# Rayostrength
# Modelo 4+1 Arquitectónico

Integrantes: *Fabián Fernández Canelo, Benjamin Gonzalez Valencia, Emilio Rubina Salinas*

Asignatura: *Capstone*

Fecha: 04/10/2025

Docente: *Cindy Contador*

# Introducción

El presente documento describe la arquitectura del sistema Rayostrength utilizando el modelo 4+1, que permite representar el software desde las vistas lógica, de desarrollo, de  procesos, física y de escenarios. Este modelo se complementa con la metodología ágil  Scrum y el patrón arquitectónico MVC para garantizar una organización modular, escalable y mantenible.

# Patrón Arquitectónico: MVC

El sistema Rayostrength se basa en el patrón arquitectónico **MVC (Modelo–Vista–Controlador)**, el cual permite separar claramente las responsabilidades dentro de la aplicación, favoreciendo la mantenibilidad, escalabilidad y reutilización del código.

Este patrón se divide en tres capas principales:

## Modelo

El modelo representa la capa encargada de la lógica del negocio y la gestión de datos. Aquí se encuentran las clases que modelan las entidades principales del sistema, junto con sus atributos y relaciones.

El modelo está compuesto por las entidades:

- Usuario
- Coach
- Progreso
- Administrador

Estas clases son responsables de interactuar con la base de datos, almacenar la información relevante y mantener la consistencia de los datos.

## Controlador

La capa “controlador” actúa como intermediaria entre la vista y el modelo. Su función es recibir las acciones del usuario desde la interfaz (vista), procesar la lógica correspondiente y comunicarse con el modelo para consultar o actualizar datos.

Se incluyen controladores tales como:

- UsuarioController
- ProgresoController
- RutinaController

Cada controlador gestiona un flujo específico del sistema. Por ejemplo, el ProgresoController valida los datos ingresados por el usuario, crea un nuevo registro de progreso y lo envía al modelo para su almacenamiento.

## Vista

La vista corresponde a la interfaz con la que interactúa el usuario.
Esta capa está representada por la aplicación móvil desarrollada en React Native, donde se muestran las pantallas de inicio de sesión, visualización de rutinas y registro de progreso.

La vista no maneja lógica de negocio, sino que únicamente presenta la información y envía las acciones del usuario hacia los controladores.

## Interacción entre capas

El flujo de interacción dentro del patrón MVC se realiza de la siguiente manera: 1.

El usuario realiza una acción en la vista (por ejemplo, registrar su progreso).

2. La vista envía la acción al controlador correspondiente.

3. El controlador procesa la solicitud y comunica con el modelo.

4. El modelo ejecuta la operación (guardar, consultar o actualizar datos).

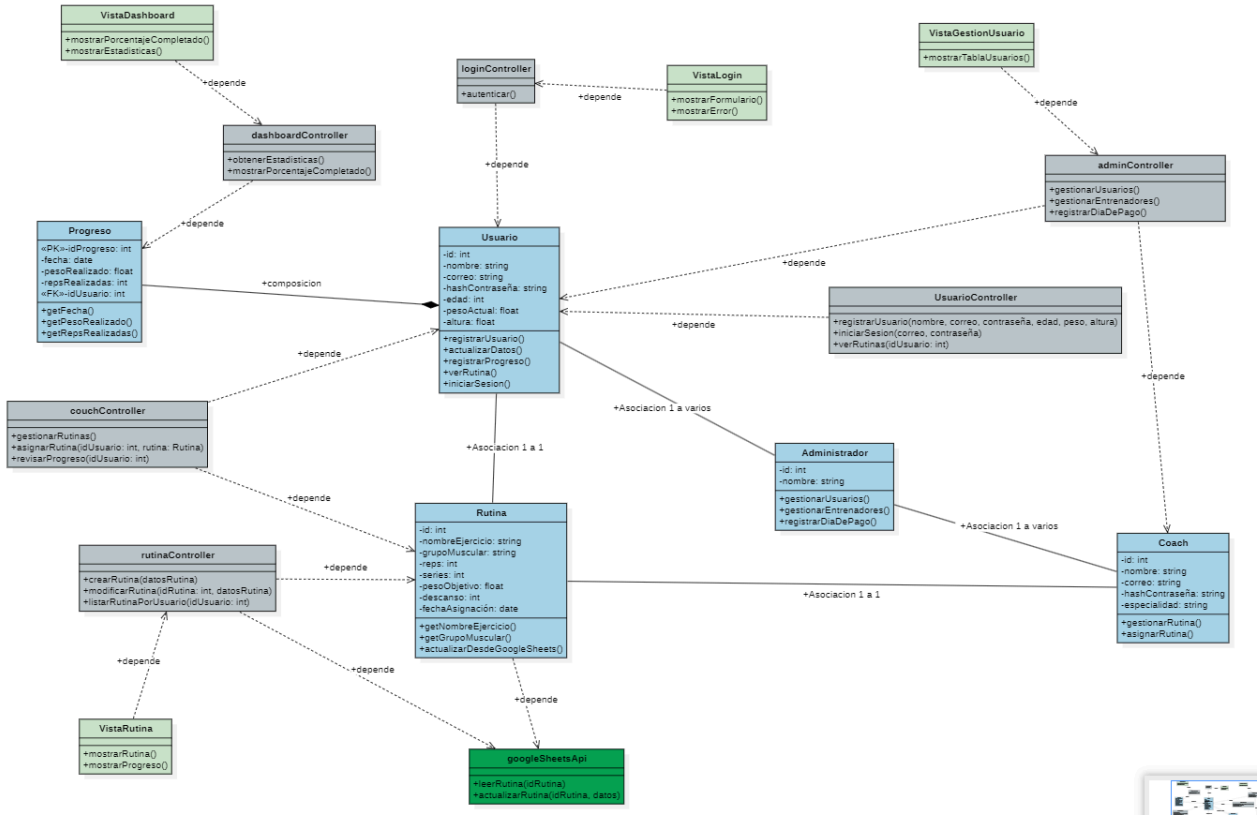
5. El controlador recibe la respuesta del modelo y actualiza la vista con los resultados.

Este flujo asegura una separación de responsabilidades clara, donde cada capa cumple una función específica, facilitando el mantenimiento del sistema y la incorporación de nuevas funcionalidades sin afectar las demás capas.

# Vista Lógica

El diseño lógico del sistema se realizó siguiendo el patrón arquitectónico MVC, donde las clases del modelo representan las entidades centrales que manejan los datos y la lógica de negocio.

## Diagrama de Clases




## Estructura general

### Modelo

Contiene las clases que representan las entidades principales del dominio:

- Usuario: Almacena los datos del usuario final (nombre, correo, peso, altura, etc.) y métodos relacionados con su interacción en la aplicación.
- Coach (Entrenador): Gestiona las rutinas personalizadas de sus usuarios asignados.
- Rutina: Representa los ejercicios asignados a un usuario, con información como nombre del ejercicio, grupo muscular, repeticiones, series, descanso y fecha de asignación.
- Progreso: Registra los avances del usuario, incluyendo peso levantado, repeticiones realizadas y fecha.
- Administrador: Gestiona usuarios y coaches, además de el panel de control de la aplicación.

Las relaciones entre estas clases permiten que:

- Un usuario tiene múltiples registros de progreso.
- Un entrenador (coach) asigna varias rutinas.
- Una rutina pertenezca a un usuario y a un entrenador.

Los atributos de estas clases son privados, para asegurar encapsulamiento, y los datos se acceden mediante métodos *getters* y *setters* públicos.

