PROYECTO No. 0: NIVELACIÓN

OBJETIVOS

- Aplicar los conocimientos necesarios para desarrollar una aplicación web, con el propósito de reforzar las habilidades y competencias en el desarrollo de software que son prerrequisitos del curso.
- Implementar una aplicación web utilizando un framework de desarrollo ágil, como Flask o FastAPI para Python, junto con estándares de desarrollo web para la creación de la capa de presentación.

RECOMENDACIONES, CONSIDERACIONES Y LECTURAS PREVIAS

En este taller, se deberá desarrollar una aplicación web. El backend será construido como una API REST, utilizando Python como lenguaje de programación y un framework de desarrollo web como Flask o FastAPI. La API REST debe ser consumida por el frontend de la aplicación. En cuanto al frontend, tienes libertad para utilizar las tecnologías de tu preferencia.

A continuación, se relacionan algunos enlaces y material de referencia para aprender cada framework.

- **Flask.** Para aprender este framework se recomienda visitar, revisar y utilizar los siguientes materiales.
  - Sitio oficial: https://flask.palletsprojects.com/
- **FastAPI.** Para aprender este framework se recomienda visitar, revisar y utilizar los siguientes materiales.
  - Sitio oficial: https://fastapi.tiangolo.com/es/tutorial/first-steps/

Adicionalmente, pueden incorporar frameworks MVC para el desarrollo de la capa de presentación (frontend), como es el caso de Angular o Vue, entre muchos otros. El diseño de las interfaces de usuario es de libre elección. Si no tienen experiencia con temas de frontend y no desea salir de Python como lenguaje de programación, pueden utilizar frameworks como Flet o Reflex.

- **Flet.** Para aprender este framework se recomienda visitar, revisar y utilizar los siguientes materiales.
  - Sitio oficial: https://flet.dev/docs/
- **Reflex.** Para aprender este framework se recomienda visitar, revisar y utilizar los siguientes materiales.
  - Sitio oficial: https://reflex.dev/

La aplicación deberá ser empaquetada en contenedores Docker y ejecutada en una máquina virtual asignada a cada estudiante. En dicha máquina, se desplegarán todos los componentes de la aplicación.

AMBIENTE DE DESPLIEGUE

Cada estudiante del curso tendrá acceso al laboratorio de aprendizaje de AWS Academy para aprovisionar una máquina virtual con el sistema operativo Ubuntu GNU/Linux y en ella ejecutar su aplicación web. En Bloqueneon, encontrará la información sobre cómo utilizar AWS Academy.

DESCRIPCIÓN DE LA APLICACIÓN WEB A DESARROLLAR

El objetivo de este proyecto es desarrollar una aplicación web para la gestión de listas de tareas, diseñada para que los usuarios puedan crear, organizar y monitorear el progreso de sus actividades diarias. La solución estará compuesta por tres componentes principales:

- **API REST**: Implementará la lógica del negocio, proporcionando los endpoints necesarios para la gestión de usuarios, tareas y categorías.
- **Base de datos**: Almacenará de manera estructurada la información relacionada con los usuarios y sus tareas.
- **Interfaz web**: Ofrecerá a los usuarios una experiencia interactiva e intuitiva para administrar sus tareas y categorías, integrándose con los otros componentes.

