# **Formulario: Vista de Desarrollo del Proyecto de Software**

**Proyecto:** Sistema de Gestión de Personas para Pymes - Madu  
**Grupo:** Fabián Carrimán – Cristian Jara   
**Fecha:** 4/10/2024

**1. Estructura del sistema**

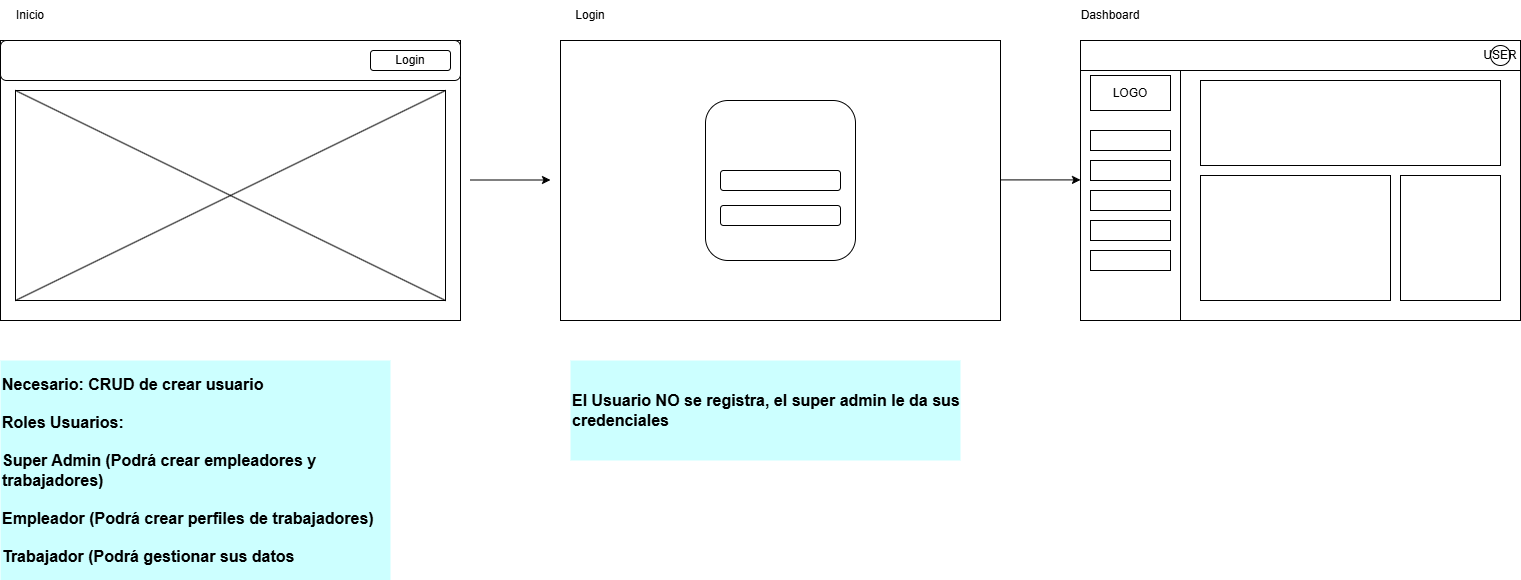
Describe la estructura del sistema de tu proyecto, mencionando los módulos principales y cómo interactúan entre sí. Puede usar diagramas de arquitectura para complementar tu descripción.

