|  |  |
| --- | --- |
| **NOMBRE ESTUDIANTE:** | Issac de la Cadena  Carlos Bayas  Kevin Donoso  Andrea Oña  Daniel Oña |
| **FECHA:** | 07-02-2025 |
|  |  |
|  |  |
| **TEMA:** | Documento Diseño Versión Final QualityTech |
|  |  |

**APRENDIZAJE OBTENIDO**

Durante la elaboración del documento de diseño de la versión final del producto, basado en la norma ISO 29110 5-1-2:2024 (Tabla 33), se adquirieron conocimientos clave sobre la estructuración y documentación de los aspectos técnicos de un sistema de software.

Se comprendió la importancia de definir claramente la arquitectura del sistema, los módulos y sus interacciones, garantizando que el producto final cumpla con los requisitos funcionales y no funcionales establecidos. Además, se fortalecieron habilidades en la especificación de detalles como la estructura de datos, la integración con otros sistemas y las consideraciones de seguridad y rendimiento.

**DESARROLLO**

**Conforme a:** ISO 29110 5-1-2:2024, Tabla 33 (WP.16 Software design)

# Introducción

## Breve descripción del juego y su propósito:

Este documento describe el diseño del software para un juego de memoria semántica web, diseñado para evaluar y estimular las capacidades cognitivas de pacientes con Alzheimer. El juego presenta al usuario una serie de palabras que deben ser arrastradas y clasificadas en categorías predefinidas. El juego registra el tiempo, los aciertos y los errores para proporcionar información al terapeuta sobre el desempeño del paciente. El objetivo principal es ofrecer una herramienta digital interactiva y accesible para complementar la terapia tradicional de pacientes con Alzheimer, enfocándose en la memoria semántica.

## Alcance del documento de diseño:

Este documento cubre el diseño arquitectónico (de alto nivel) y detallado (de bajo nivel) del software del juego. Incluye:

* + La arquitectura de tres capas del software.
  + La descripción de los componentes de software.
  + El diseño de la interfaz de usuario (UI).
  + El diseño de la base de datos (MongoDB).
  + La definición de la API REST.
  + El manejo de errores.
  + Consideraciones de rendimiento y seguridad.
  + Las convenciones de nomenclatura.
  + Las consideraciones sobre usabilidad.

Este documento no cubre:

* + El código fuente completo del juego.
  + El plan de pruebas detallado (se describe la estrategia general, pero el plan de pruebas es un documento separado).
  + El manual de usuario (es un documento separado).

## Referencias a otros documentos relevantes:

**Documento de Requisitos:** [2.3 Especificacion-Requerimientos-VersionFinal-QualityTech.docx](https://epnecuador-my.sharepoint.com/:w:/g/personal/carlos_bayas_epn_edu_ec/Efi9Jkie-SVAhkIjyHD5gasBUlbKIJc880Vh2FbuaNNtvg?e=cev9Ip)

**Resultados de la Encuesta SUS y MINIMENTAL:** [2.7 Encuesta SUS y MINIMENTAL.xlsx](https://epnecuador-my.sharepoint.com/:x:/g/personal/carlos_bayas_epn_edu_ec/EdkH0ijOFgROr0Vj5PRKlScBn74QNFaCf8pus-4_g44Qpw?e=LdJbkO)

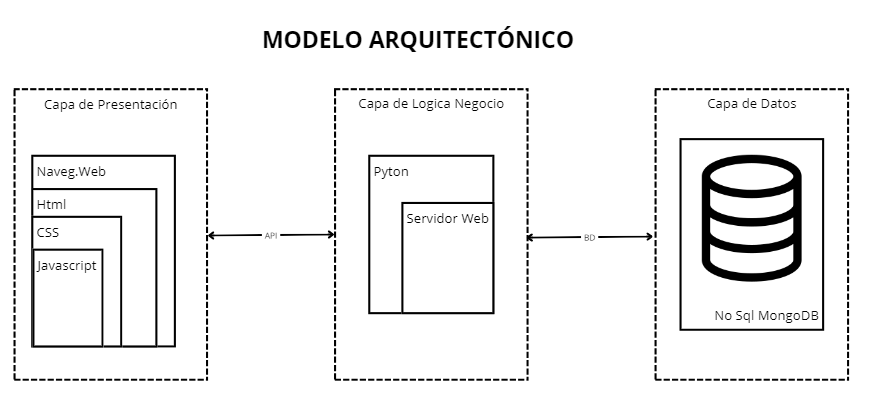
**Plan de Pruebas:** [2.6 Resultados-Pruebas-VersionFinal-QualityTech.docx](https://epnecuador-my.sharepoint.com/:w:/g/personal/carlos_bayas_epn_edu_ec/EXEzcK15p2RGihqSkPhVrdsBcNxQN2f1yVcoCq74E6x86Q?e=IXeXze)

# Diseño Arquitectónico o de Alto Nivel

## Estructura general del software:

El software se basa en una **arquitectura de tres capas**, que separa la presentación (interfaz de usuario), la lógica de negocio y el acceso a datos. Esta arquitectura promueve el modularidad, la mantenibilidad y la escalabilidad.

