Manual Técnico

Sistema "Blockly – Games | Universidad de Guanajuato"

Elaboró: *Christian Pérez Alfaro*

Correo: *c.perezalfargto.mx*

Project Owner: *Dra. María Susana Ávila García*

Scrum Master: *Mtro. Juan Carlos Avilés Díaz*

Campus Irapuato-Salamanca | División de Ingenierías
Departamento de Estudios Multidisciplinarios

Contenido

1. Introducción

1.1 Descripción del sistema

El sistema educativo **Blockly Games | UG** es una aplicación web que permite a estudiantes jugar ejercicios de programación con bloques (por ejemplo, Maze y Bird) y, de forma paralela, registrar información detallada sobre su desempeño (sesiones, intentos, tiempos, estados y acciones sobre los bloques).

El sistema incluye además un **panel para investigadores**, donde se consultan estadísticas por jugador, juego y nivel, así como la secuencia completa de eventos de cada intento.

1.2 Objetivo del documento

Este **Manual Técnico** describe la solución desde el punto de vista de ingeniería de software, con los siguientes objetivos específicos:

- Documentar la **arquitectura** del sistema (frontend, backend y base de datos).
- Definir el **modelo de datos** y los principales procesos internos.
- Especificar los **módulos de backend**, las pantallas del frontend y su relación con la base de datos.
- Proporcionar lineamientos para **instalación, configuración y mantenimiento** de la plataforma.

1.3 Alcance

El documento cubre:

- Funcionalidad técnica del **módulo de jugador** y del **módulo de investigador**.
- Componentes del **servidor de aplicaciones** (Node.js / Express).
- Diseño de la **base de datos MySQL** (tablas, vistas y procedimientos almacenados).
- Diagramas de arquitectura, datos, procesos y estados.
- Aspectos básicos de despliegue y configuración.

1.4 Público objetivo

El manual está dirigido a:

- Desarrolladores responsables de **mantener o extender** el sistema.
- Administradores encargados de **instalar y operar** la plataforma.
- Investigadores con perfil técnico que requieran entender **cómo se generan y almacenan los datos de telemetría**.

Se asume familiaridad básica con desarrollo web, Node.js y bases de datos relacionales.

2. Visión general del sistema

2.1 Descripción funcional

El sistema está compuesto por dos módulos principales:

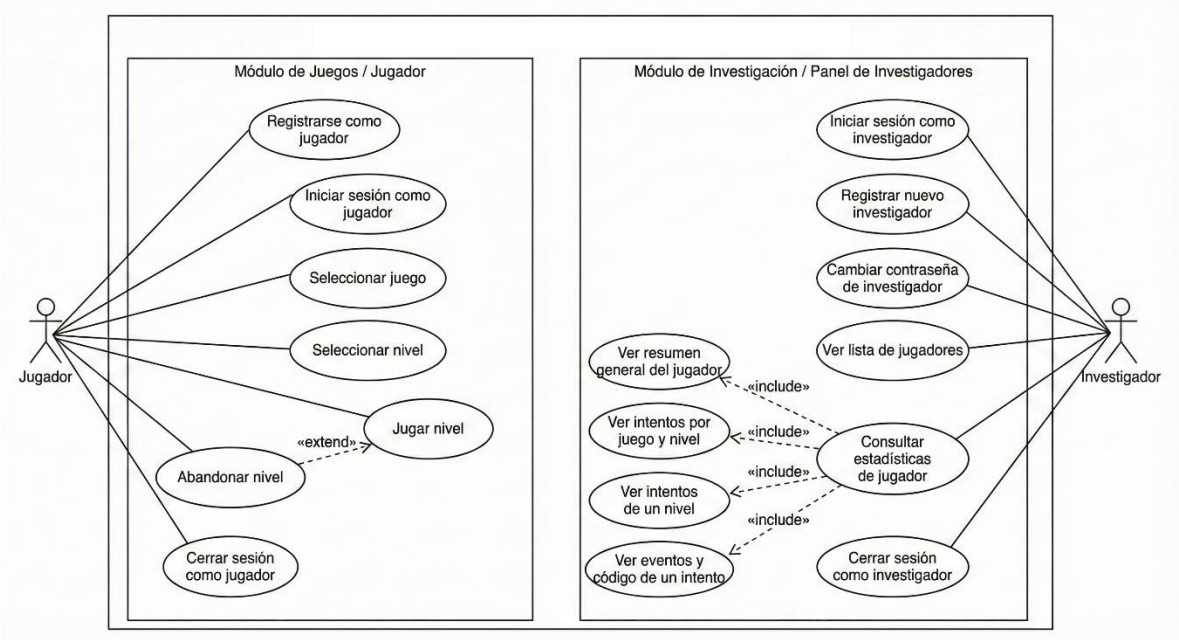
- **Módulo de Juegos (Jugador)**
  Permite a los estudiantes:
  - Registrarse mediante un PIN de 4 dígitos.
  - Iniciar sesión.
  - Seleccionar un juego (actualmente Maze y Bird) y un nivel.
  - Construir programas con bloques y ejecutar sus soluciones.
  - Generar telemetría automática sobre sesiones, intentos y acciones realizadas durante el juego.
- **Módulo de Investigación (Panel de Investigadores)**
  Permite a los investigadores:
  - Iniciar sesión con PIN y contraseña.
  - Registrar nuevos investigadores y cambiar su propia contraseña.
  - Consultar la lista de jugadores con información de sesiones y niveles completados.
  - Acceder a estadísticas detalladas por jugador, juego y nivel, incluyendo:
    - Resumen de desempeño.
    - Intentos por juego y nivel.
    - Historial de intentos de un nivel.
    - Secuencia de eventos y código generado en cada intento.

2.2 Actores del sistema

Se identifican dos actores externos:

- **Jugador**
  Estudiante que interactúa con los juegos Blockly. Su actividad genera:
  - Sesiones de uso.
  - Intentos por juego–nivel.
  - Acciones elementales sobre los bloques (crear, mover, conectar, editar, eliminar).
- **Investigador**
  Docente o analista que utiliza el panel de investigación para:
  - Administrar usuarios investigadores.
  - Analizar el comportamiento de los jugadores a partir de la telemetría almacenada.

