Documentación de parkalot

Índice:

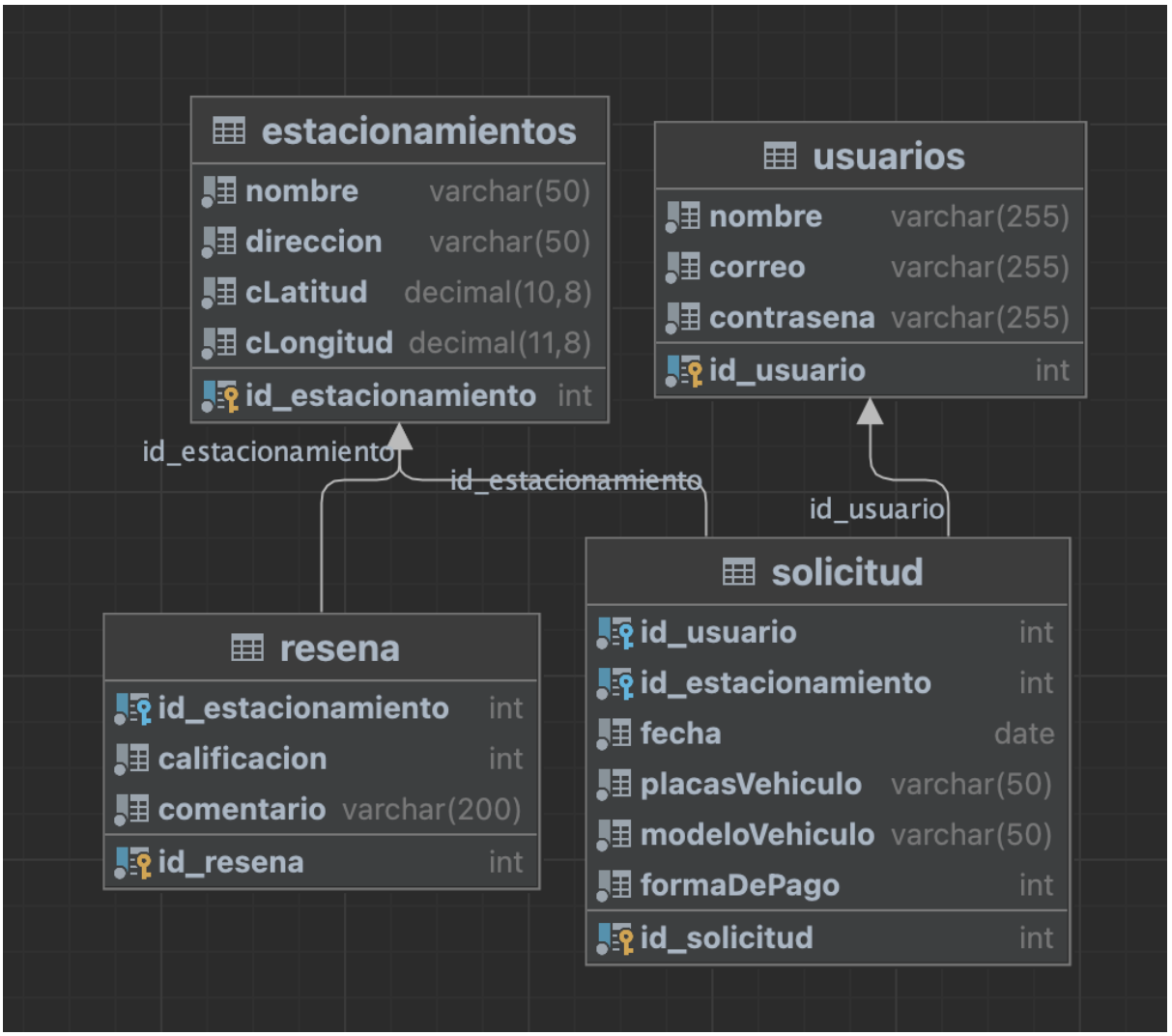
Requerimientos

Los requerimientos previos para el funcionamiento de la aplicación son:

- NodeJs
- MySQL

MySQL debe de estar corriendo en una instancia de terminal o como servicio.
Para habilitar la API se debe tener la instancia de MySQL. En la carpeta " rest-api", se encuentra un documento de variables de entorno *.env* que contiene los datos de usuario MySQL, y debe de cambiarse para generar el acceso personalizado al usuario local de la máquina. Dentro de la terminal interna de Mac, se debe de cargar la instancia dentro de la carpeta "rest-api" se debe ejecutar el comando *npm run dev,* y se tiene que esperar a que el módulo nodemon escriba en la consola "Escuchando en el puerto [número de puerto]".
La instancia de nodemon debe permanecer abierta en todo momento para que pueda escuchar las peticiones.

