Ingeniería del Software

Grado de Ingeniería Informática
Curso 2020-2021
Antonio Moruno Gracia, David Perez Dueñas, Marcos Rivera Gavilán

Prácticas

Proyecto 26

1. Introducción

La práctica 2 se ha realizado a lo largo de 3 semanas concretamente del 8 al 22 de octubre de 2020. Para esta práctica hemos tenido que realizar el análisis de requisitos de un sistema de gestión de la información de parques naturales, para una empresa de gestión de estos entornos en Andalucía. Esta empresa es la encargada de registrar los espacios que gozan de este reconocimiento, garantizando su mantenimiento y conservación, así como programar rutas por sus senderos.

Los objetivos a cumplir con la realización de la práctica eran los siguientes:

- Aprender a extraer requisitos a partir de la entrevista con el cliente.
- Identificar, refinar y documentar los distintos tipos de requisitos del sistema.
- Utilizar historias de usuario para crear la lista de producto según la metodología Scrum.
- Comenzar a planificar y priorizar funcionalidades de cara a su futura implementación
- Aplicar la técnica de casos de uso de UML para detallar el análisis de requisitos funcionales. Uso de Visual Paradigm.
- Validar los requisitos con el cliente para resolver dudas surgidas durante la toma de requisitos.

La práctica 3 se ha realizado a lo largo de 2 sesiones, concretamente la del 12 y la del 19 de noviembre de 2020. En esta práctica hemos tenido que realizar el diseño de la estructura y del comportamiento del sistema. Para ello hemos hecho uso de UML y de la herramienta Visual Paradigm Community Edition.

Los objetivos a cumplir con la realización de la práctica eran los siguientes:

- Aplicar la técnica de diagramas de clase de UML para el diseño estructural del sistema.
- Avanzar en el conocimiento y desarrollo del sistema validando y refinando la especificación de requisitos.
- Aplicar la técnica de diagramas de secuencia de UML para el diseño del comportamiento del sistema.
- Avanzar en el conocimiento y desarrollo del sistema para poder abordar la fase de implementación y pruebas

La práctica 4 se ha realizado a lo largo de 2 sesiones, concretamente la del 26 de noviembre y la del 3 de diciembre de 2020. En esta práctica hemos tenido que implementar el sistema que previamente habíamos diseñado. Para ello hemos hecho uso del IDE Visual Studio Code, la herramienta de control de versiones Git y la herramienta Google Test para testear las funcionalidades implementadas.

Los objetivos a cumplir con la realización de la práctica eran los siguientes:

- Aprender a utilizar un entorno de desarrollo integrado (IDE) para la implementación de código fuente.
- Abordar la implementación del sistema siguiendo la metodología Scrum.
- Ser capaces de diseñar y codificar pruebas unitarias para el sistema en desarrollo.

2. Planificación

Práctica 2

En esta práctica le ha tocado ser Product Owner a Marcos Rivera Gavilán y lo largo de estas tres semanas, hemos distribuido el trabajo de la siguiente forma:

2.1 Semana 1

El trabajo que debíamos realizar para esta semana era el siguiente:

- Extraer requisitos a partir de la entrevista con el cliente.
- Identificar, refinar y documentar los distintos tipos de requisitos del sistema.

Y acordamos distribuirlo de la siguiente forma:

- **Antonio:** Requisitos funcionales
- **Marcos:** Requisitos de información
- **David:** Requisitos no funcionales

Siendo cada uno responsable de:

- Añadir y documentar sus requisitos en el documento de Requisitos del Sistema

2.2 Semana 2

El trabajo que debíamos realizar para esta semana era el siguiente:

- Utilizar historias de usuario para crear la lista de producto según la metodología Scrum.
- Comenzar a planificar y priorizar funcionalidades de cara a su futura implementación.

Y acordamos distribuirlo de la siguiente forma:

- **Antonio:** Análisis de los requisitos funcionales para obtener historias de usuario
- **Marcos:** Análisis de los requisitos de información para obtener historias de usuario
- **David:** Análisis de los requisitos no funcionales para obtener historias de usuario

Siendo cada uno responsable de:

- Añadirlas al tablón haciendo uso de las funcionalidad de historias de usuario
- Añadirlas al documento Historias de Usuario

2.3 Semana 3

El trabajo que debíamos realizar para esta semana era el siguiente:

- Aplicar la técnica de casos de uso de UML para detallar el análisis de requisitos funcionales. Uso de Visual Paradigm.
- Validar los requisitos con el cliente para resolver dudas surgidas durante la toma de requisitos

Y acordamos distribuirlo de la siguiente forma:

- **Antonio:** Creación de los casos de uso
- **Marcos:** Validación de los requisitos y redacción de este documento.
- **David:** Diseño del diagrama UML

Siendo cada uno responsable de aportar su parte en tiempo y forma antes de la fecha límite para la entrega de la práctica.

Todos los documentos citados en los párrafos anteriores así como las tarjetas con las historias de usuario, se encuentran respectivamente en la sección Knowledge Base y en el Panel de Desarrollo de la sección Agile Boards de nuestra instancia de YouTrack cuyo enlace es el siguiente:

https://uco-is2021-eq26.myjetbrains.com/

Práctica 3

En esta práctica le ha tocado ser Product Owner a Antonio Moruno Gracia y lo largo de estas dos semanas, hemos distribuido el trabajo de la siguiente forma:

3.1 Semana 1

El trabajo que debíamos realizar para esta semana era el siguiente:

- Aplicar la técnica de diagramas de clase de UML para el diseño estructural del sistema.
- Avanzar en el conocimiento y desarrollo del sistema validando y refinando la especificación de requisitos.

Y acordamos distribuirlo de la siguiente forma:

- **Antonio:** Validación de los requisitos funcionales frente a los casos de uso y validación de los casos de uso frente a las clases
- **Marcos:** Diseño de clases
- **David:** Diagrama de clases

Siendo cada uno responsable de:

- Aportar su parte en tiempo y forma antes de la fecha de límite para la entrega de la práctica

3.2 Semana 2

El trabajo que debíamos realizar para esta semana era el siguiente:

- Aplicar la técnica de diagramas de secuencia de UML para el diseño del comportamiento del sistema.
- Avanzar en el conocimiento y desarrollo del sistema para poder abordar la fase de implementación y pruebas

Y acordamos distribuirlo de la siguiente forma:

- **Antonio:** Diagrama de secuencia 1 y redacción de este documento
- **Marcos:** Diagrama de secuencia 2
- **David:** Diagrama de secuencia 3

Siendo cada uno responsable de:

Aportar su parte en tiempo y forma antes de la fecha límite para la entrega de la práctica.

La organización del trabajo así como las actas de las reuniones se encuentran en el siguiente enlace:

https://uco-is2021-eq26.myjetbrains.com/

Práctica 4

En esta práctica le ha tocado ser Product Owner a David Perez Dueñas y lo largo de estas dos semanas, hemos distribuido el trabajo de la siguiente forma:

4.1 Semana 1

El trabajo que debíamos realizar para esta semana era el siguiente:

- Aprender a utilizar un entorno de desarrollo integrado (IDE) para la implementación de código fuente.
- Abordar la implementación del sistema siguiendo la metodología Scrum

4.2 Semana 2

El trabajo que debíamos realizar para esta semana era el siguiente:

- Ser capaces de diseñar pruebas unitarias para el sistema en desarrollo.
- Codificar pruebas unitarias.

Esta planificación corresponde a las clases teóricas, adicionalmente hemos realizado 4 sprints a través de los cuales hemos ido desarrollando el sistema. En estos sprints tal y como queda reflejado en la sección de implementación del sistema hemos ido alternando para ser Product Owner y hemos planificado el desarrollo siguiendo la metodología Scrum tal y como queda reflejado en la instancia de youtrack cuyo enlace es el siguiente:

https://uco-is2021-eq26.myjetbrains.com/

3. Especificación de requisitos (P2)

En esta sección procederemos a describir los requisitos del sistema que se extrajeron el pasado jueves 8 de octubre en la reunión.

3.1. Requisitos funcionales

RF1. Automatización de la gestión

El sistema debe permitir a los administrativos gestionar la información de los parques naturales de Andalucía abiertos a los visitantes, así como sus senderos y rutas. Las rutas serán dinámicas es decir se crean con una fecha y un recorrido determinado. En caso de que se elimine una ruta debe suministrarse a los administrativos la información de los senderistas que estaban apuntados a ella para que les avisen de que se ha cancelado. También deben poder crearse rutas exclusivas. Por ejemplo: para los estudiantes de un colegio. O adaptadas a las necesidades especiales de los senderistas

RF2. Senderistas y rutas

El sistema debe permitir al administrador la reserva y la cancelación de una ruta por parte de un senderista, siempre y cuando queden más de 15 minutos para el inicio de la misma. Si esta ruta quedará llena, no se podrían apuntar más senderistas a ella. Por otro lado, como al eliminar una reserva quedará en la ruta una plaza libre, el sistema debe permitir que otro senderista se pueda apuntar a esta ruta dentro del tiempo mencionado anteriormente.

RF3. Información de rutas

Previo al inicio de cada ruta el sistema debe suministrar al monitor la información de la ruta que va a realizar así como el número de senderistas que van a participar e información sobre los mismos por si tuviesen requisitos especiales ej. movilidad reducida. También el sistema deberá ser capaz de suministrar la información a un segundo monitor suplente, el cual se ocupará de la ruta en caso de que el monitor principal sea incapaz de realizarla.

RF4. Notificación de incidencias

El sistema debe permitir a los monitores notificar incidencias que ocurran durante las rutas. Estas notificaciones las recibirán los administrativos, para que así estos puedan por ejemplo, eliminar o modificar una ruta o sendero del parque de forma temporal o actualizar la información de dicha ruta.

RF5. Gestión de nóminas

El sistema debe permitir a los administrativos generar un reporte de las horas trabajadas por cada monitor, en base al número de rutas que este ha realizado y la duración de las mismas.

3.2. Requisitos de información

RI1. Información sobre los parques

El sistema debe almacenar la siguiente información sobre los parques naturales: identificador, nombre, localización en el mapa, municipio, provincia, superficie, fecha en la que se le declaró parque natural y sus premios, horario.

RI2. Información sobre los senderos

El sistema debe almacenar la siguiente información sobre los senderos: identificador, nombre, nivel de dificultad, estado del sendero y parque al que pertenece.

RI3. Información sobre las rutas

El sistema debe almacenar la siguiente información sobre las rutas: identificador, nombre, monitor que las dirige y suplente, modalidad a pie o en bici, longitud, duración de la ruta, pueden ser exclusivas ej. solo colegios y número de plazas, participantes inscritos y sus necesidades.

RI4. Información sobre los monitores

El sistema debe almacenar la siguiente información sobre los monitores: nombre y apellidos, DNI, fecha de nacimiento, teléfono de contacto, dirección, correo electrónico, horas trabajadas.

RI5. Información sobre los administrativos

El sistema debe almacenar la siguiente información sobre los administrativos: nombre y apellidos, DNI, fecha de nacimiento, dirección, correo electrónico.

RI6. Información sobre los senderistas

El sistema debe almacenar la siguiente información sobre los senderistas que participan en las rutas: nombre y apellidos, fecha de nacimiento, DNI, teléfono de contacto y si tienen requisitos especiales ej. movilidad reducida, ruta en la que participan.

3.3. Requisitos no funcionales

RNF1. Lenguaje

El programa debe de estar escrito en C++.

RNF2. Interfaz

La interfaz del programa será la terminal.

RNF3. Usabilidad

El sistema debe de estar operativo para todas los parques en horario de oficina (9:00-18:00). El sistema no debe de exceder los 5 segundos de caída.

RNF4. Acceso

Los empleados del parque deben de identificarse para acceder al sistema

RNF5. Normativa

Los datos de los senderistas deben de ser tratados conforme a la GPRD

4. Análisis de requisitos (P2)

4.1 Historias de usuario

En esta subsección procederemos a describir las historias de usuario en base a los requisitos que se extrajeron el pasado jueves 8 de octubre en la reunión.

HU1. COMO administrativo QUIERO administrar la información de los parques PARA actualizar la información del sistema

Descripción:

Los administrativos deben poder modificar la información almacenada sobre los parques.

Haciendo uso del identificador del parque, el sistema primeramente deberá comprobar si existe o no:

- Si no existe, el administrativo lo añadirá o cambiará el identificador introducido.
- Si existe, el administrador elegirá qué hacer con el parque: modificarlo, eliminarlo o simplemente consultarlo.

Criterios de aceptación:

1. El sistema debe comprobar que sea un administrativo el que modifica la información del parque ya que son los únicos con autoridad para hacerlo
2. El sistema debe impedir registrar más de una vez el mismo parque
3. El sistema debe impedir que se registre un parque sin rellenar los campos obligatorios del formulario
4. Si se elimina un parque del sistema este debe encargarse de eliminar también los senderos y las rutas asociadas al mismo

HU2. COMO administrativo QUIERO administrar la información de los senderos PARA actualizar la información del sistema

Descripción:

Los administrativos deben poder modificar la información almacenada sobre los senderos.

Haciendo uso del identificador del parque, el sistema primeramente deberá comprobar si existe o no:

- Si no existe, el administrativo deberá introducir un identificador válido para buscar el sendero a tratar.
- Si existe, el administrador introducirá el identificador del sendero y el sistema deberá comprobar si existe o no:
    - Si no existe, el administrativo lo añadirá o cambiará el identificador introducido.
    - Si existe, el administrador elegirá que hacer con el sendero: modificarlo, eliminarlo o simplemente consultarlo.

