MÓDULO 1(anexos):

Arquitectura de Software: Buses_app

iteración 1

Requerimientos teóricos para el diseño de una aplicación de venta de billetes de autobús basándonos en la ISO 12207 (Ingeniería de Sistemas y Software). Esta norma nos proporciona un marco para la gestión del ciclo de vida del software, incluyendo la definición de requerimientos.

Consideraciones Generales:

- **Alcance:** Esta aplicación estará destinada a usuarios finales para la compra de billetes, y posiblemente a administradores para la gestión de rutas, horarios y precios.
- **Contexto:** Considerar el entorno en el que se utilizará la aplicación (móvil, web, quiosco), la infraestructura de red disponible, y la integración con sistemas existentes de la empresa de autobuses (si los hay).
- **Partes Interesadas:** Identificar a los usuarios finales, la empresa de autobuses, los desarrolladores, y cualquier otro grupo que pueda verse afectado por la aplicación.

Clasificación de Requerimientos según ISO 12207:

Para organizar mejor los requerimientos, los clasificaremos en varias categorías, siguiendo los principios de la ISO 12207.

1. Requerimientos del Sistema (High-Level Requirements):

- **Funcionalidad Principal:**
    - El sistema debe permitir a los usuarios buscar billetes de autobús por origen, destino, fecha y hora.
    - El sistema debe mostrar los resultados de la búsqueda, incluyendo precios, horarios y disponibilidad de asientos.
    - El sistema debe permitir a los usuarios seleccionar asientos específicos (si aplica).
    - El sistema debe permitir a los usuarios realizar el pago del billete utilizando diferentes métodos de pago (tarjeta de crédito/débito, billeteras electrónicas, etc.).
    - El sistema debe generar y enviar una confirmación de la compra al usuario (por correo electrónico o dentro de la app).

    - El sistema debe permitir a los usuarios acceder al billete electrónico desde la aplicación.
    - El sistema debe permitir la gestión de perfiles de usuario (creación, modificación, eliminación).
    - El sistema debe permitir la gestión de rutas, horarios y precios por parte de los administradores (panel de administración).
    - El sistema debe permitir la gestión de descuentos y promociones (panel de administración).
    - El sistema debe generar informes de ventas y ocupación (panel de administración).
- **Rendimiento:**
    - El sistema debe responder a las consultas de búsqueda en un tiempo razonable (ej. menos de 3 segundos).
    - El sistema debe soportar un número determinado de usuarios concurrentes (ej. 1000 usuarios simultáneos).
    - El sistema debe ser capaz de procesar un número determinado de transacciones por hora (ej. 500 transacciones/hora).
- **Seguridad:**
    - El sistema debe proteger la información personal y financiera de los usuarios.
    - El sistema debe cumplir con las normativas de protección de datos (ej. GDPR).
    - El sistema debe prevenir el acceso no autorizado a la información de los administradores.
- **Interfaces:**
    - El sistema debe tener una interfaz de usuario intuitiva y fácil de usar.
    - El sistema debe ser compatible con diferentes dispositivos (smartphones, tablets, ordenadores).
    - El sistema debe ser compatible con diferentes navegadores web (si aplica).
    - El sistema debe integrarse con sistemas de pago seguros.
    - El sistema debe integrarse con el sistema de gestión de la empresa de autobuses (si existe).
- **Restricciones:**
    - El sistema debe cumplir con las restricciones presupuestarias del proyecto.
    - El sistema debe desarrollarse dentro de un plazo determinado.

    - El sistema debe ser compatible con la infraestructura tecnológica existente de la empresa de autobuses.
- **Atributos de Calidad**
    - Disponibilidad: La aplicación debe estar disponible el 99.9% del tiempo.
    - Usabilidad: La aplicación debe ser fácil de usar para usuarios de diferentes niveles de experiencia tecnológica.
    - Mantenibilidad: El código debe estar bien documentado y modularizado para facilitar las actualizaciones y correcciones.
    - Portabilidad: La aplicación debe ser portable a diferentes sistemas operativos (iOS, Android, Web).

