2023

Journeymate

Desarrollo de sistemas en red

Axel Jordano Morales Utrera

Daniel Eduardo Anota Paxtian

Contenido

1. Introducción

Journeymate se trata de una solución a un problema específico que encontramos, el no saber que actividades hacer cuando se viaja a un lugar nuevo, este documento tiene la finalidad de documentar distintos apartados sobre el desarrollo del proyecto Journeymate, cuenta con distintos apartados entre ellos requerimientos del sistema, en este apartado se encuentran distintas secciones en las que se incluyen distinta información relevante y necesaria para poder desarrollar un sistema como, requerimientos funcionales y no funcionales, entre otros apartados incluidos en el documento se encuentran artefactos de diseño, construcción, pruebas y despliegue del sistema.

Journeymate se desarrolló bajo la arquitectura de microservicios debido a que se determinó que era lo más viable debido a la naturaleza de la solución.

2. Requerimientos

2.1. Contexto.

Journeymate es una aplicación diseñada para los amantes de los viajes y las actividades en ciudades. Con Journeymate, puedes crear y personalizar rutinas que se ajusten a tus intereses y preferencias. Agrega tareas específicas a tus rutinas y descubre información relevante para hacer de tus viajes una experiencia inolvidable. Explora las rutinas de otros usuarios y encuentra inspiración para tus próximas aventuras. Journeymate es tu compañero de viaje perfecto para organizar y disfrutar de cada momento de tus viajes y actividades en ciudades.

2.2. Clases de usuario.

En Journeymate hemos diseñado dos clases de usuarios para brindar una experiencia personalizada y satisfactoria para todos. Estas clases de usuario son "Usuario" y "Administrador", cada una con características y capacidades distintas.

Comencemos hablando de los "Usuarios". Estos son los usuarios normales que utilizan la aplicación de manera recurrente para planificar y disfrutar de sus rutinas de viaje y actividades. Con Journeymate, los Usuarios tienen acceso a una gama de funciones que les permiten crear y personalizar rutinas de viaje según sus intereses y preferencias. El primer caso de uso, "Iniciar sesión" (CU-01), les brinda a los Usuarios la posibilidad de acceder a sus perfiles personales y utilizar todas las características de la aplicación de forma personalizada. Además, los Usuarios también pueden registrarse en Journeymate (CU-02), lo que les permite crear una cuenta y disfrutar de todos los beneficios que ofrece la aplicación.

La personalización es clave en Journeymate, y es por eso que los Usuarios tienen la opción de personalizar su perfil (CU-03) según sus preferencias individuales. Pueden agregar información personal, incluyendo su nombre, foto de perfil y cualquier otra información relevante que deseen compartir con la comunidad de Journeymate. Una vez que han

configurado su perfil, los Usuarios están listos para comenzar a crear y compartir rutinas de viaje personalizadas. Con la capacidad de crear rutinas de viaje (CU-04), los Usuarios pueden planificar sus actividades día a día, agregar tareas específicas a cada rutina y ajustar los detalles según sea necesario.

La flexibilidad es otra característica destacada de Journeymate. Los Usuarios tienen la opción de editar sus rutinas de viaje (CU-05) en cualquier momento, lo que les permite adaptarse a cambios imprevistos o agregar nuevas experiencias a su itinerario. Además, pueden ver los detalles completos de cada rutina (CU-06), lo que les proporciona información relevante sobre los lugares a visitar y cualquier otra cosa que necesiten saber para asegurarse de tener una experiencia inolvidable.

Para hacer que la planificación de viajes sea aún más completa, Journeymate permite a los Usuarios crear tareas individuales (CU-07) que se pueden agregar a sus rutinas. Estas tareas pueden incluir actividades específicas, recordatorios o cualquier otra cosa que los Usuarios deseen tener en cuenta durante su viaje. La edición de tareas (CU-08) también está disponible para permitir ajustes o cambios en cualquier momento.

La interacción social es una parte integral de Journeymate. Los Usuarios pueden explorar las rutinas de viaje creadas por otros usuarios (CU-09), lo que les brinda una fuente inagotable de inspiración y nuevas ideas para sus propias aventuras. Además, también pueden ver las rutinas que han creado personalmente (CU-10) y seguir rutinas de otros usuarios (CU-11), lo que les permite mantenerse actualizados con las últimas actividades y descubrimientos de sus compañeros viajeros. Para facilitar el seguimiento de las rutinas seguidas, los Usuarios pueden acceder a una lista de rutinas seguidas (CU-12), lo que les permite tener un acceso rápido y conveniente a la información que necesitan y vieron de otros usuarios.

Ahora, pasemos a la clase de usuario "Administrador". Como su nombre lo indica, los Administradores tienen el poder de moderar y supervisar todas las rutinas creadas por los usuarios "comunes". Al igual que los Usuarios, los Administradores también pueden iniciar sesión en la aplicación (CU-01) y personalizar su perfil (CU-03) según sus preferencias individuales.

La principal diferencia radica en que los Administradores tienen acceso a una visión más amplia de la comunidad de Journeymate. Pueden explorar todas las rutinas de viaje, tanto públicas como privadas, creadas por los usuarios "comunes" (CU-09), lo que les permite tener una idea completa de las actividades que se están llevando a cabo en la plataforma. Además, los Administradores también pueden ver las rutinas que han sido creadas (CU-10), lo que les permite mantenerse informados sobre los itinerarios populares y realizar un seguimiento de la participación de la comunidad.

2.3. Casos de uso.

- CU-01. Iniciar sesión:
- CU-02. Registrarse.
- CU-03. Personalizar perfil.
- CU-04. Crear rutina de viaje.
- CU-05. Editar rutina de viaje.
- CU-06. Ver detalles de rutina.
- CU-07. Crear tarea
- CU-08. Editar tarea
- CU-09. Explorar rutinas de viaje.
- CU-10. Ver rutinas creadas.
- CU-11. Seguir rutina.
- CU-12. Ver rutinas seguidas.
- CU-13. Completar tarea.

2.4. Prototipos UI

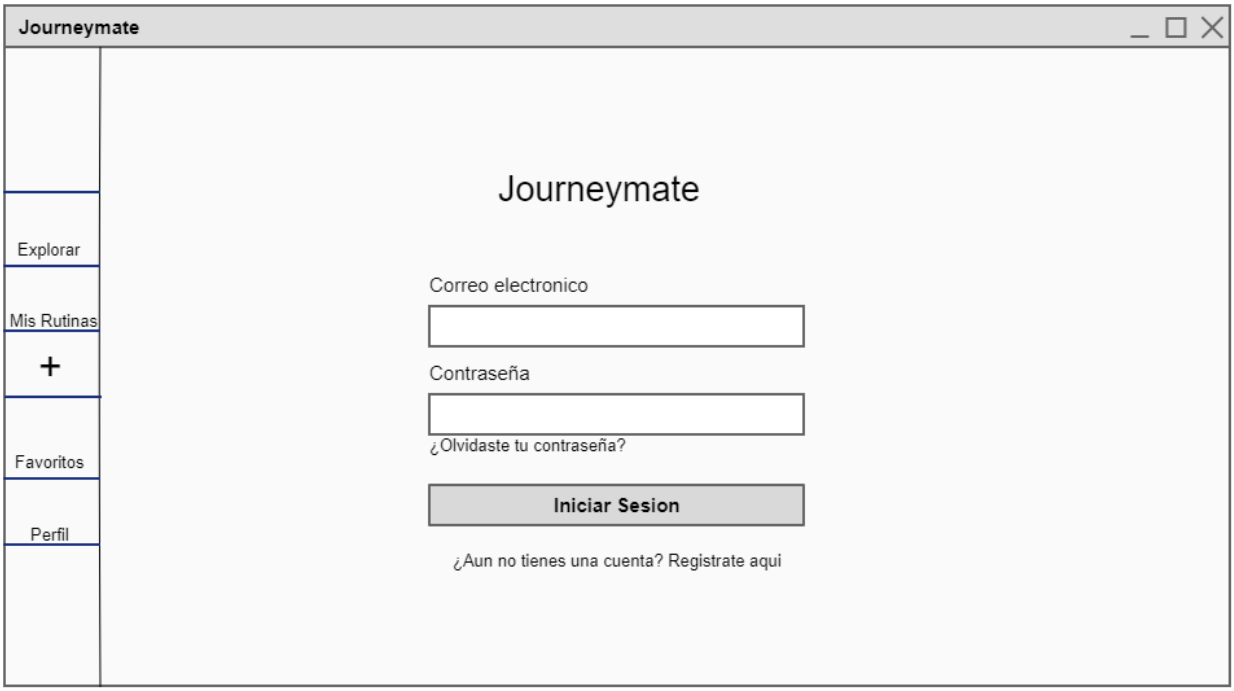


Ilustración 1 Prototipo Login

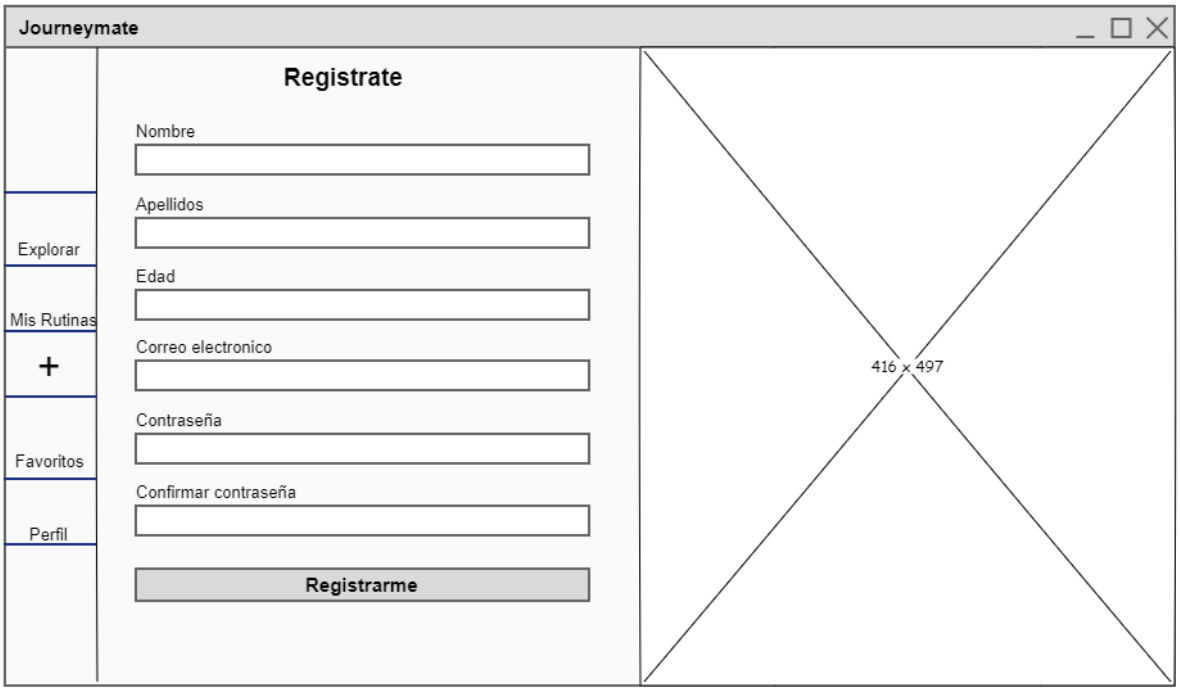


Ilustración 2 Prototipo Registrarse

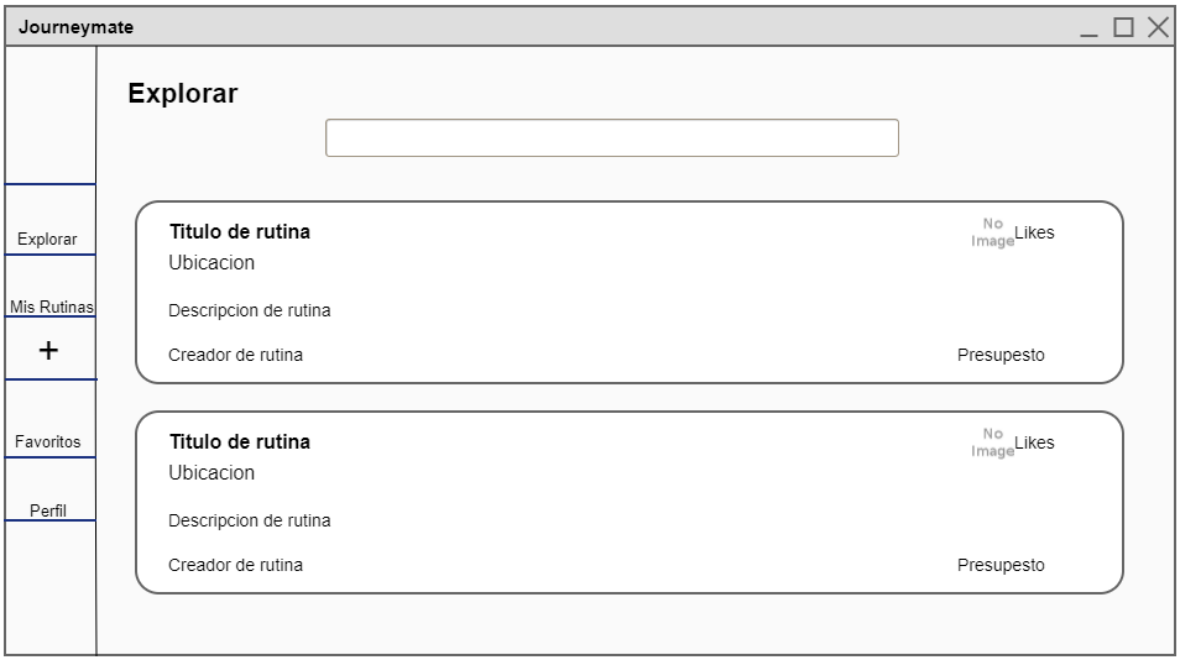


Ilustración 3 Prototipo Explorar

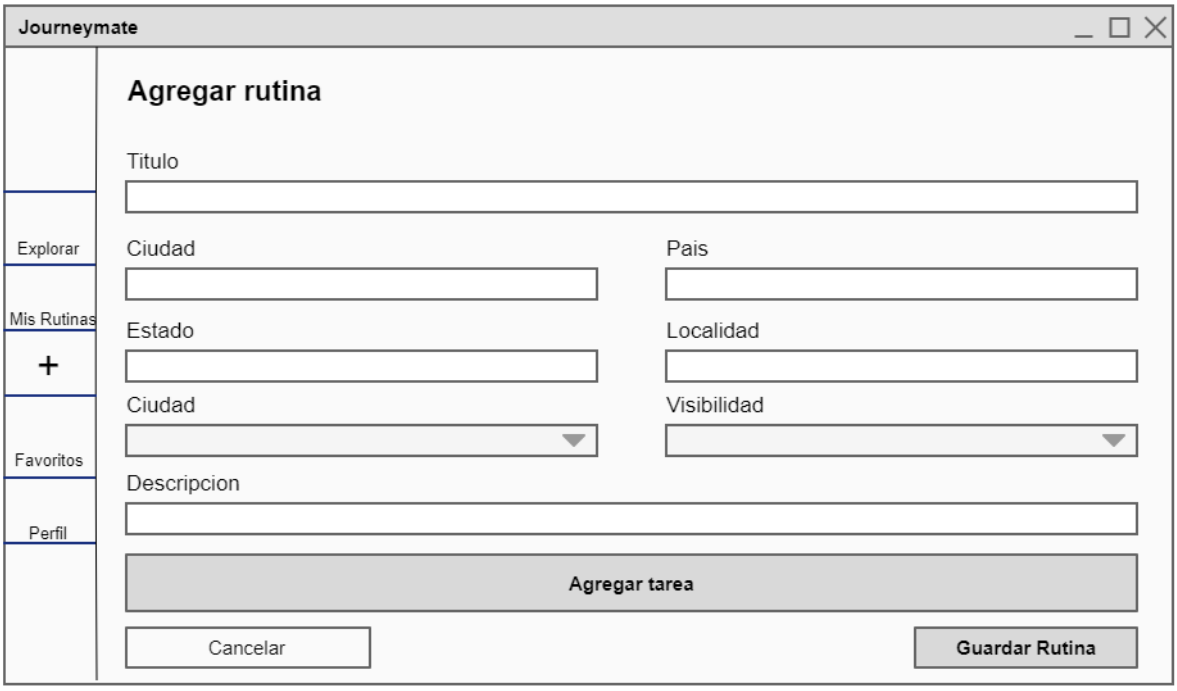


Ilustración 4 Prototipo Agregar Rutina

Ilustración 5 Prototipo Perfil de Usuario

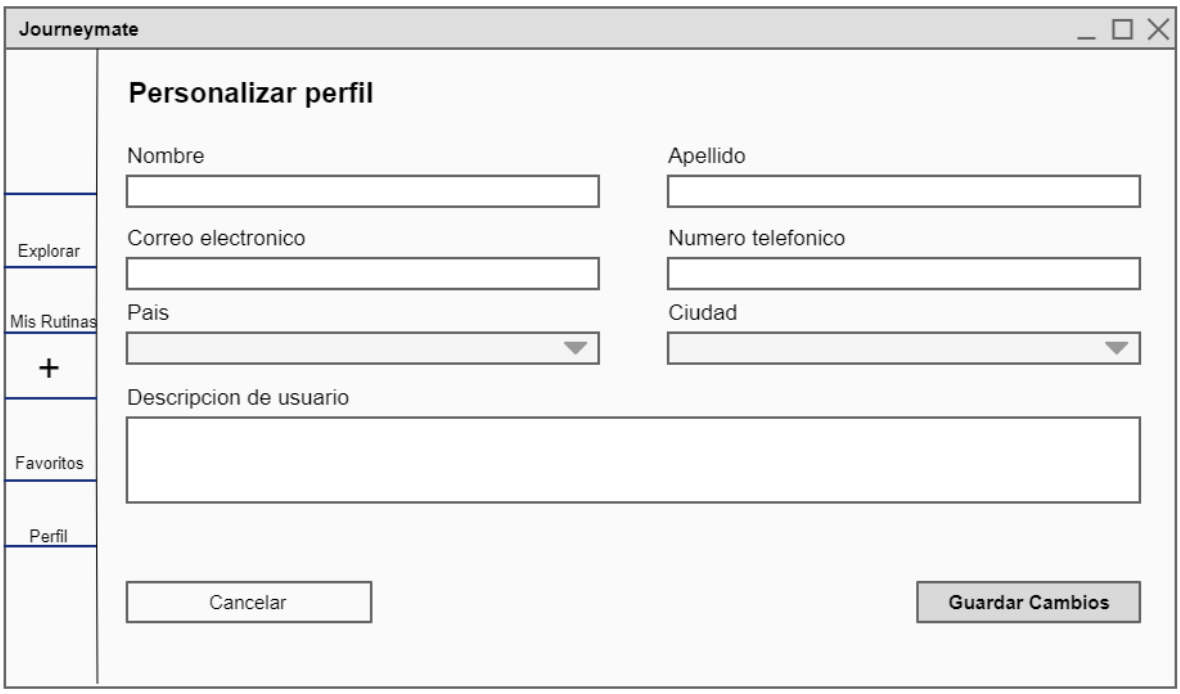


Ilustración 6 Prototipo Personalizar Perfil

Ilustración 7 Prototipo Detalles de Rutina

2.5. Requerimientos funcionales

- RF-01: Interfaz de inicio de sesión: El sistema debe proporcionar una interfaz de usuario intuitiva y fácil de usar que permita a los usuarios ingresar sus credenciales de inicio de sesión, como correo electrónico y contraseña.
- RF-02: Validación de credenciales: El sistema debe validar las credenciales ingresadas por el usuario para garantizar que sean correctas y correspondan a un usuario registrado en Journeymate.
- RF-03: Autenticación de usuario: Una vez que las credenciales son validadas con éxito, el sistema debe autenticar al usuario y establecer una sesión activa para ese usuario.
- RF-04: Gestión de errores de inicio de sesión: Si las credenciales proporcionadas son incorrectas, el sistema debe mostrar un mensaje de error adecuado sin exponer información de que es lo que esta incorrecto para garantizar la seguridad además de permitir que el usuario vuelva a intentar iniciar sesión con la información correcta.
- RF-05: Redirección a la página principal: Después de una autenticación exitosa, el sistema debe redirigir al usuario a la página principal de Journeymate, donde puede comenzar a hacer uso de las funcionalidades completas del sistema.
- RF-06: Seguridad de la sesión: El sistema debe garantizar la seguridad de la sesión del usuario, protegiendo la información confidencial y previniendo el acceso no autorizado a la cuenta del usuario.
- RF-07: Cierre de sesión: El sistema debe proporcionar una opción para que los usuarios cierren sesión de manera segura cuando hayan terminado de utilizar la aplicación.
- RF-08: nuevo registro de usuario: El sistema debe permitir a los nuevos usuarios registrarse en la aplicación Journeymate.
- RF-09: Validación de datos de registro: El sistema debe validar los datos ingresados por el usuario durante el proceso de registro, como el nombre, el correo electrónico, la contraseña, y la edad para garantizar que cumplan con los criterios establecidos (por ejemplo, longitud mínima, formato válido, etc.).
- RF-10: Comprobación de disponibilidad de correo electrónico: El sistema debe verificar la disponibilidad del nombre de usuario y la dirección de correo electrónico proporcionados por el usuario durante el registro, asegurándose de que no estén siendo utilizados por otro usuario registrado en la plataforma.
- RF-11: Creación de perfil de usuario: Después de un registro exitoso, el sistema debe crear un perfil de usuario único para el nuevo usuario en la base de datos de Journeymate, almacenando la información proporcionada durante el registro.
- RF-14: Mensajes de error de registro: Si los datos de registro proporcionados son inválidos o si ya existe un correo electrónico registrado, el sistema debe mostrar mensajes de error claros y descriptivos.