## Controladores

Contiene las clases encargadas de manejar la lógica de negocio y coordinar la comunicación entre la interfaz de usuario (vista) y el modelo.
Estas clases son:

- UsuarioController: Gestiona el registro, inicio de sesión y visualización de rutinas.
- CoachController: Maneja la gestión de usuarios y la asignación de rutinas.
- RutinaController: Controla la creación, modificación y listado de rutinas.
- ProgresoController: Gestiona el registro del progreso y la consulta del historial del usuario.
- DashboardController: Obtiene las estadísticas del usuario y muestra el porcentaje de rutinas completadas.
- LoginController: Controla la autenticación del usuario.
- AdminController: Gestiona usuarios y coaches.

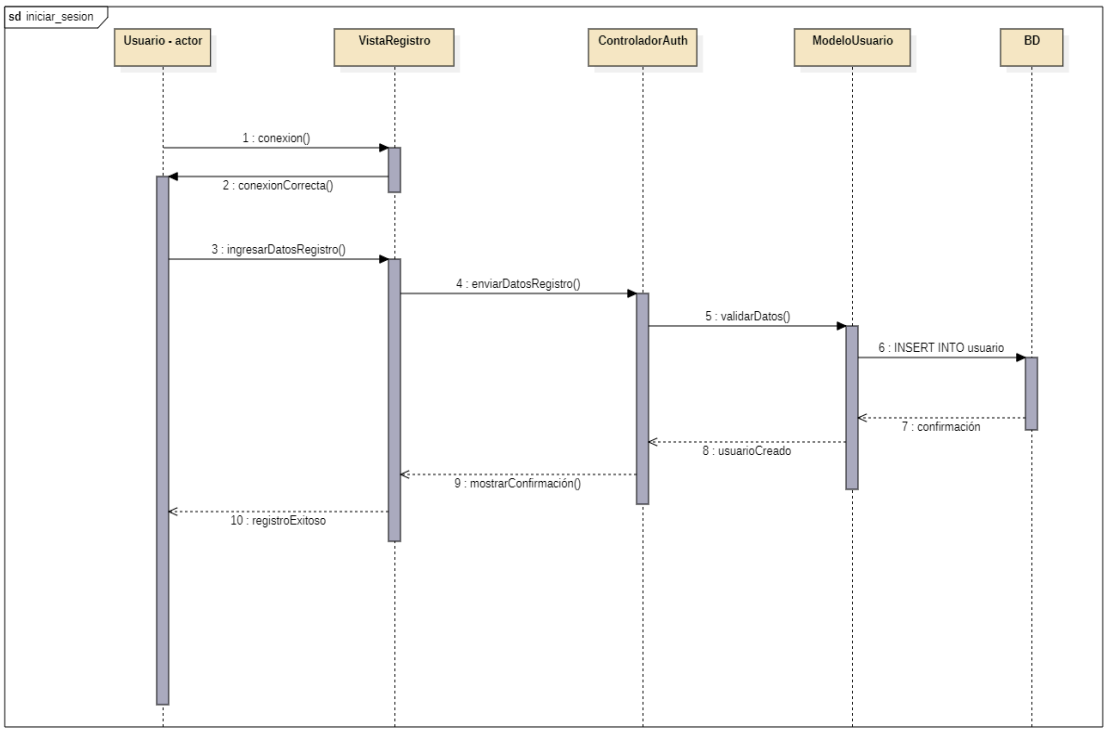
Cada controlador tiene relaciones de dependencia con las clases del modelo que manipula, siguiendo la estructura del patrón MVC.

# Diagramas de secuencia

REGISTRARSE

Este diagrama describe la secuencia de interacciones que ocurren cuando el usuario registra su progreso dentro de la aplicación.

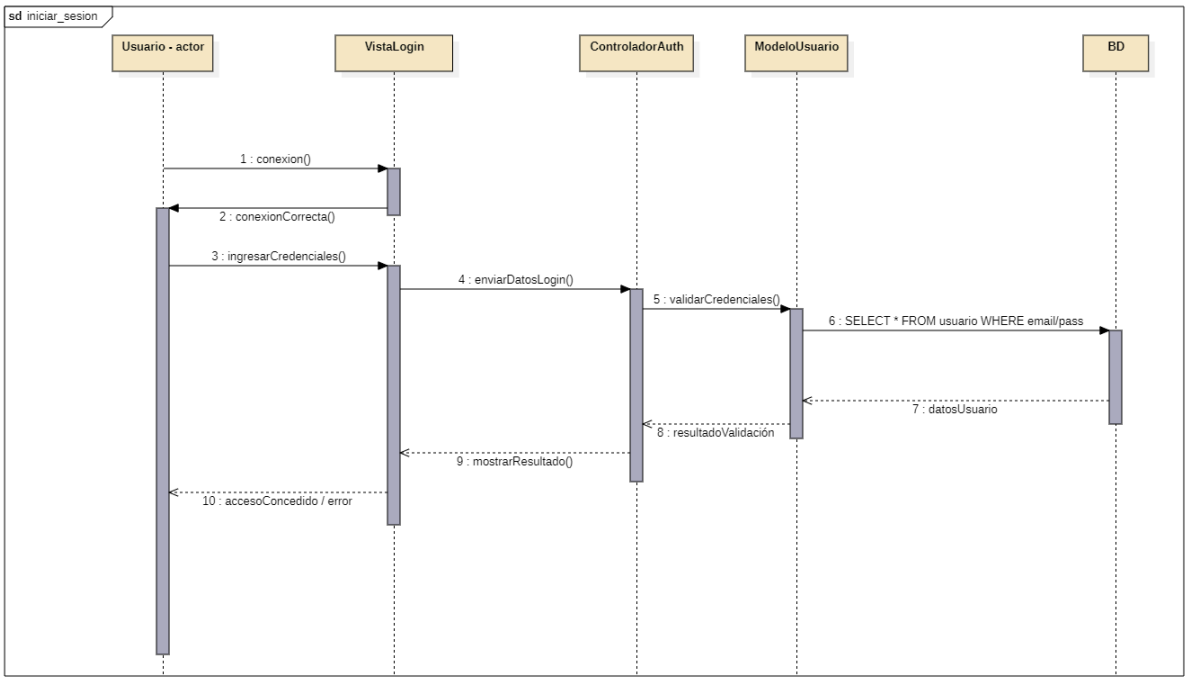
El flujo muestra cómo el controlador recibe los datos ingresados (peso, repeticiones, fecha), los valida y los almacena en el modelo de Progreso, permitiendo su posterior visualización o análisis por parte del entrenador.




INICIAR SESIÓN

Este diagrama representa la interacción entre el usuario, el controlador de usuario y el sistema de autenticación, mostrando el proceso que permite al usuario iniciar sesión en la aplicación.

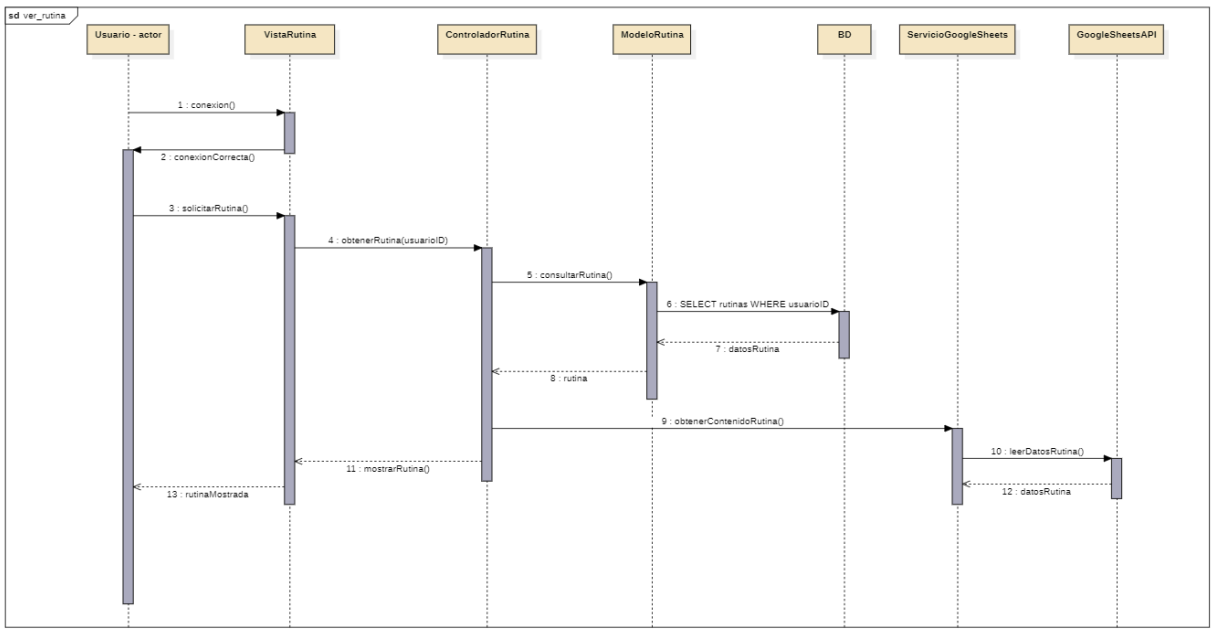
Se detalla cómo se envían las credenciales, cómo el sistema valida la información y cómo se genera la respuesta indicando si el acceso fue exitoso o no.