2. Requerimientos de Software (Detailed Requirements):

Una vez que se tienen los requerimientos del sistema, se deben detallar los requerimientos de software, dividiéndolos en:

- **Requerimientos Funcionales:** Describe cada funcionalidad en detalle. Por ejemplo:
    - "El sistema debe permitir al usuario filtrar los resultados de la búsqueda por precio, duración del viaje y número de escalas."
    - "El sistema debe mostrar un mapa con la ubicación de las estaciones de origen y destino."
    - "El sistema debe permitir al usuario guardar sus datos de pago para futuras compras."
- **Requerimientos No Funcionales:** Detalla aspectos de calidad, rendimiento, seguridad, etc. Por ejemplo:
    - "El tiempo de carga de la página de resultados de búsqueda no debe exceder los 2 segundos."
    - "La aplicación debe utilizar encriptación de extremo a extremo para proteger la información de la tarjeta de crédito."
    - "La aplicación debe ser accesible para usuarios con discapacidades visuales, cumpliendo con las directrices WCAG."
- **Requerimientos de Interfaz:** Especifica los detalles de la interfaz de usuario.
    - "La aplicación debe utilizar un diseño responsive que se adapte a diferentes tamaños de pantalla."
    - "La aplicación debe utilizar una paleta de colores coherente con la identidad visual de la empresa de autobuses."
    - "La aplicación debe proporcionar mensajes de error claros y concisos."

- **Requerimientos de Datos:** Describe la estructura y el tipo de datos que se manejarán.
    - "El sistema debe almacenar información sobre las rutas, incluyendo origen, destino, horarios, precios y número de asientos disponibles."
    - "El sistema debe almacenar información sobre los usuarios, incluyendo nombre, correo electrónico, teléfono y historial de compras."
    - "El sistema debe almacenar información sobre las transacciones, incluyendo fecha, hora, importe y método de pago."

3. Requerimientos de Interfaces Externas:

- **Sistemas de Pago:** Especificar la integración con pasarelas de pago (ej. Stripe, PayPal).
- **API de Mapas:** Especificar la integración con servicios de mapas (ej. Google Maps) para mostrar la ubicación de las estaciones.
- **Sistema de la Empresa de Autobuses:** Detallar cómo se integrará la aplicación con el sistema central de la empresa (si existe) para obtener información sobre rutas, horarios y disponibilidad.

4. Requerimientos de Seguridad:

- **Autenticación:** Definir cómo se autenticarán los usuarios (ej. nombre de usuario y contraseña, autenticación de dos factores).
- **Autorización:** Definir los niveles de acceso para diferentes usuarios (ej. usuarios finales, administradores).
- **Protección de Datos:** Especificar las medidas de seguridad para proteger la información personal y financiera de los usuarios (ej. encriptación, anonimización).
- **Cumplimiento Normativo:** Asegurarse de que la aplicación cumple con las normativas de protección de datos (ej. GDPR, CCPA).

5. Requerimientos de Rendimiento:

- **Tiempo de Respuesta:** Definir los tiempos de respuesta aceptables para diferentes acciones (ej. búsqueda de billetes, proceso de pago).
- **Escalabilidad:** Definir la capacidad del sistema para manejar un aumento en el número de usuarios y transacciones.
- **Disponibilidad:** Definir el porcentaje de tiempo que el sistema debe estar disponible (ej. 99.9%).

Proceso de Obtención de Requerimientos:

- **Entrevistas:** Realizar entrevistas con las partes interesadas (usuarios finales, empresa de autobuses) para comprender sus necesidades y expectativas.