Criterios de aceptación:

1. El sistema debe comprobar que sea un administrativo el que modifica la información del sendero ya que son los únicos con autoridad para hacerlo.
2. El sistema debe impedir registrar más de una vez el mismo sendero.
3. El sistema debe impedir que se registre un sendero sin rellenar los campos obligatorios del formulario.
4. Si se elimina un sendero del sistema este debe encargarse de eliminar también las rutas asociadas al mismo

HU3. COMO administrativo QUIERO administrar la información de las rutas PARA actualizar la información del sistema

Descripción:

Los administrativos deben poder modificar la información almacenada sobre las rutas.

Haciendo uso del identificador del parque, el sistema primeramente deberá comprobar si existe o no:

- Si no existe, el administrativo deberá introducir un identificador válido para buscar la ruta a tratar.
- Si existe, el administrador introducirá el identificador de la ruta y el sistema deberá comprobar si existe o no:
    - Si no existe, el administrativo la añadirá o cambiará el identificador introducido.
    - Si existe, el administrador elegirá qué hacer con la ruta: modificarla, eliminarla o simplemente consultarla.

Criterios de aceptación:

1. El sistema debe impedir registrar más de una ruta a la vez en el mismo sendero y a la misma hora.
2. El sistema debe impedir que se registre una ruta sin rellenar los campos obligatorios del formulario.
3. Si se elimina una ruta del sistema este debe encargarse de notificarlo a los administrativos para que avisen a los senderistas

HU4. COMO monitor QUIERO administrar la información de las rutas PARA actualizar la información del sistema

Descripción:

Los monitores deben poder modificar la información almacenada sobre las rutas.

Haciendo uso del identificador del parque, el sistema primeramente deberá comprobar si existe o no:

- Si no existe, el monitor deberá introducir un identificador válido para buscar la ruta a tratar.
- Si existe, el monitor introducirá el identificador de la ruta y el sistema deberá comprobar si existe o no:
    - Si no existe, el monitor la añadirá o cambiará el identificador introducido.
    - Si existe, el monitor elegirá qué hacer con la ruta: modificarla, eliminarla o simplemente consultarla.

Criterios de aceptación:

1. El sistema debe impedir registrar más de una ruta a la vez en el mismo sendero y a la misma hora.
2. El sistema debe impedir que se registre una ruta sin rellenar los campos obligatorios del formulario.
3. Si se elimina una ruta del sistema este debe encargarse de notificarlo a los administrativos para que avisen a los senderistas

HU5. COMO administrativo QUIERO administrar la información de los senderistas PARA actualizar la información del sistema

Descripción:

Los administrativos deben poder modificar la información almacenada sobre los senderistas.

Haciendo uso del DNI del senderista, el sistema primeramente deberá comprobar si existe o no:

- Si no existe, el administrativo lo añadirá o cambiará el DNI introducido.
- Si existe, el administrador elegirá qué hacer con el senderista: modificarlo, eliminarlo o simplemente consultarlo.

Criterios de aceptación:

1. El sistema debe comprobar que sea un administrativo el que modifica la información almacenada sobre un senderista ya que son los únicos con autoridad para hacerlo.
2. El sistema debe impedir registrar más de una vez el mismo senderista.
3. El sistema debe impedir que se registre un senderista sin rellenar los campos obligatorios del formulario.

HU6. COMO administrativo QUIERO añadir y eliminar senderistas a las rutas PARA poder organizarlas

Descripción:

Los administrativos deben poder apuntar y desapuntar a los senderistas a las distintas rutas del parque.

Una vez el administrativo acceda a la ruta (como se explica en la HU3), haciendo uso del DNI del senderista, los administrativos podrán añadir y eliminar senderistas a la ruta.

Criterios de aceptación:

1. El sistema debe comprobar que sea un administrativo el que apunta o desapunta un senderista de la ruta ya que son los únicos con autoridad para hacerlo.
2. El sistema debe impedir registrar más de una vez el mismo senderista.
3. El sistema debe impedir que se registre un senderista sin rellenar los campos obligatorios del formulario.

HU7. COMO monitor QUIERO disponer de un listado de los senderistas que participan en la ruta PARA poder prepararla

Descripción:

Los monitores deben poder acceder a un listado con la información necesaria sobre la ruta que van a realizar.

Una vez el monitor acceda a la ruta (como se explica en la HU4), estos seleccionarán la opción de mostrar senderistas para acceder al listado de estos, y así poder preparar la ruta.

Criterios de aceptación:

1. El sistema debe comprobar que sea un monitor el que solicita este listado ya que son los únicos con autoridad para hacerlo.
2. El listado debe incluir la siguiente información: Senderos por los que pasa la ruta, duración de la ruta, número de senderistas que participan en la ruta y si alguno tiene necesidades especiales, así como el material necesario para la realización de la misma.
3. El sistema debe impedir que se extraiga este listado hasta que no falten menos de 15 minutos para que empiece dicha ruta ya que hasta entonces la información del mismo puede cambiar .

HU8. COMO monitor QUIERO poder notificar incidencias en las rutas PARA informar a los administrativos del parque de lo ocurrido

Descripción:

Los monitores deben poder notificar incidencias sobre las rutas.

Una vez el monitor acceda a la ruta (como se explica en la HU4), los monitores podrán dar parte sobre las incidencias que hayan ocurrido en esta para que estas se almacenen en el sistema.

Los administrativos, accediendo a la ruta (como se explica en la HU3), podrán consultarla y así observar las incidencias que hayan ocurrido.

Criterios de aceptación:

1. El sistema debe comprobar que sea un monitor y que dicho monitor haya realizado esa ruta para poder notificar las incidencias.
2. El sistema debe impedir que se registre una incidencia sin rellenar los campos obligatorios del formulario.
3. El sistema debe notificar a los administrativos de la incidencia para que actúen en consecuencia.

HU9. COMO administrativo QUIERO administrar la información de la plantilla PARA actualizar la información del sistema

Descripción:

Los administrativos deben poder modificar la información almacenada sobre la plantilla.

Haciendo uso del DNI del trabajador, el sistema primeramente deberá comprobar si existe o no:

- Si no existe, el administrativo lo añadirá o cambiará el DNI introducido.
- Si existe, el administrador elegirá qué hacer con el trabajador: modificarlo, eliminarlo o simplemente consultarlo.

Criterios de aceptación:

1. El sistema debe comprobar que sea un administrativo el que añade un trabajador ya que son los únicos con autoridad para hacerlo.
2. El sistema debe impedir registrar más de una vez el mismo trabajador.
3. El sistema debe impedir que se registre un trabajador sin rellenar los campos obligatorios del formulario.

HU10. COMO administrativo QUIERO disponer del cómputo de horas trabajadas de los monitores PARA poder pagarles

Descripción:

Los administrativos deben poder modificar la información almacenada sobre la plantilla.

Haciendo uso del DNI del trabajador, el sistema primeramente deberá comprobar si existe o no:

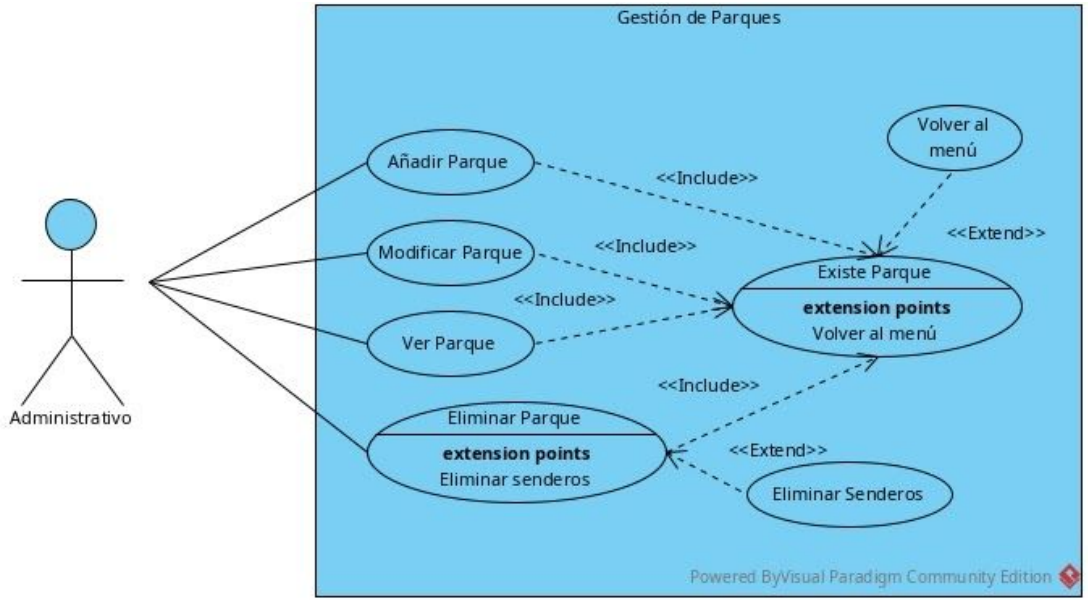
- Si no existe, el administrativo cambiará el DNI introducido para buscar un monitor que se encuentre en el sistema.
- Si existe, el administrador indicará al sistema que le muestre el cómputo de horas trabajadas del monitor correspondiente.

Criterios de aceptación:

1. El sistema debe comprobar que sea un administrativo el que consulte esta información ya que son los únicos con autoridad para hacerlo.
2. En un futuro este reporte se integrará automáticamente con el sistema de nóminas por lo que el formato de salida debe de ser compatible con este sistema.

4.2 Casos de uso

4.2.1 Gestión de parques



"La figura 1 muestra el diagrama de casos de uso para la gestión de los parques"

4.2.1.1 CU1

Caso de uso: El administrativo quiere añadir un parque al sistema

Objetivo: Administrar los parques de forma que se pueda añadir información al sistema

Contexto: El parque no debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el identificador del parque
2. El sistema debe comprobar que el parque no exista(CU2)
3. El administrativo debe continuar rellenando el resto del formulario y el sistema debe almacenar la nueva información

Extensiones:

1. Si el parque existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.1.2 CU2

Caso de uso: Comprobar si un parque existe en el sistema

Objetivo: Evitar la repetición de la información en el sistema

Contexto: Debe de existir como mínimo un parque en el sistema

Actor principal: Sistema

Escenario principal:

1. Se introduce el identificador del parque cuya existencia queremos comprobar
2. El sistema debe comparar el identificador introducido con los identificadores que existen en el sistema
3. El sistema debe devolver TRUE si el parque se encuentra en el sistema y FALSE si no se encuentra

4.2.1.3 CU3

Caso de uso: El administrativo quiere modificar un parque del sistema

Objetivo: Administrar los parques de forma que se pueda actualizar la información del sistema

Contexto: El parque debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el identificador del parque
2. El sistema debe comprobar que el parque exista(CU2)
3. El administrativo debe continuar rellenando el resto del formulario y el sistema debe almacenar la nueva información

Extensiones:

1. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.1.4 CU4

Caso de uso: El administrativo quiere eliminar un parque del sistema

Objetivo: Administrar los parques de forma que se pueda eliminar información del sistema

Contexto: El parque debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el identificador del parque
2. El sistema debe comprobar que el parque exista(CU2)
3. El sistema, este debe eliminar la información del parque.

Extensiones:

1. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal
2. El sistema debe borrar la información de los senderos asociados a dicho parque(CU9)

4.2.1.5 CU5

Caso de uso: El administrativo quiere ver la información de un parque del sistema para comprobar la información almacenada sobre el mismo (superficie, premios...) y el estado en el que se encuentran sus senderos y rutas.

Objetivo: Consultar la información de un parque almacenado en el sistema

Contexto: El parque debe de existir en el sistema

Actor principal: Administrativo

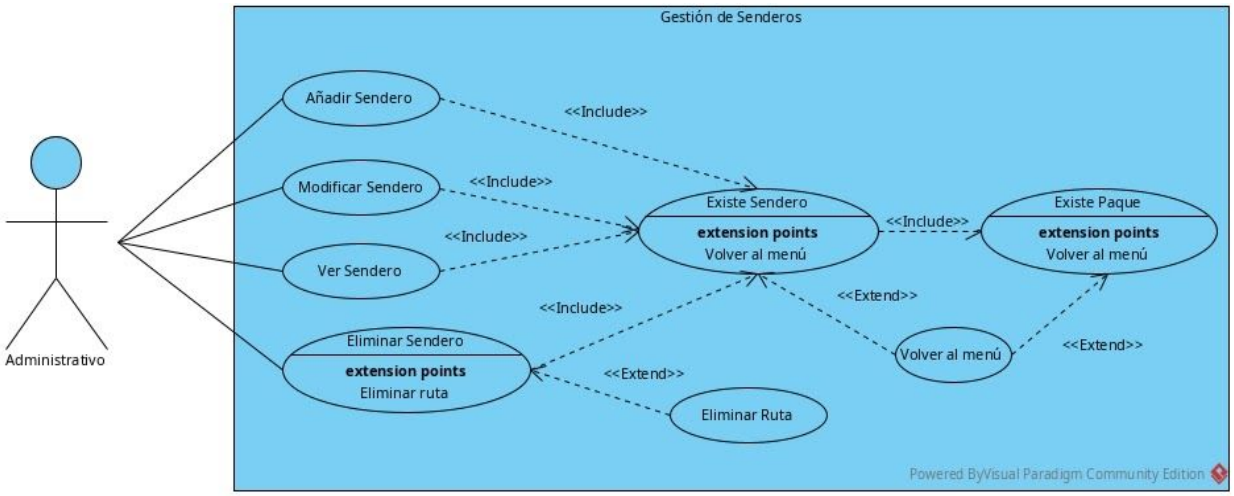
Escenario principal:

1. El administrativo debe introducir el identificador del parque
2. El sistema debe comprobar que el parque exista(CU2)
3. El sistema se imprime por pantalla la información

Extensiones:

1. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.2 Gestión de senderos



"La figura 2 muestra el diagrama de casos de uso para la gestión de los senderos"

4.2.2.1 CU6

Caso de uso: El administrativo quiere añadir un sendero a un parque

Objetivo: Administrar los senderos de forma que se puedan añadir senderos a los parques del sistema

