C.S. DAM 2022/2024 (PFC)

Santiago Fernández Seoane

JPARTY:

Índice

Descripción del proyecto y ámbito de aplicación

JustParty es una aplicación Android que ofrece una solución integral y ahorra tiempo para cualquier persona interesada en descubrir y/o participar en eventos de ocio y entretenimiento en su área cercana, ya sean residentes habituales o visitantes temporales. La aplicación está diseñada para una audiencia diversa, que incluye amantes de la música, entusiastas de la vida nocturna y aquellos que buscan experiencias sociales y culturales en su entorno.

La motivación para su desarrollo surge de la necesidad de una aplicación que satisfaga las diversas necesidades de los usuarios en este ámbito y les haga ahorrar tiempo en su día. Desde aquellos que buscan información detallada sobre conciertos, actuaciones, hasta aquellos interesados en eventos locales más informales, fiestas de pueblo etc...

En un futuro me gustaría implementar un servicio de localización más óptimo con el uso de GPS para enseñarte de manera más personalizada locales cerca tuyos.

- ### Tecnologías utilizadas

  - Lenguajes de programación:
    a. Java: Utilizado para el desarrollo de la aplicación Android.
    b. Python: Empleado en el backend del API Rest utilizando Django.
  - Frameworks y Bibliotecas:
    c. Android SDK: Utilizado para el desarrollo de la interfaz de usuario en la aplicación móvil.
    d. Django REST : Empleado para la creación del API REST en el backend.
  - Base de Datos:
    e. SQLite: Utilizado en conjunto con el API REST como BBDD.
  - Entorno de Desarrollo Integrado (IDE):
    f. Android Studio: Utilizado para el desarrollo de la aplicación móvil.
    g. Visual Studio Code: Empleado para el desarrollo del backend.
  - Control de Versiones:
    h. Git: Utilizado para el control de versiones del código fuente.
  - Documentación:
    i. Swagger (OPENAPI): Utilizado para la documentación del API.
    j. Balsamiq Wireframes: Utilizado para los mockups de la APP.

Temporalización del proyecto y fases de desarrollo

Lo siguiente es la tabla con las temporalización del proyecto que nos servirá para llevar acabo tanto el Diagrama PERT como el GANTT.

ACTIVIDAD	DESCRIPCIÓN	INICIO DE LA ACTIVIDAD	DÍAS DE DURACIÓN	PRECEDENCIA	FINAL DE LA ACTIVIDAD
Fase 1: Preparación del proyecto					
A.Investigación y Planificación	Planificación inicial, investigación de requisitos y definición de proyecto.	18/03/2024	3	-	21/03/2024
Fase 2: Documentación					
B.Wireframes	Creación de los wireframes de la aplicación.	21/03/2024	5	A	27/03/2024
C.Documentación API	Creación de la documentación del API REST con sus endpoints en Swagger.	27/03/2024	4	B	31/03/2024
Fase 3: Desarrollo					
D.Django API Rest	Desarrollo de los endpoints, los más relevantes: /user/session (Login y Logout de usuarios). /user (Registro y gestión de usuarios) /events (Obtención y creación de eventos siendo manager) /userPreferences (Gestión de preferencias de usuario)	31/03/2024	8	C	08/04/2024

E. Testing Endpoints con Postman	Testeos realizadas con peticiones postman la ayuda de Postman para la finalización de los endpoints.	08/04/2024	1	D	09/04/2024
F. Android Front-End	Creación a partir de los wireframes, los XML de las respectivas pantallas en Android Studio	08/04/2024	6	D, E	15/04/2024
G. Android Back-End	Desarrollo del backend de respectiva pantalla	15/04/2024	35	F	20/05/2024
Fase 4: Testing y Depuración					
H. API REST: Automatización de Tests.	Creación de la automatización para el TESTING para los endpoints más importantes del API REST.	15/05/2024	5	F	20/05/2024

- **PERT**

(DIAGRAMA)

- ## GANT

- ## Retos encontrados

1. Hacer funcionar correctamente la pantalla de /events con o sin filter y /ownevents ya que usan el mismo endpoint para la petición GET pero diferentes query params.

2. El XML de AssistEvents me costó que quedase como tenía pensado en principio en el swagger, pero al final quedo incluso mejor.

Recursos de software y hardware

A continuación detallaré los requisitos necesarios de software y hardware.

- ### Requisitos de software

  - Sistema Operativo: Android 5.0 (Marshmallow) o superior.
  - Entorno de Desarrollo Integrado (IDE): Android Studio JellyFish o superior para el desarrollo de la aplicación Android.
  - SDK de Android: Última versión del Android SDK compatible con Android Studio.
  - Java Development Kit (JDK): JDK 8 o superior.
  - Bibliotecas y Dependencias:

    - **Android Studio**:
      - `glide:4.12.0`
      - `volley:1.2.1`
      - `navigation-ui:2.7.7`
      - `navigation-fragment:2.7.7`
    - **Django**:
      - `bcrypt`
      - `django-admin`

  - Navegador Web: Se recomienda tener instalado un navegador web moderno y compatible para acceder a la documentación y recursos en línea durante el desarrollo.

- ### Requisitos de hardware

  - **Requisitos Mínimos**:

    Para los requisitos mínimos, se requiere un procesador compatible con arquitectura ARM o x86, al menos 2 GB de memoria RAM, 500 MB de espacio en disco,y dispositivos con Android 5.0 (Marshmallow) o superior.