*En la Figura 1 se presenta el **diagrama de casos de uso** del sistema, donde se muestran las funcionalidades asociadas a cada actor para los módulos de Juegos y de Investigación.*



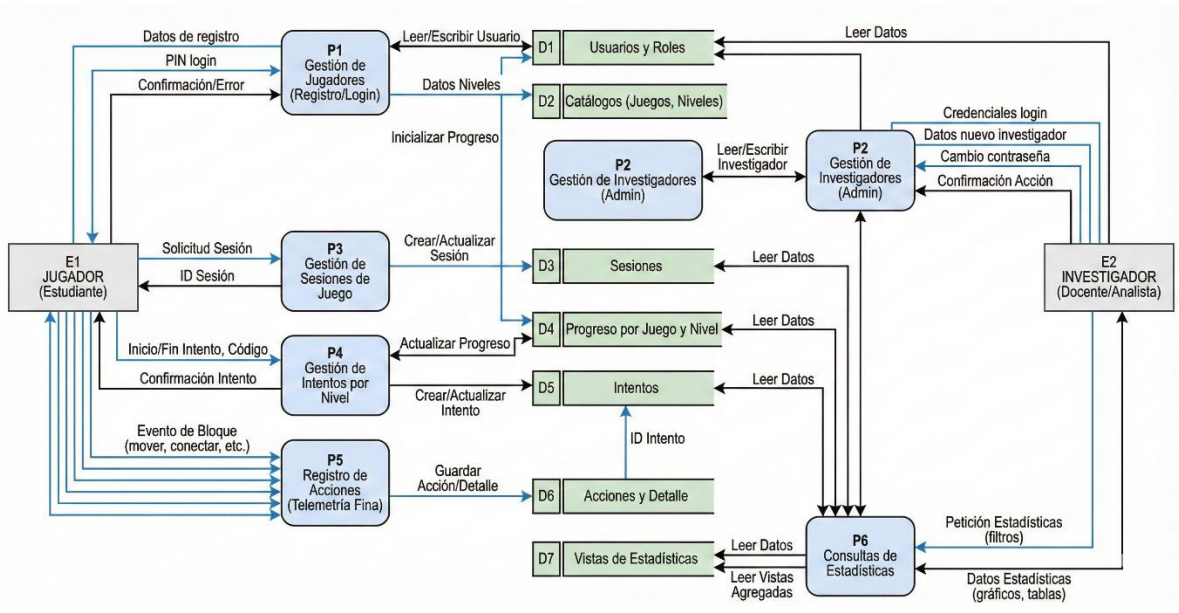
2.3 Flujo general de información

A alto nivel, el flujo de datos del sistema es el siguiente:

1. El **Jugador** se registra o inicia sesión mediante su PIN.
2. Al acceder a un juego y nivel:
   - Se crea una **sesión de juego** y, para cada intento, un registro de **INTENTO** asociado a ese juego–nivel.
   - Durante el intento se capturan **acciones de bloques** (ACCION y tablas de detalle), con información de tipo de acción, tipo de bloque y orden temporal.
   - Al finalizar el intento, se actualizan el estado del intento (éxito, fallo, abandono) y el **PROGRESO** del jugador en ese juego–nivel.
3. El **Investigador** inicia sesión en el panel y realiza consultas sobre:
   - Información agregada de **sesiones, intentos y progreso**.
   - Vistas especializadas para estadísticas y gráficos.

Estos datos se almacenan en la base de datos MySQL y se exponen al panel mediante servicios del servidor de aplicaciones.

*En la Figura [Y] se muestra el **Diagrama de Flujo de Datos (DFD – contexto + nivel 0)**, donde se representan los procesos principales.*



3. Arquitectura del sistema

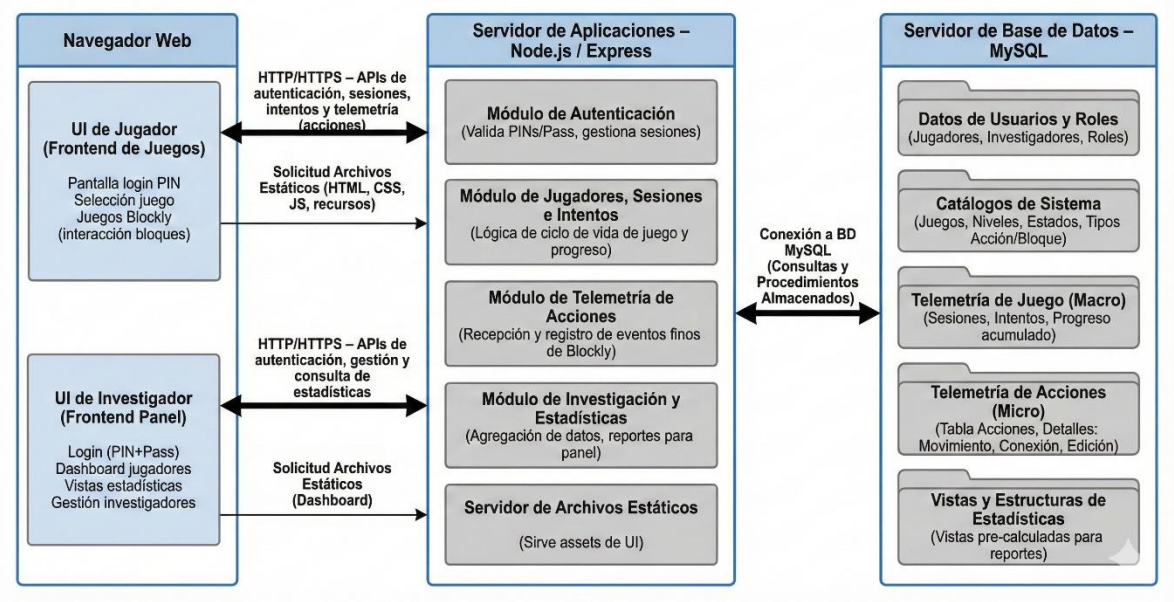
3.1 Vista general de arquitectura

El sistema se implementa con una arquitectura web de tres capas:

