugador

Codi de càlcul de puntuacions

Memory Match

```
base - (moves × 10) - (time × 2) × multiplicador_dificultat
```

Minesweeper

```
(1000 - time × 10) × multiplicador_dificultat + bonus_mines
```

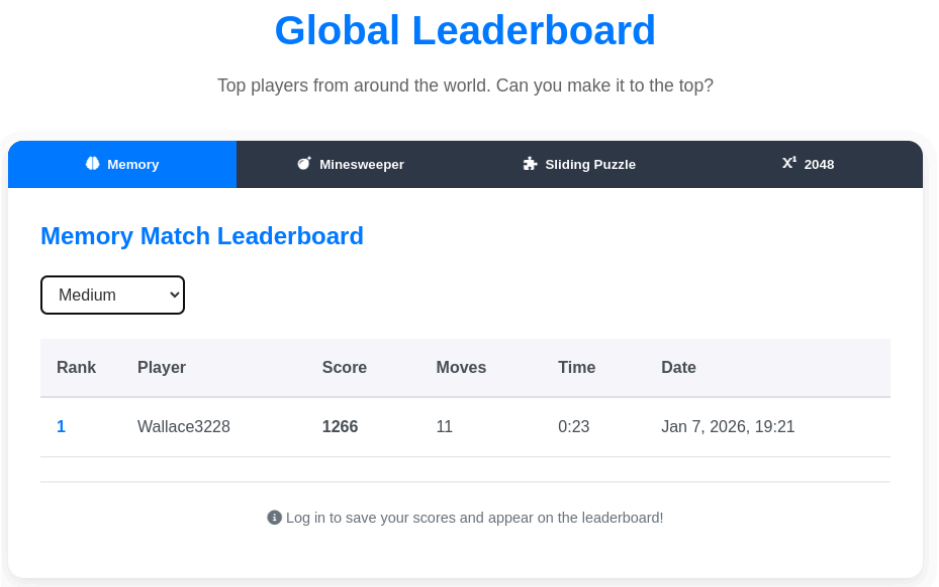
Sliding Puzzle

```
(1000 - moves × 5 - time × 3) × multiplicador_dificultat
```

2048

```
puntuació_final + bonus_tile + eficiència - penalització_temps
```

Les puntuacions es guarden automàticament quan un jugador autenticat completa una partida. Els **leaderboards** mostren les millors puntuacions globals, amb possibilitat de filtrar per joc i dificultat. A més, cada usuari pot consultar el seu **progrés personal** mitjançant estadístiques detallades.



3. Arquitectura del Sistema

3.1. Visió General Client-Servidor

L'arquitectura del sistema segueix el patró clàssic de **client-servidor**, amb una separació clara entre la interfície d'usuari i la lògica de negoci. El **frontend** s'executa al navegador de l'usuari i es comunica amb el **backend** mitjançant una **API REST** a través de peticions HTTP/HTTPS.

El sistema utilitza **SQLite** com a base de dades, evitant la necessitat d'un servidor de base de dades separat, i simplificant el desplegament sense renunciar a les funcionalitats d'una base de dades relacional.

3.2. Frontend

El frontend està construït amb tecnologies web estàndard i s'organitza en **components modulars**, separant clarament estructura, estil i lògica.

Estructura de fitxers

```
None
frontend/
├── css/
│   ├── common.css          # Estils compartits
│   ├── games.css           # Estils de la pàgina principal
│   ├── login.css           # Estils del formulari de login
│   ├── memory_game.css     # Estils específics del Memory
│   ├── buscaminas.css      # Estils del Minesweeper
│   ├── sliding_puzzle.css  # Estils del Sliding Puzzle
│   └── game_2048.css       # Estils del 2048
├── js/
│   ├── common.js           # Lògica compartida (autenticació, tema)
│   ├── games.js            # Lògica de la pàgina principal
│   ├── login.js            # Lògica del formulari de login
│   ├── memory_game_jquery.js
│   ├── buscaminas_jquery.js
│   ├── sliding_puzzle_jquery.js
│   └── game_2048_jquery.js
├── games/
│   ├── memory.html
│   ├── minesweeper.html
│   ├── sliding_puzzle.html
│   └── game_2048.html
└── svg/
    └── icons.svg           # Imatges vectorials utilitzades al Memory
```