- **Requisitos Recomendados**:

  Por otro lado, para un rendimiento óptimo, se recomienda un procesador de cuatro núcleos con velocidad mínima de 2 GHz, 4 GB o más de RAM, al menos 1 GB de espacio en disco, y dispositivos con al menos 2 GB de RAM y procesador de cuatro núcleos.

Arquitectura de software y de sistemas

A continuación detallaré los requisitos necesarios de software y hardware.

- ### Arquitectura de software

  Clases relevantes y relaciones:

  - **User**: Representa a un usuario en la aplicación. Contiene los atributos email, username, password, province, birthdate, y manager. Tiene relaciones con otras clases como UserSession, UserPreferences, UserAssist, y UserLikes.

  - **UserSession**: Modela la sesión de un usuario, con un token asociado. Relacionada con la clase User.

  - **UserPreferences**: Almacena las preferencias musicales de un usuario. Relacionada con las clases User y MusicGenre.

  - **MusicGenre**: Representa un género musical en la aplicación, incluyendo su name e image. Relacionada con las clases UserPreferences y Events.

  - **Events**: Representa un evento en la aplicación, incluyendo su title, street, province, music_genre, price, secretkey, link, date, image, y description. Relacionada con las clases User (como manager), MusicGenre, UserAssist, y UserLikes.

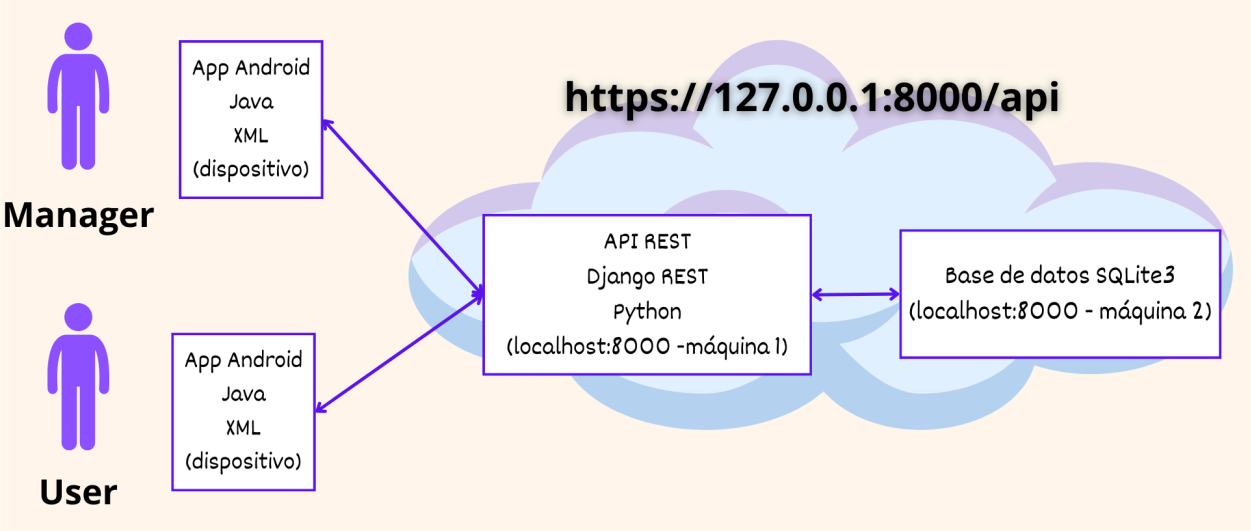
  - **UserLikes**: Almacena los eventos que un usuario ha marcado como "me gusta". Relacionada con las clases User y Events.

- **UserAssist**: Almacena los eventos a los que un usuario ha asistido. Relacionada con las clases User y Events.

Interfaz y patrones de diseño:

Estoy empleando el patrón Modelo-Vista-Controlador (MVC). Esto significa que tengo una estructura clara que separa la lógica de negocio, la presentación de la interfaz de usuario y la gestión de las solicitudes del usuario. En el backend de Django, los modelos (models.py) representan la estructura de los datos y la lógica de negocio, mientras que las vistas (user_endpoints.py o event_endpoints.py) manejan las solicitudes HTTP y coordinan las respuestas.

- ## Arquitectura de sistemas



Descripción de datos

1. Función de creación de eventos (POST en /events)

Descripción:

Esta función permite a los usuarios con permisos de administrador (managers) crear nuevos eventos en la plataforma. La función se encarga de autenticar al usuario, validar y procesar los datos de entrada, y guardar el evento en la base de datos.

Tipos de datos:

- title, street, province, link, image, description (tipo string)
- music_genre (tipo MusicGenre)
- price (tipo decimal)
- secretkey (tipo string, opcional)
- date, time (tipo string que se convierte a datetime)

Flujo de ejecución:

- Autenticar al manager.
- Extraer y validar datos del cuerpo de la solicitud.
- Convertir y formatear la fecha y hora.
- Crear y guardar el objeto Event en la base de datos.

2. Función para obtener eventos (GET en /events)

Descripción:

Esta función permite a los usuarios obtener eventos, filtrándose por defecto por género musical y además pudiendo filtrar por provincia y ordenándolos por fecha o precio.

Tipos de datos:

- sort_province, order_by, mine (tipo string)
- events (tipo QuerySet)
- json_response (tipo list de diccionarios)

Flujo de ejecución:

- Autenticar al usuario.
- Validar parámetros de consulta.
- Filtrar eventos según las preferencias del usuario y parámetros de consulta.
- Ordenar los eventos si se especifica.
- Crear respuesta en formato JSON con los datos de los eventos.