SALTOUR

Soluciones tecnológicas para una mejora de la experiencia turística

Anexo IV: Diseño

VNiVERSiDAD D SALAMANCA

Proyecto de Fin de Máster en Ingeniería Informática

Tutores:

Ángel Luis Sánchez Lázaro

Ana Belén Gil González

Alumno:

Miguel Cabezas Puerto

Diseño

Modelo de diseño

Introducción

Habiendo ya definido el modelo de dominio y la arquitectura de análisis, se procede a buscar soluciones a los requisitos de usuario de cara a la implementación. Esto es gracias a que los componentes del modelo de requisitos y de análisis suministran los detalles suficientes para poder crear los modelos de diseño. El proceso de diseño, continuación del análisis, trata de realizar una traducción de los requisitos, obtenidos en la parte de definición de requisitos, en una representación del software. La parte de análisis en la etapa de diseño consiste en una traducción de los elementos del modelo de análisis a elementos del modelo de diseño a través de diferentes iteraciones mediante la "dimensión de la abstracción". En esta etapa hay dos conceptos a destacar que logran que pasemos de la etapa de diseño del sistema a su posterior implementación software. En primer lugar, la abstracción, que permite especificar internamente datos y procedimientos, pero elimina la necesidad de conocer detalles a bajo nivel. En segundo lugar, el refinamiento, que ayuda a revelar esos detalles a medida que avanza el diseño. Inicialmente se representará a un nivel de abstracción alto, pero, a medida que tengan lugar las consecuentes iteraciones de diseño, se irán refinando y detallando los objetos de diseño e iremos descendiendo en el nivel de abstracción hasta alcanzar el nivel más bajo, donde se plantea una solución que puede implementarse directamente.

El modelo de diseño describe la realización física de los casos de uso y estará compuesto por:

- Subsistemas de diseño: trata de organizar los artefactos que vamos a tratar en el diseño en piezas más manejables.
- Clases de diseño: formado por un diagrama de clases de diseño que, a diferenciadel modelo de análisis, contendrá las operaciones y atributos necesarios para darel salto al nivel de representación.
- Realizaciones de caso de uso-diseño: se trata de una colaboración en el modelo de diseño, es decir, cómo se realiza un caso de uso específico. Esto se modelará através de los diagramas de actividad, de estado y de secuencia de dichos casos deuso.
- Arquitectura de capas: muestra los artefactos más relevantes del modelo de diseño a través de carpetas distribuidas en capas o niveles diferentes.

- Modelo de despliegue: describe la distribución física del sistema en términos delmodo de distribución de la funcionalidad entre los nodos.
- Diseño de datos: detalla cómo se han almacenado y tratado los datos requeridos en los requisitos y desarrollados en el resto de los modelos.
- Diseño de interfaces: indica el diseño elegido para el desarrollo de las interfaces y vistas de usuario.

Subsistemas de diseño

Para el desarrollo del proyecto se ha seguido el patrón MVC (Modelo-Vista-Controlador). Se puede ver en el apartado *6.3 Clases de diseño* cómo se han distribuido las diferentes clases del sistema siguiendo el patrón. Además, se ha seguido un patrón DAO (Data Access Object) para la interacción con los datos almacenados de forma persistente. En este apartado se detalla el fundamento teórico de los patrones, así como una arquitectura de diseño global del sistema que posteriormente se detallará en cada clase en el apartado anteriormente mencionado.

Fundamento teórico MVC

El patrón MVC es un patrón arquitectónico. Pertenece a la categoría de los patrones para sistemas interactivos. Se encuentra en muchos sistemas interactivos y frameworks de aplicación para software con interfaces gráficas (MacApp, ET++, bibliotecas de Smalltalk, MFC). En muchas aplicaciones las interfaces de usuario tienden a cambiar debido a la extensión de su funcionalidad o a su transporte a un entorno gráfico diferente. El patrón MVC surge para dar respuesta mediante la construcción de sistemas flexibles donde el núcleo funcional no se entremezcle con la interfaz. Para ello divide la aplicación en tres áreas: proceso, entrada y salida.

- El componente de modelo encapsula los datos y la funcionalidad central. Es independiente de la entrada y la salida
- Los componentes de vista presentan la información al usuario. Una vista obtiene datos del modelo, pudiendo haber múltiples vistas del modelo
- Los controladores reciben la entrada, normalmente eventos que son trasladados para servir las peticiones del modelo o de la vista. El usuario interactúa con el sistema solo a través de los

controladores

De este modo el modelo contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia, la vista compone la información que se envía al cliente y los mecanismos interacción con éste mientras que el controlador actúa como intermediario entre el modelo y la vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

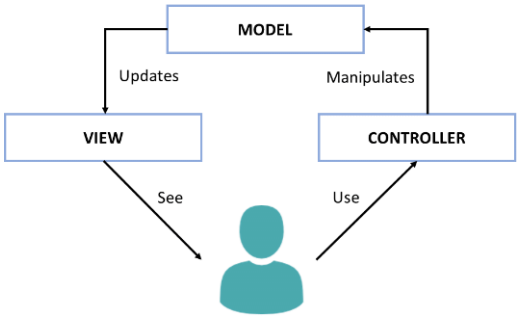


Ilustración 1 Patrón MVC

Este patrón es muy útil ya que permite tener múltiples vistas sincronizadas del mismo modelo, realizar cambios de vistas y controles en tiempo de ejecución y un sencillo cambio del aspecto externo de las aplicaciones. El inconveniente del patrón radica en el aumento de la complejidad en el desarrollo. Esta complejidad se subsana fácilmente en Android ya que la vista se realiza a través de ficheros XML que indican cuál debe ser el aspecto de la aplicación, dejando los ficheros Java para la implementación del modelo y controlador.

Fundamento teórico DAO

Data Access Object (DAO) es un patrón de diseño arquitectónico orientado a objetos utilizado para desacoplar la lógica de negocio de la lógica de acceso a datos, de manera que se pueda cambiar la fuente de datos fácilmente incluso durante la instalación/configuración de la aplicación. Mediante este patrón se puede abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos

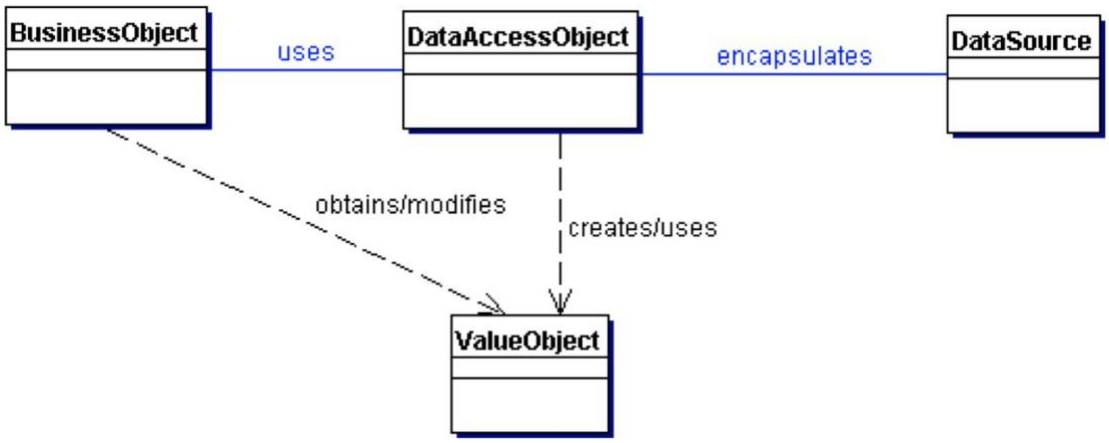


Ilustración 2 Patrón DAO

Propuesta de arquitectura diseño

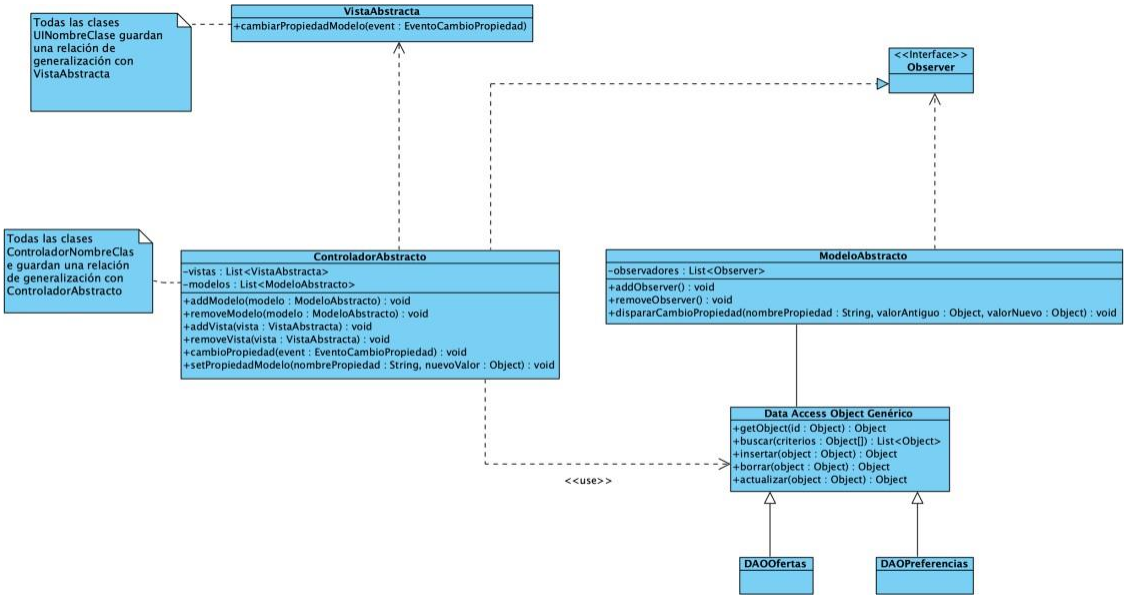


Ilustración 3 Arquitectura de diseño

En una visión a alto nivel se define esta propuesta de arquitectura MVC haciendo también uso del patrón Observer.

