# Manual Técnico - Backend de los Juegos de Química

## Índice



---

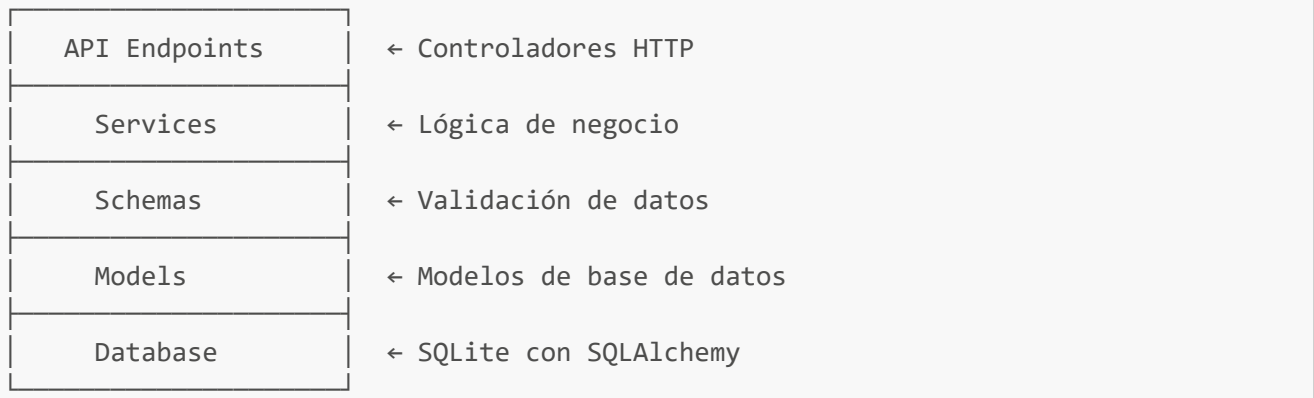
## Descripción General

El Backend de los Juegos de Química es una API REST desarrollada con FastAPI que proporciona servicios educativos interactivos para el aprendizaje de química. El sistema incluye múltiples juegos y herramientas educativas:

- **Sistema de Quiz**: Preguntas de opción múltiple sobre química con generación por IA
- **Tabla Periódica Interactiva**: Información completa de elementos químicos con audio
- **Juego de Lotería**: Juego multijugador en tiempo real usando WebSockets
- **Sistema de Ranking**: Seguimiento de puntuaciones y clasificaciones
- **Sistema de Autenticación**: Registro y login de usuarios con roles
- **Recuperación de Contraseñas**: Sistema de email para recuperación de credenciales

## Arquitectura del Sistema

El proyecto sigue una arquitectura en capas con separación clara de responsabilidades:



Principios de Arquitectura:

- **Separación de responsabilidades**: Cada capa tiene una función específica
- **Inyección de dependencias**: Uso de FastAPI Depends para gestión de dependencias
- **Validación de datos**: Pydantic para validación automática de entrada y salida
- **ORM**: SQLAlchemy para abstracción de base de datos

## Tecnologías Utilizadas

### Core Framework

- **FastAPI 0.115.12**: Framework web moderno y de alto rendimiento
- **Python 3.13**: Lenguaje de programación principal
- **SQLAlchemy**: ORM para manejo de base de datos
- **SQLite**: Base de datos embebida

### Integraciones y Servicios

- **Google Generative AI (Gemini 2.0)**: Generación de preguntas con IA
- **FastAPI Mail**: Envío de correos electrónicos
- **WebSockets**: Comunicación en tiempo real para juegos multijugador
- **Pydantic**: Validación y serialización de datos
- **Passlib + Bcrypt**: Cifrado de contraseñas

### Utilidades y Herramientas

- **CORS Middleware**: Manejo de peticiones cross-origin
- **JSON**: Almacenamiento de datos estáticos (elementos, quizzes)
- **MP3 Audio Files**: Pronunciación de elementos químicos

