Arquitectura

1. Visión general de la arquitectura

Es una aplicación **Node.js + Express** con patrón **MVC "extendido"**:

- **Cliente (Frontend)**
  - Vistas **EJS** renderizadas por el servidor.
  - Ficheros estáticos en public/ (HTML de prototipo, CSS, JS, imágenes).
- **Servidor (Backend Express)**
  - server.js + src/app.js para configuración e inicialización.
  - **Rutas (routes)** → definen las URLs.
  - **Controladores (controllers)** → lógica de negocio de cada módulo.
  - **Servicios (services)** → integración con APIs externas y lógica de cálculo.
  - **Modelos (models con Mongoose)** → representación de colecciones MongoDB.
  - **Middleware** → control de sesión/acceso.
- **Persistencia y externos**
  - **MongoDB** vía Mongoose (src/config/db.js).
  - **API externa de fútbol** vía axios (footballApi.js).

2. Entrypoint y configuración base

server.js

- Punto de entrada del proyecto.
- Carga variables de entorno con **dotenv** (.env dentro de /backend).
- Conecta a MongoDB vía connectDB() antes de levantar Express.
- Arranca el servidor con app.listen(PORT).

src/app.js

- Crea la instancia de express().
- Configura:
  - **CORS**, **logger (morgan)**, parseo de JSON / URL-encoded.
  - **express-session** para manejar sesiones de usuario.
  - Carpeta de estáticos: public/.
  - Motor de plantillas **EJS** y ruta de views.
- Registra todas las rutas:
  - /, /account, /blog, /gamble, /help, /logMenu, /results, /settings, /subscription, /faq, /exclusive/roulette, /api/users, etc.

src/config/db.js

- Encapsula la conexión a **MongoDB** con mongoose.connect(MONGO_URI).
- Maneja errores y corta el proceso si falla.

3. Capas funcionales principales

3.1 Modelos (MongoDB, src/models/)

- **user.js**
  - Define el usuario de la plataforma:
    - Datos básicos: nombre, correo, contraseña hasheada, etc.
    - **Saldo**, historial, configuración, avatares…
    - Datos específicos de ruleta: historial de giros, timestamp del último spin, etc.
  - Usa timestamps: true para createdAt / updatedAt.
- **bet.js**
  - Representa una apuesta deportiva:
    - userId (referencia a User), fixtureId, equipos, selección (home, draw, away), cuota, cantidad, posible ganancia.
    - Estado: pending, won, lost.
- **match.js**
  - Partido de fútbol guardado desde la API:
    - apiId único.
    - Info de liga, equipos, marcador, estado del partido.
    - Estructura alineada con la respuesta de la API externa.
- **rouletteBet.js**
  - Apuestas hechas en el juego de **ruleta**:
    - Usuario, número/color/tipo de apuesta, cantidad, resultado, etc.

3.2 Controladores (src/controllers/)

Manejan la lógica de cada módulo y responden a las rutas.

- **userController.js**
  - Registro, login, logout.
  - Gestión de perfil y probablemente saldo e información de cuenta.
  - Usa la sesión req.session.user para mantener al usuario autenticado.
- **betController.js**
  - Crea apuestas deportivas:
    - Lee selection, amount, datos del partido y cuotas del req.body.
    - Verifica que el usuario esté logueado.
    - Comprueba saldo del usuario, descuenta importe, calcula potentialWin.
    - Guarda la apuesta en MongoDB.
- **betViewController.js / myBetsController.js**
  - Obtienen las apuestas del usuario y las renderizan (por ejemplo, en la vista my-bets.ejs).
  - Filtran por usuario autenticado y ordenan/agrupan.
- **liveController.js**
  - Lógica de **apuestas en directo**:
    - Obtiene partidos en vivo o próximos.
    - Usa la información de servicios y modelos para mostrar cuotas actualizadas.

- **matchController.js + matchesNotStarted.js**
    - Sincronizan y gestionan la colección de **partidos**:
        - Cargan datos desde la API de fútbol.
        - Guardan/actualizan en MongoDB (Match).
        - Filtran por partidos no iniciados, en vivo, finalizados, etc.
- **rouletteController.js**
    - Lógica del juego de **ruleta**:
        - Genera resultados (número/color).
        - Valida las apuestas provenientes del frontend.
        - Actualiza saldo del usuario y registra la apuesta (RouletteBet).