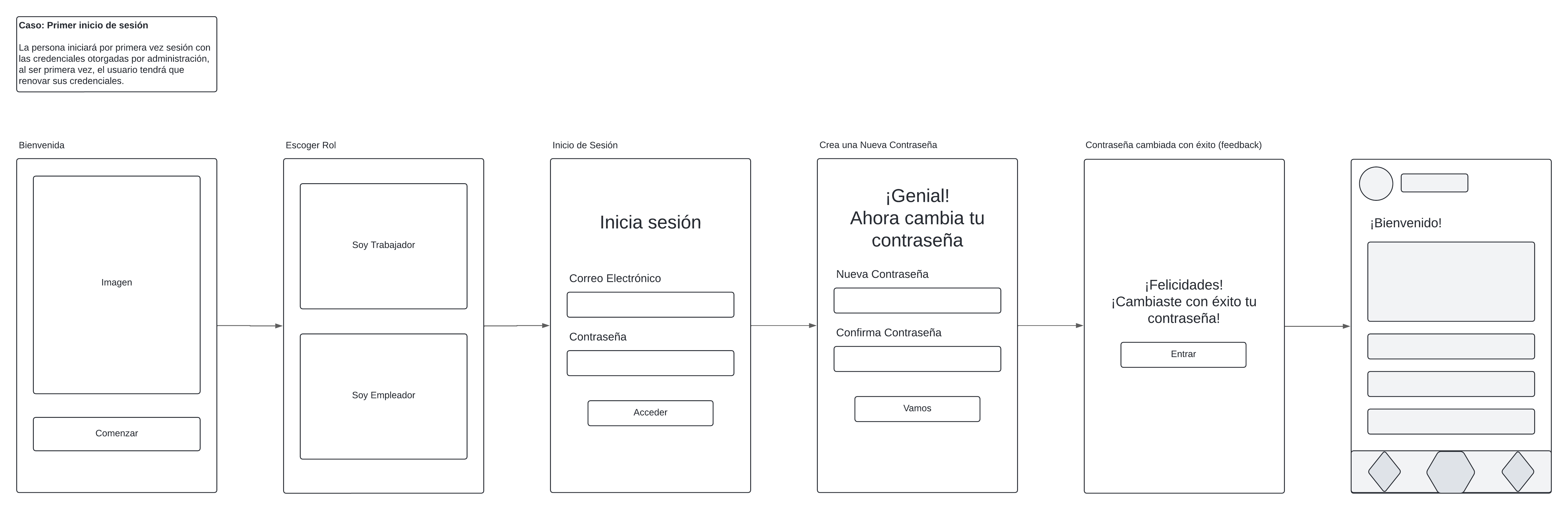
**Descripción de la estructura:**

Descripción de la estructura: El sistema Madu se compone de tres componentes principales interconectados:

1. Sitio Web Principal: Presenta información sobre el producto y permite el acceso a la WebApp.
2. Dashboard de Gestión: Administración de Empleados, Cargos, etc...
3. Aplicación Móvil: Versión móvil para acceso rápido de empleados.

Estos componentes se comunican a través de una API RESTful centralizada, que gestiona la lógica de negocio y el acceso a la base de datos

**Diagrama de la estructura del sistema:**   
**

****

**2. Módulos principales**

Enumera y describe los módulos principales del sistema, explicando sus responsabilidades y cómo interactúan con otros módulos.

**Módulo 1 (Nombre):** Sitio Web Principal  
**Responsabilidad (Funcionabilidad):** El Sitio Web Principal es la página pública de Madu, donde los usuarios (empleados y empresas) pueden obtener información sobre la plataforma, conocer sus funcionalidades y contactar a la organización. Este módulo se encarga de gestionar el contenido informativo y promocional del sistema, permitiendo a los usuarios acceder al inicio de sesión.  
**Interacción con otros módulos:**

* Se comunica con el módulo de **Autenticación** para permitir a los usuarios iniciar sesión o registrarse.
* Envía y recibe datos del **Backend (Firebase)** para la validación de credenciales y para mostrar información dinámica en el sitio.
* Proporciona enlaces al módulo de **Dashboard de Gestión**, redirigiendo a los usuarios autenticados según sus roles (empleados, administradores, etc.).

**Módulo 2 (Nombre):** Dashboard de Gestión  
**Responsabilidad (Funcionabilidad):** El Dashboard de Gestión es el área privada para usuarios autenticados, donde pueden gestionar sus datos y acceder a diferentes funcionalidades según su perfil. Proporciona a los administradores herramientas para gestionar empleados, asignar tareas, ver reportes y controlar la información de la empresa. Los empleados pueden visualizar y actualizar sus datos, ver sus asignaciones y reportar avances.

**Interacción con otros módulos:**

* **Autenticación**: Verifica el rol y los permisos del usuario a través de los servicios de autenticación de Firebase.
* **Backend (Firebase)**: Realiza operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre la base de datos, gestionando información relacionada a empleados, roles, tareas, etc.
* **Aplicación Móvil**: Comparte datos de usuarios, tareas y notificaciones para sincronizar la información entre las plataformas web y móvil.