VER RUTINA

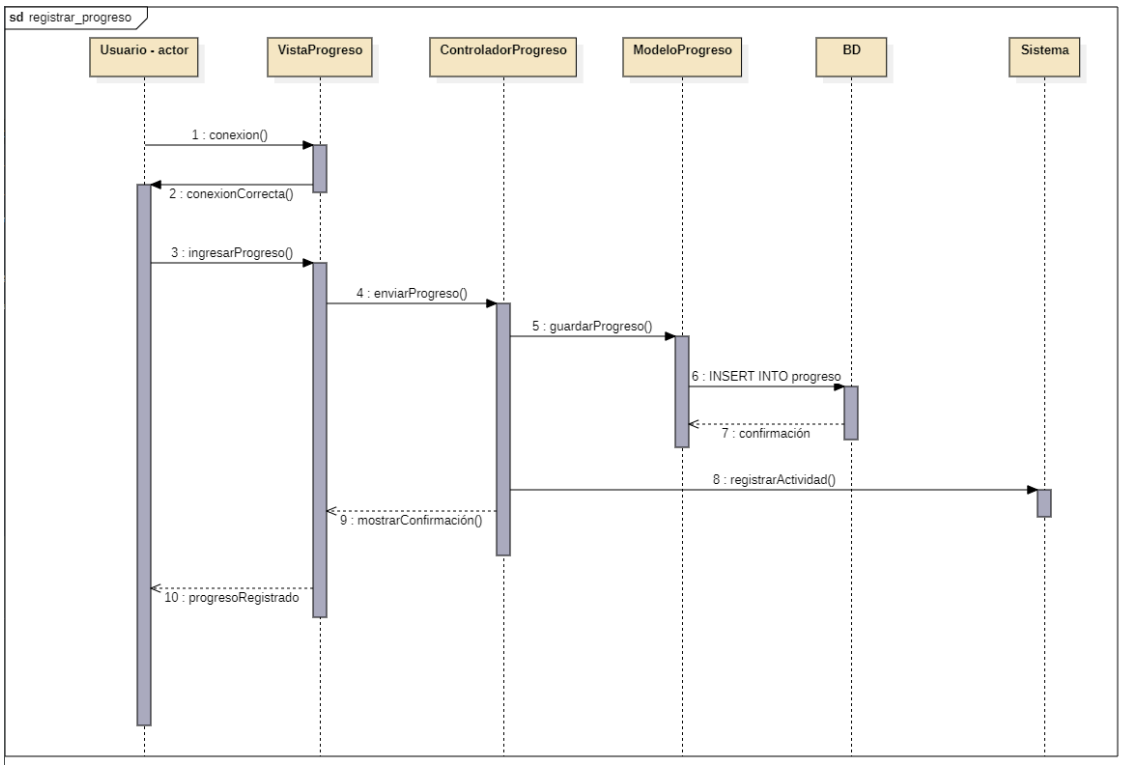
En este diagrama se ilustra el proceso mediante el cual el usuario solicita visualizar su rutina asignada.

El controlador de rutinas recibe la solicitud, consulta los datos en el modelo de Rutina y devuelve la información correspondiente al usuario, mostrando ejercicios, repeticiones y objetivos.



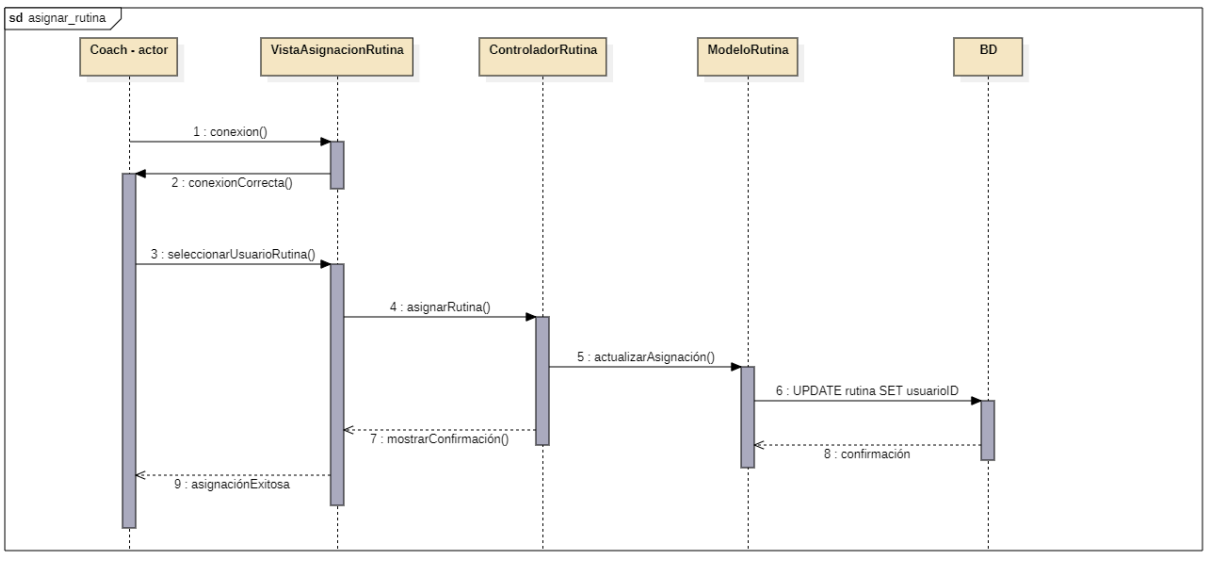
REGISTRAR PROGRESO

Este diagrama muestra el caso de uso en el que un usuario registrar su progreso de completado de rutinas, hasta como este se carga en la Base de Datos.



