

Arquitectura del Sistema

Kairos

NexTech

Centurión Valeria, Escalante Guillermo, Maldonado Agustina, Mendez Florencia, Ulloa Gonzalo.

**

**



**Tabla de contenido**

[**Introducción 4**](#_98j684wo4p0i)

[*Propósito 4*](#_y0bdl4xzem3i)

[*Alcance 4*](#_sz7hkbgsr6td)

[*Definiciones, Acrónimos, y Abreviaturas 4*](#_iycv5nwwkzi0)

[*Referencias 4*](#_jaho4q7vsjgo)

[*Panorama General 5*](#_xc2uyjmcyi5s)

[**Representación Arquitectónica 5**](#_6ovj5jqkk3x2)

[**Objetivos Arquitectónicos y Restricciones 5**](#_fjxnlodzilpx)

[*Objetivos Generales 5*](#_gngzfyfh49s1)

[*Objetivos Específicos 5*](#_o6esepygsee2)

[*Descripción de Procesos 6*](#_1h4tooopgwa0)

[**Vista de Caso de Uso 6**](#_cs6nyzfxve2e)

[*Descripción de los actores 6*](#_6gu7hp3byvuh)

[*Casos de uso significativos 6*](#_6g8hoqtlbyxc)

[*Contexto del sistema 7*](#_w6rkxf4xu8p3)

[**Vista Lógica 7**](#_1j05jkby0ste)

[*Perspectiva General 7*](#_sf59cjkiid7j)

[*Paquetes de Diseño importantes arquitectónicamente 8*](#_blhkfp3s41qc)

[**Vista de Procesos 9**](#_lyay6f36587s)

[**Vista de Liberación 11**](#_ym0a0cc9jm0)

[**Vista de Implementación 11**](#_cvb8cydg74m9)

[*Capas 11*](#_n57yxtebwpyn)

[**Diagramas 12**](#_7805pnasfes)

[*Diagrama de Objetos 12*](#_q5y08597mezs)

[*Diagramas de Paquetes 12*](#_g8lkzod0eszb)

Arquitectura del Sistema

# Introducción

## Propósito

Este documento tiene como propósito brindar una visión comprensible de la arquitectura general, utilizando diferentes vistas para ilustrar los distintos aspectos del sistema Kairos. Captura las decisiones más relevantes relacionadas con la arquitectura del sistema que fueron tomadas durante el desarrollo.

Desde el punto de vista de un desarrollador, este documento le permitirá comprender la estructura y fundamentos de la arquitectura, garantizando la correcta implementación, mantenimiento y escalabilidad del sistema.

Desde el punto de vista del cliente, este documento les permitirá entender cómo la arquitectura del sistema responde a sus necesidades y requisitos, brindando confianza en las decisiones técnicas adoptadas para cumplir con los objetivos del proyecto.

## Alcance

El Documento de Arquitectura de Software (DAS) abarca la descripción estructural y funcional del sistema Kairos, una plataforma de gestión de tareas y tiempos para equipos de desarrollo. Define cómo los componentes de software interactúan entre sí y con otros sistemas externos, detallando las vistas lógicas, físicas y de datos. Este documento influye directamente en el desarrollo, mantenimiento, integración y despliegue del sistema.

## Definiciones, Acrónimos, y Abreviaturas

* DAS: Documento de Arquitectura de Software
* API: Interfaz de Programación de Aplicaciones
* UI: Interfaz de Usuario
* REST: Representational State Transfer
* MVC: Modelo-Vista-Controlador
* JSON: JavaScript Object Notation
* DTO: Data Transfer Object.

## Referencias

* Modelos de Casos de Uso - Kairos - NexTech
* Modelo de Datos - Kairos - NexTech
* Prototipo Funcional - Kairos
* Implementación UARGFlow - Kairos
* Sommerville, Ian. (2011). Ingeniería del Software. 9ª Edición.
* Documentación oficial de Spring Boot, Angular y MySQL.
* Guía IEEE Std 1471-2000 - Recommended Practice for Architectural Description of Software-Intensive Systems.