## Estructura del Proyecto

```
quimic-app-api/
├── app/
│   ├── __init__.py                # Inicialización del paquete
│   ├── main.py                    # Punto de entrada de la aplicación
│   ├── api/                       # Definición de APIs
│   │   └── v1/
│   │       └── endpoints/         # Endpoints organizados por módulo
│   │           ├── auth/          # Autenticación y registro
│   │           ├── games/         # Juegos (quiz, lotería, ranking)
│   │           └── landing/       # Páginas informativas
│   ├── core/                      # Configuración del núcleo
│   │   ├── config.py             # Configuración de la aplicación
│   │   ├── database.py           # Configuración de base de datos
│   │   ├── mail.py               # Configuración de email
│   │   ├── security.py           # Utilidades de seguridad
│   │   └── templates/            # Plantillas de email
│   ├── models/                    # Modelos de base de datos
```

```
│   │   ├── quiz.py                # Modelo de preguntas
│   │   ├── ranking.py             # Modelo de puntuaciones
│   │   └── user.py                # Modelo de usuarios
│   ├── schemas/                    # Esquemas de validación Pydantic
│   │   ├── auth_schema.py         # Esquemas de autenticación
│   │   ├── lottery_schema.py      # Esquemas del juego de lotería
│   │   ├── periodic_table_schema.py # Esquemas de tabla periódica
│   │   ├── quiz_schema.py         # Esquemas de quiz
│   │   ├── ranking_schema.py      # Esquemas de ranking
│   │   └── user_schema.py         # Esquemas de usuario
│   ├── services/                   # Lógica de negocio
│   │   ├── audio_service.py       # Servicio de audio
│   │   ├── auth_service.py        # Servicio de autenticación
│   │   ├── gemeni_service.py      # Integración con Gemini AI
│   │   ├── periodic_table_service.py # Servicio de tabla periódica
│   │   ├── quiz_service.py        # Servicio de quiz
│   │   ├── ranking_service.py     # Servicio de ranking
│   │   ├── user_service.py        # Servicio de usuarios
│   │   └── lottery_service/       # Servicio de lotería
│   │       ├── managers/          # Gestores del juego
│   │       ├── room.py            # Gestión de salas
│   │       └── state.py           # Estado del juego
│   ├── statics/                    # Archivos estáticos
│   │   ├── audio/                 # Archivos de audio MP3
│   │   └── json/                  # Datos en formato JSON
│   └── utils/                      # Utilidades generales
├── data/
│   └── quimic.sqlite              # Base de datos SQLite
├── requirements.txt               # Dependencias de Python
├── README.md                      # Documentación básica
└── doc.md                        # Documentación adicional
```

# Módulos y Componentes

## 1. Módulo de Autenticación (`auth`)

**Propósito**: Gestión de usuarios, registro, login y recuperación de contraseñas.

**Endpoints**:

- `POST /api/v1/register`: Registro de nuevos usuarios
- `POST /api/v1/login`: Autenticación de usuarios
- `POST /api/v1/recover`: Recuperación de contraseñas

**Características**:

- Cifrado de contraseñas con bcrypt
- Validación de email
- Envío de correos de bienvenida y recuperación
- Roles de usuario (alumno/profesor)

## 2. Módulo de Quiz (`games/quiz`)

**Propósito**: Sistema de preguntas de opción múltiple sobre química con soporte de IA.

**Endpoints**:

- `GET /api/v1/quizzes/random/{count}`: Obtener preguntas aleatorias
- `GET /api/v1/quizzes/ai/{count}`: Generar preguntas con IA
- `GET /api/v1/quizzes/`: Obtener todas las preguntas
- `POST /api/v1/quizzes/`: Crear nueva pregunta

**Características**:

- Generación automática con Gemini AI
- Preguntas predefinidas desde JSON
- Diferentes niveles de dificultad
- Validación de respuestas

