Logotipo

Descripción generada automáticamente





Contenido

[**1. Introducción 3**](#_heading=h.gjdgxs)

[**2. Propósito 3**](#_heading=h.30j0zll)

[**3. Alcance 4**](#_heading=h.1fob9te)

[**4. Equipo de Trabajo-Actores del Desarrollo 5**](#_heading=h.3znysh7)

[**5. Recomendaciones de conformidad con esta práctica. 6**](#_heading=h.2et92p0)

[**6. Referencias y estándares aplicables a este documento: 7**](#_heading=h.jhkjo3dm4aop)

[**7. Definiciones, acrónimos y abreviaciones. 8**](#_heading=h.17dp8vu)

[**8. DESCRIPCIÓN DE ARQUITECTURA: 9**](#_heading=h.5qgb9itk1hn7)

[8.1 VISTAS: 9](#_heading=h.26in1rg)

[8.2 TIPOS DE VISTAS: 9](#_heading=h.lnxbz9)

[8.3 Framework Conceptual 10](#_heading=h.35nkun2)

[8.3.1 Descripción de la arquitectura en contexto 10](#_heading=h.1ksv4uv)

[8.3.1 Actores Usuarios y sus roles: 13](#_heading=h.44sinio)

[8.3.2 Actividades de arquitectura en el ciclo de vida 14](#_heading=h.2jxsxqh)

[8.3.3 Descripciones prácticas de arquitectura. 14](#_heading=h.z337ya)

[8.4 Documentación de la arquitectura 15](#_heading=h.3j2qqm3)

[**8.5 Selección de puntos de vista de la arquitectura 15**](#_heading=h.1y810tw)

[8.6 Vistas de la arquitectura 16](#_heading=h.4i7ojhp)

[8.7 Consistencia en la cantidad de vistas de la arquitectura. 35](#_heading=h.23ckvvd)

[8.7.1 DESCRIPCIÓN DE MÓDULOS 35](#_heading=h.ihv636)

[8.7.2 DESCRIPCIÓN DE COMPONENTES 35](#_heading=h.32hioqz)

[8.7.3 DESCRIPCIÓN DE CONECTORES 36](#_heading=h.1hmsyys)

[8.7.3.1 Arquitectura lógica 36](#_heading=h.oguje41w2llw)

[8.7.3.2 Performances 36](#_heading=h.kid8qsy7fzwp)

[8.7.3.3 Criterios de Calidad 36](#_heading=h.8curxy7q9weu)

[8.7.3.4 Detalles de la implementación 37](#_heading=h.457m7xr9iw8b)

[8.7.3.5 Lenguajes y plataformas 37](#_heading=h.4ixdoxk18ai8)

[9.1 Acceso a fuentes de información del proyecto 37](#_heading=h.41mghml)

# **Introducción**

El presente informe tiene como objetivo describir en detalle la arquitectura de un sistema móvil orientado a la gestión y monitorización de hábitos, diseñado para apoyar el desarrollo personal de los usuarios. El sistema propuesto busca facilitar la implementación de hábitos de forma intuitiva y organizada, ofreciendo funcionalidades de seguimiento y gestión de los avances personales en los intereses de cada usuario.

Para garantizar una experiencia accesible y amigable, el sistema será implementado en plataformas móviles, permitiendo que los usuarios accedan a sus datos en cualquier momento y lugar. Además, se seguirá un enfoque basado en estándares de diseño, asegurando una interfaz intuitiva y eficiente.

# **Propósito**

El propósito de este documento es servir como guía para el diseño e implementación de la arquitectura del sistema de gestión de hábitos. En él se detallan las decisiones técnicas, los componentes principales y los puntos de integración que permitirán implementar una solución robusta y escalable, orientada a la administración eficiente de hábitos y su seguimiento personalizado para cada usuario.

Este informe es esencial para los desarrolladores y stakeholders del proyecto, ya que proporciona una visión detallada de la estructura y el comportamiento del sistema, así como de las relaciones entre las diferentes funcionalidades y componentes. Además, establece las bases para futuras expansiones y mejoras, con especial atención a factores de disponibilidad, seguridad, escalabilidad y usabilidad.

# Alcance

A continuación, se establecen los alcances contemplados en el proyecto, definiendo los requisitos funcionales y no funcionales, estableciendo los alcances no contemplados.

**Requisitos funcionales:**

* Administración de Hábitos.
* Creación de cuenta e inicio de sesión.
* Cronómetro pomodoro.
* Dashboard de progreso.
* Calendario.

**Requisitos no funcionales:**

* Diseño intuitivo.
* Seguridad de datos de usuario.
* Alto rendimiento en dispositivos móviles.

**Alcances:**

* Desarrollo de la Aplicación: Programar la interfaz de usuario (UI) para la aplicación móvil.
* Integrar un sistema de seguimiento de hábitos. Implementar la funcionalidad de recordatorios automáticos.
* Crear un sistema de retroalimentación visual (gráficos y estadísticas).
* Probar la aplicación para asegurar su funcionamiento y corregir errores.