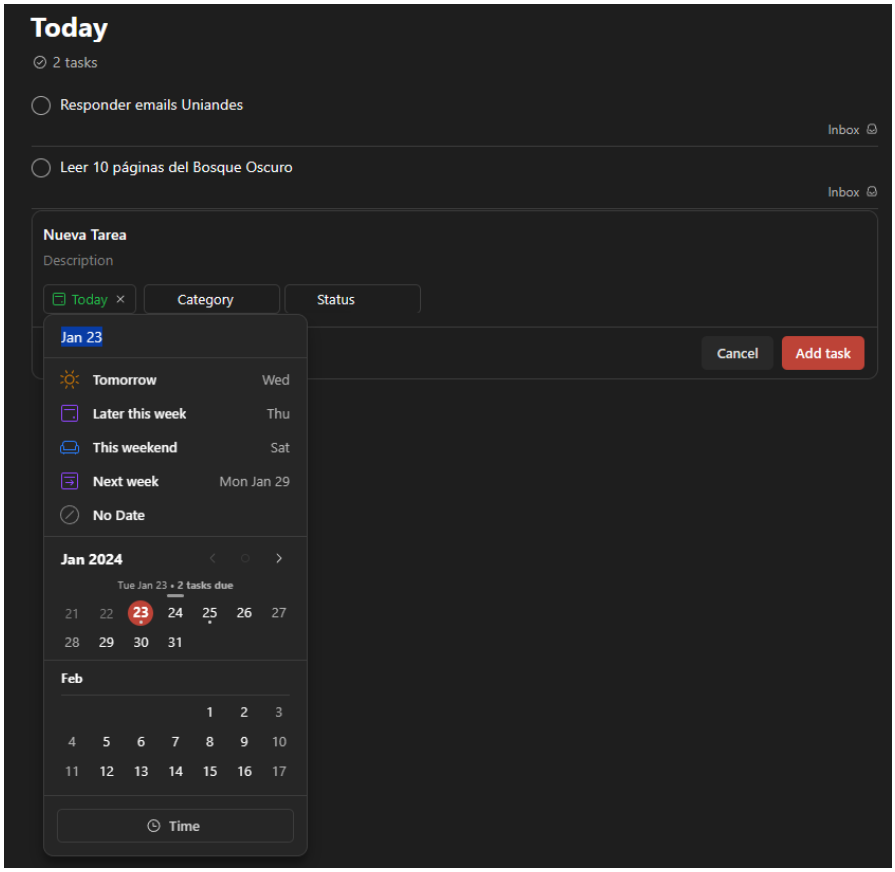
Funcionalidades Principales:

1. Autenticación de Usuarios:
    - Creación de cuenta con usuario y contraseña.
    - Posibilidad de cargar una imagen de perfil.
    - Si el usuario no carga una foto, el sistema colocará un icono por defecto.

- Inicio de sesión y cierre de sesión.

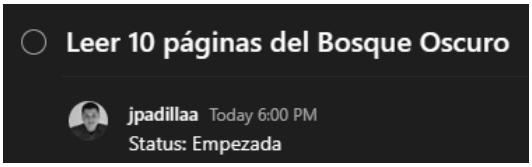
2. Gestión de Listas y Tareas:

- Captura de texto que representa una tarea.
- Organización de tareas en categorías (por ejemplo, Hogar, Trabajo, Urgente, entre otras).
- (Opcional) Creación y eliminación de categorías. Sin embargo, **si esto no se implementa, la aplicación deberá tener una conjunto de categorías predefinidas**.



3. Estados y Fechas:

- Asignación de cada tarea a al menos un estado: Sin Empezar, Empezada, Finalizada.
- Actualización de estados por parte del usuario.



- Registro de la fecha de creación de la tarea para futura trazabilidad y analítica.
- (Opcional) Registro de la fecha de creación de la tarea para futura trazabilidad y analítica.

4. Operaciones sobre Tareas:
   - Creación de tareas por parte del usuario.
   - Eliminación de tareas por parte del usuario.