Components principals del Frontend

1. **Gestió d'Autenticació**
   Controla el registre, inici de sessió i emmagatzematge del token JWT.
2. **Sistema de Tema Clar/Fosc**
   Permet a l'usuari canviar l'aparença de la interfície segons les seves preferències.
3. **Comunicació amb l'API**
   Capa d'abstracció per gestionar les peticions AJAX al backend.
4. **Gestió d'Estat**
   Manté la informació de l'usuari autenticat i les preferències de configuració.

L'ús de **jQuery** simplifica la manipulació del DOM i la gestió de peticions AJAX, oferint una experiència d'usuari fluida i responsiva sense necessitat de recarregar les pàgines.

3.3. Backend

El backend està desenvolupat amb **Node.js** i **Express.js**, proporcionant una **API REST modular i extensible**. Aquesta arquitectura facilita l'organització del codi i el manteniment del sistema.

Estructura de fitxers

```
None
backend/
├── server.js             # Punt d'entrada del servidor
├── package.json          # Dependències i scripts
├── .env                  # Variables d'entorn
├── database.sqlite       # Base de dades SQLite
├── config/
│   └── database.js       # Configuració i inicialització de la BD
├── routes/
│   ├── auth.js           # Rutes d'autenticació
│   └── scores.js         # Rutes de puntuacions
├── models/
│   ├── User.js           # Model d'usuari
│   └── Score.js          # Model de puntuació
└── middleware/
    └── auth.js           # Middleware d'autenticació JWT
```

3.3.1. Estructura de l'API

L'API segueix les **millors pràctiques REST**, assegurant claredat i coherència:

- **Recursos:** /auth, /scores
- **Verbs HTTP**: GET, POST
- **Codis d'estat**: 200, 201, 400, 401, 404, 500
- **Respostes estandarditzades en format JSON**

Flux de processament de les peticions

```
None
Petició HTTP → Middleware (CORS, logging)
→ Validació de dades
→ Autenticació (si és necessària)
→ Controlador
→ Model (Base de dades)
→ Resposta JSON
→ Client
```

Aquesta cadena de processament garanteix seguretat, consistència i facilitat de depuració.

3.4. Base de Dades SQLite

La base de dades utilitza **SQLite** per la seva simplicitat, portabilitat i baix consum de recursos. L'esquema està format per dues taules principals:

Taula d'usuaris

```sql
SQL
CREATE TABLE IF NOT EXISTS users (
    id            INTEGER PRIMARY KEY AUTOINCREMENT,
    username      TEXT UNIQUE NOT NULL,
    email         TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    created_at    DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Taula de puntuacions

```sql
SQL
CREATE TABLE IF NOT EXISTS scores (
    id           INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id      INTEGER NOT NULL,
    game         TEXT NOT NULL,
    difficulty   TEXT,
    score        INTEGER NOT NULL,
    moves        INTEGER,
    time_seconds INTEGER,
    played_at     DATETIME DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (user_id) REFERENCES
users(id)
);
```

Índexs per millorar el rendiment

```sql
CREATE INDEX IF NOT EXISTS idx_scores_game
ON scores(game);

CREATE INDEX IF NOT EXISTS idx_scores_user_game
ON scores(user_id, game);