**Módulo 3 (Nombre):** Aplicación Móvil  
**Responsabilidad (Funcionabilidad):** La Aplicación Móvil permite a los usuarios acceder a la información de su cuenta desde sus dispositivos móviles. Los empleados pueden revisar su perfil, actualizar su información personal, recibir notificaciones y reportar avances o problemas en tiempo real. Facilita el acceso rápido a funcionalidades críticas del sistema desde cualquier lugar.

**Interacción con otros módulos:**

* **Backend (Firebase)**: Se sincroniza en tiempo real con la base de datos, manteniendo actualizada la información de empleados, tareas y notificaciones.
* **Dashboard de Gestión**: Los cambios realizados en el Dashboard de Gestión (como la asignación de tareas) se reflejan en la aplicación móvil, y viceversa. Esto garantiza una experiencia unificada para los usuarios.
* **Autenticación**: Utiliza el mismo módulo de autenticación que el sitio web para validar usuarios y gestionar sesiones.

**3. Dependencias y Librerías**

Especifica las dependencias y librerías/frameworks que estás utilizando en tu proyecto, indicando para qué funcionalidad o módulo se usan.

**Dependencias y Librerías:**

* Angular: Framework para el desarrollo del frontend (WebApp y Sitio Web)
* Ionic: Framework para el desarrollo de la aplicación móvil
* Node.js con Express: Para el backend y API RESTful
* JWT: Para autenticación y manejo de sesiones
* Angular Material, Tailwind con Flowbite: Para estilos y componentes de UI
* Sweetalert
* Firebase

**4. Organización del Código**

Explica cómo está organizado el código en términos de carpetas y paquetes dentro del proyecto. Incluye ejemplos o un esquema de la estructura de directorios.

**Organización del código:**

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Raíz del proyecto ("madu-app")**:

* Contiene carpetas y archivos de configuración generales, como .angular, .vscode, node\_modules, y public.

**Carpeta src/app**:

* Es la carpeta principal donde reside todo el código fuente de la aplicación.
* Dentro de app, la estructura está dividida en varias subcarpetas que separan las funcionalidades y componentes de la aplicación.

**Subcarpetas en src/app**

1. **auth**:
   * Contiene componentes relacionados con la autenticación de usuarios (login, registro, recuperación de contraseña, etc.).
2. **dashboard**:
   * Contiene componentes y vistas dedicadas a las funcionalidades principales de la aplicación, donde los usuarios autenticados pueden gestionar su información o acceder a distintas herramientas.
3. **layout**:
   * Dentro de layout, hay subcarpetas como footer, header, y main-layout.
   * Esta carpeta se encarga de los componentes que conforman la estructura visual general de la aplicación (cabecera, pie de página, y la disposición principal).
4. **pages**:
   * En pages, se encuentra otra subcarpeta home, que contiene los archivos del componente home.component.
   * Los archivos de home.component (html, scss, ts, spec.ts) indican que aquí se define la vista principal o de bienvenida de la aplicación.
   * Pages contiene diferentes vistas o componentes que forman las distintas páginas de la aplicación.

**5. Reglas de Integración**

Especifica las reglas que tu equipo sigue para la integración del código entre los módulos. Esto puede incluir restricciones sobre el acceso a datos, convenciones de desarrollo, etc.

**Reglas de integración:**

Las reglas de integración que nosotros seguimos para estandarizar el desarrollo y mantener un orden consistente entre los módulos del proyecto.  
Podemos decir que:  
1) Los componentes y servicios están nombrados con prefijos que reflejan su módulo respectivo (auth-, dashboard-, etc.) para facilitar su identificación.

2) Se utilizan interfaces y models definidos en un módulo común (shared) para garantizar que la estructura de los datos sea consistente entre todos los módulos.

3) Se implementan linters y prettiers para mantener un formato de código homogéneo en todo el proyecto

4) Trabajamos en ramas individuales (feature/, bugfix/, hotfix/) y las integramos a la rama principal (main) a través de Pull Requests.

**6. Prácticas de desarrollo**

Detalla las prácticas de desarrollo que estás utilizando en el proyecto, como las pruebas unitarias, integración continua, revisión de código, etc.

