
CORPORACION UNIVERSITARIA IBEROAMERICANA
FACULTAD DE INGENIERIA
INGENIERIA DE SOFTWARE

# ACTIVIDAD 6: APLICATIVO DE ARQUITECTURA DE SOFTWARE

AUTOR
DUBAN LEONARNO GUZMAN MEDINA
ID: 100126620

DOCENTE

JOAQUIN SANCHEZ

MAJAGUAL(SUCRE) – COLOMBIA

2024

**Introducción:**

La planificación de viajes es una tarea que requiere tiempo y organización, especialmente cuando se trata de visitar múltiples destinos, gestionar horarios y considerar diferentes modos de transporte. A pesar de la gran cantidad de aplicaciones relacionadas con viajes, muchas de ellas carecen de la flexibilidad para crear itinerarios personalizados que satisfagan las necesidades específicas de los usuarios. En este contexto, se propone el desarrollo de TripRoute, una aplicación destinada a facilitar la creación de itinerarios personalizados, la optimización de tiempos y el suministro de recomendaciones en función de intereses específicos. Esta aplicación no sólo permite organizar múltiples destinos de viaje, sino que también integra servicios externos como Google Maps y recomendaciones de transporte, brindando al viajero una experiencia completa y eficiente. El principal objetivo de TripRoute es simplificar la planificación de viajes, reducir el tiempo invertido y ofrecer una solución personalizada que se adapte a las preferencias de cada usuario.

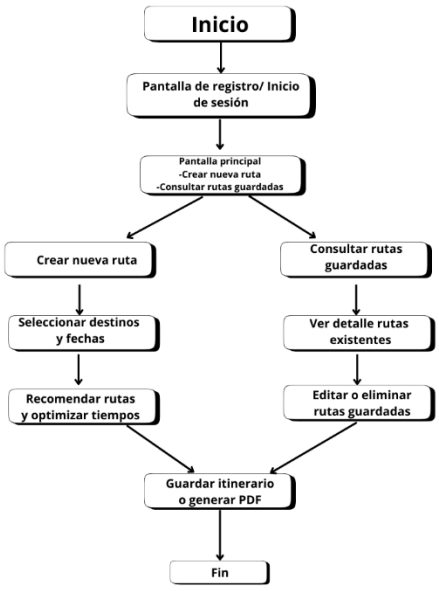
**Título del Proyecto.**

**TripRoute: Aplicación para Organizar Rutas de Viaje.**

- **Objetivo del documento**: El propósito de este trabajo es presentar una propuesta para el desarrollo de un software llamado TripRoute que permitirá a los usuarios organizar y planificar rutas de viaje de manera eficiente.

- **Contexto y necesidad**: La planificación de viajes puede resultar complicada cuando se trata de gestionar múltiples destinos, horarios y transporte. Muchos viajeros necesitan herramientas que les permitan optimizar su tiempo y organizar rutas en función de sus preferencias y necesidades. Actualmente, la mayoría de las aplicaciones existentes no tienen la flexibilidad de crear rutas personalizadas. Es por eso que propusimos el desarrollo de una aplicación intuitiva que satisfaga estas necesidades, ayudando a los viajeros a organizar sus viajes de una manera eficiente y atractiva.

**Diagrama de flujo.**



**Objetivos de la app.**

**Objetivo general:** Desarrollar una aplicación móvil que permita a los usuarios organizar itinerarios de viaje personalizados integrando destinos, horarios, medios de transporte y atracciones.

**Objetivos específicos:**

- ✓ Permita a los usuarios agregar y organizar múltiples destinos en una sola ruta.
- ✓ Proporcionar recomendaciones automáticas de rutas basadas en tiempos y preferencias del usuario.
- ✓ Agregue información relevante como horarios de transporte, mapas interactivos y clima.
- ✓ Ofrecer integración con servicios externos como Google Maps, servicios de transporte y recomendaciones de alojamiento.
- ✓ Le permite crear y almacenar planes de viaje sin conexión para obtener asesoramiento sin conexión.