- RF-15: Redirección a la página de inicio de sesión: Después de un registro exitoso, el sistema debe redirigir al usuario a la página de inicio de sesión para que puedan acceder a su cuenta recién creada.
- RF-16: Edición de información personal: El sistema debe permitir a los usuarios editar y actualizar la información personal en su perfil.
- RF-17: Creación de nueva rutina de viaje: El sistema debe permitir a los usuarios la creación de una nueva rutina de viaje, permitiendo ingresar datos como el nombre de la rutina, lugar donde se llevar a acabo la rutina, la descripción entre otros datos relevantes en la rutina de viaje.
- RF-18: Edición de rutina de viaje: El sistema debe permitir al usuario propietario de una rutina de viaje, modificar la misma, permitiendo así la edición de todos los datos de la rutina de viaje.
- RF-19: Actualización de información de rutina de viaje: El sistema debe guardar los cambios en la base de datos el momento de editar los datos de la rutina de viaje.
- RF-20: Creación de nueva tarea: El sistema debe permitir la creación de tareas al momento de ingresar los datos para registrar una nueva rutina de viaje permitiendo así la captación y tratamiento de datos como lo son el nombre de la tarea, la descripción, el presupuesto, entre otros datos necesarios para la creación de la tarea.
- RF-21: Actualización de tarea: El sistema debe permitir al usuario propietario de la rutina de viaje, la actualización de una tarea existente en el sistema, permitiendo así la edición de todos los datos de la tarea.
- RF-22: Visualización de detalles de rutina: El sistema debe permitir al usuario la visualización de los detalles de la rutina, tanto los datos de esta, como las tareas que contiene la misma.
- RF-23: Visualización de todas las rutinas de los usuarios: El sistema debe permitir la visualización de todas las rutinas que los usuarios hayan creado con un estado público.
- RF-23: Visualizar rutinas de un usuario: El sistema debe permitir la visualización de las rutinas propias de un usuario.
- RF-24: Seguimiento de rutina: El sistema debe permitir al usuario guardar una rutina en sus favoritos para que así él la pueda realizar.
- RF-25: Visualización de rutinas seguidas: El sistema debe permitir la visualización de las rutinas que un usuario haya decidido seguir, estas rutinas deben mostrarse en un apartado por separado por llamado por ejemplo “Mis me gusta”.
- RF-26: Completar tarea: El sistema debe permitir al usuario marcar como completada una tarea solo con la pulsación de un botón.

2.6. Requerimientos no funcionales

- RNF-01: Seguridad: Journeymate debe garantizar la seguridad de la información de los usuarios, mediante el uso de mecanismos de autenticación y autorización confiables y robustos.
- RNF-02: Escalabilidad: Journeymate debe ser capaz de soportar un alto número de usuarios y transacciones sin afectar su rendimiento o disponibilidad.
- RNF-03: Usabilidad: Journeymate debe contar con una interfaz de usuario amigable e intuitiva, que permita a los usuarios navegar fácilmente por la aplicación y realizar sus tareas de forma eficiente.
- RNF-04: Disponibilidad: Journeymate debe estar disponible las 24 horas del día, los 7 días de la semana, con un tiempo de inactividad mínimo planificado para el mantenimiento y actualizaciones del sistema.
- RNF-05: Portabilidad: Journeymate debe ser compatible con diferentes dispositivos móviles y sistemas operativos, para que los usuarios puedan acceder a la aplicación desde cualquier lugar y en cualquier momento.
- RNF-06: Integración: Journeymate debe ser capaz de integrarse con otros sistemas y servicios externos, como servicios de pago, servicios de geolocalización, entre otros, para ofrecer una experiencia completa a los usuarios.
- RNF-07: Rendimiento: Journeymate debe ser capaz de procesar y entregar los datos en tiempo real, sin retrasos significativos, para que los usuarios puedan interactuar con la aplicación sin interrupciones.

3. Diseño

3.1. Diseño arquitectónico

3.1.1. Vista de casos de uso.

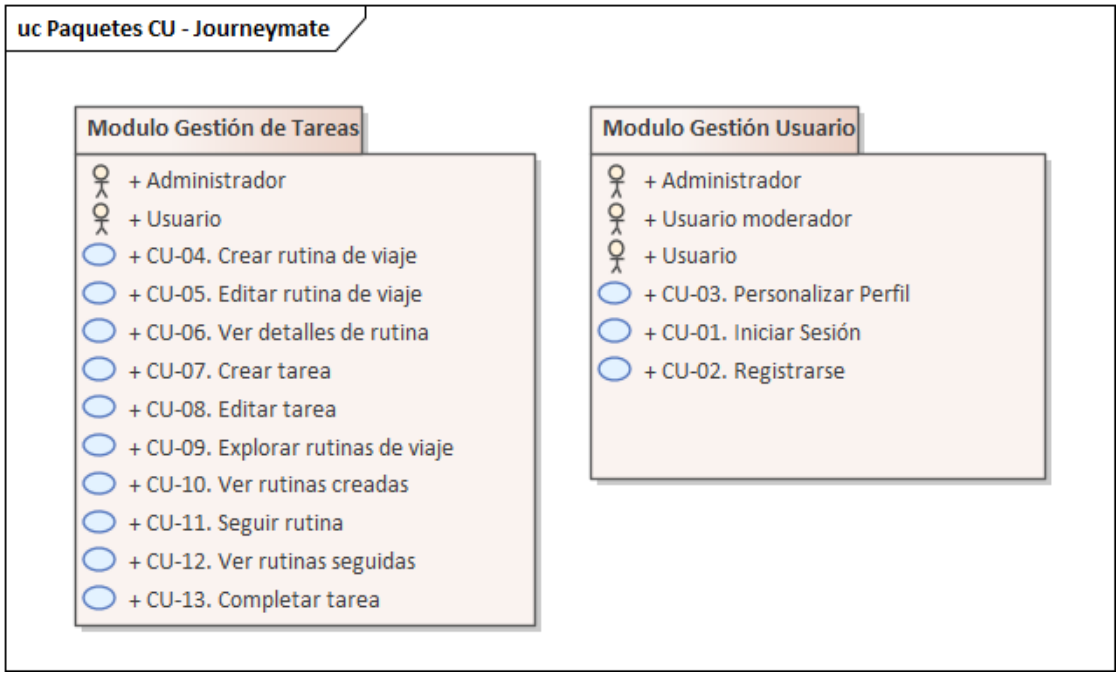


Ilustración 8 Diagrama de CU por paquete

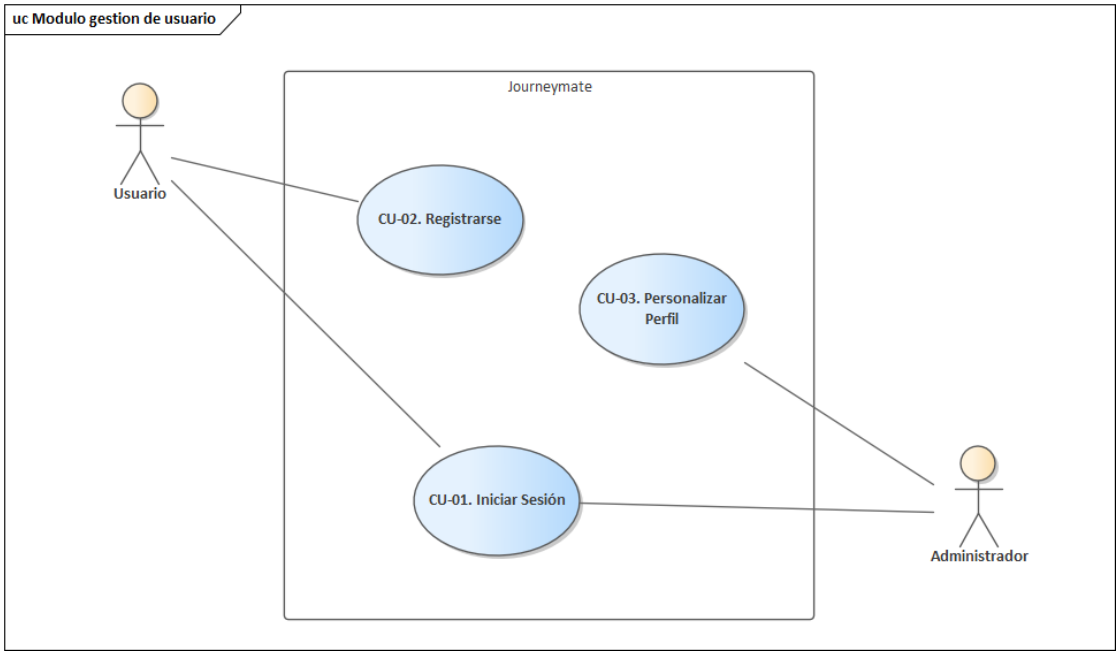


Ilustración 9 Diagrama de caso de uso paquete "Modulo gestión de usuario"

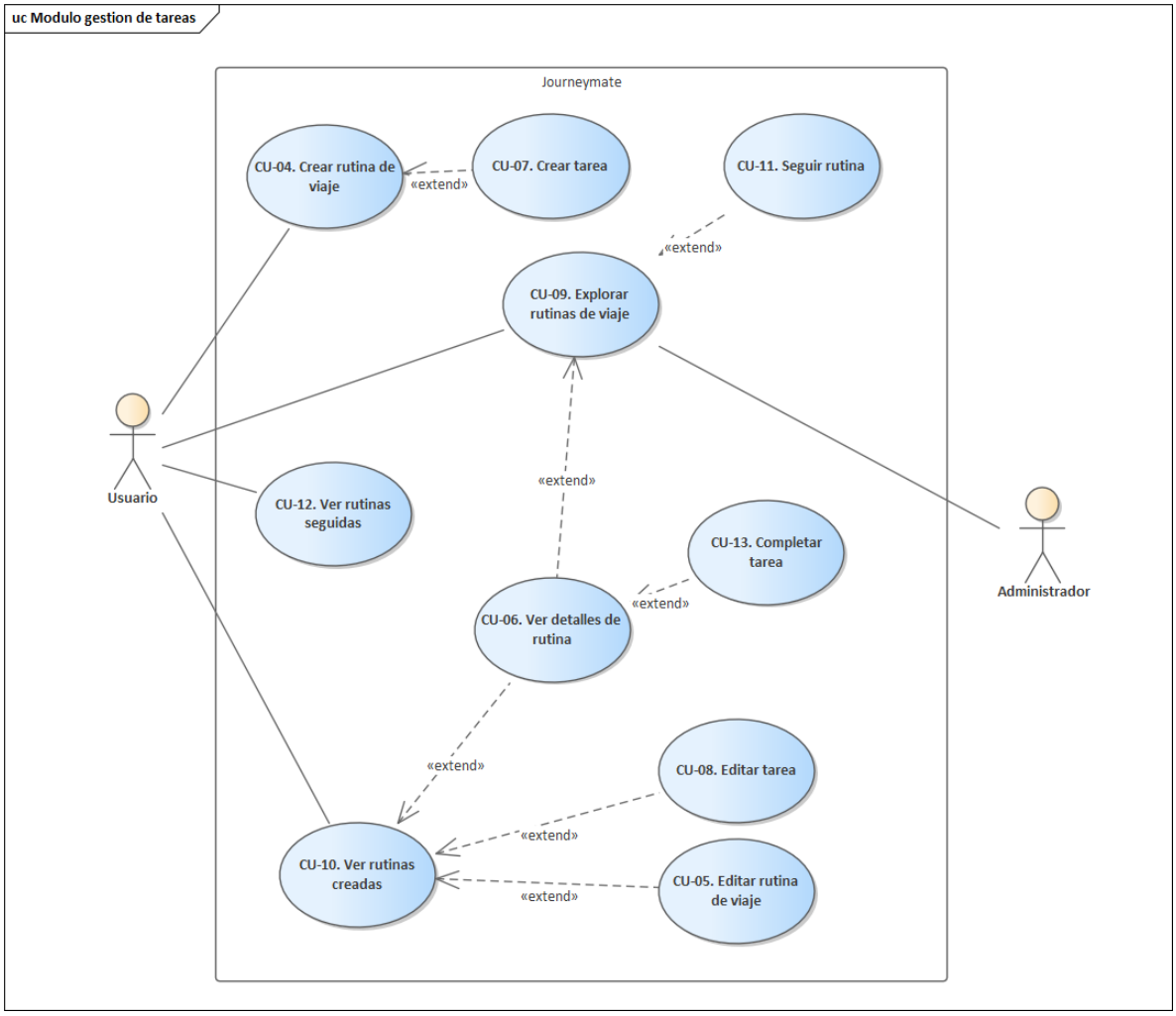


Ilustración 10 Diagrama de paquete de caso "Modulo gestión de tareas"

