

UNIVERSIDAD DEL VALLE DE GUATEMALA

Diseño e Innovación

Sección 21

Eddy Castro



Prototipo

Protocolo Borrador 2

José Daniel Gómez Cabrera 21429

Introducción

El presente documento describe el prototipo del servicio y la infraestructura para un sistema inteligente de orientación vocacional. Este sistema tiene como objetivo ayudar a estudiantes graduandos a elegir una licenciatura, generar temarios de estudio personalizados y ofrecer sugerencias de especialización profesional. A continuación, se detallan las tecnologías tentativas, la arquitectura del sistema tentativa y las funcionalidades previstas para cada módulo consideradas hasta el momento.

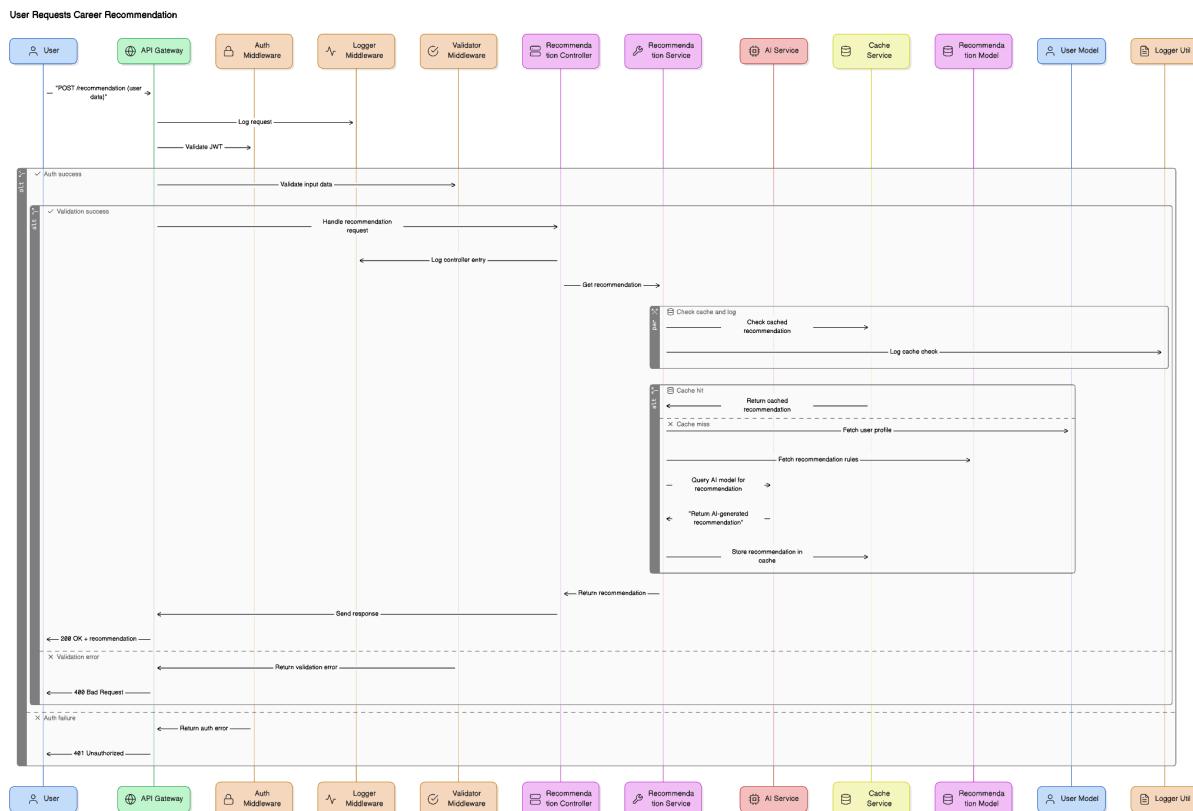
Repository

https://github.com/JDgomez2002/career_path_server

En este repositorio se puede observar las bases del prototipo para poder empezar con el desarrollo del sistema. En el repositorio también se encuentra una imagen que hace referencia a la infraestructura del servicio web.

Diagrama del prototipo

[Este diagrama](#) representa la base del prototipo del servicio web, como se puede observar en el [repositorio](#).



Protocolo

En el [repositorio](#) se puede observar en **assets** que se encuentra el documento del [protocolo del trabajo](#).