El patrón de diseño Observer permite observar los cambios producidos por un objeto, de esta forma, cada cambio que afecte el estado del objeto observado lanzará una notificación a los observadores; a esto se le conoce como Publicador-Suscriptor. Observer es uno de los principales patrones de diseño utilizados en interfaces gráficas de usuario (GUI), ya que permite desacoplar al componente gráfico de la acción a realizar.

A partir de la idea recogida en esta arquitectura se detallará, en el siguiente apartado, cada controlador, vista y modelo asociado a las diferentes clases. Los elementos correspondientes a esta arquitectura MVC con patrón Observer son:

- Observer: establece un protocolo para la implementación del patrón de diseñoObserver
- VistaAbstracta: clase que establece un protocolo para las clases que quieranfuncionar como Vista
- Métodos
    - cambioPropiedadModelo: actualiza la vista cuando se produce unevento de cambio en alguna propiedad del modelo
- ControladorAbstracto: clase que establece un protocolo para las clases quequieran funcionar como Controlador
    - Atributos
        - vistas: lista de vistas asociadas al controlador, cuyos eventos éstegestiona
        - modelos: lista de modelos asociados al controlador, cuyos datosson fuente del estado de las vistas
    - Métodos
        - addVista/removeVista: añade o borra un objeto a la lista devistas
        - addModelo/removeModelo: añade o borra un objeto a la listade modelos
        - cambioPropiedad: notifica a todas las vistas del cambio en unapropiedad de algún modelo
        - setPropiedadModelo: establece el valor de cierta propiedad delmodelo
- ModeloAbstracto: clase que establece un protocolo para las clases que quieranfuncionar como Modelo
    - Atributos
        - observadores: lista de objetos de tipo Observer que monitorizan elestado de este modelo
    - Métodos
        - addObservador/removeObservador: añade o borra

un objeto a lalista de observadores

- dispararCambioPropiedad: genera un evento de cambio depropiedad para notificar a los observadores

Los elementos correspondientes al patrón DAO son:

- DataAccessObject: objeto que se comunica con el sistema de almacenaje persistente de los datos, de él heredarán diferentes objetos DAO especializados encada tipo de dato que se almacena en el sistema de forma persistente.
  - Métodos
    - getObject: obtiene un objeto desde el almacenamiento persistentea partir de un identificador
    - buscar: busca un objeto en almacenamiento persistente que cumplaciertos criterios
    - insertar: inserta un objeto en almacenamiento persistente
    - actualizar: actualiza un objeto en almacenamiento persistente ▪ borrar: borrar un objeto del almacenamiento persistente
- DAOX: objeto DAO especializado en cada tipo de dato almacenado de manera persistente en el sistema, es decir, las ofertas de empleo y formación y las preferencias de usuario.