3.1.2. Vista lógica

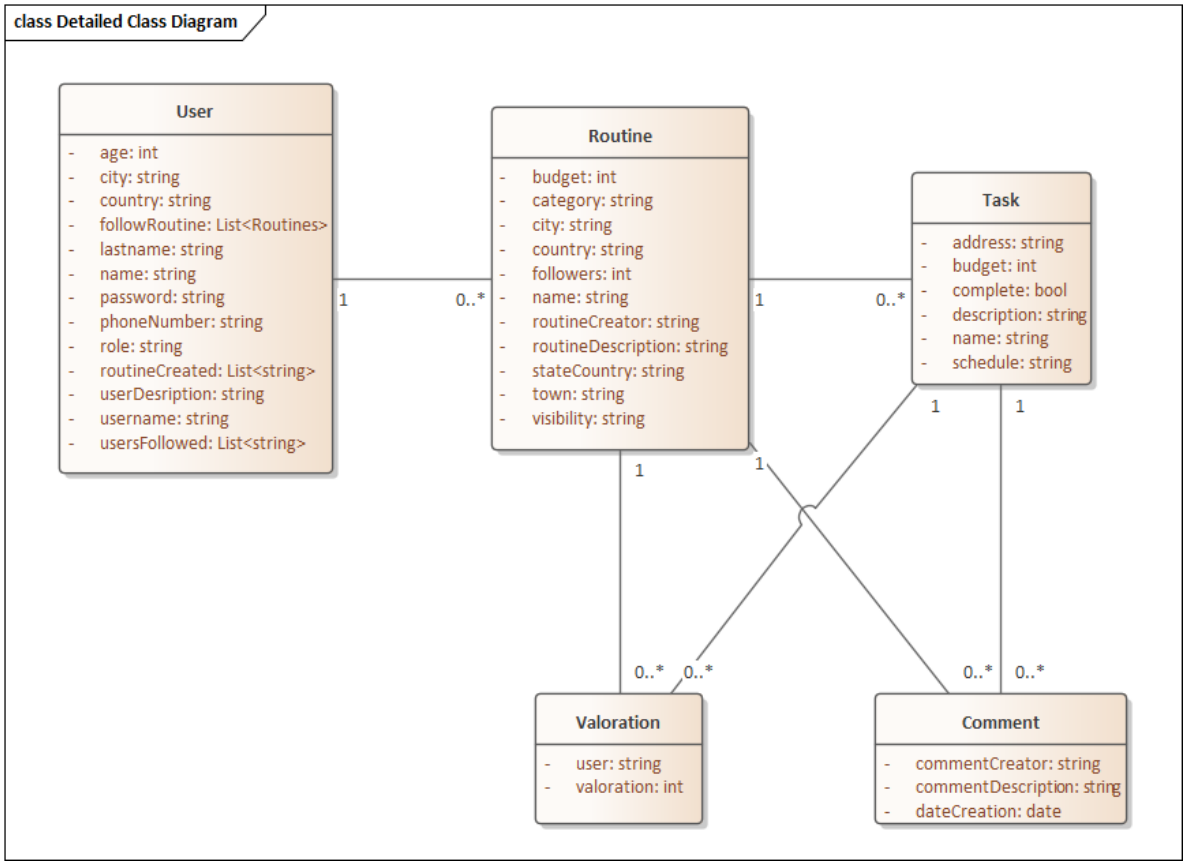


Ilustración 11 Diagrama de clases Journeymate

3.1.3. Vista de implementación

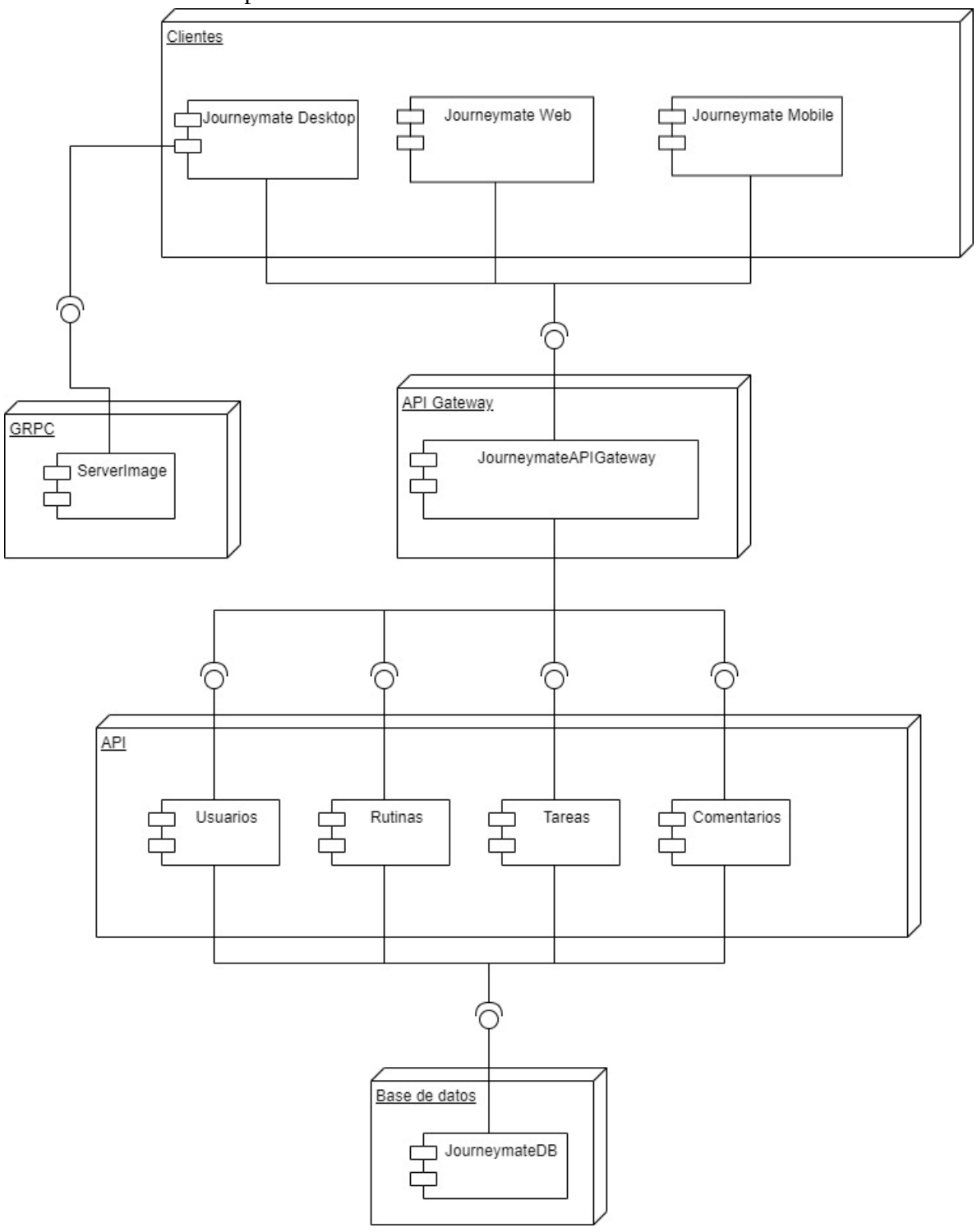


Ilustración 12 Diagrama de componentes Journeymate

3.1.4. Vista de procesos

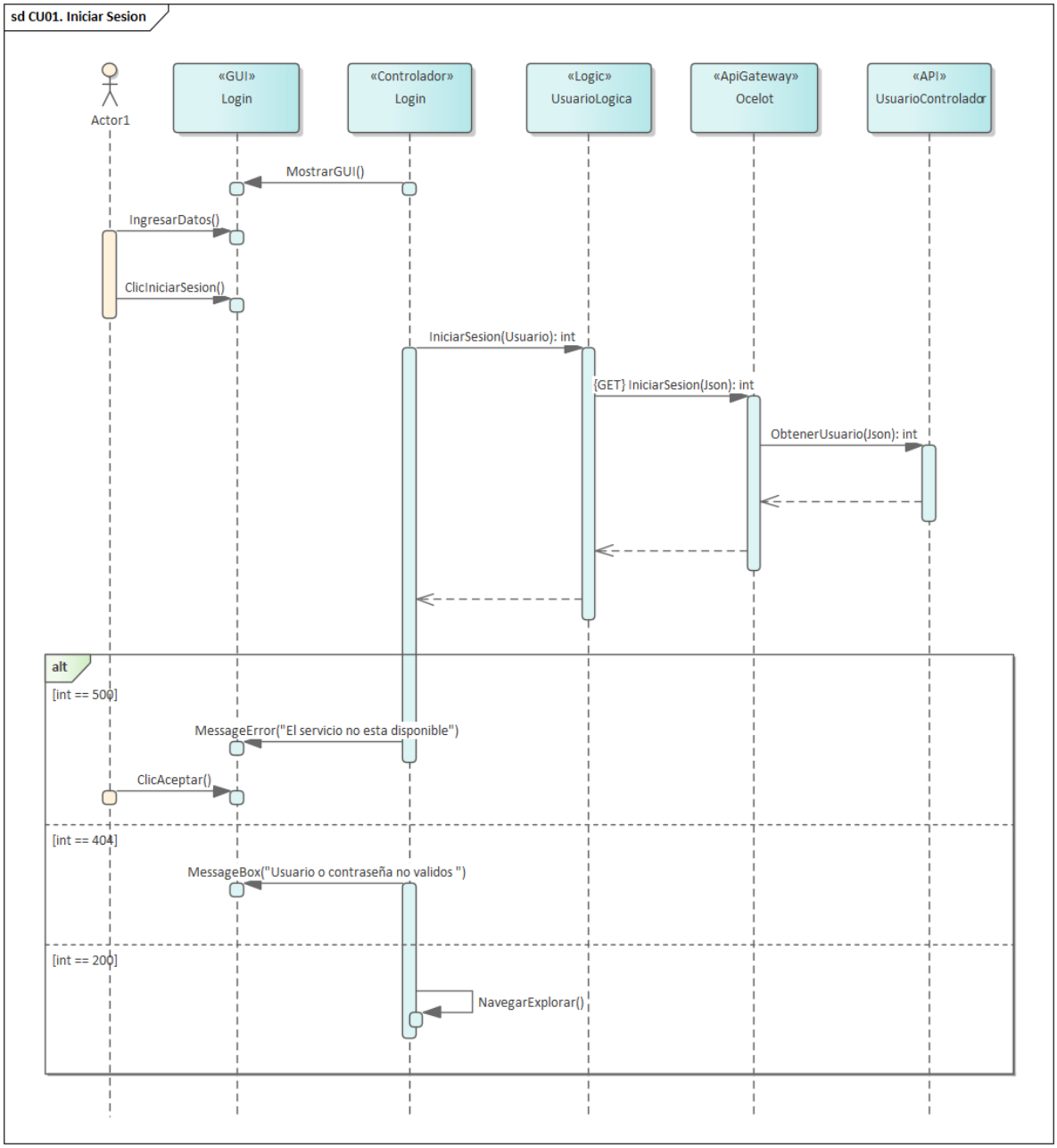


Ilustración 13 Diagrama de secuencia CU-01. Iniciar sesion

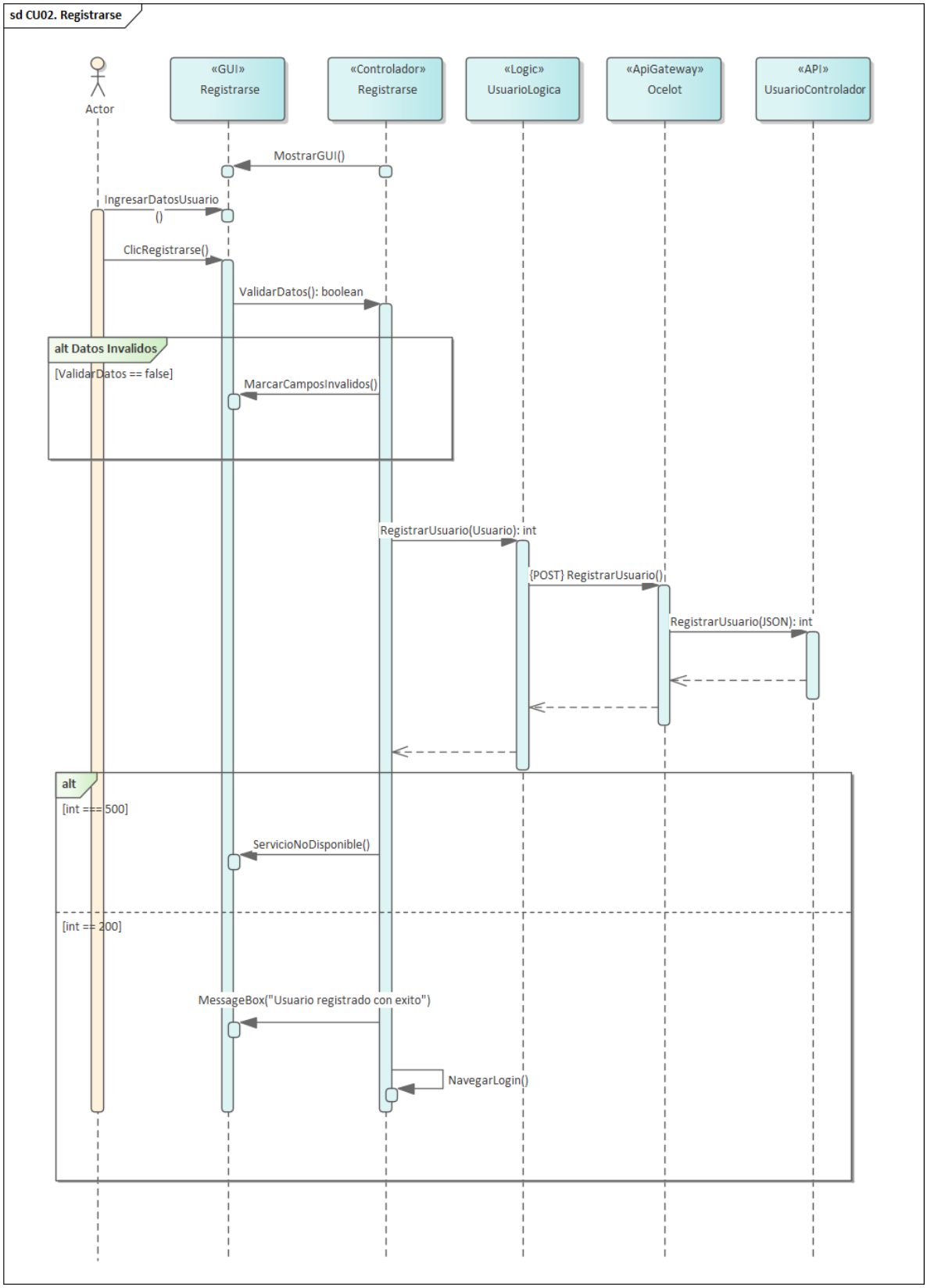


Ilustración 14Diagrama de secuencia CU-02. Registrarse

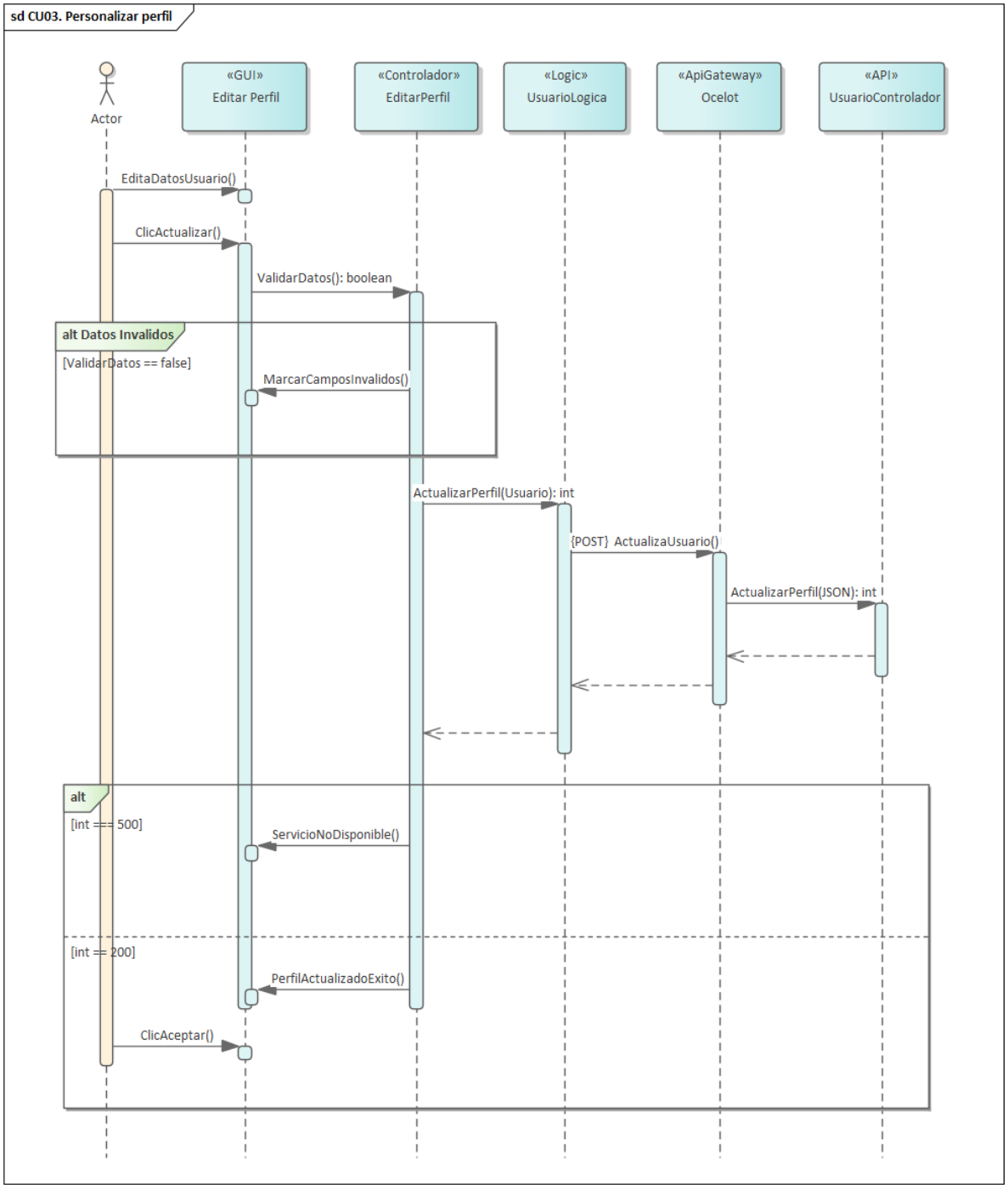


Ilustración 15 Diagrama de secuencia CU-03. Personalizar perfil

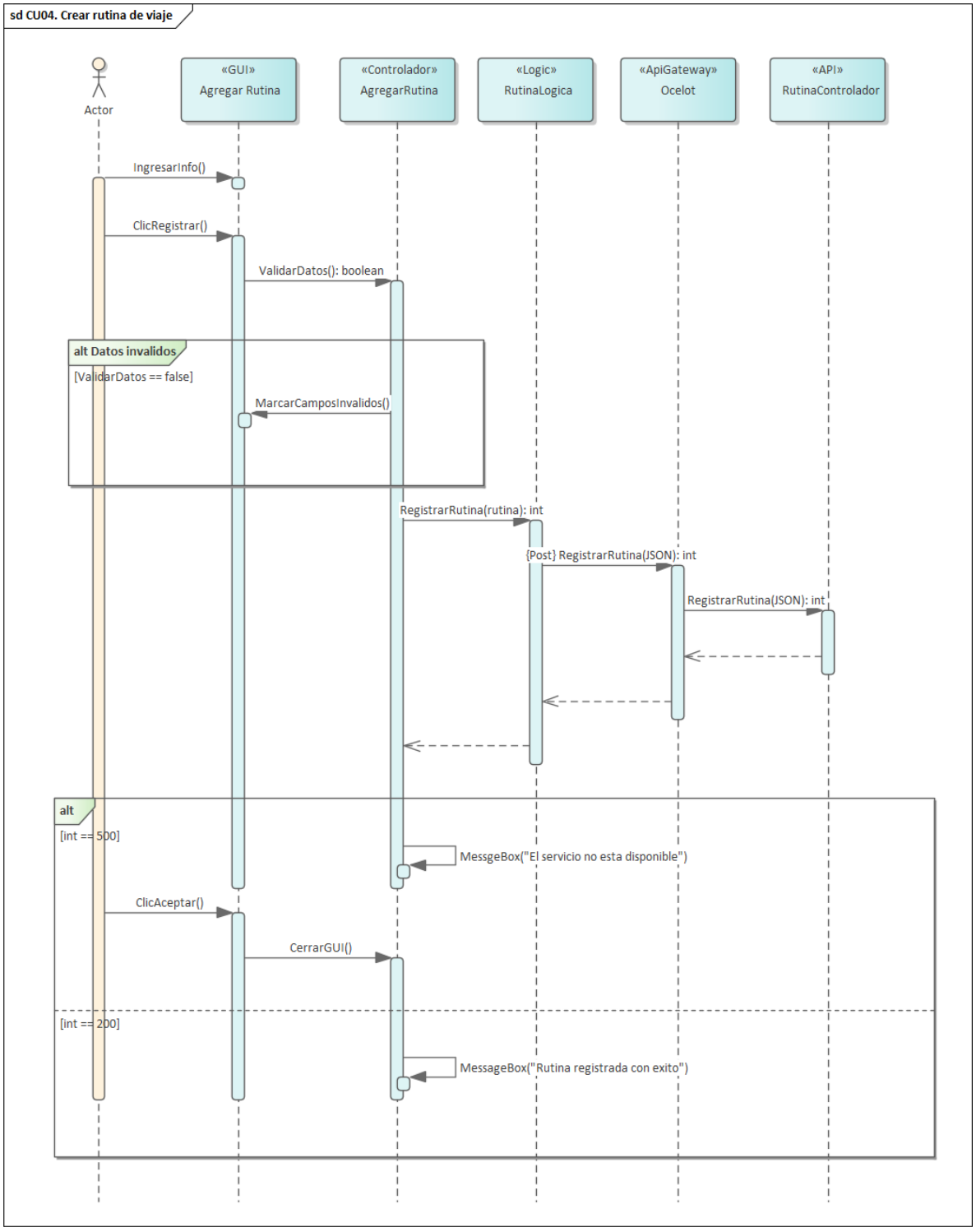


Ilustración 16 Diagrama de secuencia CU-04. Crear rutina de viaje

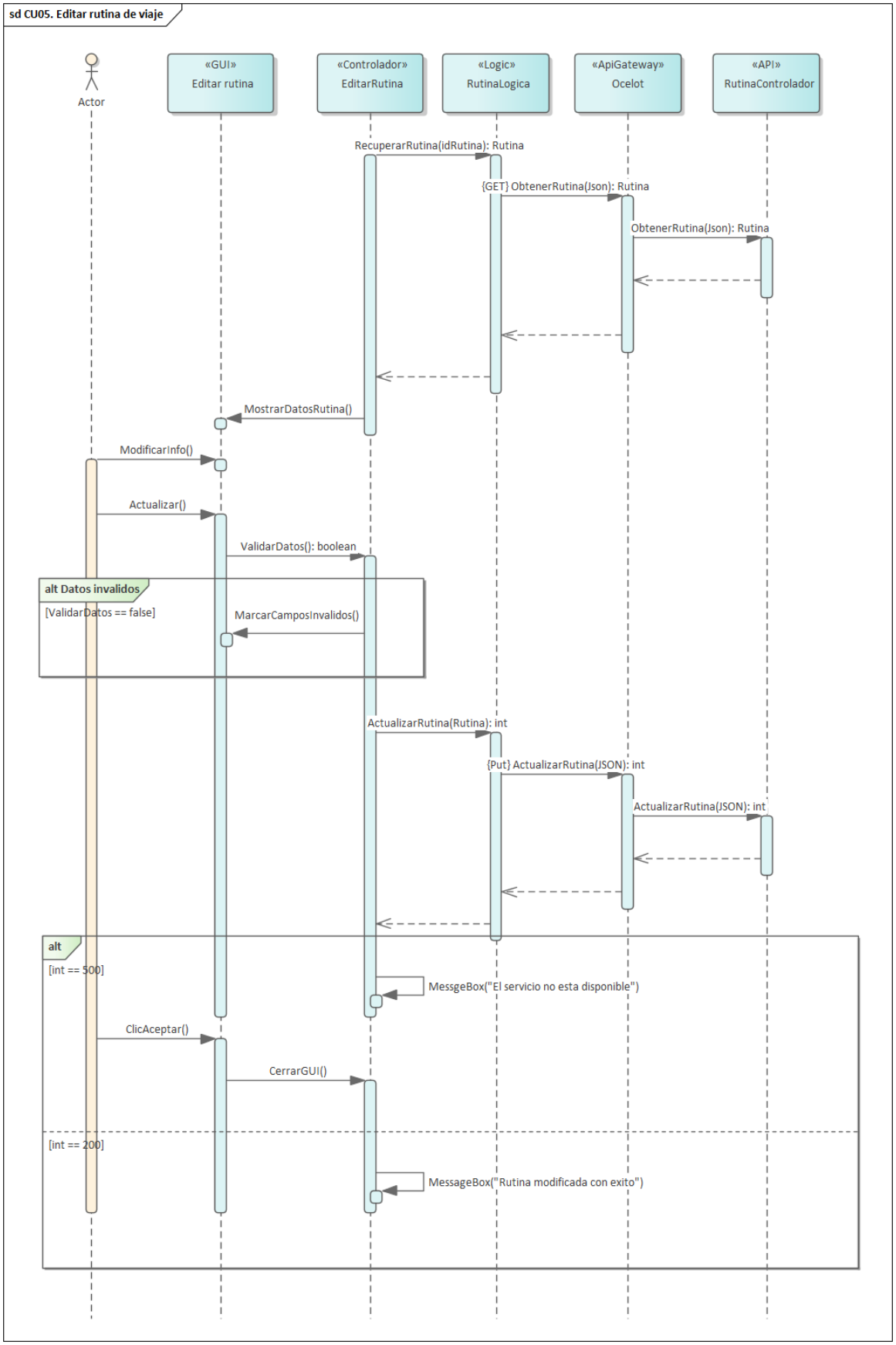

sd CU05. Editar rutina de viaje
Actor
«GUI»
Editar rutina
«Controlador»
EditarRutina
«Logic»
RutinaLogica
«ApiGateway»
Ocelot
«API»
RutinaControlador
RecuperarRutina(idRutina): Rutina
{GET} ObtenerRutina(Json): Rutina
ObtenerRutina(Json): Rutina
MostrarDatosRutina()
ModificarInfo()
Actualizar()
ValidarDatos(): boolean
alt Datos invalidos
[ValidarDatos == false]
MarcarCamposInvalidos()
ActualizarRutina(Rutina): int
{Put} ActualizarRutina(JSON): int
ActualizarRutina(JSON): int
alt
[int == 500]
MessgeBox("El servicio no esta disponible")
ClicAceptar()
CerrarGUI()
[int == 200]
MessageBox("Rutina modificada con exito")