## Panorama General

El presente documento se organiza en secciones que describen las vistas principales del sistema: lógica, de procesos, de implementación y de datos. Cada una refleja un enfoque distinto sobre el sistema Kairos y contribuye a una comprensión integral de su arquitectura.

# Representación Arquitectónica

La arquitectura del sistema Kairos se representa mediante un conjunto de vistas complementarias que reflejan distintos niveles de abstracción. Estas incluyen la vista lógica, de procesos, de implementación y de datos. En conjunto, muestran cómo los componentes interactúan para cumplir con los objetivos del sistema.

# Objetivos Arquitectónicos y Restricciones

## Objetivos Generales

El objetivo general del sistema Kairos es proporcionar una solución eficiente y colaborativa para la gestión de tareas con sus tiempos, permitiendo a los equipos de desarrollo planificar, asignar y controlar actividades dentro de los proyectos de software.

## Objetivos Específicos

* Facilitar la asignación y seguimiento de tareas entre los miembros del equipo.
* Permitir el seguimiento de los tiempos aplicados en las tareas en tiempo real.
* Permitir la autenticación de usuarios mediante cuentas de Google (OAuth 2.0).
* Proveer una interfaz moderna e intuitiva desarrollada en Angular.
* Garantizar la integridad y disponibilidad de los datos mediante una base de datos MySQL.
* Asegurar la escalabilidad y mantenimiento del sistema mediante una arquitectura en capas implementada en Spring Boot.

## Descripción de Procesos

Los procesos principales del sistema Kairos incluyen la autenticación de usuarios, la gestión de proyectos, tareas y tiempos, así cómo también el control de estado de las mismas. Cada proceso se compone de interacciones entre el frontend (Angular), el backend (Spring Boot) y la base de datos (MySQL). El flujo de información se mantiene a través de servicios REST y JSON cómo formato de intercambio de datos.

# Vista de Caso de Uso

La vista de casos de uso describe la percepción que tienen los actores de las funcionalidades del sistema. A continuación se listan los actores y los casos de uso significativos.

## Descripción de los actores

| **Actor** | **Descripción** |
| --- | --- |
| **Administrador** | Usuario del sistema encargado de la gestión inicial de proyectos y usuarios. Tiene permisos de gestión global, pero no interviene en las tareas internas de los equipos. |
| **Líder de proyecto** | Encargado de la gestión de etapas, usuarios y roles en cada proyecto. Tiene permisos de gestión en proyectos, etapas, tareas, reportes y miembros. |
| **Miembro** | Usuario encargado del registro de tiempos efectivos aplicados en cada tarea. Puede proponer tareas pendientes de aprobación y comentar tareas. |
| **Toggl Track** | Servicio externo de seguimiento automático de tiempos integrado con el sistema. |

### Casos de uso significativos

A continuación se listan los casos de uso agrupados por área funcional. Cada grupo contiene varios casos de uso de agregación, modificación y consulta; solo se describen los más relevantes para la arquitectura.

* Gestión de usuarios y autenticación
* CU01: Iniciar sesión con Google. Permite a cualquier actor autenticarse mediante OAuth 2.0.
* Gestión de proyectos e iteraciones
* CU05: Crear proyecto. El líder crea un nuevo proyecto indicando nombre, equipo, fechas y descripción.
* CU06: Modificar proyecto. Permite editar un proyecto, asignar usuarios o eliminarlo.
* Planificación
* CU07: Crear etapa y CU08: Crear iteración. Permiten dividir un proyecto en etapas e iteraciones.
* Gestión de tareas
* CU09: Crear tarea. El líder crea tareas con estimaciones y categorías.
* CU13: Asignar dependencia. Define dependencias entre tareas, evitando que una comience sin que otra finalice.
* Registro y seguimiento de tiempo
* CU17: Categorizar tiempo, CU18: Añadir categoría y CU19: Modificar categoría. Permiten definir y clasificar categorías de trabajo.
* CU20: Registrar tiempo por cronómetro. Integra Toggl Track para registrar tiempos en tiempo real.
* CU21: Registrar tiempo manual. Los miembros ingresan el tiempo trabajado.