### 3. Módulo de Tabla Periódica (`games/periodic-table`)

**Propósito**: Información interactiva de elementos químicos con audio.

**Endpoints**:

- `GET /api/v1/periodic-table/elements`: Obtener todos los elementos
- `GET /api/v1/periodic-table/elements/{number}/stream`: Audio del elemento

**Características**:

- 118 elementos químicos completos
- Propiedades físicas y químicas
- Pronunciación en audio MP3
- Datos curiosos e información educativa

### 4. Módulo de Lotería (`games/lottery`)

**Propósito**: Juego multijugador en tiempo real basado en elementos químicos.

**Endpoints WebSocket**:

- `/api/v1/lottery/ws/{room_id}`: Conexión WebSocket para juego en tiempo real

**Características**:

- Salas de juego multijugador
- Cartones con símbolos químicos
- Comunicación en tiempo real
- Estado persistente de juego

### 5. Módulo de Ranking (`games/ranking`)

**Propósito**: Sistema de puntuaciones y clasificaciones por juego.

**Endpoints**:

- `POST /api/v1/ranking/create`: Crear nueva puntuación
- `GET /api/v1/ranking/list/{name}`: Obtener ranking por juego
- `PUT /api/v1/ranking/update/{user}/{name}`: Actualizar puntuación

**Características**:

- Rankings por juego específico
- Máxima puntuación por usuario
- Ordenamiento automático

### 6. Módulo de Landing (`landing`)

**Propósito**: Contenido informativo y educativo sobre química.

**Endpoints**:

- `GET /api/v1/landing/periodic-table`: Información de tabla periódica
- `GET /api/v1/landing/history`: Historia de la química
- `GET /api/v1/landing/timeline`: Línea de tiempo química

## Modelos de Datos

### User Model (`models/user.py`)

```
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String(255), unique=True, index=True)
    full_name = Column(String(255), index=True)
    hashed_password = Column(String(255))
    is_active = Column(Boolean, default=True)
    role = Column(Enum(UserRole), default=UserRole.alumno)
```

**Propósito**: Almacena información de usuarios registrados con roles diferenciados.

### Quiz Model (`models/quiz.py`)

```
class Quiz(Base):
    __tablename__ = "quizzes"

    id = Column(Integer, primary_key=True, index=True)
    question = Column(String, nullable=False)
    options = Column(JSON, nullable=False)
    answer = Column(String(255), nullable=False)
    difficulty = Column(String(50), default="medium")
    created_by = Column(String(50), default="system")
```

**Propósito**: Almacena preguntas de quiz con opciones múltiples y metadatos.

### Ranking Model (`models/ranking.py`)

```
class Ranking(Base):
    __tablename__ = "rankings"

    id = Column(Integer, primary_key=True, index=True)
    user = Column(Integer, nullable=False)
    score = Column(Integer, nullable=False)
    name = Column(String(100), nullable=False)
    username = Column(String(100), nullable=False)
```

**Propósito**: Registra puntuaciones de usuarios en diferentes juegos.

## Esquemas de Validación

Los esquemas Pydantic proporcionan validación automática y serialización:

### ElementSchema (`schemas/periodic_table_schema.py`)

- Validación de propiedades químicas y físicas
- Conversión automática de tipos de datos
- Mapeo desde JSON a objetos Python

### QuizSchema (`schemas/quiz_schema.py`)

- Validación de preguntas y opciones
- Esquemas separados para creación y lectura
- Validación de respuestas correctas

### UserSchema (`schemas/user_schema.py`)

- Validación de email con EmailStr
- Esquemas diferenciados por operación (Create, Read, Update)
- Validación de roles de usuario

## Servicios

### Auth Service (`services/auth_service.py`)

**Funciones principales**:

- `authenticate_user()`: Validación de credenciales
- `password_recovery()`: Generación y envío de nueva contraseña
- `send_password_recovery_email()`: Envío de email de recuperación