Arquitectura de capas

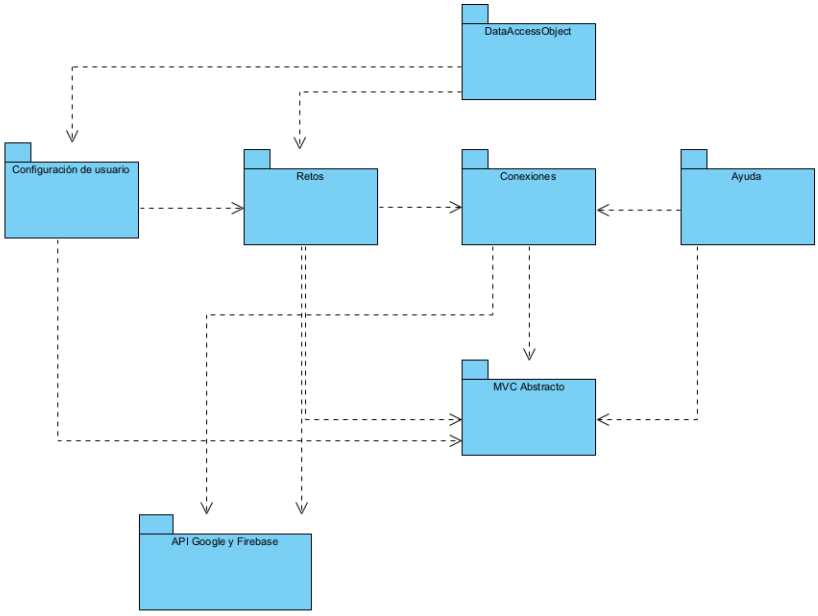


Ilustración 4 Arquitectura de capas

Clases de diseño

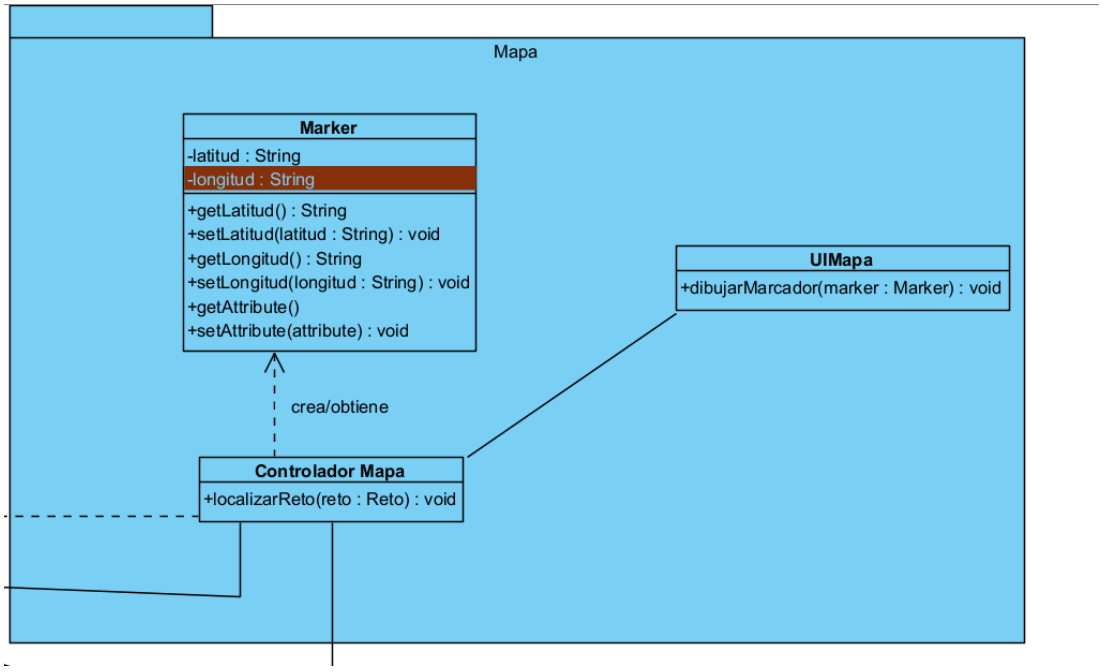


Ilustración 5 Clases de diseño Mapa

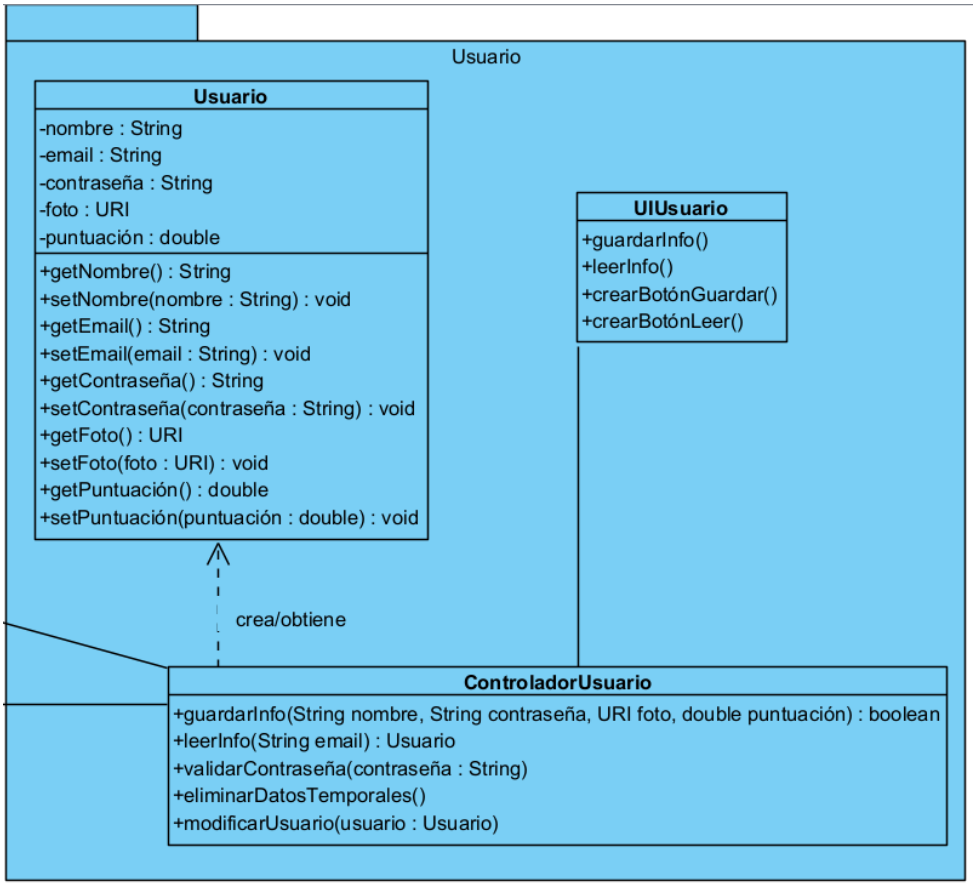


Ilustración 6 Clases de diseño Usuario

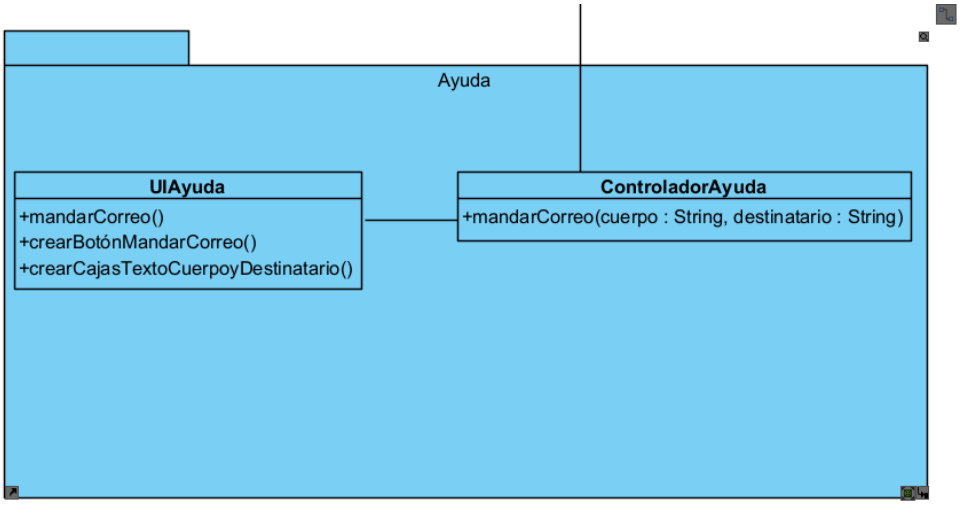


Ilustración 7 Clases de diseño ayuda

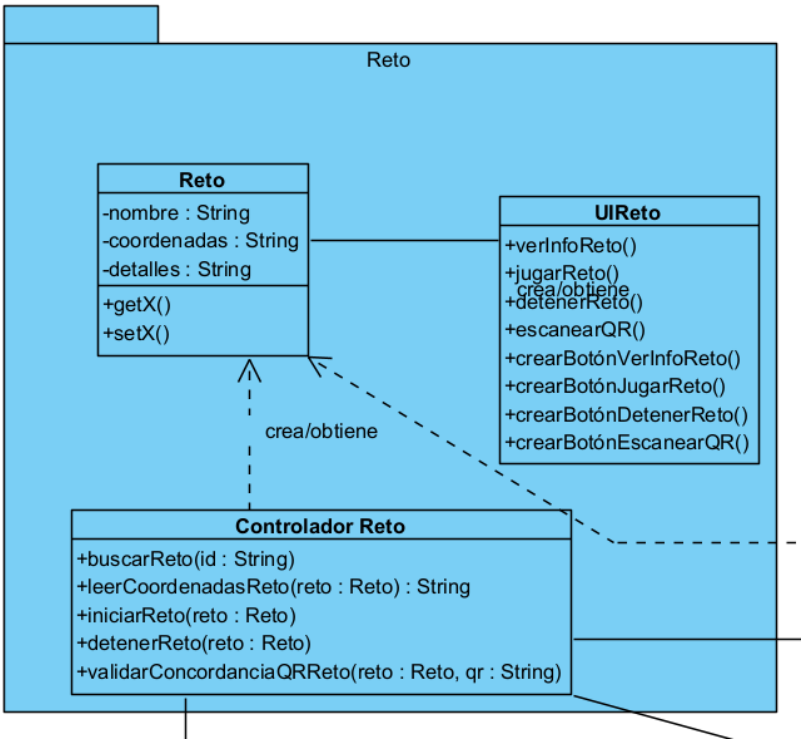


Ilustración 8 Clases de diseño reto

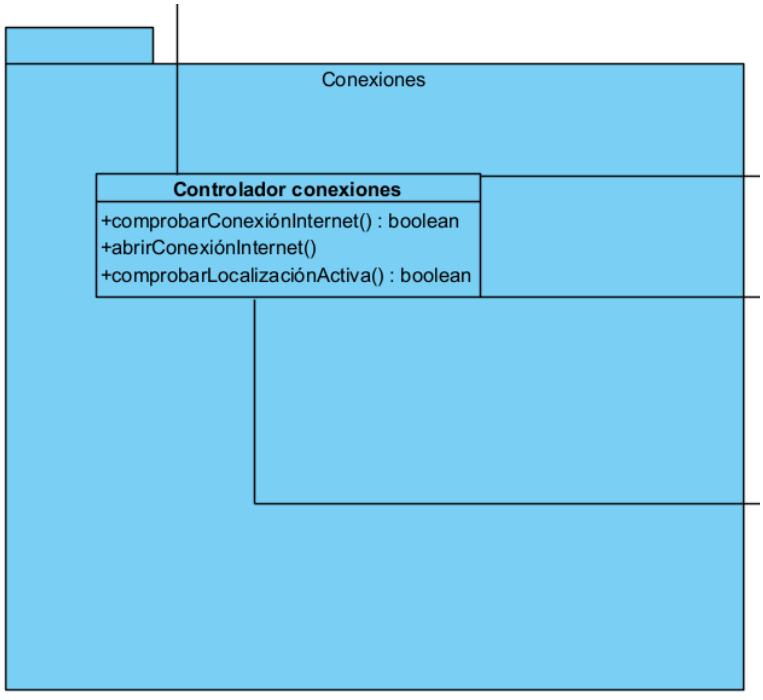


Ilustración 9 Claases de diseño conexiones

En las anteriores imágenes podemos ver cada uno de los paquetes del sistema por separado. En la siguiente imagen podemos ver el conjunto de ellos con todas sus relaciones. Para un mayor detalle y una mejor visualización se recomienda acudir al proyecto Visual Paradigm adjunto al archivo Diagramas clases diseño.