Ilustración 17 Diagrama de secuencia CU-05. Editar Rutina de viaje

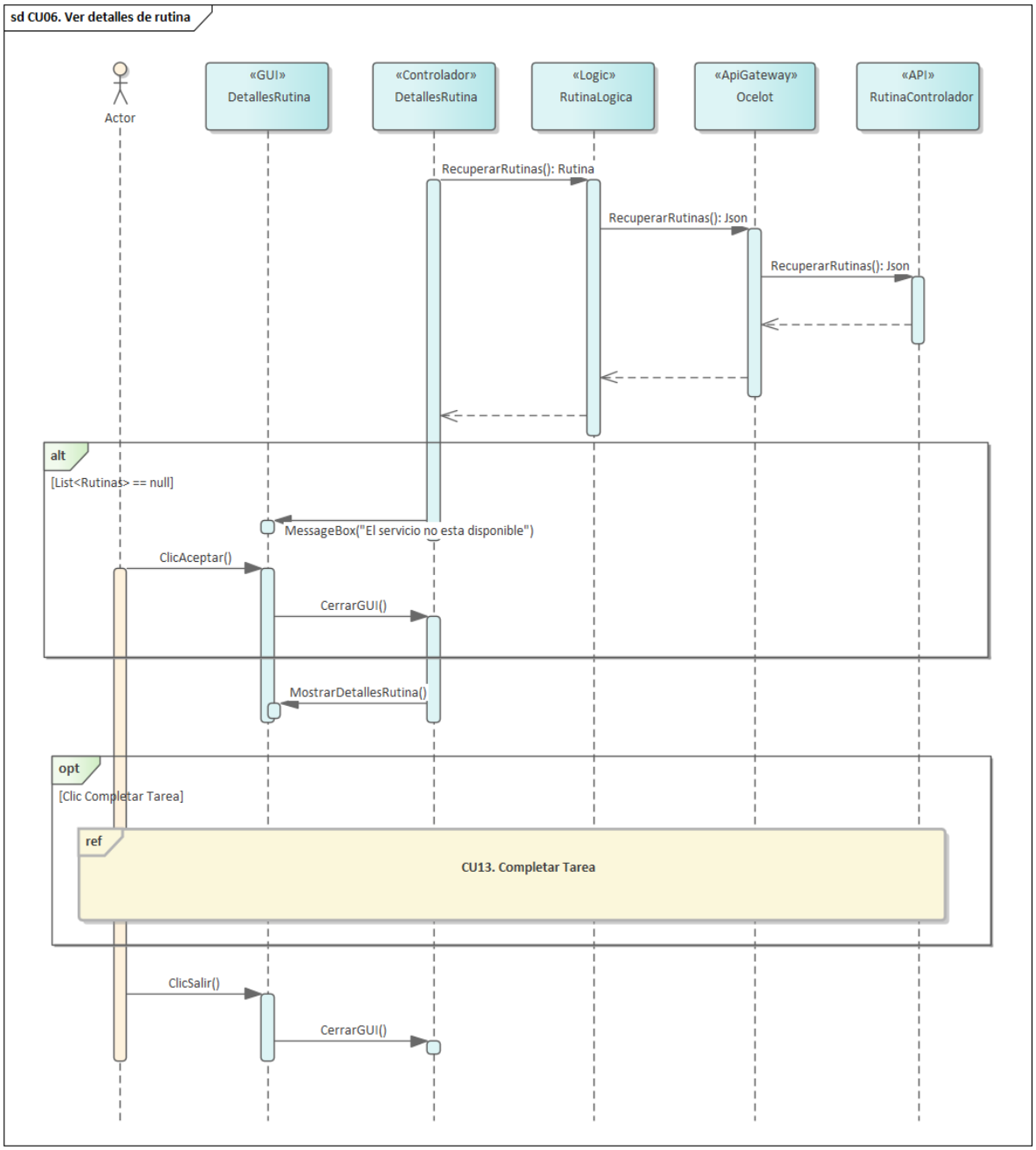


Ilustración 18 Diagrama de Secuancia CU-06. Ver detalles de rutina

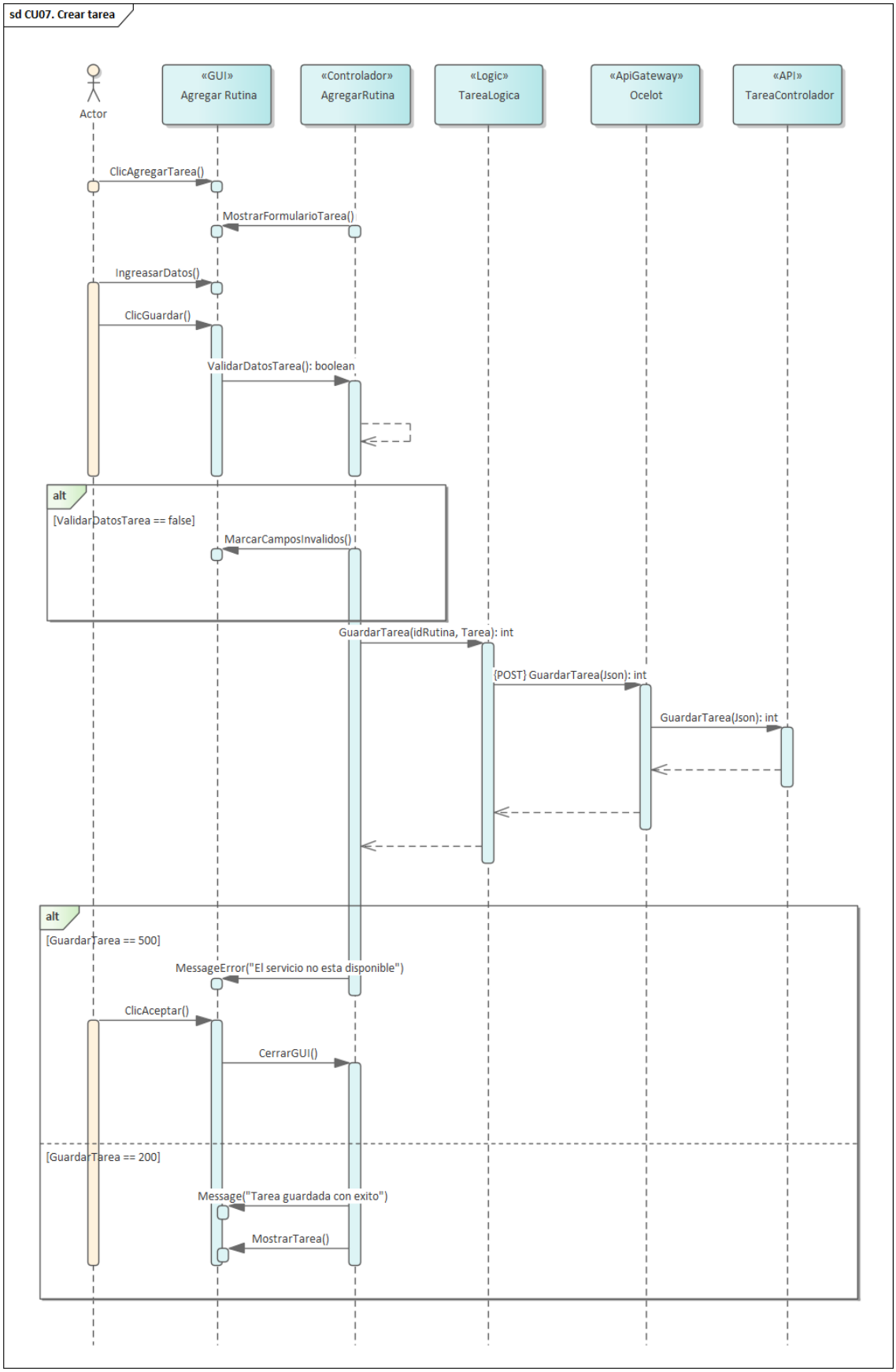

sd CU07. Crear tarea
Actor
«GUI»
Agregar Rutina
«Controlador»
AgregarRutina
«Logic»
TareaLogica
«ApiGateway»
Ocelot
«API»
TareaControlador
ClicAgregarTarea()
MostrarFormularioTarea()
IngreasarDatos()
ClicGuardar()
ValidarDatosTarea(): boolean
alt
[ValidarDatosTarea == false]
MarcarCamposInvalidos()
GuardarTarea(idRutina, Tarea): int
{POST} GuardarTarea(Json): int
GuardarTarea(Json): int
alt
[GuardarTarea == 500]
MessageError("El servicio no esta disponible")
ClicAceptar()
CerrarGUI()
[GuardarTarea == 200]
Message("Tarea guardada con exito")
MostrarTarea()

Ilustración 19 Diagrama de secuencia CU-07. Crear tarea

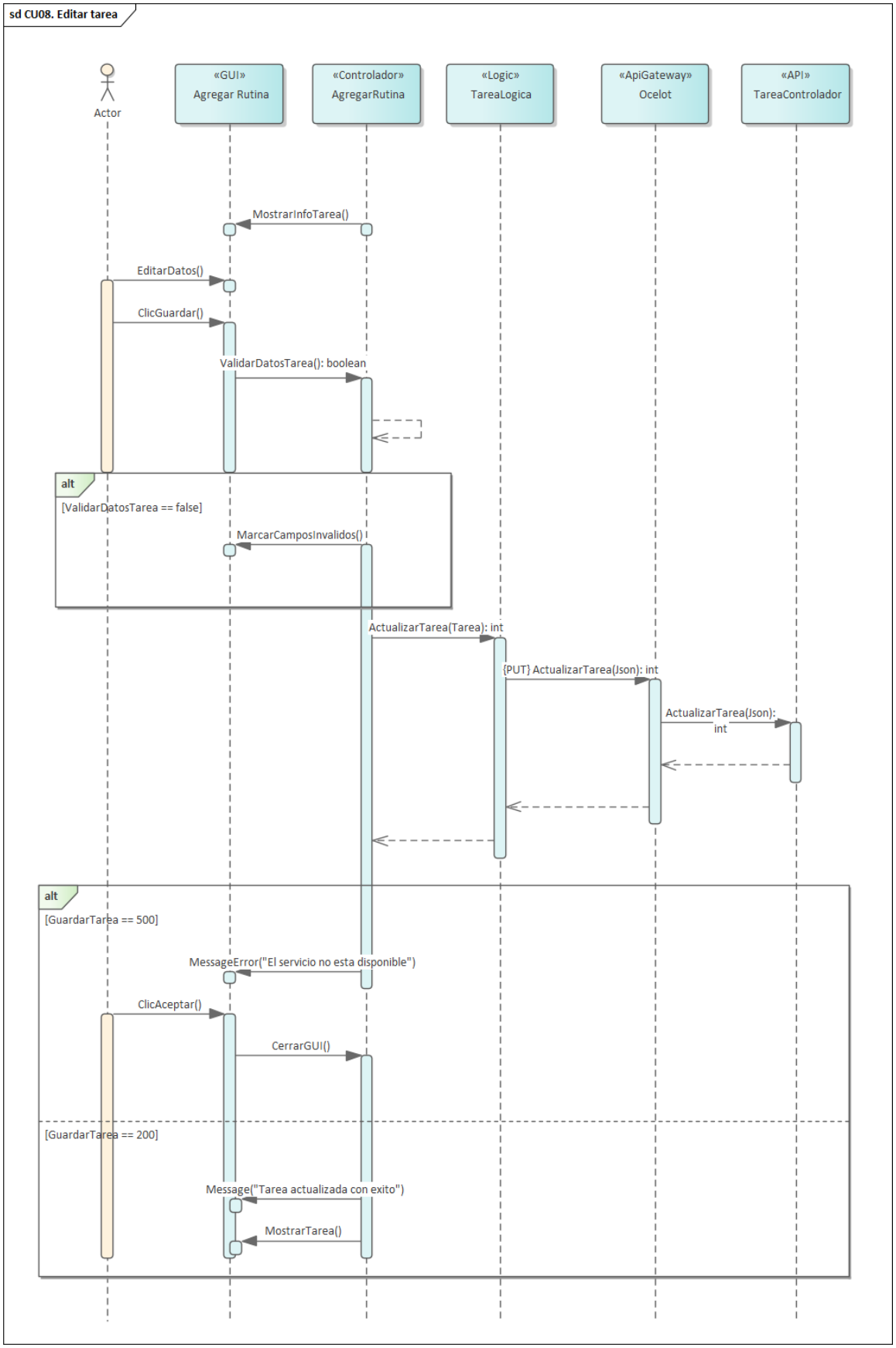

sd CU08. Editar tarea
Actor
«GUI»
Agregar Rutina
«Controlador»
AgregarRutina
«Logic»
TareaLogica
«ApiGateway»
Ocelot
«API»
TareaControlador
MostrarInfoTarea()
EditarDatos()
ClicGuardar()
ValidarDatosTarea(): boolean
alt
[ValidarDatosTarea == false]
MarcarCamposInvalidos()
ActualizarTarea(Tarea): int
{PUT} ActualizarTarea(Json): int
ActualizarTarea(Json): int
alt
[GuardarTarea == 500]
MessageError("El servicio no esta disponible")
ClicAceptar()
CerrarGUI()
[GuardarTarea == 200]
Message("Tarea actualizada con exito")
MostrarTarea()