## Contexto del sistema

Kairos se despliega como un sistema cliente‑servidor donde los usuarios interactúan a través de una aplicación web desarrollada en Angular. El servidor se implementa en Spring Boot y expone servicios RESTful. La base de datos MySQL almacena la información de usuarios, proyectos, tareas, tiempos y configuraciones. La base de datos MySQL almacena la información de usuarios, proyectos, tareas, tiempos y configuraciones. El sistema se integra con:

* Google OAuth 2.0 para la autenticación.
* Toggl Track para la medición de tiempos.

Los actores utilizan un navegador para acceder a la UI. El servidor gestiona la lógica de negocio, controla la persistencia de datos y se comunica con servicios externos (Google y Toggl).

# Vista Lógica

La vista lógica presenta la organización del sistema en términos de paquetes y clases principales. Kairos utiliza una arquitectura en capas basada en el patrón arquitectónico MVC.

## Perspectiva General

El patrón MVC permite una clara separación de responsabilidades y una estructura jerárquica organizada.

Cada capa cumple una función específica dentro del modelo de diseño, garantizando la escalabilidad, mantenibilidad y facilidad de integración con otros sistemas.

* **Capa de Presentación:** En esta capa se presentan las vistas del usuario (frontend) la cual se encontrará desarrollada en Angular, siendo la responsable de la interacción con el usuario, la visualización de la información y la comunicación con el backend a través de servicios.
* **Capa de Servicio:** En esta capa se encuentra el Backend, implementado en Spring Boot, el cual contendrá la lógica del dominio, los servicios y los controladores que procesan lassolicitudes del cliente y gestionan las operaciones principales del sistema.
* **Capa de Persistencia (Base de Datos):** implementada con MySQL, gestiona el almacenamiento de los datos y garantiza la integridad de la información mediante entidades y repositorios JPA.

## Paquetes de Diseño importantes arquitectónicamente

* Paquete de controladores: Contiene las clases que gestionan las peticiones HTTP y permiten la interacción con el Front-End.
  + UsuarioController: Gestión de operaciones relacionadas con los usuarios (altas, bajas, modificaciones).
  + RolController: Gestión de asignación de roles y permisos.
  + WebController: Gestión de las peticiones para la interfaz de usuario basada en vistas.
  + AuthController: Gestión de la autenticación y la información del usuario en el contexto de una API REST (manejo de sesiones/tokens).
* Paquete de servicios (service): implementa la lógica de negocio y se comunica con los repositorios para la gestión de los datos.
  + UsuarioService: Gestión de creación, modificación y eliminación de usuarios.
  + AuthService: Autenticación de usuarios y lógica relacionada con la sesión.
  + RolService: Administración de roles y permisos.
  + PermisoService: Gestiona los permisos que tiene un usuario dentro del sistema.
* Paquete de seguridad (config): Contiene las clases que gestionan la seguridad del sistema.
  + SecurityConfig: Contiene las reglas de autorización y autorización mediante Spring Security.
  + JwtUtil: Utilidad para la generación y validación de JSON Web Tokens (JWT).
  + SessionHelper: Clase de ayuda para la gestión de la sesión del usuario en el contexto de la web tradicional (basada en sesiones HTTP).
* Paquete de transferencia de datos (dto): Contiene las clases utilizadas exclusivamente para la transferencia de datos entre las distintas capas del sistema, estas clases no contienen lógica de negocio.
  + UsuarioRequest: Estructura los datos de entrada para las operaciones de creación o modificación de un usuario (ej. nombre, email, lista de IDs de roles).
  + UsuarioResponse: Estructura los datos de salida con la información de un usuario que se envía al Front-End.
  + AuthResponse: Contiene la respuesta después de una operación de autenticación (ej. un mensaje de éxito y el JWT/Token de acceso).
  + UserInfoResponse: Estructura la información detallada del usuario autenticado para la sesión (ej. ID, nombre, email, lista de permisos y si es administrador).
  + RolRequest/RolResponse: Objetos para transferir datos de creación/modificación de roles y para mostrar la información de un rol.
  + PermisoRequest/PermisoResponse: Objetos para transferir datos de creación/modificación de permisos y para mostrar la información de un permiso.

