Escuela
Politécnica
Nacional

Documento de Arquitectura del Software

Grupo D

Averos D, Becerra J, Salazar J, Revelo K

Profesor

Sandra Sánchez PhD.

v3.0-Marzo 2024

INDICE

Directrices de Desarrollo y Estándares

Este tema aborda las convenciones de codificación, estándares de diseño y mejores prácticas que deben seguirse durante el desarrollo del software. En el contexto de nuestra organización, estamos trabajando bajo la norma ISO/IEC/IEEE 12207, que proporciona un marco para el ciclo de vida del software, incluyendo procesos relacionados con la gestión de requisitos, desarrollo, mantenimiento y gestión de la configuración.

Estándares y Convenciones

Estándares de Codificación para el Proyecto:

1. **Sangría:**
   - La sangría del código será gestionada por el IDE utilizado en el proyecto. Se recomienda utilizar una sangría consistente para mejorar la legibilidad del código.
2. **Convenciones de Nomenclatura en Java:**
   - Clases: Utilizar UpperCamelCase. Ejemplo: Calculadora.
   - Métodos y Variables: Utilizar camelCase. Ejemplo: calcularSuma.
   - Constantes: Utilizar mayúsculas con guiones bajos. Ejemplo: MAXIMO_INTENTOS.
3. **Comentarios y Documentación:**
   - Incluir comentarios significativos en el código para explicar el propósito y la lógica detrás de las implementaciones.
   - Utilizar Javadoc para documentar clases y métodos, proporcionando descripciones claras y ejemplos si es necesario.
4. **Convenciones del Lenguaje:**
   - Utilizar el idioma español para todos los comentarios, nombres de variables, clases y métodos. Ejemplo: // Este es un comentario en español.
5. **Importaciones:**
   - Evitar importaciones comodín (por ejemplo, import java.util.*). Importar solo las clases necesarias para reducir la complejidad y mejorar la claridad del código.

Guía para Implementar los Estándares:

1. **Configuración del IDE:**
   - Asegúrate de configurar el IDE para que maneje la sangría automáticamente según las convenciones establecidas.
2. **Nomenclatura Consistente:**
   - Al crear nuevas clases, métodos o variables, sigue las convenciones de nomenclatura establecidas.
   - Revisa y ajusta el código existente para que cumpla con las nuevas convenciones.
3. **Documentación:**
   - Incluye comentarios en el código para explicar la lógica y proporcionar contexto.
   - Utiliza Javadoc para documentar clases y métodos de manera más detallada.
4. **Idioma Español:**
   - Asegúrate de que todos los comentarios y nombres en el código estén escritos en español.
5. **Gestión de Importaciones:**

- Evita importaciones comodín y solo importa las clases necesarias para cada archivo.

Seguimiento de Cambios

En las versiones anteriores del documento (v1 y v2), se denominaba "Documento de Diseño". Sin embargo, en la versión 3, se ha renombrado como "Documento de Arquitectura del Software" para reflejar con mayor precisión su contenido y alcance. Esta modificación se realizó para una mejor alineación con las normativas y prácticas de la industria, así como para una mayor claridad en la estructura del documento.

Arquitectura global del sistema

Diagramas de Componentes

En esta sección se presentarán los diagramas que representan los distintos componentes del sistema y sus interacciones. Se mostrará una vista detallada de cómo están estructurados los módulos y cómo se relacionan entre sí.

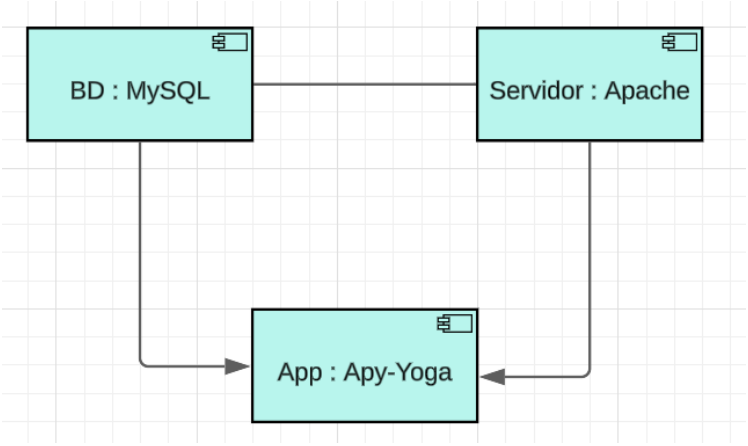


Diagrama de Arquitectura

En esta sección, se proporcionará una visión general de la arquitectura seleccionada y su relevancia para el proyecto. Además, se destacará por qué se ha elegido la arquitectura Modelo-Vista-Controlador (MVC).