Contexto: El sendero no debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el identificador del parque al que se desea añadir y el identificador del sendero
2. El sistema debe comprobar que el sendero no exista en ese parque (CU7)
3. El administrativo debe continuar rellenando el resto del formulario y el sistema debe almacenar la nueva información

Extensiones:

1. Si el sendero existe en el parque, la operación se cancela y vuelve al menú principal
2. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.2.2 CU7

Caso de uso: Comprobar si un sendero pertenece a un parque

Objetivo: Validar la integridad de la información del sistema

Contexto: Debe de existir como mínimo un parque y un sendero en el sistema

Actor principal: Sistema

Escenario principal:

1. Se introduce el identificador del sendero cuya existencia queremos comprobar y el identificador del parque al que pertenece
2. El sistema debe comprobar que el parque exista(CU2)
3. El sistema debe comparar el identificador del sendero introducido con los identificadores de senderos que existen en el parque
4. El sistema debe devolver TRUE si el sendero se encuentra en el parque y FALSE si no se encuentra

4.2.2.3 CU8

Caso de uso: El administrativo quiere modificar un sendero de un parque

Objetivo: Administrar los senderos de forma que se pueda actualizar la información del sistema

Contexto: El sendero debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el identificador del parque en el que se encuentra el sendero y el identificador del sendero que se desea modificar
2. El sistema debe comprobar que el sendero exista en ese parque (CU7)
3. El administrativo debe continuar rellenando el resto del formulario y el sistema debe almacenar la nueva información

Extensiones:

1. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
2. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.2.4 CU9

Caso de uso: Eliminar un sendero del sistema

Objetivo: Administrar los senderos de forma que se pueda eliminar información del sistema

Contexto: El sendero debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el identificador del parque en el que se encuentra el sendero y el identificador del sendero que se desea eliminar
2. El sistema debe comprobar que el sendero exista en ese parque (CU7)
3. Si el sendero existe en el parque el sistema debe eliminar la información del mismo

Extensiones:

1. El sistema debe borrar la información de las rutas asociadas a dicho sendero(CU14)
2. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
3. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.2.5 CU10

Caso de uso: El administrativo quiere ver la información de un sendero del sistema para comprobar la información almacenada sobre el mismo y el estado en el que se encuentra.

Objetivo: Consultar la información de un sendero almacenado en el sistema

Contexto: El sendero debe de existir en el sistema

Actor principal: Administrativo

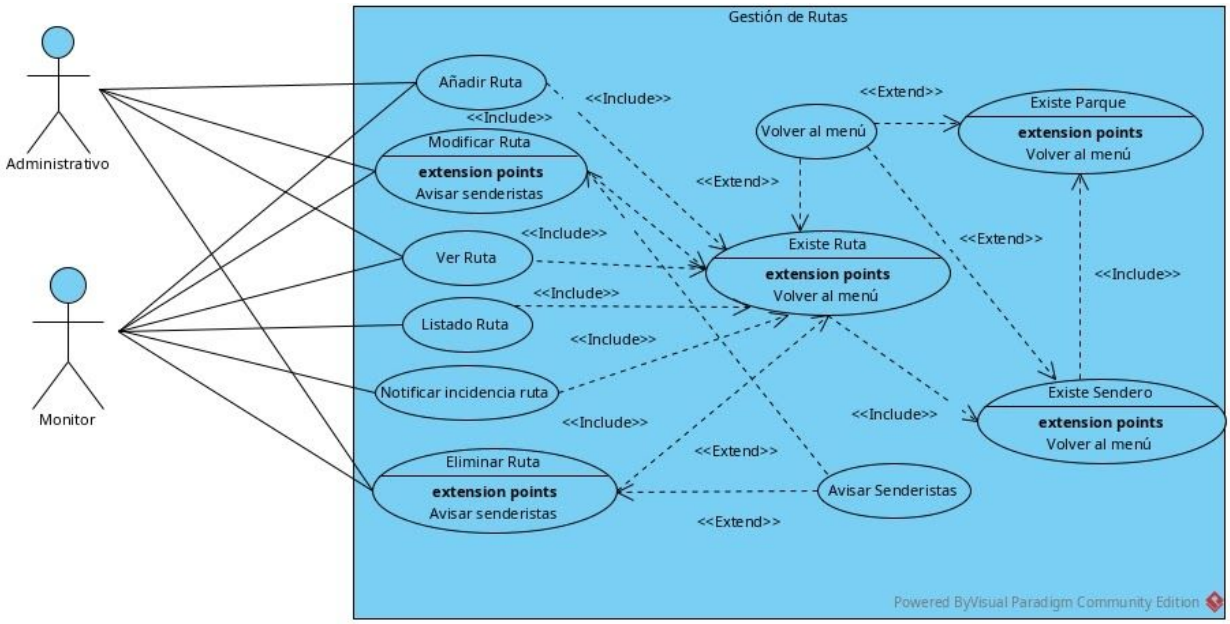
Escenario principal:

1. El administrativo debe introducir el identificador del parque en el que se encuentra el sendero y el identificador del sendero que se desea mostrar
2. El sistema debe comprobar que el sendero exista en ese parque (CU7)
3. Si el sendero existe en el parque el sistema debe mostrar la información del mismo

Extensiones

1. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
2. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.3 Gestión de rutas



"La figura 3 muestra el diagrama de casos de uso para la gestión de las rutas"

4.2.3.1 CU11

Caso de uso: El administrativo o el monitor quiere añadir una ruta

Objetivo: Administrar las rutas de forma que se puedan añadir rutas al sistema

Contexto: La ruta no debe de existir en el sistema

Actor principal: Administrativo o monitor

Escenario principal:

1. El administrativo o monitor debe introducir la fecha, la hora, el identificador del sendero y el identificador del parque al que pertenece la ruta
2. El sistema debe comprobar que no exista una ruta en la misma fecha a la misma hora en ese sendero de ese parque(CU12)
3. El administrativo o monitor debe continuar rellenando el resto del formulario y el sistema debe almacenar la nueva información

Extensiones

1. Si existe, la operación se cancela y vuelve al menú principal
2. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
3. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.3.2 CU12

Caso de uso: Comprobar si una ruta existe

Objetivo: Evitar la repetición de la información en el sistema

Contexto: Debe de existir como mínimo un parque, un sendero y una ruta en el sistema

Actor principal: Sistema

Escenario principal:

1. Se introduce el identificador del sendero y el identificador del parque al que pertenece la ruta
2. El sistema debe comprobar que el parque exista(CU2)
3. El sistema debe comparar el identificador del sendero introducido con los identificadores de senderos que existen en el parque
4. El sistema debe comprobar la fecha y la hora de las rutas asociadas a ese sendero
5. El sistema debe devolver TRUE si existe dicha ruta y false si no existe

Extensiones:

1. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
2. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.3.3 CU13

Caso de uso: El administrativo o el monitor quiere modificar una ruta

Objetivo: Administrar las rutas de forma que se pueda actualizar la información del sistema

Contexto: La ruta debe de existir en el sistema

Actor principal: Administrativo o monitor

Escenario principal:

1. El administrativo o monitor debe introducir la fecha, la hora, el identificador del sendero y el identificador del parque al que pertenece la ruta
2. El sistema debe comprobar que exista una ruta en la misma fecha a la misma hora en ese sendero de ese parque(CU12)
3. El administrativo o monitor debe continuar rellenando el resto del formulario y el sistema debe almacenar la nueva información

Extensiones:

1. El sistema debe recordarle al administrador o monitor avisar a los excursionistas
2. apuntados a dicha ruta(CU15)
3. Si no existe, la operación se cancela y vuelve al menú principal
4. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
5. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.3.4 CU14

Caso de uso: Eliminar una ruta del sistema

Objetivo: Administrar las rutas de forma que se pueda eliminar información del sistema

Contexto: La ruta debe de existir en el sistema

Actor principal: Administrativo o monitor

Escenario principal:

1. El administrativo o monitor debe introducir la fecha, la hora, el identificador del sendero y el identificador del parque al que pertenece la ruta
2. El sistema debe comprobar que exista una ruta en la misma fecha a la misma hora en ese sendero de ese parque(CU12)
3. El sistema elimina la información de dicha ruta del parque y la eliminará de la lista de rutas del monitor y de la listas de rutas de los senderistas que estuviesen apuntados.

Extensiones:

1. El sistema debe recordarle al administrador o monitor avisar a los excursionistas apuntados a dicha ruta(CU15)
2. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal
3. Si la ruta no existe, la operación se cancela y vuelve al menú principal
4. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal

4.2.3.5 CU15

Caso de uso: El sistema debe mostrarle al administrativo o monitor una lista con los datos de contacto de los senderistas que estaban apuntados a la ruta que se ha eliminado

Objetivo: Informar a los senderistas de la cancelación de una ruta

Contexto: El senderista debe existir en el sistema

Actor principal: Sistema

Escenario principal:

1. El sistema debe mostrar los datos de contacto los senderistas que estaban apuntados a dicha actividad

4.2.3.6 CU16

Caso de uso: El administrativo o el monitor quieren poder ver la información de una ruta del sistema para preparar la actividad y adaptarla a los requisitos de los senderistas.

Objetivo: Consultar la información de una ruta almacenada en el sistema para poder preparar las rutas adecuadamente

Contexto: La ruta debe de existir en el sistema

Actor principal: Administrativo o monitor

Escenario principal:

1. El administrativo o monitor debe introducir la fecha, la hora, el identificador del
2. sendero y el identificador del parque al que pertenece la ruta
3. El sistema debe comprobar que exista una ruta en la misma fecha a la misma
4. hora en ese sendero de ese parque(CU12)
5. El sistema muestra la información de dicha ruta del sistema.

Extensiones:

1. Si no existe, la operación se cancela y vuelve al menú principal
2. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
3. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.3.7 CU30

Caso de uso: El monitor quiere un listado con la información de los senderistas que participan en la ruta para tener información sobre el grupo que va a realizar la actividad

Objetivo: Preparar el material y adaptar la ruta a las necesidades de los senderistas

Contexto: La ruta debe existir en el sistema

Actor principal: Monitor

Escenario principal:

1. El monitor debe introducir la fecha, la hora, el identificador del sendero y el
2. identificador del parque al que pertenece la ruta
3. El sistema debe comprobar que exista una ruta en la misma fecha a la misma
4. hora en ese sendero de ese parque(CU12)
5. El sistema debe generar un documento que incluya la información de la ruta y la de los senderistas que van a participar en ella

Extensiones:

1. Si no existe, la operación se cancela y vuelve al menú principal
2. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
3. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.3.8 CU31

Caso de uso: El monitor quiere poder notificar a los administrativos de lo ocurrido durante la ruta

Objetivo: Dar parte a los administrativos de lo ocurrido

Contexto: La ruta debe existir en el sistema

Actor principal: Monitor

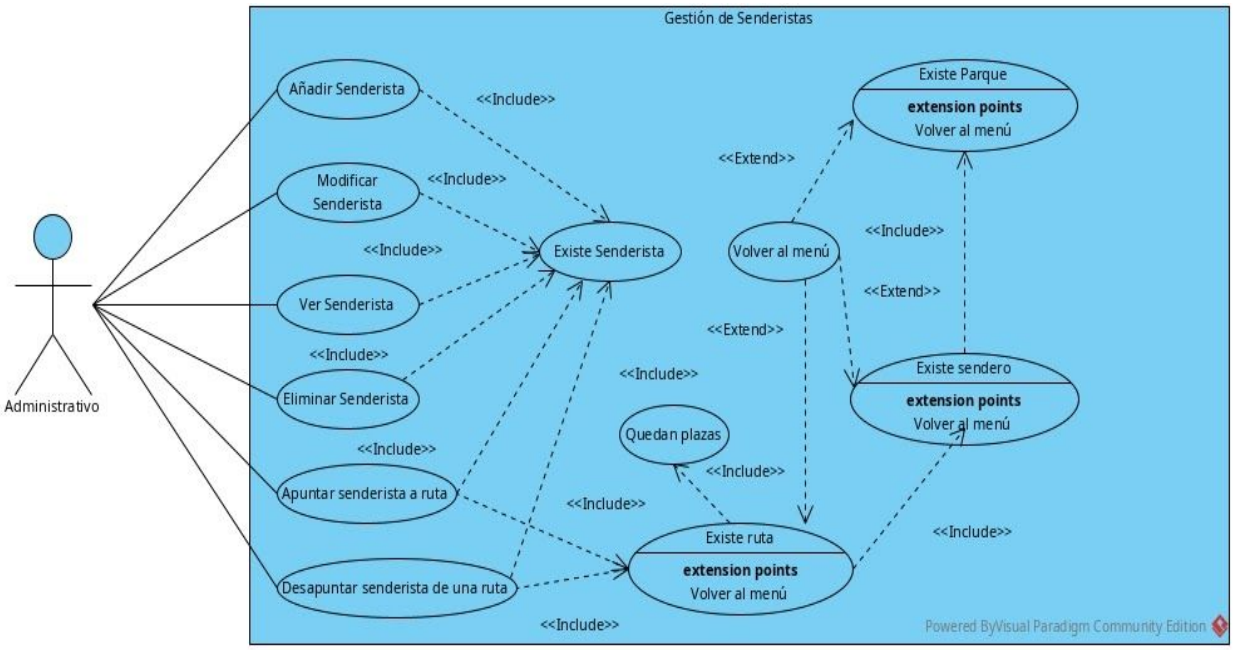
Escenario principal:

1. El monitor debe introducir la fecha, la hora, el identificador del sendero y el
2. identificador del parque al que pertenece la ruta
3. El sistema debe comprobar que exista una ruta en la misma fecha a la misma
4. hora en ese sendero de ese parque(CU12)
5. El sistema debe crear un parte de incidencias que el monitor deberá
6. rellenar y que el sistema hará llegar a los administrativos

Extensiones:

1. Si no existe, la operación se cancela y vuelve al menú principal
2. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
3. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal.