1. **Capa de presentación (Navegador web)**
    - Interfaz de **Jugador**:
        - Pantalla de login por PIN.
        - Panel de selección de juego y nivel.
        - Juegos Blockly (Maze y Bird) embebidos en el navegador.
    - Interfaz de **Investigador**:
        - Pantalla de acceso por PIN y contraseña.
        - Panel con tabla de jugadores y vistas de estadísticas.
2. **Servidor de aplicaciones – Node.js / Express**
    - Expone servicios HTTP/HTTPS para:
        - Autenticación de jugadores e investigadores.
        - Gestión de sesiones de juego, intentos y progreso.
        - Registro de acciones de bloques (telemetría fina).
        - Consultas de estadísticas para el panel de investigación.
    - Implementa la lógica de negocio que coordina la interacción entre las interfaces web y la base de datos.
3. **Servidor de base de datos – MySQL**
    - Almacena:
        - Usuarios, roles y catálogos (juegos, niveles, estados, tipos de acción/bloque).

- Sesiones, progreso e intentos (telemetría de juego a nivel macro).
- Acciones detalladas de bloques y sus parámetros (telemetría micro).
- Vistas y estructuras de apoyo para estadísticas y reportes.

En la *Figura 3* se presenta el **Diagrama C4 de contenedores**, donde se muestran estos tres bloques principales y las comunicaciones entre ellos.



3.2 Componentes del servidor de aplicaciones

El servidor Node.js / Express se organiza en módulos lógicos que agrupan controladores, servicios y rutas relacionadas. A nivel conceptual se distinguen los siguientes componentes:

- **Módulo de Autenticación**
  - Valida PIN de jugador y PIN+contraseña de investigador.
  - Crea y gestiona la sesión de aplicación correspondiente a cada tipo de usuario.
- **Módulo de Jugadores, Sesiones e Intentos**
  - Gestiona el alta de jugadores (registro por PIN).
  - Crea y cierra **SESION** de juego cuando el jugador entra y sale del sistema.
  - Inicia y finaliza **INTENTO** por juego–nivel, actualizando la entidad **PROGRESO**.
- **Módulo de Telemetría de Acciones**
  - Recibe los eventos generados por los juegos Blockly (crear/mover/conectar/editar/eliminar bloques).
  - Registra cada evento en las tablas **ACCION** y tablas de detalle correspondientes.
- **Módulo de Investigación y Estadísticas**
  - Expone servicios para el panel de investigadores:
    - Lista de jugadores y sus métricas globales.
    - Estadísticas por juego y nivel.
    - Historial de intentos y detalle de eventos por intento.
  - Consume vistas y consultas especializadas de la base de datos.

- **Servidor de archivos estáticos**
  - Sirve las páginas HTML, hojas de estilo, scripts JavaScript y recursos de los juegos y del panel.

3.3 Componentes del servidor de base de datos

En la base de datos MySQL se organizan los datos en varios grupos funcionales:

- **Datos de usuarios y roles**
  - Tablas: USUARIO, ROL.
  - Almacenan información de jugadores e investigadores, incluyendo PIN y, para investigadores, contraseña.
- **Catálogos de sistema**
  - Tablas: JUEGO, NIVEL, ESTADO, TIPO_ACCION, TIPO_BLOQUE.
  - Definen los valores fijos necesarios para describir juegos, niveles, estados de progreso/intentento y tipos de acciones/bloques.
- **Telemetría de juego (macro)**
  - Tablas: SESION, PROGRESO, INTENTO.
  - Registran sesiones de uso, avance por juego–nivel y cada intento realizado, con tiempos y estados asociados.
- **Telemetría de acciones (micro)**
  - Tablas: ACCION, MOVIMIENTO_BLOQUE, CONEXION_BLOQUE, EDICION_BLOQUE, y otras tablas de detalle.
  - Almacenan eventos atómicos generados por la interacción con los bloques (tipo de acción, tipo de bloque, parámetros, orden temporal).

- **Vistas y estructuras para estadísticas**
  - Vistas como vw_research_players, vw_intento_acciones y otras vistas agregadas.
  - Simplifican las consultas del módulo de investigación, exponiendo datos ya combinados y listos para gráficos y reportes.

4. Modelo de datos lógico

El sistema se organiza en cuatro grupos de información principales:

4.1 Usuarios y roles

- **USUARIO**: almacena jugadores e investigadores (PIN, datos básicos y, para investigadores, contraseña).
- **ROL**: define el tipo de usuario (jugador, investigador).
- Relación: un ROL puede tener muchos USUARIO.

4.2 Catálogos

Tablas de referencia usadas en todo el sistema:

- **JUEGO** y **NIVEL**: describen los juegos disponibles y sus niveles.
- **ESTADO**: estados posibles (pendiente, en proceso, completado, fallado, abandonado).
- **TIPO_ACCION** y **TIPO_BLOQUE**: tipos de eventos de telemetría y tipos de bloques de Blockly.