Desarrollo de un servicio y su infraestructura para abastecer un sistema inteligente de orientación vocacional, generación de temarios de estudio a medida y generación de sugerencias de especialización profesional, enfocado a estudiantes graduandos con aspiración a estudiar una licenciatura

José Daniel Gómez Cabrera



Tecnologías Tentativas

Para el desarrollo de este servicio, se consideran las siguientes tecnologías:

- **Backend:** Bun con Hono (Un framework de JavaScript, similar a Express pero más moderno) (como se observa en `src/index.ts`) para la creación de la API REST.
- **Base de Datos:**
 - **MongoDB:** Una base de datos NoSQL orientada a documentos. Sería adecuada por su flexibilidad para almacenar perfiles de usuario con estructuras variadas, resultados de evaluaciones y contenido dinámico de recomendaciones y planes de estudio. Su escalabilidad horizontal también es una ventaja para un sistema que podría crecer en usuarios y datos.
 - **PostgreSQL:** Una base de datos relacional SQL robusta y con gran capacidad para manejar relaciones complejas. Sería útil si se prevé una estructura de datos muy interconectada entre usuarios, carreras, universidades y evaluaciones, donde la integridad referencial y las transacciones ACID son cruciales.
 - **Elección:** La elección final dependerá de un análisis más profundo de los patrones de acceso a datos y la complejidad de las relaciones. Una aproximación híbrida o el uso de características específicas de cada una (ej. JSONB en PostgreSQL) también podría considerarse.
- **Autenticación:** JSON Web Tokens (JWT) para la gestión de sesiones y protección de rutas, como se infiere del `Auth Middleware` en el diagrama y el `auth.middleware.js` en la estructura de archivos.
- **Cache:** Un servicio de caché (como Redis) para almacenar temporalmente resultados de recomendaciones frecuentes o datos de acceso común, mejorando el rendimiento y reduciendo la carga en la base de datos y el servicio de IA, como se observa en el diagrama con el `Cache Service`.
- **Servicio de IA:** Integración con modelos de IA externos o desarrollados internamente para la lógica de recomendación y generación de planes de estudio. El diagrama lo identifica como `AI Service`.
- **Contenerización:** Docker y Docker Compose (`Dockerfile`, `docker-compose.yml`) para la gestión del entorno de desarrollo y despliegue.

Arquitectura del Sistema

El sistema sigue una arquitectura de microservicios o una aplicación monolítica modular bien estructurada, como se desprende del diagrama de secuencia y la organización de carpetas proporcionada en el `README.md`.

El flujo general de una solicitud de recomendación, basado en el diagrama, es el siguiente:

1. **User (Usuario):** Inicia una solicitud POST /recommendation con sus datos.
2. **API Gateway:** Actúa como punto de entrada único. Podría gestionar el enrutamiento inicial, la limitación de tasa y la terminación SSL.
3. **Auth Middleware (Middleware de Autenticación):** Valida el token JWT del usuario. Si la autenticación falla, retorna un error 401.
4. **Logger Middleware (Middleware de Registro):** Registra la solicitud entrante.
5. **Validator Middleware (Middleware de Validación):** Valida los datos de entrada de la solicitud. Si la validación falla, retorna un error 400.
6. **Recommendation Controller (Controlador de Recomendaciones):** Recibe la solicitud validada. Llama al servicio correspondiente para manejar la lógica de negocio.
7. **Recommendation Service (Servicio de Recomendaciones):**
 - Orquesta la obtención de la recomendación.
 - Interactúa con el Cache Service para verificar si existe una recomendación en caché.
 - **Cache Hit:** Si la recomendación está en caché, se retorna directamente.
 - **Cache Miss:** Si no está en caché:
 - Interactúa con el User Model (o un servicio de usuario) para obtener el perfil del usuario.
 - Interactúa con el Recommendation Model para obtener reglas o datos base para la recomendación.
 - Consulta al AI Service para generar la recomendación.
 - Almacena la nueva recomendación en el Cache Service.
 - Retorna la recomendación al Recommendation Controller.
8. **Recommendation Controller:** Envía la respuesta (la recomendación o un error) al API Gateway.
9. **API Gateway:** Retransmite la respuesta al User.
10. **Logger Util (Utilidad de Registro):** Componentes como el Cache Service pueden utilizar una utilidad de registro para registrar eventos específicos (ej. "Log cache check").

Esta arquitectura se refleja en la estructura de directorios:

- config/: Configuraciones de la aplicación, base de datos, autenticación, modelos de IA.