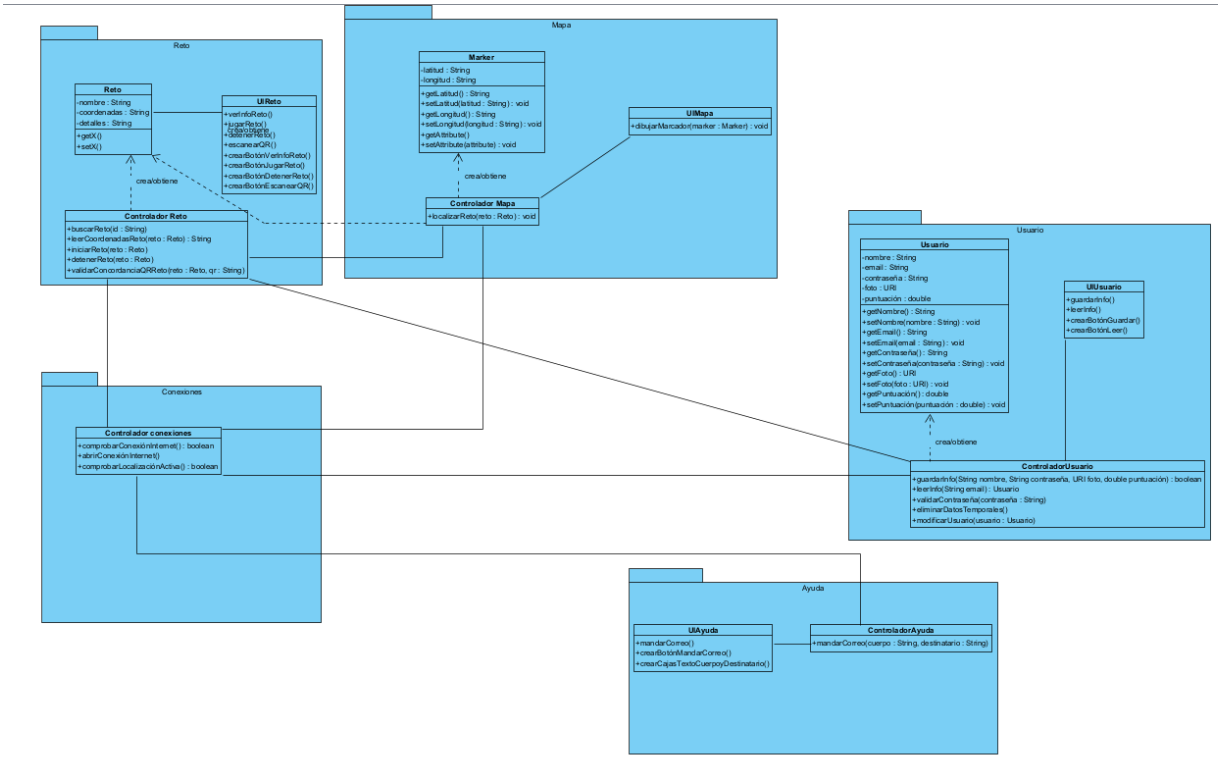


Ilustración 10 Diagrama de clases de diseño completo

Glosario clases de diseño

Reto

- Reto
  - Atributos
    - nombre. Nombre del punto de interés
    - coordenadas. Posición de la oferta en la superficie de la Tierra. Contiene latitud y longitud.
    - detalles. Breve descripción, historia y datos curiosos del punto de interés.
  - Métodos
    - getX: devuelve el valor del atributo X ▪ setX: guarda el valor del atributo X.
- UIReto
  - Métodos

- verInfoReto(): permite al usuario acceder a la información de un reto lanzando el método respectivo del controlador.
- jugarReto(): permite al usuario comenzar un reto lanzando el método respectivo del controlador
- detenerReto(): permite al usuario detener un reto lanzando el método respectivo del controlador
- escanearQR(): permite al usuario detener escanear un código QR de un reto lanzando el método respectivo del controlador
- crearBotónVerInfoReto(): crea el botón para lanzar la acción dever la información de una oferta.
- crearBotónJugarReto(): crea el botón para lanzar la acción de jugar a un reto.
- crearBotónDetenerReto(): crea el botón para lanzar la acción de detener un reto.
- crearBotónEscanearQR(): crea el botón para lanzar la acción de escanear un código QR.

➢ ControladorReto

  o Métodos

    - +buscarReto(nombre:String) : Reto . Busca el reto que coincidan con los parámetros recibidos y lo devuelve al usuario.
    - +leerCoordenadasReto(reto : Reto) : String. Accede a los datos de localización del reto que recibe para devolverlos al sistema en forma de texto y que este lo procese.
    - +iniciarReto(reto : Reto, usuario: Usuario) : boolean. Marca el reto como activo para el usuario e inicia el contador de tiempo
    - +detenerReto(reto : Reto, usuario: Usuario) : boolean. Marca el reto como inactivo para el usuario y para el contador de tiempo
    - +validarConcordanciaQRReto(reto : Reto, String

QR) : boolean. Valida si el QR recibido coincide con el QR del reto activo

Configuración de usuario

- Usuario
  - Atributos
    - nombre: *nickname* visible del usuario.
    - email: dirección de correo vinculada a la cuenta
    - contraseña: clave de acceso a la aplicación
    - foto: foto de perfil del usuario.
    - Puntuación: tiempo medio invertido por el usuario en los retos completados
  - Métodos
    - getX: devuelve el valor del atributo X ▪
    - setX: guarda el valor del atributo X
- UIUsuario
  - Métodos
    - +guardarInfo(): permite al usuario guardar/actualizar los valores de su nombre, foto de perfil y contraseña.
    - +leerInfo(): permite al usuario consultar la información de usuario propia
    - +crearBotónGuardar(): crea el botón para lanzar la acción de guardar la información personal.
    - +crearBotónLeer (): crea el botón para lanzar la acciónde leer la información personal.
    - +registarUsuario(): permite al usuario anónimo registrarse
    - +loginUsuario(): permite al usuario anónimo autenticarse
    - +logoutUsuario(): permite al usuario salir de la

aplicación

- +modificarUsuario(): permite al usuario modificar su perfil
- +crearBotónLogin (): crea el botón de login
- +crearBotónLogout (): crea el botón de logout
- +crearBotónRegistrar (): crea el botón de registro
- +crearBotónModificar (): crea el botón para lanzar la acciónde modificar usuario

- ControladorUsuario
  - Métodos
    - +guardarInfo(String nombre, String contraseña, URI foto, Double puntuación) : boolean. Almacena la información del usuario
    - +leerInfo(String email) : Usuario. Recupera la información del usuario a partir de su email vinculado
    - +validarContraseña(String contraseña) : boolean. Comprueba que la contraseña pasa los estándares de seguridad marcados
    - +modificarUsuario(Usuario usuario) : boolean. Actualiza los datos del usuario.

Ayuda usuario

- UIAyuda
  - Métodos
    - +mandarCorreo(): permite al usuario enviar un correo al soportede la aplicación lanzando el método respectivo del controlador.
    - +crearBotónMandaCorreo(): crea el botón para lanzar la acción deenviar un correo electrónico.
    - +crearCajasTextoCuerpoyDestinatario(): crea las cajas de texto para introducir el destinatario del correo (relleno por defecto), el asunto (relleno por

defecto) y el cuerpo del mensaje.

- ControladorAyuda
  - Métodos
    - +mandarCorreo(cuerpo : String, destinatario : String). Llamando al controlador de conexiones verifica que el usuario tiene conexióna Internet, que los valores de destinatario del email son correctos y lo manda.

Mapa

- Marker
  - Atributos
    - latitud: distancia angular que hay desde el punto donde se encuentra una oferta en la superficie de la Tierra hasta el paralelo del ecuador. Medida en grados, minutos y segundos sobre los meridianos
    - longitud: distancia angular expresada en grados, entre el meridianodel punto donde se encuentra una oferta sobre la superficie de la Tierra y otro tomado como referencia en el Ecuador. Medida en grados, minutos y segundos.
  - Métodos
    - getX: devuelve el valor del atributo X
    - setX: guarda el valor del atributo X
- UIMapa
  - Métodos
    - +dibujarMarcador(marker : Mapa.Marker) : void. Pinta los marcadores relativos a las coordenadas de los retos sobre el mapa
- ControladorMapa
  - Métodos

- +localizarReto(reto: Reto) : void. Ubica los retos que recibe sobre un mapa mediante sus coordendas haciendo uso de algún sistema geolocalizador, por ejemplo, la API de Google Maps.

Conexiones

- ControladorConexiones
  - Métodos
    - +comprobarConexiónInternet(): verifica si el usuario mantiene la conexión a Internet.
    - +abrirConexiónInternet() : void. Abre una conexión a Internet para poder hacer peticiones a servidores externos
    - +comprobarLocalizaciónActiva() : boolean. Verifica si el usuario tiene la geolocalización del dispositivo activa

Realizaciones de caso de uso-diseño

Con el fin de reducir el tamaño de los diagramas se ha dado por supuesto la llamada a la interfaz y consecuente pedida de datos por parte de esta indicada en diseño, así como la ocurrencia de errores en algunos procesos y los métodos “get” y “set” de las clases del modelo. Además, se modelarán aquellos casos de uso en los que el actor desencadenante de la acción sea el usuario ya que los realizados por el sistema se derivan de acciones que el usuario realice y por tanto se incluyen en los propios diagramas.