Para el presente proyecto se ha optado por esta arquitectura debido a que la misma proporciona una clara separación de responsabilidades entre los componentes del sistema: el Modelo para la gestión de datos y lógica de negocio, la Vista para la presentación de la interfaz de usuario, y el Controlador para la gestión de interacciones y flujo de datos entre Modelo y Vista. Esta separación facilita la mantenibilidad, escalabilidad y reutilización del código, ya que cada componente puede modificarse o actualizarse independientemente sin afectar drásticamente a los otros.

Del mismo modo, en la siguiente imagen se puede apreciar los archivos creados y usados en cada una de las dependencias de la arquitectura Modelo-Vista-Controlador y como se relacionan entre ellos.

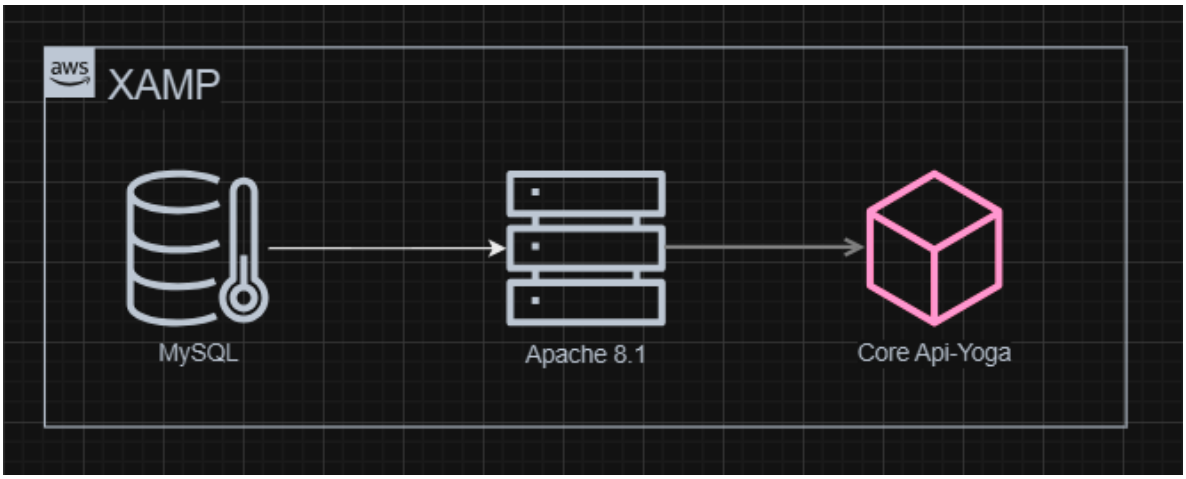
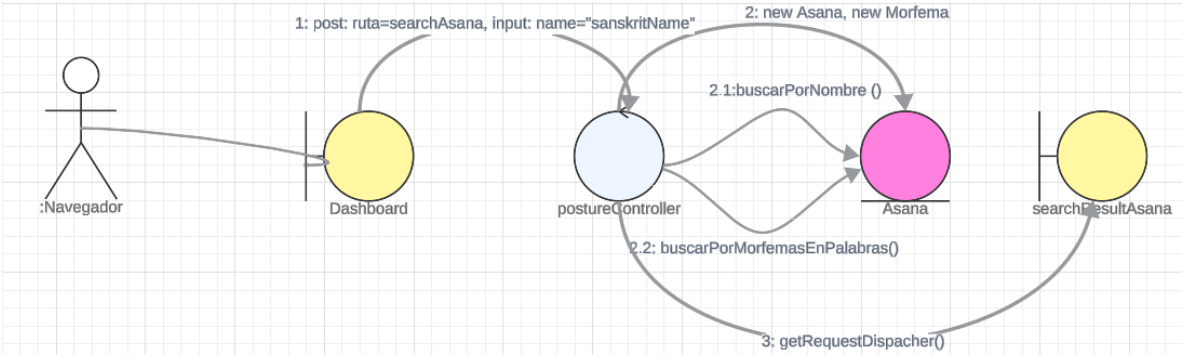


Diagrama de Secuencia de Historias de Usuario Épicas

Estos diagramas ilustrarán las secuencias de interacciones entre los distintos componentes del sistema para cada historia de usuario épica. Se describirán los componentes involucrados, los módulos utilizados y las decisiones de diseño significativas que se han tomado.