Base de datos MySQL



rest-api

La rest-api que utiliza la aplicación genera requests HTTP por medio de URLs que devuelven datos en formato de arreglo de JSON, y también puede escribir dentro de la base de datos MySQL.
La API se basa en las tecnologías de Express y está escrito en javascript puro. Se maneja en peticiones de get y push para intercambiar información con la app que se explica más a continuación.

Lottie

Lottie es una biblioteca de animación para iOS, Android y la web desarrollada por Airbnb. Esta biblioteca permite a los desarrolladores agregar animaciones vectoriales de alta calidad en sus aplicaciones utilizando JSON como formato de archivo.
En el contexto de Swift, Lottie se refiere a la biblioteca de animación Lottie para iOS, que se puede utilizar en proyectos de aplicaciones iOS escritos en el lenguaje de programación Swift. Los desarrolladores de Swift pueden agregar la biblioteca Lottie a sus proyectos y

utilizarla para agregar animaciones complejas y llamativas a sus aplicaciones sin tener que crearlas desde cero.
Lottie utiliza un formato de archivo JSON para definir las animaciones, lo que permite a los diseñadores y animadores trabajar con las herramientas de diseño que prefieran y a los desarrolladores incorporar fácilmente las animaciones en sus aplicaciones. La biblioteca Lottie también proporciona un conjunto de funciones para controlar la animación, como la capacidad de reproducir, detener o cambiar la velocidad de la animación.
En resumen, Lottie es una biblioteca de animación para iOS (y otras plataformas) que permite a los desarrolladores de Swift agregar animaciones vectoriales de alta calidad en sus aplicaciones utilizando JSON como formato de archivo.

Files .swift

Estacionamientos

Este es un ejemplo de código en Swift que incluye una estructura Park y una clase parking que actúa como un modelo de datos y utiliza una función getParks() para recuperar información de una API a través de una solicitud HTTP y decodificar los datos JSON de la respuesta en una matriz de objetos Park.
La estructura Park tiene propiedades que corresponden a las claves del JSON que se espera recibir de la API. También implementa el protocolo Identifiable y el protocolo Codable para permitir la identificación única de cada objeto Park y la decodificación de los datos JSON.
La clase parking utiliza una matriz observable parks que contiene los objetos Park recuperados de la API y se actualiza automáticamente cuando se reciben nuevos datos. El método getParks() envía una solicitud HTTP a la URL proporcionada y, si se recibe una respuesta exitosa, decodifica los datos JSON en objetos Park y los agrega a la matriz parks. Se utiliza un hilo de la interfaz de usuario para actualizar la matriz observable para que los cambios se reflejen en la interfaz de usuario.

File

Este código muestra la definición de una estructura Req y una clase DataPersistance. La estructura Req es identificable y codificable. Contiene las propiedades id_usuario, nombre_estacionamiento, fecha, placasVehiculo, modeloVehiculo y formaDePago.
La clase DataPersistance se encarga de manejar la persistencia de las solicitudes de reserva. La clase tiene dos métodos, saveReq y loadReq. El método saveReq toma un array de solicitudes y lo codifica como JSON antes de guardarlo en un archivo. El método loadReq carga las solicitudes previamente guardadas en el archivo, lo decodifica desde JSON y lo devuelve como un array de objetos Req.

LoginView

Este código define una estructura LoginView que muestra una pantalla de inicio de sesión para una aplicación móvil. La vista contiene dos campos de texto para introducir el correo electrónico y la contraseña, un botón "Iniciar sesión" y un botón "Iniciar sesión" que lleva a una pantalla de registro. También muestra un logotipo, una animación Lottie y un mensaje de error si el usuario introduce un correo electrónico o una contraseña incorrectos. El LoginView utiliza una clase de red para validar las credenciales del usuario llamando al

método getUsers y comprobando las propiedades correo y contrasena. Si el usuario está autenticado, la vista navega a la PrincipalView. El LoginView está incrustado en un NavigationView y presenta el PrincipalView de forma modal utilizando fullScreenCover. El código también incluye una estructura de vista previa para LoginView y MapView, que no se utiliza en el código.