**Metodología:**

Para el desarrollo de TripRoute se utilizará una metodología ágil basada en Scrum. El proyecto se dividirá en sprints de dos semanas, lo que permitirá un desarrollo incremental y adaptativo a los cambios que puedan surgir en función de los comentarios de los usuarios y las partes interesadas. Cada sprint incluirá planificación de tareas, desarrollo de software, validación y pruebas.

Las etapas principales del desarrollo incluyen:

1. Recolección de requisitos.
2. Prototipado y diseño de la interfaz.
3. Desarrollo del backend y frontend.

4. Integración con APIs externas.
5. Pruebas y ajustes.
6. Lanzamiento beta y ajustes finales.

**Herramientas a utilizar.**

**Lenguaje de programación:**

Interfaz: React Native (para garantizar la compatibilidad con Android e iOS).

Backend: Node.js

**Frameworks y Librerías:** Mapas e indicaciones: API de Google Maps, OpenStreetMap.

Recomendaciones y datos: API de TripAdvisor o Yelp para recomendaciones de atracciones.

**Base de datos:** MongoDB (NoSQL) para almacenar rutas y configuraciones de usuario.

**Herramientas de gestión de proyectos:** Jira para gestión de problemas y GitHub para control de versiones.

**Entorno de desarrollo:** Visual Studio Code.

**Descripción funcional del software.**

**Las características clave de TripRoute incluyen:**

**Creación de rutas personalizadas:** El usuario podrá añadir destinos, seleccionar fecha y hora de llegada y salida y el sistema calculará la mejor ruta para maximizar el tiempo disponible.

**Recomendaciones automáticas:** La aplicación sugiere rutas alternativas o destinos cercanos en función de los intereses del usuario.

**Mapas interactivos:** visualice la ruta planificada con opciones para mostrar atracciones, paradas y restaurantes cercanos.

**Sincronización de horarios:** Integración con servicios de transporte público para mostrar opciones de viaje en tiempo real.