4.2.4 Gestión de senderistas



"La figura 4 muestra el diagrama de casos de uso para la gestión de senderistas"

4.2.4.1 CU17

Caso de uso: El administrativo quiere dar de alta un senderista en el sistema

Objetivo: Administrar los senderistas de forma que se pueda añadir información al sistema

Contexto: El senderista no debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el DNI del senderista
2. El sistema debe comprobar que el senderista no exista(CU18)
3. El administrativo debe continuar rellenando el resto del formulario y el sistema debe almacenar la nueva información

Extensiones:

1. Si existe la operación se cancela y se vuelve al menú principal

4.2.4.2 CU18

Caso de uso: Comprobar si un senderista está en el sistema

Objetivo: Evitar la repetición de la información en el sistema

Contexto: Debe de existir como mínimo un senderista en el sistema

Actor principal: Sistema

Escenario principal:

1. Se introduce el DNI del senderista cuya existencia queremos comprobar
2. El sistema debe comparar la información introducida con la información disponible
3. El sistema debe devolver TRUE si el senderista se encuentra en el sistema y FALSE si no se encuentra

4.2.4.3 CU19

Caso de uso: El administrativo quiere modificar la información de un senderista del sistema

Objetivo: Corregir la información de los senderistas en caso de que se hayan equivocado al introducirla o haya cambiado algo. Ej: su teléfono de contacto.

Contexto: El senderista debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

El administrativo debe introducir el DNI del senderista
El sistema debe comprobar que el senderista exista(CU18)
El administrativo debe continuar rellenando el resto del formulario y el sistema debe almacenar la nueva información

Extensiones:

1. Si no existe la operación se cancela y se vuelve al menú principal

4.2.4.4 CU20

Caso de uso: El administrativo quiere eliminar un senderista del sistema

Objetivo: Administrar los senderistas de forma que se pueda eliminar información del sistema

Contexto: El senderista debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el DNI del senderista
2. El sistema debe comprobar que el senderista exista(CU18)
3. El sistema debe eliminar la información del senderista y borrarlo de las rutas en las que este participase

Extensiones:

1. Si no existe el senderista la operación se cancela y se vuelve al menú principal

4.2.4.5 CU21

Caso de uso: El administrativo quiere ver la información de un senderista del sistema

Objetivo: Poder ver la información que el sistema almacena del senderista por si hay que contactar con el.

Contexto: El senderista debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el DNI del senderista
2. El sistema debe comprobar que el senderista exista(CU18)
3. El sistema debe mostrar la información del senderista

Extensiones:

1. Si no existe la operación se cancela y se vuelve al menú principal

4.2.4.6 CU22

Caso de uso: El administrativo quiere apuntar un senderista a una ruta

Objetivo: Añadir un nuevo participante a una ruta

Contexto: El senderista debe de existir en el sistema y deben quedar plazas en la ruta

Actor principal: Administrativo

Escenario principal:

1. El administrativo o monitor debe introducir la fecha, la hora, el identificador del sendero y el identificador del parque al que pertenece la ruta
2. El sistema debe comprobar que exista una ruta en la misma fecha a la misma hora en ese sendero de ese parque(CU12)
3. El sistema debe comprobar si quedan plazas disponibles(CU32)
4. El administrativo debe introducir el DNI del senderista
5. El sistema debe comprobar que el senderista exista(CU18)
6. El administrativo debe continuar rellenando el formulario de inscripción y el sistema debe almacenar la nueva información además de apuntar el senderista a la ruta

Extensiones:

1. Si existe el senderista se añade a la ruta
2. Si no quedan plazas la operación se cancela y vuelve al menú principal
3. Si no existe, la operación se cancela y vuelve al menú principal
4. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
5. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.4.7 CU23

Caso de uso: El administrativo quiere eliminar un senderista de una ruta

Objetivo: Dar de baja un participante de una ruta ya que este no va a poder asistir y dejando su plaza libre para otro senderista

Contexto: El senderista debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo o monitor debe introducir la fecha, la hora, el identificador del sendero y el identificador del parque al que pertenece la ruta
2. El sistema debe comprobar que exista una ruta en la misma fecha a la misma hora en ese sendero de ese parque(CU12)
3. El administrativo debe introducir el DNI del senderista
4. El sistema debe comprobar que el senderista exista(CU18)
5. El sistema debe borrarlo de la lista de senderistas de esa ruta

Extensiones:

1. Si no existe el senderista la operación se cancela y vuelve al menú principal
2. Si no existe, la operación se cancela y vuelve al menú principal
3. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
4. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.4.8 CU32

Caso de uso: Comprobar si quedan plazas en la ruta

Objetivo: Gestionar el aforo de las rutas

Contexto: La ruta debe existir en el sistema

Actor principal: Sistema

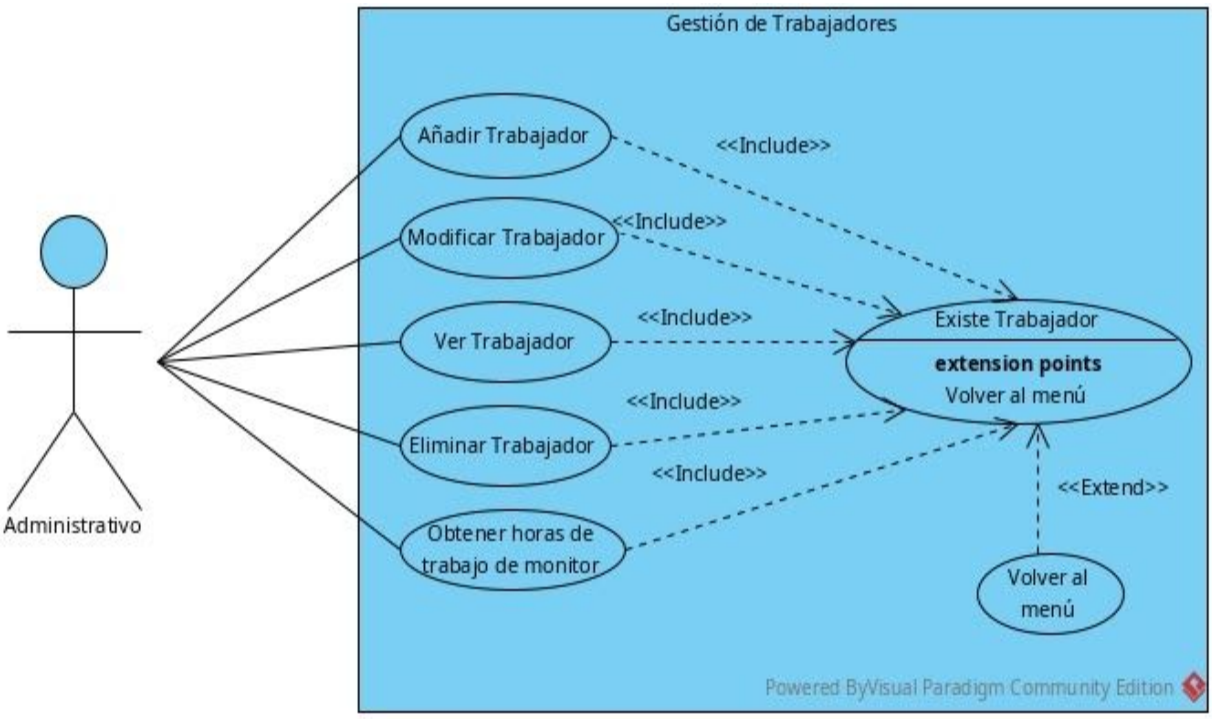
Escenario principal:

1. Se introduce la fecha, la hora, el identificador del sendero y el identificador del
2. parque al que pertenece la ruta
3. El sistema debe comprobar que exista una ruta en la misma fecha a la misma
4. hora en ese sendero de ese parque(CU12)
5. El sistema devolverá TRUE si hay al menos una plaza libre o FALSE si no
6. la hay

Extensiones:

1. Si no existe, la operación se cancela y vuelve al menú principal
2. Si el sendero no existe en el parque, la operación se cancela y vuelve al menú principal
3. Si el parque no existe en el sistema, la operación se cancela y vuelve al menú principal

4.2.5 Gestión de trabajadores



"La figura 5 muestra el diagrama de casos de uso para la gestión de trabajadores"

4.2.5.1 CU24

Caso de uso: El administrativo quiere añadir un trabajador al sistema

Objetivo: Dar de alta un nuevo trabajador del sistema

Contexto: El trabajador no debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el DNI del trabajador
2. El sistema debe comprobar que el trabajador no exista(CU25)
3. El administrativo debe continuar rellenando el resto del formulario y el sistema debe almacenar la nueva información

Extensiones:

1. Si existe la operación se cancela y se vuelve al menú principal

4.2.5.2 CU25

Caso de uso: Comprobar si un trabajador está en el sistema

Objetivo: Evitar la repetición de la información en el sistema

Contexto: Debe de existir como mínimo un trabajador en el sistema

Actor principal: Sistema

Escenario principal:

1. Se introduce el DNI del trabajador cuya existencia queremos comprobar
2. El sistema debe comparar la información introducida con la información disponible
3. El sistema debe devolver TRUE si el trabajador se encuentra en el sistema y FALSE si no se encuentra

4.2.5.3 CU26

Caso de uso: El administrativo quiere modificar la información de un trabajador del sistema

Objetivo: Editar y corregir la información que el sistema almacena sobre los trabajadores para poder corregirla en caso de error

Contexto: El trabajador debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el DNI del trabajador
2. El sistema debe comprobar que el trabajador exista(CU25)
3. El administrativo debe continuar rellenando el resto del formulario y el sistema debe almacenar la nueva información

Extensiones:

1. Si no existe la operación se cancela y se vuelve al menú principal

4.2.5.4 CU27

Caso de uso: El administrativo quiere eliminar un trabajador del sistema

Objetivo: Eliminar del sistema un extrabajador del la empresa

Contexto: El trabajador debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el DNI del trabajador
2. El sistema debe comprobar que el trabajador exista(CU25)
3. El sistema debe eliminar la información del trabajador y si es un monitor, borrarlo de todas las rutas en las que sea el encargado.

Extensiones:

1. Si no existe la operación se cancela y se vuelve al menú principal

4.2.5.5 CU28

Caso de uso: El administrativo quiere ver la información de un trabajador del sistema para comprobar la información almacenada sobre el mismo y sus datos de contacto.

Objetivo: Consultar la información de un trabajador almacenada en el sistema

Contexto: El trabajador debe de existir en el sistema

Actor principal: Administrativo

Escenario principal:

1. El administrativo debe introducir el DNI del trabajador
2. El sistema debe comprobar que el trabajador exista(CU25)
3. El sistema debe mostrar su información

Extensiones:

1. Si no existe la operación se cancela y se vuelve al menú principal

4.2.5.6 CU29

Caso de uso: El administrativo quiere obtener el cómputo de las horas de trabajo de los monitores

Objetivo: Administrar los monitores de tal forma que se puedan visualizar sus horas de trabajo en el sistema

Contexto: El monitor debe de existir en el sistema

Actor principal: Administrativo

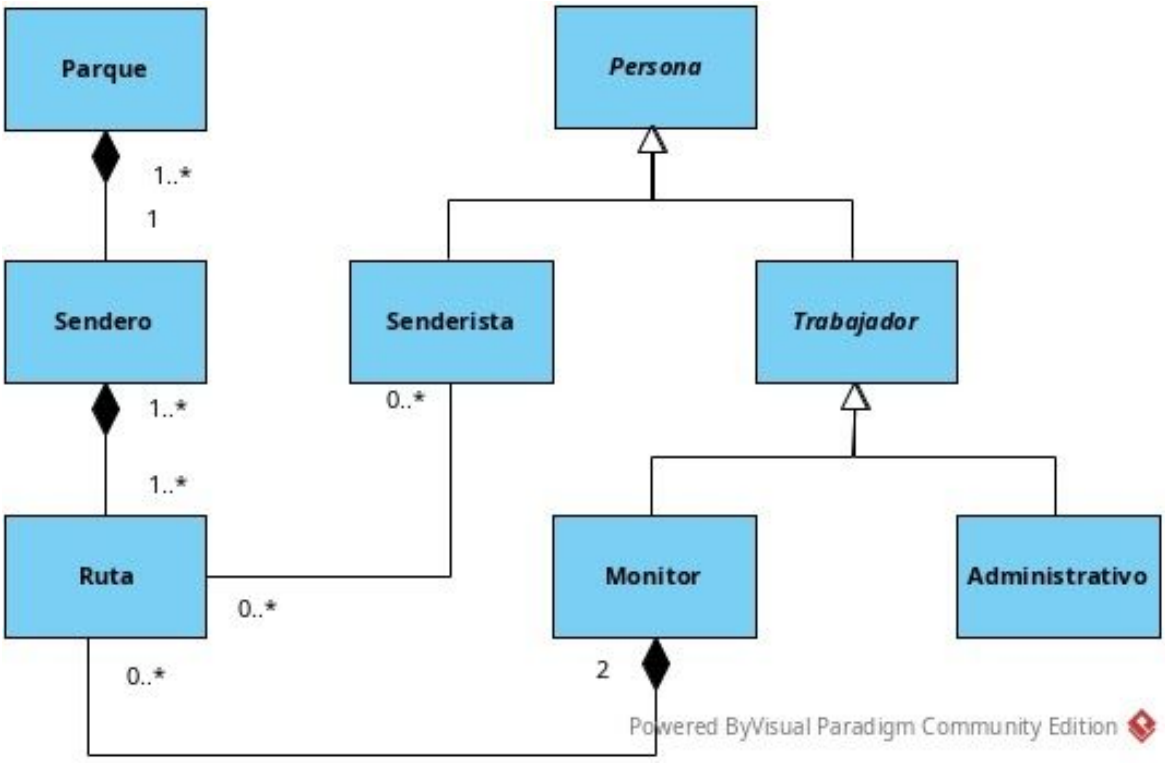
Escenario principal:

1. El sistema debe mostrar un listado con las horas trabajadas de cada monitor y su DNI

5. Diseño del sistema (P3)

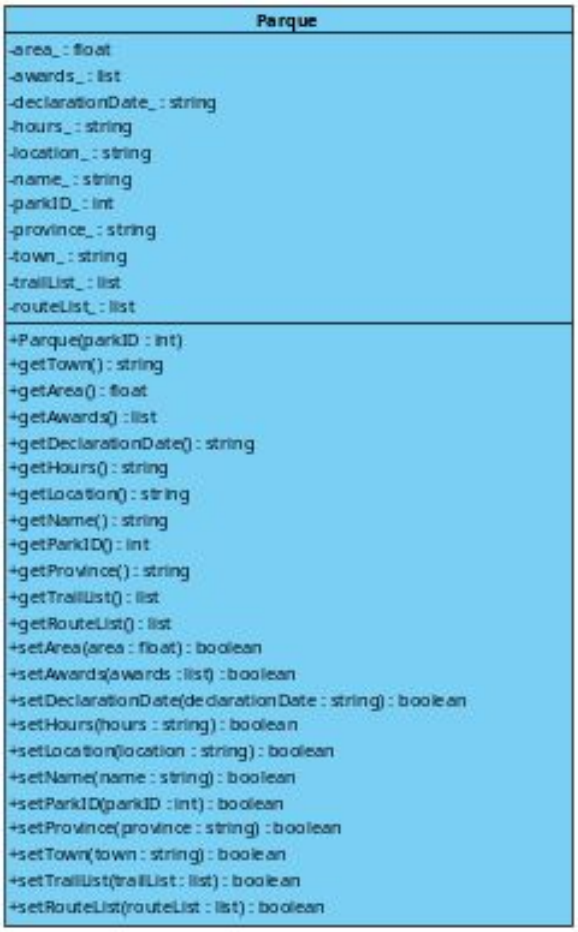
5.1 Diseño estructural

5.1.1 Diagrama de clases

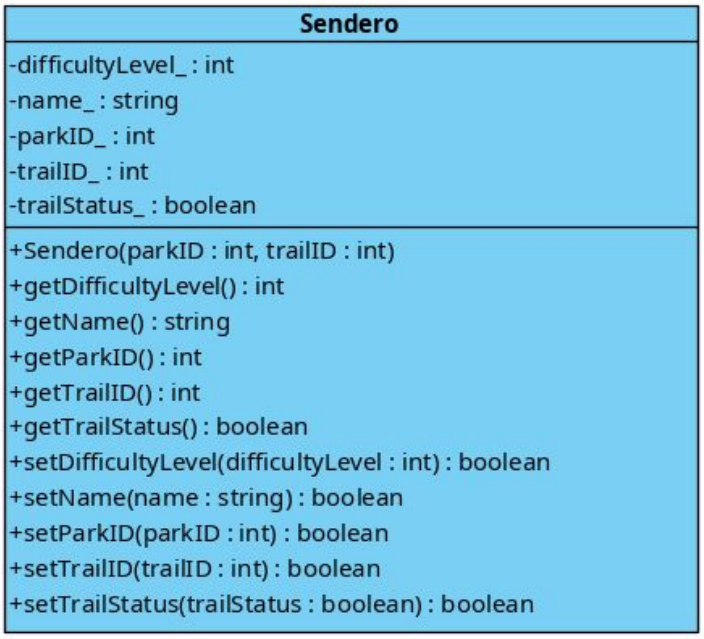


"La imagen 1 muestra el diagrama de clases con las correspondientes relaciones"

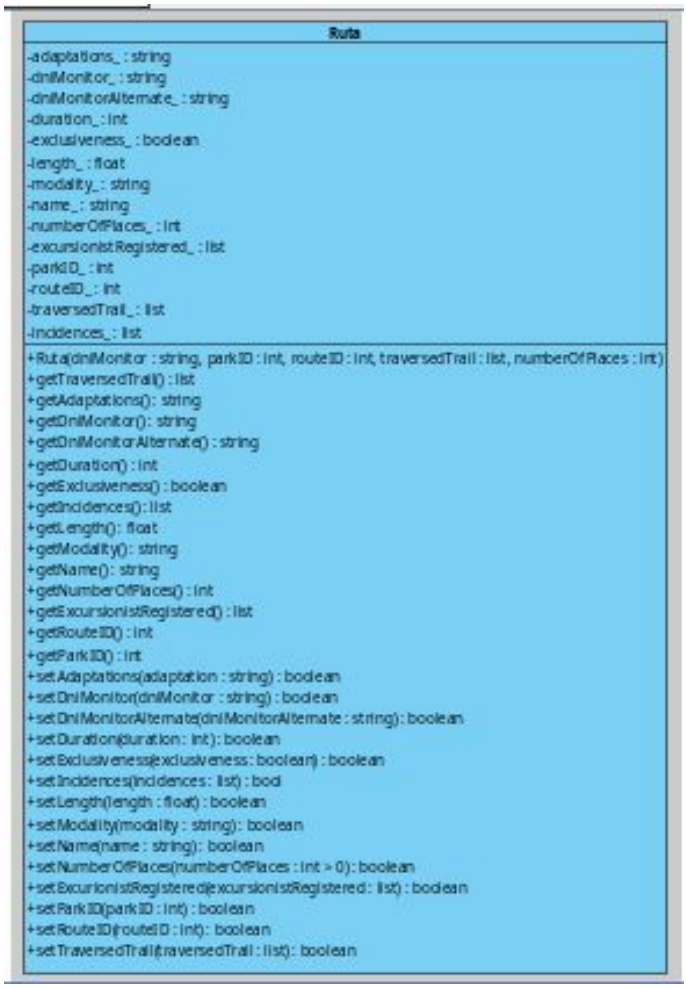
La imagen anterior muestra el diagrama de clases simplemente con las relaciones establecidas entre clases, ya que si incluimos en la misma imagen todos los atributos y operaciones de las clases, quedaría una imagen difícil de interpretar. A continuación se muestran las distintas clases con sus atributos y operaciones.



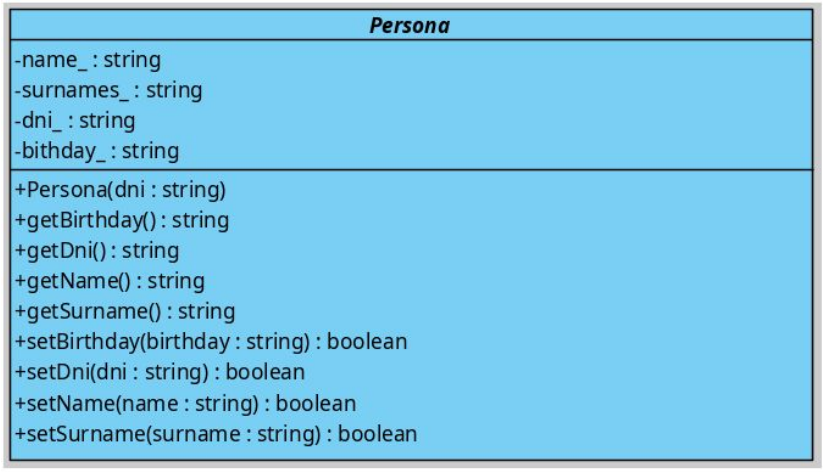
"La imagen 2 muestra la clase parque con sus atributos y operaciones"



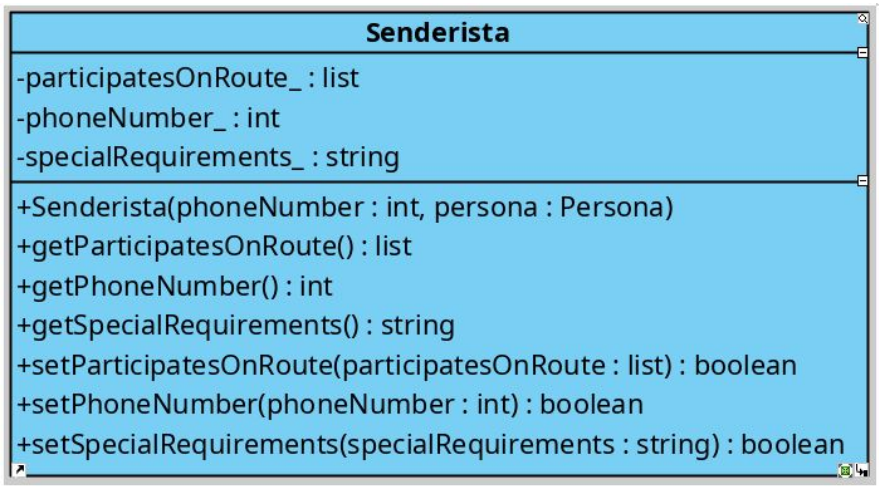
"La imagen 3 muestra la clase sendero con sus atributos y operaciones"



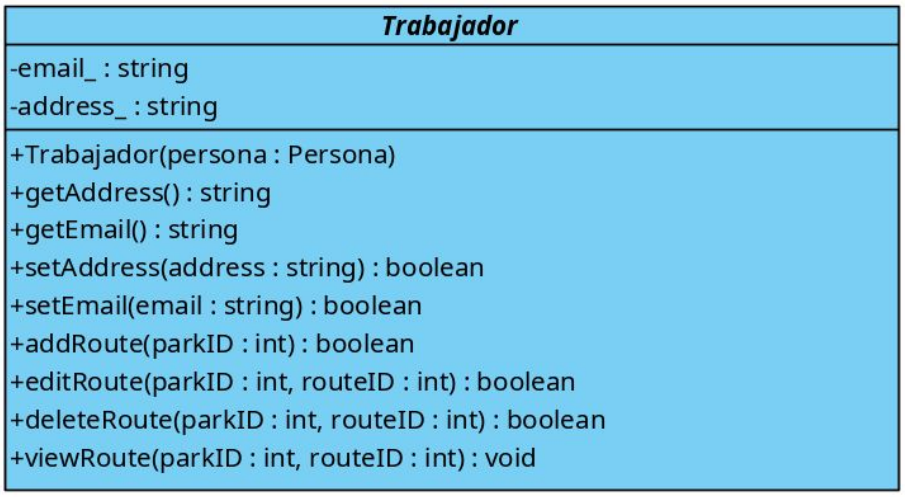
"La imagen 4 muestra la clase ruta con sus atributos y operaciones"



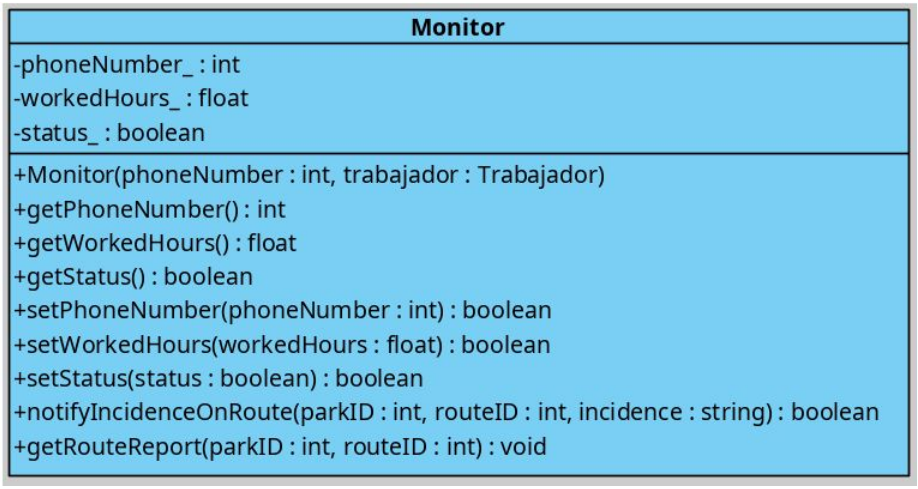
"La imagen 5 muestra la clase persona con sus atributos y operaciones"



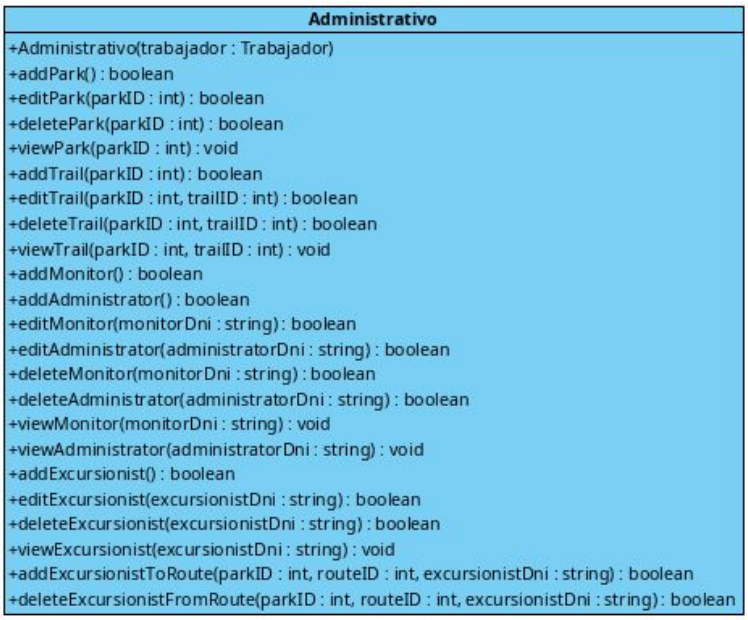
"La imagen 6 muestra la clase senderista con sus atributos y operaciones"



"La imagen 7 muestra la clase trabajador con sus atributos y operaciones"



"La imagen 8 muestra la clase monitor con sus atributos y operaciones"



"La imagen 9 muestra la clase administrativo con sus atributos y operaciones"

5.1.2 Especificación de clases

Se muestran a continuación la información de diseño en forma de tabla para que quede aún más clara la visualización del diagrama de clases.