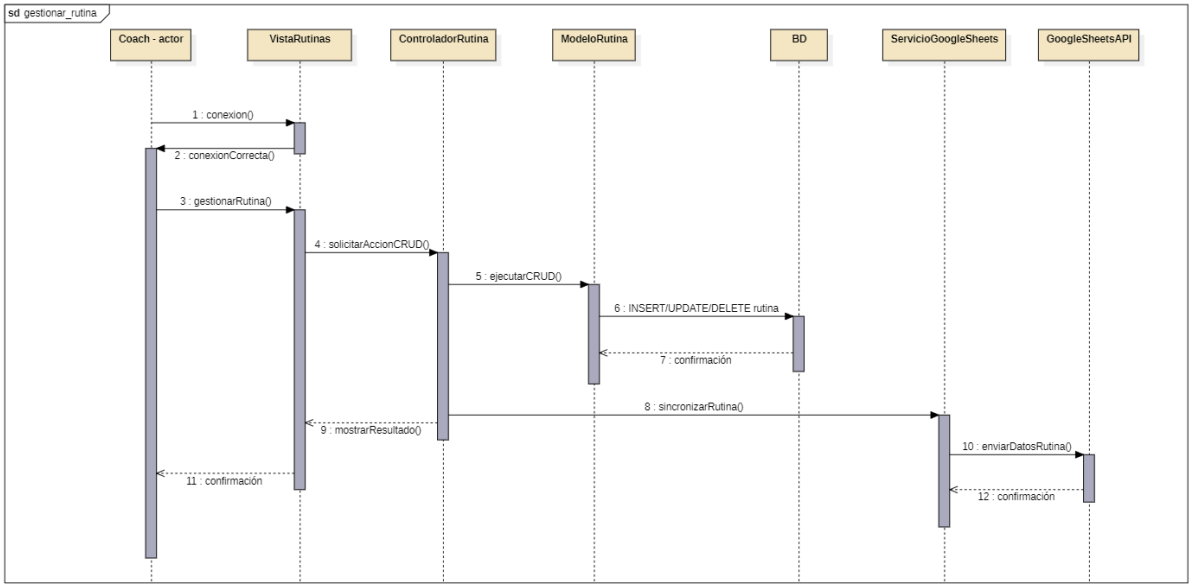

ASIGNAR RUTINA

Este diagrama ilustra la secuencia de un Coach asignando una rutina específica a un usuario específico.



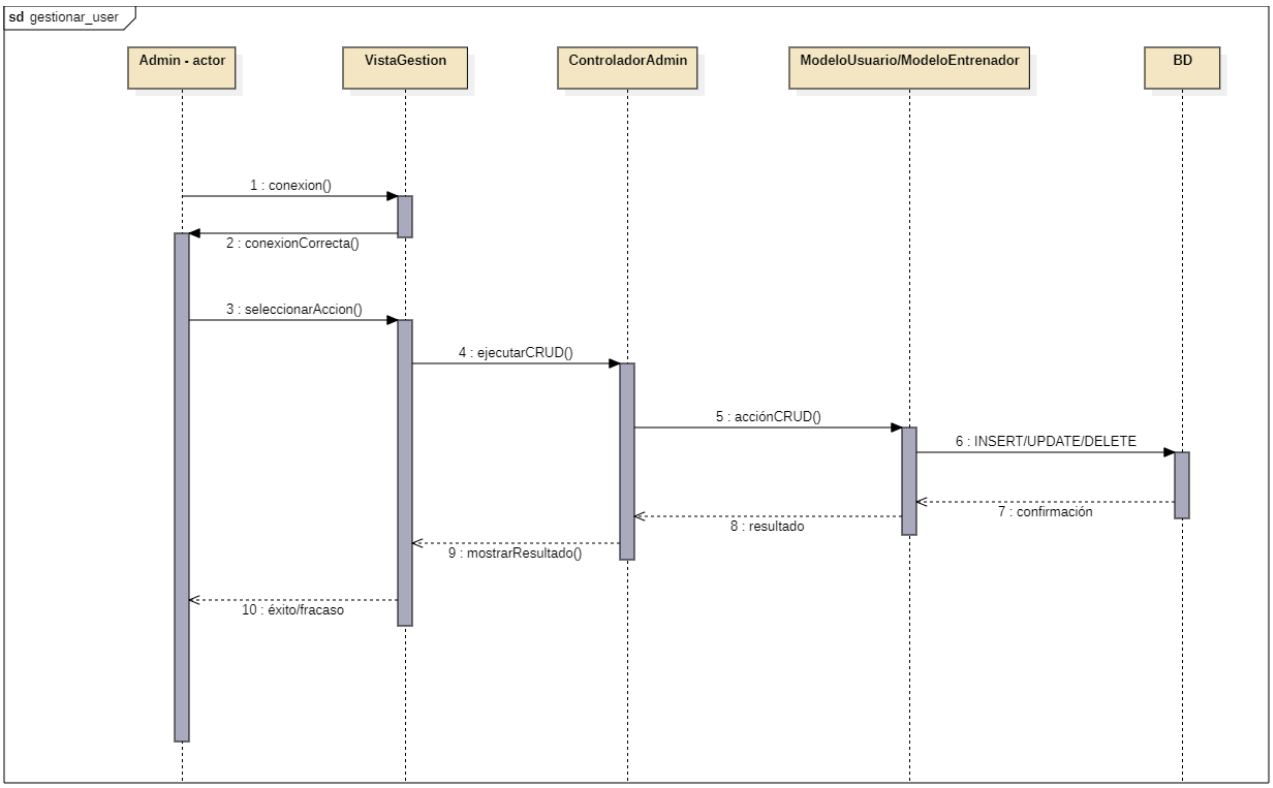
GESTIONAR RUTINA

Esta secuencia demuestra cómo el Coach gestiona las rutinas de su plantilla de Sheets asignada, pudiendo ejecutar un CRUD en estas.



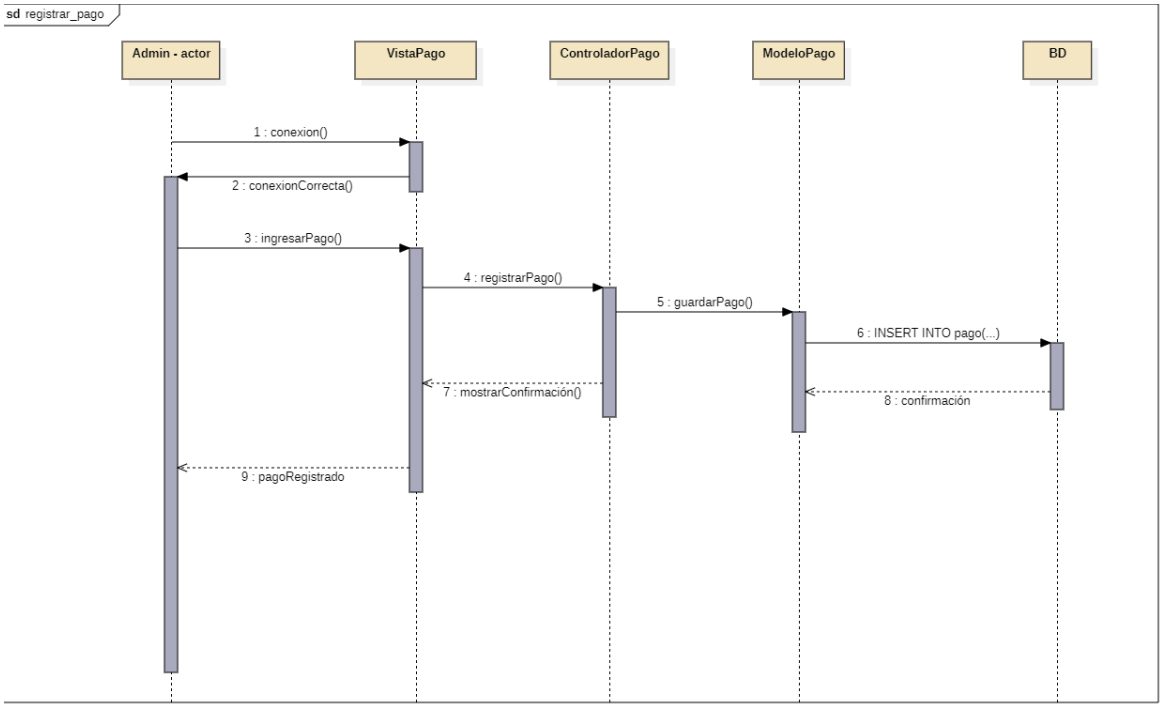

GESTIONAR USUARIO

Este gráfico ilustra la secuencia de un administrador ejecutando un CRUD a un usuario x de la Base de Datos.