**Prácticas de desarrollo:**

* 1. **Pruebas Unitarias:**

Cada componente, servicio y módulo del proyecto cuenta con pruebas unitarias desarrolladas con Jasmine y Karma (Angular) para asegurar que la lógica de cada parte de la aplicación funcione correctamente de manera aislada.

* 1. **Pruebas End-to-End (E2E):**
* Se utilizan herramientas como **Cypress** para ejecutar pruebas E2E que simulan el comportamiento de un usuario en la aplicación, validando flujos completos desde el frontend hasta el backend.
  1. **Integración Continua (CI):**
* Se ha configurado un pipeline de Integración Continua (CI) con GitHub Actions que ejecuta automáticamente todas las pruebas unitarias, de integración y E2E en cada push o pull request.

**4) Revisión de Código (Code Review):**

* Antes de fusionar cualquier cambio a la rama principal (main), se realiza una revisión de código por parte de al menos otro miembro del equipo para asegurar la calidad del código.

**5) Control de Versiones:**

* Se utiliza Git para el control de versiones, siguiendo una convención de ramas basada en GitFlow (feature/, bugfix/, hotfix/, release/).
* Las ramas se organizan de manera que cada nueva funcionalidad, corrección de errores o mejoras se desarrolle en una rama independiente y luego se integre a main a través de Pull Requests.

**6) Metodología Ágil (Scrum):**

* El proyecto se gestiona bajo la metodología ágil Scrum, realizando entregas incrementales y revisiones periódicas para asegurar que el desarrollo se alinea con los objetivos del cliente.

**7. Herramientas de desarrollo**

Mencione las herramientas que está utilizando para el desarrollo del proyecto (por ejemplo, IDE, sistemas de control de versiones, etc.).

**Herramientas:**

IDE: Visual Studio Code.

Backend: Google Firebase

Control de versiones: Git/GitHub.

Sistemas de Diseño y Prototipado: Draw.io, Figma

Gestión del Proyecto: Trello

Documentación del proyecto: Google Drive

Formateador: Prettier, HTML Default Formatter (VS Code)

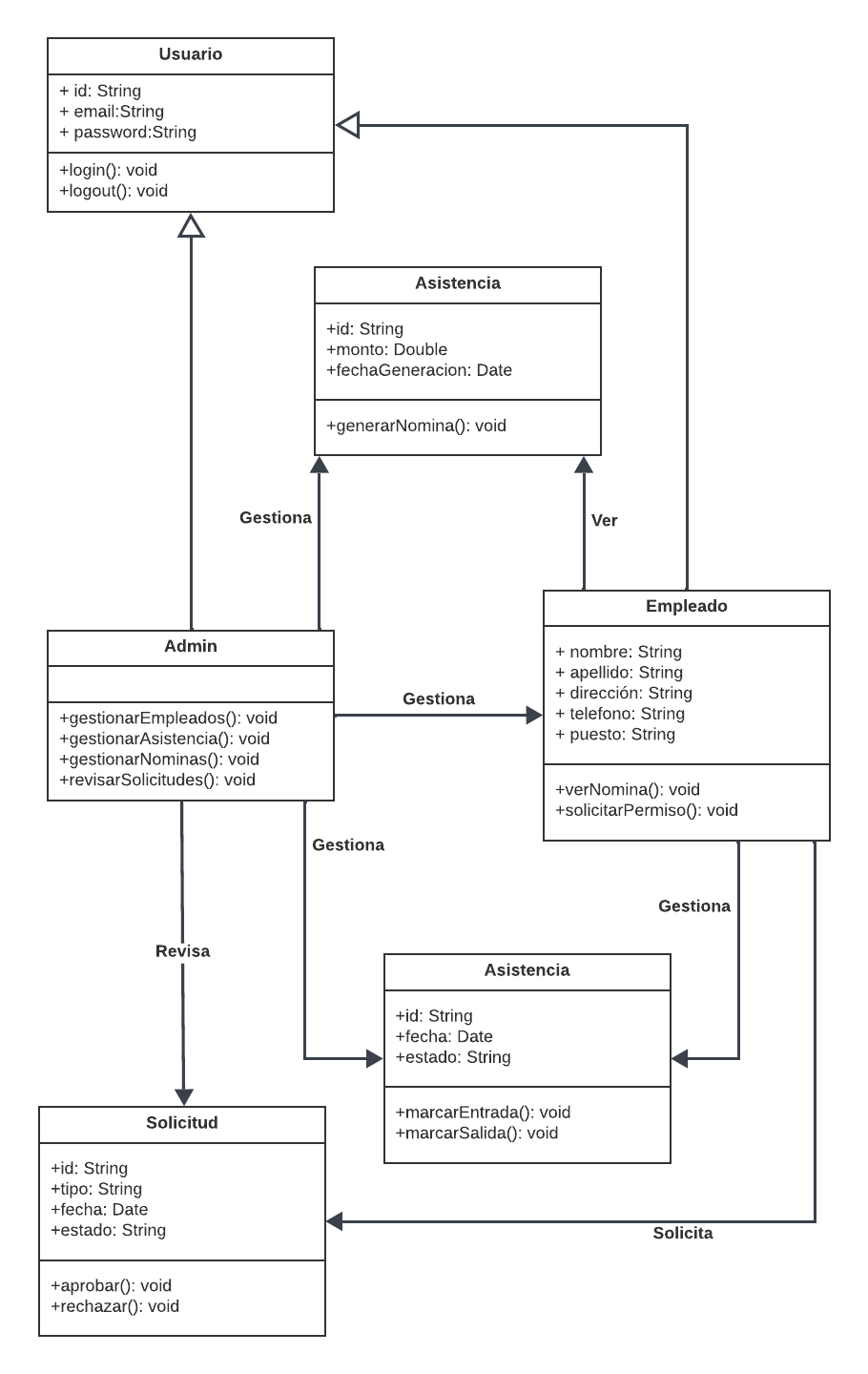
**8. Diagrama de Arquitectura del Sistema**