Clase	**Parque**		
La clase parque representa a un parque con sus correspondientes atributos y operaciones			
Atributos			
-	area_	float	Área del parque
-	awards_	list	Lista de los premios del parque
-	declarationDate_	string	Fecha de declaración del parque
-	hours_	string	Horario del parque
-	location_	string	Localización (latitud y longitud)
-	name_	string	Nombre del parque
-	parkID_	int	ID del parque
-	province_	string	Provincia del parque
-	town_	string	Ciudad del parque
-	trailList_	list	Lista de los senderos del parque
-	routeList_	list	Lista de las rutas del parque

Clase	Parque		
La clase parque representa a un parque con sus correspondientes atributos y operaciones			
Operaciones			
+	Parque(int parkID)	Constructor	Constructor de la clase parque. Recibe el ID del parque creado
+	getArea()	float	Devuelve el área del parque
+	getAwards()	list	Devuelve el listado de premios
+	getDeclarationDate()	string	Devuelve la fecha de declaración
+	getHours()	string	Devuelve el horario
+	getLocation()	string	Devuelve la localización
+	getName()	string	Devuelve el nombre
+	getParkID()	int	Devuelve el ID del parque
+	getProvince()	string	Devuelve la provincia
+	getTown()	string	Devuelve la ciudad
+	getTrailList()	list	Devuelve la lista de senderos del parque
+	getRoutesList()	list	Devuelve la lista de rutas del parque
+	setArea(float area)	bool	Asigna un área
+	setAwards(list awards)	bool	Asigna la lista de premios
+	setDeclarationDate(string date)	bool	Asigna la fecha de declaración

Clase	Parque		
La clase parque representa a un parque con sus correspondientes atributos y operaciones			
Operaciones			
+	setHours(string hours)	bool	Asigna el horario
+	setLocation(string location)	bool	Asigna la localización
+	setName(string name)	bool	Asigna el nombre
+	setParkID(int parkID)	bool	Asigna un ID
+	setProvince(string province)	bool	Asigna la provincia
+	setTown(string town)	bool	Asigna la ciudad
+	setTrailList(list trailList)	bool	Asigna los senderos del parque
+	setRouteList(list routeList)	bool	Asigna las rutas del parque

"La tabla 1 muestra la clase parque con sus atributos y operaciones"

Clase	Sendero		
La clase sendero representa a un sendero con sus correspondientes atributos y operaciones			
Atributos			
-	difficultyLevel_	int	Nivel de dificultad del sendero (del 1 al 10)
-	name_	string	Nombre del sendero
-	parkID_	int	ID del parque del sendero
-	trailID_	int	ID del sendero
-	trailStatus_	bool	Estado del sendero (disponible o deshabilitado)
Operaciones			
+	Sendero(int parkID, int trailID)	Constructor	Constructor de la clase sendero. Recibe el ID de su parque y el ID del sendero
+	getDifficultyLevel()	int	Devuelve el nivel de dificultad del sendero
+	getName()	string	Devuelve el nombre del sendero
+	getParkID	int	Devuelve el ID del parque del sendero
+	getTrailID()	int	Devuelve el ID del sendero
+	getTrailStatus()	bool	Devuelve el estado del sendero
+	setDifficultyLevel(int difficultyLevel)	bool	Asigna un nivel de dificultad al sendero

Clase	Sendero		
La clase sendero representa a un sendero con sus correspondientes atributos y operaciones			
Operaciones			
+	setName(string name)	bool	Asigna un nombre al sendero
+	setParkID(int parkID)	bool	Asigna el ID del parque al que pertenece el sendero
+	setTrailID(int trailID)	bool	Asigna un ID al sendero
+	setTrailStatus(bool trailStatus)	bool	Asigna un estado al sendero

"La tabla 2 muestra la clase sendero con sus atributos y operaciones"

Clase	Ruta		
La clase ruta representa a una ruta con sus correspondientes atributos y operaciones			
Atributos			
-	adaptations_	string	Adaptaciones de la ruta (por si algún senderista tiene requisitos especiales)
-	dniMonitor_	string	Dni del monitor de la ruta
-	dniMonitorAlternate_	string	Dni del monitor alternativo de la ruta
-	duration_	int	Duración de las rutas en minutos
-	exclusiveness_	bool	Almacena si una ruta es exclusiva o no (solo para colegios por ejemplo)
-	length_	float	Longitud de la ruta en kilómetros
-	modality_	string	Modalidad de la ruta (a pie, en bici...)
-	name_	string	Nombre de la ruta
-	numberOfPlaces_	int	Número de plazas de la ruta
-	excursionistRegistered_	list	Lista de senderistas apuntados a la ruta
-	parkID_	int	ID del parque de la ruta
-	routeID_	int	ID de la ruta
-	traversedTrail_	list	Lista de los senderos atravesados por la ruta
-	incidences_	list	Lista de las incidencias

Clase	Ruta		
La clase ruta representa a una ruta con sus correspondientes atributos y operaciones			
Operaciones			
+	Ruta(string dniMonitor, int parkID, int routeID, list traversedTrail, int numberOfPlaces)	Constructor	Constructor de la clase ruta. Recibe el dni del monitor, el ID de su parque, el ID de la ruta, la lista de senderos atravesados por la ruta y el número de plazas
+	getAdaptations()	string	Devuelve el número de adaptaciones de la ruta
+	getDniMonitor()	string	Devuelve el dni del monitor de la ruta
+	getDniMonitorAlternate()	string	Devuelve el dni del monitor alternativo de la ruta
+	getDuration()	int	Devuelve la duración de la ruta
+	getExclusiveness()	bool	Devuelve si la ruta es exclusiva
+	getIncidences()	list	Devuelve la lista de incidencias de la ruta
+	getLength()	float	Devuelve la longitud de la ruta
+	getModality()	string	Devuelve la modalidad de la ruta
+	getName()	string	Devuelve el nombre de la ruta
+	getNumberOfPlaces()	int	Devuelve el número de plazas de la ruta
+	getExcursionistRegistered()	list	Devuelve la lista de senderistas apuntados a la ruta

Clase	Ruta		
La clase ruta representa a una ruta con sus correspondientes atributos y operaciones			
Operaciones			
+	getParkID()	int	Devuelve el ID del parque de la ruta
+	getRouteID()	int	Devuelve el ID de la ruta
+	getTraversedTrail()	int	Devuelve el número de senderos atravesados por la ruta
+	setAdaptations(string adapts)	bool	Asigna la adaptación de una ruta
+	setDniMonitor(string dni)	bool	Asigna el dni del monitor
+	setDniMonitorAlternate(string dni)	bool	Asigna el dni del monitor alternativo
+	setDuration(int duration)	bool	Asigna la duración de la ruta
+	setExclusiveness(bool exclnss)	bool	Asigna la exclusividad a la ruta
+	setIncidences(list incidences)	bool	Asigna incidencias a la ruta
+	setLength(float length)	bool	Asigna la longitud de la ruta
+	setModality(string modality)	bool	Asigna la modalidad de la ruta
+	setName(string name)	bool	Asigna un nombre a la ruta

Clase	Ruta		
La clase ruta representa a una ruta con sus correspondientes atributos y operaciones			
Operaciones			
+	setNumberOfPlaces(int n)	bool	Asigna el número de plazas de la ruta
+	setExcursionistRegistered(list excursionist)	bool	Asigna la lista de senderistas apuntados a una ruta
+	setParkID(int parkID)	bool	Asigna el ID del parque al que pertenece la ruta
+	setRouteID(int routeID)	bool	Asigna el ID de la ruta
+	setTraversedTrails(list traversedTrails)	bool	Asigna la lista de senderos que atraviesa la ruta

"La tabla 3 muestra la clase ruta con sus atributos y operaciones"

Clase	Persona		
La clase persona representa a una persona con sus correspondientes atributos y operaciones			
Atributos			
-	name_	string	Nombre de la persona
-	surnames_	string	Apellidos de la persona
-	dni_	string	Dni de la persona
-	birthday_	string	Fecha de nacimiento de la persona
Operaciones			
+	Persona(string dni)	Constructor	Constructor de la clase Persona. Recibe un dni
+	getBirthday()	string	Devuelve la fecha de nacimiento
+	getDni()	string	Devuelve el dni de la persona
+	getName()	string	Devuelve el nombre de la persona
+	getSurname()	string	Devuelve los apellidos de la persona
+	setBirthday(string birthday)	bool	Asigna la fecha de nacimiento
+	setDni(string dni)	bool	Asigna el dni de la persona
+	setName(string name)	bool	Asigna el nombre de la persona
+	setSurname(string name)	bool	Asigna los apellidos de la persona

"La tabla 4 muestra la clase persona con sus atributos y operaciones"

Clase	Senderista		
La clase senderista representa a un senderista con sus correspondientes atributos y operaciones			
Atributos			
-	participatesOnRoute_	list	Lista de rutas a las que está apuntado el senderista
-	phoneNumber_	int	Número de teléfono del senderista
-	specialRequirements_	string	Requisitos especiales del senderista
Operaciones			
+	Senderista(int number, Persona persona)	Constructor	Constructor de la clase Senderista. Recibe un teléfono y un objeto de la clase Persona
+	getParticipatesOnRoute()	list	Devuelve la lista de rutas a las que está apuntado el senderista
+	getPhoneNumber()	int	Devuelve el número de teléfono del senderista
+	getSpecialRequirements()	string	Devuelve los requisitos especiales del senderista
+	setParticipatesOnRoute(list participatesOnRoute)	bool	Asigna la lista de rutas a las que está apuntado el senderista
+	setPhoneNumber(int number)	bool	Asigna el número de teléfono del senderista
+	setSpecialRequirements(string specialRequirements)	bool	Asigna los requisitos especiales del senderista

"La tabla 5 muestra la clase senderista con sus atributos y operaciones"

Clase	Trabajador		
La clase trabajador representa a un trabajador con sus correspondientes atributos y operaciones			
Atributos			
-	email_	string	Email del trabajador
-	address_	string	Dirección
Operaciones			
+	Trabajador(Persona persona)	Constructor	Constructor de la clase trabajador. Recibe un objeto del tipo persona
+	getAddress()	string	Devuelve la dirección del trabajador
+	getEmail()	string	Devuelve el email del trabajador
+	setAddress(string address)	bool	Asigna la dirección
+	setEmail(string email)	bool	Asigna el email
+	addRoute(int parkID)	bool	Añade una ruta. Recibe el ID del parque donde se encuentre
+	editRoute(int parkID, int routeID)	bool	Modifica una ruta. Recibe el ID del parque donde se encuentre y el ID de la ruta a modificar
+	deleteRoute(int parkID, int routeID)	bool	Elimina una ruta. Recibe el ID del parque donde se encuentre y el ID de la ruta a eliminar
+	viewRoute(int parkID, int routeID)	void	Visualiza una ruta. Recibe el ID del parque donde se encuentre y el ID de la ruta a visualizar

"La tabla 6 muestra la clase trabajador con sus atributos y operaciones"

Clase	Monitor		
La clase monitor representa a un monitor con sus correspondientes atributos y operaciones			
Atributos			
-	phoneNumber_	int	Número de teléfono del monitor
-	workedHours_	float	Horas trabajadas del monitor
-	status_	bool	Almacena si el monitor está disponible o está de baja
Operaciones			
+	Monitor(int phoneNumber, Trabajador trabajador)	Constructor	Constructor de la clase monitor. Recibe un número de teléfono y un objeto de la clase Trabajador
+	getPhoneNumber()	int	Devuelve el número de teléfono del monitor
+	getWorkedHours()	float	Devuelve el número de horas trabajadas del monitor
+	getStatus()	bool	Devuelve el status del monitor
+	setPhoneNumber(int number)	bool	Asigna el número de teléfono del monitor
+	setWorkedHours(float hours)	bool	Asigna el número de horas trabajadas del monitor
+	setStatus(bool status)	bool	Asigna el status del monitor

Clase	Monitor		
La clase monitor representa a un monitor con sus correspondientes atributos y operaciones			
Operaciones			
+	notifyIncidentOnRoute(int parkID, int routeID, string incidence)	bool	Guarda en incidencias una incidencia ocurrida en la ruta
+	getRouteReport(int parkID, int routeID)	void	Muestra por pantalla la información de la ruta

"La tabla 7 muestra la clase monitor con sus atributos y operaciones"