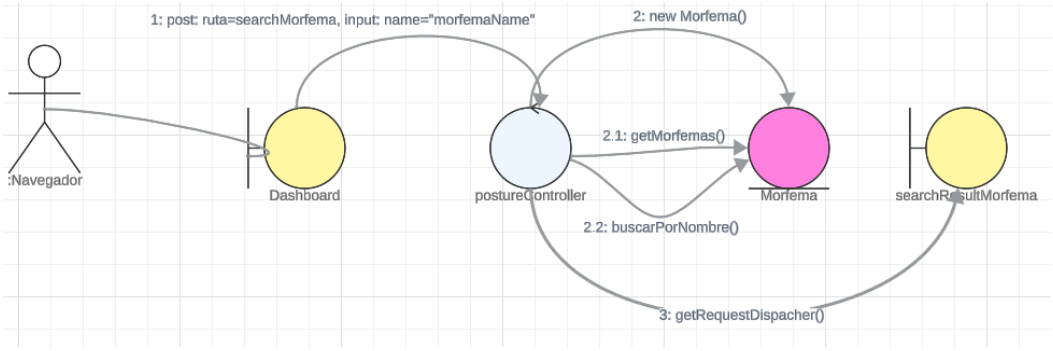
Caso de Uso: Búsqueda por Asana

Se describirá cómo se realiza la búsqueda por asana en el sistema, incluyendo los pasos que realiza el usuario, los componentes involucrados y las interacciones entre ellos.



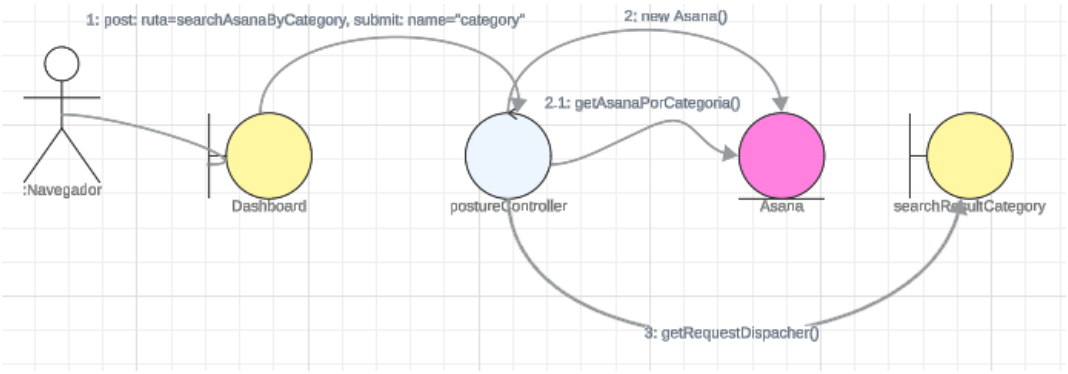
Caso de Uso: Búsqueda por Morfema

Se detallará el caso de uso para la búsqueda por morfema, incluyendo los pasos que sigue el usuario, los módulos utilizados y cómo se lleva a cabo la búsqueda en el sistema.



Caso de Uso: Búsqueda por Categoría

Se describirá el caso de uso para la búsqueda por categoría, destacando los pasos que sigue el usuario, los componentes del sistema implicados y cómo se realiza la búsqueda dentro del sistema.



Patrón de Diseño

El patrón de diseño MVC (Modelo-Vista-Controlador) es una arquitectura de software que separa la representación de la información de la interacción del usuario. Este patrón se utiliza para organizar el código en aplicaciones con interfaces gráficas de usuario y facilita la gestión, mantenimiento y prueba del software. A continuación, se detallan los componentes principales del patrón MVC y cómo se aplican en el diagrama que mencionas:

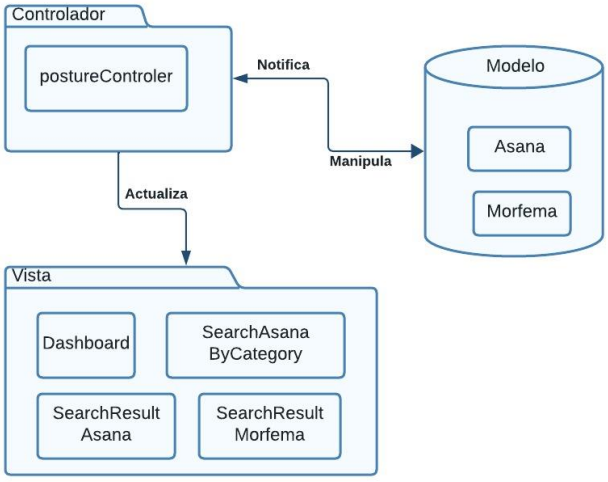
1. **Modelo**: Representa la lógica de negocio y los datos. Es responsable de acceder a la base de datos, definir las reglas de negocio, almacenar y recuperar los estados de la aplicación. En tu diagrama, el "Modelo" contiene "Asana" y "Morfema", que probablemente son entidades o estructuras de datos que la aplicación gestiona.
2. **Vista**: Se encarga de la presentación de los datos y de la interacción con el usuario. La vista observa el modelo y refleja los cambios en la presentación cuando el modelo se actualiza. En el diagrama, la "Vista" incluye "Dashboard", "SearchAsanaByCategory", "SearchResultAsana" y "SearchResultMorfema", que son las diferentes pantallas o componentes de la interfaz de usuario que el usuario ve y con las que interactúa.
3. **Controlador**: Actúa como intermediario entre la Vista y el Modelo. Controla el flujo de datos entre el modelo y las vistas, y maneja los eventos de entrada del usuario, a menudo a través de callbacks o manejadores de eventos. En el diagrama, "postureController" parece ser el controlador que gestiona la lógica de cómo las peticiones de la vista afectan al modelo y viceversa.

Principios de diseño en MVC:

- **Separación de intereses**: Cada parte del patrón MVC se encarga de aspectos distintos de la aplicación, lo que facilita su desarrollo y prueba de manera independiente.
- **Acoplamiento débil**: MVC favorece un acoplamiento débil entre los componentes, lo que significa que las dependencias entre el modelo, la vista y el controlador son mínimas.
- **Principio de responsabilidad única**: Cada componente del MVC se centra en una sola responsabilidad, lo que facilita la gestión del código y reduce la complejidad.

Restricciones técnicas en MVC:

- **Sincronización de la vista y el modelo**: La vista debe actualizarse de manera eficiente cuando el modelo cambia, lo que a veces puede ser un desafío en términos de rendimiento, especialmente en aplicaciones web o móviles grandes y complejas.
- **Gestión de estado**: En aplicaciones con múltiples vistas y estados complejos, gestionar la consistencia del estado a través de las vistas puede ser desafiante.
- **Transiciones de estado y lógica de negocio**: El controlador debe gestionar las transiciones de estado de la aplicación sin convertirse en un punto de alta complejidad y manteniendo la lógica de negocio dentro del modelo.



Riesgos Arquitectónicos

Al utilizar el patrón de diseño MVC en un proyecto de desarrollo de software, algunos riesgos arquitectónicos que pueden surgir son:

1. **Complejidad en el Controlador**: Si no se maneja adecuadamente, el controlador puede volverse muy complejo y acumular mucha lógica que debería estar en el modelo o en la vista, lo que lleva a una violación del principio de responsabilidad única.
2. **Acoplamiento entre Vista y Modelo**: Aunque el patrón MVC promueve un acoplamiento débil, en la práctica puede ser desafiante mantener un bajo acoplamiento, especialmente si la vista requiere múltiples datos del modelo, lo que puede llevar a un acoplamiento no deseado y dificultar las pruebas unitarias.
3. **Rendimiento**: La actualización de las vistas en respuesta a cambios en el modelo puede ser costosa en términos de rendimiento. Si no se maneja correctamente, puede resultar en una interfaz de usuario lenta o congelada, especialmente en aplicaciones con grandes volúmenes de datos o en dispositivos con recursos limitados.
4. **Dificultades en la Gestión de Estado**: Mantener el estado sincronizado entre el modelo y la vista puede ser complicado, especialmente en aplicaciones complejas donde múltiples vistas dependen del mismo modelo.
5. **Rigidez en la Arquitectura**: Aplicar estrictamente el patrón MVC puede llevar a una arquitectura rígida que es difícil de adaptar a requisitos cambiantes o a la incorporación de nuevas tecnologías.
6. **Sobreabstracción**: Intentar ajustarse demasiado al patrón MVC puede llevar a la creación de capas de abstracción innecesarias que agregan complejidad al código sin un beneficio claro.