CREATE INDEX IF NOT EXISTS idx_scores_top
ON scores(game, difficulty, score DESC);
```

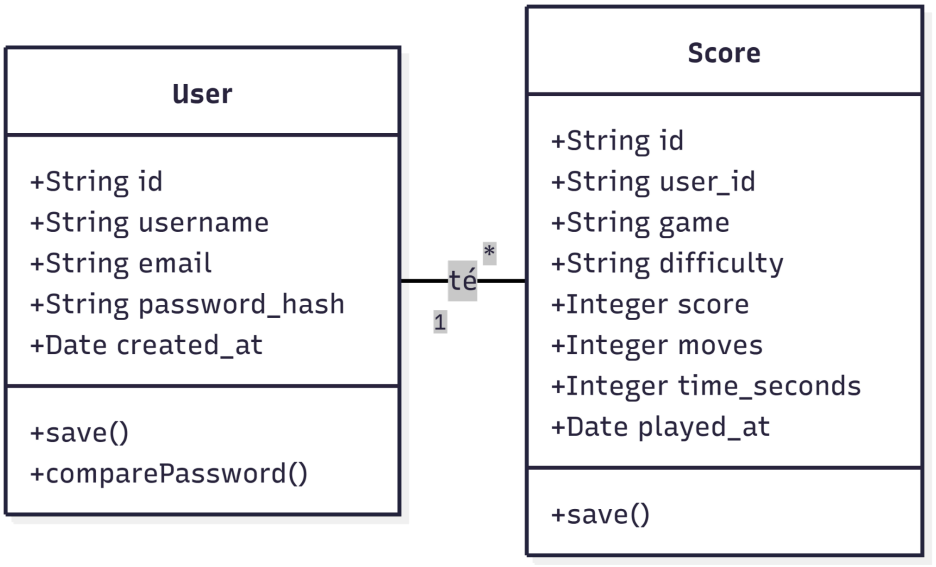
La base de dades s'inicialitza automàticament en iniciar el servidor, creant les taules i índexs necessaris si no existeixen. Aquesta estratègia **simplifica el desplegament** i redueix la configuració manual, mantenint un bon rendiment en consultes de classificacions i estadístiques.

4. Diagrames del Sistema

Els diagrames del sistema permeten visualitzar de manera clara tant l'estructura interna de les dades com l'organització general dels mòduls que formen la plataforma. Aquests diagrames ajuden a comprendre el funcionament global de l'aplicació i la relació entre els seus components principals.

4.1. Diagrama de Classes UML

El **diagrama de classes UML** representa l'estructura del model de dades utilitzat al backend de l'aplicació. En aquest projecte, el sistema es basa principalment en dues entitats fonamentals: **User** i **Score**.



Classe User

La classe **User** representa els usuaris registrats a la plataforma. Conté els següents atributs:

- **id**: identificador únic de l'usuari.
- **username**: nom d'usuari únic.
- **email**: correu electrònic de l'usuari.
- **password_hash**: contrasenya xifrada mitjançant bcrypt.
- **created_at**: data de creació del compte.

Pel que fa als mètodes, la classe inclou:

- **save()**: permet guardar o actualitzar la informació de l'usuari a la base de dades.
- **comparePassword()**: compara una contrasenya introduïda amb el hash emmagatzemat, utilitzat durant el procés d'autenticació.

Aquesta classe és clau per al sistema d'autenticació basat en JWT i per a la gestió de la informació personal dels jugadors.

Classe Score

La classe **Score** representa les puntuacions obtingudes pels usuaris als diferents jocs. Els seus atributs principals són:

- **id**: identificador únic de la puntuació.
- **user_id**: referència a l'usuari que ha obtingut la puntuació.
- **game**: joc al qual pertany la puntuació.
- **difficulty**: nivell de dificultat de la partida.
- **score**: valor numèric de la puntuació final.
- **moves**: nombre de moviments realitzats.
- **time_seconds**: temps total de la partida.
- **played_at**: data i hora en què s'ha jugat la partida.

El mètode principal d'aquesta classe és:

- **save()**: emmagatzema la puntuació a la base de dades.

Relació entre classes

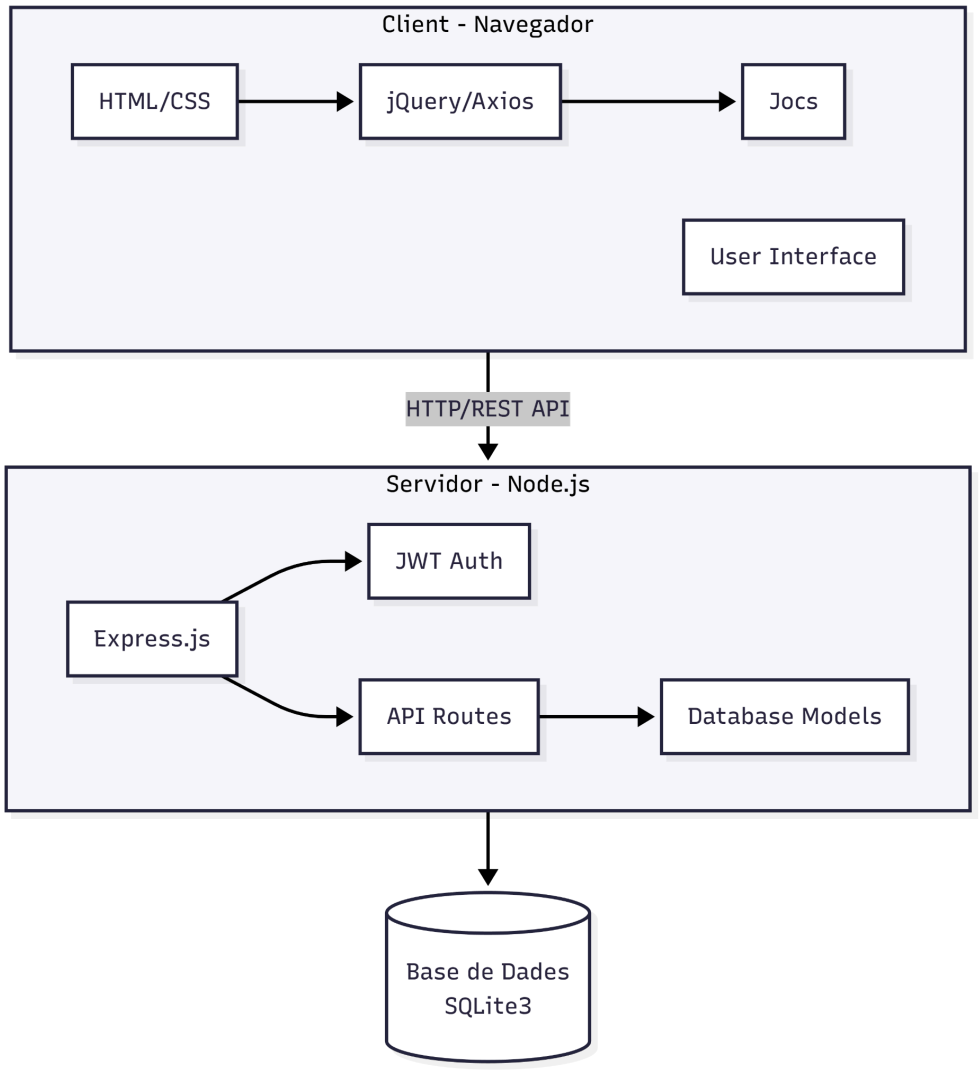
Entre **User** i **Score** existeix una relació **1 a ***, ja que:

- Un usuari pot tenir múltiples puntuacions.
- Cada puntuació pertany únicament a un usuari.

Aquesta relació reflecteix fidelment el funcionament del sistema de leaderboards i permet consultar tant les millors puntuacions globals com l'historial individual de cada jugador.

4.2. Diagrama de Mòduls

El **diagrama de mòduls** mostra l'arquitectura general del sistema seguint el model **client-servidor**, destacant les tecnologies utilitzades i el flux de comunicació entre components.



Client – Navegador

La part client s'executa al navegador de l'usuari i està formada per:

- **HTML / CSS**: definició de l'estructura i l'estil de la interfície.
- **jQuery / Axios**: gestió de la lògica de client i comunicació amb el backend mitjançant peticions HTTP.
- **Quatre jocs**: implementats com a mòduls independents però integrats dins la mateixa plataforma.
- **Interfície d'usuari**: proporciona una experiència visual coherent i interactiva.

El client és responsable de capturar les accions de l'usuari, mostrar els resultats i enviar les dades necessàries al servidor.

Comunicació HTTP / REST API

La comunicació entre client i servidor es realitza mitjançant una **API REST**, utilitzant peticions HTTP segures. Les dades s'intercanvien en format JSON, i les peticions protegides requereixen un token JWT vàlid.

Servidor – Node.js

El servidor està implementat amb **Node.js** i **Express.js**, i inclou els següents mòduls principals:

- **Express.js**: gestiona el servidor web i el routing.
- **JWT Auth**: sistema d'autenticació que valida les peticions dels usuaris.
- **API Routes**: rutes REST per a l'autenticació i la gestió de puntuacions.
- **Database Models**: models que encapsulen l'accés a la base de dades.

Aquest disseny modular permet separar clarament les responsabilitats i facilita l'escalabilitat i el manteniment del codi.

Base de Dades

Finalment, el servidor es connecta a la **base de dades SQLite**, que emmagatzema la informació dels usuaris i les puntuacions, garantint persistència i consistència de les dades.

5. Documentació de l'API (Endpoints i verbs)

5.1. Endpoints del sistema

GET: /

Shell