Reto

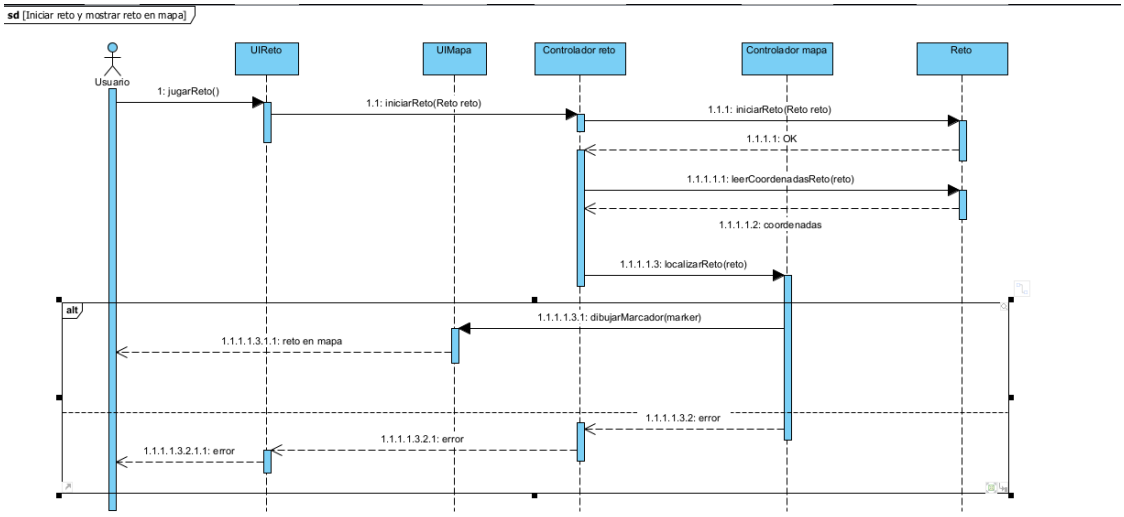


Ilustración 11 Realización CU diseño iniciar y mostrar reto

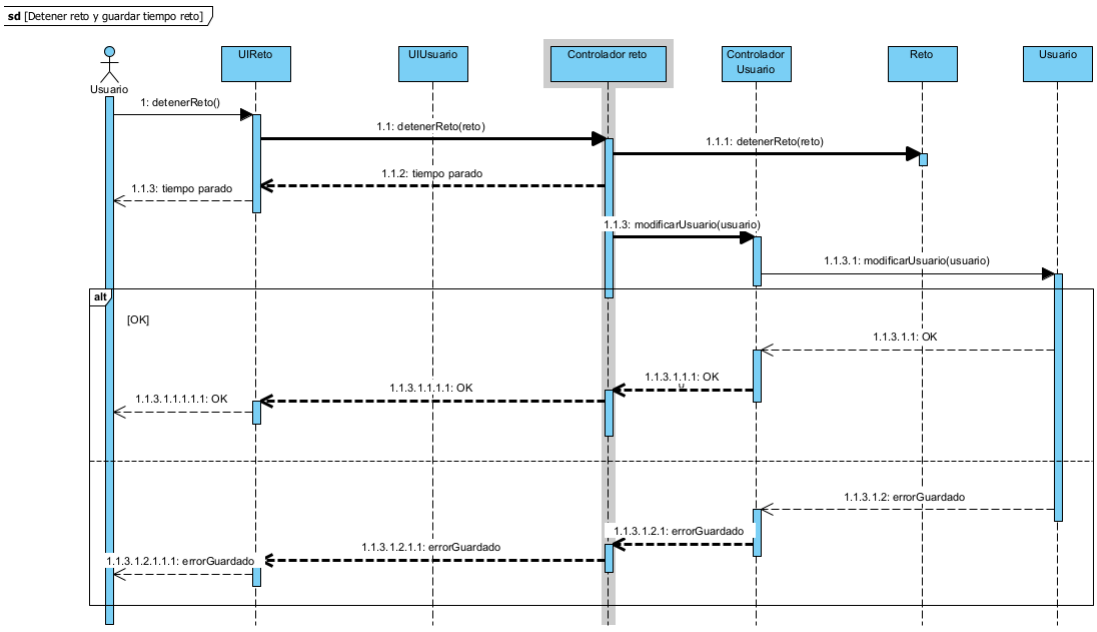


Ilustración 12 Realización CU diseño detener y guardar reto

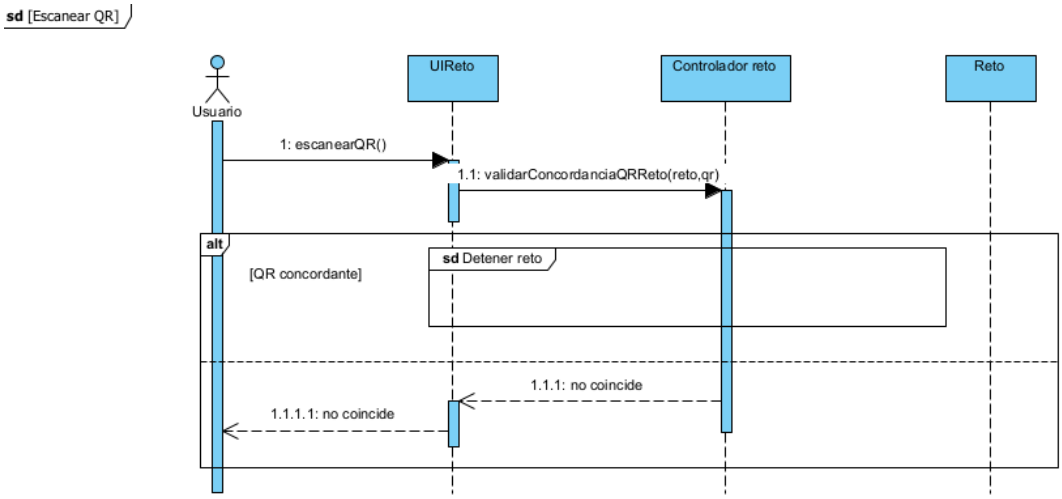


Ilustración 13 Realización CU diseño escanear QR

Usuarios

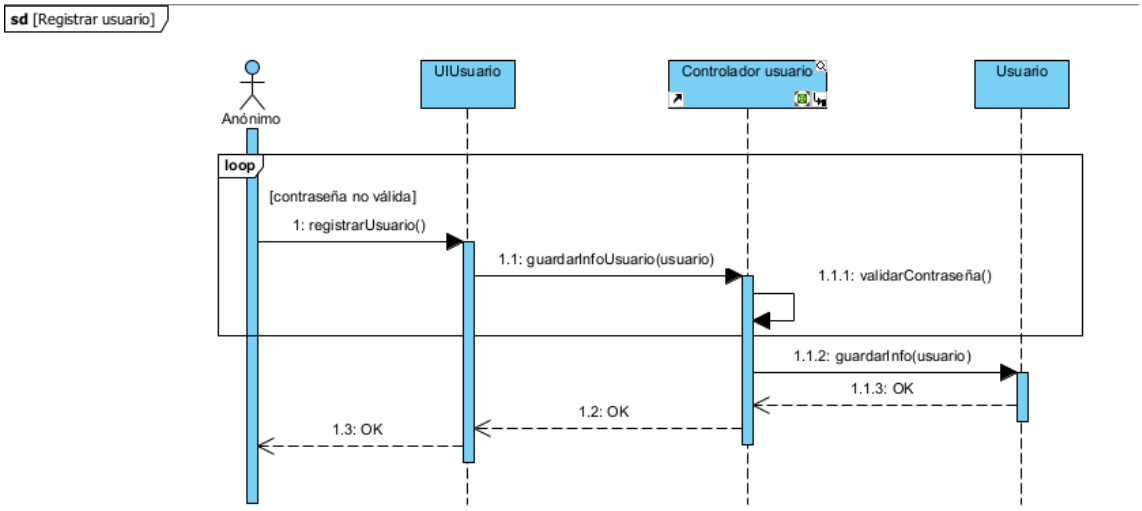


Ilustración 14 Realización CU diseño registrar usuario

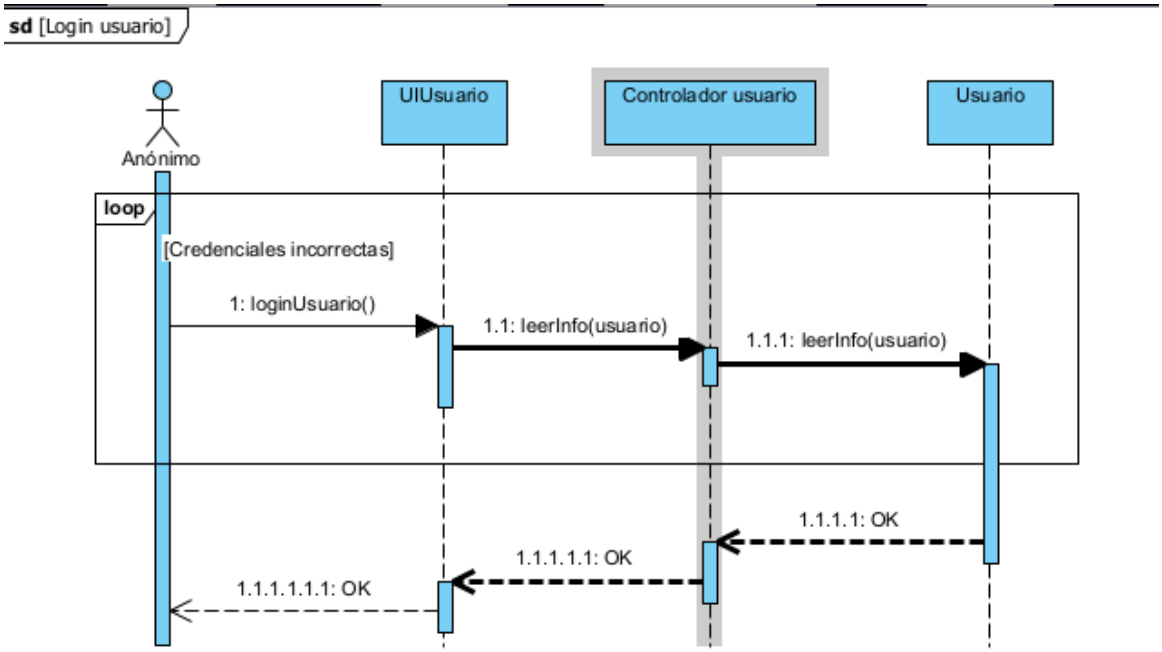


Ilustración 15 Realización CU diseño login usuario

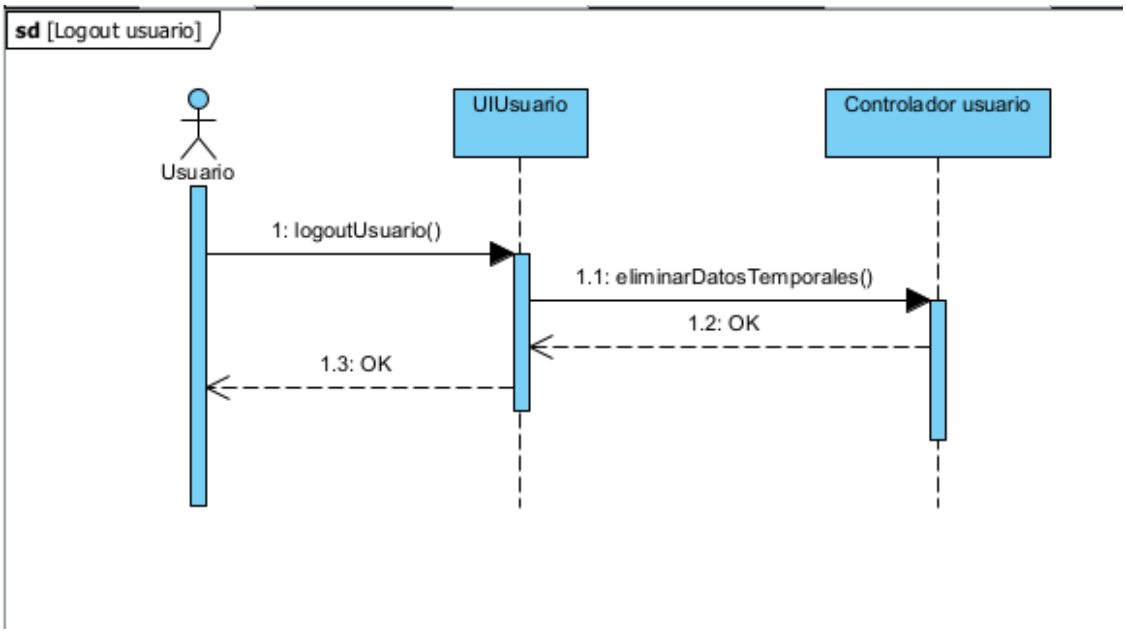


Ilustración 16 Realización CU diseño logout usuario

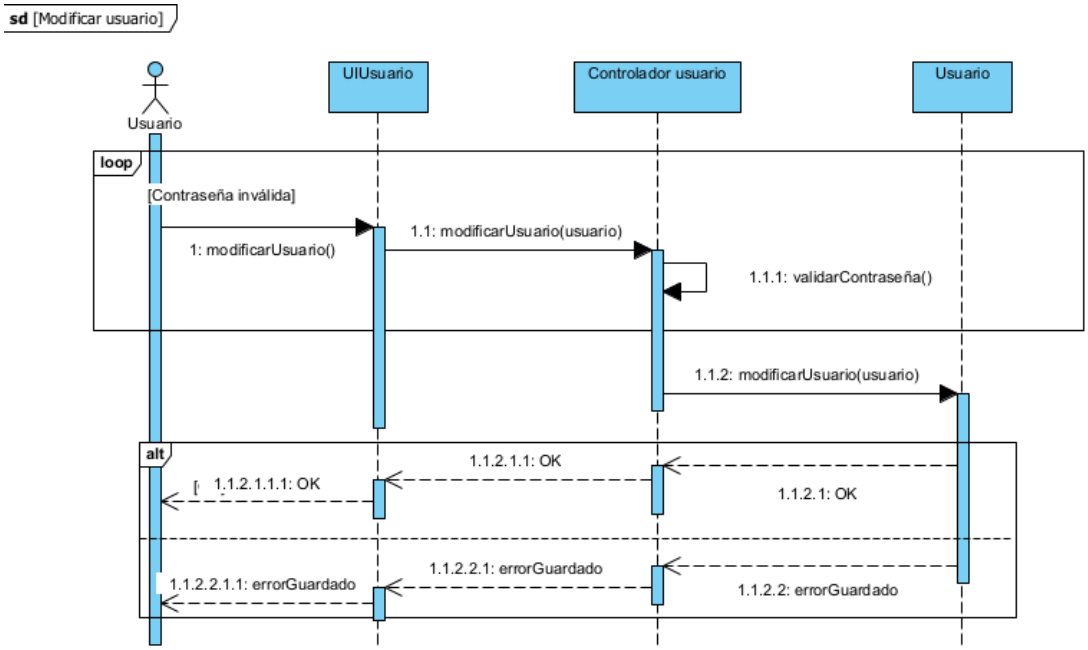


Ilustración 17 Realización CU diseño modificar usuario

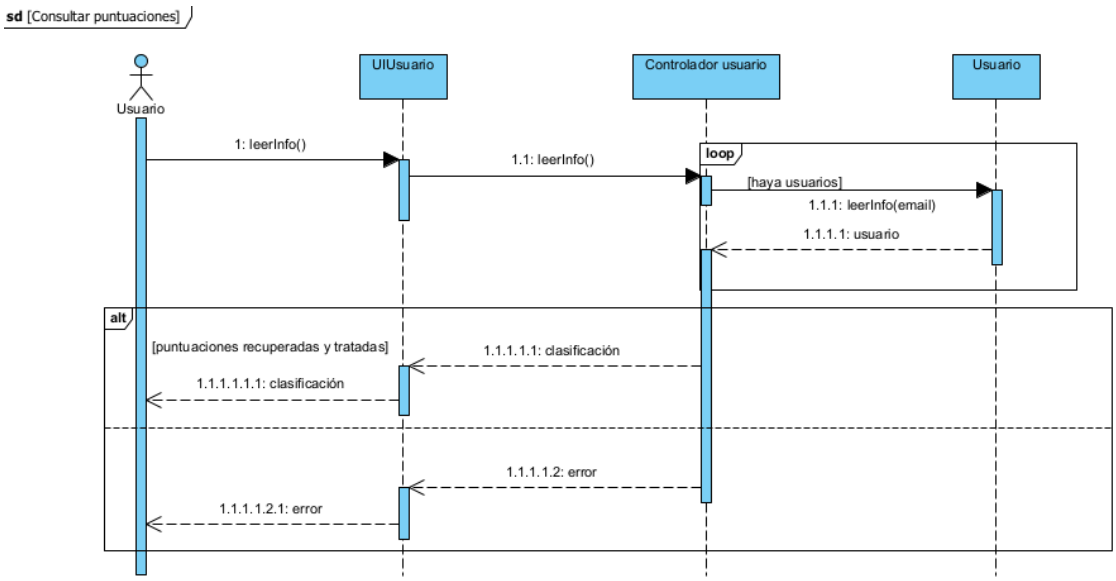


Ilustración 18 Realización CU diseño consultar puntuaciones

Ayuda

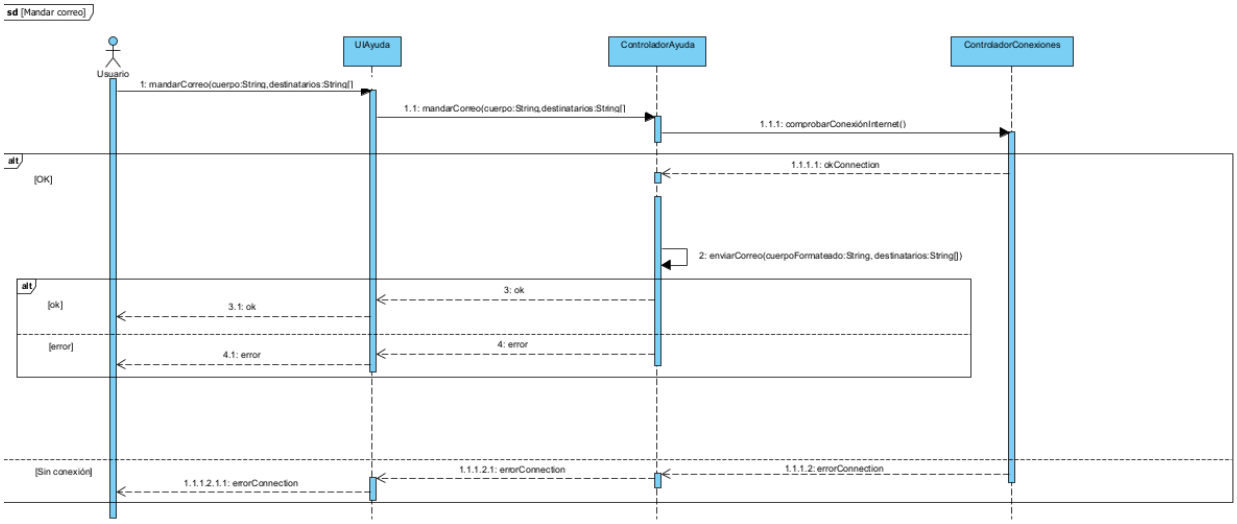


Ilustración 19 Realización CU diseño mandar correo

Diagrama de despliegue

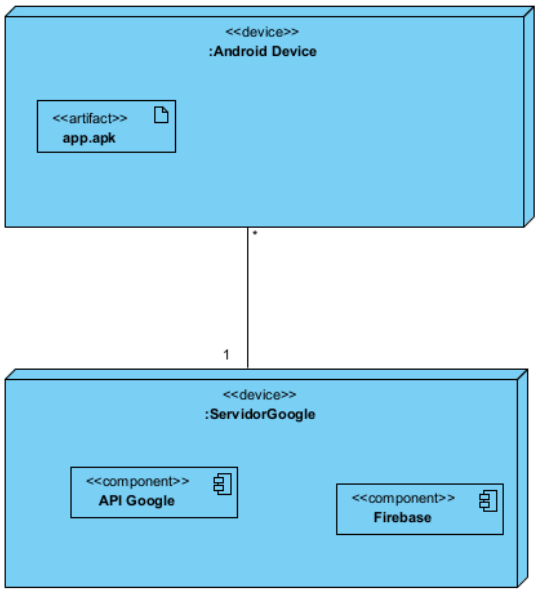


Ilustración 20 Diagrama de despliegue

Diseño de datos

La aplicación almacena una cantidad de datos compartidos en tiempo real entre múltiples usuarios por lo cual se debe contar con una arquitectura cliente-servidor en la cual el dispositivo móvil consuma los datos de una base de datos que se encuentre en un servidor compartido a todos los usuarios.

Modelo relacional

Modelo de organización y gestión de bases de datos consistente en el almacenamiento de

datos en tablas compuestas por filas, o tuplas, y columnas o campos. Se distingue de otros modelos, como el jerárquico, por ser más comprensible para el usuario inexperto, y por basarse en la lógica de predicados para establecer relaciones entre distintos datos. Surge como solución a la creciente variedad de los datos que integran las data warehouses y podemos resumir el concepto como una colección de relaciones. Sus principales conceptos son:

- Tabla: es el nombre que recibe cada una de las relaciones que se establecen entre los datos almacenados. Están formadas por filas, también llamadas tuplas, donde se describen los elementos que configuran la tabla (es decir, los elementos de la relación establecida por la tabla), columnas o campos, con los atributos y valores correspondientes, y el dominio, concepto que agrupa a todos los valores quepueden figurar en cada columna.
- Claves: elementos que impiden la duplicidad de registros, una de las grandes desventajas que presentan otros modelos de organización y gestión de bases de datos. Existen dos grandes tipos de claves: las claves primarias y las secundariaso externas.
- Claves primarias: son los atributos según el tipo de relación que se ha definido en la tabla. Pueden añadirse otros atributos específicos y propios.
- Claves externas o secundarias: son las claves que se definen para cada una de las claves primarias establecidas para los elementos o entidades de unarelación.
- Restricción de identidad: límites que se imponen en las relaciones, imprescindibles para mantener la significación correcta de la base de datos. Es un concepto íntimamente vinculado a las reglas de integridad propias del modelo relacional, el cumplimiento de las cuales está garantizado por las claves primariasy externas.
- Reglas de integridad: reglas que garantizan la integridad de los datos, es decir, la correspondencia plausible de los datos con la realidad.

En este apartado se ve la descripción de las tablas contenidas en la base de datos junto con las relaciones entre ellas.

- Usuario (nombre, email, contraseña, foto, puntuación)

- Reto(nombre,latitud,longitud,detalles)

Modelo entidad relación

El diagrama muestra las relaciones de entidad relación de las tablas de la base de datos y sus características.

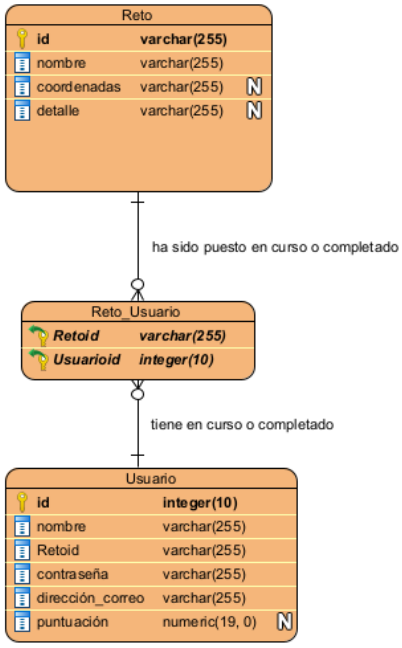


Ilustración 21 Diagrama entidad-relación

Diseño de la interfaz