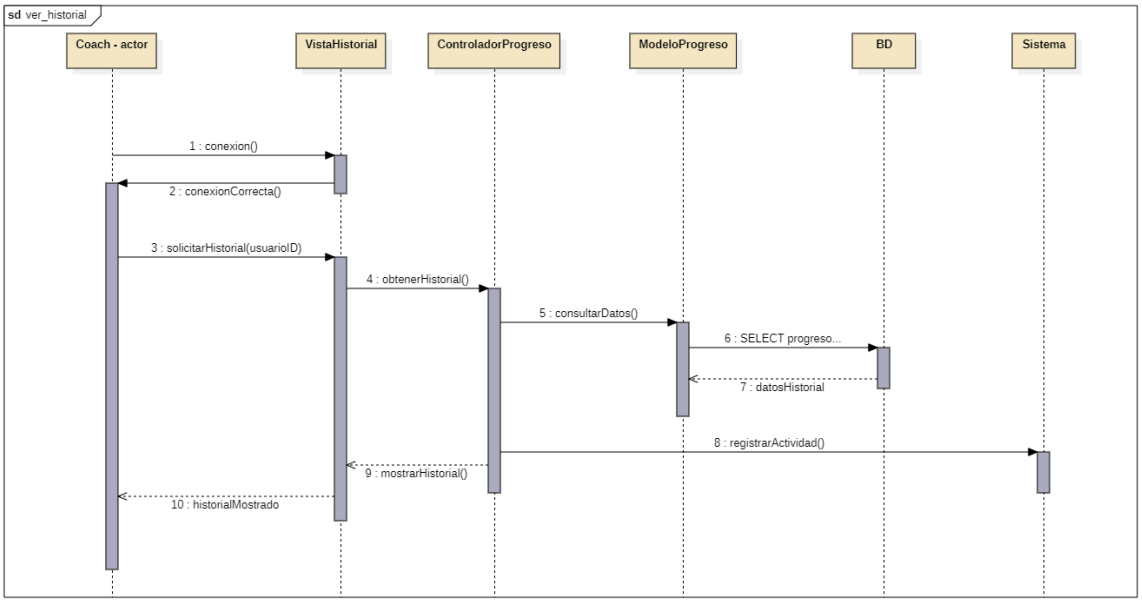
REGISTRAR PAGO

Esta secuencia muestra el proceso de registrar un dia de pago para un usuario.



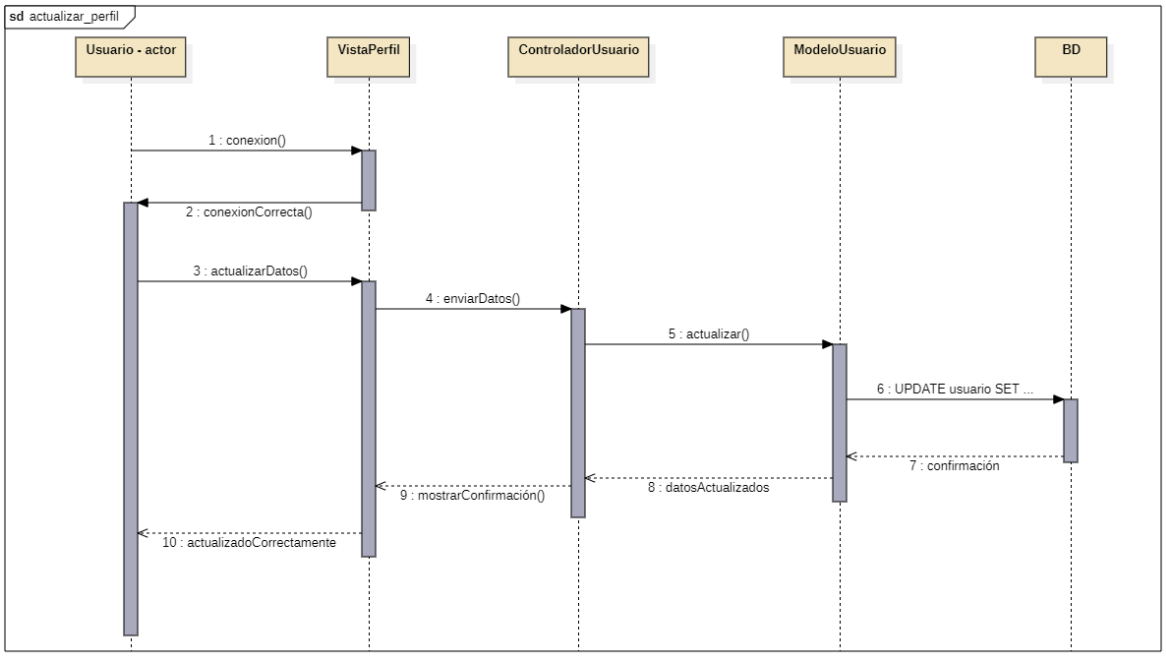

VER HISTORIAL

Esta secuencia ilustra cómo el Coach puede ver el historial de rutinas completadas o no completadas de sus usuarios asignados, pudiendo así entregar un feedback.



ACTUALIZAR PERFIL

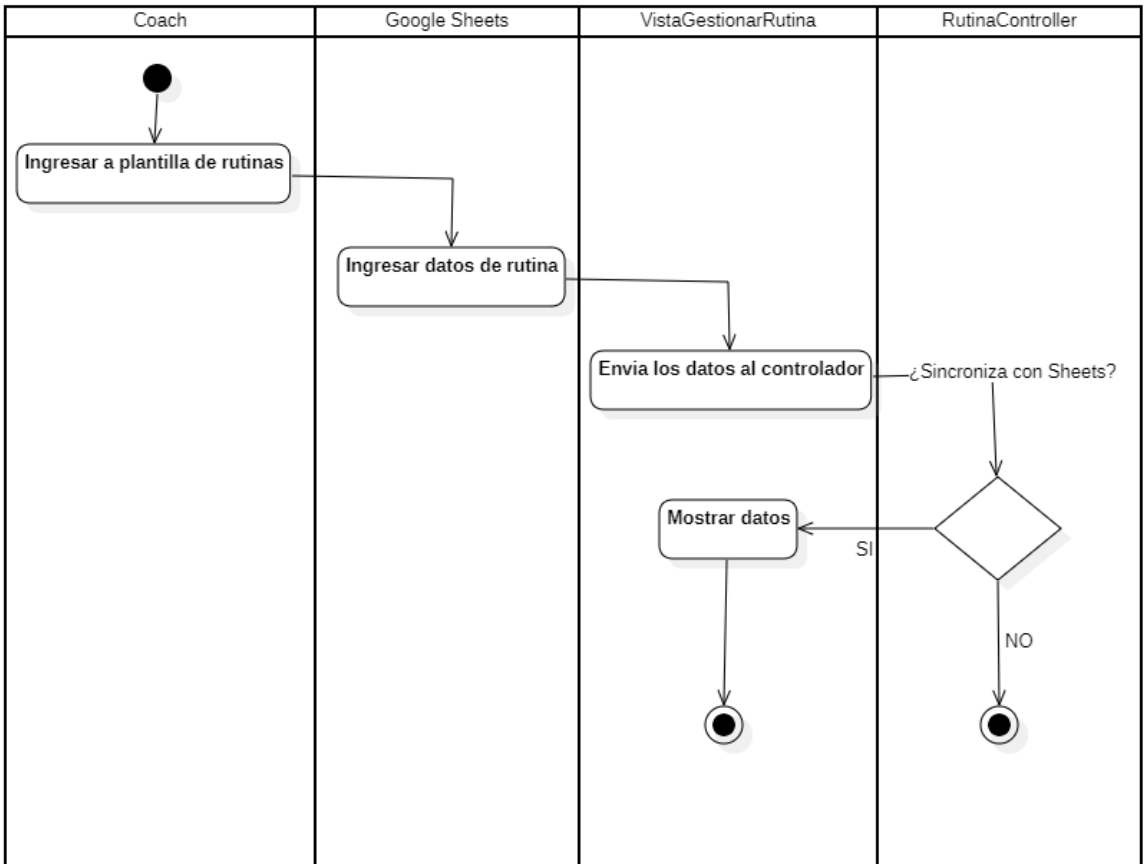
Esta secuencia muestra el proceso de UPDATE de perfil de usuario por parte de un usuario.