**Diagrama de Arquitectura (Mejorado):**



**Descripción de las Capas:**

**Capa de Presentación (Frontend):** Es la interfaz de usuario del juego, construida con HTML5, CSS3 y JavaScript (utilizando el framework Vue.js). Se encarga de:

* + Mostrar la información al usuario (pantallas, menús, palabras, categorías, etc.).
  + Capturar la interacción del usuario (clics, arrastrar y soltar).
  + Enviar solicitudes al backend a través de la API REST.
  + Recibir y procesar las respuestas del backend.
  + Validación de entrada del lado del cliente (básica).

**Capa de Lógica de Negocio (Backend):** Implementada en Python utilizando el framework Django (o Flask). Se encarga de:

* + Gestionar la lógica del juego (generación de palabras y categorías, validación de respuestas, cálculo de puntajes, control del tiempo).
  + Exponer una API REST para la comunicación con el frontend.
  + Validación de entrada del lado del servidor (completa).
  + Autenticación y autorización (inicio de sesión de terapeutas).
  + Interacción con la capa de datos (MongoDB).
  + Manejo de errores.

**Capa de Datos:** Utiliza MongoDB, una base de datos NoSQL orientada a documentos. Se encarga de:

* + Almacenar la información del sistema (terapeutas, pacientes, sesiones de juego y resultados, palabras y categorías).
  + Proporcionar mecanismos para acceder y manipular los datos.
  + Garantizar la persistencia de los datos.
  1. **Componentes de software:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Componente | Capa | Función Principal | Entradas | Salidas | Dependencias |
| Vista Inicio de Sesión | Presentación | Permite a los terapeutas autenticarse en el sistema mediante credenciales seguras. Implementa funcionalidades de "Recordar Contraseña" y "Recuperar Contraseña" (si aplica). | correo electrónico (string, formato email), contraseña (string, enmascarada). | Redirección al Panel Principal si la autenticación es exitosa. Mensaje de error específico (credenciales inválidas, usuario no encontrado, etc.). Posiblemente, un token de sesión. | Módulo Autenticación |
| Vista Panel Principal | Presentación | Muestra la lista de pacientes y opciones para agregar/seleccionar pacientes e iniciar el juego. | Lista de pacientes (array de objetos paciente). Opciones de filtrado y ordenamiento (si aplican). | Selección de un paciente (envía el ID del paciente). Redirección a Vista Registro Paciente. Redirección a Vista Juego. Redirección a Vista Feedback. | Módulo Gestión Pacientes |
| Vista Registro Paciente | Presentación | Permite a los terapeutas registrar nuevos pacientes. | nombre (string), apellido (string), edad (número entero, positivo), otros datos relevantes (e.g., diagnóstico, observaciones). | Mensaje de éxito y redirección al Panel Principal. Mensaje de error detallado (campos faltantes, formato incorrecto, etc.). | Módulo Gestión Pacientes |
| Vista Juego | Presentación | Muestra las palabras y categorías, y permite al paciente arrastrar y soltar las palabras en las categorías. | Lista de palabras y categorías (array de objetos palabra, array de objetos categoría). Nivel de dificultad (si aplica). ID del paciente. | Retroalimentación visual inmediata (correcto/incorrecto). Envío de respuestas al backend. Registro del tiempo de inicio y fin. Eventos de interacción (arrastrar, soltar). | Módulo Lógica del Juego |
| Vista Resultados | Presentación | Muestra el tiempo total, los aciertos y errores del paciente. | Resultados del juego (objeto con tiempo total, aciertos, errores, porcentaje de aciertos, etc.). Identificador de la sesión de juego. | Redirección a la pantalla principal. Opción para iniciar una nueva sesión de juego. Opción para ver el feedback. | Módulo Lógica del Juego |