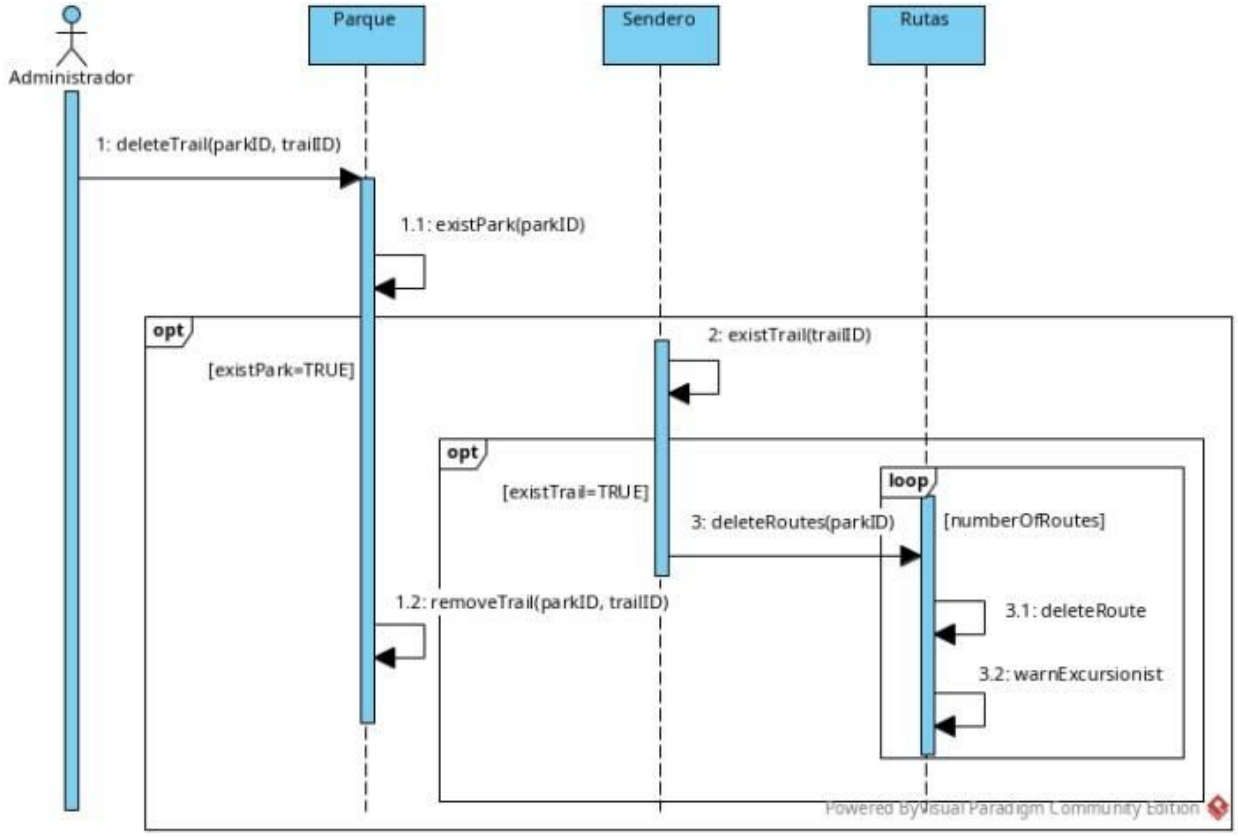
Clase	Administrativo		
La clase administrativo representa a un administrativo con sus correspondientes atributos y operaciones			
Operaciones			
+	Administrativo(Trabajador t)	Constructor	Constructor de la clase administrativo. Recibe un objeto t de la clase Trabajador
+	addPark()	bool	Añade un parque
+	editPark(int parkID)	bool	Modifica un parque
+	deletePark(int parkID)	bool	Elimina un parque
+	viewPark(int parkID)	void	Visualiza un parque
+	addTrail(int parkID)	bool	Añade un sendero
+	editTrail(int parkID, int trailID)	bool	Modifica un sendero
+	deleteTrail(int parkID, int trailID)	bool	Elimina un sendero
+	viewTrail(int parkID, int trailID)	void	Visualiza un sendero
+	addMonitor()	bool	Añade un monitor
+	addAdministrator()	bool	Añade un administrativo
+	editMonitor(string dni)	bool	Modifica un monitor
+	editAdministrator(string dni)	bool	Modifica un administrativo
+	deleteMonitor(string dni)	bool	Elimina un monitor
+	deleteAdministrator(string dni)	bool	Elimina un administrativo
+	viewMonitor(string dni)	void	Visualiza un monitor

Clase	Administrativo		
La clase administrativo representa a un administrativo con sus correspondientes atributos y operaciones			
Operaciones			
+	viewAdministrator(string dni)	void	Visualiza un administrativo
+	addExcursionist()	bool	Añade un senderista
+	editExcursionist(string dni)	bool	Modifica un senderista
+	deleteExcursionist(string dni)	bool	Elimina un senderista
+	viewExcursionist(string dni)	void	Visualiza un senderista
+	addExcursionistToRoute(int parkID, int routeID, string dni)	bool	Añade un senderista a una ruta
+	deleteExcursionistFromRoute(int parkID, int routeID, string dni)	bool	Elimina a un senderista de una ruta

"La tabla 8 muestra la clase administrativo con sus atributos y operaciones"

5.2 Diseño de comportamiento

5.2.1 Diagrama de secuencia 1



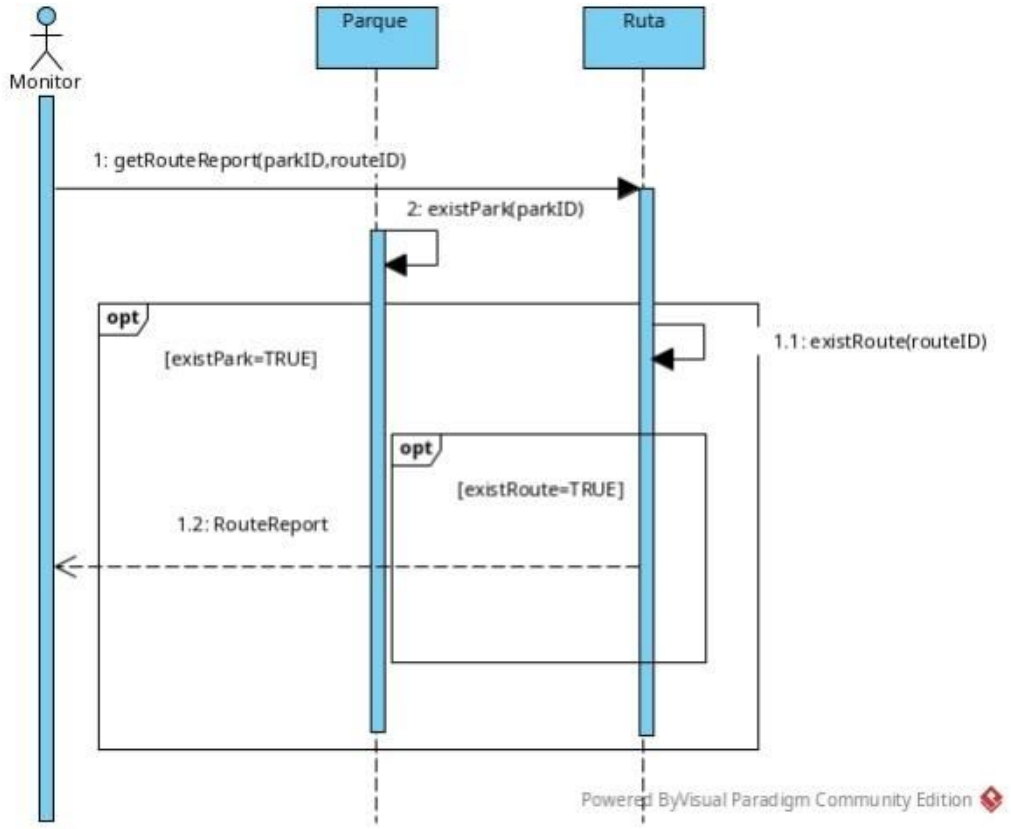
"La imagen 11 muestra el diagrama de secuencia del CU9"

En este caso de uso el administrador desea eliminar un sendero de un parque.

Para ello lo primero que hace es introducir el ID del parque al que pertenece el sendero y el ID del sendero que queremos eliminar.

Si existe el parque y el sendero. Elimina el sendero con ese ID de la lista de senderos del parque así como las rutas que pasan por ese sendero y avisa a los senderistas.

5.2.2 Diagrama de secuencia 2

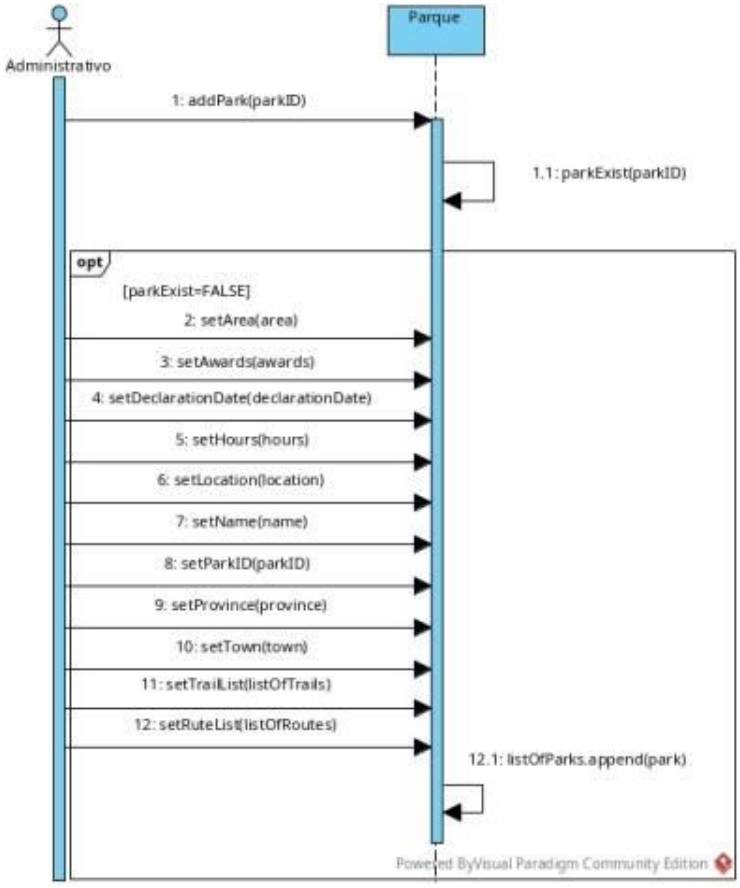


"La imagen 12 muestra el diagrama de secuencia del CU30"

En este caso de uso el monitor que está apunto de comenzar la ruta desea obtener un reporte con la información de la ruta.

Se comprueba si existe ese parque con esa ruta y una vez comprobado comienza a extraer la información correspondiente de la ruta y la vuelca en un fichero (parkID)-(route ID)Report.txt donde routeID representa en ID de la ruta de la que se extrae el reporte y parkID el ID del parque al quepertenece.

5.2.3 Diagrama de secuencia 3



"La imagen 13 muestra el diagrama de secuencia del CU1"

En este caso de uso el administrativo desea añadir un parque al sistema para ello introduce el ID del parque que desea añadir. Se comprueba si ya existe un parque con ese ID

Si no existe se procede a crear y rellenar un nuevo objeto de tipo parque haciendo uso de los métodos set de dicha clase y al final se añade al listado de parques.

6. Validación del sistema (P3)

6.1 Requisitos funcionales frente a casos de uso

RF vs CU	CU1	CU2	CU3	CU4	CU5
RF1	X	X	X	X	X
RF2					
RF3					
RF4					
RF5					

"La tabla 10 muestra los requisitos funcionales frente a los casos de uso 1-2-3-4-5"

RF vs CU	CU6	CU7	CU8	CU9	CU10
RF1	X	X	X	X	X
RF2					
RF3					
RF4					
RF5					

"La tabla 11 muestra los requisitos funcionales frente a los casos de uso 6-7-8-9-10"

RF vs CU	CU11	CU12	CU13	CU14	CU15
RF1	X	X	X	X	X
RF2		X	X		
RF3		X			
RF4					
RF5					

"La tabla 12 muestra los requisitos funcionales frente a los casos de uso 11-12-13-14-15"

RF vs CU	CU16	CU17	CU18	CU19	CU20
RF1	X	X	X	X	X
RF2	X	X	X	X	X
RF3	X				
RF4					
RF5					

"La tabla 13 muestra los requisitos funcionales frente a los casos de uso 16-17-18-19-20"

RF vs CU	CU21	CU22	CU23	CU24	CU25
RF1	X	X	X	X	X
RF2	X	X	X		
RF3	X				X
RF4					
RF5					X

"La tabla 14 muestra los requisitos funcionales frente a los casos de uso 21-22-23-24-25"

RF vs CU	CU26	CU27	CU28	CU29	CU30
RF1	X	X	X	X	X
RF2					
RF3			X		X
RF4					
RF5			X	X	

"La tabla 15 muestra los requisitos funcionales frente a los casos de uso 26-27-28-29-30"

RF vs CU	CU31	CU32	CU33
RF1	X	X	X
RF2		X	
RF3			X
RF4	X		
RF5			

"La tabla 16 muestra los requisitos funcionales frente a los casos de uso 31-32"

6.2 Casos de uso frente a clases

Matrices de validación de clases frente a los casos de uso. Las clases son, según se indican:

CL1: Parque **CL2: Sendero** **CL3: Ruta** **CL4: Persona**

CL5: Senderista **CL6: Trabajador** **CL7: Monitor** **CL8: Administrativo**

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8
CU1	X			X		X		X
CU2	X							
CU3	X			X		X		X
CU4	X	X	X	X		X		X
CU5	X			X		X		X

"La tabla 17 muestra los casos de uso 1-2-3-4-5 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8
CU6	X	X		X		X		X
CU7	X	X						
CU8	X	X		X		X		X
CU9	X	X	X	X		X		X
CU10	X	X		X		X		X

"La tabla 18 muestra los casos de uso 6-7-8-9-10 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8
CU11	X	X	X	X		X	X	X
CU12	X	X	X					
CU13	X	X	X	X	X	X	X	X
CU14	X	X	X	X	X	X	X	X
CU15	X	X	X	X	X	X	X	X

"La tabla 19 muestra los casos de uso 11-12-13-14-15 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8
CU16	X	X	X	X		X	X	X
CU17				X	X	X		X
CU18				X	X	X		X
CU19	X	X	X	X	X	X		X
CU20	X	X	X	X	X	X		X

"La tabla 20 muestra los casos de uso 16-17-18-19-20 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8
CU21	X	X	X	X	X	X		X
CU22	X	X	X	X	X	X		X
CU23	X	X	X	X	X	X		X
CU24				X		X	X	X
CU25				X		X		

"La tabla 21 muestra los casos de uso 21-22-23-24-25 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8
CU26				X		X	X	X
CU27				X		X	X	X
CU28				X		X	X	X
CU29				X		X	X	X
CU30	X	X	X	X	X	X	X	

"La tabla 22 muestra los casos de uso 26-27-28-29-30 frente a las clases"

CU vs CL	CL1	CL2	CL3	CL4	CL5	CL6	CL7	CL8	CL9
CU31			X	X		X	X	X	X
CU32	X	X	X						X
CU33				X		X	X	X	X

"La tabla 23 muestra los casos de uso 31-32 frente a las clases"

7. Implementación del sistema (P4)

Sprint 1 - 26 Noviembre a 6 Diciembre

En este sprint, el Product Owner fue David Pérez Dueñas y se han implementado las siguientes historias de usuario: HU1, HU6

Durante este primer sprint organizamos el trabajo de las próximas semanas haciendo uso de la sección Agile Boards de nuestra instancia de Youtrack.