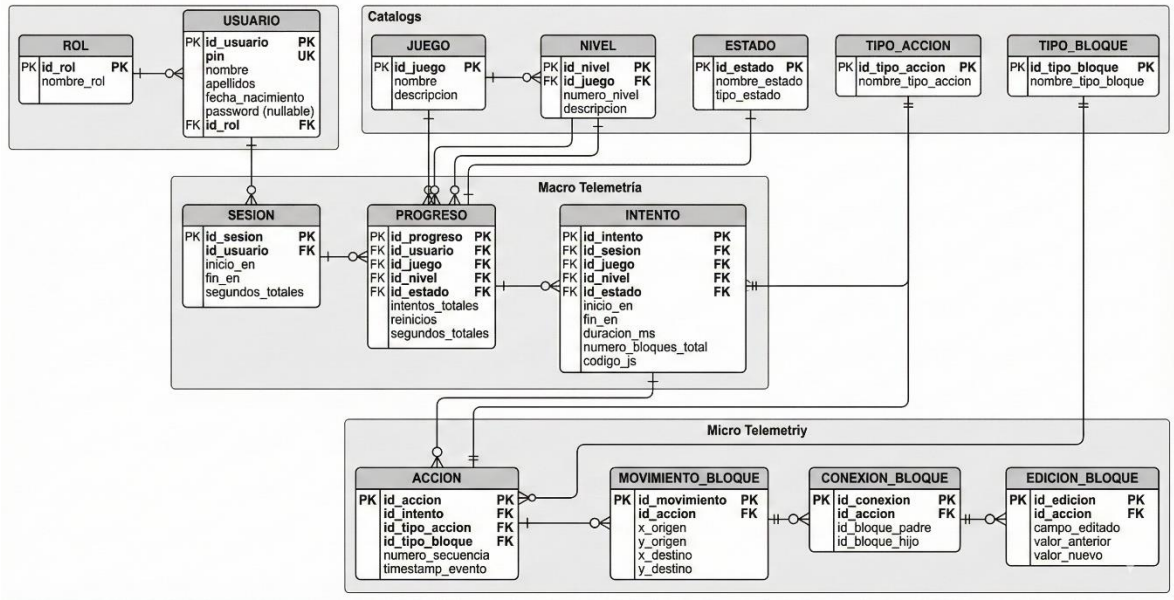
4.3 Telemetría de juego (resumen)

Registra el desempeño general del jugador:

- **SESION**: cada vez que el jugador entra y sale del sistema.
- **PROGRESO**: estado del jugador por cada combinación juego–nivel (estado actual, intentos, tiempo acumulado).
- **INTENTO**: cada intento concreto de resolver un nivel dentro de una sesión (estado final, duración, bloques y código generado).

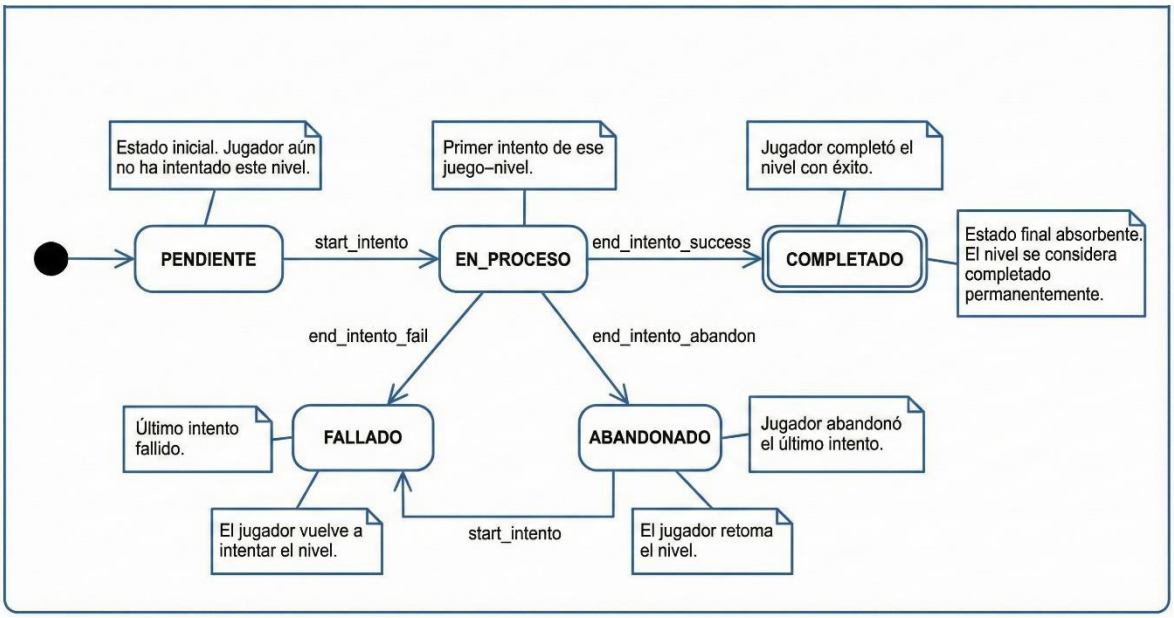
Registra lo que pasa dentro de cada intento:

- **ACCION**: evento atómico (crear, mover, conectar, editar, eliminar un bloque), ligado a un INTENTO.
- Tablas de detalle (como MOVIMIENTO_BLOQUE, CONEXION_BLOQUE, EDICION_BLOQUE) guardan información específica según el tipo de acción.



5. Comportamiento del sistema

5.1 Ciclo de vida del progreso (por juego–nivel)



El **PROGRESO** describe cómo va un jugador en un juego y nivel específicos. Su ciclo de vida se resume así:

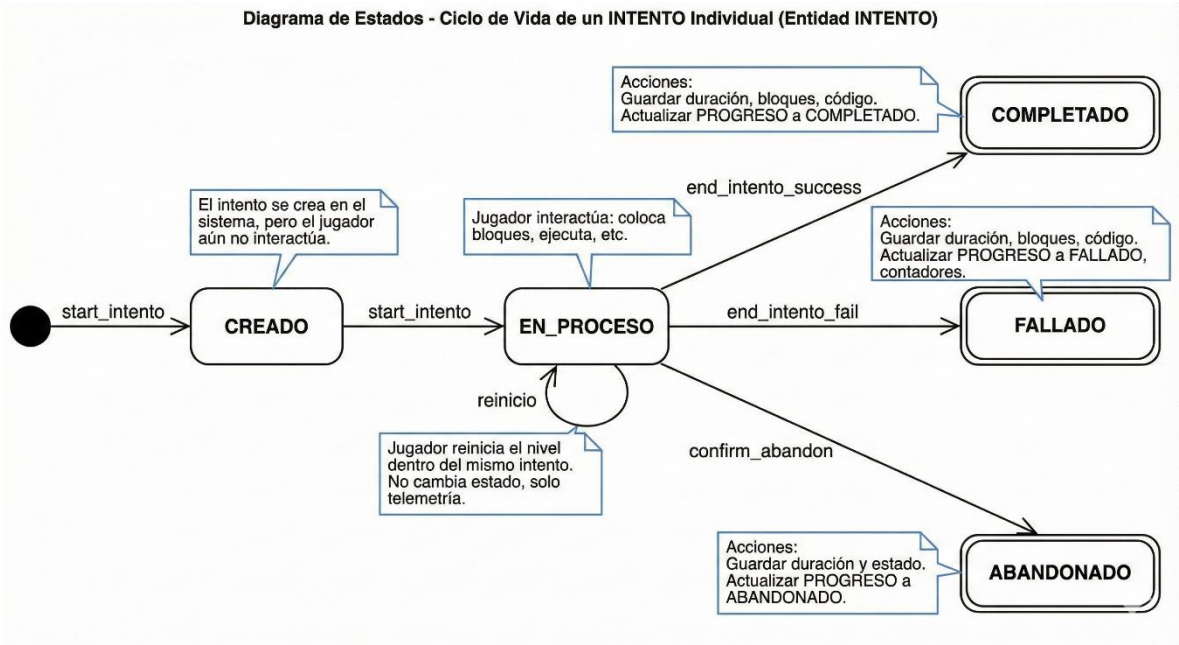
- **PENDIENTE**
  Estado inicial. El jugador aún no ha intentado ese nivel.
- **EN PROCESO**
  El jugador ya comenzó a trabajar en ese nivel al menos una vez.
- **COMPLETADO**
  El jugador logró finalizar correctamente el nivel.
- **FALLADO**
  El intento más reciente terminó en fallo.

- **ABANDONADO**
  El intento más reciente terminó porque el jugador salió del nivel antes de completarlo.

Cambios típicos de estado:

- De **PENDIENTE** a **EN PROCESO** cuando el jugador intenta el nivel por primera vez.
- De **EN PROCESO** a **COMPLETADO**, **FALLADO** o **ABANDONADO** según el resultado del último intento.
- Desde **FALLADO** o **ABANDONADO**, un nuevo intento vuelve a colocar el progreso en **EN PROCESO**.
- Habitualmente, **COMPLETADO** se considera el estado final deseado para ese juego–nivel.

5.2 Ciclo de vida de un intento individual



Cada **INTENTO** representa un intento concreto del jugador por resolver un nivel dentro de una sesión.

Los estados principales son:

- **CREADO**
  El intento ha sido registrado y está listo para comenzar.
- **EN PROCESO**
  El jugador está interactuando con el nivel: coloca bloques, los mueve, ejecuta el programa y puede reiniciarlo.
  - Los reinicios dentro del mismo nivel no cambian de estado, simplemente actualizan contadores y telemetría.
- **COMPLETADO**
  El sistema determina que la solución del jugador resuelve correctamente el nivel.
  - Se registran la hora de finalización, la duración, el número de bloques y el código generado.
  - Se actualiza el PROGRESO del jugador en ese juego–nivel como completado.
- **FALLADO**
  El intento termina con una solución incorrecta.
  - Se registran duración, bloques y código final.
  - Se actualizan los contadores e indicadores de PROGRESO como intento fallido.
- **ABANDONADO**
  El jugador decide salir del nivel antes de resolverlo.
  - Se registra la finalización del intento con estado abandonado.
  - Se actualiza el PROGRESO para reflejar este resultado.

Transiciones típicas:

- De **CREADO** a **EN PROCESO** cuando el jugador comienza a trabajar en el nivel.
- De **EN PROCESO** a **COMPLETADO**, **FALLADO** o **ABANDONADO** cuando el intento termina.

Con este ciclo de vida, cada intento queda claramente identificado con un estado final y con los datos necesarios para alimentar las estadísticas y el seguimiento del PROGRESO del jugador.

6. Base de datos – diseño e implementación

6.1 Descripción general

La base de datos del sistema está implementada en **MySQL** y se organiza en un único esquema lógico que concentra:

- Información de **usuarios y roles**.
- **Catálogos** de juegos, niveles, estados y tipos.
- Registros de **sesiones**, **progreso** e **intentos** de los jugadores.
- Telemetría detallada de **acciones sobre bloques**.
- **Vistas** y estructuras de apoyo para las estadísticas del módulo de investigación.

El diseño sigue el modelo lógico descrito en el capítulo anterior, garantizando integridad referencial mediante claves primarias y foráneas.

6.2 Dump de base de datos

En la raíz del proyecto se incluye un archivo de *dump* de MySQL (por ejemplo, blockly_db_dump.sql) que contiene:

- La creación de todas las tablas y sus relaciones.
- La definición de vistas y, en su caso, procedimientos almacenados asociados al sistema.
- La carga inicial de catálogos (juegos, niveles, estados, tipos de acción y tipos de bloque).

Este archivo permite recrear la base de datos completa en un servidor MySQL limpio, asegurando la compatibilidad con el backend y el frontend descritos en este manual.

7. Backend del sistema

El backend del sistema educativo con Blockly y telemetría está desarrollado con **Node.js** y **Express**, y utiliza **MySQL** como motor de base de datos.
Su objetivo es exponer servicios HTTP para:

- la **interacción de jugadores** con los juegos (sesiones, intentos, acciones), y
- el **panel de investigadores** (gestión de investigadores y consulta de estadísticas).

El código se organiza en carpetas por responsabilidad:

- server.js – arranque del servidor Express.
- src/config – configuración de entorno y conexión a MySQL.
- src/routes – definición de rutas HTTP.
- src/middlewares/auth – middlewares de autenticación.
- src/controllers – lógica de negocio separada por dominio (auth, user, player, investigator).