El diseño de la interfaz de la aplicación ha sido un proceso también iterativo, ya que se han tomado en cuenta las opiniones de gente para mejorar la interfaz de usuario.

En los requisitos se recogía que la interfaz fuera sencilla de utilizar, por ello las pruebas con usuarios han cobrado mucha importancia desde el primer momento al no existir una aplicación ampliamente conocida en el mercado similar a la propuesta.

Fondos y colores

Los colores que emplea la aplicación constituyen uno de los primeros filtros para el usuario. Los tonos empleados indican qué se quiere transmitir con la aplicación y simbolizan uno de los focos más influyentes sobre la interpretación del usuario.

Se ha hecho uso de tonalidades rojizas y granates para el diseño de la aplicación.

Según la teoría y psicología del color, el granate simboliza la pasión, el atrevimiento, la energía y la estimulación, sensaciones que se desean potenciar para así fomentar la competitividad dentro de la aplicación que conllevaría un mayor uso y un disfrute del turismo alternativo obedeciendo la ley de consumo rápido y competición que impera en las sociedades de hoy en día. Además, normalmente se emplean dichas tonalidades para estimular a las personas a tomar decisiones rápidas lo cual refuerza la idea anteriormente expuesta.

Disposición de los elementos.

Basándose en la distribución en aplicaciones estándares para Android, experiencias de usuarios en dichas aplicaciones y opiniones de gente a lo largo del proceso de desarrollo, se ha ido modificando la interfaz para finalmente quedar constituido el diseño por los elementos que posteriormente se detallan

Pantalla de login y registro

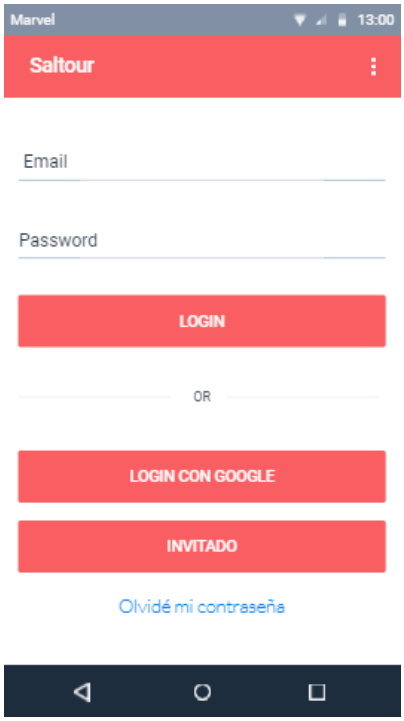


Ilustración 22 Pantalla login y registro

La pantalla de login y registro tratrará de ser lo más parecida al estándar seguido por las aplicaciones Android, esto se hace así para que la curva de aprendizaje del usuario sea menos costosa al estar ya familiarizado debido a experiencias pasadas.