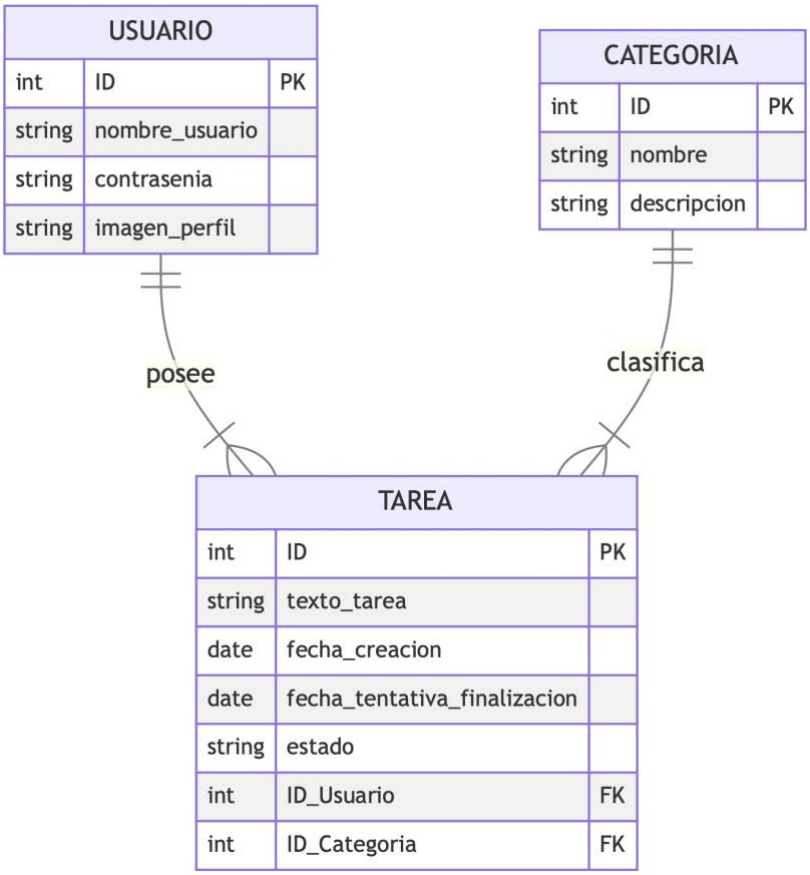
Componentes de la Aplicación:

1. API REST (Flask o FastAPI):
   - Encargada de la lógica del negocio.
   - Gestión de usuarios, autenticación y autorización.
   - Manipulación de listas, tareas, categorías y estados.
2. Base de Datos:
   - Almacenamiento de información de usuarios (cuentas y preferencias). SQL es suficiente para este proyecto. l
   - Registro de tareas y estados.
3. Interfaz Web:
   - Permite a los usuarios interactuar de forma amigable con la aplicación.
   - Visualización, creación, actualización y eliminación de tareas.

Modelo Entidad - Relación:

A continuación, se sugiere un modelo entidad-relación (ER) que representa las entidades mínimas necesarias para la aplicación de gestión de tareas. Siéntase libre de modificarlo o proponer uno diferente según su conveniencia y experiencia.

Este modelo incluye las entidades principales (Usuario, Tarea y Categoría) y establece relaciones entre ellas. Cada tarea está asociada a un usuario y a una categoría (Puede reemplazarse por una enumeración).



API REST:

A continuación, se define la lista de endpoints necesarios para el backend REST de la aplicación de gestión de tareas. Estos endpoints cubren las funcionalidades básicas requeridas. Puede ajustar o ampliar esta definición según sea necesario, de acuerdo con sus necesidades y experiencia.

Usuarios:

- **Crear Usuario (POST /usuarios)**
- **Iniciar Sesión (POST /usuarios/iniciar-sesion)**
- (Opcional) Se podría considerar un endpoint para refrescar el token (por ejemplo, **POST /usuarios/refresh-token**) y otro para cerrar sesión (**POST /usuarios/logout**), según tu estrategia de autenticación.

Tareas:

- **Obtener Lista de Tareas por Usuario (/usuarios/{id}/tareas)**
- **Crear Tarea (POST /tareas):** Se recomienda agregar validaciones para la fecha tentativa de finalización (por ejemplo, que no sea anterior a la fecha actual).
- **Actualizar Tarea (PUT /tareas/{id}):** Permite cambiar texto y estado.
- **Eliminar Tarea (DELETE /tareas/{id})**
- **Obtener Tarea por ID (GET /tareas/{id})**

Categorías (depende directamente de sus decisiones de diseño):

- **Crear Categoría (POST /categorias)**
- **Eliminar Categoría (DELETE /categorias/{id})**
- **Obtener Lista de Categorías (GET /categorias)**
- (Opcional) Se podría considerar un endpoint para actualizar una categoría (**PUT /categorias/{id}**), en caso de que desee modificar nombre o descripción.