7.1 Módulos y responsabilidades

La siguiente tabla resume los módulos principales del backend:

Módulo	**Ubicación principal**	**Responsabilidad central**
Configuración y arranque	server.js, src/config/env/env.js, src/config/db/db.js	Iniciar Express, leer variables de entorno y crear el pool de conexión MySQL.
Ruteo	src/routes/routes.js, src/routes/auth/authRoutes.js, src/routes/general/*.js	Agrupar y registrar las rutas públicas y protegidas para jugador e investigador.
Middlewares de autenticación	src/middlewares/auth/player.js, src/middlewares/auth/investigator.js	Validar la sesión de jugador o investigador antes de acceder a las rutas protegidas.
Autenticación	src/controllers/auth/*.js	Inicio y cierre de sesión de jugadores e investigadores.
Gestión de usuarios	src/controllers/user/*.js	Registro de jugadores e investigadores y cambio de contraseña de investigadores.
Módulo de jugador	src/controllers/player/*.js	Sesiones de juego, intentos por nivel, registro de acciones y lectura de progreso.
Módulo de investigación	src/controllers/investigator/*.js	Listado de jugadores, overview, estadísticas por juego/nivel e históricos de intentos.
Servicios auxiliares	getGamesCatalog.js, getLevelsCatalogByGame.js (en controllers/investigator)	Devolver listas de juegos y niveles para poblar la interfaz del panel.

7.2 Flujo general de peticiones

7.2.1 Jugador

1. El jugador accede a la interfaz web de juegos y realiza **login con PIN**.
2. El frontend llama a las rutas de autenticación definidas en authRoutes.js, que utilizan los controladores de controllers/auth/player.js.
3. Una vez autenticado, el middleware middlewares/auth/player.js protege el resto de rutas bajo el prefijo de jugador (por ejemplo, acceso a juegos, creación de intentos y registro de acciones).
4. Durante la sesión:
    - Se crean y cierran **INTENTOS** mediante startTry.js y endTry.js.
    - Se envía la telemetría de bloques con actionControllers.js.
    - Se consulta el progreso del jugador con readProgress.js y otros controladores relacionados.

7.2.2 Investigador

1. El investigador accede al panel y realiza **login con PIN y contraseña**.
2. La petición se procesa en controllers/auth/investigator.js, y, si es correcta, se genera una sesión protegida por middlewares/auth/investigator.js.
3. Las rutas de routes/general/investigator.js permiten:
    - administrar investigadores (registro y cambio de contraseña), y
    - consultar las estadísticas de jugadores (overview, intentos, eventos).

7.3 Configuración y arranque

7.3.1 server.js

- Crea la instancia principal de Express.
- Carga la configuración de entorno desde src/config/env/env.js.
- Inicializa la conexión a MySQL mediante el módulo src/config/db/db.js.
- Registra el enrutador raíz src/routes/routes.js.
- Levanta el servidor HTTP escuchando en el puerto indicado en las variables de entorno.

7.3.2 src/config/env/env.js

- Lee las variables de entorno (.env) y exporta un objeto de configuración centralizado (puerto, host de BD, usuario, contraseña, base de datos, etc.).
- Permite que el resto de módulos use la configuración sin acceder directamente al archivo .env.

7.3.3 src/config/db/db.js

- Crea un **pool de conexiones** MySQL reutilizable por todos los controladores.
- Expone funciones auxiliares para ejecutar consultas, vistas y procedimientos almacenados.

7.4 Ruteo y middlewares de autenticación

7.4.1 src/routes/routes.js

- Actúa como **enrutador principal**.
- Monta:
  - las rutas de autenticación definidas en routes/auth/authRoutes.js,
  - las rutas generales de jugador (routes/general/player.js),
  - las rutas generales de investigador (routes/general/investigator.js),
  - y cualquier ruta global compartida (routes/general/global.js).

7.4.2 src/routes/auth/authRoutes.js

- Define las rutas de:
  - **inicio de sesión de jugador**,
  - **inicio de sesión de investigador**,
  - **cierre de sesión**.
- Cada ruta delega la lógica en su controlador correspondiente de controllers/auth.

7.4.3 Middlewares de autenticación

- **src/middlewares/auth/player.js**
  Comprueba que la petición proviene de un jugador autenticado.
  - Valida la sesión almacenada.
  - Añade al objeto req los datos mínimos necesarios (por ejemplo, identificador del usuario) para que los controladores puedan registrar sesiones, intentos y acciones.
- **src/middlewares/auth/investigator.js**
  Realiza la validación equivalente para investigadores, protegiendo

todas las rutas del panel de investigación.

Estos middlewares se aplican en las rutas de routes/general/player.js y routes/general/investigator.js respectivamente.

7.5 Módulo de autenticación

Ubicación: src/controllers/auth/.

- **Controlador de jugador (player.js)**
  - Recibe el PIN del jugador.
  - Valida el usuario en la base de datos.
  - Crea la sesión de jugador y devuelve la información necesaria para el frontend.
- **Controlador de investigador (investigator.js)**
  - Recibe PIN y contraseña.
  - Verifica las credenciales frente a la vista de autenticación.
  - Genera la sesión de investigador.
- **Controlador de cierre de sesión (logout.js)**
  - Invalida la sesión activa, tanto para jugador como para investigador, según el contexto.

7.6 Módulo de gestión de usuarios

Ubicación: src/controllers/user/.

- **registerPlayer.js**
  - Registra nuevos jugadores.
  - Inserta un registro en USUARIO con rol de jugador y PIN de 4 dígitos.
- **registerInvestigator.js**
  - Registra nuevos investigadores.
  - Valida PIN, datos personales y contraseña.
  - Genera un hash seguro de la contraseña y crea el usuario con rol de investigador.
- **changePasswordInvestigator.js**
  - Requiere que el investigador esté autenticado.
  - Verifica la contraseña actual contra el hash almacenado.
  - Actualiza la contraseña con un nuevo hash si la verificación es correcta.

7.7 Módulo de jugador, sesiones e intentos

Ubicación: src/controllers/player/ con rutas en routes/general/player.js.