LottieView

Este código define una vista personalizada llamada LottieView que se ajusta al protocolo UIViewRepresentable. Utiliza la biblioteca Lottie para mostrar animaciones en una vista SwiftUI.
El LottieView tiene tres parámetros:
name: una cadena que representa el nombre de la animación Lottie a mostrar.
loopMode: un LottieLoopMode que especifica cómo la animación debe repetirse.
animationSpeed: un CGFloat que representa la velocidad a la que debe reproducirse la animación.
La función makeUIView crea y devuelve una instancia de LottieAnimationView inicializada con el nombre de la animación, el modo de bucle y la velocidad proporcionados. También inicia la reproducción de la animación.

MapView

Este código define una estructura MapView que se ajusta al protocolo UIViewRepresentable en SwiftUI, permitiendo al MKMapView del framework MapKit ser utilizado en una vista SwiftUI.
La estructura MapView contiene varias propiedades, incluyendo una matriz de objetos ParkingLot, un enlace a un objeto ParkingLot seleccionado, un cierre que toma un objeto ParkingLot como entrada y no tiene valor de retorno, y un objeto CLLocationCoordinate2D para las coordenadas de Ciudad de México.
El método makeUIView crea una instancia de MKMapView y establece su ubicación inicial a la Ciudad de México, agrega anotaciones para los estacionamientos cercanos, y agrega un botón para ir a la ubicación del usuario. También establece el delegado de la vista del mapa a una instancia de la clase Coordinator, que se define dentro de la estructura MapView e implementa el protocolo MKMapViewDelegate para manejar los clics en las anotaciones del mapa.
El método updateUIView se ejecuta cada vez que se actualiza el enlace selectedParkingLot y actualiza la anotación del aparcamiento seleccionado en el mapa. También establece el modo de seguimiento del usuario de la vista del mapa para seguir la ubicación del usuario.
Por último, el método makeCoordinator crea una instancia de la clase Coordinator, pasando self como parámetro.

Network

Se trata de código Swift que define dos entidades: una estructura de usuario y una clase de red.

La estructura Usuario se ajusta a los protocolos Identificable y Codificable. Tiene cuatro propiedades: id_usuario, nombre, correo y contrasena. La propiedad id_usuario es de tipo Int, mientras que las otras son de tipo String. También tiene un enum CodingKeys anidado que mapea las propiedades a sus correspondientes claves en una representación JSON de un objeto Usuario. La propiedad id se inicializa a una instancia UUID.
La clase network se ajusta al protocolo ObservableObject, que le permite publicar cambios en sus propiedades en cualquier vista suscrita. Tiene dos propiedades publicadas: users, un array de objetos User, y correo, una cadena que representa la dirección de correo electrónico de un usuario. También tiene un método llamado getUsers, que toma un parámetro de correo y un parámetro de contraseña, construye una URL usando esos parámetros, y envía una petición GET a esa URL. Si la petición tiene éxito (es decir, si el código de estado de la respuesta es 200), el método decodifica los datos de la respuesta en un array de objetos User y los asigna a la propiedad users. También asigna a la propiedad correo la dirección de correo electrónico del primer usuario del array.

ParkingLotDetailView

Esta es una vista SwiftUI que muestra detalles sobre un aparcamiento y permite al usuario solicitar una plaza de aparcamiento en ese aparcamiento.
La vista toma un objeto ParkingLot y un cierre onRequestParkingSpot que será llamado cuando el usuario solicite una plaza de aparcamiento.
En la vista, se muestra el nombre y la dirección del aparcamiento junto con un botón que dice "Solicitar lugar". Cuando el usuario pulsa el botón, se llama al cierre onRequestParkingSpot con el objeto ParkingLot.
En general, esta vista proporciona una forma sencilla y directa para que el usuario solicite una plaza de aparcamiento en un aparcamiento determinado.

ParkingLotDetailViewWithDetail

Esta vista es similar a ParkingLotDetailView, pero añade un botón para navegar a una ReviewFormView donde el usuario puede escribir una reseña para el aparcamiento.
Las variables @State showReviewForm, rating y comment se declaran para gestionar el estado del formulario de revisión. showReviewForm no se utiliza en esta vista, pero se podría utilizar para mostrar u ocultar thcrm.
El NavigationLink se utiliza para navegar a la ReviewFormView cuando el usuario pulsa el botón "Escribir reseña". La calificación y el comentario se pasan a la ReviewFormView utilizando un Binding para que cualquier cambio realizado en esa vista se propague de nuevo a esta vista.