En primer lugar decidimos implementar las historias de usuario que contenían las clases que más dependencias generaban. Es por eso que decidimos comenzar implementando la clase persona y parque a las que les asignamos la más alta prioridad (Show-Stopper). Posteriormente pasamos a implementar las clases sendero, trabajador y senderista a las que les asignamos una prioridad menor (Critical). Además debíamos implementar las clases ruta, monitor y administrativo a los que se les asignó una prioridad menor (Major) y por último debíamos unir todas las clases en un programa principal que sería el que generaría el ejecutable del sistema al que se le asignó la menor prioridad (Normal) ya que dependía de todas las clases anteriores.

Además repartimos la implementación de las clases de la siguiente forma:

- **Antonio:** Clase Sendero, programa principal, clase persona
- **Marcos:** Clase Parque, clase senderistas, clase administrador
- **David:** Clase Rutas, clase trabajador, clase monitor

Sprint 2 - 7 Diciembre a 11 Diciembre

En este sprint, el Product Owner fue Marcos Rivera Gavilán y se han implementado las siguientes historias de usuario: HU2, HU9, HU10

Se realizaron las siguientes tareas:

- En la clase Parque se implementó: el constructor, la función setArea(), la función setAwards(), la función setDeclarationDate(), la función setHours(), la función setLocation(), la función setName(), la función setParkID(), la función setProvince(), la función setTown(), la función setTrailList(), la función setRouteList().

- En la clase Sendero se implementó: el constructor, la función existExcursionist(), la función existPark(), la función existRoute(), la función existTrail(), la función setParkID() y la función setTrailID().

- En el Sistema se implementó: un autocompilador para la automatización de la compilación, la función existAdministrator(), la función existRoute(), la función existTrail(), la función existPark(), la función existMonitor() y la función existExcursionist().

- En la clase Ruta se implementó: el constructor, la función setAdaptations(), la función setDniMonitor(), la función setDniMonitorAlternate(), la función setDuration(), la función setExclusiveness(), la función setIncidences(), la función setLength(), la función Modality(), la función setName(), la función setNumberOfPlaces(), la función setExcursionistregistered() la función setParkID(), la función setRouteID(), la función setTraversedTrails(), la función setParkID() y la función existRoute().

- En la clase Senderista se implementó: el constructor, la función setparticipatOnRoute(), la función setPhoneNumber()y la función setSpecialrequirements().

-En la clase Monitor se implementó: el constructor, la función setPhoneNumber(), la función setWorkedHours(), la función setStatus(), la función notifyIncidenceOnRoute() y la función getRouteReport().

Sprint 2 - 7 Diciembre a 11 Diciembre

- En la clase Trabajador se implementó: el constructor, la función setAdress(), la función setEmail(), la función addRoute(), la función editRoute(), la función deleteRoute() y la función viewRoute().

-En la clase Administrativo se implementó: el constructor, la función addMonitor(), la función editMonitor(), la función deleteMonitor(), la función viewMonitor(), la función addAdministrator(), la función editAdministrator(), la función deleteAdministrator(), la función viewAdministrator(), la función addExcursionist(), la función editExcursionist(), la función deleteExcursionist(), la función viewExcursionist(), la función addExcursionistToRoute() y la función deleteExcursionistFromRoute().

Sprint 3 - 12 Diciembre a 16 Diciembre

En este sprint, el Product Owner fue Antonio Moruno Gracia y se han implementado las siguientes historias de usuario: HU4, HU5, HU7

Se realizó el diseño de los test del sistema que se iban a implementar y que se ven reflejados en la sección de pruebas del sistema .

Estos tests comprueban el funcionamiento de las funciones: dniIsValid(), existAdministrator(), placesLeft(), getRouteReport(), notifyIncidence() y addExcursionistToRoute().

Decidimos implementar estos test ya que probaban funciones claves del sistema y de las cuales depende el buen desempeño del mismo.

Sprint 4 - 17 Diciembre a 23 Diciembre

En este sprint, el Product Owner fue David Pérez Dueñas y se han implementado las siguientes historias de usuario: HU3

Se terminó la creación de las clases Parque, Sendero, Ruta, Senderistas, Persona, Monitor, Trabajador y Administrativo puliendo algunos fallos y optimizando el funcionamiento de las mismas.

También se terminó de implementar todas la funciones del sistema y la creación de los test correspondientes a las clases anteriormente mencionadas.

El enlace al repositorio en el que se ha implementado el programa es el siguiente: https://github.com/MarcosRigal/Is

8. Pruebas del sistema (P4)

8.1 Pruebas de Antonio Moruno

8.1.1 Test 1

ID	0001	Fecha	21-12-2020
Historia de usuario	HU8. COMO monitor QUIERO poder notificar incidencias en las rutas PARA informar a los administrativos del parque de lo ocurrido		
Título de la prueba	Notificar incidencia en una ruta		
Responsable de la prueba	Antonio Moruno Gracia		
Descripción			
Este test comprueba el correcto funcionamiento de la función notifyIncidentOnRoute()			
Prerrequisitos			
-Tener 1 monitor que pueda notificar la incidencia -Tener 1 ruta donde almacenar la incidencia -Tener las incidencias para almacenar			
Entradas			
1 monitor , 3 incidencias, 1 ruta y 1 lista de rutas			
Acciones a realizar			
-Se crea 1 monitor -Se crea 1 ruta y se añade a la lista de rutas -Se crean 3 incidencias y se añaden mediante la función notifyIncidentOnRoute() a la ruta -Se comprueba que se han añadido correctamente			

Salidas esperadas
-La lista de incidencias tiene tamaño 3 -La 1º incidencia introducida coincide con la 1º incidencia de la lista y sucesivamente
Salida obtenida
El test pasa satisfactoriamente.
Observaciones

8.1.2 Test 2

ID	0002	Fecha	21-12-2020
Historia de usuario	HU9. COMO administrativo QUIERO administrar la información de la plantilla PARA actualizar la información del sistema		
Título de la prueba	Comprobar la existencia de un Administrativo		
Responsable de la prueba	Antonio Moruno Gracia		
Descripción			
Este test comprueba el correcto funcionamiento de la función existAdministrator()			
Prerrequisitos			
-Tener 1 lista de Administrativos donde buscar el administrativo			
Entradas			
3 administrativos y 1 lista de administrativos			
Acciones a realizar			
-Se añadirán esos 3 administrativos a la lista -Se comprobará que dicha lista contiene a los administrativos -Se creará un 4º Administrativo que no se añadirá -Se comprobará que este último no ha sido añadido			
Salidas esperadas			
-La lista de administrativos tiene tamaño 3 -Los 3 primeros administrativos está en la lista y el 4º no está en la lista			
Salida obtenida			
El test pasa satisfactoriamente.			
Observaciones			

8.2 Pruebas de Marcos Rivera

8.2.1 Test 1

ID	0003	Fecha	21-12-2020
Historia de usuario	HU7. COMO monitor QUIERO disponer de un listado de los senderistas que participan en la ruta PARA poder prepararla		
Título de la prueba	Obtener el informe de la ruta		
Responsable de la prueba	Marcos Rivera Gavilán		
Descripción			
Este test comprueba el correcto funcionamiento de la función getRouteReport()			
Prerrequisitos			
-Tener 1 monitor que pueda solicitar el informe -Tener 1 administrativo que pueda apuntar un senderista -Tener 1 ruta de la que solicitar el informe -Tener 1 un senderista apuntado a la ruta			
Entradas			
1 monitor, 1 administrativo, 1 ruta, 1 lista de rutas, 1 senderista			
Acciones a realizar			
-Se crea 1 monitor -Se crea 1 administrativo -Se crea 1 ruta y se añade a la lista de rutas -Se crea 1 senderista y se le apunta a la ruta -Se crea el reporte -Se lee el reporte y se comprueba que esté bien			

Salidas esperadas
Los elementos leídos del reporte han de ser iguales a los elementos que se han introducido
Salida obtenida
El test pasa satisfactoriamente
Observaciones

8.2.2 Test 2

ID	0004	Fecha	21-12-2020
Historia de usuario	HU6. COMO administrativo QUIERO añadir y eliminar senderistas a las rutas PARA poder organizarlas		
Título de la prueba	Añadir excursionista a ruta		
Responsable de la prueba	Marcos Rivera Gavilán		
Descripción			
Este test comprueba el correcto funcionamiento de la función addExcursionistToRoute()			
Prerrequisitos			
-Tener 1 administrativo que pueda apuntar al senderista -Tener 1 ruta a la que apuntarlo			
Entradas			
1 administrativo, 1 senderista, 1 ruta, 1 lista de rutas			
Acciones a realizar			
-Se crea 1 administrativo -Se crea 1 ruta y se añade a la lista de rutas -Se crea 1 senderista y se le apunta a la ruta -Se comprueba que el senderista esté apuntado a la ruta			
Salidas esperadas			
El dni del senderista debe aparecer en la lista de dni de senderistas de la ruta			
Salida obtenida			
El test pasa satisfactoriamente			
Observaciones			

8.3 Pruebas de David Pérez

8.3.1 Test 1

ID	0005	Fecha	21-12-2020
Historia de usuario	HU9. COMO administrativo QUIERO administrar la información de la plantilla PARA actualizar la información del sistema		
Título de la prueba	Comprobar que un DNI es válido		
Responsable de la prueba	David Pérez Dueñas		
Descripción			
Este test comprueba el correcto funcionamiento de la función DNIIsValid()			
Prerrequisitos			
-Tener 4 DNIs			
Entradas			
4 DNIs, 2 son correctos y otros dos son incorrectos			
Acciones a realizar			
-Se crean 4 variables de tipo string -En 2 de ellas se almacenan DNIs incorrectos y en otras 2 se almacenan DNIs correctos			
Salidas esperadas			
Dos DNIs son incorrectos y otros dos DNIs son correctos			
Salida obtenida			
El test pasa satisfactoriamente			
Observaciones			

8.3.2 Test 2

ID	0006	Fecha	21-12-2020
Historia de usuario	HU6. COMO administrativo QUIERO añadir y eliminar senderistas a las rutas PARA poder organizarlas		
Título de la prueba	Comprobar los sitios libres de una Ruta		
Responsable de la prueba	David Pérez Dueñas		
Descripción			
Este test comprueba el correcto funcionamiento de la función PlacesLeft()			
Prerrequisitos			
-Tener 1 DNI de un monitor -Tener 1 lista de Senderos -Tener un objeto de tipo Ruta			
Entradas			
1 DNI de monitor, 1 parkID, 1 routeID, 1 lista de senderos y 1 número de plazas			
Acciones a realizar			
-Se crea un DNI de monitor, 1 lista de Senderos y 1 objeto de tipo Ruta -Se realiza un bucle for para la creación aleatoria de DNIs de senderistas -Se asignan los DNIs a la lista de Senderistas			
Salidas esperadas			
Se espera que queden 20 sitios libres			
Salida obtenida			
El test pasa satisfactoriamente			
Observaciones			

8.4 Informe de errores

Id Prueba	Acción	Error	Solución
0001	Comparar las dos listas para ver que los elementos de ambas coinciden	Con EXPECT_EQ no se pueden comparar listas	Comparar los primeros y últimos elementos de las listas, en vez de comparar las listas directamente
0002	Testear la función existAdministrator()	-	-
0003	Testear la función getRouteReport()	Había un error a la hora de realizar la lectura ya que se incluían las comas en las cadenas de texto leídas	Modificar la lectura para que leyese hasta antes de la coma
0004	Testear la función addExcursionistToRoute()	La función no comprobaba que si al añadir el senderista a la ruta se superan el número de plazas	Comprobar el número de plazas restantes con la función placesLeft()
0005	Testear la función dniIsValid()	-	-
0006	Comprobar que el número de plazas de la ruta no fueran incorrectos	No poder usar EXPECT_FALSE para ello	Comprobar solamente que el número de plazas sea correcto

9. Referencias

La documentación utilizada para elaborar esta práctica ha sido la aportada por la profesora en el moodle de la asignatura. Esto incluye los contenidos teóricos aportados en los diferentes documentos de cada una de las semanas de prácticas y la información adjuntada en los siguientes enlaces:

- https://git-scm.com/
- https://docs.github.com/es
- https://training.github.com/downloads/es_ES/github-git-cheat-sheet/
- https://docs.github.com/en/free-pro-team@latest/github/authenticating-to-github/connecting-to-github-with-ssh
- https://medium.com/@wc.testing.qa/agile-testing-de-historias-de-usuario-criterios-de-aceptaci%C3%B3n-y-c%C3%B3mo-escribir-una-historia-2a94149ecec7
- https://apiumhub.com/es/tech-blog-barcelona/como-escribir-buenas-historias-de-usuario/
- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/
- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/
- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/
- https://docs.github.com/es/free-pro-team@latest/github/setting-up-and-managing-your-github-user-account/inviting-collaborators-to-a-personal-repository
- https://www.cplusplus.com/
- https://github.com/google/googletest