# 

# Equipo de Trabajo-Actores del Desarrollo

El desarrollo del sistema requiere la participación de un equipo multidisciplinario con roles claramente definidos para asegurar que el proyecto se lleve a cabo de manera eficiente. A continuación, se presenta una tabla con los principales roles y sus responsabilidades:

| **ROL** | **ENCARGAD@** | **RESPONSABILIDADES** |
| --- | --- | --- |
| **PRODUCT OWNER** | Christian Lazcano | Responsable de identificar y priorizar las necesidades del cliente, además de asegurar que el equipo esté enfocado en brindar valor al cliente y alcanzar los objetivos comerciales. |
| **SCRUM MASTER** | Juan Velasquez | Encargado de administrar el proceso Scrum, superar obstáculos y garantizar que el equipo siga los principios y prácticas de Scrum. |
| **DESARROLLADOR BACKEND** | Felipe Martinez | Encargado de desarrollar las funcionalidades internas y código referente a cada acción que realizará la aplicación |
| **DESARROLLADOR FRONTEND** | Antonia Corbalán | Encargada de desarrollar el esquema visual de la aplicación, con cada una de las páginas y elementos visibles de la misma. |
| **QA** | Antonia Corbalán | Encargada de asegurar la calidad del software mediante la planificación y ejecución de pruebas. Identifica errores, verifica que el producto cumpla con los requisitos y estándares establecidos, y asegura que funcione correctamente antes de su lanzamiento. |
| **ADMINISTRADOR BD** | Juan Velasquez | Encargado de la gestión y control de las bases de datos, con respecto a tareas de administración, como lo son asignación de permisos y manejo de tablas |

# Recomendaciones de conformidad con esta práctica.

Para asegurar que la arquitectura del sistema cumpla con los requisitos y estándares establecidos, se recomiendan las siguientes prácticas:

* **Realizar revisiones periódicas de arquitectura** en cada sprint para evaluar el cumplimiento con los requisitos actuales y adaptar el sistema a cambios o mejoras identificadas en las necesidades del usuario y el mercado.
* **Adoptar patrones de diseño desacoplados** como el Modelo-Vista-Controlador (MVC) o la arquitectura por capas, favoreciendo la modularidad y escalabilidad del sistema, lo que facilita la integración de nuevas funcionalidades sin afectar la estabilidad general.
* **Implementar pruebas de calidad de código y de integración continua (CI/CD)**. Automatizar pruebas unitarias, de integración y de regresión permite detectar errores en etapas tempranas del desarrollo y mejora la confiabilidad del sistema.
* **Adoptar metodologías ágiles** como Scrum para promover entregas frecuentes de mejoras y funcionalidades, facilitando la adaptación rápida a cambios de requisitos y asegurando una colaboración continua con los stakeholders.
* **Incorporar prácticas de seguridad y protección de datos** desde el diseño (security by design). Considerar principios de seguridad, como la encriptación y la autenticación robusta, permite que el sistema cumpla con estándares de privacidad y se proteja de amenazas.
* **Fomentar una documentación técnica clara y actualizada** que incluya diagramas de arquitectura, especificaciones de componentes y flujos de datos. Una documentación completa facilita la comprensión y mantenimiento del sistema, especialmente para nuevos miembros del equipo y en futuras fases de expansión.
* **Realizar auditorías de código y revisiones de seguridad periódicas** para identificar vulnerabilidades y áreas de mejora en el código. Estas auditorías, llevadas a cabo por expertos o a través de herramientas de análisis estático, ayudan a mantener la calidad del software y asegurar que cumpla con los estándares de seguridad y rendimiento a lo largo del tiempo.

# Referencias y estándares aplicables a este documento:

El documento sigue los siguientes estándares de calidad para asegurar la robustez y escalabilidad del sistema:

* **IEEE 12207:2017** - **Proceso de ciclo de vida de software**: Define los procesos necesarios para el desarrollo y mantenimiento de software, asegurando una gestión y documentación organizadas y consistentes a lo largo de todo el ciclo de vida del sistema.
* **IEEE 830-1998 ST - Estandarización de requisitos de software**: Proporciona una guía para la especificación de requisitos de software, incluyendo formato, contenido y calidad de los documentos de requisitos.
* **ISO 9126-2001 - Métricas de calidad del software**: Define un modelo para evaluar la calidad del software mediante características como funcionalidad, confiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad.
* **Modelo 4+1 de Kruchten - Modelo estructural basado en múltiples vistas de escenarios**: Organiza la arquitectura del software en cinco vistas diferentes (lógica, de desarrollo, de proceso, física y de casos de uso) para facilitar el diseño y análisis del sistema desde distintos ángulos.
* **ATAM (Architecture Tradeoff Analysis Method) - Metodología para evaluar y documentar decisiones arquitectónicas**: Proporciona un marco sistemático para evaluar decisiones arquitectónicas y sus implicaciones en la calidad del sistema. ATAM permite identificar y analizar riesgos, beneficios y posibles compensaciones (trade-offs) en la arquitectura, asegurando que el sistema cumpla con los requisitos clave de calidad, como rendimiento, seguridad y disponibilidad.

# Definiciones, acrónimos y abreviaciones.

A continuación se explican términos importantes para una mayor compresión del documento.

| **Término** | **Definición** |
| --- | --- |
| **DAS** | Documento de Arquitectura de Software |
| **MVC** | Modelo-Vista-Controlador, patrón de diseño utilizado para separar la lógica de negocio de la interfaz de usuario |
| **UML** | Unified Modeling Language, lenguaje gráfico para representar la arquitectura del sistema |
| **CRUD** | Operaciones básicas de Create, Read, Update, Delete para la gestión de datos. |
| **CI/CD** | Integración Continua y Entrega Continua (Continuous Integration/Continuous Delivery), prácticas que automatizan el proceso de construcción, prueba y despliegue de software, garantizando una entrega rápida y fiable. |

# DESCRIPCIÓN DE ARQUITECTURA:

La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas. La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas.

## VISTAS:

Cada vista se representará en un diagrama UML para describir la estructura y comportamiento del sistema.