- `middlewares/`: Lógica transversal como autenticación, registro, validación y manejo de errores.
- `models/`: Definiciones de esquemas de datos o interacciones directas con la base de datos para cada entidad (User, Assessment, Career, Recommendation, StudyPlan).
- `controllers/`: Manejan las solicitudes HTTP, validan entradas (aunque esto podría delegarse a middlewares específicos) y orquestan las respuestas llamando a los servicios.
- `services/`: Contienen la lógica de negocio principal, interactuando con los modelos, otros servicios (como el de IA o caché) y preparando los datos para los controladores.
- `routes/`: Definen los endpoints de la API y los asocian con sus respectivos controladores.
- `utils/`: Herramientas auxiliares como loggers, validadores, funciones de seguridad.

Detalle de Features (Rutas Principales)

A continuación, se describen las funcionalidades que podrían desarrollarse para cada una de las rutas principales identificadas en `src/index.ts` y el `README.md`.

1. Autenticación (`/auth`)

Endpoints relacionados con la autenticación de usuarios y gestión de sesiones.

- **Registro de Nuevos Usuarios (POST `/auth/register`):**
 - Recopilación de datos básicos (nombre, correo electrónico, contraseña).
 - Validación de datos (formato de email, fortaleza de contraseña).
 - Hashing de la contraseña antes de almacenarla.
 - Creación del nuevo usuario en la base de datos.
 - Opcional: Envío de correo de verificación.
- **Inicio de Sesión (POST `/auth/login`):**
 - Validación de credenciales (email y contraseña).
 - Generación de un token JWT si las credenciales son correctas.
 - Retorno del token JWT al cliente.
- **Cierre de Sesión (POST `/auth/logout`):**
 - Invalidación del token JWT (puede ser mediante una lista negra en el servidor o simplemente instruyendo al cliente a borrar el token).
- **Renovación de Token (POST `/auth/refresh-token`):**
 - Si se utilizan tokens de corta duración con tokens de refresco, este endpoint permitiría obtener un nuevo token de acceso.

- **Recuperación de Contraseña (POST /auth/forgot-password y POST /auth/reset-password):**
 - forgot-password: Envía un enlace o código de recuperación al correo del usuario.
 - reset-password: Permite al usuario establecer una nueva contraseña utilizando el enlace/código.

2. Gestión de Usuarios (/user)

Gestión de perfiles de usuario y configuraciones personales. Requiere autenticación.

- **Obtener Perfil del Usuario (GET /user/profile o GET /user/me):**
 - Retorna la información del usuario autenticado (nombre, email, preferencias, etc.).
- **Actualizar Perfil del Usuario (PUT /user/profile o PATCH /user/profile):**
 - Permite al usuario modificar su información personal.
 - Validación de los datos actualizados.
- **Cambiar Contraseña (PUT /user/change-password):**
 - Permite al usuario cambiar su contraseña actual por una nueva, requiriendo la contraseña antigua.
- **Ver Historial (ej. de evaluaciones, recomendaciones vistas) (GET /user/history/assessments, GET /user/history/recommendations):**
 - Muestra un registro de las actividades relevantes del usuario dentro de la plataforma.
- **Configuraciones de Cuenta (GET /user/settings, PUT /user/settings):**
 - Preferencias de notificaciones, privacidad, etc.
- **Eliminar Cuenta (DELETE /user/account):**
 - Permite al usuario eliminar su cuenta y datos asociados (considerar políticas de retención de datos).

3. Evaluaciones (/assessment)

Realización de pruebas vocacionales y gestión de sus resultados. Requiere autenticación.

- **Listar Evaluaciones Disponibles (GET /assessment):**
 - Muestra las diferentes pruebas vocacionales que el usuario puede tomar.

- **Iniciar/Tomar una Evaluación (POST**
/assessment/{assessmentId}/take o GET
/assessment/{assessmentId}/questions):
 - Presenta las preguntas de una evaluación específica.
 - Podría implementarse pregunta por pregunta o todo el cuestionario de una vez.
- **Guardar Respuestas de una Evaluación (POST**
/assessment/{assessmentId}/answers):
 - Almacena las respuestas del usuario para una evaluación en curso o completada.
 - Validación de las respuestas.
- **Obtener Resultados de una Evaluación (GET**
/assessment/{sessionId}/results o GET
/user/assessments/{assessmentId}/results):
 - Calcula y muestra los resultados de una evaluación completada por el usuario.
 - Interpretación de los resultados.
- **Historial de Evaluaciones del Usuario (GET /user/assessments):**
 - Lista todas las evaluaciones que el usuario ha tomado y sus estados (completada, en curso).