Ilustración 20 Diagrama de seuencia CU-08. Editar tarea

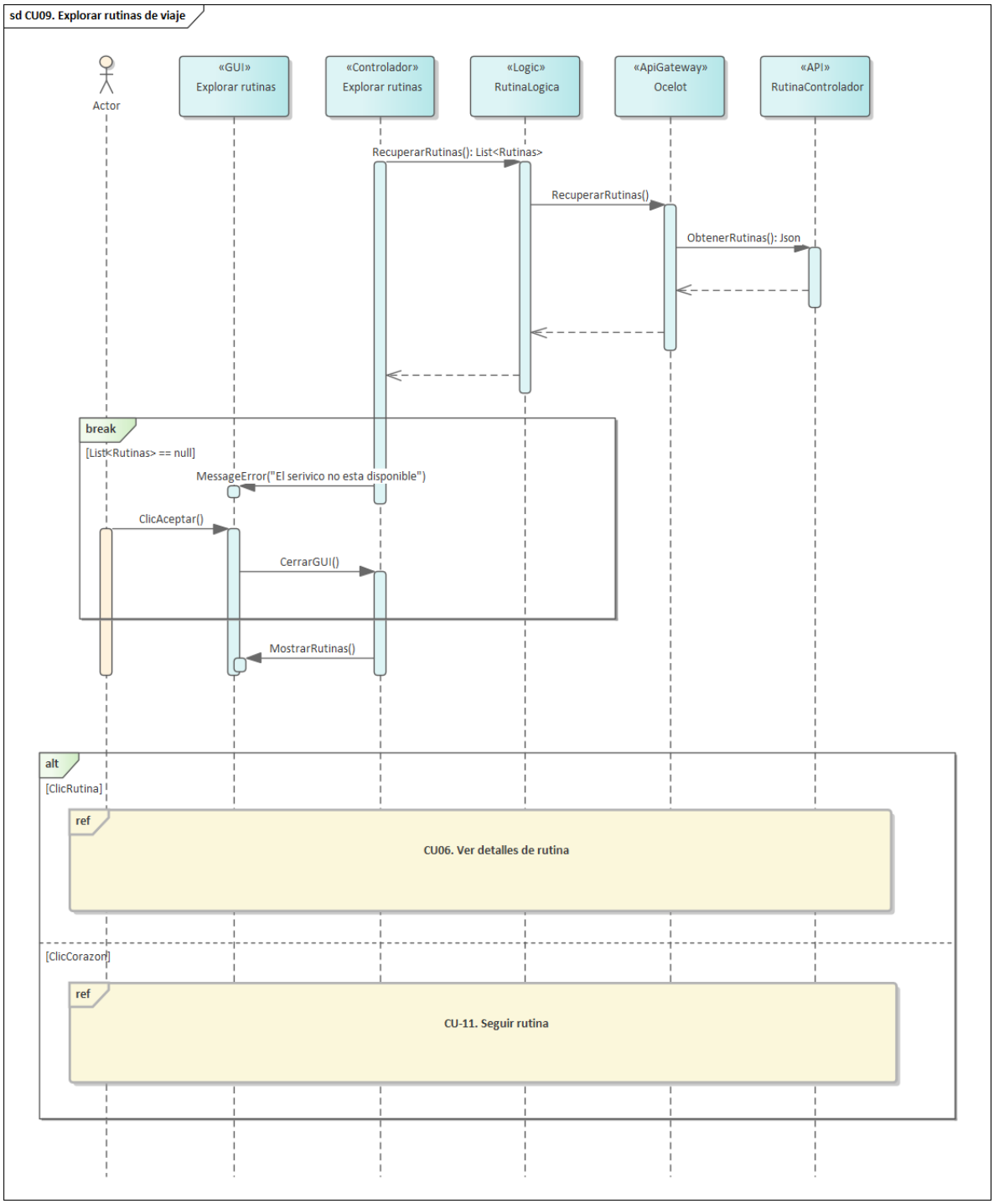


Ilustración 21 Diagrama de secuencia CU-09. Explorar rutinas de viaje

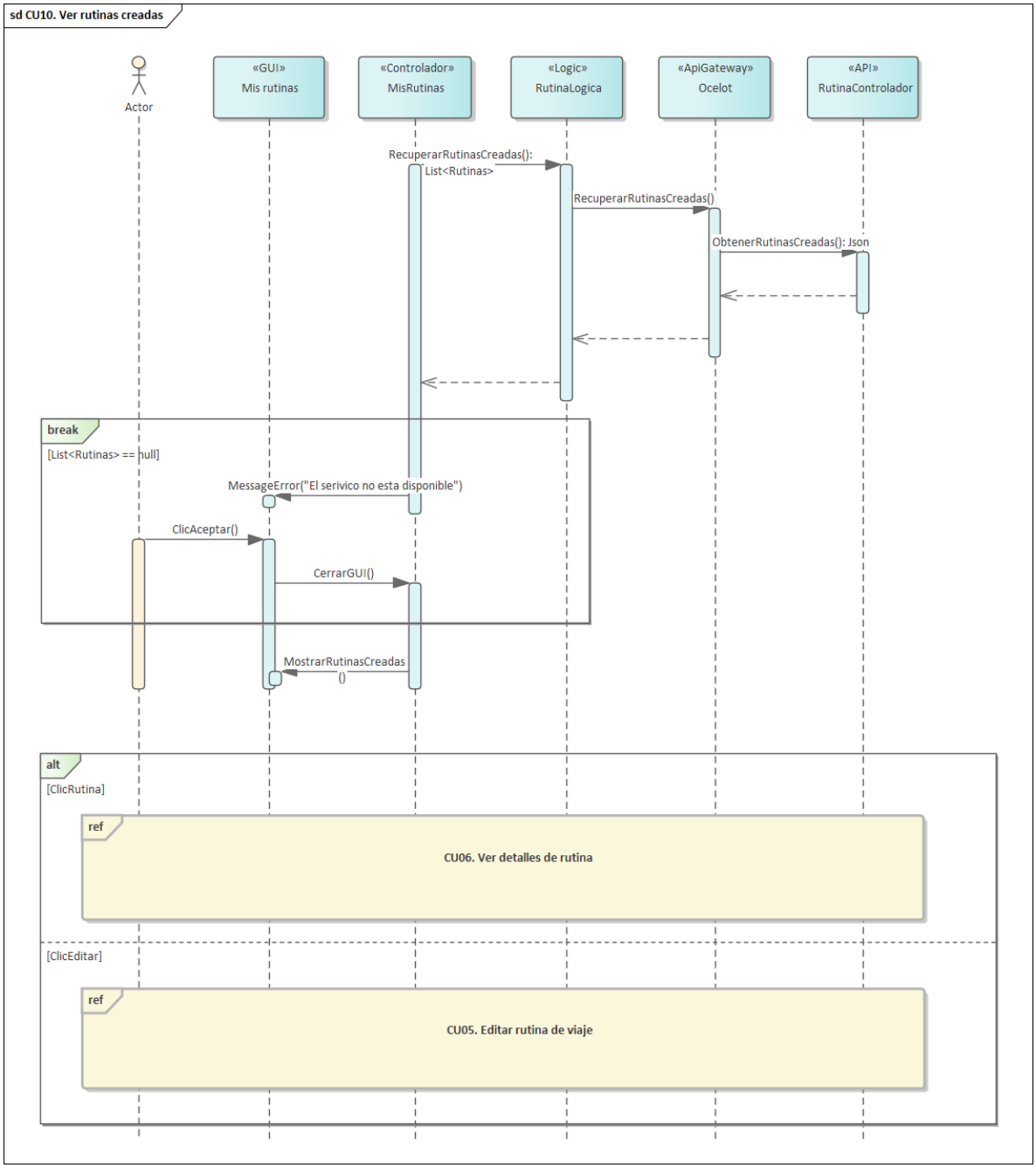


Ilustración 22 diagrama de secuencia CU-10. Ver rutinas creadas

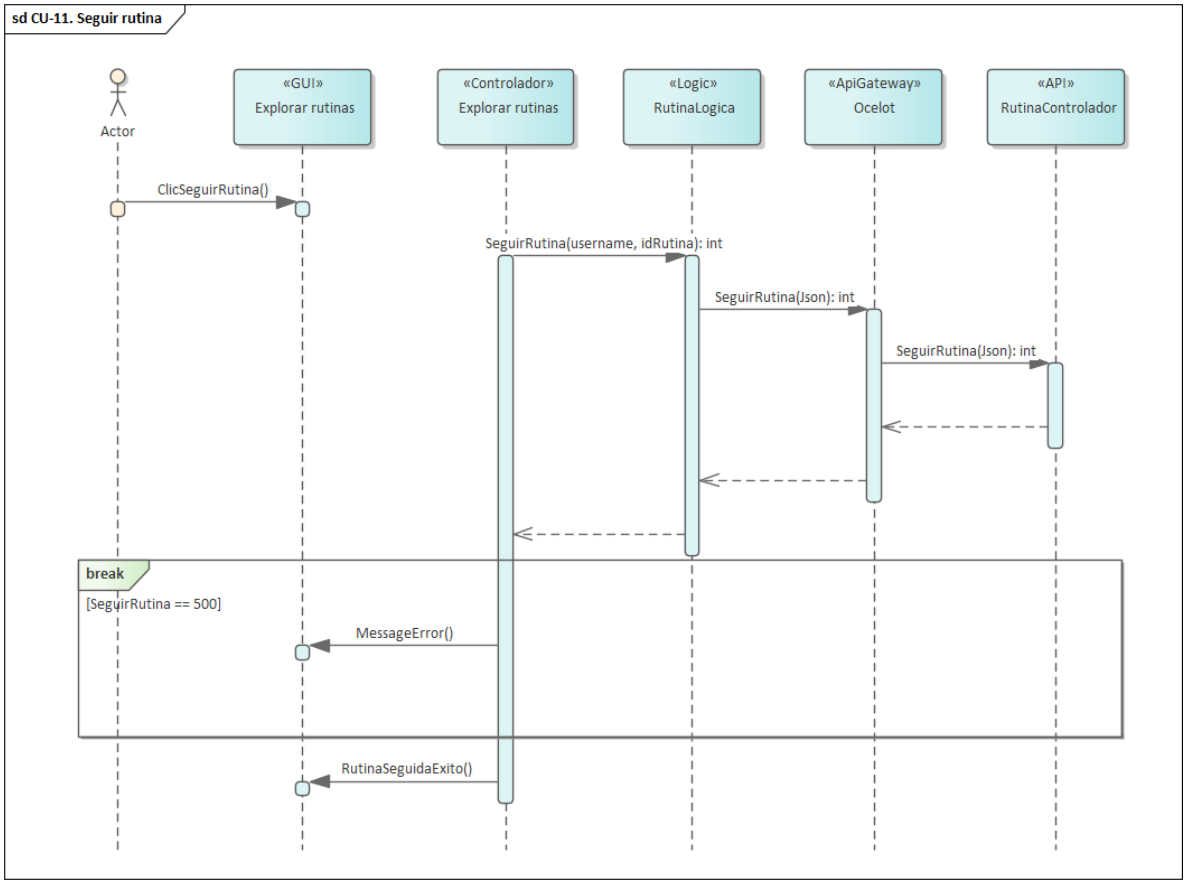


Ilustración 23 Diagrama de secuencia CU-11. Seguir rutina

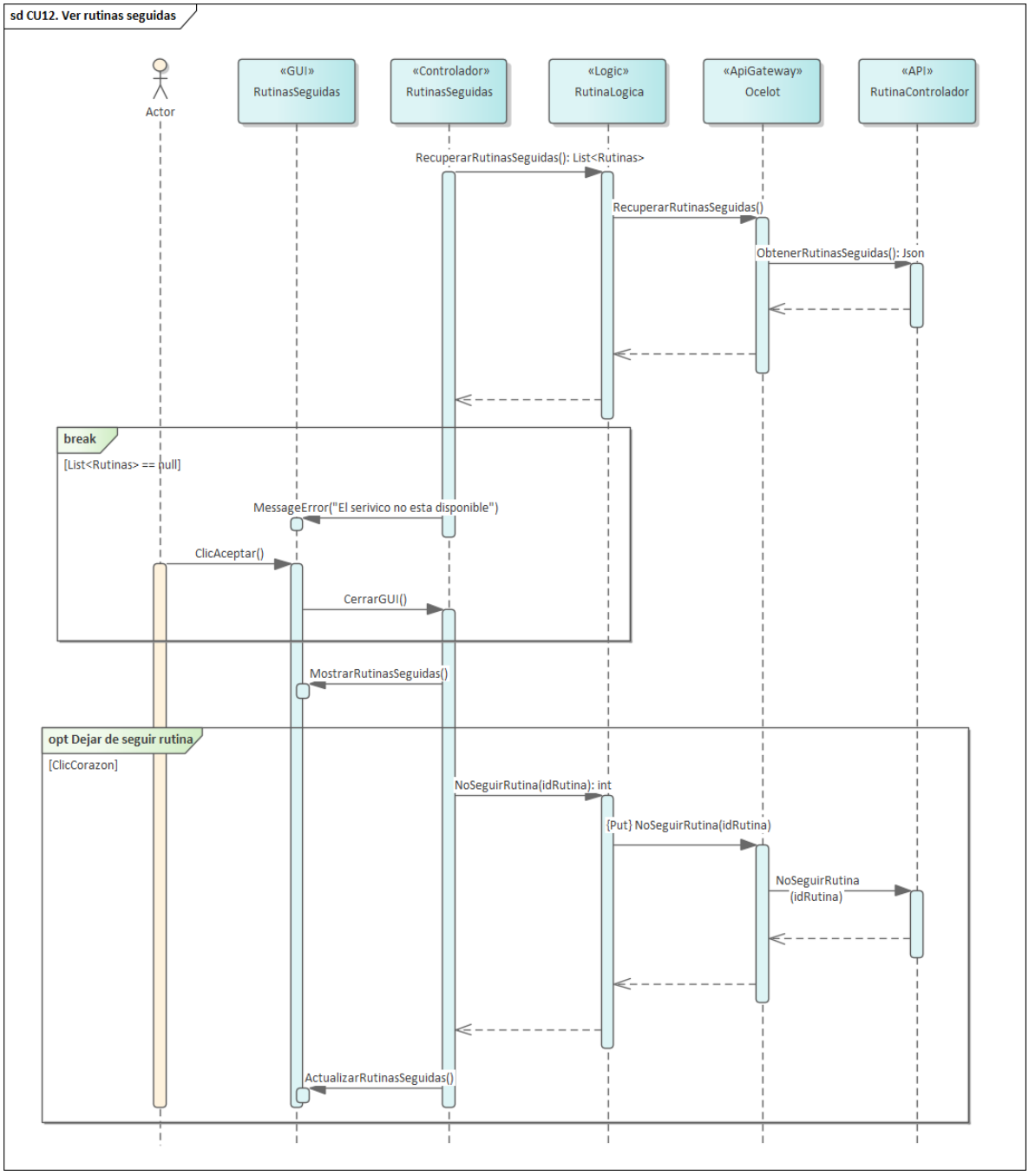


Ilustración 24 Diagrama de secuencia CU-12. Ver rutinas seguidas

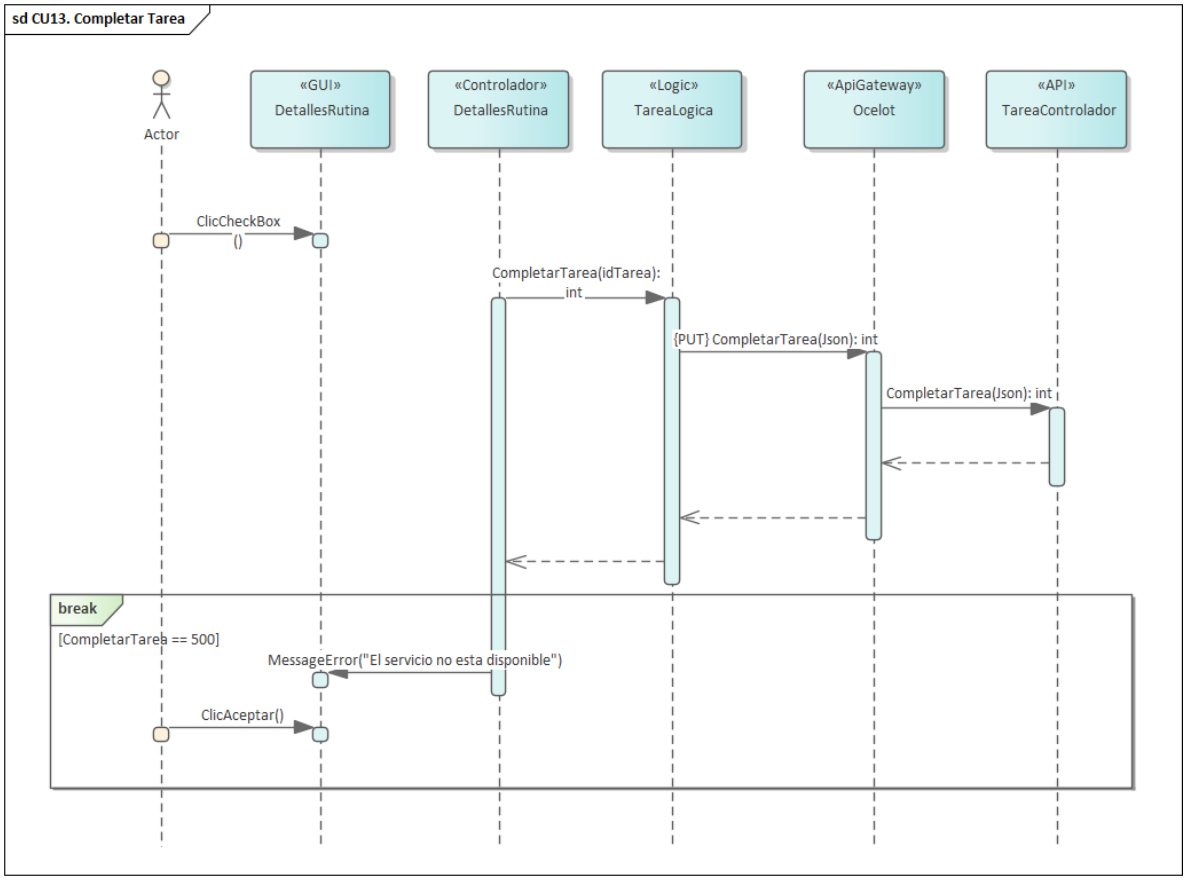


Ilustración 25 Diagrama de sucuencia CU-13. Completar tarea

3.1.5. Vista de despliegue

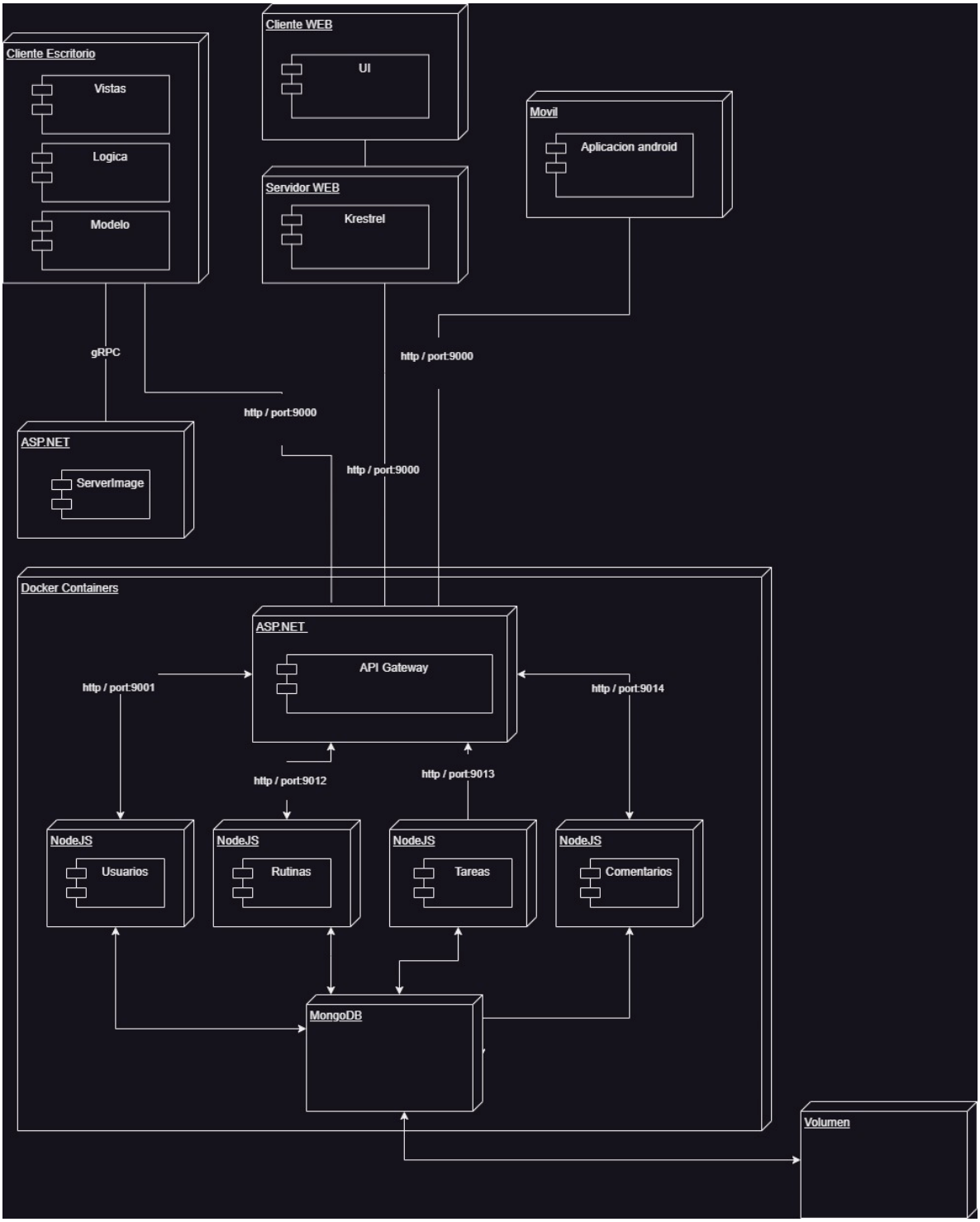
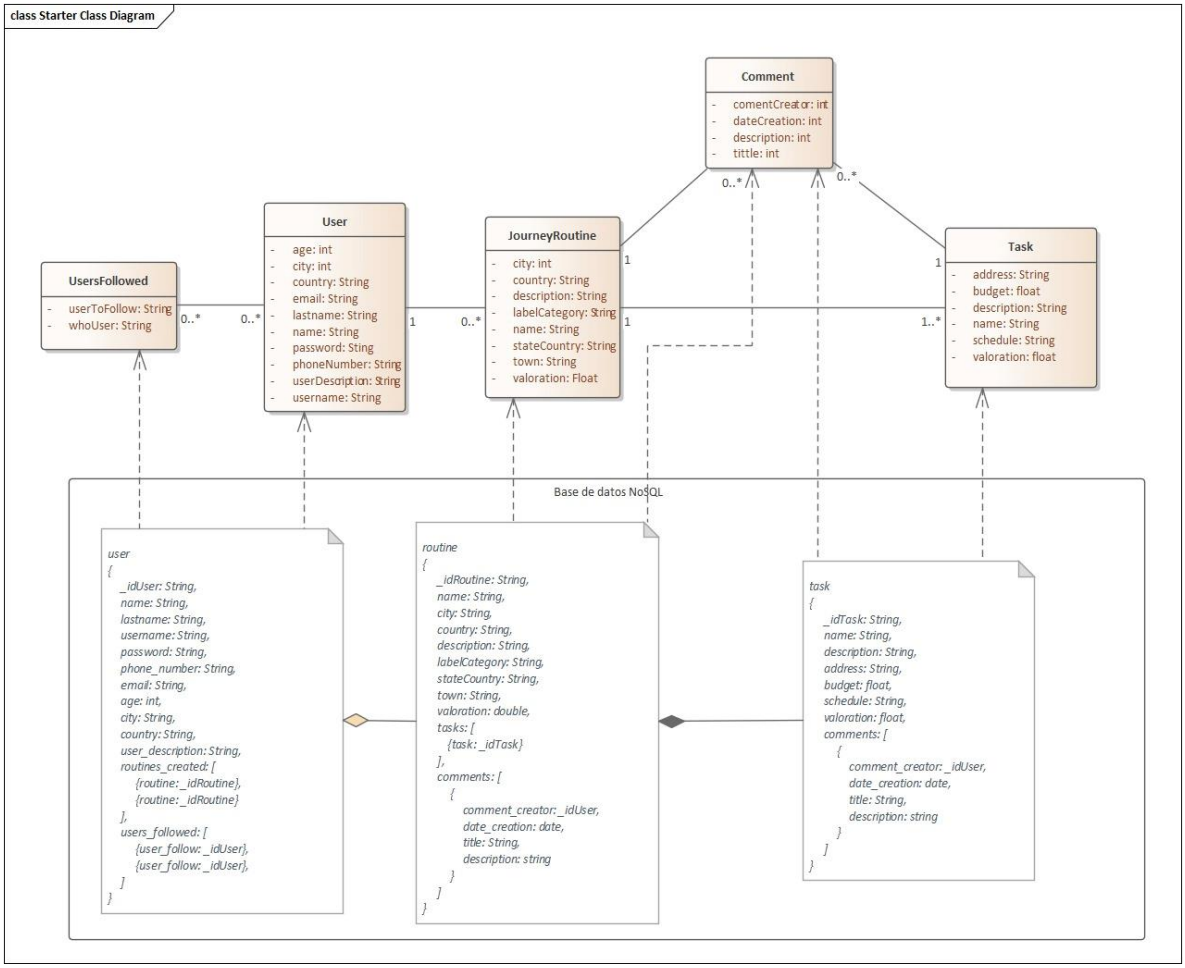


Ilustración 26 Diagrama de despliegue Journeymate

3.2. Modelo de datos

Al utilizar una base de datos NoSQL (MongoDB) en lugar del diagrama E-R hicimos este para hacer la representación de la base de datos



3.3. Descripciones de casos de uso

ID	CU-01
Nombre	Iniciar sesión
Descripción	Permite al actor iniciar sesión en el sistema
Actor(es)	Usuario, Administrador
Precondición(es)	PRE01. El actor se encuentra registrado en el sistema
Disparador	El actor da clic en el botón “Iniciar Sesión”
Flujo normal	1. El sistema muestra la ventana de inicio de sesión 2. El actor ingresa los datos de inicio de sesión (correo y contraseña) y da clic en el botón “Iniciar Sesión” 3. El sistema verifica que los datos ingresados se encuentren registrados en el sistema y muestra el menú “Explorar”. (FA01) (EX01) Termina el caso de uso
Flujo alterno	FA01. Datos no encontrados 1. El sistema muestra el mensaje “Usuario o contraseña no validos”. 2. El actor da clic en el botón “Aceptar”