## TIPOS DE VISTAS:

1. **Vista de Escenarios (Diagrama de Caso de Uso)**: Describe las funcionalidades del sistema y cómo interactúan los diferentes actores con él.
2. **Vista Lógica (Diagrama de Clases)**: Detalla la estructura de datos y las relaciones entre las principales entidades del sistema.
3. **Vista de Procesos (Diagrama de Actividad)**: Muestra el flujo de actividades dentro del sistema y cómo se coordinan los diferentes procesos.
4. **Vista de Despliegue (Diagrama de Componentes)**: Representa la distribución de componentes en la infraestructura física.
5. **Vista Física (Diagrama Topológico)**: Representa la infraestructura de red y la comunicación entre los diferentes servicios del sistema

## Framework Conceptual

### Descripción de la arquitectura en contexto

El sistema será diseñado y desarrollado utilizando un enfoque ágil con metodologías como Scrum, lo que permitirá la entrega incremental de funcionalidades, la retroalimentación constante y la capacidad de adaptarse a cambios en los requisitos a lo largo del ciclo de vida del proyecto. Esto asegurará que se puedan abordar necesidades emergentes y maximizar el valor del producto para los usuarios.

El diseño del sistema se basará en la separación de vistas, que ofrecerán diferentes perspectivas de la arquitectura para una comprensión integral. A continuación, se describen las principales vistas:

* **Vista de Escenarios**: Representada mediante diagramas de Casos de Uso, que describen las interacciones del usuario con el sistema y los flujos principales.
* **Vista Lógica**: Incluye el Modelo de Clases, que muestra las entidades y relaciones del sistema, así como el Modelo de Datos que detalla la estructura de la base de datos.
* **Vista de Procesos**: Utiliza Diagramas de Actividad para ilustrar los flujos internos del sistema y las interacciones entre los distintos módulos.
* **Vista de Despliegue**: Se implementará a través de Diagramas de Componentes o Modelos de Capas para reflejar la distribución de la lógica de negocio, la interfaz de usuario y la capa de acceso a datos.
* **Vista Física**: Utiliza Diagramas Topológicos o de Servicios para mostrar la distribución del sistema en el entorno físico, como servidores, redes y otros componentes de infraestructura.

Cada una de estas vistas está interconectada para garantizar la coherencia y la alineación con los requisitos de negocio. No hay ninguna vista separada de una implementación específica, sino que todas contribuyen a una visión holística de la arquitectura del sistema descrita en este documento.

Estas vistas se han creado utilizando el Lenguaje de Modelado Unificado (UML) en su versión 2.0 y han sido desarrolladas empleando herramientas de modelado estándar como **StarUML** y Lucidchart y en front-end **Canva, Miro, Figma u otro** para facilitar la creación de diagramas visuales que respalden la documentación arquitectónica.

El estilo arquitectónico se centra en una combinación de patrones de diseño orientados a servicios (SOA) y arquitectura por capas. En este contexto:

**Arquitectura Orientada a Servicios (SOA):**

* **Descripción:** SOA es una estrategia de arquitectura que se basa en la descomposición de un sistema en servicios discretos que se pueden implementar y gestionar de forma independiente. Cada servicio ofrece una funcionalidad específica y expone interfaces para interactuar con otros servicios.
* **Ventajas:** Permite una alta modularidad, facilita la escalabilidad y el mantenimiento, y mejora la reutilización de componentes.

**Arquitectura por Capas:**

* **Descripción:** Esta arquitectura divide el sistema en capas distintas, cada una con responsabilidades específicas. En nuestro sistema, se identifican las siguientes capas:
* **Capa de Presentación:** Maneja la interfaz de usuario y la interacción con el usuario. Utiliza frameworks como React o Angular para la creación de interfaces dinámicas y responsivas.
* **Capa de Lógica de Negocio:** Contiene la lógica central del sistema, implementando las reglas y flujos de negocio. Aquí se desarrollan los módulos para el seguimiento de hábitos, método pomodoro, y análisis de progreso.
* **Capa de Acceso a Datos:** Gestiona la comunicación con la base de datos, utilizando ORM (Object-Relational Mapping) como Entity Framework o Hibernate para facilitar el acceso y manipulación de datos.
* **Capa de Servicios:** Implementa la lógica de negocio como servicios reutilizables que pueden ser consumidos por diferentes aplicaciones y clientes. Aquí se utilizan tecnologías como RESTful APIs o GraphQL.

Cada módulo del sistema se corresponde con un conjunto específico de casos de uso, resolviendo los procesos definidos para la operación. Por ejemplo, el módulo de seguimiento de hábitos se integra con la capa de presentación para ofrecer una interfaz de usuario intuitiva y se comunica con la capa de lógica de negocio para registrar y analizar el progreso del usuario. Del mismo modo, el cronómetro pomodoro interactúa con la capa de servicios para gestionar la temporización y proporcionar notificaciones en tiempo real.

Estos procesos corresponden a:

1. **Gestión de Hábitos:**
   * Interfaz para añadir y editar hábitos.
   * Almacenamiento de hábitos en una base de datos.
2. **Visualización de Gráficos y Estadísticas:**
   * Generación de gráficos de progreso en tiempo real.
   * Visualización de estadísticas detalladas de hábitos.
3. **Recordatorios Automáticos:**
   * Configuración de recordatorios para cada hábito.
   * Notificaciones push para recordar a los usuarios.
4. **Sugerencias de Hábitos Saludables:**
   * Algoritmo para generar sugerencias de hábitos basadas en las preferencias del usuario.
   * Interfaz para mostrar y aceptar/rechazar sugerencias.
5. **Gestión de Tareas:**
   * Interfaz para Añadir y Editar Tareas.
   * Almacenamiento de Tareas en una Base de Datos.
   * Visualización de Tareas.
6. **Preguntas y Retroalimentación sobre Hábitos:**
   * Módulo de preguntas interactivas sobre los hábitos del usuario.
   * Análisis de respuestas y generación de retroalimentación personalizada.
7. **Recomendaciones para Mejorar la Calidad de Vida:**
   * Algoritmo para proporcionar recomendaciones basadas en los hábitos y respuestas del usuario.
   * Interfaz para mostrar recomendaciones y permitir la interacción.
8. **Gestión de la Base de Datos:**
   * Almacenamiento seguro y eficiente de datos de usuario.
   * Mantenimiento y actualización de la base de datos conforme sea necesario.