- **Talleres:** Organizar talleres con las partes interesadas para discutir y validar los requerimientos.
- **Análisis de Documentos:** Analizar documentos existentes (ej. manuales de usuario, especificaciones de sistemas existentes) para identificar requerimientos.
- **Prototipos:** Crear prototipos de la aplicación para validar los requerimientos con los usuarios.

Documentación de los Requerimientos:

Es crucial documentar todos los requerimientos de forma clara y concisa, utilizando un formato estandarizado (ej. una plantilla de requerimientos). Cada requerimiento debe ser:

- **Único:** Identificado con un ID único.
- **Verificable:** Se debe poder comprobar si el requerimiento se ha cumplido.
- **Rastreable:** Se debe poder relacionar el requerimiento con las necesidades del usuario y las funcionalidades del sistema.
- **Priorizado:** Se debe indicar la importancia del requerimiento (ej. alta, media, baja).

Ejemplo de Plantilla de Requerimiento:

ID	Tipo	Descripción	Prioridad	Criterio de Aceptación
REQ-001	Funcional	El sistema debe permitir al usuario buscar billetes por origen y destino.	Alta	El sistema muestra los resultados de la búsqueda con opciones de origen y destino disponibles.
REQ-002	No Funcional	El tiempo de respuesta de la búsqueda no debe superar los 2 segundos.	Media	El tiempo de respuesta promedio de la búsqueda es inferior a 2 segundos en pruebas con un volumen de datos representativo.

Consideraciones Adicionales:

- **Evolución:** Los requerimientos pueden cambiar a lo largo del ciclo de vida del proyecto. Es importante establecer un proceso de gestión de cambios para gestionar estos cambios de forma controlada.
- **Trazabilidad:** Mantener la trazabilidad entre los requerimientos, el diseño, el código y las pruebas.
- **Validación:** Validar los requerimientos con las partes interesadas para asegurarse de que son correctos y completos.

Al seguir un proceso estructurado para la definición y gestión de los requerimientos, basado en la ISO 12207, se aumenta significativamente la probabilidad de desarrollar una aplicación de venta de billetes de autobús que satisfaga las necesidades de los usuarios y de la empresa.

iteración 2

Vamos a planificar los sprints para el desarrollo del backend de la aplicación de venta de billetes de autobús, utilizando una arquitectura de microservicios, Python como lenguaje principal y con iteraciones de 1 semana. La estrategia será iterativa e incremental, priorizando el valor para el usuario lo antes posible.

Estrategia General:

1. **Usuarios y Autenticación (Sprints 1-2):** Establecer la base para el manejo de usuarios y la seguridad.
2. **Búsqueda y Catálogo (Sprints 3-4):** Permitir a los usuarios encontrar billetes.
3. **Carrito y Pago (Sprints 5-6):** Permitir a los usuarios seleccionar y pagar billetes.
4. **Gestión de Viajes y Billetes (Sprints 7-8):** Permitir a los usuarios ver sus billetes y a la empresa gestionarlos.
5. **Administración (Sprints 9-10):** Dotar a los administradores de las herramientas necesarias.

Suposiciones:

- Equipo de desarrollo con conocimientos en Python, microservicios y las tecnologías seleccionadas.
- Infraestructura de microservicios (ej. Kubernetes, Docker) ya establecida o en desarrollo paralelo.
- Existencia de una API para sistemas de pago o la definición de una.

Detalle de los Sprints (Centrado en Usuarios Inicialmente):

Sprint 1: Usuarios - Autenticación y Registro (Base)

- **Objetivo del Sprint:** Permitir a los usuarios registrarse y autenticarse en el sistema. Crear el microservicio fundamental de usuarios.
- **Funcionalidades a Desarrollar:**
    - API para registro de usuarios (nombre, email, contraseña).
    - API para autenticación de usuarios (login con email y contraseña).
    - Esquema de base de datos para almacenar la información básica de los usuarios.
    - Implementación de un sistema de hashing seguro de contraseñas (ej. bcrypt).
    - Implementación de JWT (JSON Web Tokens) para la autenticación.
    - Documentación de la API.
- **Criterios de Aceptación:**