- **startTry.js**
  - Crea un nuevo INTENTO cuando el jugador inicia un nivel.
  - Asocia el intento a la sesión activa, el juego y el nivel.
  - Inicializa campos como estado y marcas de tiempo.
- **endTry.js**

- Cierra un intento existente cuando el jugador termina o abandona el nivel.
- Actualiza el estado final (completado, fallado, abandonado), duración, número de bloques y código generado.
- Actualiza también el PROGRESO del jugador en ese juego–nivel.

- **actionControllers.js**
  - Recibe la secuencia de eventos de Blockly (crear, mover, conectar, editar, eliminar bloques).
  - Inserta los registros correspondientes en la tabla ACCION y en sus tablas de detalle.
  - Cada acción queda asociada al intento actual y a un tipo de acción/bloque.
- **readProgress.js**
  - Devuelve información de avance del jugador utilizando vistas de progreso por juego y nivel.
  - Se usa en el panel del jugador para mostrar qué niveles ha completado o intentado.

7.8 Módulo de investigación y estadísticas

Ubicación: src/controllers/investigator/ con rutas en routes/general/investigator.js.

Principales controladores:

- **listPlayers.js**
  Obtiene la lista de jugadores con datos resumidos: PIN, nombre, niveles completados, número de sesiones y fecha de última sesión.
- **getPlayerIdentity.js**
  Devuelve los datos de identificación de un jugador concreto (PIN, nombre, apellidos).
- **getPlayerOverview.js**
  Recupera indicadores globales del jugador:
  - intentos por estado (éxito, fallo, abandono),
  - tiempos promedio,
  - niveles completados y total de intentos.
- **getPlayerGameSummary.js**
  Proporciona un resumen por juego:
  - intentos totales,
  - niveles completados,
  - porcentaje de éxito/fallo/abandono,
  - tiempos promedio por juego y nivel.
- **getPlayerGameLevels.js**
  Devuelve, para un juego específico, el detalle de niveles del jugador:
  - estado del progreso,
  - tiempo acumulado,
  - intentos totales,

- fechas de primer y último intento.

- **getLevelAttempts.js**
  Lista los intentos realizados por un jugador en un nivel concreto, con estado, duración, número de bloques y código generado. Incluye soporte de paginación para niveles con muchos intentos.
- **getAttemptEvents.js**
  Devuelve la secuencia de eventos registrada para un intento, en orden temporal.
  Se utiliza para mostrar, en el panel, la narrativa de acciones y el código final asociado a ese intento.
- **getPlayerSessionsSeries.js**
  Obtiene una serie temporal de sesiones e intentos del jugador, con posibilidad de filtrar por rango de fechas.
  Alimenta las gráficas de intentos por sesión y tiempos promedio.

7.9 Servicios auxiliares (juegos y niveles)

También ubicados en src/controllers/investigator/ y expuestos por routes/general/global.js:

- **getGamesCatalog.js**
  - Lee la información de los juegos disponibles en la base de datos.
  - Devuelve una lista de identificadores y nombres de juegos, ordenados de forma estable.
- **getLevelsCatalogByGame.js**
  - Devuelve los niveles asociados a un juego concreto.
  - Se usa para poblar selectores de nivel y validar que el progreso del jugador se corresponda con los niveles configurados.

Estos servicios no modifican datos; únicamente exponen información de referencia para las interfaces de jugador e investigador.

7.10 Relación backend – base de datos (resumen)

De forma resumida, la interacción del backend con la base de datos es la siguiente:

- **Módulo de autenticación y usuarios**
  - Lectura de vistas de autenticación y datos de usuario.
  - Escrituras sobre USUARIO (registro de jugadores e investigadores, cambio de contraseña).
- **Módulo de jugador**
  - Escrituras en SESION, INTENTO, PROGRESO, ACCION y tablas de detalle de acciones.
  - Lecturas de vistas de progreso y catálogos de juegos/niveles.
- **Módulo de investigación**
  - Lecturas de vistas agregadas que resumen sesiones, intentos, progreso y eventos.
  - No realiza escrituras sobre la telemetría; su función es exclusivamente de consulta.

8. Frontend del sistema

El frontend está basado en páginas HTML estáticas servidas por Express y en archivos JavaScript que conectan la interfaz con los servicios del backend y con los juegos de Blockly (Maze y Bird). Todo el código del cliente se encuentra en la carpeta client/public.

8.1 Estructura general del frontend

Dentro de client/public se distinguen tres grupos:

- **views/**
  Contiene las páginas principales del sistema:
  - index.html – página de inicio del jugador (login por PIN).
  - login.html – pantalla de login de jugador (equivalente a index en la versión actual).
  - login-inv.html – pantalla de login para investigadores (PIN + contraseña).
  - player.html – panel principal del jugador con la selección de juegos (Maze y Bird).
  - maze.html, bird.html – contenedores HTML de los juegos Maze y Bird con Blockly embebido.
  - dashboard.html – panel web de investigadores para consultar las estadísticas de los jugadores.
- **static-files/**
  Reúne los recursos estáticos creados para este proyecto:
  - static-files/css/ – hojas de estilo para login de jugador, login de investigador, panel del jugador, dashboard y demás pantallas personalizadas.
  - static-files/img/ – logotipos e imágenes de la interfaz.
  - static-files/js/ – scripts JavaScript que gestionan formularios, navegación y llamadas al backend (ver 8.2).

- **Carpetas de juegos** (maze/, bird/, common/, generated/, third-party/, src/)
  Contienen los archivos originales de Blockly Games (HTML, JS y recursos) necesarios para que Maze y Bird funcionen en el navegador.
  Sobre estos archivos se añadieron "hooks" mínimos para enlazar la telemetría y el control de intentos con el backend.