**Notificaciones y alertas:** Recordatorios sobre el próximo destino, cambios climáticos o alertas de tráfico.

**Modo sin conexión:** posibilidad de descargar mapas y rutas para verlos sin conexión a Internet.

**Requisitos técnicos.**

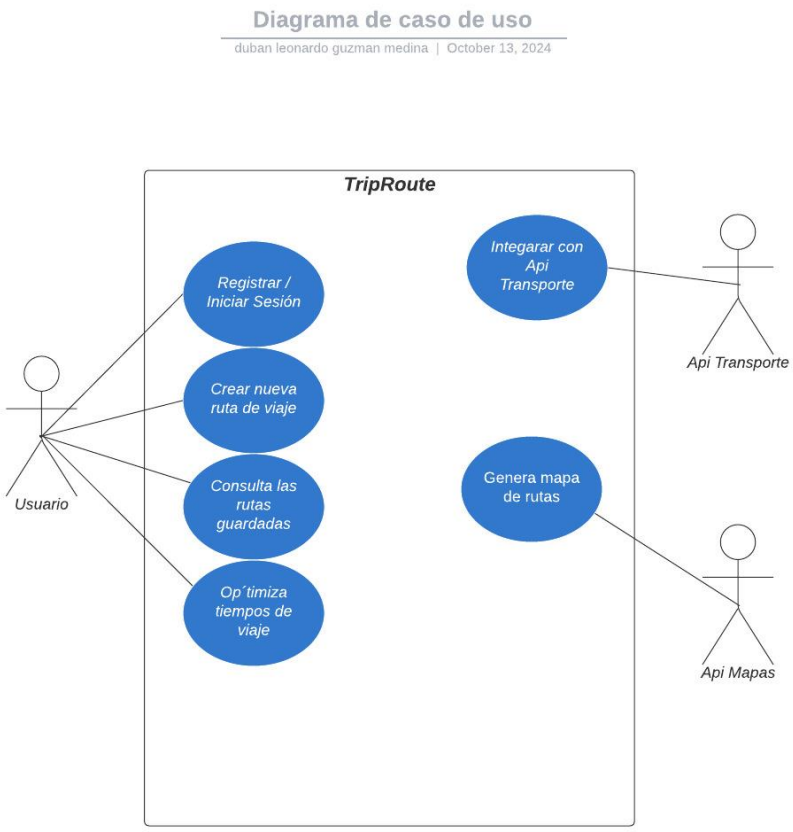
**Requisitos funcionales:**

- ✓ Permitir a los usuarios crear, editar y eliminar rutas.
- ✓ Sincronización de la aplicación con servicios de transporte, hotelería y meteorología en tiempo real.
- ✓ Mostrar tiempos de viaje estimados entre destinos.

**Requisitos no funcionales:**

- ✓ Seguridad: utilice cifrado de datos confidenciales del usuario.
- ✓ Rendimiento: la aplicación debería responder en menos de 2 segundos para todas las operaciones críticas.
- ✓ Escalabilidad: capacidad de admitir una gran cantidad de usuarios simultáneos.
- ✓ Plataformas de despliegue: Android e iOS (usando React Native) y versión web básica para consultas y sincronización.

**Diagrama de flujo.**



**Arquitectura propuesta: Arquitectura cliente-servidor.**

La aplicación TripRoute se implementará utilizando una arquitectura cliente-servidor, que es ideal para aplicaciones que requieren una interacción constante entre clientes (dispositivos móviles) y servidores que administran datos y lógica empresarial. Esta arquitectura separa claramente las responsabilidades entre el cliente, que es responsable de la interfaz de usuario, y el servidor, que realiza el procesamiento, almacenamiento y gestión de datos.

**¿Por qué utilizar la arquitectura cliente-servidor en TripRoute?**

**Optimización de recursos:** La lógica compleja, como la optimización de rutas o la integración con API de transporte, se ejecuta en el servidor, liberando al cliente de estos tediosos procesos. Esto mejora el rendimiento de la aplicación móvil, que puede centrarse en la interacción del usuario sin verse ralentizada por cálculos complejos.

**Actualización y mantenimiento centralizados:** La separación entre el cliente y el servidor facilita la actualización y el mantenimiento del sistema. Los servicios del servidor se pueden mejorar y optimizar sin necesidad de modificar la aplicación en el dispositivo del usuario.

**Seguridad centralizada:** La arquitectura cliente-servidor facilita el control centralizado de autenticación y autorización. La implementación de sistemas como OAuth y JWT garantiza que solo los usuarios autorizados puedan acceder a los datos y servicios.

**Mejorar la escalabilidad:** En aplicaciones de viajes como TripRoute, se espera que la cantidad de usuarios crezca con el tiempo. Esta arquitectura permite que el servidor escale de manera eficiente para manejar más solicitudes, mientras que el cliente sigue siendo liviano y receptivo en dispositivos móviles.

**Comunicación asincrónica y el resto de los datos:** El servidor puede manejar procesos asincrónicos, como optimización de enrutamiento o solicitudes de servicios externos, sin necesidad de que el cliente espere directamente. Esto garantiza una experiencia de usuario más fluida.