Adjunta un diagrama de arquitectura general del sistema que incluye los módulos, las interacciones entre ellos y las tecnologías involucradas.

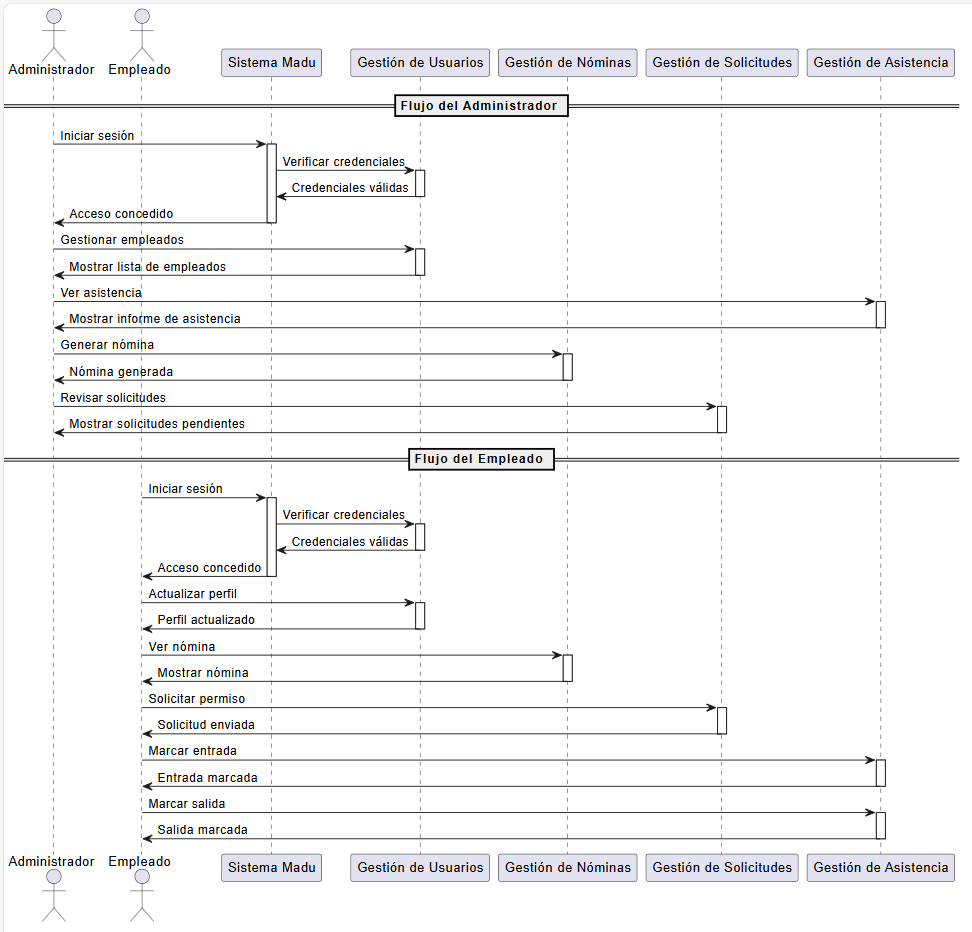
**Diagrama de arquitectura general:**

* 1. Vista Lógica

