Plan It Together

Proyecto del Grado Superior: Desarrollo de Aplicaciones MultiPlataforma

Santiago José Santos Costoya

I.E.S Portada Alta

¿Que es PlanItTogether?

Es una aplicación que permite a los usuarios organizar planes con otros usuarios.

- Permite registrar sitios a los que ir o que ya has visitado y puntuarlos.
- Permite añadir amigos
- Permite crear planes de tipo actividad o encuentro, para poder organizar planes tanto en físico como online.
- Permite recibir notificaciones cuando un plan se acerca

Aplicación Android

La aplicación de android se ha desarrollado utilizando las siguientes tecnologías y estrategias:

- **Jetpack Compose**
- **Retrofit**
- **OkHTTP**
- **ViewModel**
- **LiveData**
- **Navigator**
- **Repository Pattern**
- **HTTP Requests**
- **DataStore Preferences**
- **FireBase Cloud Messaging** (Notificaciones Push)

Backend

El backend se ha desarrollado en Ktor (Kotlin para servidores) e incluye las siguientes tecnologías/estrategias:

- **Routing**: Para establecer los distintos endpoints de la API
- **Kotlinx.Serialization**: Para manipular JSON
- **Content Negotiation**: Para convertir automáticamente los datos recibidos y enviados
- **Exposed ORM**: ORM que permite crear la base de datos y sus tablas siguiendo la estructura establecida en código y realizar queries en lenguaje de Kotlin
- **CORS**: Para configurar las conexiones entre orígenes
- **Authentication**: Permite manejar el encabezado de autenticación en las peticiones HTTP
- **Authentication JWT**: Permite generar y validar tokens JWT
- **Jakarta Mail**: Permite conectarse al servidor local de correo y mandar correos desde la API

¿Cómo se comunican las aplicaciones entre sí?

La comunicación se realiza mediante peticiones HTTP de la aplicación android al backend.

Cuando el usuario inicia sesión o se registra, se genera un token JWT, este token se utiliza en el resto de peticiones que realiza la aplicación mediante un interceptor de OkHTTP.

Todas las peticiones excepto login y registro deben ir autenticadas con un token JWT válido generado previamente por el servidor.

Las respuestas recibidas por la aplicación android, se mapean en una clase modelo auxiliar que permite manejar los códigos de estado, excepciones y el tipo de contenido.

¿Cómo está configurado el servidor?

El servidor se ejecuta en una VPS, registrado como un servicio systemd que se ejecuta constantemente.

La base de datos que utiliza el servidor es MariaDB.

Mediante un proxy reverso configurado en nginx, todas las peticiones a la API (https://planit-api.santisc.es) se redirigen automáticamente al servidor, y lo mismo pasa con las respuesta del servidor.

Además, el servidor ejecuta constantemente una co-rutina de Kotlin, que comprueba cada 30 minutos los planes que ocurrirán entre la hora de ejecución a media hora después.

¿Qué dificultades y retos se han encontrado?

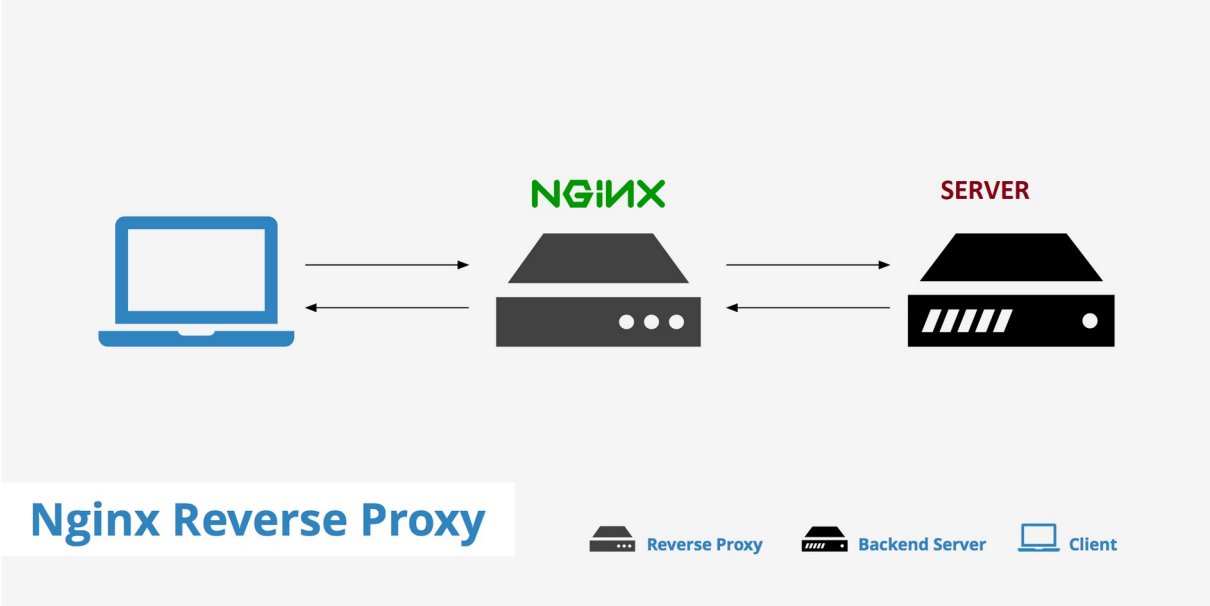
Entre los retos más destacables de la aplicación podemos destacar:

- Construir un servidor desde cero
- Realizar las consultas SQL adecuadas uniendo tablas y devolviendo los datos necesarios
- Manejar los estados correctamente en Jetpack Compose
- Manejar los códigos de estado de las peticiones para poder mostrar mensajes de error en la aplicación Android
- Mandar notificaciones push con firebase, entender el proceso de configuración en la aplicación Android y el recibo de notificaciones

Web

The server is running in a VPS, set up as a systemd service.

When a request is made to https://planit-api.santisc.es, the nginx reverse proxy redirects the requests to that URL towards the server and does the same with the responses.



End Points

END POINT	DESCRIPTION	TYPE	FILE	URL PARAMETERS	BODY	REQUIRES JWT AUTHENTICATION
/users	Register	POST	Routing.kt	-	email, password	❌
/login	Login	POST	Routing.kt	-	email, password	❌
/users/{id}	Get user information	GET	Routing.kt	ID	-	✔
/users/id	Update an user	PUT	Routing.kt	ID	userName, email, password	✔
/users/id	Delete an user	DELETE	Routing.kt	ID	-	✔
/plans	Add a plan	POST	PlansRouting.kt	-	title, type, users, place, date, time	✔
/plans/{id}	Get plan information	GET	PlansRouting.kt	ID	-	✔
/plans/{id}/ participants	Get all plan's participants	GET	PlansRouting.kt	ID	-	✔
/user/{id}/ plans	Get all user's plans	GET	PlansRouting.kt	ID	-	✔
/plans/{id}	Update a plan	PUT	PlansRouting.kt	ID	title, type, users, place, date, time	✔
/plans/{id}/ leave	Leave a plan	POST	PlansRouting.kt	ID	-	✔
/plans/{id}	Delete a plan	DELETE	PlansRouting.kt	ID	-	✔

Aplicación Móvil