**Código Python.**

```
from flask import Flask, request, jsonify
from flask_jwt_extended import JWTManager, create_access_token, jwt_required
from pymongo import MongoClient
import bcrypt

# Crear la aplicación Flask
app = Flask(__name__)

# Configuración de clave secreta para JWT
app.config['JWT_SECRET_KEY'] = 'clave_super_secreta'  # Cambiar por una clave más segura en producción
jwt = JWTManager(app)

# Conectar a MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['triproute_db']  # Base de datos para la aplicación
users_collection = db['users']
routes_collection = db['routes']
```

```
# Función para registrar un nuevo usuario
@app.route('/register', methods=['POST'])
def register():
    data = request.json
    username = data['username']
    password = data['password']

    # Verificar si el usuario ya existe
    if users_collection.find_one({'username': username}):
        return jsonify({'message': 'Usuario ya registrado'}), 400

    # Encriptar contraseña
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

    # Guardar nuevo usuario en la base de datos
    users_collection.insert_one({'username': username, 'password': hashed_password})
    return jsonify({'message': 'Usuario registrado exitosamente'}), 201

# Función para inicio de sesión
@app.route('/login', methods=['POST'])
def login():
    data = request.json
    username = data['username']
    password = data['password']
```

```
    # Verificar si el usuario existe
    user = users_collection.find_one({'username': username})
    if not user or not bcrypt.checkpw(password.encode('utf-8'), user['password']):
        return jsonify({'message': 'Credenciales incorrectas'}), 401

    # Crear token JWT
    access_token = create_access_token(identity=username)
    return jsonify({'access_token': access_token}), 200

# Función para crear una nueva ruta de viaje
@app.route('/create_route', methods=['POST'])
@jwt_required()
def create_route():
    data = request.json
    username = data['username']  # Se obtiene del token, en la realidad
    route_name = data['route_name']
    destinations = data['destinations']  # Lista de destinos

    # Guardar la nueva ruta
    routes_collection.insert_one({
        'username': username,
        'route_name': route_name,
        'destinations': destinations
    })
    return jsonify({'message': 'Ruta creada exitosamente'}), 201
```

```
# Función para consultar rutas guardadas
@app.route('/get_routes', methods=['GET'])
@jwt_required()
def get_routes():
    username = request.args.get('username')  # En una implementación real, el nombre de usuario vendría del token

    # Consultar rutas del usuario
    routes = list(routes_collection.find({'username': username}, {'_id': 0}))

    if routes:
        return jsonify({'routes': routes}), 200
    else:
        return jsonify({'message': 'No se encontraron rutas'}), 404

# Iniciar la aplicación
if __name__ == '__main__':
    app.run(debug=True)
```

**Enlace github:**

**https://github.com/DubAngk/TripRoute2.git**

**Explicación del Código:**

1. **Flask**: Se utiliza Flask como el framework principal para crear el backend de la aplicación.
    - ✓ **@app.route()**: Define las rutas de la API. Las solicitudes se manejan a través de las rutas HTTP (POST, GET).
    - ✓ **request.json**: Para obtener los datos JSON enviados por el cliente (por ejemplo, datos de usuario, ruta, etc.).
2. **JWT (JSON Web Token)**: Utilizamos el paquete flask_jwt_extended para la autenticación y autorización de usuarios.
3. **create_access_token()**: Crea un token JWT que se devuelve al usuario una vez que inicia sesión correctamente.
4. **@jwt_required()**: Protege las rutas, haciendo que el usuario deba estar autenticado para acceder a ellas.
5. **MongoDB**: Se utiliza **MongoDB** para almacenar los datos de usuarios y rutas.

- **pymongo.MongoClient()**: Conecta con una base de datos MongoDB.
- **insert_one()**: Inserta documentos (datos) en la colección de la base de datos.
- **find()**: Recupera los datos de la base de datos.

6. **bcrypt**: Esta biblioteca se utiliza para encriptar las contraseñas de los usuarios antes de almacenarlas en la base de datos.

**Características:**

**Registro de usuario:** Los usuarios pueden registrarse proporcionando un nombre de usuario y contraseña.

**Iniciar sesión:** los usuarios pueden iniciar sesión y recibir un token JWT.

**Creación de rutas:** los usuarios pueden crear nuevas rutas de viaje enviando el nombre de la ruta y una lista de destinos.