| Vista Feedback | Presentación | Muestra al terapeuta todos los intentos, el tiempo y la fecha de un paciente en específico | Id del Paciente (obtenido en Panel Principal). | Muestra todos los intentos del paciente (con fecha, hora, tiempo, aciertos, errores, etc.). Opción de filtrar los intentos. Redirección al Panel Principal. | Módulo Lógica del Juego, (Posible Módulo de Reportes/Feedback) |
| API REST (Endpoints) | Lógica de Negocio | Proporciona los puntos de acceso para la comunicación entre el frontend y el backend. Endpoints: /api/login, /api/pacientes, /api/pacientes/{id}, /api/juego/iniciar, /api/juego/resultados, etc. | Solicitudes HTTP (GET, POST, PUT, DELETE) con datos en el cuerpo (JSON) y/o parámetros en la URL. | Respuestas HTTP (códigos de estado y datos en formato JSON). | Módulo Autenticación, Módulo Gestión Pacientes, Módulo Lógica del Juego, Módulo Acceso a Datos |
| Módulo Autenticación | Lógica de Negocio | Valida las credenciales del terapeuta. Genera un token de autenticación (JWT, por ejemplo) si la autenticación es exitosa. Gestiona la expiración y renovación de tokens. | correo electrónico (string), contraseña (string). | Booleano (éxito/fallo). Token de sesión (JWT). Información del terapeuta autenticado (ID, rol, etc.). | Módulo Acceso a Datos (para interactuar con la colección terapeutas) |
| Módulo Gestión Pacientes | Lógica de Negocio | Proporciona funciones CRUD para pacientes: crear (validando datos), obtener lista (o filtrar), obtener por ID, actualizar, eliminar (con validaciones). | Objetos de datos (Paciente). | Objetos de datos (Paciente) o mensajes de error. | Módulo Acceso a Datos (para interactuar con la colección pacientes) |
| Módulo Lógica del Juego | Lógica de Negocio | Genera palabras/categorías. Valida respuestas. Calcula tiempo, aciertos y errores. Registra resultados. | ID del paciente, Nivel de dificultad, respuestas del paciente. | Resultados del juego (objeto: tiempo, aciertos, errores) o mensajes de error. | Módulo Acceso a Datos (colecciones palabras categorías y sesiones\_juego), Módulo Palabras y Categorías |
| Módulo Acceso a Datos | Lógica de Negocio | Implementa funciones CRUD para interactuar con MongoDB. Oculta la implementación específica. Convierte entre objetos de la aplicación y documentos de MongoDB. | Objetos de la capa de Lógica (Paciente, Terapeuta, SesionJuego, etc.). | Objetos de la capa de Lógica (datos leídos de la BD) o confirmación/error de las operaciones. | PyMongo |
| Base de Datos (MongoDB) | Datos | Almacena la información de forma persistente. Colecciones: terapeutas, pacientes, sesiones\_juego, palabras\_categorias. | Documentos JSON. | Documentos JSON (resultados de consultas) o confirmaciones de operaciones. | - |
| Palabras y Categorías | Lógica de Negocio/Datos | Gestiona las palabras y las categorías que se utilizaran en el juego (Crear, editar, borrar y asignar dificultad). | id de la palabra o categoría (opcional). Datos para la creación/edición. | palabra, las categorías o la unión de ambas. Mensajes de error. | Base de Datos (colección palabras\_categorias) |

* 1. **Interfaces**

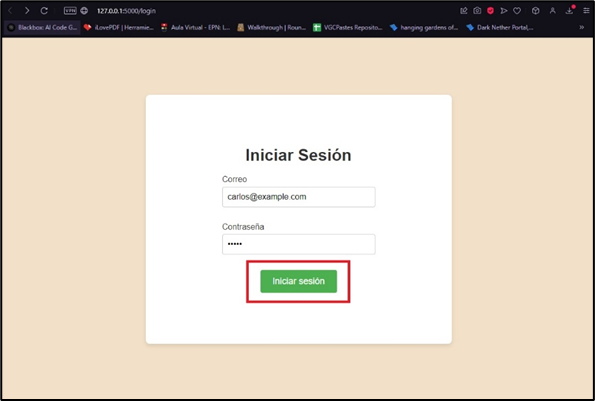
### Interfaces de usuario (HCI):

**Diseño general:** La interfaz del juego seguirá un diseño limpio, intuitivo y accesible, con especial atención a las necesidades de los usuarios con Alzheimer. Se utilizarán:

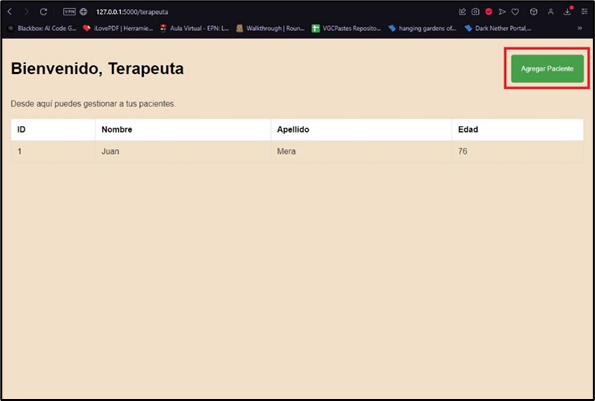
* + - Fuentes grandes y legibles (Arial, Verdana, o similar, al menos 16pt).
    - Alto contraste entre texto y fondo.
    - Botones grandes y claramente etiquetados.
    - Espacios amplios entre elementos para evitar clics accidentales.
    - Instrucciones claras y concisas.
    - Retroalimentación visual y auditiva inmediata.
    - Diseño responsivo (adaptable a diferentes tamaños de pantalla).

**Mockups:** A continuación, se incluyen imágenes a los mockups que vendrían a ser la primera versión de PRODUCTO, incluyendo cada pantalla. Los mockups muestran la disposición de los elementos, y los mockups, la apariencia visual.)

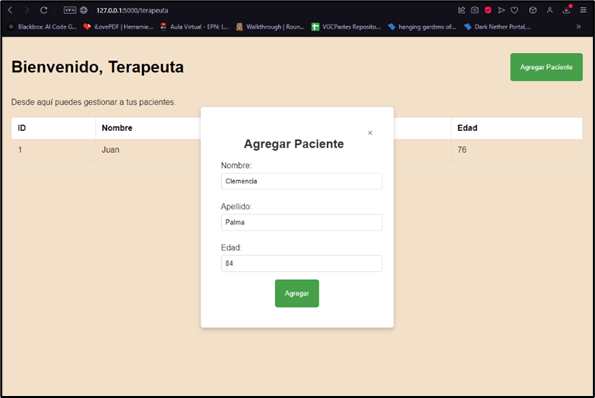
* + - **Inicio de sesión:** Un formulario con campos para correo electrónico y contraseña, y un botón "Iniciar Sesión".



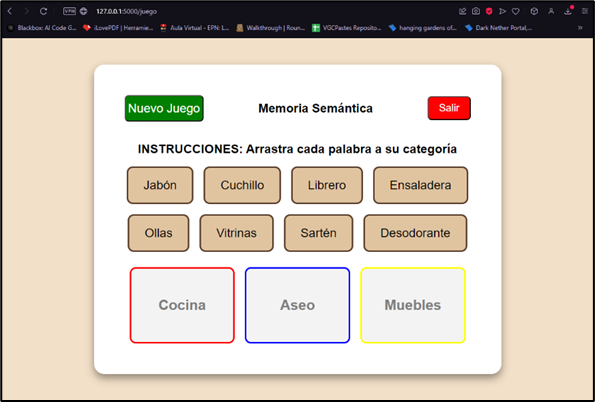
* + - **Panel Principal:** Una lista de pacientes (nombre, apellido, edad) con botones para "Seleccionar" y "Agregar Paciente".



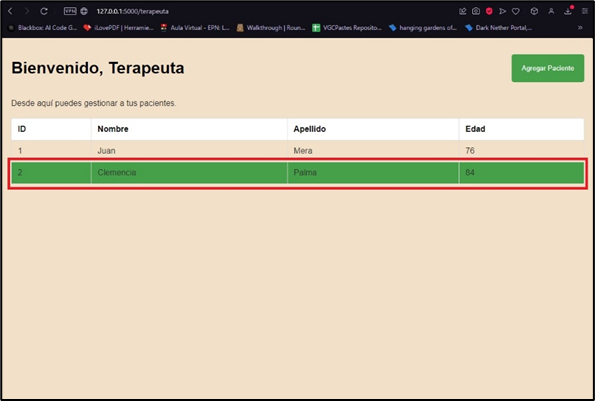
* + - **Registro Paciente:** Campos para nombre, apellido y edad del paciente, y un botón "Guardar".



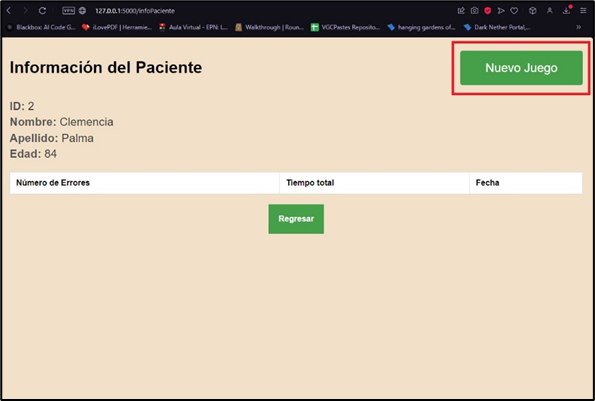
* + - **Juego:** Área superior con las palabras a clasificar (arrastrables). Área inferior con las categorías (destino del arrastre). Indicador de tiempo.