Para mitigar estos riesgos, es importante:

- Realizar un diseño cuidadoso del controlador para asegurarse de que solo maneje la lógica de flujo y delegación.
- Definir claramente las responsabilidades de cada componente MVC y adherirse a ellas.
- Utilizar patrones adicionales como Observer o Data Binding para manejar las actualizaciones de las vistas de manera eficiente.
- Diseñar un mecanismo de gestión de estado robusto y escalable.
- Estar dispuesto a adaptar o flexibilizar el patrón MVC si el contexto del proyecto lo requiere.
- Evitar la sobreingeniería y mantener la simplicidad en la arquitectura del software.

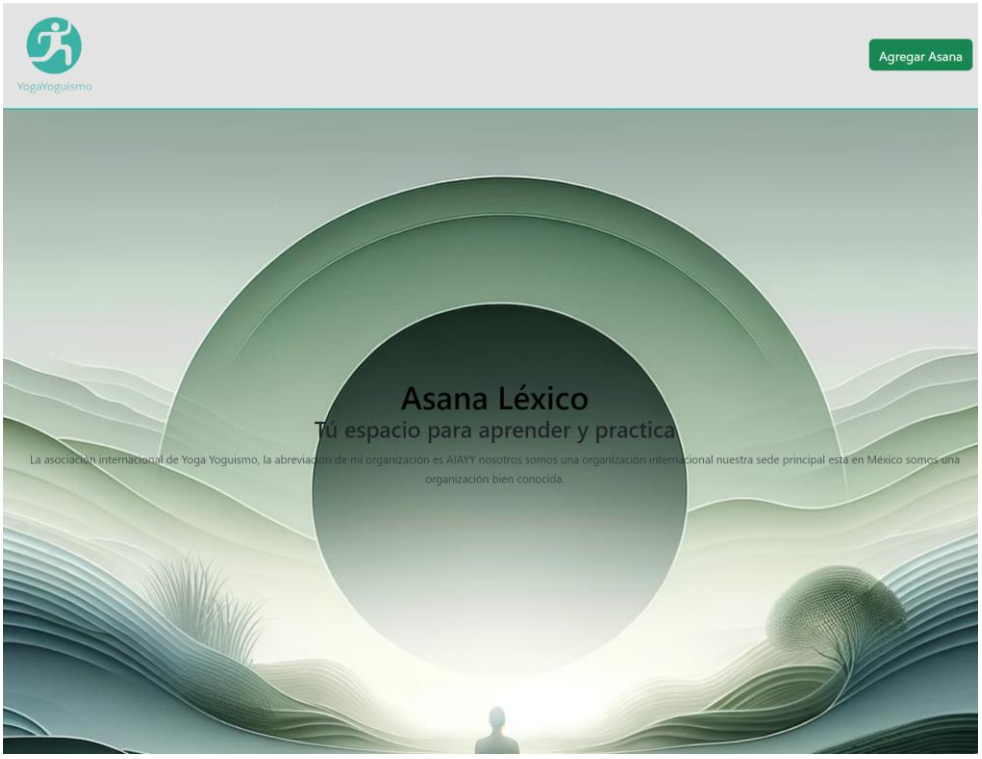
Diseños

Diseño de Interfaz de Usuario

Representación artística

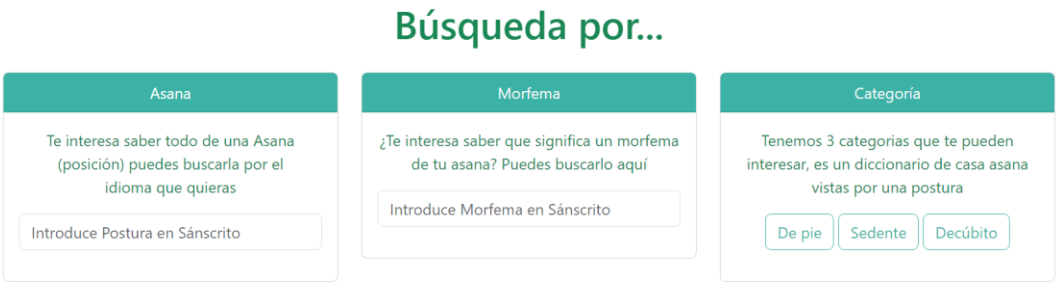
En esta sección, se presenta el diseño detallado de la interfaz de usuario del sistema. Se incluyen los siguientes elementos:

1. **Portada de la Organización:**
    - Se muestra la portada inicial del sistema, que incluye el logotipo y el nombre de la organización. Esta portada proporciona una primera impresión profesional y establece la identidad visual del sistema.