ParkingModel

La estructura ParkingLot representa la ubicación de un aparcamiento y contiene las siguientes propiedades:
id: Un identificador único para el aparcamiento, representado como UUID.
name: Nombre del aparcamiento.
address (dirección): La dirección del aparcamiento.

coordenada: Las coordenadas geográficas del aparcamiento, representadas como una estructura CLLocationCoordinate2D.
La propiedad coordinate se utiliza para mostrar la ubicación del aparcamiento en un mapa. Esta estructura no incluye información sobre el número de plazas disponibles o el coste del aparcamiento.

PrincipalView

Este código define la vista principal de una aplicación de búsqueda de aparcamientos. Incluye una vista de mapa que muestra los aparcamientos disponibles, y una vista detallada que muestra información sobre un aparcamiento seleccionado.
La estructura PrincipalView se ajusta al protocolo View y muestra una NavigationView que contiene una MapView y una ParkingLotDetailViewWithReview view, envueltas en una ZStack. El MapView es responsable de mostrar los aparcamientos en el mapa, y cuando un usuario pulsa sobre un aparcamiento, establece la propiedad selectedParkingLot, activando la visualización de la vista ParkingLotDetailViewWithReview.
La estructura PrincipalView también incluye una función loadEstacionamientos, que carga los datos de los aparcamientos desde un objeto ObservableObject. Esta función rellena una matriz de objetos ParkingLot con datos del objeto parkings, que luego se utiliza para mostrar los aparcamientos en el mapa.
La función generateRandomParkingLots está comentada y no se utiliza actualmente. Genera un conjunto de aparcamientos aleatorios y los añade a la matriz parkingLots.
La estructura PrincipalView también incluye elementos de barra de navegación en los bordes anterior y posterior. El elemento principal es un botón circular con el icono de un coche, y el elemento secundario es un enlace de navegación que lleva al usuario a una vista de reservas.

RegisterView

Este es un archivo Swift que contiene una vista SwiftUI llamada RegisterView. La vista muestra un formulario para registrar un usuario e incluye campos para nombre, email, contraseña y confirmar contraseña, junto con un botón para enviar el formulario. La vista también incluye una función llamada registrarUsuario() que envía una petición POST a una URL con el email y la contraseña del usuario y actualiza una variable de mensaje con el resultado de la petición. Las vistas previas se generan con RegisterView() como contenido.

resenas.Swift

Este código define una estructura Rtng conforme a los protocolos Identificable y Codificable, con cuatro propiedades: id_resena, id_estacionamiento, calificacion, y comentario.
También define una clase Ratings que se ajusta al protocolo ObservableObject y tiene una propiedad @Published ratings que es un array de objetos Rtng.
El método init de la clase Ratings llama al método getRatings() que realiza una petición GET a la URL especificada (http://localhost:3000/resenas/) y decodifica los datos de respuesta en una matriz de objetos Rtng, que luego se asigna a la propiedad ratings de la clase.

El método getRatings() hace uso de la clase URLSession para realizar la petición GET de forma asíncrona y actualiza la propiedad ratings en el hilo principal una vez que los datos de la respuesta han sido decodificados con éxito.

ReservationsView

Esta es una vista SwiftUI que muestra una lista de reservas. Tiene una vista de navegación y una VStack que contiene una cabecera y una lista de reservas.
La envoltura de la propiedad @State se utiliza para almacenar una matriz de objetos Req, que representa las reservas.
La vista comprueba si la matriz de reservas está vacía y, si lo está, muestra un mensaje al usuario. Si la matriz no está vacía, muestra una lista de reservas utilizando una vista de lista. Cada reserva se muestra como un botón, y cuando se pulsa el botón, la reserva correspondiente se elimina de la matriz.
La vista también utiliza DataPersistance para cargar y guardar los datos de las reservas. El modificador onAppear se utiliza para cargar los datos cuando aparece la vista, y el modificador onDisappear se utiliza para guardar los datos cuando desaparece la vista.