**Ver rutas guardadas:** los usuarios pueden obtener una lista de sus rutas guardadas.

**Aplicación del ejercicio:**

Este ejercicio implica implementar una aplicación backend simple para organizar itinerarios d e viaje utilizando Flask (un micromarco de Python) y otras tecnologías como MongoDB para almacenamiento de datos, JWT para autenticación y bcrypt para contraseña de seguridad. La aplicación está diseñada para funcionar como una API que permite a los usuarios registrarse, iniciar sesión, crear itinerarios de viaje y ver itinerarios guardados.

La aplicación simula un sistema que podría servir como base para una aplicación de viajes m ás avanzada, en la que los usuarios puedan planificar y optimizar rutas, guardar sus destinos f avoritos y ver rutas anteriores.

**Objetivo del Ejercicio:**

El objetivo de este ejercicio es ilustrar los principios básicos del desarrollo de aplicaciones w eb basadas en la arquitectura cliente-servidor, con énfasis en los siguientes aspectos:

**Diseñe una API Backend:** Implemente un sistema de enrutamiento RESTful que le permita i nteractuar con el servidor para registrar usuarios, autenticar sesiones y administrar datos sobr e rutas de viaje.

**Autenticación segura:** Implemente un sistema de autenticación utilizando tokens JWT que p ermita que solo los usuarios autenticados creen y vean rutas. Este enfoque simula una solucio n moderna y segura para proteger los recursos de una aplicación.

**Gestión de datos con MongoDB:** Utilice una base de datos NoSQL como MongoDB para al macenar y gestionar de manera eficiente la información del usuario y sus rutas, demostrando l

a flexibilidad que ofrece al tratar con datos no estructurados.

**Optimización y seguridad:** Utilice tecnologías como bcrypt para cifrar contraseñas y proteg er la información confidencial del usuario, lo cual es esencial en cualquier aplicación que trat e con datos personales. Implementación práctica: proporciona una base sobre la que se puede desarrollar una aplicación completa que ofrezca más funcionalidades, como la integración co n servicios cartográficos (Google Maps, OpenStreetMap) o herramientas de optimización de carreteras.

**Conclusión:**

TripRoute es una solución completa para viajeros que buscan una forma eficiente y flexible de organizar sus itinerarios. Al combinar la personalización de rutas con la integración de servicios externos como mapas interactivos, transporte público y recomendaciones de destinos, la aplicación permitirá a los usuarios disfrutar de una planificación más eficiente y conveniente. El desarrollo de esta aplicación, basado en una metodología ágil, permitirá asegurar que la funcionalidad se adapta a las necesidades reales de los usuarios. Además, con su capacidad para operar sin conexión y su enfoque en la simplicidad y accesibilidad, TripRoute promete ser una herramienta valiosa para cualquier viajero, mejorando el proceso de planificación y optimizando el tiempo durante el viaje. La viabilidad técnica y los beneficios potenciales hacen que este proyecto no sólo sea factible, sino también altamente rentable en el mercado actual de aplicaciones de viajes.

**Bibliografía:**

Arciniegas Herrera, J. L., Collazos Ordóñez, C. A., Fernández de Valdenebro, M. V., Hormiga Juspian, M. A., Tulande Arroyo, A. (2010). Patrones arquitectónicos sobre usabilidad en el dominio de las aplicaciones web. Ingeniería e Investigación, 30 (1), 52-55. https://revistas.unal.edu.co/index.php/ingeinv/article/view/15207/16001

Pantaleo, G. y Rinaudo, l. (2015). Ingeniería de Software. Buenos Aires: Alfaomega. https://www-alfaomegacloud-com.ibero.basesdedatosezproxy.com/auth/ip?intended_url=https://www-alfaomegacloud-com.ibero.basesdedatosezproxy.com/library/publication/ingenieria-de-software

Google Cloud. (n.d.). Google Maps API documentation. https://developers.google.com/maps/documentation

Shaw, Z. A. (2021). Learn Python 3 the hard way: A very simple introduction to the terrifyingly beautiful world of computers and code. Addison-Wesley.

Wells, L. (2021). The definitive guide to Flutter: Planning, designing, and building cross-platform apps. Apress.