2. **Funcionalidades Principales:**

    - Se presentan las funcionalidades principales del sistema, que son la búsqueda por asana, por morfema y por categoría. Para cada funcionalidad, se incluye una descripción breve y clara de su propósito y cómo se accede a ella desde la interfaz de usuario.



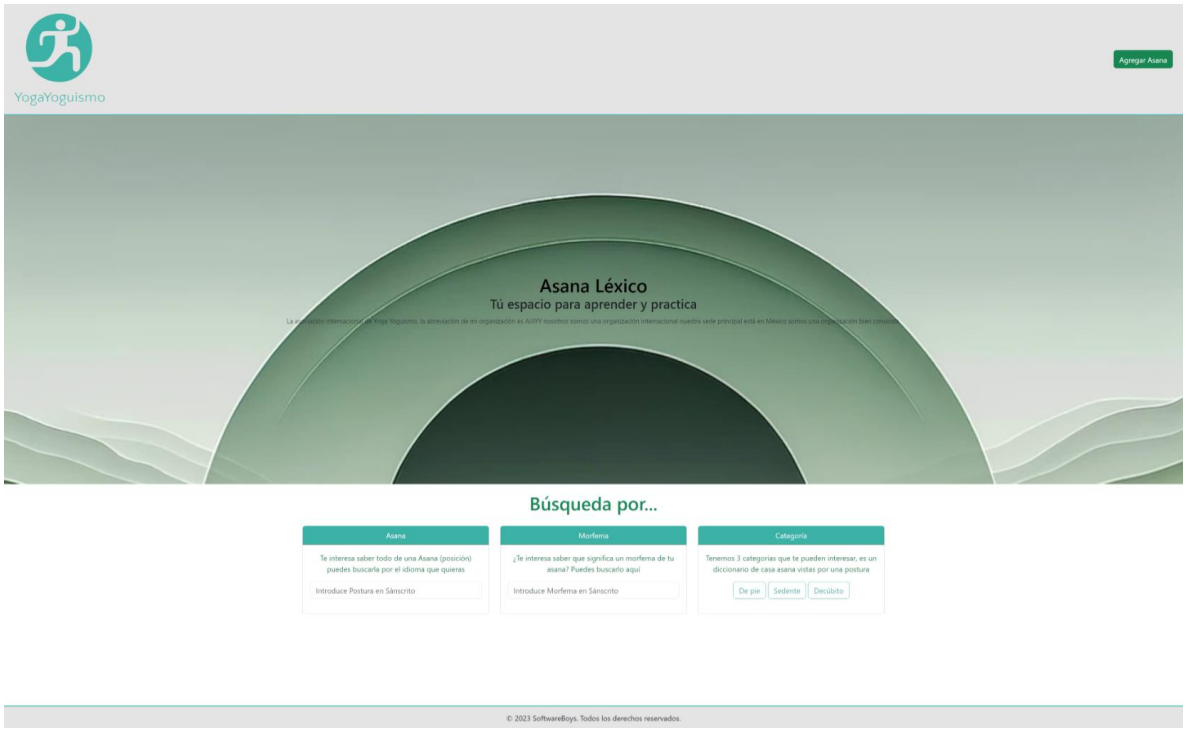
3. **Mockup de la Interfaz:**

    - Se muestra un mockup detallado de la interfaz de usuario, que representa cómo se espera que se vea la pantalla principal del sistema. El mockup incluye elementos como botones, campos de entrada, menús desplegables, etc., que permiten al usuario interactuar con el sistema de manera intuitiva y eficiente.