Pantalla principal

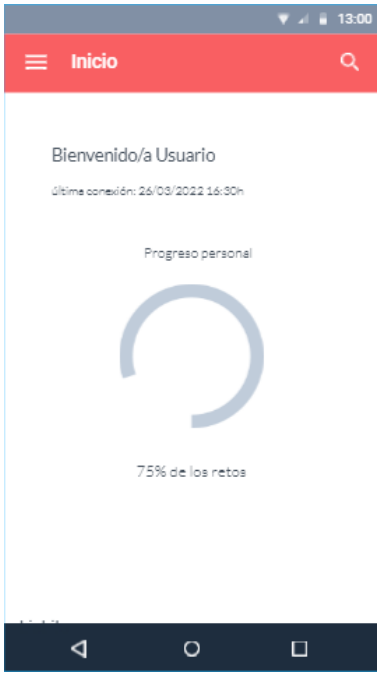


Ilustración 23 Pantalla principal

Se ha elegido un formato simple y a través de un gráfico que crezca mediante animación

para dar en un simple vistazo toda la información general de una forma estética y llamativa para el usuario.

Tanto en esta como en el resto de pantallas se ha elegido un menú lateral hamburguesa desplegable común en aplicaciones Android y con el que se ocupa un tamaño ínfimo de pantallas, evitando crear otra pantalla al efecto a la que tendría que navegar continuamente el usuario para visualizar las diferentes funcionalidades de la aplicación. Así, mediante el despliegue, se ahorran pasos y tiempo de navegación que podría hacer que el usuario abandonase la aplicación.

En este menú lateral, se primará la agrupación de funcionalidades similares, separado cada bloque por cierta distancia, para así facilitar su distinción al usuario.

Pantalla olvido de contraseña

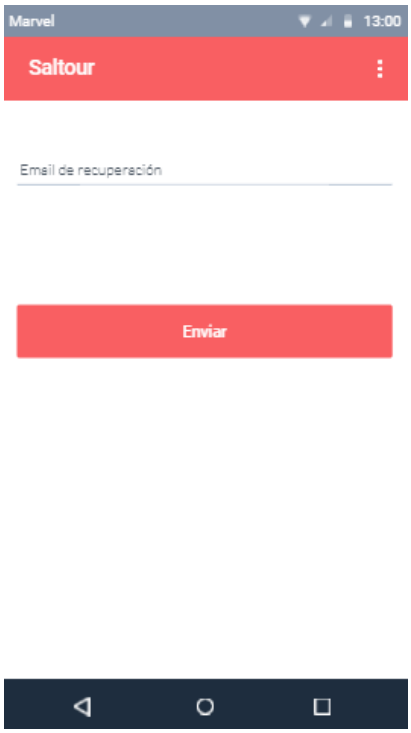


Ilustración 24 Pantalla olvido contraseña

Se ha decidido tener un único cuadro de texto donde introducir el correo de recuperación y un botón ya que cuando el usuario acuda a esta pantalla es probable que no esté tranquilo debido a que ha olvidado su contraseña. Por ello los estímulos recibidos tienen que ser pocos y directos.

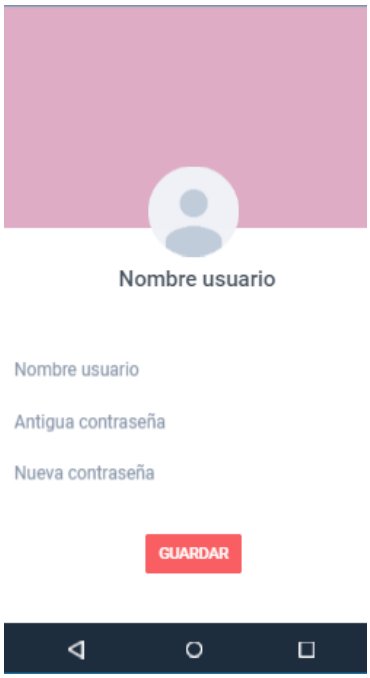
Pantalla de jugar

Ilustración 25 Pantalla jugar

Un listado con todos los retos (nombre y foto descriptiva). Al pulsar en cada uno de ellos se comenzará a jugar a encontrarlo activando el conteo de tiempo. El color de los retos dependerá de su grado de compleción, asignando una escala (verde, amarillo, sin color) para representar retos completados, en curso o sin empezar. Se han seleccionado estos colores ya que generalmente se utilizan para estos fines.

Respecto a los botones, el asociado a detener el tiempo, así como el de escanear QR únicamente aparecerán si el usuario tiene un reto con conteo de tiempo activo. Así por una parte se evita un mal uso de la aplicación por sobrecarga de información innecesaria.

Pantalla perfil de usuario



Formada por elementos editables al pulsar sobre ellos (nombre de usuario, contraseña y foto de perfil). Esta información es ampliable con tantos parámetros e información se recopile en la aplicación acerca del usuario.

Disposición lineal para que el usuario pueda seguir un paso tras otro en la edición de su perfil.

Pantalla de estadísticas

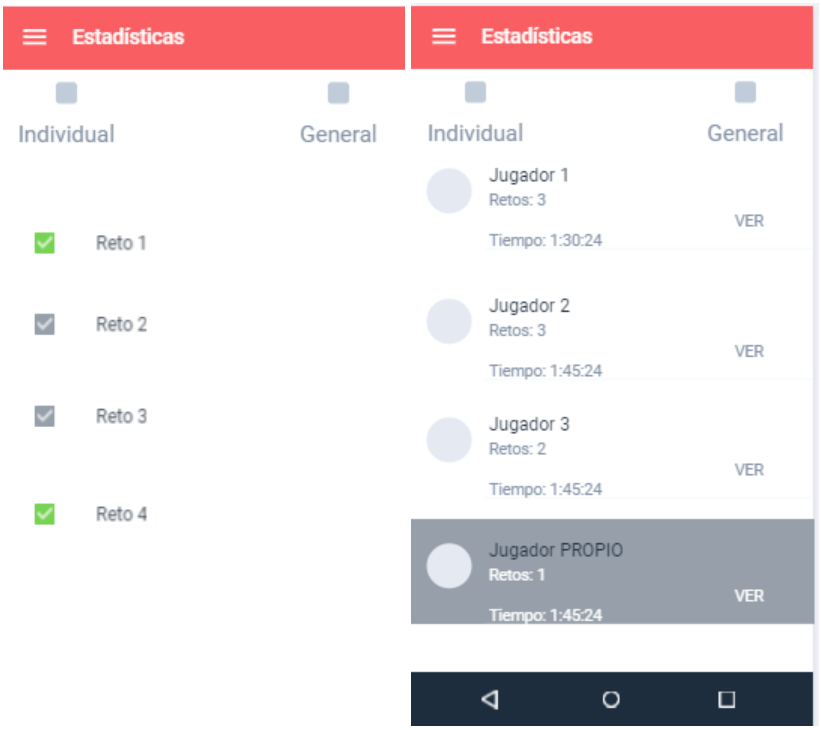


Ilustración 26 Pantalla estadísticas

Pantallas separadas en función de su alcance: una para el progreso individual y otra para clasificaciones generales.

Respecto al progreso individual los tics se automarcarán a medida que el usuario vaya completando retos no siendo estos editables por su parte.

Pantalla de ayuda/preguntas frecuentes

Ilustración 27 Pantalla ayuda

Pese a que no aparezca en la ilustración, cada una de las preguntas consiste en un desplegable en el que el encabezado es la propia pregunta y su contenido la respuesta. Además, se situará un link en la parte inferior para contactar con los desarrolladores vía correo electrónico en caso de que el usuario tenga alguna duda no recogida en el momento en este apartado. Cabe destacar que, siguiendo las recomendaciones para desarrolladores de Android, se suprimirán en todas las pantallas el botón de volver dejando esta funcionalidad en manos de los botones de retorno ya disponibles en los dispositivos.

Tipografía

Al igual que el color, la tipografía también despierta sensaciones en los lectores, por lo que, de la misma forma, juega un papel importante en la aplicación.

Tanto para la parte desarrollada en Android (Roboto) como para la parte de Unity (Arial), se han elegido tipografías sans-serif. El motivo principal radica en que son las mejores para textos cortos y están especialmente indicadas para visualizaciones en pantallas. Además, transmiten seguridad y alegría (Flores, s.f.), que son sensaciones buscadas mediante el uso de nuestra aplicación.