### Actores Usuarios y sus roles:

Este documento representa la identificación de Actores/Usuarios Stakeholders y sus roles a partir de la interpretación de los casos de uso del Negocio asociados.

| ACTOR | ROL / CARGO | FUNCION |
| --- | --- | --- |
| Usuario | Creador de Hábitos | Registrar y monitorear hábitos diarios, recibir sugerencias y recordatorios, y visualizar estadísticas y gráficos de progreso. |
| Usuario | Profesional Ocupado | Maximizar la productividad, gestionar el tiempo y equilibrar trabajo y vida personal con el seguimiento de hábitos. |
| Usuario | Estudiante Organizado | Planificar y estructurar actividades rutinarias, seguir hábitos de estudio y mejorar calificaciones. |
| Administrador del Sistema | Administrador | Gestionar accesos y permisos de los usuarios, monitorear la salud del sistema, implementar actualizaciones y mejoras, asegurar la seguridad y privacidad de los datos. |
| Equipo Desarrollador | Desarrolladores | Construir y mantener la aplicación, implementar nuevas funcionalidades y mejoras, resolver problemas técnicos, optimizar el rendimiento de la aplicación. |

### 8.3.2 Actividades de arquitectura en el ciclo de vida

El desarrollo de la arquitectura del sistema **Keiko** se abordó utilizando un ciclo de vida **iterativo incremental** basado en **Scrum**. Esta metodología fue seleccionada porque permite realizar entregas parciales y frecuentes de software funcional, facilitando la validación temprana de los requisitos por parte de los stakeholders y el productowner con la adaptación a cambios en el entorno de negocio.

El ciclo de vida se organizó en **sprints** de 1 semana, con cada iteración centrada en la construcción y mejora de un módulo específico del sistema. Al final de cada sprint, se realiza una revisión detallada de los resultados para asegurar la alineación con las expectativas del cliente y se priorizan las tareas para el siguiente sprint.

### 8.3.3 Descripciones prácticas de arquitectura.

Durante el proceso de desarrollo, la arquitectura del sistema fue escalada desde un **escenario principal de caso de uso** hasta un diseño modular a través de los siguientes pasos:

1. **Análisis de Requerimientos**: Se analizaron los requisitos iniciales del sistema para identificar los casos de uso principales, tales como "Administrar Gastos Comunes", "Reservar Espacios Comunes" y "Generar Reportes de Cobros".
2. **Definición de la Vista Lógica**: A partir del escenario principal, se identificaron las entidades y relaciones claves en un diagrama de clases UML, representando el núcleo del sistema.
3. **Modelo de Datos**: Se desarrolló un diagrama de base de datos para definir la estructura de la información, manteniendo la coherencia con la vista lógica.
4. **Vista de Procesos**: Se modelaron los flujos de las actividades y transacciones principales del sistema mediante diagramas de actividad.
5. **Despliegue**: Se definió la infraestructura física y lógica para alojar el sistema en la nube, detallando los servicios necesarios, como balanceadores de carga, servidores de aplicaciones y bases de datos distribuidas.
6. **Definición de Planes de pruebas y técnicas de pruebas testing**.

## Documentación de la arquitectura

* **ISO/IEC 25010**: Se utilizó para evaluar la calidad del sistema en términos de funcionalidad, seguridad y eficiencia.
* **IEEE 1471-2000**: Proporcionó la guía para la representación de vistas arquitectónicas y la identificación de stakeholders.
* **Modelo 4+1 de Kruchten**: Estructura la arquitectura en vistas lógicas, de desarrollo, de procesos y física, facilitando la visualización desde múltiples perspectivas.

## Selección de puntos de vista de la arquitectura

| VISTAS | UML | Cantidad |
| --- | --- | --- |
| Escenario Principal | Diagrama de Caso de uso | 1 (2-3 los necesarios) |
| Vista Lógica | Diagrama de Clases  MER | 1  1 |
| Vista de Proceso | Diagrama de Actividad:  Nombre 1.  Nombre 2  Nombre 3. | 1 (2-3, los necesarios) |
| Vista Física | Diagrama Topológico de Servicios | 1 |
| Vista de Despliegue | Diagrama de componentes  Diagrama de Capas (MVC) | 1  1 |

## Vistas de la arquitectura

| UML-VISTA DE ESCENARIO |
| --- |
| Diagrama de Caso de uso extendido /Subproyecto GESTIÓN DE HÁBITOS |
|  |

* **LOGIN:** Identificar usuario por ID y contraseña.
* **INICIO:** Acceder a la página principal después de iniciar sesión.
* **VER HÁBITOS:** Mostrar hábitos existentes.
* **CREAR HÁBITO:** Añadir un nuevo hábito, definiendo nombre, fecha de inicio, fecha de término, descripción, categoría e ícono.
* **EDITAR HÁBITO:** Modificar un hábito existente, incluyendo nombre, fechas, descripción, categoría e ícono.
* **ELIMINAR HÁBITO:** Borrar un hábito, mostrando un mensaje de confirmación antes de hacerlo.