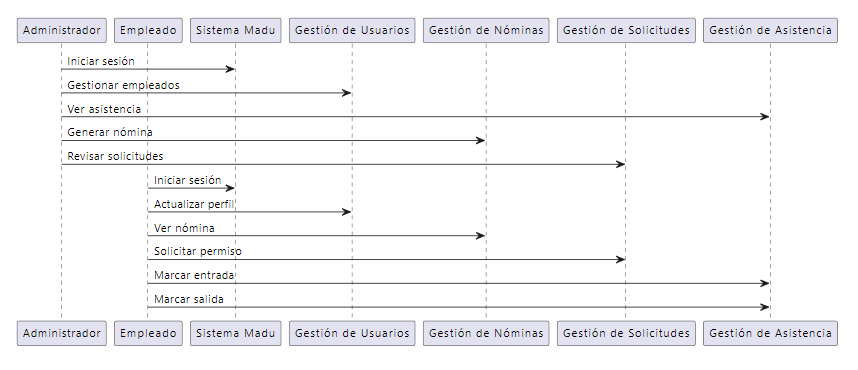
**Diagrama de Clases**



**Diagrama de Secuencia**

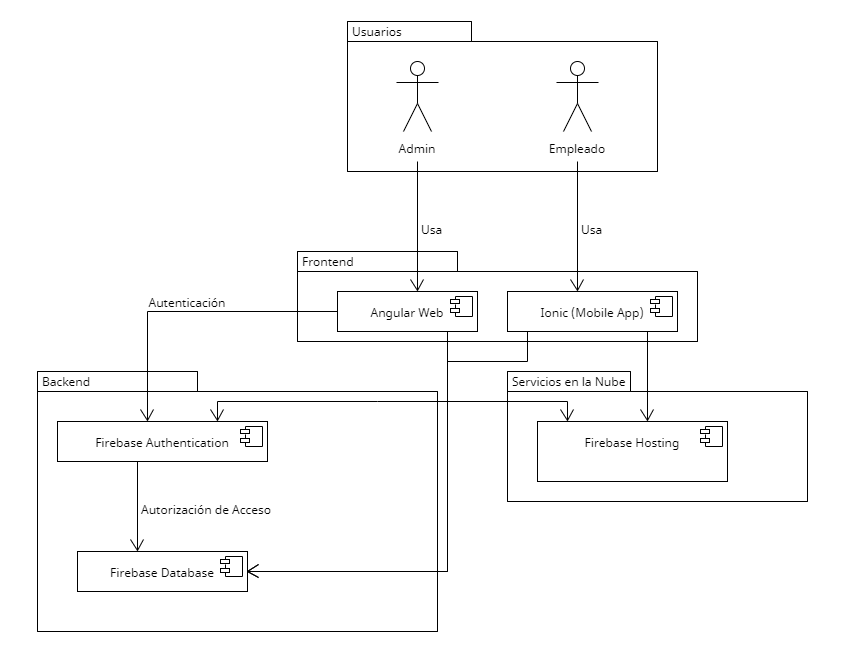
****

**Diagrama de Comunicación**

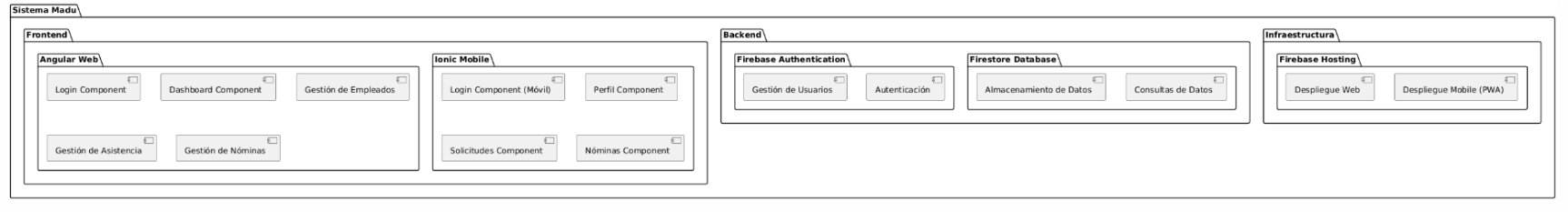