```
# Descripció: Mostra informació sobre l'API i els seus endpoints disponibles.
# Paràmetres: Cap
# CURL
curl -X 'GET' \
  'http://localhost:3000/' \
  -H 'accept: application/json'

# RESPOSTA --> 200 (OK)
{
  "message": "🎮 Welcome to Arcade Games API",
  "documentation": "Check /api/health for API status",
  "endpoints": {
    "auth": {
      "register": "POST /api/auth/register",
      "login": "POST /api/auth/login",
      "logout": "POST /api/auth/logout"
    },
    "scores": {
      "save": "POST /api/scores",
      "top": "GET /api/scores/top/:game",
      "user": "GET /api/scores/user/me",
      "stats": "GET /api/scores/stats",
      "games": "GET /api/scores/games"
    }
}
```

GET: /api/health

Shell

```
# Descripció: Verifica l'estat actual de l'API.
# Paràmetres: Cap
# CURL
curl -X 'GET' \
  'http://localhost:3000/api/health' \
  -H 'accept: application/json'

# RESPOSTA --> 200 (OK)
{
  "status": "OK",
  "message": "Arcade Games API is running",
  "timestamp": "2026-01-07T22:09:33.177Z",
  "version": "1.0.0"
}
```

5.2. Endpoints d'Autenticació

POST: /api/auth/register

Shell

```
# Descripció: Crea un nou compte d'usuari i retorna un token JWT.

# Paràmetres: Cap

# Body
{
  "username": "AlexMatilla",
  "email": "alexmatilla@example.com",
  "password": "password123"
}

# CURL
curl -X 'POST' \
  'http://localhost:3000/api/auth/register' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "username": "AlexMatilla",
    "email": "alexmatilla@example.com",
    "password": "password123"
  }'

# RESPOSTA --> 201 (CREATED)
{
  "success": true,
  "message": "Usuari registrat exitosament",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": 33,
    "username": "AlexMatilla",
    "email": "alexmatilla@example.com",
    "created_at": "2026-01-07T16:51:17.682Z"
  }
}

# Errors:
# 400: Error en la sol·licitud
# 500: Error intern del servidor
```

POST: /api/auth/login

Shell

```
# Descripció: Autentica un usuari existent i retorna un token JWT.

# Paràmetres: Cap

# Body
{
  "username": "AlexMatilla",
  "password": "password123"
}

# CURL
curl -X 'POST' \
  'http://localhost:3000/api/auth/login' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "username": "AlexMatilla",
    "password": "password123"
  }'

# RESPOSTA --> 200 (OK)
{
  "success": true,
  "message": "Login exitós",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": 33,
    "username": "AlexMatilla",
    "email": "alexmatilla@example.com",
    "created_at": "2026-01-07T16:51:17.682Z"
  }
}


# Errors:
# 401: Credencials invàlides
# 500: Error intern del servidor
```

POST: /api/auth/logout

```shell
# Descripció: Tanca la sessió de l'usuari (el client ha d'eliminar el token emmagatzemat).