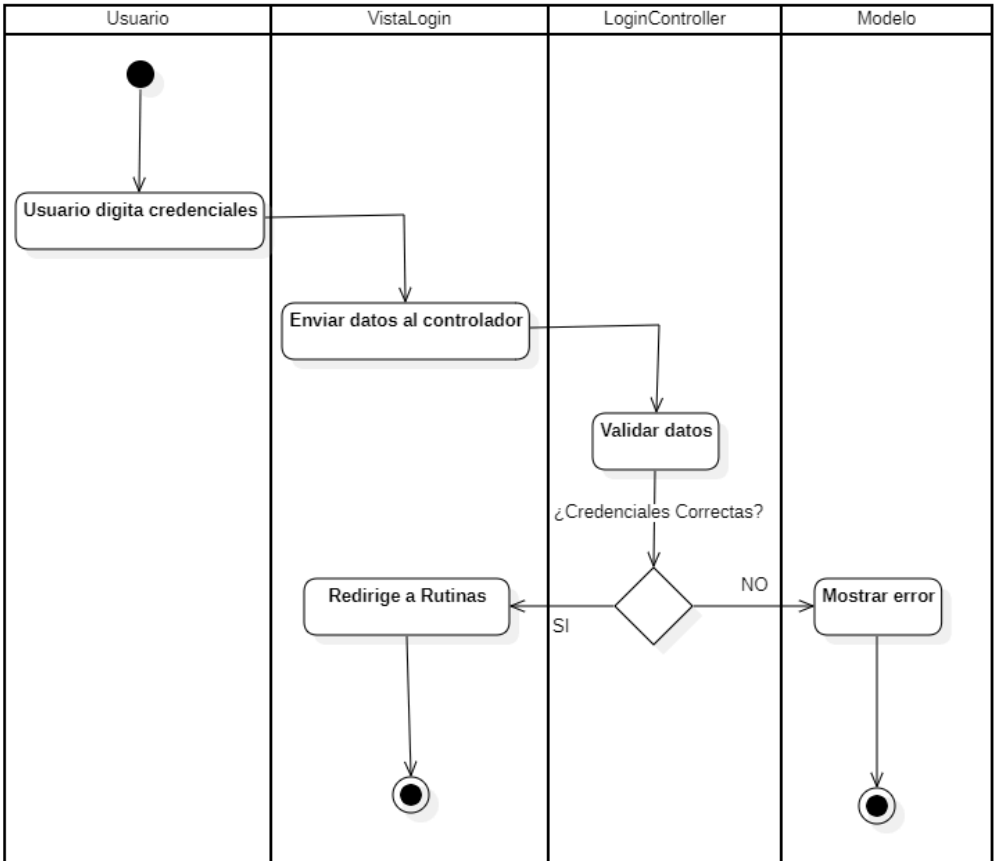
# Vista de Procesos

A continuación, se presentan los principales diagramas de actividad, los cuales ilustran el flujo de mensajes entre los controladores y las clases del modelo durante la ejecución de los casos de uso más relevantes:

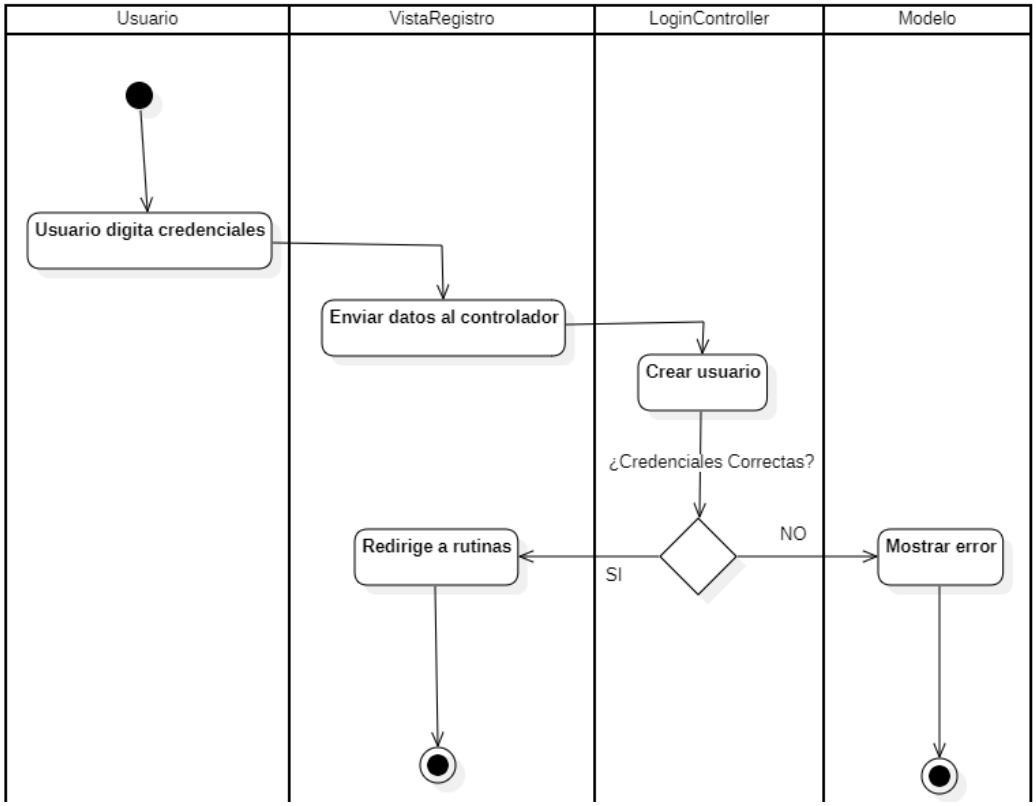
ACTIVIDAD GESTIONAR RUTINAS



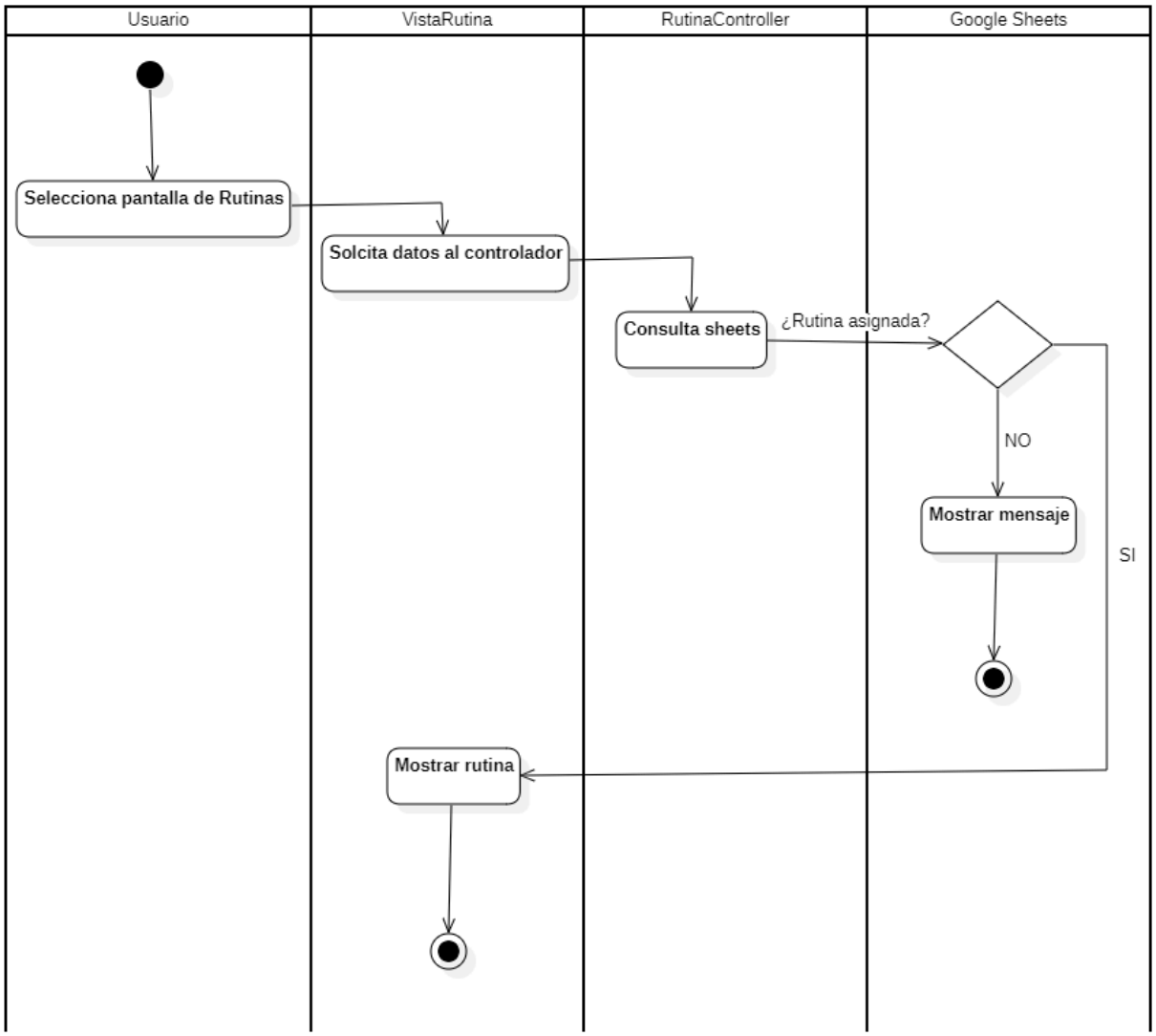
ACTIVIDAD INICIAR SESIÓN




ACTIVIDAD REGISTRARSE



ACTIVIDAD VER RUTINAS



# Vista de Desarrollo

La vista de desarrollo muestra cómo se estructura internamente el sistema Rayostrength, dividiendo sus componentes principales en Frontend (Aplicación móvil) y Backend (API y base de datos).

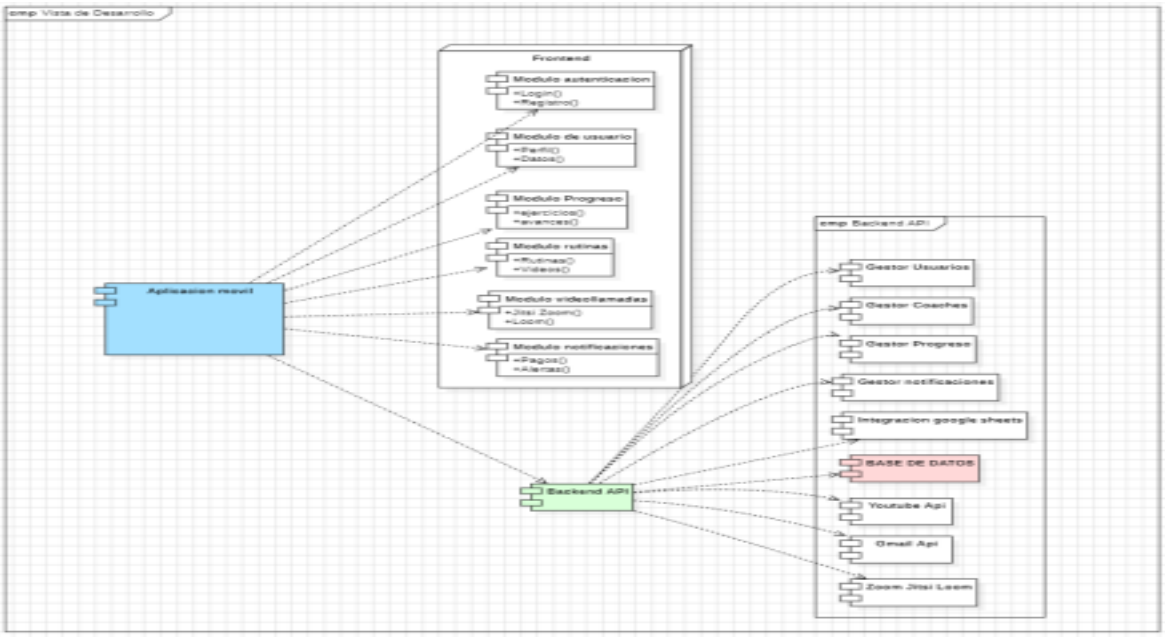
En el Frontend, la aplicación móvil se organiza en módulos:

- Autenticación: login y registro de usuarios.
- Usuario: perfil y datos personales.
- Progreso: ejercicios y avances.
- Rutinas: acceso a rutinas y videos desde Google Sheets o YouTube.
- Videollamadas: conexión con Jitsi o Loom.
- Notificaciones: alertas y recordatorios.

El Backend gestiona la lógica del sistema mediante componentes como:

- Gestores de usuarios, coaches, progreso y notificaciones.
- Integración con Google Sheets y APIs externas (YouTube, Gmail, Jitsi/Loom).
- Base de datos MySQL para almacenamiento.

La comunicación entre la app y el backend se realiza mediante servicios REST, asegurando una arquitectura modular, escalable y mantenible, alineada con la metodología ágil SCRUM.



# Vista Física

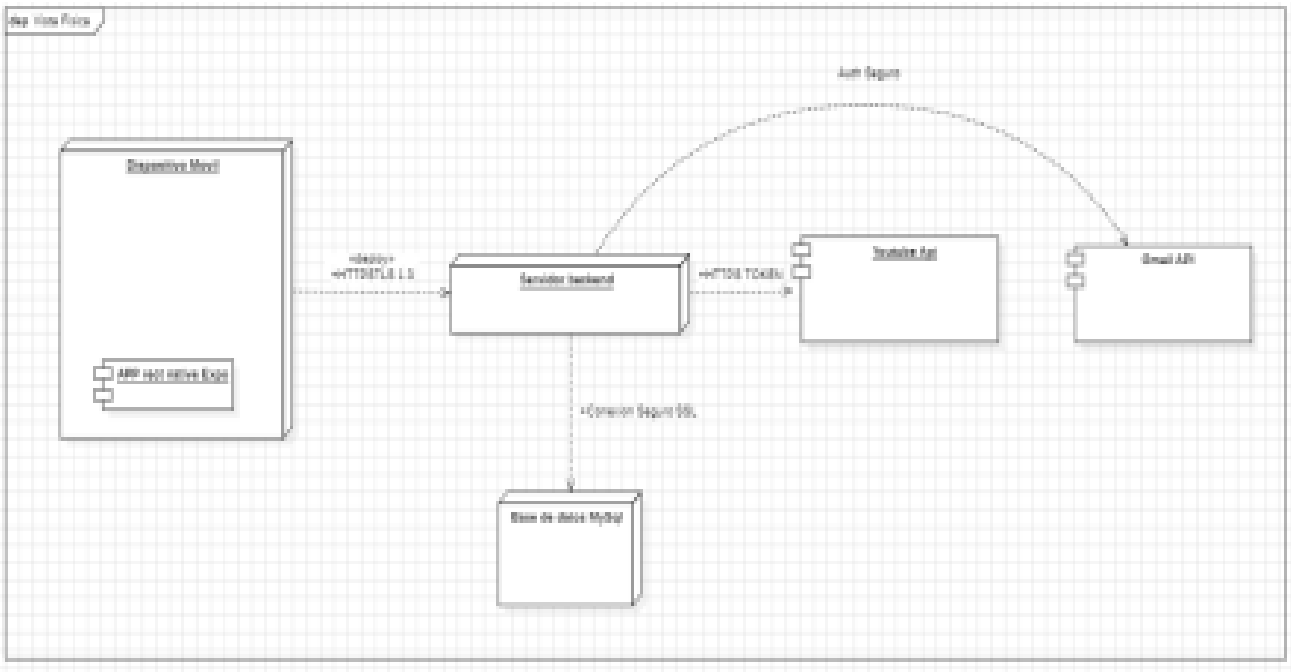
La vista física representa la arquitectura de despliegue del sistema *Rayostrength*, mostrando cómo los diferentes componentes del software se ejecutan sobre la infraestructura física y los servicios externos.