****

* 1. Vista de Desarrollo

**Diagrama de Componentes**

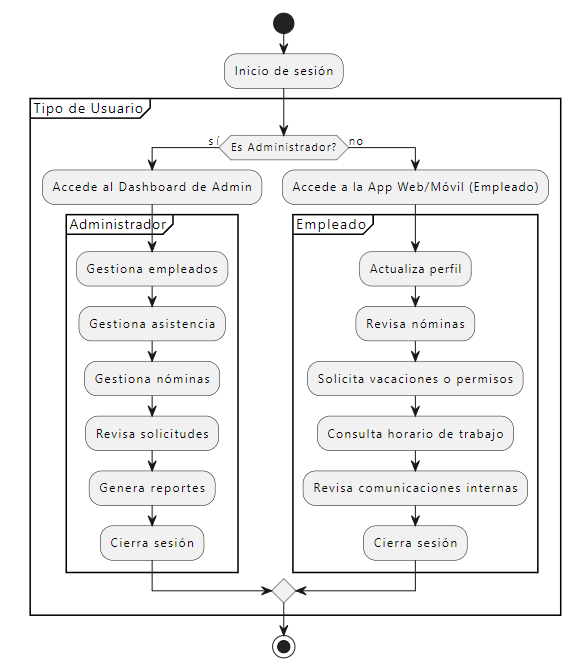


**Diagrama de Paquetes**

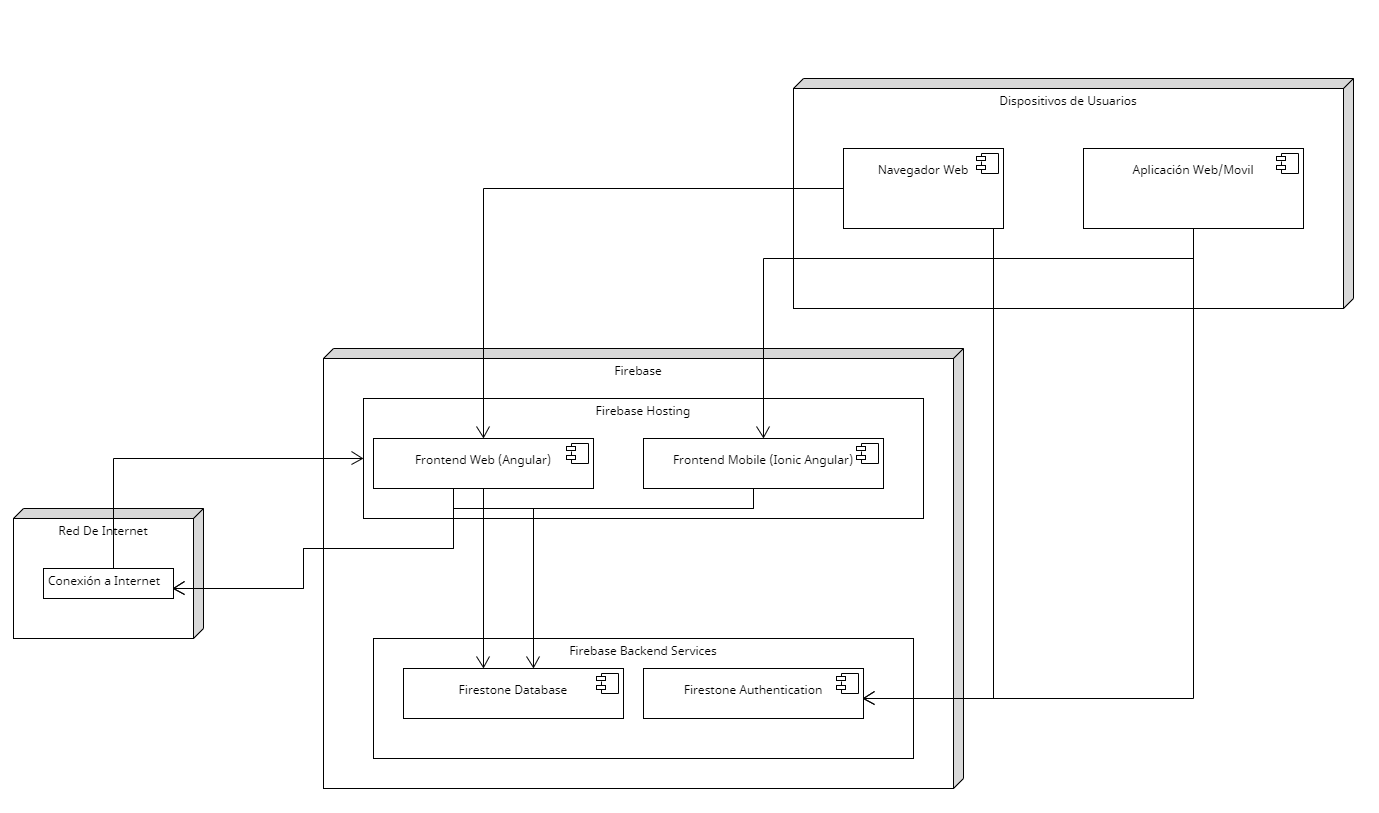
****

* 1. Vista Proceso