# Paràmetres: Cap

# Headers
  - Authorization: Bearer <token>

# CURL
curl -X 'POST' \
  'http://localhost:3000/api/auth/logout' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer <token>'
# RESPOSTA --> 200 (OK)
{
  "success": true,
  "message": "Logout exitós. Esborra el token al client."
}

# Errors:
# 401: No autoritzat
```

GET: /api/auth/registered-players

```shell
# Descripció: Retorna el nombre total d'usuaris registrats.

# Paràmetres: Cap

# Headers
Authorization: Bearer <token>

# CURL
curl -X 'GET' \
  'http://localhost:3000/api/auth/registered-players' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer <token>'

# RESPOSTA --> 200 (OK)
{
  "success": true,
  "count": 6
}

# Errors:
# 401: No autoritzat
# 500: Error intern del servidor
```

5.3. Endpoints de Puntuacions

POST: /api/scores

Shell

```
# Descripció: Guarda el resultat d'un joc per a l'usuari autenticat.
# Paràmetres: Cap

# Headers
  - Authorization: Bearer <token>

# Body
{
  "game": "memory",
  "difficulty": "medium",
  "score": 1500,
  "moves": 25,
  "time_seconds": 120
}

# CURL
curl -X 'POST' \
  'http://localhost:3000/api/scores' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer <token>' \
  -d '{
    "game": "memory",
    "difficulty": "medium",
    "score": 1500,
    "moves": 25,
    "time_seconds": 120
  }'

# RESPOSTA --> 201 (CREATED)
{
  "success": true,
  "message": "Puntuació guardada exitosament",
  "score": {
    "id": 42,
    "user_id": 1,
    "game": "memory",
    "difficulty": "medium",
    "score": 1500,
    "moves": 25,
    "time_seconds": 120,
    "played_at": "2024-01-15T10:30:00Z"
  }
}

# Errors:
# 400: Dades invàlides
# 401: No autoritzat
# 500: Error intern del servidor
```

GET: /api/scores/top/{game}

```shell
# Descripció: Retorna les millors puntuacions per a un joc específic.

# Paràmetres URL
  - game = <identificador del joc>: [memory, minesweeper, sliding_puzzle, 2048]

# Paràmetres Query (opcionals)
  - difficulty = <dificultat>: [easy, medium, hard]
  - limit      = <nombre màxim de resultats>

# CURL
curl -X 'GET' \
  'http://localhost:3000/api/scores/top/memory?difficulty=medium&limit=5' \
  -H 'accept: application/json'

# RESPOSTA --> 200 (OK)
{
  "success": true,
  "game": "memory",
  "difficulty": "medium",
  "scores": [
    {
      "rank": 1,
      "username": "AlexMatilla",
      "score": 1500,
      "moves": 25,
      "time_seconds": 120,
      "played_at": "2024-01-15T10:30:00Z",
      "is_current_user": true
    }
  ],
  "count": 5
}

# Errors:
# 400: Joc invàlid
# 500: Error intern del servidor
```

GET: /api/scores/user/me

Shell

```
# Descripció: Retorna les puntuacions de l'usuari autenticat.

# Paràmetres: Cap

# Headers
  - Authorization: Bearer <token>

# Paràmetres Query (opcionals)
  - game = <identificador del joc>: [memory, minesweeper, sliding_puzzle, 2048]
  - limit = <nombre màxim de resultats>

# CURL
curl -X 'GET' \
  'http://localhost:3000/api/scores/user/me?game=memory&limit=10' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer <token>'

# RESPOSTA --> 200 (OK)
{
  "success": true,
  "user": {
    "id": 1,
    "username": "AlexMatilla"
  },
  "scores": [
    {
      "id": 42,
      "user_id": 1,
      "game": "memory",
      "difficulty": "medium",
      "score": 1500,
      "moves": 25,
      "time_seconds": 120,
      "played_at": "2024-01-15T10:30:00Z"
    }
  ],
  "best_scores": [
    {
      "game": "memory",
      "best_score": 1500
    }
  ],
  "total_games": 15
}