### Quiz Service (`services/quiz_service.py`)

**Funciones principales**:

- `get_random_quizzes()`: Obtiene preguntas aleatorias de BD
- `get_ai_quizzes()`: Genera preguntas nuevas con IA
- `generate_ai_quizzes_with_gemini()`: Integración con Gemini AI
- `load_initial_quizzes()`: Carga preguntas desde JSON

### Periodic Table Service (`services/periodic_table_service.py`)

**Funciones principales**:

- `get_all_elements()`: Carga elementos desde JSON
- `get_audio_path()`: Obtiene ruta de archivo de audio
- `stream_audio_response()`: Streaming de audio con soporte de rangos

### Gemini Service (`services/gemeni_service.py`)

**Propósito**: Integración con Google Generative AI para generación de contenido educativo.

**Configuración**:

- API Key desde variables de entorno
- Modelo Gemini 2.0 Flash
- Prompts especializados en química

### Lottery Service (`services/lottery_service/`)

**Componentes**:

- `managers/game.py`: Lógica del juego de lotería
- `managers/ws.py`: Gestión de WebSockets
- `room.py`: Gestión de salas de juego
- `state.py`: Estado global del juego

## APIs y Endpoints

### Estructura de Rutas

```
/api/v1/
├── auth/
│   ├── POST /register
│   ├── POST /login
│   └── POST /recover
├── games/
│   ├── quizzes/
│   │   ├── GET /random/{count}
│   │   ├── GET /ai/{count}
│   │   ├── GET /
│   │   └── POST /
│   ├── periodic-table/
│   │   ├── GET /elements
│   │   └── GET /elements/{number}/stream
│   ├── ranking/
```

```
│   │   ├── POST /create
│   │   ├── GET /list/{name}
│   │   └── PUT /update/{user}/{name}
│   └── lottery/
│       └── WS /ws/{room_id}
└── landing/
    ├── GET /periodic-table
    ├── GET /history
    └── GET /timeline
```

### Códigos de Respuesta

- `200 OK`: Operación exitosa
- `201 Created`: Recurso creado exitosamente
- `400 Bad Request`: Datos de entrada inválidos
- `404 Not Found`: Recurso no encontrado
- `500 Internal Server Error`: Error interno del servidor

## Base de Datos

### Configuración

- **Motor**: SQLite embebido
- **ORM**: SQLAlchemy con declarative_base
- **Archivo**: `./data/quimic.sqlite`
- **Pool de conexiones**: Configurado para SQLite

### Migraciones

- Creación automática de tablas al inicio: `Base.metadata.create_all(bind=engine)`
- No requiere migraciones complejas por usar SQLite

### Gestión de Sesiones

```python
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

## Configuración

### Variables de Entorno (`.env`)

```
GEMINI_API_KEY=tu_api_key_de_gemini
EMAIL_HOST=smtp.gmail.com
```

```
EMAIL_PORT=587
EMAIL_FROM=tu_email@gmail.com
EMAIL_PASSWORD=tu_password_de_aplicacion
```

### Settings Class (`core/config.py`)

- Carga automática desde archivo `.env`
- Validación de tipos con Pydantic
- Cache con `@lru_cache` para optimización

### CORS Configuration

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

## Instalación y Configuración Local

### Prerrequisitos

- Python 3.11 o superior
- pip (gestor de paquetes de Python)

### Pasos de instalación

1. **Clonar el repositorio**

   ```
   git clone <repository-url>
   cd quimic-app-api
   ```

2. **Crear entorno virtual**

   ```
   python -m venv venv
   ```

3. **Activar entorno virtual**

   ```
   # Windows
   venv\Scripts\activate
   ```

```
# Linux/Mac
source venv/bin/activate
```

4. **Instalar dependencias**

```
pip install -r requirements.txt
```

5. **Configurar variables de entorno**

```
# Crear archivo .env en la raíz del proyecto
cp .env.example .env
# Editar .env con tus credenciales
```

6. **Ejecutar en desarrollo**