**Diagrama de Actividades**

****

* 1. Vista Física

Diagrama de Despliegue

* 1. Vista Escenarios
* **Diagrama de Caso de Uso de Empleado (Web)**

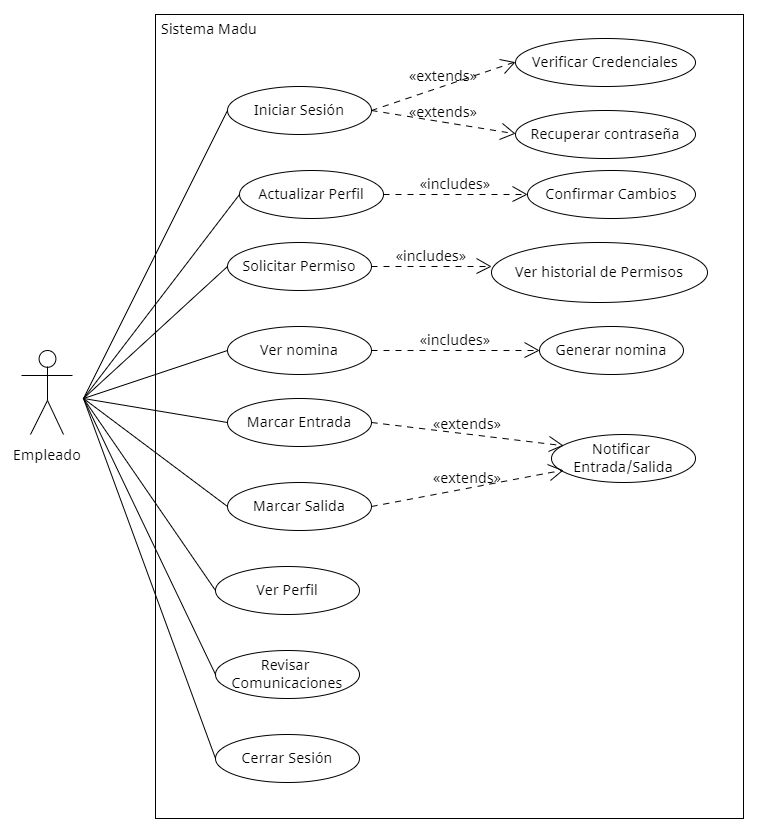


Diagrama de caso de uso (Móvil)