* + - **Resultados:** Presentación del tiempo total, número de aciertos y número de errores. Botones para "Nueva Partida" y "Volver al Inicio".



* + - **Feedback:** Presentación del tiempo total, número de aciertos, número de errores y fecha.



**Interacción del usuario:**

* + - **Inicio de Sesión:** El terapeuta ingresa su correo electrónico y contraseña, y hace clic en "Iniciar Sesión".
    - **Registro de Paciente:** El terapeuta ingresa el nombre, apellido y edad del paciente, y hace clic en "Guardar".
    - **Selección de Paciente:** El terapeuta selecciona un paciente de la lista y hace clic en su nombre.
    - **Inicio del Juego:** El terapeuta hace clic en "Nuevo Juego".
    - **Juego:** El paciente arrastra las palabras a las categorías correspondientes. Recibe retroalimentación visual y/o auditiva inmediata (cambio de color, mensaje, sonido).
    - **Resultados:** Se muestra el tiempo total, los aciertos y errores.

### Interfaces de hardware (si aplica):

No aplica en este caso, ya que es una aplicación web.

### Interfaces de software (si aplica):

**API REST:** La comunicación entre el frontend (Vue.js) y el backend (Django/Flask) se realizará a través de una API REST. Se utilizará el formato JSON para el intercambio de datos.

**Documentación de la API:**

**Herramienta:** Swagger/OpenAPI. (Se incluye un enlace a la documentación de la API generada con Swagger).

**Endpoints:** Se definen los siguientes endpoints (sin ejemplos de código, solo descripción):

* + - * /api/login/ (POST): Autenticación de terapeutas. Recibe correo y contraseña. Devuelve un token de autenticación (si es exitoso) o un mensaje de error.
      * /api/patients/ (GET): Obtener lista de pacientes. Requiere autenticación. Devuelve una lista de objetos, cada uno representando un paciente.
      * /api/patients/ (POST): Crear un nuevo paciente. Requiere autenticación. Recibe los datos del paciente. Devuelve el objeto del paciente creado o un mensaje de error.
      * /api/patients/<int:pk>/ (GET): Obtener detalles de un paciente específico. Requiere autenticación. Devuelve el objeto del paciente.
      * /api/patients/<int:pk>/ (PUT): Actualizar un paciente. Requiere autenticación.
      * /api/patients/<int:pk>/ (DELETE): Eliminar un paciente. Requiere autenticación.
      * /api/game/start/ (POST): Iniciar una nueva sesión de juego. Requiere autenticación. Recibe el ID del paciente y la dificultad. Devuelve una lista de palabras y categorías.
      * /api/game/submit/ (POST): Enviar una respuesta (palabra arrastrada a categoría). Requiere autenticación. Recibe el ID del juego, la palabra y la categoría. Devuelve una indicación de si la respuesta es correcta o incorrecta.
      * /api/game/results/<int: juego\_id>/ (GET): Obtener los resultados de un juego. Requiere autenticación. Devuelve el tiempo, los aciertos y los errores.
      * /api/game/feedback/<int: paciente\_id>/ (GET): Obtener los resultados de todos los intentos de juego de un paciente. Requiere autenticación.
      * /api/words/ (POST): Agrega palabras y categorías.
  1. **Características de rendimiento:**

**Tiempos de Respuesta:**

* + Las solicitudes a la API REST deben responder en menos de 500ms en promedio.
  + La carga de las pantallas debe ser inferior a 2 segundos.

**Uso de Recursos:**

* + Minimizar el uso de memoria y CPU en el cliente (navegador).
  + Optimizar las consultas a la base de datos para minimizar el tiempo de ejecución y el uso de recursos del servidor.