| UML-VISTA LÓGICA-DIAGRAMA DE CLASES |
| --- |
| [Diagrama de Clases] |
|  |

| UML-VISTA LÓGICA-DIAGRAMA DE BASE DE DATOS |
| --- |
| [Diagrama de BD - MER] |
|  |

**2. Elementos Principales**

**Entidades:**

* **PLANES**

**Atributos:**

* id\_plan (PK)
* nombre\_plan
* descripcion
* es\_premiun
* **CATEGORIA**

**Atributos:**

* id\_categoria (PK)
* nombre\_categoria
* descripcion
* icono
* color\_icono
* **FRECUENCIAS**

**Atributos:**

* id\_frecuencia (PK)
* tipo\_frecuencia
* frecuencia
* descripcion
* **USUARIO**

**Atributos:**

* id\_usuario (PK)
* nickname
* email
* contraseña
* fecha\_registro
* fecha\_nacimiento
* id\_plan (FK)
* **HABITO\_PREDETERMINADO**

**Atributos:**

* id\_habito\_preder (PK)
* nombre\_habito\_preder
* descripcion
* id\_categoria (FK)
* **HABITO**

**Atributos:**

* id\_habito (PK)
* nombre\_habito
* descripcion
* icono
* color\_icono
* id\_habito\_preder (FK)
* id\_categoria (FK)
* **USUARIO\_HABITO**

**Atributos:**

* id\_usuario\_habito (PK)
* id\_usuario (FK)
* id\_habito (FK)
* id\_frecuencia (FK)
* fecha\_inicio
* fecha\_fin
* estado
* **DIAS**

**Atributos:**

* id\_dia\_habito (PK)
* id\_usuario\_habito (FK)
* nombre\_dia
* **TAREAS**

**Atributos:**

* id\_tarea (PK)
* nombre\_tarea
* descripcion
* fecha
* estado
* icono
* color
* id\_usuario (FK)
* observaciones
* **TIPO\_COLABORADOR**

**Atributos:**

* id\_tipocolaborador (PK)
* tipo\_colaborador
* **COLABORADOR**

**Atributos:**

* id\_colaborador (PK)
* id\_tipocolaborador (FK)
* id\_usuario (FK)
* rut
* primer\_nombre
* segundo\_nombre
* primer\_apellido
* segundo\_apellido
* fecha\_nacimiento
* pais
* **PERMISO\_COLABORADOR**

**Atributos:**

* id\_permiso (PK)
* id\_tip
* **RECOMENDACIONES**

**Atributos:**

* id\_recomendacion (PK)
* contenido
* fecha\_hora
* estado
* id\_usuario (FK)
* **INTERACCIONES**

**Atributos:**

* id\_interaccion (PK)
* id\_usuario (FK)
* tipo\_interaccion
* fecha\_hora
* descripcion
* **HISTORIAL\_HABITOS**

**Atributos:**

* id\_historial (PK)
* id\_usuario\_habito (FK)
* fecha
* estado
* observaciones
* **NOTIFICACIONES**

**Atributos:**

* id\_notificacion (PK)
* id\_usuario (FK)
* hora\_notificacion
* fecha\_notificacion
* mensaje
* estado
* tipo\_notificacion
* id\_frecuencia (FK)

**Relaciones:**

* **USUARIO <-> PLANES**
  + **Relación:** USUARIO (id\_plan) referencia a PLANES (id\_plan)
  + **Tipo de Relación:** Muchos a Uno
* **HABITO\_PREDETERMINADO <-> CATEGORIA**
  + **Relación:** HABITO\_PREDETERMINADO (id\_categoria) referencia a CATEGORIA (id\_categoria)
  + **Tipo de Relación:** Muchos a Uno
* **HABITO <-> HABITO\_PREDETERMINADO**
  + Relación: HABITO (id\_habito\_preder) referencia a HABITO\_PREDETERMINADO (id\_habito\_preder)
  + Tipo de Relación: Muchos a Uno
* **HABITO <-> CATEGORIA**
  + **Relación:** HABITO (id\_categoria) referencia a CATEGORIA (id\_categoria)
  + **Tipo de Relación:** Muchos a Uno
* **USUARIO\_HABITO <-> USUARIO**
  + **Relación:** USUARIO\_HABITO (id\_usuario) referencia a USUARIO (id\_usuario)
  + **Tipo de Relación:** Muchos a Uno
* **USUARIO\_HABITO <-> HABITO**
  + **Relación:** USUARIO\_HABITO (id\_habito) referencia a HABITO (id\_habito)
  + **Tipo de Relación:** Muchos a Uno
* **USUARIO\_HABITO <-> FRECUENCIAS**
  + **Relación:** USUARIO\_HABITO (id\_frecuencia) referencia a FRECUENCIAS (id\_frecuencia)
  + **Tipo de Relación:** Muchos a Uno
* **DIAS <-> USUARIO\_HABITO**
  + **Relación:** DIAS (id\_usuario\_habito) referencia a USUARIO\_HABITO (id\_usuario\_habito)
  + **Tipo de Relación:** Muchos a Uno
* **TAREAS <-> USUARIO**
  + **Relación:** TAREAS (id\_usuario) referencia a USUARIO (id\_usuario)
  + **Tipo de Relación:** Muchos a Uno
* **COLABORADOR <-> TIPO\_COLABORADOR**
  + **Relación:** COLABORADOR (id\_tipocolaborador) referencia a TIPO\_COLABORADOR (id\_tipocolaborador)
  + **Tipo de Relación:** Muchos a Uno
* **COLABORADOR <-> USUARIO**
  + **Relación:** COLABORADOR (id\_usuario) referencia a USUARIO (id\_usuario)
  + **Tipo de Relación:** Muchos a Uno
* **PERMISO\_COLABORADOR <-> TIPO\_COLABORADOR**
  + **Relación:** PERMISO\_COLABORADOR (id\_tipocolaborador) referencia a TIPO\_COLABORADOR (id\_tipocolaborador)
  + **Tipo de Relación:** Muchos a Uno
* **RECOMENDACIONES <-> USUARIO**
  + **Relación:** RECOMENDACIONES (id\_usuario) referencia a USUARIO (id\_usuario)
  + **Tipo de Relación:** Muchos a Uno
* **INTERACCIONES <-> USUARIO**
  + **Relación:** INTERACCIONES (id\_usuario) referencia a USUARIO (id\_usuario)
  + **Tipo de Relación:** Muchos a Uno
* **HISTORIAL\_HABITOS <-> USUARIO\_HABITO**
  + **Relación:** HISTORIAL\_HABITOS (id\_usuario\_habito) referencia a USUARIO\_HABITO (id\_usuario\_habito)
  + **Tipo de Relación:** Muchos a Uno
* **NOTIFICACIONES <-> USUARIO\_HABITO**
  + **Relación:** NOTIFICACIONES (id\_usuario\_habito) referencia a USUARIO\_HABITO (id\_usuario\_habito)
  + **Tipo de Relación:** Muchos a Uno
* **NOTIFICACIONES <-> FRECUENCIAS**
  + **Relación:** NOTIFICACIONES (id\_frecuencia) referencia a FRECUENCIAS (id\_frecuencia)
  + **Tipo de Relación:** Muchos a Uno