```
fastapi dev app/main.py
```

7. **Acceder a la aplicación**

   - API: `http://localhost:8000`
   - Documentación interactiva: `http://localhost:8000/docs`
   - ReDoc: `http://localhost:8000/redoc`

## Despliegue en Railway

Railway es una plataforma de despliegue moderna que facilita el deployment de aplicaciones Python. Aquí están los pasos completos para desplegar tu API en Railway:

### Paso 1: Preparar el Proyecto para Producción

1. **Crear archivo `railway.json`** en la raíz del proyecto:

```
{
  "$schema": "https://railway.app/railway.schema.json",
  "build": {
    "builder": "NIXPACKS"
  },
  "deploy": {
    "startCommand": "uvicorn app.main:app --host 0.0.0.0 --port $PORT",
    "restartPolicyType": "ON_FAILURE",
    "restartPolicyMaxRetries": 10
  }
}
```

2. **Crear archivo** `Procfile` (opcional, pero recomendado):

```
web: uvicorn app.main:app --host 0.0.0.0 --port $PORT
```

3. **Actualizar** `requirements.txt` para incluir uvicorn si no está:

```
uvicorn[standard]==0.27.1
```

### Paso 2: Configurar Variables de Entorno para Producción

Modifica `app/core/config.py` para manejar la URL de base de datos de Railway:

```python
from pydantic_settings import BaseSettings, SettingsConfigDict
from functools import lru_cache
import os

class Settings(BaseSettings):
    # Configuración existente
    gemini_api_key: str
    email_host: str
    email_port: int = 587
    email_from: str
    email_password: str

    # Nueva configuración para Railway
    database_url: str = "sqlite:///./data/quimic.sqlite"
    port: int = 8000
    environment: str = "development"

    model_config = SettingsConfigDict(env_file=".env")

@lru_cache
def get_settings():
    return Settings()
```

### Paso 3: Actualizar Configuración de Base de Datos

Modifica `app/core/database.py` para usar la configuración dinámica:

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import declarative_base, sessionmaker
from .config import get_settings
import os

settings = get_settings()
```

```
# Para Railway, usar la variable de entorno DATABASE_URL si existe
DATABASE_URL = os.getenv("DATABASE_URL", settings.database_url)

engine = create_engine(
    DATABASE_URL,
    connect_args={"check_same_thread": False} if "sqlite" in DATABASE_URL else {}
)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

### Paso 4: Despliegue en Railway

**Opción A: Desde GitHub (Recomendado)**

1. **Subir código a GitHub**:

```
git init
git add .
git commit -m "Initial commit"
git branch -M main
git remote add origin https://github.com/tu-usuario/quimic-app-api.git
git push -u origin main
```

2. **Conectar con Railway**:
   - Ve a railway.app
   - Regístrate con tu cuenta de GitHub
   - Click en "New Project"
   - Selecciona "Deploy from GitHub repo"
   - Selecciona tu repositorio `quimic-app-api`

**Opción B: Desde Railway CLI**

1. **Instalar Railway CLI**:

```
# Windows (PowerShell)
iwr -useb https://railway.app/install.ps1 | iex

# macOS/Linux
curl -fsSL https://railway.app/install.sh | sh
```

2. **Login y desplegar**:

```
railway login
railway init
railway up
```

### Paso 5: Configurar Variables de Entorno en Railway

En el dashboard de Railway:

1. **Ve a tu proyecto desplegado**
2. **Click en "Variables"**
3. **Agregar las siguientes variables**:

```
GEMINI_API_KEY=tu_api_key_de_gemini
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_FROM=tu_email@gmail.com
EMAIL_PASSWORD=tu_password_de_aplicacion
ENVIRONMENT=production
PORT=8000
```

### Paso 6: Configurar Base de Datos (Opcional)

Railway soporta PostgreSQL de forma nativa:

1. **En tu proyecto de Railway**:
   - Click en "Add Service"
   - Selecciona "PostgreSQL"
   - Railway automáticamente creará la variable `DATABASE_URL`

2. **Actualizar requirements.txt** para PostgreSQL:

```
psycopg2-binary==2.9.9
```

3. **Actualizar el código** para usar PostgreSQL en producción:

```
# En app/core/database.py
import os

if os.getenv("ENVIRONMENT") == "production":
    DATABASE_URL = os.getenv("DATABASE_URL")  # PostgreSQL en Railway
else:
    DATABASE_URL = "sqlite:///./data/quimic.sqlite"  # SQLite local
```

### Paso 7: Verificar el Despliegue

1. **Acceder a la aplicación**:
   - Railway te proporcionará una URL como: `https://tu-app.railway.app`
   - Documentación: `https://tu-app.railway.app/docs`
2. **Verificar logs**:

```
railway logs
```

3. **Monitorear métricas**:
   - En el dashboard de Railway puedes ver CPU, memoria y tráfico

### Paso 8: Configurar Dominio Personalizado (Opcional)

1. **En Railway Dashboard**:
   - Ve a "Settings" > "Domains"
   - Click en "Add Custom Domain"
   - Ingresa tu dominio: `api.tu-dominio.com`
   - Configura los DNS records según las instrucciones

### Paso 9: Configurar CI/CD Automático

Railway automáticamente detecta cambios en tu repositorio de GitHub y despliega:

1. **Auto-deploy está habilitado por defecto**
2. **Para configurar branch específico**:
   - Ve a "Settings" > "Deploy"
   - Cambia "Deploy Branch" a `main` o `production`

### Paso 10: Optimizaciones para Producción

1. **Crear archivo `.railwayignore`**:

```
__pycache__/
*.pyc
.env
.git/
README.md
.pytest_cache/
```

2. **Configurar health checks** en `app/main.py`:

```

```

```
@app.get("/health")
async def health_check():
    return {"status": "healthy", "timestamp": datetime.now().isoformat()}
```

3. **Configurar logging para producción**:

```
import logging
import os

if os.getenv("ENVIRONMENT") == "production":
    logging.basicConfig(level=logging.INFO)
else:
    logging.basicConfig(level=logging.DEBUG)
```

## Troubleshooting Común

1. **Error de puerto**:
   - Asegúrate de usar `--port $PORT` en el comando de inicio
   - Railway asigna el puerto automáticamente
2. **Variables de entorno**:
   - Verifica que todas las variables están configuradas en Railway
   - No incluyas el archivo `.env` en el despliegue
3. **Base de datos**:
   - Si usas SQLite, los datos se perderán en cada despliegue
   - Considera migrar a PostgreSQL para persistencia
4. **Archivos estáticos**:
   - Los archivos en `/app/statics/` se incluirán en el despliegue
   - Para archivos grandes, considera usar un CDN

## Comandos Útiles de Railway

```
# Ver logs en tiempo real
railway logs --follow

# Conectar a la base de datos
railway connect postgres

# Ejecutar comandos en el contenedor
railway run python app/main.py

# Ver variables de entorno
```

```
railway variables

# Rollback a despliegue anterior
railway redeploy --previous
```

### Costos y Límites

- **Plan Hobby**: Gratuito con limitaciones
  - 500 horas de ejecución/mes
  - $5 USD de crédito mensual
- **Plan Pro**: $20 USD/mes
  - Recursos ilimitados
  - Soporte prioritario

Con estos pasos, tu API de Juegos de Química estará completamente desplegada en Railway y lista para ser utilizada en producción.

---

## Conclusión

Este manual técnico proporciona una guía completa del backend de Juegos de Química, desde la arquitectura hasta el despliegue. El sistema está diseñado para ser escalable, mantenible y educativo, proporcionando una base sólida para el aprendizaje interactivo de química.

Para obtener más información o reportar problemas, consulta la documentación adicional en los archivos `README.md` y `doc.md` del proyecto.