**Estrategias de Optimización:**

* + **Caching:** Implementar caching del lado del cliente (en el navegador) para recursos estáticos (imágenes, CSS, JavaScript). Considerar caching del lado del servidor (en Django/Flask) para datos que no cambian con frecuencia.
  + **Optimización de Consultas a la Base de Datos:** Utilizar índices en MongoDB para acelerar las consultas. Diseñar cuidadosamente el esquema de datos para evitar consultas complejas y costosas.
  + **Minificación y Compresión:** Minimizar y comprimir los archivos JavaScript y CSS para reducir el tamaño de la descarga.
  + Lazy Loading: Cargar los recursos (imágenes, scripts) solo cuando sean necesarios (no precargar todo).
  1. **Características de seguridad:**

**Protección de Datos:**

* + - ***Contraseñas:*** Las contraseñas de los terapeutas se almacenarán *hasheadas* utilizando un algoritmo seguro (bcrypt o Argon2). *Nunca* se almacenará la contraseña en texto plano.
    - ***Datos Sensibles:*** Los datos de los pacientes (nombre, apellido, edad, resultados del juego) se considerarán datos sensibles y se protegerán contra accesos no autorizados.
    - ***Comunicación Segura:*** La comunicación entre el cliente (navegador) y el servidor se realizará a través de HTTPS (TLS/SSL) para cifrar los datos en tránsito y prevenir ataques *man-in-the-middle*.

**Validación de Entrada:**

* + - ***Frontend:* Se** realizará una validación básica en el cliente (navegador) para mejorar la experiencia del usuario y prevenir errores comunes.
    - ***Backend:*** Se realizará una validación *exhaustiva* en el servidor (Django/Flask) para *todas* las entradas del cliente, para prevenir ataques de inyección (SQL injection, NoSQL injection, XSS, CSRF). Se utilizarán las herramientas de validación proporcionadas por el framework (Django Forms o WTForms, por ejemplo).

**Prevención de Ataques:**

* + - ***Cross-Site Scripting (XSS):*** Se escaparán o satirizarán todas las salidas generadas por el servidor para prevenir la inyección de scripts maliciosos.
    - ***Cross-Site Request Forgery (CSRF):*** Se utilizará el middleware de protección CSRF de Django (o equivalente en Flask) para prevenir ataques CSRF.
    - ***Inyección SQL/NoSQL:*** Se utilizarán las herramientas proporcionadas por el framework (ORM de Django o PyMongo) para interactuar con la base de datos de forma segura, evitando la concatenación directa de cadenas en las consultas.
    - Se realizará una correcta gestión de sesiones
  1. **Requisitos de la base de datos (MongoDB):**

**Modelo de Datos (Esquema de Documentos):** Se utilizará MongoDB, una base de datos NoSQL orientada a documentos. Se definen las siguientes colecciones:

* + **terapeutas:**

{

\_id: ObjectId (), // Identificador único (generado por MongoDB)

nombre: String,

apellido: String,

correo: String,

contraseña: String, // Hash de la contraseña (bcrypt o Argon2)

}

* + - * \_id: Clave primaria.
      * correo: Debe ser único.
  + **pacientes:**

{

\_id: ObjectId (),

nombre: String,

apellido: String,

edad: Number,

terapeuta\_id: ObjectId, // Referencia al terapeuta

estadisticas: [ // Array de estadísticas (embebido)

{

juego\_id: ObjectId (), // Identificador único del intento

fecha: Date,

tiempo\_juego: Number, // Tiempo en segundos

numero\_errores: Number,

aciertos: Number,

palabras\_usadas: [String], // Lista de palabras que salieron en ese intento

categorias\_usadas: [String]

}

]

}

* + - * \_id: Clave primaria.
      * terapeuta\_id: Clave foránea que referencia a terapeutas. \_id.
      * Se embeben las estadísticas, para mejorar la consulta de estas.
  + **palabras\_categorias**

{

\_id: ObjectId (),

palabra: String,

categoria: String,

dificultad: String // (ej.: "fácil", "medio", "difícil")

}

* + - * \_id: Clave primaria.
      * palabra: Debe ser único.

**Justificación de la Estructura:**

* + Se utiliza MongoDB por su flexibilidad de esquema, lo que facilita la evolución del modelo de datos a medida que el proyecto avanza.
  + Se embeben las estadísticas dentro del documento del paciente para optimizar las consultas de los resultados del juego (se obtiene toda la información del paciente y sus estadísticas en una sola consulta).
  + Se crea la colección palabras\_categorias de manera separada, ya que es una entidad con vida propia y no depende directamente del resto.