| UML-VISTA DE PROCESO-DIAGRAMAS DE ACTIVIDAD |
| --- |
| [Diagramas de Actividad] |
|  |

**Componentes**:

1. **Inicio:**
   * El usuario abre la aplicación e inicia sesión.
2. **Registro de Hábitos:**
   * El usuario accede a la sección de "Crear Hábito".
   * El usuario ingresa detalles del hábito (nombre, descripción, fechas, categoría, ícono).
   * El sistema guarda el hábito en la base de datos.
   * El sistema confirma que el hábito ha sido creado exitosamente.
3. **Visualización de Hábitos:**
   * El usuario selecciona "Ver Hábitos".
   * El sistema muestra la lista de hábitos registrados.
4. **Gestión de Tareas:**
   * El usuario accede a la sección "Crear Tarea".
   * El usuario ingresa detalles de la tarea (nombre, descripción, fecha, hora, prioridad).
   * El sistema guarda la tarea en la base de datos.
   * El sistema confirma que la tarea ha sido creada exitosamente.
5. **Configuración de Recordatorios:**
   * El usuario configura recordatorios para hábitos o tareas.
   * El sistema programa las notificaciones push.
6. **Recepción de Notificaciones:**
   * El usuario recibe notificaciones push según los recordatorios configurados.
7. **Sugerencias de Hábitos Saludables:**
   * El sistema analiza las preferencias del usuario.
   * El sistema genera sugerencias de hábitos saludables.
   * El usuario acepta o rechaza las sugerencias.
8. **Establecimiento de Metas y Hábitos Personalizados:**
   * El usuario accede a la sección de "Establecer Metas".
   * El usuario define y guarda sus metas y hábitos personalizados.
   * El sistema integra estas metas con estadísticas y recordatorios.
9. **Preguntas y Retroalimentación sobre Hábitos:**
   * El usuario responde preguntas interactivas sobre sus hábitos.
   * El sistema analiza las respuestas y genera retroalimentación personalizada.
10. **Recomendaciones para Mejorar la Calidad de Vida:**
    * El sistema analiza hábitos y respuestas del usuario.
    * El sistema genera recomendaciones personalizadas.
    * El usuario visualiza y aplica las recomendaciones.
11. **Gestión de la Base de Datos:**
    * El sistema almacena y mantiene datos de usuario de manera segura.
    * El sistema realiza mantenimiento y actualizaciones según sea necesario.
12. **Salir de la App:**

* El usuario cierra sesión y sale de la aplicación.

| UML-VISTA DE DESPLIEGUE |
| --- |
| Sistema Keiko |
|  |

El diagrama de despliegue muestra la arquitectura de tu aplicación de gestión y monitoreo de hábitos, ilustrando cómo los componentes se distribuyen en diferentes entornos y cómo interactúan entre sí. A continuación se describen los componentes y sus interacciones:

1. **Dispositivo Móvil del Usuario:**
   * Aplicación Móvil: La aplicación instalada en el dispositivo del usuario para registrar y monitorear hábitos.
   * Notificaciones Push: Servicio que permite enviar notificaciones al dispositivo móvil del usuario para recordar hábitos y tareas.
2. **Servidores en la Nube:**
   * Servidor de la Aplicación: Maneja la lógica de la aplicación, procesa las solicitudes de los usuarios y se comunica con otros componentes.
   * API Gateway: Actúa como un punto de entrada para todas las solicitudes externas hacia los servicios internos, garantizando la seguridad y gestión del tráfico.
   * Servidor de autenticación: Gestiona la autenticación de los usuarios, asegurando que solo usuarios autorizados accedan a la aplicación.
   * Interfaz de Base de Datos: Se comunica con la base de datos para realizar operaciones de almacenamiento y recuperación de datos.
3. **Servicios Externos:**
   * Servicio de Notificaciones Push: Servicio externo que facilita el envío de notificaciones push a los dispositivos móviles de los usuarios.
   * Servicios de IA/ML: Proveen funcionalidades adicionales como la generación de sugerencias y recomendaciones basadas en los hábitos y preferencias del usuario.
4. **Base de Datos:**
   * Firebase: Base de datos utilizada para almacenar y gestionar los datos de la aplicación, como los hábitos, tareas, usuarios y configuraciones.