8.2 Scripts JavaScript del proyecto

Los archivos JavaScript propios del sistema se encuentran en static-files/js/. A nivel funcional se dividen en dos grupos:

- **Pantallas de acceso y navegación:**
  - login.js – lógica de la pantalla de acceso del jugador (lectura de PIN, validaciones básicas y envío al backend).
  - login-inv.js – lógica de la pantalla de acceso del investigador (PIN + contraseña).
  - index-player.js / player.js – manejo de la vista principal del jugador: bienvenida, lectura del PIN activo, botones "Jugar" de Maze y Bird y cierre de sesión.
  - dashboard.js – consumo de las APIs del módulo de investigación para poblar tablas, tarjetas de KPIs y gráficas del panel.
- **Integración con cada juego de Blockly:**
  - mazeTryStart.js, birdTryStart.js – lógica relacionada con la creación de un intento cuando el jugador entra a un nivel de Maze o Bird (comunicación con el backend para registrar el inicio).
  - mazeANL.js, birdANL.js – scripts auxiliares asociados al análisis de intentos y telemetría para cada juego.

- mazeF.js, birdF.js – funciones específicas de finalización de intento (llamadas al backend cuando el jugador completa, falla o abandona el nivel).

Estos scripts no modifican la mecánica de Blockly, sino que se apoyan en los eventos de los juegos para:

- informar al backend del **inicio y fin de intentos**, y
- enviar los **eventos de bloques** (crear, mover, conectar, editar, eliminar) que alimentan la telemetría.

8.3 Flujo de navegación del jugador

1. El usuario accede a index.html / login.html.
2. login.js captura el PIN de 4 dígitos y lo envía al backend para validar al jugador.
3. Si la autenticación es correcta, el sistema redirige a player.html, donde:
   - se muestra un mensaje de bienvenida, y
   - se presentan las tarjetas de los juegos disponibles (Maze y Bird) con el botón **"Jugar"**.
4. Al elegir un juego:
   - el navegador abre maze.html o bird.html,
   - los scripts mazeTryStart.js o birdTryStart.js coordinan la creación del intento en el backend,
   - y los scripts de integración registran la telemetría mientras el jugador arrastra y conecta bloques.
5. Cuando el nivel termina (éxito, fallo o abandono), los scripts de finalización (mazeF.js, birdF.js) notifican al backend y, según el caso, ofrecen continuar al siguiente nivel o regresar al panel del jugador.

8.4 Flujo de navegación del investigador

1. El investigador accede a login-inv.html.
2. login-inv.js captura PIN y contraseña y realiza la solicitud de autenticación al backend.
3. Tras un login exitoso, se carga dashboard.html, que utiliza dashboard.js para:
    - obtener la **lista de jugadores** (PIN, nombre, niveles completados, sesiones, última sesión),
    - mostrar el **detalle de un jugador** (overview, estadísticas por juego y nivel, intentos e historial de eventos),
    - abrir los modales para **registrar nuevos investigadores** y **cambiar contraseña**.

Toda la información de estadísticas se obtiene a través de las APIs descritas en la sección de backend, usando vistas agregadas de la base de datos.

8.5 Integración con Blockly

Las páginas maze.html y bird.html cargan los archivos JavaScript originales de Blockly Games ubicados en las carpetas maze/ y bird/. Sobre estos juegos se añadieron:

- **Hooks de inicio de intento**: cuando el jugador entra a un nivel, se ejecuta el código de mazeTryStart.js o birdTryStart.js para registrar el intento en el backend.
- **Hooks de fin de intento**: cuando el juego determina que el nivel fue resuelto, fallado o abandonado, se ejecutan funciones de mazeF.js o birdF.js que envían el estado final y las métricas del intento.
- **Hooks de telemetría**: los eventos de Blockly (por ejemplo, crear, mover, conectar o eliminar bloques) son interceptados y enviados

al backend mediante actionControllers, de forma transparente para el jugador.

De este modo, el frontend conserva la experiencia original de Blockly Games, pero añade la capa de registro de sesiones, intentos y acciones que alimenta el sistema de telemetría y el panel de investigación.

9. Instalación y despliegue

9.1 Requisitos generales

Para ejecutar el sistema se recomienda contar con:

- **Node.js**
  Versión LTS vigente (por ejemplo, 18.x o superior).
- **MySQL**
  Versión reciente y estable (por ejemplo, 8.x o compatible).

Con estas dos herramientas es suficiente para correr backend, frontend y base de datos en una máquina local.

9.2 Código fuente del proyecto

El código completo del sistema se encuentra en el siguiente repositorio:

Repositorio Git:
https://github.com/Christianpa0212/project-blockly.git

Dentro del repositorio:

- El **backend** está en la carpeta:
  server/
- El **frontend** (páginas de juegos y panel de investigación) está en:
  client/
  (principalmente client/public).

9.3 Base de datos

En el proyecto se incluye un **dump completo de MySQL** con:

- todas las tablas necesarias (usuarios, sesiones, intentos, progreso, acciones, etc.),
- catálogos (juegos, niveles, tipos de acción/bloque),
- y vistas de apoyo para el módulo de investigación.

Este archivo puede importarse directamente en cualquier servidor MySQL (Workbench, CLI o gestor equivalente) para crear la base de datos con su estructura y datos iniciales.
No se requieren pasos adicionales más allá de disponer de un servidor MySQL operativo.

9.4 Configuración del servidor (archivo .env)

En la carpeta server/ se incluye un archivo .env de ejemplo con los parámetros básicos:

- Puerto HTTP del servidor Node.
- Host, puerto, usuario, contraseña y nombre de la base de datos MySQL.

Antes de ejecutar el sistema se deben revisar estos valores y ajustarlos a la configuración local (por ejemplo, cambiar usuario y contraseña de MySQL o el puerto de conexión).

El módulo src/config/env/env.js lee estas variables y las expone al resto del backend, por lo que no es necesario modificar el código para cambiar de entorno; basta con editar el .env.

9.5 Ejecución del servidor

Dentro de la carpeta server/, una vez configurado el .env y con la base de datos disponible, el servidor puede ejecutarse de dos formas:

- **Modo desarrollo (con recarga automática):**
- npm run dev
- **Modo producción simple:**
- node server.js

Al ejecutarse, el backend:

- se conecta a la base de datos configurada, y
- sirve el frontend ubicado en client/public, permitiendo acceder tanto al módulo de **jugadores** como al **panel de investigadores** desde el navegador.