	Regresa al paso 2 del flujo normal
Excepciones	EX01. Sistema no disponible 1. El sistema muestra el mensaje “El sistema no está disponible por el momento, inténtelo más tarde” 2. El actor da clic en el botón “Aceptar” Termina el caso de uso
Postcondición(es)	PC01. El actor ahora tiene una sesión activa

ID	CU-02
Nombre	Registrarse
Descripción	Permite al actor registrarse en el sistema
Actor(es)	Usuario
Precondición(es)	PRE01. El actor NO se encuentra registrado en el sistema
Disparador	El actor da clic en el botón “Registrarse”
Flujo normal	1. El sistema muestra la ventana “Registro de usuario” 2. El actor ingresa los datos de registro (Nombre, Apellidos correo electrónico, edad y contraseña) y da clic en “Registrar” (FA03) 3. El sistema valida que los datos ingresados sean válidos y de ser así redirige al usuario a la pantalla inicio de sesión (FA01) 4. Termina caso de uso
Flujo alterno	FA01. Datos inválidos 1. El sistema muestra un mensaje de información inválidos 2. Regresa al paso 2 del flujo normal
Excepciones	EX01. Sistema no disponible 1. El sistema muestra el mensaje “El sistema no está disponible por el momento, inténtelo más tarde” 2. El actor da clic en el botón “Aceptar” Termina el caso de uso
Postcondición(es)	PC01. Se registra al usuario en el sistema

ID	CU-03
Nombre	Personalizar perfil
Descripción	Permite al actor editar la información de su perfil de usuario
Actor(es)	Usuario, Administrador
Precondición(es)	PRE01. El actor tiene se encuentra registrado en el sistema PRE01. El actor tiene una sesión activa en el sistema
Disparador	El actor da clic en el botón “Editar Perfil”
Flujo normal	1. El sistema muestra la GUI Editar perfil

	2. El actor modifica los campos del su perfil y da clic en el botón “Actualizar” (FA02) 3. El sistema valida que los campos ingresados sean válidos, guarda los datos actualizados en la base de datos y muestra el mensaje “Perfil actualizado con éxito” (FA01) (EX01) 4. El actor da clic en el botón “Aceptar” 5. El sistema cierra la GUI 6. Termina el caso de uso
Flujo alterno	FA01. Campos inválidos 1. El sistema resalta los campos de información que sean inválidos 2. Regresa al paso 2 del flujo normal FA02. Cancelar personalización 1. El sistema muestra el mensaje “¿Desea cancelar la personalización?” 2. El actor da clic en el botón “Aceptar” 3. El sistema cierra la GUI 4. Termina el caso de uso
Excepciones	EX01. Servicio no disponible 1. El sistema muestra el mensaje “El sistema no está disponible, inténtelo más tarde” 2. El actor da clic el botón “Aceptar” 3. Termina el caso de uso
Postcondición(es)	PC01. Se actualiza la información del usuario

ID	CU-04
Nombre	Crear rutina de viaje
Descripción	Permite al actor crear una nueva rutina
Actor(es)	Usuario, Administrador
Precondición(es)	PRE01. El actor tiene una sesión activa en el sistema
Disparador	El actor da clic en el botón “+”
Flujo normal	1. El sistema muestra la GUI Agregar rutina 2. El actor llena los campos de información necesaria, y de forma opcional agrega tareas a la rutina y da clic en el boton “Registrar” (FA02) 3. El sistema verifica que los datos ingresados sean validos, guarda la rutina y las tareas en la base de datos y muestra el mensaje “Rutina registrada con éxito” (FA01) (EX01) 4. El actor da clic en el boton “Aceptar” 5. Termina el caso de uso
Flujo alterno	FA01. Campos inválidos 1. El sistema resalta los campos de información que sean inválidos 2. Regresa al paso 2 del flujo normal

	FA02. Cancelar personalización 1. El sistema muestra el mensaje "¿Desea cancelar el registro de rutina?" 2. El actor da clic en el botón "Aceptar" 3. El sistema cierra la GUI 4. Termina el caso de uso
Excepciones	EX01. Servicio no disponible 1. El sistema muestra el mensaje "El sistema no está disponible, inténtelo más tarde" 2. El actor da clic el botón "Aceptar" 3. Termina el caso de uso
Postcondición(es)	PC01. Se registra una nueva rutina en el sistema

ID	CU-05 (Extiende de CU-10)
Nombre	Editar rutina de viaje
Descripción	Permite al actor editar una rutina de viaje
Actor(es)	Usuario, Administrador
Precondición(es)	PRE01. El actor cuenta con una sesión activa en el sistema PRE02. La rutina existe en el sistema
Disparador	El actor da clic en el boton "Editar rutina"
Flujo normal	1. El sistema muestra la GUI Editar rutina 2. El actor edita los campos necesarios y da clic en el boton "Actualizar" (FA02) 3. El sistema verifica que los campos modificados sean válidos, actualiza la información en la base de datos y muestra el mensaje "Rutina actualizada con éxito" (FA01) (EX01) 4. El actor da clic en el boton "Aceptar" 5. Termina el caso de uso
Flujo alterno	FA01. Campos inválidos 1. El sistema resalta los campos de información que sean inválidos 2. Regresa al paso 2 del flujo normal FA02. Cancelar personalización 1. El sistema muestra el mensaje "¿Desea cancelar el registro de rutina?" 2. El actor da clic en el botón "Aceptar" 3. El sistema cierra la GUI 4. Termina el caso de uso
Excepciones	EX01. Servicio no disponible 1. El sistema muestra el mensaje "El sistema no está disponible, inténtelo más tarde" 2. El actor da clic el botón "Aceptar" 3. Termina el caso de uso
Postcondición(es)	PC01. Se actualizan los datos de la rutina

ID	CU-06 (Extendido de CU-06, CU-10)
Nombre	Ver detalles de rutina
Descripción	Permite al actor ver los detalles de una rutina seleccionada
Actor(es)	Usuario, administrador
Precondición(es)	PRE01. La rutina debe existir en el sistema
Disparador	El actor da clic a una rutina
Flujo normal	1. El sistema recupera de la base de datos la rutina seleccionada y sus tareas asociadas y muestra su información en la GUI Detalles rutina (EX01) 2. El actor da clic en el boton “Salir” 3. El sistema cierra la GUI 4. Termina el caso de uso
Flujo alterno	FA01. Completar tarea 1. El actor da clic en el ComboBox de la tarea que quiera completar 2. Extiende al caso de uso CU-13. Completar tarea
Excepciones	EX01. Servicio no disponible 1. El sistema muestra el mensaje “El sistema no está disponible, inténtelo más tarde” 2. El actor da clic el botón “Aceptar” 3. Termina el caso de uso
Postcondición(es)	Ninguna

ID	CU-07 (Extiende de CU-04)
Nombre	Crear tarea
Descripción	Permite al usuario crear una tarea dentro de una rutina de viaje
Actor(es)	Usuario, Administrador
Precondición(es)	PRE01. El usuario cuenta con una sesión activa PRE02. La rutina de viaje a la que se agregara la tarea debe existir en el sistema
Disparador	El usuario da clic en el “Agregar tarea” en una rutina de viaje
Flujo normal	1. El sistema muestra una ventana con el formulario para añadir una tarea a una rutina de viaje 2. El Actor llena los campos de información (nombre, lugar, presupuesto, descripción) y el clic en el botón “Guardar” (FA03) 3. El sistema verifica que los datos ingresados sean válidos, guarda la tarea en la base de datos y muestra el mensaje y muestra la nueva tarea en la lista de tareas ubicada debajo de la rutina de viaje (FA01) (FA02) 4. Termina el caso de uso

Flujo alterno	FA01. Datos inválidos 1. El sistema marca en rojo los campos de información inválidos 2. Regresa al paso 2 del flujo normal FA03. Cancelar creación de tarea 1. El actor da clic en el botón “Ocultar” 2. Termina el caso de uso
Excepciones	EX01. Sistema no disponible 1. El sistema muestra el mensaje “El servició no está disponible por el momento, inténtelo más tarde” 2. El actor da clic en el botón “Aceptar” 3. Termina el caso de uso
Postcondición(es)	PC01. Se agrega una tarea nueva a una rutina de viaje

ID	CU-08 (Extiende de CU-04)
Nombre	Editar tarea
Descripción	Permite al usuario editar la información de una tarea
Actor(es)	Usuario, Administrador
Precondición(es)	PRE01. El usuario cuenta con una sesión activa PRE02. La tarea está registrada en el sistema
Disparador	El actor da clic en el botón “Editar” de una tarea registrada
Flujo normal	1. El sistema muestra una ventana para editar la información de la tarea 2. El actor modifica los campos de información y da clic en el botón “Guardar” 3. El sistema verifica que la información ingresada sea válida, actualiza la tarea en la base de datos y muestra los cambios en la lista de tareas de la rutina (FA01) (FA02) (EX01) 4. Termina el caso de uso
Flujo alterno	FA01. Datos inválidos 1. El sistema marca en rojo los campos de información inválidos 2. Regresa al paso 2 del flujo normal FA03. Cancelar edición de tarea 1. El actor da clic en el botón “Cancelar” 2. El sistema muestra el mensaje “¿Desea cancelar la edición de la tarea?

	3. El actor da clic en el botón “Aceptar” Termina el caso de uso
Excepciones	EX01. Sistema no disponible 1. El sistema muestra el mensaje “El sistema no está disponible por el momento, inténtelo más tarde” 2. El actor da clic en el botón “Aceptar” Termina el caso de uso
Postcondición(es)	PC01. La tarea ha sido actualizada.

ID	CU-09
Nombre	Explorar rutinas de viaje
Descripción	Permite al actor explorar las rutinas públicas del sistema
Actor(es)	Usuario, Administrador
Precondición(es)	Ninguna
Disparador	El usuario inicia el sistema
Flujo normal	1. El sistema recupera las rutinas registradas en el sistema que tengan su visibilidad en publica (en caso de que el usuario sea Administrador recupera las rutinas publicas y privadas) y las muestra en la en GUI Explorar. (EX01)
Flujo alterno	FA01. Ver detalles de rutina 1. El actor da clic en una de las rutinas 2. Extiende al caso de uso CU-06. Ver detalles de rutina FA02. Seguir rutina 1. El actor da clic en el “Corazón de la rutina” 2. Extiende al caso de uso CU-11. Seguir rutina
Excepciones	EX01. Servicio no disponible 1. El sistema muestra el mensaje “El sistema no está disponible, inténtelo más tarde” 2. El actor da clic el botón “Aceptar” 3. Termina el caso de uso
Postcondición(es)	Ninguna

ID	CU-10
Nombre	Ver rutinas creadas
Descripción	Permite al actor ver las rutinas que ha registrado
Actor(es)	Usuario, Administrados
Precondición(es)	PRE01. El actor cuenta con una sesión activa en el sistema
Disparador	El actor da clic en el boton “Mis Rutinas”

Flujo normal	1. El sistema recupera de la base de datos las rutinas que ha registrado, sus tareas asociadas y las muestra en la GUI Rutinas seguidas (EX01)
Flujo alterno	FA01. Ver detalles de rutina 1. El actor da clic en una rutina 2. Extiende al caso de uso CU-06. Ver detalles de rutina FA02. Editar rutina 1. El actor da clic en el boton “Editar rutina” 2. Extiende al caso de uso CU-05. Editar rutina de viaje
Excepciones	EX01. Servicio no disponible 1. El sistema muestra el mensaje “El sistema no está disponible, inténtelo más tarde” 2. El actor da clic el botón “Aceptar” 3. Termina el caso de uso
Postcondición(es)	Ninguna

ID	CU-11 (Extiende de CU-09)
Nombre	Seguir rutina
Descripción	Permite seguir una rutina publica
Actor(es)	Usuario, Administrador
Precondición(es)	PRE01. El usuario cuenta con una sesión activa en el sistema
Disparador	El actor da clic en el “Corazón” de una rutina
Flujo normal	1. El sistema recupera los datos de la rutina seleccionada y la copia en las rutinas seguidas del actor (EX01)
Flujo alterno	Ninguno
Excepciones	EX01. Servicio no disponible 1. El sistema muestra el mensaje “El sistema no está disponible, inténtelo más tarde” 2. El actor da clic el botón “Aceptar” 3. Termina el caso de uso
Postcondición(es)	PC01. Se agrega la rutina a rutinas seguidas del actor

ID	CU-12
Nombre	Ver rutinas seguidas
Descripción	Permite al actor ver las rutinas que el actor ha seguido
Actor(es)	Usuario, Administrador
Precondición(es)	PRE01. El actor cuenta con una sesión activa en el sistema
Disparador	El actor da clic en el boton “Rutinas Favoritas”
Flujo normal	1. El sistema recupera las rutinas seguidas del actor y sus tareas asociadas y las muestra en la GUI Rutinas favoritas

Flujo alterno	FA01. Dejar de seguir rutina 1. El actor da clic en el “Corazón” de una rutina 2. El sistema borra la rutina de las rutinas seguidas del actor y actualiza la GUI 3. Termina el caso de uso
Excepciones	EX01. Servicio no disponible 1. El sistema muestra el mensaje “El sistema no está disponible, inténtelo más tarde” 2. El actor da clic el botón “Aceptar” 3. Termina el caso de uso
Postcondición(es)	PC01. Se quita de rutinas seguidas del usuario la rutina seleccionada

ID	CU-13 (Extiende de CU-06)
Nombre	Completar tarea
Descripción	El actor desea completar una tarea la cual registro en alguna de sus rutinas o de la lista de sus me gusta.
Actor(es)	Usuario, Administrador
Precondición(es)	PRE01. El actor cuenta con una sesión activa en el sistema
Disparador	El Actor da clic en la casilla vacía ubicada en el lado izquierdo de la tarea.
Flujo normal	1. El sistema marca la casilla de la tarea y guarda en el sistema el cambio realizado. 2. Fin de caso de uso.
Flujo alterno	1. FA01-Editar tarea 1.1 El Actor da clic en el botón “Editar” de algúna de las tareas de la rutina. 1.2 Continua en el CU-08 Editar tarea.
Excepciones	EX01. Servicio no disponible 1. El sistema muestra el mensaje “El sistema no está disponible, inténtelo más tarde” 2. El actor da clic el botón “Aceptar” Termina el caso de uso
Postcondición(es)	PC01-El estado de la tarea se actualizo en la base de datos.

4. Construcción

Links a repositorio remoto

- API

AxelUtrera/JourneymateAPI (github.com)

- Aplicación escritorio

DPaxtian/Journeymate-Desktop (github.com)

- gRPC

DPaxtian/JourneymateImageServer (github.com)

4.1. Selección justificada de la pila tecnológica.

4.1.1. Aplicación de cliente rico.

Para este proyecto se desarrollaron 3 clientes (escritorio, web y móvil), cada uno utilizando diferentes tecnologías, seleccionadas de acuerdo con las necesidades obtenidas a continuación, se detallada cada una.

- Cliente de escritorio: Se eligió trabajar con WPF (Windows Presentation Foundation), esta tecnología ofrece herramientas muy útiles para desarrollar aplicaciones como la creación de interfaces de usuario utilizando XAML, además esta tecnología hace muy fácil trabajar con el patrón de diseño MVVC y además al ser una tecnología nativa de Windows asegura la compatibilidad y el alto rendimiento al trabajar con el sistema operativo.
- Cliente WEB: se eligió trabajar con ASP .Net Core, con el framework Razor Pages, se eligió esta tecnología porque permite crear aplicaciones web de manera rápida y eficiente. Elegimos esta tecnología porque utilizar el lenguaje C# con el que estamos familiarizados bastante, además al utilizar el lenguaje de marcado Razor hace que el desarrollo de las interfaces con HTML sea más sencillo.
- Cliente Movil: se eligió trabajar con Android utilizando Kotlin como lenguaje y Android Studio como IDE, utilizar Android para desarrollar una aplicación por un lado nos ofreció compatibilidad con la mayoría de los teléfonos en el mercado, y por otro lado nos ofreció la eficiencia de Kotlin ya que ofrece una sintaxis sencilla.

4.1.2. Aplicación de servicios.

Por el lado del backend para este proyecto se han desarrollado varias API que corresponden a cada módulo de funcionalidad de la aplicación (usuarios, rutinas, tareas y comentarios) se decidió separar cada funcionalidad en módulos independientes para asegurar la disponibilidad de la aplicación en caso de que de alguna funcionalidad falle, también para facilitar el mantenimiento y la escalabilidad a futuro.

También se ha desarrollado un API Gateway para facilitar la comunicación entre los clientes y las funcionalidades desarrolladas en las API. Adicional a esto se desarrollo un servidor de imágenes para guardar multimedia en la aplicación.

API: se decidió usar la tecnología NodeJS, esta tecnología nos permite trabajar con el lenguaje JavaScript que, aunque no conocíamos su facilidad de uso nos permitió adaptarnos rápidamente y nos permitió trabajar eficazmente, además de esto se eligió trabajar con esta

tecnología porque permite manejar solicitudes sin bloquear la ejecución principal haciendo así que sea mas sencillo desarrollar las funcionalidades.

API Gateway: para el Api Gateway se eligió trabajar con Ocelot y ASP.NET, principalmente porque tiene un enfoque especializado a trabajar con arquitecturas de microservicios haciendo que el consumo de los distintos servicios sea muy sencillo, otra razón por la que se eligió esta tecnología es porque es bastante fácil de usar y configurar el middleware toma poco tiempo.

Servidor de Imágenes: para desarrollar el servidor encargado de almacenar las imágenes se eligió trabajar con gRPC y el lenguaje C#, con ASP .Net Core, gRPC permite una comunicación eficiente y de baja latencia entre clientes y servidores, lo que es especialmente beneficioso en entornos distribuidos y de microservicios. Una de las razones principales por la que elegimos utilizar esta tecnología para el servidor de imágenes es su capacidad de generar el código a partir de los archivos proto, escribir los archivos proto es realmente sencillo y rápido por lo que nos ahorro mucho tiempo de desarrollo.

4.2. Estándares de codificación.

4.2.1. Estándar codificación C#

- Convenciones de nomenclatura

Al realizar la asignación de un nombre a class, record o struct esto se realizará utilizando la grafía “Pascal Case” a continuación se muestra un ejemplo correcto y uno incorrecto de cómo es que se debe realizar.

Ejemplo de escritura correcta:

```
public class DataService
{

}

public struct ValueCoordinate
{

}
```

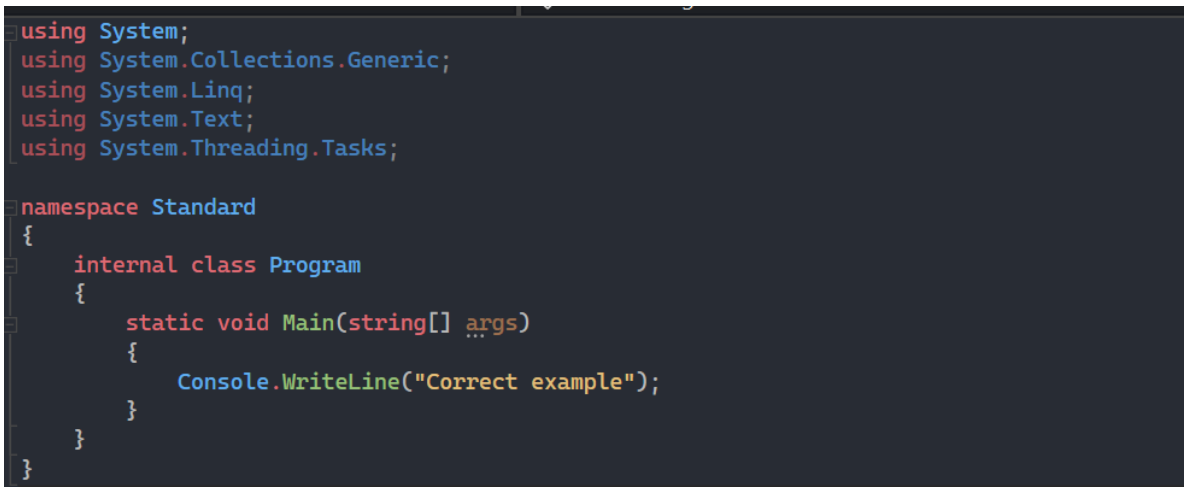
Ejemplo de escritura incorrecta:

```
public class dataservice
{

}

public struct valuEcoordinate
{

}
```

- Indentación

Debe existir un espacio que divide los bloques de código, estructuras de control, o métodos. Este espacio debe estar conformado por cuatro espacios, o en su caso una tabulación.