# Errors:
# 401: No autoritzat
# 500: Error intern del servidor
```

GET: /api/scores/stats

Shell

```
# Descripció: Retorna estadístiques generals de tots els jocs

# Paràmetres: Cap

# CURL
curl -X 'GET' \
  'http://localhost:3000/api/scores/stats' \
  -H 'accept: application/json'

# RESPOSTA --> 200 (OK)
{
  "success": true,
  "stats": {
    "byGame": [
      {
        "game": "memory",
        "total_games": 100,
        "unique_players": 25,
        "highest_score": 2000,
        "average_score": 1250.5
      }
    ],
    "recent_games": [
      {
        "game": "memory",
        "username": "AlexMatilla",
        "score": 1500,
        "played_at": "2024-01-15T10:30:00Z"
      }
    ]
  },
  "generated_at": "2024-01-15T10:30:00Z"
}

# Errors:
# 500: Error intern del servidor
```

GET: /api/scores/games

Shell

```
# Descripció: Retorna la lista de juegos soportados con sus características.

# Paràmetres: Cap

# CURL
curl -X 'GET' \
  'http://localhost:3000/api/scores/games' \
  -H 'accept: application/json'

# RESPOSTA --> 200 (OK)
{
  "success": true,
  "games": [
    {
      "id": "memory",
      "name": "Memory Match",
      "description": "Test your memory with card matching",
      "difficulties": ["novice", "standard", "expert"],
      "metrics": ["moves", "time"],
      "icon": "brain"
    },
    {
      "id": "minesweeper",
      "name": "Minesweeper",
      "description": "Find mines without detonating them",
      "difficulties": ["easy", "medium", "hard"],
      "metrics": ["time", "flags"],
      "icon": "bomb"
    },
    {
      "id": "sliding_puzzle",
      "name": "Sliding Puzzle",
      "description": "Slide tiles to arrange them in order",
      "difficulties": ["3x3", "4x4", "5x5"],
      "metrics": ["moves", "time"],
      "icon": "puzzle-piece"
    },
    {
      "id": "2048",
      "name": "2048",
      "description": "Join numbers to get to the 2048 tile",
      "difficulties": ["standard"],
      "metrics": ["score", "moves"],
      "icon": "superscript"
    }
  ],
  "count": 4
}
```

6. Conclusions

Finalment, podem concloure que el projecte presenta una implementació completa i satisfactòria d'una aplicació multiplataforma distribuida. El sistema demostra una arquitectura ben definida que combina una interfície d'usuari intuïtiva amb un backend robust, oferint als usuaris una experiència d'entreteniment cohesionada, estable i tècnicament sòlida.

S'han assolit amb èxit tots els objectius funcionals plantejats inicialment. Entre aquests destaquen la implementació dels quatre jocs amb mecàniques diferenciades, el sistema d'autenticació basat en JSON Web Tokens (JWT) i el sistema de classificacions que permet la competició global entre usuaris. L'arquitectura modular adoptada, amb una clara separació entre frontend i backend, contribueix a facilitar tant el manteniment del sistema com la seva possible escalabilitat futura.

Les decisions tecnològiques preses —Node.js i Express per al desenvolupament del backend, SQLite com a sistema de gestió de bases de dades i jQuery per a la lògica del client— han resultat adequades per als requisits del projecte. L'API REST desenvolupada segueix les bones pràctiques del sector, amb una estructura clara, endpoints ben definits i una gestió correcta de la seguretat, la qual cosa proporciona una base sòlida per a futures ampliacions o integracions amb altres sistemes.

Des d'un punt de vista formatiu, el desenvolupament d'aquest projecte ha permès consolidar coneixements fonamentals en diverses àrees del desenvolupament web modern, com ara la programació full-stack, la gestió d'autenticació i autorització, el disseny d'APIs RESTful i la integració entre client i servidor.

En conclusió, podem afirmar que el projecte **Arcade Games** ha resultat satisfactori, i compleix correctament els requisits funcionals plantejats.