    - Los usuarios pueden registrarse con éxito proporcionando un email y contraseña válidos.
    - Los usuarios pueden autenticarse con éxito utilizando las credenciales registradas.
    - La información de los usuarios se almacena de forma segura en la base de datos.
    - La API está documentada y puede ser utilizada por otras partes del sistema.
- **Tareas:**
    - Diseño del esquema de la base de datos de usuarios.
    - Desarrollo de las APIs de registro y autenticación.
    - Implementación del sistema de hashing de contraseñas.
    - Implementación de JWT.
    - Pruebas unitarias y de integración.
    - Documentación de la API.

Sprint 2: Usuarios - Perfiles y Seguridad Mejorada

- **Objetivo del Sprint:** Permitir a los usuarios gestionar su perfil y agregar funcionalidades de seguridad (recuperación de contraseña)
- **Funcionalidades a Desarrollar:**
    - API para gestionar el perfil del usuario (actualizar nombre, email, teléfono).
    - API para recuperar la contraseña (envío de email con enlace de restablecimiento).
    - Implementación de validaciones en la API (ej. formato de email, longitud de la contraseña).
    - Implementación de Rate Limiting para prevenir ataques de fuerza bruta.
    - Documentación de la API.
- **Criterios de Aceptación:**
    - Los usuarios pueden actualizar su perfil con éxito.
    - Los usuarios pueden recuperar su contraseña utilizando su email registrado.
    - La API valida los datos de entrada correctamente.
    - La API está protegida contra ataques de fuerza bruta.
- **Tareas:**
    - Desarrollo de la API de gestión de perfiles.
    - Desarrollo de la API de recuperación de contraseña.

- Implementación de validaciones en la API.
- Implementación de Rate Limiting.
- Pruebas unitarias y de integración.
- Documentación de la API.

Sprint 3: Búsqueda - Microservicio de Rutas y Horarios (Base)

- **Objetivo del Sprint:** Crear el microservicio que gestiona las rutas, horarios y disponibilidad, y permite a los usuarios buscar rutas.
- **Funcionalidades a Desarrollar:**
  - API para buscar rutas por origen, destino y fecha.
  - Esquema de base de datos para almacenar la información de las rutas y horarios.
  - Implementación de algoritmos de búsqueda eficientes.
  - API para que los administradores gestionen las rutas y horarios (inicialmente, solo crear y leer). Este backend de administración no estará expuesto al frontend todavía.
  - Documentación de la API.
- **Criterios de Aceptación:**
  - Los usuarios pueden buscar rutas utilizando el origen, destino y fecha.
  - La API devuelve resultados relevantes y precisos.
  - La API es escalable y puede manejar un gran volumen de búsquedas.
  - Los administradores pueden crear y consultar rutas y horarios.
- **Tareas:**
  - Diseño del esquema de la base de datos de rutas y horarios.
  - Desarrollo de la API de búsqueda de rutas.
  - Implementación de algoritmos de búsqueda eficientes.
  - Desarrollo de las APIs de gestión de rutas y horarios (para administradores, backend-only).
  - Pruebas unitarias y de integración.
  - Documentación de la API.

Sprint 4: Búsqueda - Filtrado y Ordenamiento

- **Objetivo del Sprint:** Ampliar las funcionalidades de búsqueda con filtros y ordenamiento.
- **Funcionalidades a Desarrollar:**

        - Implementación de filtros en la API de búsqueda (ej. por precio, duración del viaje, número de escalas).
        - Implementación de ordenamiento en la API de búsqueda (ej. por precio, duración del viaje).
        - Mejorar la eficiencia de la API de búsqueda.
        - Documentación de la API.
    - **Criterios de Aceptación:**
        - Los usuarios pueden filtrar los resultados de la búsqueda utilizando diferentes criterios.
        - Los usuarios pueden ordenar los resultados de la búsqueda utilizando diferentes criterios.
        - La API de búsqueda es eficiente y responde rápidamente.
    - **Tareas:**
        - Implementación de filtros en la API de búsqueda.
        - Implementación de ordenamiento en la API de búsqueda.
        - Optimización de la API de búsqueda.
        - Pruebas unitarias y de integración.
        - Documentación de la API.