El sistema se compone de los siguientes nodos principales:

- Dispositivo móvil: ejecuta la aplicación desarrollada en React Native con Expo, permitiendo la interacción directa del usuario.
- Servidor Backend: gestiona la lógica de negocio, procesamiento de datos y comunicación con servicios externos.
- Base de datos MySQL: almacena de forma segura la información de usuarios, rutinas y progreso.
- APIs externas (YouTube y Gmail): permiten la integración de videos y envío de notificaciones automáticas.

La comunicación entre los componentes se realiza mediante conexiones seguras HTTPS/TLS 1.3 y autenticación mediante tokens, garantizando la confidencialidad y seguridad de los datos.

Esta infraestructura favorece la escalabilidad, disponibilidad y rendimiento del sistema.



# Vista de Escenarios

La vista de escenarios describe las principales interacciones entre los actores y el sistema Rayostrength, representando las funcionalidades clave desde la perspectiva del usuario.

Este diagrama permite comprender los requerimientos funcionales del sistema y sirve como base para los demás modelos (lógico, de procesos, de desarrollo y físico).

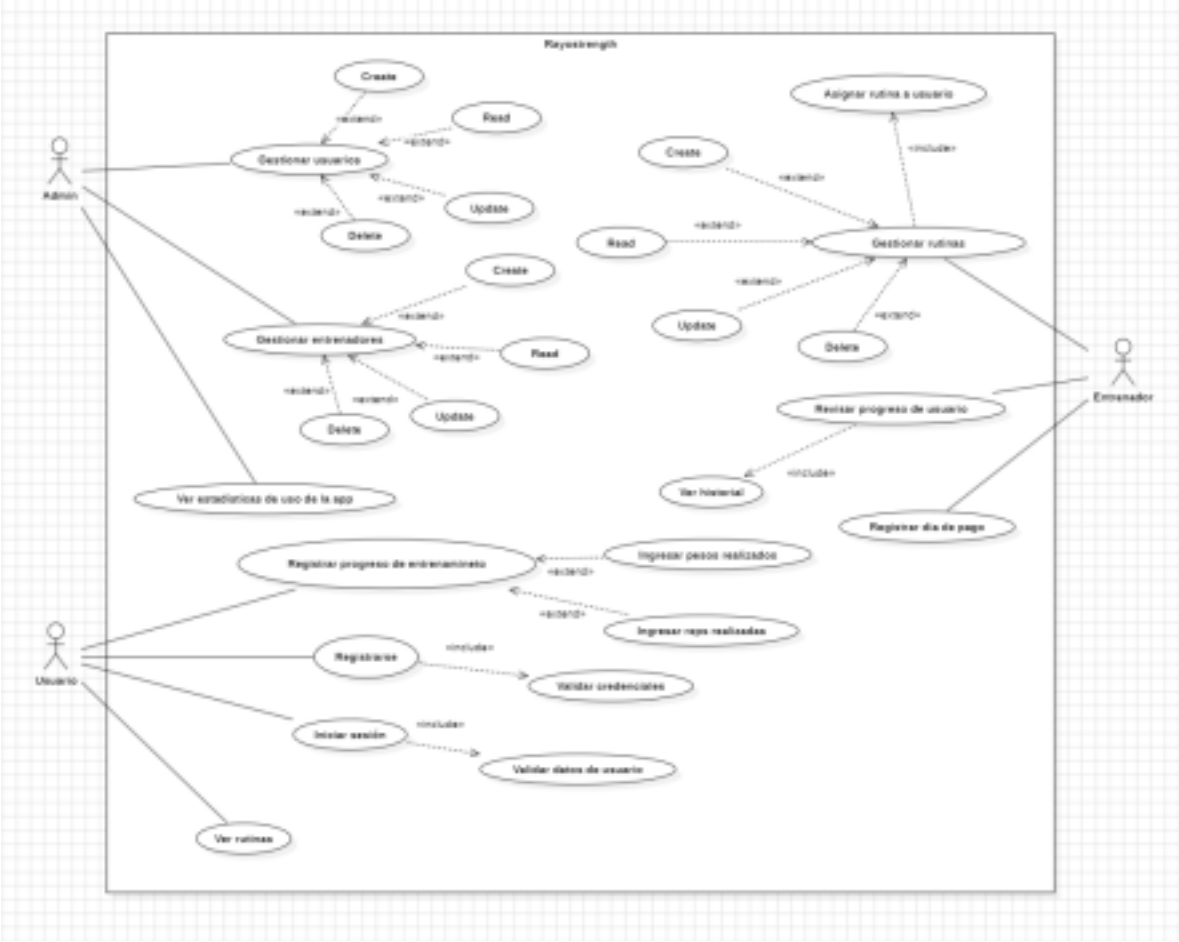
Los actores principales son:

- Usuario: puede registrarse, iniciar sesión, visualizar rutinas y registrar su progreso de entrenamiento.
- Entrenador: asigna rutinas, revisa el progreso de los usuarios y registra los pagos de entrenamiento.
- Administrador: gestiona usuarios, entrenadores, rutinas y puede consultar estadísticas de uso de la aplicación.

Cada caso de uso principal (como *Gestionar usuarios*, *Registrar progreso* o *Asignar rutina*) se descompone en acciones específicas (Create, Read, Update, Delete) e incluye o extiende otros procesos según la necesidad.

Esta vista muestra de forma general cómo los distintos roles interactúan con la aplicación para cumplir sus objetivos, siendo un punto de partida esencial para el diseño funcional y la planificación ágil.

DIAGRAMA DE CASO DE USO



# Conclusión

El modelo arquitectónico 4+1 aplicado en el proyecto Rayostrength permitió representar de manera integral la estructura y funcionamiento del sistema, abordando sus distintos niveles desde la perspectiva lógica, de procesos, desarrollo, física y de escenarios.

A través de estas vistas, se logró definir:

- La organización interna del sistema mediante el modelo lógico y sus clases.
- El flujo de interacción y comportamiento del sistema con los diagramas de secuencia en la vista de procesos.
- La distribución modular y responsabilidades del software con el diagrama de componentes en la vista de desarrollo.
- La infraestructura tecnológica y despliegue en la vista física, donde se detalla la relación entre la aplicación móvil, el backend y los servicios externos.
- Y finalmente, la interacción entre usuarios y el sistema mediante los casos de uso en la vista de escenarios.

Este enfoque permitió validar que la arquitectura de Rayostrength es coherente, escalable y alineada con los objetivos funcionales del proyecto. Además, facilitó la comunicación entre los miembros del equipo de desarrollo, sirviendo como guía para la implementación técnica y la toma de decisiones en futuras iteraciones.

En conclusión, el modelo 4+1 proporcionó una visión completa del sistema, asegurando una arquitectura bien estructurada y adaptable, que respalda el éxito del desarrollo y mantenimiento de la aplicación.