Autorización y seguridad:

- Asegúrate de exigir un token de autenticación (el obtenido en /usuarios/iniciar-sesion) para los endpoints sensibles (crear/actualizar/eliminar tareas, crear/eliminar categorías, etc.).

Nota: Se recomienda documentar los endpoints del API REST con una herramienta que utilice el estándar de OpenAPI. En el caso de FastAPI, este framework ofrece funcionalidades integradas que permiten generar y actualizar la documentación de manera automatizada.

CONSIDERACIONES

El servidor web debe poder manejar solicitudes de forma concurrente. Se recomienda utilizar el siguiente stack tecnológico para el desarrollo de la entrega:

1. Python 3.12 o superior.
2. Flask Framework 3.1.x o FastAPI 0.115.x
3. SQLAlchemy.
4. JWT: JSON Web Tokens
5. SQLite o PostgreSQL.
6. Gunicorn o Uvicorn: servidor HTTP WSGI para Python y ambientes Unix.
7. Postman o Swagger: Para documentar el API REST.
8. Docker: Para empaquetar la aplicación.
9. Para el fronend utilice la tecnología de su preferencia.

Recomendaciones adicionales:

- La página inicial (home) de la aplicación debe mostrar la página de inicio de sesión si el usuario no ha iniciado sesión. Si el usuario ya ha iniciado sesión, debe redirigir al listado de tareas.
- Siéntase en libertad de tomar las decisiones de análisis y diseño que considere más convenientes.

EJECUCIÓN DE LA APLICACIÓN

Para la ejecución de la aplicación es necesario que cada estudiante configure el puerto y la IP por el que escucha el servidor, de la siguiente forma:

IP: 0.0.0.0 **Puerto:** 8080

- Al especificar la dirección IP 0.0.0.0 al exponer un servicio, estás indicando que el servicio debe escuchar en todas las interfaces de red disponibles en el sistema. En términos de configuración de servidores y redes, esto se interpreta como "escuchar en todas las direcciones IP disponibles".
- El 8080 es un puerto de ejemplo, puede utilizar el de su preferencia

Para comprobar que la aplicación puede ser accedida, abra el navegador, ingrese en el navegador la dirección ip de su máquina virtual (puede consultar la IP de su máquina mediante el comando *ipconfig o ip a si está haciendo pruebas en local*) dos puntos el puerto 8080, por ejemplo:

- http://192.168.0.1:8080

EQUIPO DE TRABAJO

Es un ejercicio de nivelación, por lo que este proyecto debe ser realizado de forma individual.

ESQUEMA DE EVALUACIÓN

La distribución de la calificación del taller está distribuida de la siguiente manera:

- Verificación funcional de los requerimientos de la aplicación en la sustentación: **100%**
- Proyectos que no compilen o que no se puedan ejecutar durante la verificación tendrán como nota cero (0.0).
- Para el día de la entrega, la aplicación Web debe estar funcionando en una máquina virtual usando el Learner Lab de AWS Academy.

RUBRICA DE EVALUACIÓN

Criterios	Puntos
Backend (65 puntos)	
- Implementación de la API REST • Creación, Actualización y Eliminación de Tareas • Asignación de Estados y Fechas • Gestión de Categorías	50
- Uso de Base de Datos	5
- Implementación de la Autenticación	10
Frontend (20 puntos)	
- Interfaz de Usuario	10
- Integración con Backend (Consumo de API)	5
- Autenticación de Usuarios (Frontend - Backend)	5
Documentación (7 puntos)	
- Documentación de la API con OpenAPI	5
- Instrucciones de Despliegue y Uso	2
Despliegue (8 puntos)	
- Uso de Contenedores Docker	8
Total (100 puntos)	