Justificación de la Priorización:

- **Usuarios Primero:** Comenzar con los usuarios y la autenticación es fundamental para establecer la base del sistema. Sin usuarios registrados y autenticados, no se puede avanzar con otras funcionalidades.
- **Búsqueda Temprano:** La búsqueda es la funcionalidad principal de la aplicación. Permitir a los usuarios buscar rutas lo antes posible proporciona valor y permite obtener feedback temprano.
- **Iterativo e Incremental:** Cada sprint se centra en agregar valor de forma incremental, permitiendo adaptar el desarrollo en función del feedback y las necesidades del usuario.

Siguientes Sprints (Resumen):

- **Sprints 5-6: Carrito y Pago:** Implementar el microservicio de carrito, integrarlo con la API de búsqueda, y conectar con una pasarela de pago.

- **Sprints 7-8: Gestión de Viajes y Billetes:** Implementar el microservicio para gestionar los billetes (generación, validación), permitir a los usuarios ver sus billetes comprados y a los administradores gestionarlos.
- **Sprints 9-10: Administración:** Desarrollar las APIs completas para la gestión de rutas, horarios, precios, descuentos y la generación de informes para los administradores. Este microservicio se comunicaría con los otros microservicios (usuarios, rutas, billetes) para obtener la información necesaria.

Consideraciones Finales:

- **Comunicación:** Mantener una comunicación constante entre los miembros del equipo.
- **Refactorización:** Dedicar tiempo a la refactorización del código para mejorar la calidad y mantenibilidad.
- **Monitorización:** Implementar la monitorización de los microservicios para detectar y solucionar problemas de forma temprana.
- **Integración Continua y Despliegue Continuo (CI/CD):** Automatizar el proceso de construcción, prueba y despliegue de los microservicios.

Este es un plan de sprints inicial. Es importante ser flexible y adaptar el plan en función de los resultados de cada sprint, el feedback de los usuarios y las necesidades del negocio.

Iteración 3

Vamos a crear el árbol de directorios y ficheros inicial para los microservicios de la aplicación de venta de billetes de autobús, junto con los archivos README.md en cada directorio para documentar su propósito. Esta estructura es una base, y puede ser modificada y expandida a medida que el proyecto avance.

Estructura del Repositorio Git:

```
      bus-ticket-app/
├── README.md        (README general para todo el proyecto)
├── users-service/
│   ├── README.md      (README para el servicio de usuarios)
│   ├── src/
│   │   ├── __init__.py
│   │   ├── app.py       (Archivo principal de la aplicación)
│   │   ├── models.py    (Modelos de la base de datos)
│   │   ├── routes.py    (Rutas de la API)
│   │   ├── utils.py     (Funciones de utilidad)
│   │   ├── config.py    (Configuración de la aplicación)
│   │   ├── tests/
│   │   │   ├── __init__.py
│   │   │   ├── conftest.py (Configuración de Pytest)
│   │   │   ├── test_users.py (Tests para registro y autenticación de usuarios)
│   ├── requirements.txt (Dependencias de Python)
│   ├── Dockerfile       (Configuración de Docker)
│   ├── .gitignore       (Ficheros a ignorar en Git)
├── routes-service/
│   ├── README.md      (README para el servicio de rutas)
│   ├── src/
│   │   ├── __init__.py
│   │   ├── app.py
│   │   ├── models.py
│   │   ├── routes.py
│   │   ├── utils.py
│   │   ├── config.py
│   │   ├── tests/
│   │   │   ├── __init__.py
│   │   │   ├── conftest.py
│   │   │   ├── test_routes.py (Tests para la búsqueda y gestión de rutas)
│   ├── requirements.txt
│   ├── Dockerfile
│   ├── .gitignore
├── payments-service/   (Ejemplo de estructura, no estrictamente necesario para
los Sprints 1-4)
│   ├── README.md
│   ├── ...
├── common/             (Componentes reutilizables)
│   ├── README.md
│   ├── database.py     (Configuración de la conexión a la base de datos -
Ejemplo)
│   ├── auth.py         (Utilidades de autenticación - Ejemplo)
└── .gitignore          (Archivo .gitignore general)
```