**Interacciones y Flujo de Datos**

1. **Interacción Usuario-Aplicación:**
   * El usuario interactúa con la aplicación a través de su dispositivo móvil, enviando y recibiendo datos a través de la red.
2. **Comunicación Frontend-Backend:**
   * La aplicación móvil envía solicitudes al servidor de la aplicación, que procesa las solicitudes y devuelve las respuestas necesarias.
3. **Comunicación Backend-Base de Datos:**
   * El servidor de la aplicación se comunica con Firebase para almacenar y recuperar datos, como información de usuarios, hábitos y tareas.
4. **Servicios de Notificaciones Push:**
   * El servidor de la aplicación utiliza el servicio de notificaciones push para enviar recordatorios y notificaciones a los dispositivos móviles de los usuarios.
5. **Servicios de IA/ML:**
   * El servidor de la aplicación se comunica con servicios de IA/ML para proporcionar funcionalidades avanzadas, como la generación de sugerencias y recomendaciones personalizadas.

| UML-VISTA FÍSICA – DIAGRAMA INFRAESTRUCTURA |
| --- |
| Sistema Gestión De Condominios |
|  |

El diagrama muestra el flujo de infraestructura de una aplicación móvil desde su descarga hasta la sincronización de datos en tiempo real.

* **Google Play:** Los usuarios descargan la aplicación desde Google Play.
* **Usuarios:** Los usuarios instalan y usan la aplicación en sus dispositivos móviles.
* **Conexión API:** La aplicación se conecta a una API para interactuar con el backend.
* **OAuth2.0 (Autenticación):** Se utiliza OAuth2.0 para la autenticación de los usuarios, validando su identidad.
* **SSL/TLS (Cifrado de Datos):** La comunicación se asegura mediante SSL/TLS para cifrar los datos.
* **Backend (Node.js):** El backend, desarrollado en Node.js, maneja los datos recibidos.
* **Firebase (Almacenamiento y Sincronización de Datos):** Los datos se almacenan y sincronizan en tiempo real utilizando Firebase.
* **SQLite (Base de Datos Local):** Finalmente, los datos se sincronizan en una base de datos local SQLite.

| Usabilidad y Calidad de software- Front-end -Vistas Principales |
| --- |
| Vistas principales del sistema – Vista Home Principal de entrada |
|  |

| Usabilidad y Calidad de software – Vistas específicas de usuario |
| --- |
| Vistas principales del sistema – Vista Principal de Usuario a escritorio de trabajo principal de su componente de sistema |
|  |

## Consistencia en la cantidad de vistas de la arquitectura.

Se garantiza que cada vista esté alineada con los requisitos funcionales y no funcionales del sistema, evitando redundancias y manteniendo la coherencia en toda la arquitectura

### 

### DESCRIPCIÓN DE MÓDULOS

| Nombre del módulo | Descripción | Componentes incluidos |
| --- | --- | --- |
| Gestión de Usuarios | Maneja el registro, autenticación y gestión de perfiles de usuarios, incluyendo roles y permisos. | * Registro de Usuarios * Autenticación y Autorización * Perfil de Usuario * Administración de Roles y Permisos |
| Gestión de Hábitos | Permite a los usuarios crear, editar, eliminar y organizar sus hábitos diarios, incluyendo sugerencias predeterminadas. | * Creación y Edición de Hábitos * Eliminación de Hábitos * Clasificación de Hábitos por Categorías * Sugerencias de Hábitos Predeterminados |
| Gestión de Tareas | Facilita la creación, edición y organización de tareas, con opciones de recordatorio. | * Creación y Edición de Tareas * Eliminación de Tareas * Recordatorios de Tareas * Integración de Tareas con el Calendario |
| Seguimiento y Visualización | Proporciona herramientas para monitorear el progreso de los hábitos mediante gráficos y estadísticas detalladas. | * Monitoreo de Progreso de Hábitos * Visualización de Estadísticas y Gráficos * Registro de Actividades diarias |
| Recordatorios y Notificaciones | Configura y envía notificaciones push y recordatorios automáticos para hábitos y tareas. | * Configuración de Recordatorios * Envío de Notificaciones Push * Gestión de Frecuencias de Notificaciones |
| Sugerencias y Recomendaciones | Genera sugerencias personalizadas basadas en los datos y comportamientos del usuario utilizando IA/ML. | * Algoritmos de IA/ML para Generación de Sugerencias * Interfaz de Usuario para Mostrar Sugerencias * Sistema de Recomendaciones Basadas en el Comportamiento del Usuario |
| Administración | Proporciona herramientas para la gestión de usuarios, roles y monitoreo del sistema a nivel administrativo. | * Panel de Administración * Gestión de Usuarios y Roles * Monitoreo del Sistema y Logs de Actividades |
| Interfaz de Usuario | Diseña y desarrolla la interfaz de usuario, asegurando una experiencia intuitiva y responsiva en múltiples dispositivos. | * Diseño de la Interfaz de Usuario * Implementación de Wireframes y Prototipos * Asegurar la Responsividad y Usabilidad en Diferentes Dispositivos |