ReviewFormView

Esta es una vista SwiftUI que presenta un formulario para que los usuarios envíen comentarios. Toma dos parámetros Binding, uno para la valoración (un entero entre 1 y 5) y otro para el comentario (una cadena).
La vista presenta un título ("¡Reseña!") y una fila de cinco estrellas, donde el usuario puede tocar para seleccionar la valoración. Las estrellas seleccionadas aparecen resaltadas en cian, mientras que las no seleccionadas aparecen en gris. Debajo de las estrellas, hay un TextEditor donde el usuario puede introducir el comentario.
En la parte inferior de la vista, hay un botón "Enviar" que el usuario puede pulsar para enviar la valoración. Cuando se pulsa el botón, la vista se cierra y muestra una alerta para confirmar que la opinión se ha enviado correctamente.
El modificador de alerta se utiliza para mostrar el mensaje de confirmación como una alerta. Cuando la variable showConfirmationMessage se establece en true, se presenta la alerta. Cuando el usuario pulsa el botón "Aceptar" en la alerta, la variable showConfirmationMessage se establece de nuevo en false, descartando la alerta.
En general, esta vista proporciona una interfaz sencilla e intuitiva para que los usuarios envíen valoraciones, con información en tiempo real sobre la valoración seleccionada y un mensaje de confirmación tras el envío.

ViewCofirmation

Este código define una vista SwiftUI llamada ViewConfirmation, que muestra un mensaje de confirmación después de que un usuario envía un formulario para reservar una plaza de aparcamiento. La vista recibe varios parámetros, incluyendo nombreUsuario (nombre de usuario), placasCoche (número de matrícula del coche), modeloCoche (modelo de coche), formaPago (forma de pago), y nombreEstacionamiento (nombre de la plaza de

aparcamiento). También utiliza un objeto DataPersistance para guardar los datos de la solicitud en un almacenamiento persistente.
La vista muestra al usuario los datos introducidos y le permite aceptar el mensaje de confirmación pulsando el botón "Aceptar". Al pulsar este botón, se añade un nuevo objeto Req a un array llamado dataReq, que se guarda en el almacenamiento persistente mediante el objeto DataPersistance. Una vez completada la operación de guardado, la vista se cierra a sí misma utilizando la variable de entorno de presentación.
La vista también incluye un bloque onAppear que carga la matriz dataReq desde el almacenamiento persistente, de modo que cualquier solicitud enviada previamente se mostrará en el mensaje de confirmación. También hay un NavigationLink comentado que podría utilizarse para navegar de vuelta a la PrincipalView después de que se acepte el mensaje de confirmación, aunque actualmente está oculto.

ViewPark

Este código define la vista ViewPark, que es un formulario que permite al usuario introducir información personal y del vehículo, así como seleccionar un método de pago, y luego navegar a la vista ViewConfirmation para confirmar la reserva.
La vista contiene un VStack que incrusta un Formulario, el cual se compone de tres secciones: "Datos personales", "Datos del vehículo" y "Forma de pago".
La sección "Datos personales" contiene un TextField para introducir el nombre del usuario, mientras que la sección "Datos del vehículo" contiene dos TextFields para introducir la matrícula y el modelo del vehículo. La sección "Forma de pago" contiene un Picker que permite al usuario seleccionar una forma de pago entre tres opciones: "Efectivo", "Tarjeta de crédito" y "PayPal".
Por último, la vista contiene un NavigationLink que navega a la vista ViewConfirmation, pasando como parámetros la información personal y del vehículo del usuario y la forma de pago.

ContentView

Este código define el ContentView principal de la aplicación SwiftUI, que simplemente muestra el LoginView.
El LoginView es la primera vista que el usuario ve cuando se inicia la aplicación. Contiene un formulario donde el usuario puede introducir su nombre de usuario y contraseña. Si el usuario introduce credenciales válidas y pulsa el botón "Login", la aplicación pasa a la PrincipalView, que muestra una lista de aparcamientos donde el usuario puede aparcar su coche.
Si, por el contrario, el usuario pulsa el botón "Registrarse", la aplicación pasa a RegisterView, donde el usuario puede crear una nueva cuenta.
En general, este código representa el punto de entrada de la aplicación y establece la navegación entre las diferentes vistas.

HackatonV02App

La Hackaton_V02App es el punto de entrada principal de la aplicación. Se ajusta al protocolo App y define la escena principal de la aplicación, que en este caso es una instancia de ContentView envuelta en un WindowGroup.
La propiedad body de la Hackaton_V02App devuelve una Scene, que representa la interfaz principal de la aplicación. El WindowGroup es un contenedor para una única vista que llena toda la pantalla. En este caso, contiene una instancia de ContentView, que sirve como vista raíz de la aplicación.
Cuando se lanza la aplicación, se crea la instancia Hackaton_V02App y su propiedad body se utiliza para configurar el estado inicial de la aplicación. El WindowGroup se crea con una instancia de ContentView como vista raíz, y la escena resultante se muestra en la pantalla del dispositivo.