# Vista de Procesos

La vista de procesos del sistema Kairos describe cómo se organiza la concurrencia y la comunicación entre los distintos componentes que conforman la arquitectura cliente-servidor. Kairos se compone de procesos ligeros (hilos de control) y procesos pesados (grupos de procesos concurrentes) que operan en conjunto para permitir la interacción fluida entre los usuarios, el servidor y los servicios externos integrados. Este modelo permite que múltiples usuarios interactúen con el sistema de manera concurrente sin afectar la integridad ni el rendimiento general, garantizando la sincronización de datos y una experiencia fluida.

Grupo de Procesos de Interfaz de Usuario (Cliente Web): Corresponde a la aplicación web desarrollada en Angular, a la cual los usuarios acceden a través del navegador. Este grupo se encarga de gestionar la interacción con el usuario, capturar sus acciones y mostrar los resultados obtenidos desde el servidor.

* Procesos ligeros:
  + Hilo de eventos de usuario: escucha las acciones del usuario (inicio de sesión, registro de tiempo, gestión de tareas, etc.).
  + Hilo de renderizado y actualización: actualiza dinámicamente las vistas en función de los datos recibidos del backend.
* Comunicación:  
  Se comunica con el servidor Spring Boot a través de solicitudes HTTP RESTful, intercambiando información en formato JSON. La comunicación es asíncrona (mediante llamadas HTTP), lo que permite mantener la interfaz reactiva y sin bloqueos durante las operaciones.

Grupo de Procesos de Lógica de Negocio (Servidor Spring Boot): Representa los procesos ejecutados en el servidor, encargados de gestionar las reglas del negocio, validar los datos, controlar la persistencia y coordinar la comunicación con los servicios externos.

* Procesos ligeros:
  + Controladores REST: reciben y procesan las solicitudes provenientes del cliente web.
  + Servicios de negocio: implementan la lógica del dominio, como la gestión de proyectos, tareas y tiempos.
  + Manejador de autenticación: valida usuarios y tokens a través de la integración con Google OAuth 2.0.
* Comunicación:Se comunica con la base de datos MySQL mediante el uso de repositorios JPA y con los servicios externos (Toggl y Google) mediante solicitudes REST. Internamente, los distintos componentes del servidor se comunican por medio de invocación de métodos directos (síncronos) y el mecanismo de Inyección de Dependencias de Spring.

Grupo de Procesos de Persistencia y Datos (Base de Datos MySQL): Este grupo gestiona la persistencia de la información relacionada con usuarios, proyectos, tareas, tiempos y configuraciones del sistema.

* Procesos ligeros:
  + Hilo de conexión persistente: mantiene la sesión entre el servidor y la base de datos.
  + Hilo de transacción: ejecuta las operaciones CRUD requeridas por la capa de negocio.
* Comunicación:  
  Se comunica únicamente con el servidor Spring Boot mediante consultas SQL generadas por el framework JPA. Las respuestas son devueltas como entidades JPA. La conversión a Objetos de Transferencia (DTO) es responsabilidad de la Capa de Servicio.

Grupo de Procesos de Integración Externa: Este grupo agrupa los procesos que gestionan la comunicación con servicios externos utilizados por Kairos; Google OAuth 2.0, para la autenticación de usuarios, y Toggl Track, para la sincronización y registro de tiempos de trabajo.