3.2 Servicios (src/services/)

Encapsulan lógica que no es puramente de controlador.

- **footballApi.js**
    - Integra la **API de fútbol** (https://v3.football.api-sports.io):
        - getMatchesByDate(date): devuelve fixtures de un día concreto.
        - getFixtureById(id): datos detallados de un partido.
    - Usa axios y cabeceras con FOOTBALL_API_KEY del .env.
- **oddsService.js**
    - Se encarga del **cálculo de cuotas (odds)**:
        - Usa heurísticas/probabilidades según estadísticas del partido/equipos.
        - Se apoya en datos de rankingService y/o Match para ajustar cuotas.
- **rankingService.js**
    - Calcula una especie de **"fuerza"/ranking híbrido** por equipo:
        - Combina posición en liga, racha, goles a favor/contra, etc.
        - Devuelve un valor numérico (por ejemplo, entre 0 y 100) para usar en odds.

3.4 Middleware (src/middleware/)

- **isLogged.js**
    - Middleware de protección:
        - Si no existe req.session.user, redirige a /logMenu.
        - Se usa en rutas que requieren usuario autenticado (por ejemplo /account).

3.5 Rutas (src/routes/)

Usan express.Router() y conectan URLs con controladores/vistas.

Principales:

- **index.js**
    - GET / → Renderiza index.ejs, pasando user si está en sesión.
- **logMenu.js**
    - GET /logMenu → Vista de login/registro.
    - Muestra errores de login almacenados en req.session.loginError.
- **account.js**
    - GET /account (protegido con isLogged).
    - Lee la carpeta public/avatars y pasa la lista a la vista account.ejs para que el usuario elija su avatar.

- **gamble.js, live.js, faq.js, blog.js, help.js, results.js, settings.js, subscription.js**
    - Cada uno renderiza la vista correspondiente (apuestas, blog, FAQ, ayuda, resultados, ajustes, suscripciones…).
    - Algunos llaman a controladores específicos para montar los datos antes de res.render.
- **roulette.js (montado en /exclusive/roulette)**
    - Rutas privadas para el juego de ruleta.
    - Usa rouletteController para gestionar el juego y las apuestas.
- **apiUsers.js**
    - API bajo /api/users (endpoints REST para usuario: obtener info, actualizar, etc.).

4. Vistas y Frontend

src/views/ (EJS)
Plantillas que componen el frontend del sitio:

- index.ejs – página principal.
- logMenu.ejs – login / registro.
- account.ejs – perfil de usuario y avatares.
- gamble.ejs, bet.ejs, my-bets.ejs, live.ejs, results.ejs – área de apuestas deportivas (partidos, apuestas, mis apuestas, en directo, resultados).
- roulette.ejs – interfaz del juego de ruleta.
- blog.ejs, faq.ejs, help.ejs, settings.ejs, subscription.ejs – contenido y configuración.

Cada EJS usa:

- CSS desde /css/*.css.
- JS desde /Js/*.js.
- Variables pasadas desde los controladores (user, matches, bets, avatars, etc.).

5. Recursos estáticos (public/)

- **public/css/**
    - Hojas de estilo por página: index.css, gamble.css, results.css, logMenu.css, settings.css, subscription.css, etc.
- **public/Js/**
    - Scripts de frontend:
        - index.js – lógica de la home.
        - logMenu.js – manejo de formularios de login/registro.
        - gamble.js – interacciones en la página de apuestas.
        - results.js, settings.js, subscription.js, account.js, blog.js, etc.
        - cardsGame.js – mini juego o animaciones/cartas dentro de la web.
- **public/html/**
    - Versiones HTML estáticas (prototipos) de las vistas.
    - Sirvieron como base antes de migrar a EJS.
- **public/imagenes/, public/avatars/**
    - Imágenes, logotipos, iconos, banner, avatares de usuario.

6. Diagrama de arquitectura (texto)