**Índices:** Se crearán los siguientes índices para optimizar las consultas:

* + - terapeutas: Índice único en el campo correo.
    - pacientes: Índice en el campo terapeuta\_id.
    - palabras\_categorias: Índice en el campo palabra.
  1. **Manejo de errores y recuperación:**

**Estrategia General:**

* + - *Frontend:* Se mostrarán mensajes de error claros y amigables al usuario en caso de errores (ej.: error de conexión, datos inválidos). Se implementarán *retries* (reintentos) para peticiones fallidas a la API (con un límite de intentos y un tiempo de espera entre ellos).
    - *Backend:* Se utilizará el mecanismo de manejo de excepciones de Python (try-except) para capturar y manejar errores. Se registrarán los errores en un archivo de log (con detalles del error, fecha, hora, usuario, etc.). Se devolverán códigos de estado HTTP apropiados al frontend (ej.: 400 Bad Request, 401 Unauthorized, 500 Internal Server Error). Se evitará exponer información sensible en los mensajes de error.
    - *Base de datos*: Se realizará una copia de seguridad periódico de la base de datos.

**Mecanismos de Recuperación:**

* + - *Guardado Automático:* Se considerará la implementación de un guardado automático del progreso del juego (en el backend) cada cierto tiempo o después de cada respuesta, para evitar la pérdida de datos en caso de fallo del cliente.
    - *Transacciones:* No se consideran necesarias las transacciones de MongoDB, ya que las operaciones son atómicas a nivel de documento, y la estructura elegida (estadísticas embebidas) minimiza la necesidad de modificar múltiples documentos a la vez.
    - *Backups:* Se realizarán *backups* regulares de la base de datos MongoDB para poder restaurar el sistema en caso de un fallo grave.

# Diseño de Bajo Nivel o Detallado

* 1. **Vista Inicio de Sesión**

### Diseño detallado:

* La vista consistirá en un formulario con dos campos de entrada (correo electrónico y contraseña) y un botón "Iniciar Sesión".
* Se utilizará un diseño simple y claro, con etiquetas descriptivas para los campos.

### Formato de datos de entrada/salida:

**Entrada:** Correo electrónico (string), Contraseña (string).

**Salida:** Redirección a la vista "Panel Principal" (si la autenticación es exitosa), o un mensaje de error (si falla la autenticación).