* Procesos ligeros:
  + Hilo de autenticación externa: envía y recibe tokens de Google para validar las credenciales del usuario.
  + Hilo de sincronización de tiempos: se encarga de enviar y recibir datos desde la API de Toggl mediante solicitudes REST.
* Comunicación:  
  Se realiza mediante peticiones HTTP/HTTPS con intercambio de información en formato JSON, utilizando control de errores y validación de respuestas. La comunicación es asincrónica para no afectar el rendimiento del servidor principal.

Modo Principal de Comunicación entre Procesos

El sistema Kairos utiliza un modelo cliente-servidor con comunicación basada en servicios RESTful.

* Entre cliente y servidor, la interacción se realiza mediante mensajes HTTP y respuestas JSON, de forma asincrónica.
* Dentro del servidor, la coordinación entre procesos se efectúa mediante invocaciones de métodos y dependencias gestionadas por Spring.
* La comunicación con la base de datos y los servicios externos se maneja a través de transacciones controladas y peticiones REST, respectivamente.

# Vista de Liberación

El sistema Kairos se libera en un entorno de red local (LAN) perteneciente a la UNPA UARG, garantizando conectividad interna, seguridad y un acceso controlado para alumnos y docentes. En esta vista se contempla diferentes componentes físicos y configuraciones sobre los mismos:

Red local (LAN): en esta configuración, los componentes serán nodos conectados al sistema por medio de la red LAN de la UNPA UARG, lo qué asegura buena comunicación estable y segura.

Los nodos conectados serán los equipos pertenecientes a los laboratorios de informática de las aulas del área de informática de la UNPA UARG, donde los estudiantes podrán hacer uso para acceder al sistema y gestionar sus proyectos, tiempos y tareas.

Por fuera de los equipos pertenecientes a los laboratorios de informática, se encuentran también los equipos personales de los estudiantes, estos nodos podrán acceder al sistema cuando se encuentren dentro del área de cobertura de la red local de la UNPA UARG.

# Vista de Implementación

El modelo de implementación sigue una estructura basada en capas.