### DESCRIPCIÓN DE COMPONENTES

| Nombre del Componente | Descripción | Componentes Relaciones |
| --- | --- | --- |
| Registro de Usuarios | Permite a los nuevos usuarios registrarse en la aplicación proporcionando sus datos personales y de acceso. | Módulo de Gestión de Usuarios, Módulo de Seguridad. |
| Autenticación y Autorización | Implementa un sistema de autenticación seguro que permite a los usuarios iniciar sesión y proteger sus cuentas. | Módulo de Gestión de Usuarios, Módulo de Seguridad. |
| Perfil de Usuario | Gestiona la información del perfil del usuario, incluyendo detalles personales y preferencias. | Módulo de Gestión de Usuarios. |
| Administración de Roles y Permisos | Permite asignar y gestionar roles y permisos a los usuarios, asegurando que solo los usuarios autorizados tengan acceso a ciertas funcionalidades. | Módulo de Gestión de Usuarios, Módulo de Seguridad. |
| Creación y Edición de Hábitos | Permite a los usuarios crear y editar hábitos, especificando detalles como nombre, descripción, fechas y categoría. | Módulo de Gestión de Hábitos, Módulo de Sugerencias y Recomendaciones. |
| Eliminación de Hábitos | Permite a los usuarios eliminar hábitos que ya no desean seguir. | Módulo de Gestión de Hábitos. |
| Clasificación de Hábitos por Categorías | Organiza los hábitos en categorías específicas para una mejor gestión y seguimiento. | Módulo de Gestión de Hábitos. |
| Sugerencias de Hábitos Predeterminados | Proporciona una lista de hábitos predeterminados basados en categorías para ayudar a los usuarios a comenzar. | Módulo de Gestión de Hábitos, Módulo de Sugerencias y Recomendaciones. |
| Monitoreo de Progreso de Hábitos | Rastrea el progreso diario, semanal y mensual de los hábitos del usuario. | Módulo de Seguimiento y Visualización. |
| Visualización de Estadísticas y Gráficos | Genera gráficos y estadísticas detalladas para visualizar el progreso de los hábitos. | Módulo de Seguimiento y Visualización. |
| Registro de Actividades Diarias | Permite registrar y ver las actividades diarias relacionadas con los hábitos del usuario. | Módulo de Seguimiento y Visualización. |
| Configuración de Recordatorios | Permite a los usuarios configurar recordatorios automáticos para sus hábitos y tareas. | Módulo de Recordatorios y Notificaciones. |
| Envío de Notificaciones Push | Envía notificaciones push a los dispositivos de los usuarios para recordarles sus hábitos y tareas. | Módulo de Recordatorios y Notificaciones. |

### DESCRIPCIÓN DE CONECTORES

| **Nombre del Conector** | **Tipo** | **Propósito** |
| --- | --- | --- |
| HTTP/HTTPS Connector | Comunicación | Permitir la comunicación segura entre el frontend y el backend mediante solicitudes y respuestas HTTP/HTTPS. |
| API Gateway | Gestión de Solicitudes | Administrar todas las solicitudes entrantes a los servicios del backend, asegurando la seguridad y el balanceo de carga. |
| Machine Learning Connector | Inteligencia Artificial | Permitir la comunicación entre el backend y los servicios de IA/ML para generar sugerencias y recomendaciones personalizadas. |

### Arquitectura lógica

### Performances

El sistema está diseñado para manejar una alta concurrencia, soportando más de 1000 usuarios simultáneamente sin comprometer el rendimiento. Para lograr tiempos de respuesta inferiores a 2 segundos por transacción, se implementarán las siguientes estrategias:

* **Escalabilidad Horizontal**: Se utilizarán contenedores Docker para desplegar microservicios en múltiples instancias en la infraestructura de AWS, permitiendo la adición de recursos según la demanda.
* **Balanceo de Carga**: Se implementará un balanceador de carga que distribuye las solicitudes de los usuarios entre las diferentes instancias de los microservicios, asegurando una distribución equitativa del tráfico y evitando sobrecargas en un único servicio.
* **Optimización de Consultas**: Se aplicarán técnicas de optimización en las consultas a la base de datos PostgreSQL, incluyendo índices y consultas asíncronas para reducir los tiempos de respuesta.

### Criterios de Calidad

El sistema debe cumplir con los siguientes criterios de calidad:

* **Usabilidad:**
  + **Intuitividad**: La interfaz debe ser fácil de entender y usar sin necesidad de formación extensa.
  + **Feedback del Usuario**: Incluir mecanismos para recopilar y actuar sobre la retroalimentación de los usuarios.
* **Mantenibilidad:**
  + **Documentación:** Proveer documentación completa y clara para desarrolladores y usuarios.
  + **Pruebas**: Implementar pruebas automatizadas y unitarias para asegurar la calidad continua del código.
* **Escalabilidad:** El sistema debe ser capaz de escalar horizontal o verticalmente según la demanda.
* **Alta Seguridad:** Protección de datos sensibles y robustez en autenticación y autorización.

### Detalles de la implementación

La implementación del sistema se basará en una arquitectura de microservicios, utilizando las siguientes tecnologías:

* **Backend**: Node.js con Express para la creación de APIs RESTful.
* **Frontend**: React junto con Material-UI para la creación de una interfaz de usuario moderna y responsiva.
* **Base de Datos**: PostgreSQL para la gestión eficiente de datos estructurados.  
  Firebase para la gestión de datos en línea.
* **Infraestructura**: AWS utilizando contenedores Docker para una gestión flexible y escalable de los microservicios.

### 

### Lenguajes y plataformas

Las tecnologías y versiones específicas que se utilizarán en el desarrollo del sistema incluyen:

* **Node.js**: v10.x
* **SQL Lite**: v9.6
* **Python**: v3.7
* **Visual Studio Code**: v1.50
* **Ionic:** 7.2.0
* **Firebase:** 10.14.1

## Acceso a fuentes de información del proyecto

Para acceder a la carpeta de evidencias acumuladas y otros documentos relevantes del proyecto, puede utilizar el siguiente enlace: [Evidencias de Proyecto](https://drive.google.com/drive/folders/116h_eutbAfKjIG7j3JIAHDMYtZVBqAbn?usp=drive_link)  
Esta carpeta incluye documentación técnica, diagramas, pruebas de rendimiento y cualquier otra evidencia relacionada con el desarrollo y la implementación del sistema.