Contenido de los Ficheros README.md:

bus-ticket-app/README.md (README General del Proyecto):

```
      # Aplicación de Venta de Billetes de Autobús

Este repositorio contiene los microservicios backend para una aplicación de
venta de billetes de autobús.

## Microservicios

*   **users-service:** Gestiona las cuentas de usuario, la autenticación y los
perfiles.
*   **routes-service:** Gestiona las rutas de autobús, los horarios y la
disponibilidad.
*   **payments-service:** (Marcador de posición) Gestiona el procesamiento de
pagos.
*   **common:** Contiene componentes reutilizables compartidos entre
microservicios.

## Empezando

Las instrucciones sobre cómo configurar y ejecutar la aplicación se añadirán más
adelante.
```

IGNORE_WHEN_COPYING_START

content_copy download

Use code with caution.Markdown

IGNORE_WHEN_COPYING_END

bus-ticket-app/users-service/README.md (README del Servicio de Usuarios):

```
      # Servicio de Usuarios

Este microservicio es responsable de gestionar las cuentas de usuario, la
autenticación y los perfiles de usuario.

## Funcionalidad

*   Registro y autenticación de usuarios.
*   Gestión del perfil (actualizar nombre, email, teléfono).
*   Recuperación de contraseña.

## Tecnologías

*   Python
*   Flask/FastAPI (Elige uno)
*   SQLAlchemy (o un ORM similar)
*   PostgreSQL (u otra base de datos)
*   JWT (JSON Web Tokens) para la autenticación

## Configuración

(Marcador de posición para las instrucciones de configuración)
```

IGNORE_WHEN_COPYING_START

content_copy download
Use code with caution.Markdown
IGNORE_WHEN_COPYING_END

bus-ticket-app/routes-service/README.md (README del Servicio de Rutas):

```
      # Servicio de Rutas

Este microservicio gestiona las rutas de autobús, los horarios y la
disponibilidad.

## Funcionalidad

*   Búsqueda de rutas por origen, destino y fecha.
*   Filtrado y ordenamiento de los resultados de la búsqueda.
*   (Admin) Gestión de rutas y horarios.

## Tecnologías

*   Python
*   Flask/FastAPI (Elige uno)
*   SQLAlchemy (o un ORM similar)
*   PostgreSQL (u otra base de datos)

## Configuración

(Marcador de posición para las instrucciones de configuración)
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.Markdown
IGNORE_WHEN_COPYING_END

bus-ticket-app/common/README.md (README de Componentes Comunes):

```
      # Componentes Comunes

Este directorio contiene componentes reutilizables compartidos entre múltiples
microservicios.

## Componentes

*   **database.py:** Configuración de la conexión a la base de datos (ejemplo).
*   **auth.py:** Utilidades de autenticación (ejemplo).