4. Información de Carreras (/career)

Información sobre carreras universitarias y programas académicos. Podría tener partes públicas y partes que requieran autenticación para personalización.

- **Buscar Carreras (GET /career/search?q={query}):**
 - Permite buscar carreras por nombre, área de estudio, universidad, etc.
 - Filtros avanzados (ubicación, duración, tipo de institución).
- **Obtener Detalles de una Carrera (GET /career/{careerId}):**
 - Muestra información detallada de una carrera: descripción, plan de estudios, universidades que la ofrecen, perspectivas laborales, habilidades requeridas.
- **Listar Universidades (GET /career/universities):**
 - Catálogo de universidades con información relevante.
- **Comparar Carreras (GET**
/career/compare?ids={careerId1}, {careerId2}):
 - Funcionalidad para comparar lado a lado varias carreras.

5. Recomendaciones (/recommendation)

Sistema de recomendaciones personalizadas basadas en el perfil del usuario, sus evaluaciones y preferencias. Requiere autenticación.

- **Obtener Recomendaciones de Carrera (POST /recommendation/career o GET /recommendation/career):**
 - Endpoint principal que activa el flujo descrito en el diagrama.
 - Toma como entrada (implícita o explícita) el perfil del usuario, resultados de evaluaciones, e intereses.
 - Proceso:
 1. Logger Middleware: Log de la petición.
 2. Validator Middleware: Validación de datos de entrada (si los hay explícitos).
 3. Recommendation Controller maneja la petición.
 4. Recommendation Service orquesta:
 - Cache Service: Check de caché.
 - Si hay cache miss:
 - User Model: Obtención de perfil de usuario.
 - Recommendation Model: Obtención de reglas/criterios de recomendación.
 - AI Service: Generación de la recomendación.
 - Cache Service: Almacenamiento de la recomendación.
 - Retorno de la recomendación.
 - La respuesta incluiría una lista de carreras recomendadas con justificaciones.
- **Obtener Sugerencias de Especialización Profesional (POST /recommendation/specialization o GET /recommendation/specialization):**
 - Similar al anterior, pero enfocado en especializaciones dentro de un área o carrera ya definida.
 - Podría tomar como entrada una carrera base o el perfil avanzado del usuario.
- **Feedback sobre Recomendaciones (POST /recommendation/{recommendationId}/feedback):**
 - Permite al usuario calificar o dar feedback sobre la utilidad de una recomendación, para refinar el modelo de IA.

6. Planes de Estudio (/plan)

Generación y gestión de planes de estudio personalizados. Requiere autenticación.

- **Generar Plan de Estudio (POST /plan):**
 - Toma como entrada una carrera objetivo (posiblemente de una recomendación), conocimientos previos del usuario (quizás de una evaluación de diagnóstico), y metas de aprendizaje.
 - El StudyPlan Service interactuaría con el AI Service para generar un temario estructurado y recursos sugeridos.
 - Almacenamiento del plan generado asociado al usuario.
- **Obtener Plan de Estudio Actual (GET /plan/{planId}):**
 - Muestra el plan de estudio activo o uno específico del usuario.
 - Detalle de módulos, temas, recursos, cronograma.
- **Actualizar Progreso en el Plan (PUT /plan/{planId}/progress):**
 - Permite al usuario marcar temas como completados, ajustar fechas.
- **Listar Planes de Estudio del Usuario (GET /user/plans):**
 - Muestra todos los planes de estudio creados por o para el usuario.
- **Personalizar Plan de Estudio (PUT /plan/{planId}):**
 - Permite al usuario modificar un plan existente (añadir/quitar temas, cambiar orden).

Consideraciones Adicionales

- **Seguridad:** Además de la autenticación JWT, se deben considerar otras medidas de seguridad como protección contra XSS, CSRF, SQL injection (si aplica), y la correcta configuración de CORS.
- **Escalabilidad:** La arquitectura debe diseñarse pensando en la escalabilidad, especialmente para los servicios de IA y base de datos.
- **Sockets:** Se debe de considerar la implementación correcta de web sockets para poder mantener una comunicación efectiva en tiempo real con el cliente que desee utilizar el servicio.
- **Pruebas:** Se debe implementar una estrategia de pruebas robusta, incluyendo tests unitarios, de integración y end-to-end (como se sugiere en la estructura de carpetas tests/).
- **Documentación:** Generar y mantener una documentación clara de la API del servicio (ej. usando Swagger/OpenAPI), para que se facilite consumir el servicio.