Ejemplo correcto:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Standard
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Correct example");
        }
    }
}
```

- Ejemplo incorrecto:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Standard
{
internal class Program
{
static void Main(string[] args)
{
Console.WriteLine("Incorrect example");
}
}
}
```

- Comentarios

Los comentarios únicamente están permitidos cuando sea estrictamente necesario, los comentarios deben ser breves y concisos, no se permiten los comentarios demasiado largos y cuyo contenido sea irrelevante para el código.

- Comentarios en una línea

Estos comentarios deberán escribirse en una línea de código en blanco para así facilitar su comprensión, en caso de que el comentario tenga una longitud considerable, se deberá colocar en su lugar un comentario multilínea, así mismo este debe contener un espacio entre el "//" y la descripción del comentario.

Ejemplo correcto:

```
//This is a correct example for comment
Console.WriteLine("Correct example");
```

Ejemplo incorrecto:

```
/*
This is a incorrect example for comment
*/
Console.WriteLine("Incorrect example");
```

- Comentarios multilínea

Los comentarios multilínea deben usarse únicamente cuando el comentario de una línea no sea suficiente, estos comentarios deben ser concisos, no se permiten los comentarios multilínea que no aporten información relevante al código.

- Número por línea

Solo se permite hacer una declaración por la línea de código, no se permiten realizar varias declaraciones en una línea

Ejemplo correcto:

```
int age;
int number;
string address;
string name;
```

Ejemplo incorrecto:

```
int age, number;
string address, name;
```

- Colocación

El ámbito de las variables deberá ser lo más cercano posible a su uso. Sin embargo, si la variable se llegara a usar en múltiples módulos se permitirá su declaración en la cabecera de la clase.

Ejemplo correcto:

```
for (int i = 0; i < limit; i++)
{
    Console.WriteLine("Correct example");
}
```

Ejemplo incorrecto:

```
int i;
for (i = 0; i < limit; i++)
{
    Console.WriteLine("Incorrect example");
}
```

- Nombramiento de métodos

El objetivo de esta sección es definir la notación que se usará para el nombramiento de todos los métodos.

Los métodos deben ser verbos y se debe utilizar la notación conocida como “Pascal Case”, con la primera letra de cada palabra en mayúscula. Así mismo, su nombre debe tener relación con la tarea que éste realizará.

En el caso de que los métodos sean métodos de eventos correspondientes a elementos como botones de WPF el nombramiento de estos deberá seguir las reglas anteriormente descritas y también deberá seguir la siguiente nomenclatura: [Tipo de elemento del evento]_[Funcionalidad a realizar]_[Acción de ejecución].

Ejemplo de declaración correcta:

```
GetAlumnData();
DeleteAlumn();
```

```
private void Button_Login_Click(object sender, RoutedEventArgs e)
```

Ejemplo de declaración incorrecta:

```
registeracademicMember();
deleteAcademicMember();
```

```
private void SaveProfileChanges(object sender, RoutedEventArgs e)
```

- Visibilidad de métodos

Si un método no exige que su acceso sea “private”, entonces se debería de permitir usar un modificador de acceso “default”, no obstante, si se necesita tener mayor accesibilidad del método, este debe ser “protected”. Si el método necesita ser accesible desde cualquier parte del programa, entonces debe ser “public”.

- Idioma

El idioma seleccionado para el desarrollo del proyecto será inglés, es decir, todo el código escrito dentro del proyecto y todo lo que este involucra, será escrito en inglés.

Ejemplo correcto:

```
class Student
{
    int age;
}
```

Ejemplo incorrecto:

```
class Alumno
{
    int edad;
}
```

- Nombramiento de campos y variables.

Para escribir el nombre de una variable se hace uso de la sintaxis “camel Case” a excepción de los casos los cuales serán explicados en secciones siguientes.

Los nombres de las variables deben comenzar con una letra, solo se pueden escribir números dentro del nombre de la variable siempre y cuando sea después de la primera letra.

Ejemplo correcto:

```
string lastName;
int idUser;
```

Ejemplo incorrecto:

```
string nomBrePaTerno;
int IDUSuario;
```

- Nombramiento de campos internal y private.

El nombramiento de este tipo de campos se debe realizar siguiendo la notación Camel (“camelCasing”) además del prefijo “_”

Ejemplo correcto:

```
private IWorkerQueue _workerQueue;
```

Ejemplo incorrecto:

```
private IWorkerQueue WorkerQueue;
```

- Nombramiento de clases

Los nombres de las clases deben seguir el tipo de nombramiento “Pascal Case” es decir la primera letra de cada palabra (incluyendo la primera) debe ser mayúscula, además este nombre debe ser escrito en singular, y debe representar algún sustantivo.

Ejemplo correcto:

```
class User
```

Ejemplo incorrecto:

```
class iniciosesion
```

6.6 Nombramiento de contadores de ciclos

Se aceptará el uso de i, j, o k para contadores de bucle que estén definidos dentro de este. Para bucles anidados complejos, los contadores deben recibir un nombre descriptivo.

Ejemplo correcto:

```
for (int i = 0; i < limit; i++)
{
    Console.WriteLine("Correct example");
}
```

Ejemplo incorrecto:

```
for(int b = 0; b < args.Length; b++)
{
    Console.WriteLine(args[b]);
}
```

- Manejo de excepciones

El manejo de excepciones dentro del proyecto consistirá en capturarlas dentro de bloques try-catch.

- Códigos de estado

Para el mejor manejo del estado de los métodos al momento de ejecución se implementarán códigos de estado en los retornos de los métodos que sean necesarios, para identificar de mejor manera que es lo que ocurre con ejecución de los métodos mencionados. Los códigos de estado de retorno se especifican a continuación:

Descripción	Código
Consulta nula	404
Operación exitosa	200
Operación fallida	500

4.3. Reporte de análisis estático de código.

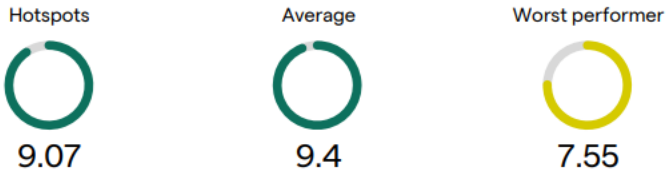
Para el análisis de código estático se decidió usar la plataforma CodeScene, debido a que es una plataforma que analiza el código alojado en GitHub ahorrando tiempo y haciendo muy fácil el análisis, además ofrece métricas sencillas de interpretar y reportes con la información del análisis.

- Journeymate API

Technical Health Overview Report for JourneymateAPI

Behavioral Code Analysis
2023-06-12
by CodeScene

Overall code health



Code Health identifies factors known to impact Maintenance costs and Delivery Risks. View dashboard to examine further.

Hotspots with Code Health Trends

Hotspot Name	Code Health	Identified Code Health Issue	Goals
routinesController.js JourneymateAPI/routinesModule/src/controllers	7	• Many Conditionals • Complex Method • Duplicated Function Blocks	
usersController.js JourneymateAPI/usersModule/src/controllers	8	• Many Conditionals • Complex Method • Complexity Trend Growth • Code Health Degradations	
routineServices.js JourneymateAPI/routinesModule/src/services	9	• Low Overall Code Complexity • High Degree of Code Duplication	
usersModel.js JourneymateAPI/usersModule/src/models	10	• Low Overall Code Complexity	
usersModel.js JourneymateAPI/routinesModule/src/models	10	• Low Overall Code Complexity	
userServices.js JourneymateAPI/usersModule/src/services	10	• Low Overall Code Complexity	
taskController.js JourneymateAPI/tasksModule/src/controllers	10		

- Journeymate Desktop

Technical Health Overview Report for Journeymate-Desktop

Behavioral Code Analysis
2023-06-12
by CodeScene

Overall code health

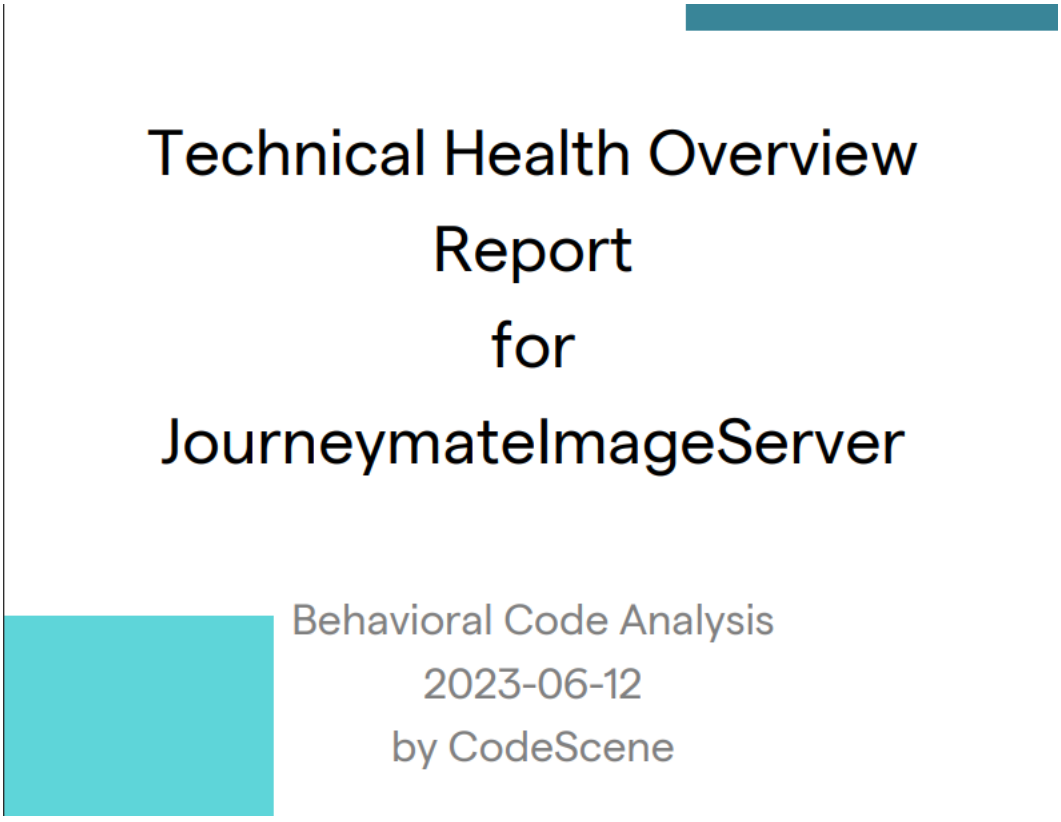
Hotspots	Average	Worst performer
		
9.2	9.62	8.2

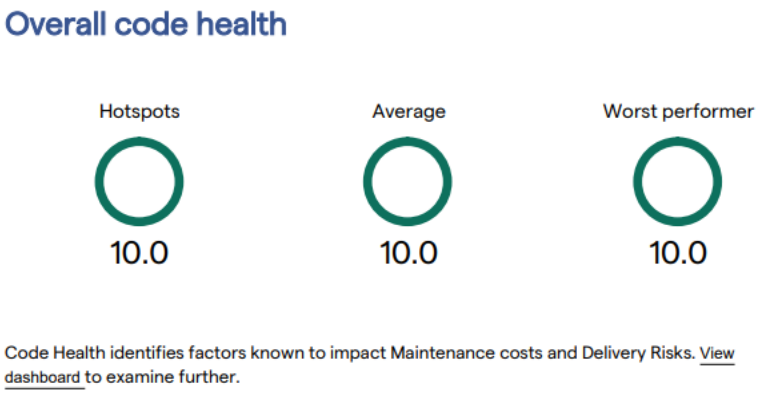
Code Health identifies factors known to impact Maintenance costs and Delivery Risks. View dashboard to examine further.

Hotspots with Code Health Trends

Hotspot Name	Code Health	Identified Code Health Issue	Goals
Page_AddRoutine.xaml.cs Journeymate-Desktop/Client/Pages	8	• Bumpy Road • Deep, Nested Complexity • Complex Method • Code Health Degradations	
MainWindow.xaml.cs Journeymate-Desktop/Client	9	• Bumpy Road • Complex Method • Code Health Degradations	
RoutineLogic.cs Journeymate-Desktop/Logic	9	• Low Overall Code Complexity • Duplicated Function Blocks	
Page_Explorer.xaml.cs Journeymate-Desktop/Client/Pages	9	• Bumpy Road • Deep, Nested Complexity • Code Health Degradations	
Page_EditProfile.xaml.cs Journeymate-Desktop/Client/Pages	9	• Complex Method • Code Health Degradations	
Page_SignUp.xaml.cs Journeymate-Desktop/Client/Pages	9	• Low Overall Code Complexity • Primitive Obsession • Code Health Degradations	
Page_Login.xaml.cs Journeymate-Desktop/Client/Pages	9	• Low Overall Code Complexity • Complex Conditional	
Page_Profile.xaml.cs Journeymate-Desktop/Client/Pages	10	• Low Overall Code Complexity	
Autentication.cs Journeymate-Desktop/Logic	10		

- Journeymate Image Server





5. Pruebas

5.1. Plan de pruebas para la aplicación de servicios.

En el caso de la aplicación de servicios que estamos abordando, se ha diseñado un plan de pruebas exhaustivo para garantizar la calidad y funcionalidad de los módulos principales (Usuario, Tarea, Rutina y Comentarios).

En primer lugar, se realizaron pruebas de unidad en la capa de controladores y servicios de cada uno de los módulos. Estas pruebas se centran en evaluar cada unidad individualmente para asegurarse de que funcione correctamente de manera aislada.

Además de las pruebas de unidad, también se llevaron a cabo pruebas con Postman, esta es una herramienta de colaboración y desarrollo de API que permite probar y documentar servicios web. En este contexto, se utilizaron las capacidades de Postman para realizar pruebas funcionales en los módulos de Usuario, Tarea, Rutina y Comentarios. Estas pruebas se centraron en verificar la interacción correcta entre los diferentes módulos y su integración con otros componentes del sistema.

El objetivo principal de este plan de pruebas fue garantizar la calidad y la correcta funcionalidad de los módulos de Usuario, Tarea, Rutina y Comentarios en la aplicación de servicios. Mediante la combinación de pruebas de unidad y pruebas con Postman, se

pudo asegurar que cada módulo funcione correctamente de manera individual y en conjunto con otros componentes del sistema.

A continuación, se detallan los procesos de prueba mencionados anteriormente.

5.2. Procedimiento de prueba

1. Configuración inicial:
    a. Instala Jest como el framework de pruebas para JavaScript.
    b. Configura Jest para que ejecute las pruebas en tu entorno de desarrollo.
    c. Instala Postman en tu sistema para realizar pruebas de API.
2. Identificación de los métodos a probar:
    a. Se identificaron los métodos utilizados para la conexión con el API, estos métodos son: GET, POST, PATCH, PUT y DELETE
3. Creación de casos de prueba con Jest:
    a. Define un archivo de pruebas separado para cada método a probar.
    b. Dentro de cada archivo de prueba, crea una o varias pruebas individuales para diferentes escenarios (Se realizaron pruebas para los casos de prueba exitosos, erróneos, así como también en los casos donde se presentan excepciones).
    c. Utiliza las funciones y aserciones proporcionadas por Jest para verificar que el resultado de cada método sea el esperado.
    d. Realiza pruebas exhaustivas considerando diferentes casos y situaciones límite.
3. Ejecución de pruebas de unidad con Jest:
    a. Ejecuta las pruebas unitarias utilizando Jest.
    b. Observa los resultados de las pruebas y verifica si todas las pruebas pasan con éxito.
    c. Si una prueba falla, identifica la causa del error y realiza los ajustes necesarios en el código de la aplicación.
    d. Vuelve a ejecutar las pruebas afectadas para confirmar que el problema se ha solucionado correctamente.
4. Pruebas de API con Postman:
    a. Crea una colección de pruebas en Postman para cada método que se desee probar, los métodos probados son: GET, POST, PATCH, PUT y DELETE.
    b. Define solicitudes individuales para diferentes escenarios, asegurándote de incluir datos de entrada válidos y válidos.
    c. Ejecuta las pruebas en Postman y verifica los resultados.
    d. Si una prueba falla, analiza los detalles del error y realiza las modificaciones necesarias en la configuración de la API o en el código de la aplicación.

4.1. Casos de prueba

1. Obtener usuario por nombre de usuario:
    a. Escenario: Devolver el usuario cuando se encuentra.
        i. Resultado esperado: Se espera que se llame al método getUserByUsername de UserService con el nombre de usuario proporcionado en los parámetros de solicitud. Se espera que se llame al método status del objeto de respuesta con el código OK y al método json con el usuario encontrado.
    b. Escenario: Devolver un mensaje de error cuando el usuario no se encuentra.

i. Resultado esperado: Se espera que se llame al método getUserByUsername de UserService con el nombre de usuario proporcionado en los parámetros de solicitud. Se espera que se llame al método status del objeto de respuesta con el código INVALID_DATA.

c. Escenario: Manejar un error durante la obtención del usuario por nombre de usuario y devolver un mensaje de error.
   i. Resultado esperado: Se espera que se llame al método getUserByUsername de UserService con el nombre de usuario proporcionado en los parámetros de solicitud. Se espera que se llame al método status del objeto de respuesta con el código INVALID_DATA y al método json con un mensaje de error.

2. Validar datos no vacíos:
   a. Escenario: Devolver OK cuando se proporcionan todos los campos.
      i. Resultado esperado: Se espera que se llame al método validateNotEmptyData de UserController con el objeto de usuario. Se espera que el resultado sea OK.
   b. Escenario: Devolver DATA_REQUIRED cuando falta un campo.
      i. Resultado esperado: Se espera que se llame al método validateNotEmptyData de UserController con el objeto de usuario. Se espera que el resultado sea DATA_REQUIRED.
         ii. Eliminar usuario:
   c. Escenario: Eliminar un usuario exitosamente.
      i. Resultado esperado: Se espera que se llame al método findUserByUsername de UserService con el nombre de usuario proporcionado en los parámetros de solicitud. Se espera que se llame al método deleteUserByUsername de UserService. Se espera que se llame al método json del objeto de respuesta con un mensaje de éxito.
   d. Escenario: Devolver un error para un usuario inexistente.
      i. Resultado esperado: Se espera que se llame al método findUserByUsername de UserService con el nombre de usuario proporcionado en los parámetros de solicitud. Se espera que se llame al método json del objeto de respuesta con un mensaje de error.
   e. Escenario: Manejar errores durante la eliminación y devolver un mensaje de error.
      i. Resultado esperado: Se espera que se llame al método findUserByUsername de UserService con el nombre de usuario proporcionado en los parámetros de solicitud. Se espera que se llame al método deleteUserByUsername de UserService. Se espera que se llame al método json del objeto de respuesta con un mensaje de error.

3. Validar tipos de datos de entrada:
   a. Escenario: Devolver OK cuando todos los tipos de datos son válidos.
      i. Resultado esperado: Se espera que se llame al método validateDataTypesEntry de UserController con el objeto de usuario. Se espera que el resultado sea OK.
   b. Escenario: Devolver DATA_REQUIRED cuando el campo "name" no es una cadena.
      i. Resultado esperado: Se espera que se llame al método validateDataTypesEntry de UserController con el objeto de usuario. Se espera que el resultado sea DATA_REQUIRED.

4. Validar que el usuario no esté registrado:
    a. Escenario: Devolver OK cuando el usuario no está registrado.
        i. Resultado esperado: Se espera que se llame al método findUserByEmail y findUserByUsername de UserService con el correo electrónico y el nombre de usuario proporcionados en el objeto de usuario. Se espera que el resultado sea OK.
    b. Escenario: Devolver INVALID_DATA cuando el usuario ya está registrado.
        i. Resultado esperado: Se espera que se llame al método findUserByEmail y findUserByUsername de UserService con el correo electrónico y el nombre de usuario proporcionados en el objeto de usuario. Se espera que el resultado sea INVALID_DATA.