      El archivo esta en el GitHub

4. **Diseño Responsivo:**

- Se discute cómo se ha diseñado la interfaz de usuario para que sea responsive, es decir, que se adapte automáticamente a diferentes tamaños de pantalla y dispositivos. Se garantiza que el sistema sea accesible y fácil de usar en dispositivos móviles, tabletas y computadoras de escritorio.
- 2560x1571

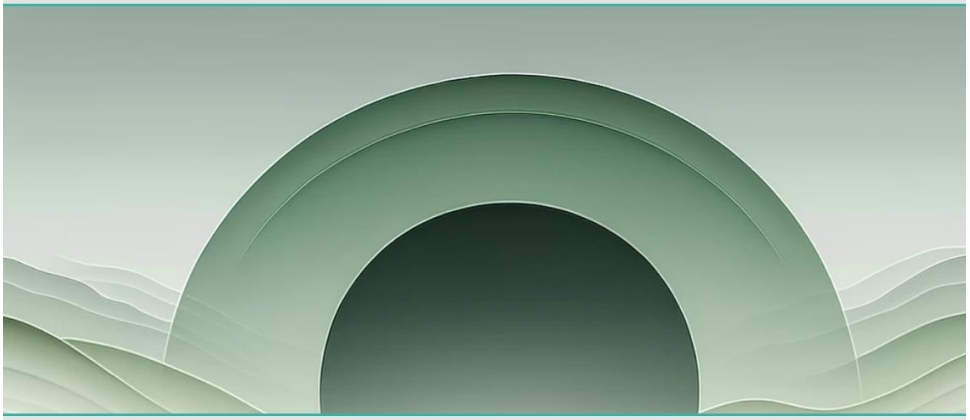


- 1440x942

- Vista 768x471

YogaYoguismo

Agregar Asana



Búsqueda por...

Asana	Morfema	Categoría
Te interesa saber todo de una Asana (posición) puedes buscarla por el idioma que quieras Introduce Postura en	¿Te interesa saber que significa un morfema de tu asana? Puedes buscarlo aquí Introduce Morfema e	Tenemos 3 categorias que te pueden interesar, es un diccionario de casa asana vistas por una postura De pie Sedente Decúbito

- 425x471


Agregar Asana
© 2023 SoftwareBoys. Todos los derechos reservados.
Búsqueda por...
Asana
Te interesa saber todo de una Asana (posición) puedes buscarla por el idioma que quieras
Introduce Postura en Sánscrito
Morfema
¿Te interesa saber que significa un morfema de tu asana? Puedes buscarlo aquí
Introduce Morfema en Sánscrito
Categoría
Tenemos 3 categorias que te pueden interesar, es un diccionario de casa asana vistas por una postura
De pie
Sedente
Decúbito

- 320x471

5. **Feedback del Usuario:**

    - Se describe cómo se ha obtenido feedback del usuario durante el proceso de diseño de la interfaz, y cómo se ha integrado este feedback para mejorar la usabilidad y la experiencia del usuario final. Se destaca la importancia de involucrar a los usuarios en el proceso de diseño para garantizar que el sistema satisfaga sus necesidades y expectativas.

6. **Iteraciones de Diseño:**
    - Se discuten las diferentes iteraciones y versiones del diseño de la interfaz de usuario, explicando cómo ha evolucionado a lo largo del tiempo y qué cambios se han realizado en respuesta al feedback del usuario y a las necesidades del proyecto.

Diseño de Persistencia

Para el diseño de persistencia en el entorno de phpMyAdmin utilizando MySQL como base de datos, se propone la siguiente estructura:

1. **Base de Datos:**
    - Se crea la base de datos llamada **api_yoga** utilizando el script proporcionado.
2. **Tablas:**
    - **Tabla asanas:**
        - Esta tabla almacena información relacionada con las posturas de yoga.
        - Campos:
            - **id**: Identificador único de la postura (clave primaria).
            - **nombreIngles**: Nombre en inglés de la postura.
            - **nombreEsp**: Nombre en español de la postura.
            - **nombreSanscrito**: Nombre en sánscrito de la postura.
            - **imagenRuta**: Ruta de la imagen asociada a la postura.
            - **categoria**: Categoría a la que pertenece la postura (por ejemplo: De pie, Sedentes, Decúbito).
    - **Tabla morfemas:**
        - Esta tabla almacena información relacionada con los morfemas.
        - Campos:
            - **id**: Identificador único del morfema (clave primaria).
            - **nombreMorfema**: Nombre del morfema.
            - **traduccionEsp**: Traducción en español del morfema.
            - **traduccionIngles**: Traducción en inglés del morfema.
3. **Relaciones:**
    - No se han definido relaciones entre las tablas en el diseño proporcionado. Sin embargo, es importante considerar la posibilidad de establecer relaciones, como

por ejemplo, una relación entre las tablas **asanas** y **morfemas** si se necesita relacionar morfemas con posturas específicas.