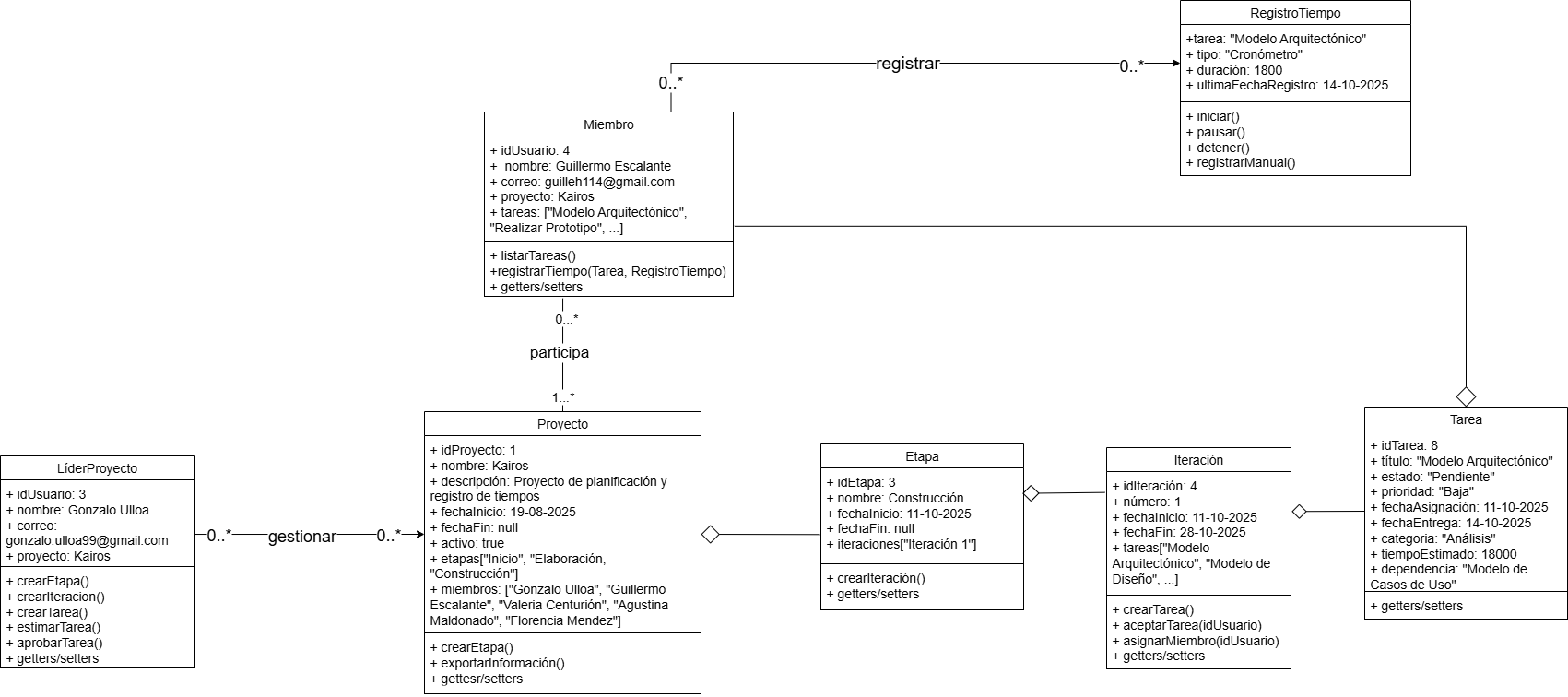
* Capa de Persistencia: MySQL.
* En esta capa se maneja la persistencia de los datos de los usuarios, roles, permisos, proyectos, etc.
* Capa de servicio: Spring Boot, Java.
* Se utilizan servicios REST para comunicar el Front-End con el Back-End.
* Capa de Presentación: Angular, HTML, CSS, TypeScript.
* Visualización en servidor web.

## Capas

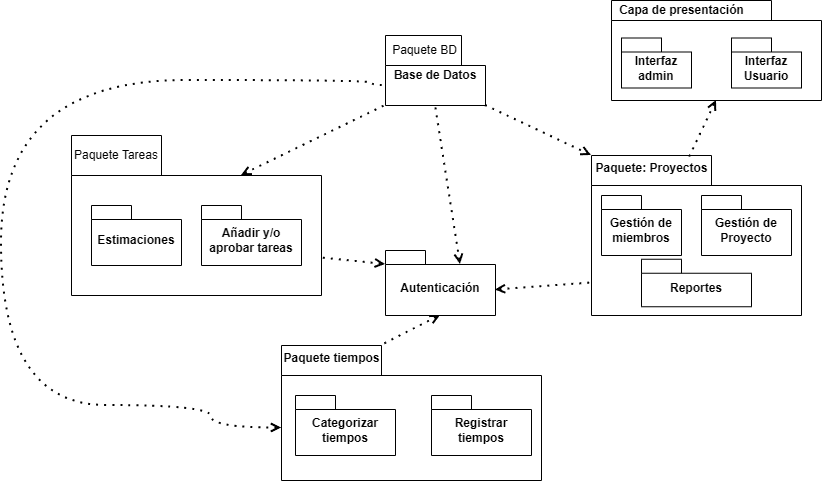
* **Presentación:** HTML, CSS y TypeScript.
* **Servicio:** clases @Service en Spring Boot.
  + **Controladores:** clases @RestController que exponen endpoints.
  + **Modelos y repositorios:** clases @Entity y @Repository (Repositorios JPA).
* **Clientes externos:** componentes para consumir APIs de Google y Toggl.

# Diagramas

## Diagrama de Objetos



## Diagramas de Paquetes

**

#### 