(Añade más descripciones a medida que añadas más componentes)
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.Markdown
IGNORE_WHEN_COPYING_END

Contenido de .gitignore (Ejemplo):

```
      # Ficheros compilados / optimizados / DLL
__pycache__/
*.py[cod]
*$py.class

# Entornos
.env
venv/
env/
ENV/
*.egg-info/

# Artefactos de construcción
dist/
build/
sdist/

# IDEs y Editores
.idea/
.vscode/
*.swp
*.swo
*~
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.
IGNORE_WHEN_COPYING_END

Ejemplo de users-service/requirements.txt:

```
      Flask
SQLAlchemy
psycopg2-binary  # O el controlador de base de datos apropiado
python-dotenv
Flask-JWT-Extended
bcrypt
```

IGNORE_WHEN_COPYING_START
content_copy download
Use code with caution.
IGNORE_WHEN_COPYING_END

Consideraciones Adicionales:

- **Framework:** Decidir entre Flask y FastAPI. FastAPI se recomienda generalmente para APIs modernas por su rendimiento y validación de datos incorporada.
- **Database:** Seleccionar una base de datos (PostgreSQL, MySQL, etc.).

- **ORM:** Elegir un ORM (SQLAlchemy, Peewee).
- **Testing:** Usar pytest como framework de testing.
- **Linting:** Configurar linters (flake8, pylint) para mantener la calidad del código.
- **CI/CD:** Integrar con un sistema de CI/CD (GitHub Actions, GitLab CI, Jenkins).
- **Asincronía:** Considerar el uso de asyncio si se requiere un alto rendimiento.
- **Configuración:** Usar variables de entorno para configurar la aplicación (ej. claves de API, URLs de bases de datos).

Cómo Proceder:

1. **Crear el repositorio Git:** Crea un nuevo repositorio en GitHub, GitLab o Bitbucket.
2. **Clonar el repositorio:** Clona el repositorio a tu máquina local.
3. **Crear la estructura de directorios y ficheros:** Crea la estructura de directorios y ficheros como se muestra arriba.
4. **Completar los archivos README.md:** Edita los archivos README.md con la información correspondiente.
5. **Crear los archivos .gitignore:** Crea los archivos .gitignore con las entradas apropiadas.
6. **Crear los archivos requirements.txt:** Lista las dependencias de cada microservicio en su respectivo archivo requirements.txt.
7. **Inicializar el código base:** Crea los archivos app.py, models.py, routes.py, etc., con el código inicial para cada microservicio.
8. **Realizar el primer commit:** Agrega todos los archivos al repositorio Git, realiza un commit con un mensaje descriptivo (ej. "Commit inicial - Estructura del proyecto").
9. **Subir al repositorio remoto:** Sube los cambios al repositorio remoto.

Este punto de partida te permitirá tener una estructura organizada y documentada para comenzar a desarrollar los microservicios de tu aplicación de venta de billetes de autobús. A partir de aquí, puedes comenzar a implementar la lógica de negocio en los archivos app.py, models.py y routes.py de cada microservicio.

Comando con la estructura. Es más legible ;-)

mkdir -p bus-ticket-app/{users-service/src/tests,routes-service/src/tests,payments-service,common}

touch bus-ticket-app/README.md bus-ticket-app/.gitignore

touch bus-ticket-app/users-service/README.md bus-ticket-app/users-service/.gitignore bus-ticket-app/users-service/Dockerfile bus-ticket-app/users-service/requirements.txt

touch bus-ticket-app/users-service/src/{__init__.py,app.py,models.py,routes.py,utils.py,config.py}

touch bus-ticket-app/users-service/src/tests/{__init__.py,conftest.py,test_users.py}

touch bus-ticket-app/routes-service/README.md bus-ticket-app/routes-service/.gitignore bus-ticket-app/routes-service/Dockerfile bus-ticket-app/routes-service/requirements.txt

touch bus-ticket-app/routes-service/src/{__init__.py,app.py,models.py,routes.py,utils.py,config.py}

touch bus-ticket-app/routes-service/src/tests/{__init__.py,conftest.py,test_routes.py}

touch bus-ticket-app/payments-service/README.md

touch bus-ticket-app/common/{README.md,database.py,auth.py}

https://github.com/ricardoinstructor/bus_app.git