### Necesidades de almacenamiento de datos: No aplica.

### Convenciones de nomenclatura de datos:

### correo: Para el campo de correo electrónico.

### contraseña: Para el campo de contraseña.

### btnIniciarSesion: Para el botón de inicio de sesión.

### Formato de estructuras de datos: No aplica.

### Campos de datos y su propósito:

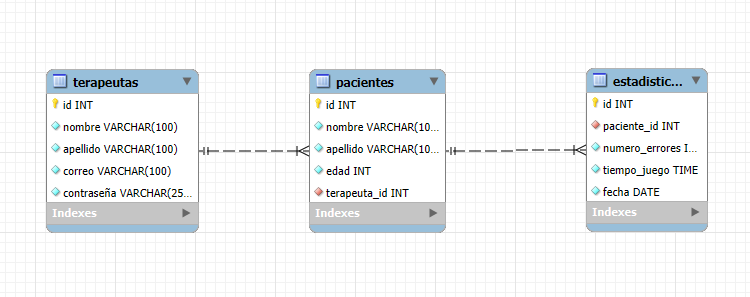
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Entidad | Campo | Tipo de Dato | Tamaño (Estimado) | Propósito | Obligatorio | Consideraciones |
| Terapeuta | \_id | ObjectId | 24 bytes | Identificador único del terapeuta (generado automáticamente por MongoDB). | Sí | Clave primaria. |
| correo | String | 255 | Correo electrónico del terapeuta. Único en el sistema. | Sí | Formato de email válido. Usado para inicio de sesión. |
| contraseña | String | 60-255 | Hash de la contraseña del terapeuta (usar bcrypt, Argon2, etc.). Nunca texto plano. | Sí | Usar una librería de hashing segura. El tamaño depende del algoritmo de hashing. |
| nombre | String | 100 | Nombre del terapeuta. | Sí |  |
| apellido | String | 100 | Apellido del terapeuta. | Sí |  |
| activo | Boolean |  | Indica si el terapeuta está habilitado dentro del sistema. | Sí |  |
| Paciente | \_id | ObjectId | 24 bytes | Identificador único del paciente (generado automáticamente por MongoDB). | Sí | Clave primaria. |
| terapeuta\_id | ObjectId | 24 bytes | Identificador del terapeuta que atiende a este paciente (clave foránea a la entidad Terapeuta). | Sí | Relación uno a muchos (un terapeuta puede tener muchos pacientes). |
| nombre | String | 100 | Nombre del paciente. | Sí |  |
| apellido | String | 100 | Apellido del paciente. | Sí |  |
| edad | Integer | 3 | Edad del paciente. | Sí | Validar que sea un número entero positivo. Considerar un rango razonable (ej., 3-99). |
| diagnostico | String | 255 | Diagnóstico principal del paciente (opcional). | No | Podría ser un campo de texto libre, o podrías tener una tabla separada de diagnósticos predefinidos. |
| observaciones | String | 1000 | Observaciones adicionales sobre el paciente (opcional). | No |  |
| SesionJuego | \_id | ObjectId | 24 bytes | Identificador único de la sesión de juego (generado automáticamente por MongoDB). | Sí | Clave primaria. |
| paciente\_id | ObjectId | 24 bytes | Identificador del paciente que jugó esta sesión (clave foránea a la entidad Paciente). | Sí | Relación uno a muchos (un paciente puede tener muchas sesiones). |
| fecha\_hora\_inicio | DateTime |  | Fecha y hora en que se inició la sesión de juego. | Sí | Importante para el seguimiento y análisis. |
| fecha\_hora\_fin | DateTime |  | Fecha y hora en que finalizó la sesión de juego. | No | Se puede calcular a partir de fecha\_hora\_inicio y duración, pero es útil tenerlo almacenado directamente. Puede ser nulo si la sesión no se completó. |
| duración | Integer (segundos) |  | Duración total de la sesión de juego en segundos. | No | Se podría calcular, pero es útil tenerlo almacenado. |
| aciertos | Integer | 4 | Número total de respuestas correctas. | Sí |  |
| errores | Integer | 4 | Número total de respuestas incorrectas. | Sí |  |
| nivel dificultad | String | 50 | Nivel de dificultad de la sesión (ej., "Fácil", "Medio", "Difícil"). | Sí | Podría ser un valor de una lista predefinida (enum). |
| Palabra | \_id | ObjectId | 24 bytes | Identificador único de la palabra (generado automáticamente por MongoDB). | Sí | Clave Primaria |
| palabra | String | 50 | La palabra que el paciente categorizará. | Sí |  |
| categorías | Array de ObjectId |  | Lista de las categorías a la que pertenece la palabra | Sí |  |
| Categoria | \_id | ObjectId | 24 bytes | Identificador único de la categoria (generado automáticamente por MongoDB). | Sí | Clave Primaria |
| nombre | String | 50 | Nombre de la categoria | Sí |  |
| descripción | String | 255 | Descripción opcional de la categoria | No |  |
| dificultad | String | 10 | Dificultad que se le asigna a la categoria. | No | Valores podrían ser: fácil, media, difícil |
| PalabrasEnSesion | \_id | ObjectId | 24 bytes | Identificador único del registro de palabra en sesión. | Sí | Clave primaria. |
| sesion\_juego\_id | ObjectId | 24 bytes | Identificador de la sesión de juego a la que pertenece este registro (clave foránea a SesionJuego). | Sí | Relación uno a muchos (una sesión puede tener muchas palabras). |
| palabra\_id | ObjectId | 24 bytes | Identificador de la palabra que se mostró al paciente (clave foránea a Palabra). | Sí |  |
| categoria\_correcta\_id | ObjectId | 24 bytes | Identificador de la categoría correcta para esta palabra (clave foránea a Categoria). | Sí |  |
| categoria\_seleccionada\_id | ObjectId | 24 bytes | Identificador de la categoría en la que el paciente colocó la palabra (clave foránea a Categoria, puede ser nulo). | No | Será nulo si el paciente no respondió. Si es diferente de categoria\_correcta\_id, es un error. |
| tiempo\_respuesta | Integer (milisegundos) |  | Tiempo que tardó el paciente en responder (en milisegundos). | No | Puede ser nulo si el paciente no respondió. |

### Especificaciones de la estructura del programa:

* + - La vista se implementará como un componente Vue.js.
    - Se utilizará un formulario HTML estándar.
    - La validación básica se realizará en el cliente (Vue.js) y la validación completa en el servidor (Django/Flask).

# Anexos (opcional)

* Diagramas UML (si se utilizan).



**Revisión y Aprobación**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rol** | **Nombre** | **Firma** | **Fecha** |
| Líder de Diseño (Kevin) |  |  |  |
| Arquitecto de Software |  |  |  |
| Jefe de Proyecto |  |  |  |
| Cliente (si aplica) |  |  |  |

**REFERENCIAS**

1. International Organization for Standardization, *ISO/IEC 29110-5-1-2:2024 - Software Engineering — Lifecycle Profiles for Very Small Entities (VSEs) — Part 5-1-2: Management and Engineering Guide: Generic Profile Group, Table 33*. ISO, 2024.