5. Validar la función "getAllTasks":
    a. Escenario: Debe recuperar todas las tareas correctamente.
        i. Resultado esperado: Se espera que se llame al método "getAllTasks" de TaskService. Se espera que el estado de respuesta sea OK (200) y que el JSON de respuesta contenga el código de estado OK, un mensaje indicando que se recuperaron todas las tareas y las tareas recuperadas.
    b. Escenario: Debe devolver "Tasks not found" cuando no hay tareas disponibles.
        i. Resultado esperado: Se espera que se llame al método "getAllTasks" de TaskService. Se espera que el estado de respuesta sea NOT_FOUND (404) y que el JSON de respuesta contenga el código de estado NOT_FOUND, un mensaje indicando que no se encontraron tareas y la respuesta sea null.
    c. Escenario: Debe manejar errores y registrarlos.
        i. Resultado esperado: Se espera que se llame al método "getAllTasks" de TaskService y que arroje un error. Se espera que se registre el error utilizando el método "error" de Logger. Se espera que el estado de respuesta sea PROCESS_ERROR (500) y que el JSON de respuesta contenga el código de estado PROCESS_ERROR, un mensaje indicando que ocurrió un error y la respuesta sea null.
6. Validar la función "getTaskById":
    a. Escenario: Debe recuperar una tarea por ID correctamente.
        i. Resultado esperado: Se espera que se llame al método "getTaskByID" de TaskService con el ID de la tarea proporcionado. Se espera que el estado de respuesta sea OK (200) y que el JSON de respuesta contenga el código de estado OK, un mensaje indicando que esta es la tarea solicitada y la tarea recuperada.
    b. Escenario: Debe devolver "Task not found" cuando la tarea no existe.
        ii. Resultado esperado: Se espera que se llame al método "getTaskByID" de TaskService con el ID de la tarea proporcionado. Se espera que el estado de respuesta sea NOT_FOUND (404) y que el JSON de respuesta contenga el código de estado NOT_FOUND, un mensaje indicando que no se encontró la tarea y la respuesta sea null.
    c. Escenario: Debe manejar errores y registrarlos.
        iii. Resultado esperado: Se espera que se llame al método "getTaskByID" de TaskService con el ID de la tarea proporcionado y que arroje un error. Se espera que se registre el error utilizando el método "error" de Logger. Se espera que el estado de respuesta sea PROCESS_ERROR (500) y que el JSON de respuesta contenga el código de estado PROCESS_ERROR, un mensaje indicando que ocurrió un error y la respuesta sea null.
7. Validar la función "getAllTasksByIdRoutine":
    a. Escenario: Debe recuperar tareas por ID de rutina correctamente.

i. Resultado esperado: Se espera que se llame al método "getTaskByIDRoutine" de TaskService con el ID de rutina proporcionado. Se espera que el estado de respuesta sea OK (200) y que el JSON de respuesta contenga el código de estado OK, un mensaje indicando que estas son las tareas solicitadas y las tareas recuperadas.

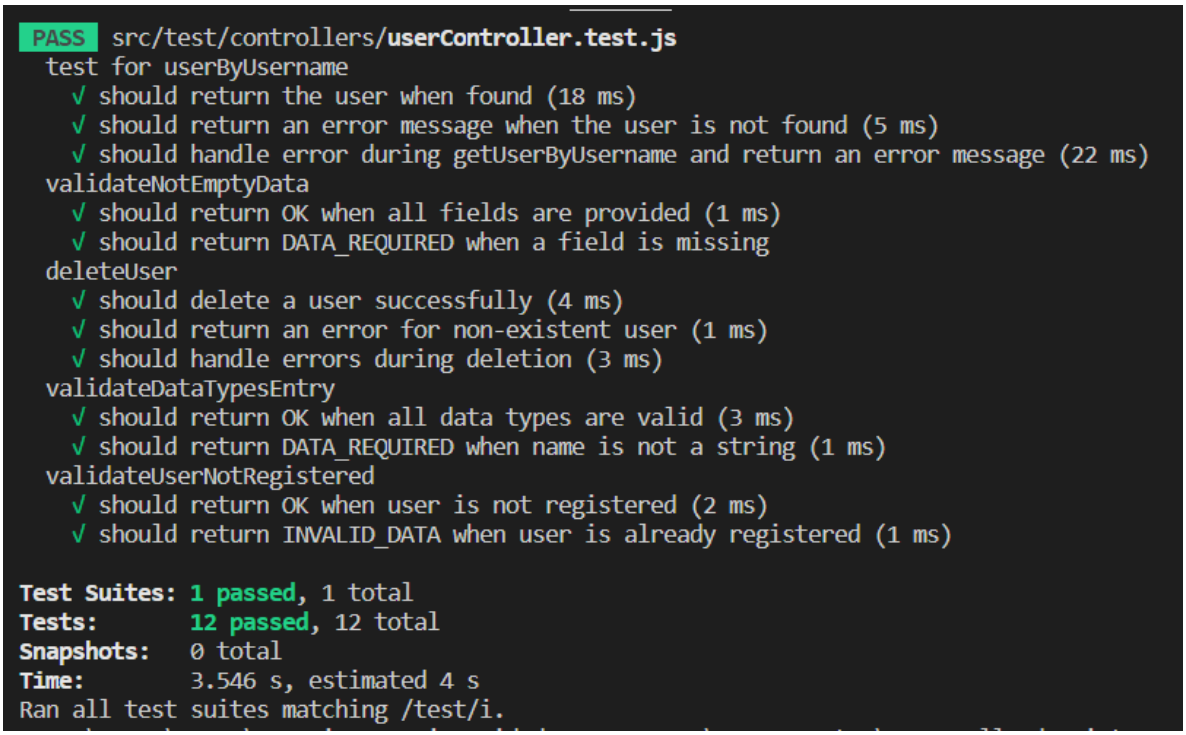
b. Escenario: Debe manejar errores.

i. Resultado esperado: Se espera que se llame al método "getTaskByIDRoutine" de TaskService con el ID de rutina.

Cabe mencionar que los anteriores fueron los casos de prueba para las pruebas de unidad, las pruebas realizadas en postman se muestran en el siguiente punto:

4.2. Resultados

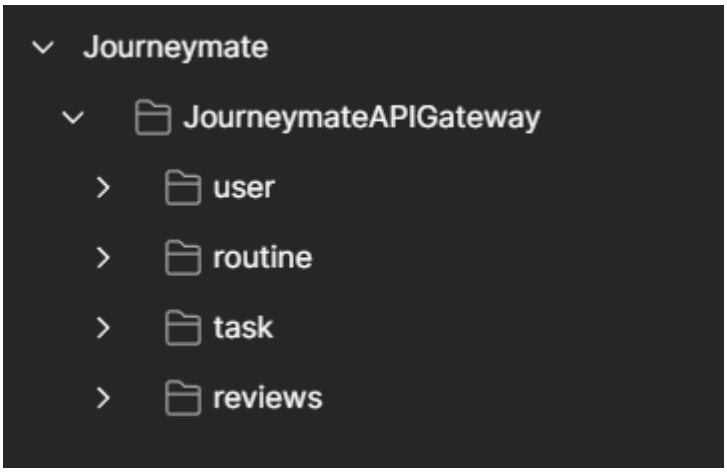
1. Pruebas de unidad del módulo “Usuario”.



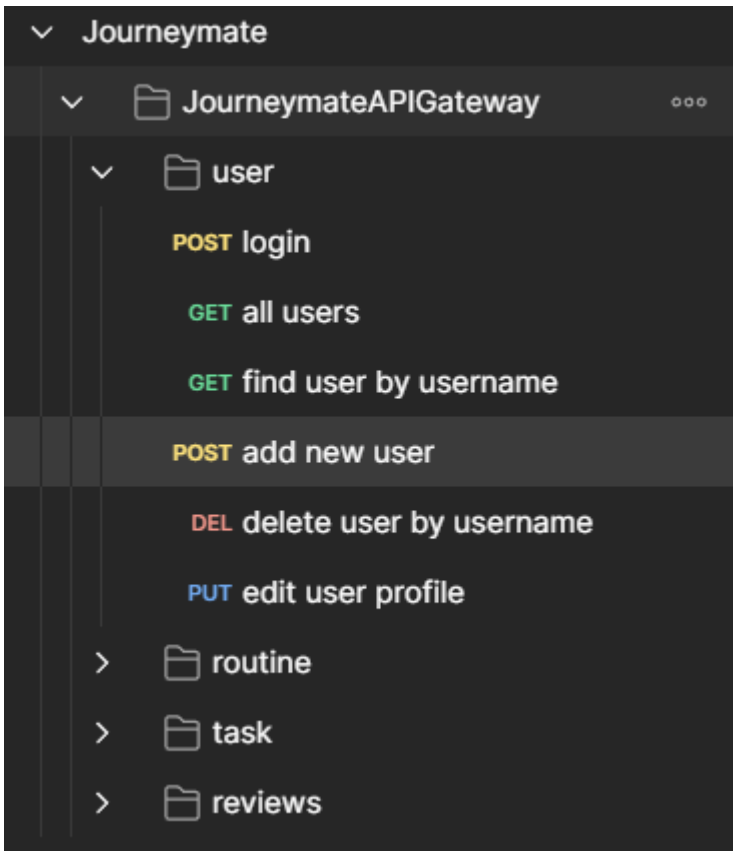
2. Pruebas de unidad del módulo “Tareas”.

```
PASS src/test/services/taskServiceTest.test.js
  getAllTasks
    √ should return an empty array if no tasks found (8 ms)
    √ should return an array of tasks if tasks found (1 ms)
    √ should handle error and return an empty array (6 ms)
  getTaskByID
    √ should return a task object if task found (2 ms)
    √ should return CodeStatus.NOT_FOUND if task not found (2 ms)
    √ should handle error and return an empty object (3 ms)
  getTaskByIDRoutine
    √ should return an array of tasks if routine and tasks exist (10 ms)
    √ should return CodeStatus.PROCESS_ERROR if routine not found (3 ms)
    √ should handle error and return an empty array (5 ms)

PASS src/test/controllers/taskControllerTest.test.js
  getAllTasks
    √ should retrieve all tasks successfully (9 ms)
    √ should return "Tasks not found" when no tasks are available (2 ms)
    √ should handle errors and log them (14 ms)
  getTaskById
    √ should retrieve a task by ID successfully (4 ms)
    √ should return "Task not found" when the task does not exist (4 ms)
    √ should handle errors and log them (2 ms)
  getAllTasksByIdRoutine
    √ should retrieve tasks by routine ID successfully (3 ms)
    √ should handle errors (4 ms)
  deleteTask
    √ should delete a task successfully (5 ms)
    √ should handle errors (4 ms)
  validateTypesOfDataEntry
    √ should return OK for valid data entry (1 ms)
    √ should return INVALID_DATA for invalid name (2 ms)
    √ should return INVALID_DATA for invalid task_description (3 ms)
    √ should return INVALID_DATA for invalid address (2 ms)
    √ should return INVALID_DATA for invalid budget (2 ms)
  validateDataNotEmpty
    √ should return OK for all fields filled (1 ms)
    √ should return DATA_REQUIRED for undefined name (5 ms)
    √ should return DATA_REQUIRED for undefined task_description
    √ should return DATA_REQUIRED for undefined address (1 ms)
    √ should return DATA_REQUIRED for undefined budget
```

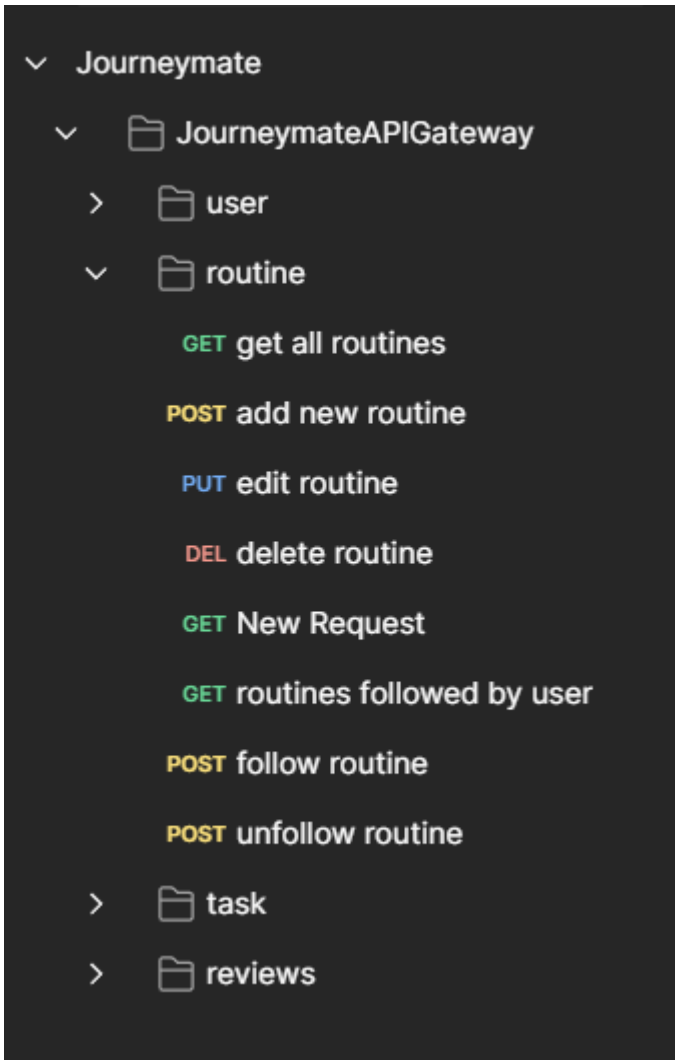
3. Pruebas en Postman:
   3.1 Paquete de pruebas general de apigateway:



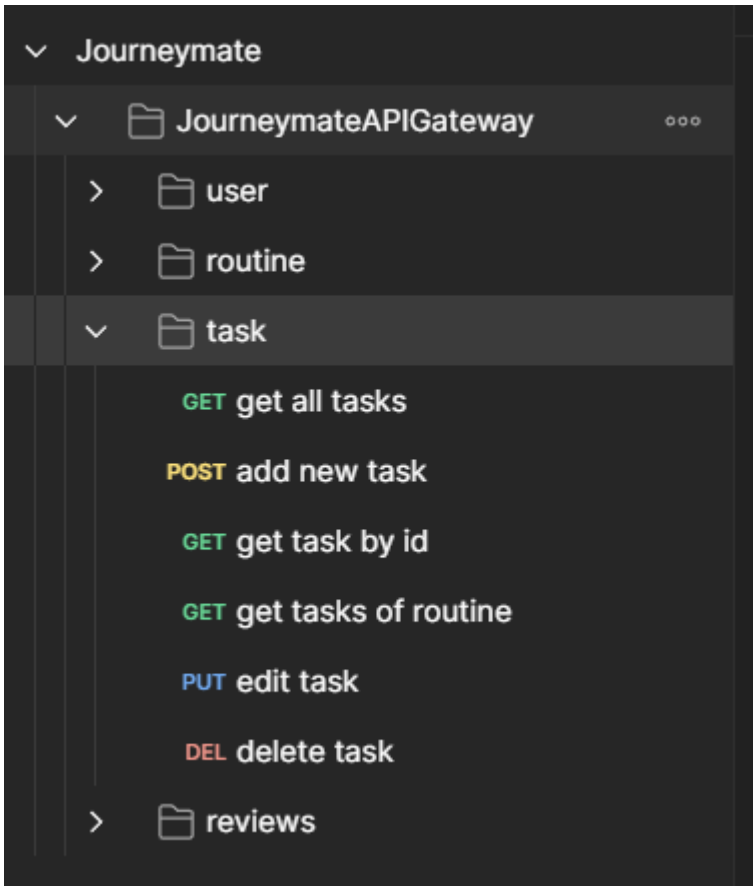
3.2 Pruebas de modulo user:



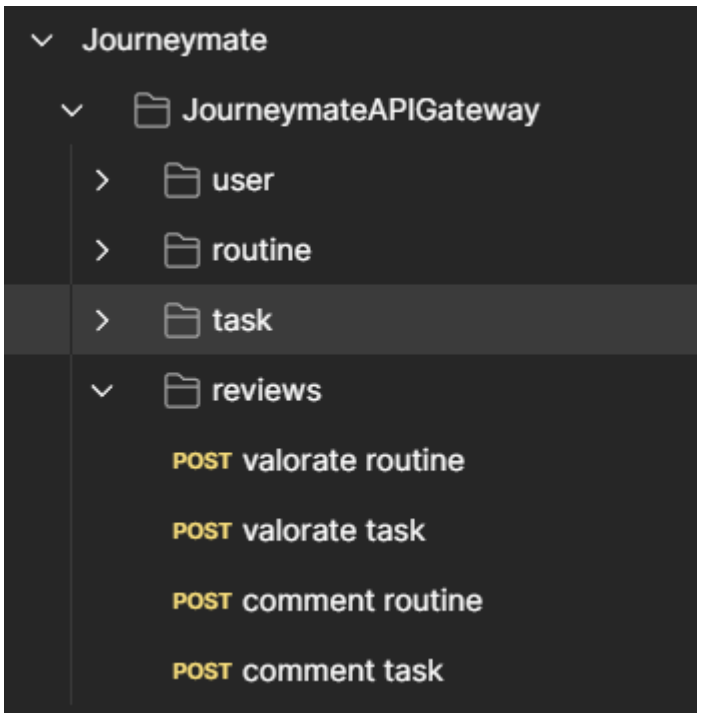
3.3 Pruebas de modulo Routine:



3.4 Pruebas del modulo Task:



3.5 Pruebas al módulo Review:



5. Estrategias de despliegue

Para el despliegue de Journeymate, se ha utilizado Docker para encapsular cada módulo, lo que permite una fácil implementación y escalabilidad. A continuación, se detallan las estrategias implementadas:

1. Contenedores Docker: Cada módulo, incluyendo Usuario, Rutina, Tarea, Comentarios y el API Gateway, se ha encapsulado en un contenedor Docker independiente. Esto facilita la gestión y el despliegue de los módulos de forma aislada, asegurando su portabilidad y reproducibilidad.
2. API Gateway con Ocelot: Se ha implementado un API Gateway utilizando Ocelot en ASP.NET. El API Gateway actúa como punto de entrada único para los clientes web, móviles y de escritorio, enrutando las solicitudes a los módulos correspondientes de manera transparente. También permite gestionar aspectos de autenticación, autorización y caching.
3. Base de datos MongoDB: Se ha utilizado una base de datos MongoDB para almacenar los datos relacionados con Usuario, Rutina, Tarea y Comentarios. La base de datos se ha configurado dentro de un contenedor Docker y se ha asociado a un volumen persistente para garantizar la persistencia de los datos incluso en caso de reinicios o actualizaciones.
4. Servidor de imágenes con gRPC: Se ha implementado un servidor de imágenes utilizando gRPC para facilitar la transferencia eficiente de imágenes entre los módulos y los clientes. Esta elección de tecnología permite una comunicación rápida y de alto rendimiento, asegurando una experiencia fluida al visualizar y compartir imágenes en Journeymate.
5. Cliente web en contenedor Docker: El cliente web se ha encapsulado en un contenedor Docker independiente, lo que permite su fácil despliegue y gestión. El cliente web se conecta al API Gateway para acceder a los distintos módulos y funcionalidades de Journeymate.
6. Acceso a los módulos a través del API Gateway: Los clientes desktop, móvil y web se conectan al API Gateway para acceder a los distintos módulos de Journeymate. Esto asegura una capa de abstracción y seguridad, ya que los clientes no tienen acceso directo a los módulos individuales.

Con estas estrategias implementadas, Journeymate se encuentra en un entorno altamente escalable y eficiente. El uso de contenedores Docker permite una fácil administración de los módulos, mientras que el API Gateway con Ocelot simplifica el enrutamiento de las solicitudes de los clientes y la gestión de la seguridad. Además, la utilización de MongoDB y el servidor de imágenes con gRPC garantizan un almacenamiento confiable y una transferencia eficiente de datos en el sistema Journeymate.

6. Conclusiones

Como equipo concluimos que realizar este proyecto al ser prácticamente un desarrollo completo y con una arquitectura que no habíamos implementado fue todo un reto, además al

desarrollar una idea original el proceso fue más interesante, en el pasado todas las problemáticas para las que habíamos desarrollado una solución ya estaban predefinidas y lo único que hacíamos era hacer nuestra versión de la solución, pero en este resolver la problemática fue un proceso interesante ya que prácticamente hicimos todo desde cero, únicamente con las ideas establecidas entre los miembros del equipo.

En la cuestión de la tecnología usar una arquitectura de microservicios nos proporcionó una nueva visión sobre cómo se pueden resolver distintas problemáticas que no sean usando una arquitectura monolítica, además salir de nuestra zona de confort utilizando una tecnología que no conocíamos presento un reto sin embargo, adaptarnos a ella fue bastante fácil y el proceso de desarrollo fue cómodo debido a la naturaleza de NodeJS.

Por último realizar un desarrollo fuera de nuestra zona de confort nos proporcionó nuevas perspectivas y conocimientos que sin duda nos serán útiles en el futuro.