**Documentación Técnica - StudyForge (Fase 2)**

# **Introducción**

El presente documento tiene como objetivo detallar los aspectos técnicos del proyecto **StudyForge**, un asistente de estudio basado en inteligencia artificial que permite a los estudiantes **subir documentos** en distintos formatos (PDF, Word, TXT), procesarlos automáticamente y generar **resúmenes claros, glosarios conceptuales y quizzes adaptativos**.

La Documentación Técnica busca dar cuenta de la **arquitectura utilizada, los modelos de datos, los procesos internos, la infraestructura de despliegue y las decisiones tecnológicas** adoptadas, siguiendo el enfoque **Modelo 4+1 de vistas arquitectónicas**. Con ello se asegura que el sistema no solo cumpla con sus funcionalidades, sino que también sea **escalable, seguro y mantenible en el tiempo**.

# **Arquitectura General**

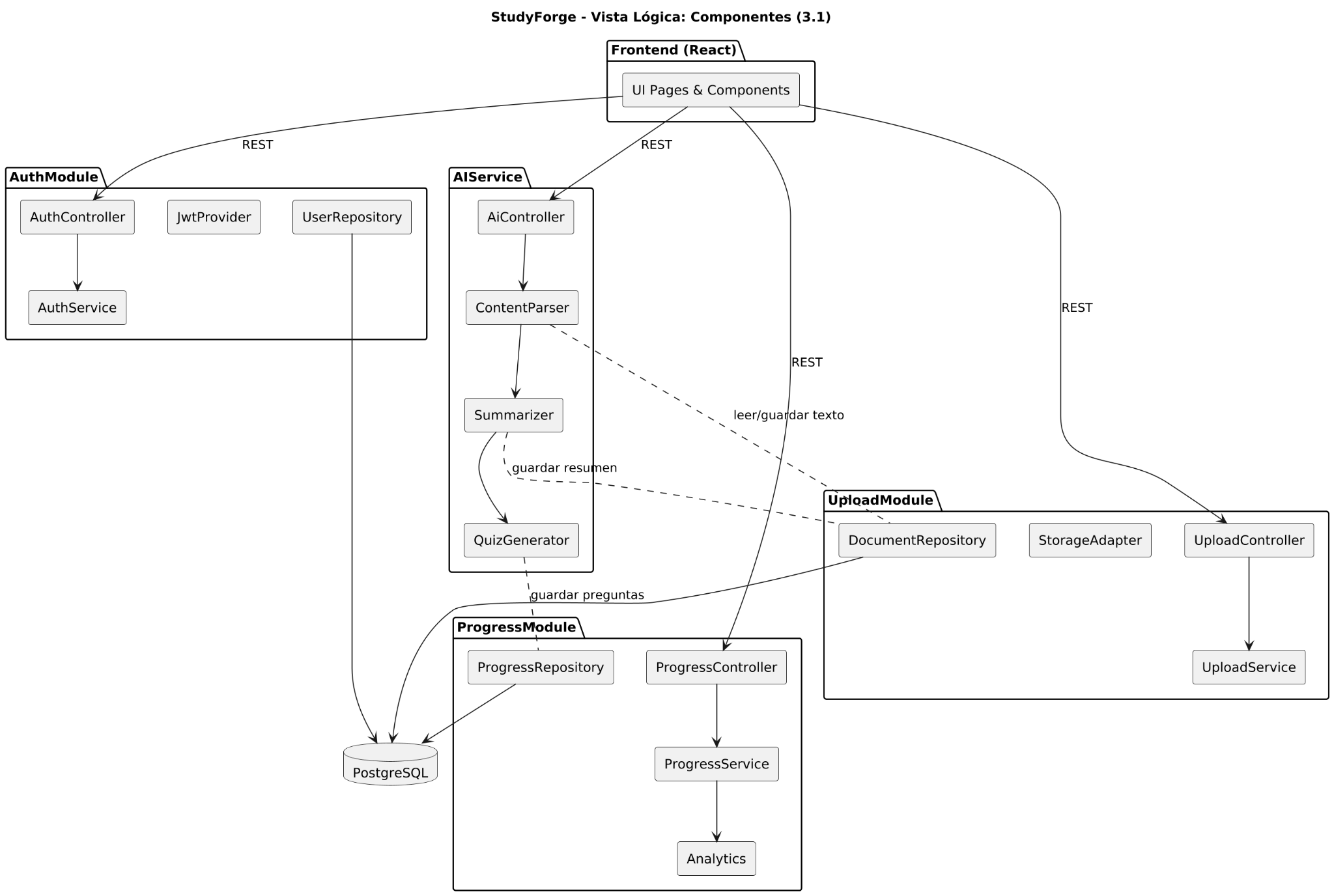
El estilo arquitectónico seleccionado para StudyForge corresponde a una **arquitectura en capas (N-tier)**. Se definieron tres niveles principales:

1. **Capa de Presentación (Frontend):** desarrollada en **React con Vite y TailwindCSS**, encargada de la interfaz de usuario, la navegación y la comunicación con el backend mediante API REST.
2. **Capa de Lógica de Negocio (Backend):** implementada en **Python con FastAPI**, contiene las reglas de negocio, la gestión de usuarios, la orquestación de la IA y los endpoints que sirven los datos al frontend.
3. **Capa de Persistencia (Base de Datos):** construida sobre **PostgreSQL**, encargada de almacenar de manera persistente la información relacionada con usuarios, documentos, resúmenes, quizzes y métricas de progreso.

Esta separación de responsabilidades ofrece **claridad en el desarrollo, independencia de módulos y facilidad de escalamiento**.

# **3. Vistas Arquitectónicas (Modelo 4+1)**

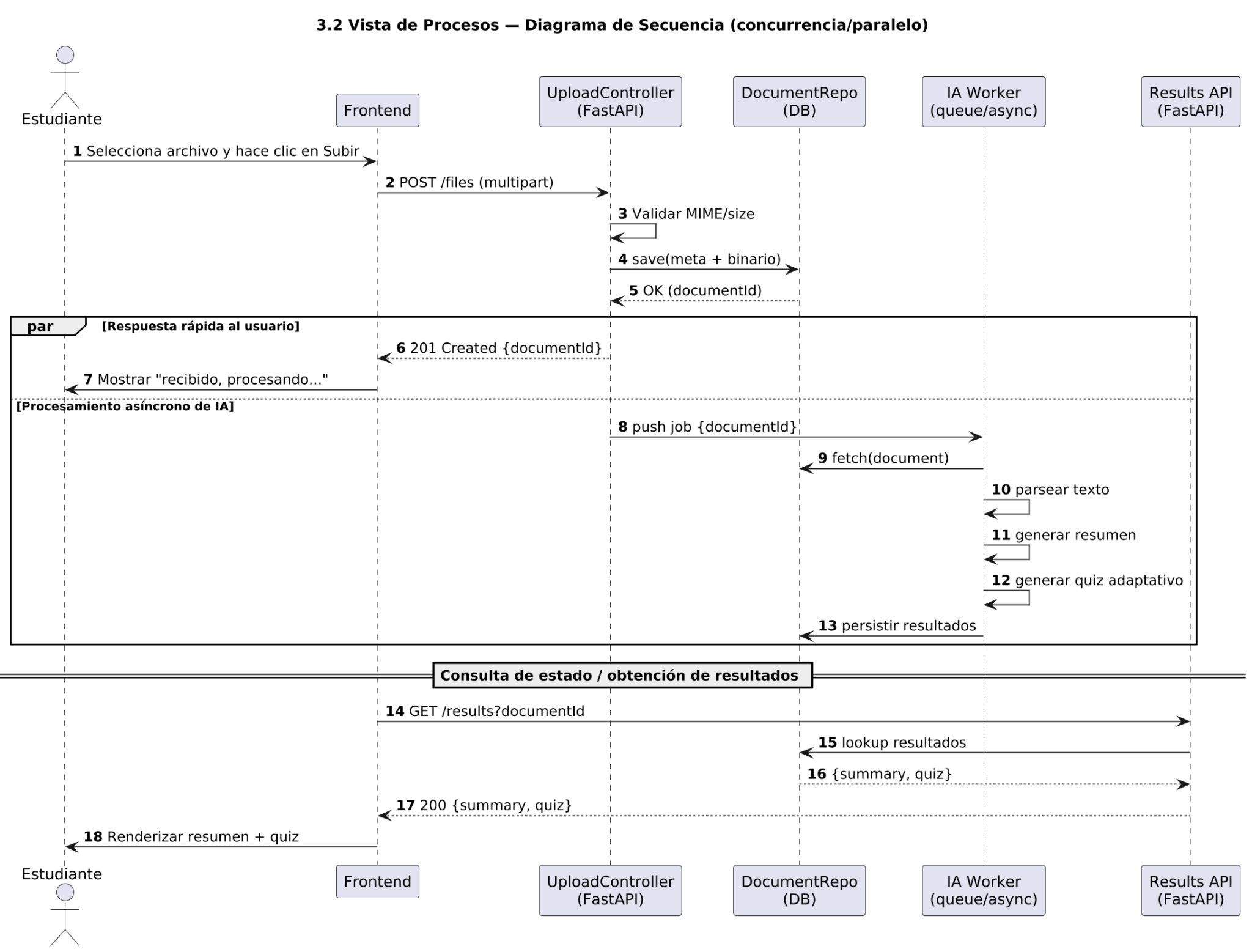
## **3.1.- Vista lógica**



En esta vista se describen los **componentes clave del sistema**:

* **Módulo de autenticación:** gestiona el inicio y cierre de sesión, así como la creación de cuentas de usuario, utilizando JWT para la autenticación segura.
* **Módulo de Carga de Archivos:** permite al estudiante subir documentos, validando el formato y tamaño, para luego almacenarlos en el servidor.
* **Módulo de Procesamiento IA:** encargado de enviar los documentos a los modelos de IA, generando resúmenes y quizzes adaptativos.
* **Módulo de Seguimiento del Progreso:** registra las interacciones del usuario, mide el avance y genera estadísticas personalizadas.

## **3.2.- Vista de procesos**



Esta vista explica cómo el sistema maneja las interacciones de los usuarios en paralelo:

* Cuando un estudiante sube un documento, el **frontend** envía la petición al **backend**.
* El **backend** valida el archivo y lo guarda en la base de datos.
* Paralelamente, se activa el **servicio de IA**, que procesa el contenido y genera los resultados.
* El estudiante recibe en la interfaz un resumen y, posteriormente, quizzes generados automáticamente.

Esto asegura **fluidez en la experiencia**, evitando bloqueos y permitiendo la concurrencia de múltiples usuarios.

## **3.3.- Vista de desarrollo**

La organización del código se ha estructurado de la siguiente forma:

**Frontend:**

/src

├── pages/ # Vistas principales (Login, Signup, Features, AboutUs)

├── components/ # Componentes reutilizables (Navbar, Footer, Cards)

├── assets/ # Imágenes, íconos, fuentes

└── styles/ # Configuración de Tailwind

**Backend:**

/backend

├── app/

│ ├── main.py # Punto de entrada

│ ├── routers/ # Endpoints de API

│ ├── models/ # Modelos de base de datos (SQLAlchemy)

│ ├── schemas/ # Validación de datos con Pydantic

│ ├── services/ # Lógica de negocio e integración IA

│ └── tests/ # Pruebas unitarias

│ └── ai/

│ └── \_\_**init**\_\_.py

## **3.4.- Vista física**

La infraestructura actual se despliega en un entorno **local**:

* Frontend en localhost:5173 mediante Vite.
* Backend en localhost:8000 utilizando Uvicorn.
* Base de Datos en localhost:5432 con PostgreSQL.

A futuro, se plantea la contenerización con **Docker** y el despliegue en un servicio en la nube (AWS, Azure o GCP), utilizando un **reverse proxy con Nginx** para balancear carga y permitir HTTPS.

## **3.5.- Vistas de caso de uso**

Los principales casos de uso definidos son:

* **Subir documentos** (PDF, Word, TXT).
* **Generar resumen** automático
* **Generar quizzes adaptativos**.
* **Visualizar progreso y estadísticas**.

Estos casos de uso permiten representar cómo interactúan los usuarios con el sistema.

# **4. Modelo de datos**

La base de datos se diseñó con un modelo **Entidad-Relación** que incluye las siguientes tablas principales:

* **Users**: gestiona la información de usuarios (id, nombre, correo, contraseña).
* **Documentos**: almacena los documentos subidos y sus metadatos.
* **Summaries**: contiene los resúmenes generados por la IA.
* **Quizzes**: preguntas adaptativas asociadas a cada documento.
* **progress**: registros del avance del estudiante.

Cada tabla incluye claves primarias y foráneas para mantener la **integridad referencial**

# **5. Tecnologías y herramientas**

* **Frontend:** React, Vite, TailwindCSS.
* **Backend:** Python, FastAPI, Pydantic.
* **Base de Datos:** PostgreSQL.
* **Gestión Ágil:** Trello / Jira.
* **Control de versiones:** Git + GitHub.
* **Documentación y diagramas:** Word, Lucidchart, [draw.io](http://draw.io).

# **6. Estilo arquitectónico**

La elección de la **arquitectura en capas (N-tier)** responde a la necesidad de separar responsabilidades. Esto permite **independencia entre frontend, backend y base de datos**, de modo que cada capa pueda evolucionar de forma aislada.

Además, se aplican principios de **MVC** en el backend, donde los **routers (controladores)**, los **schemas (modelos de datos)** y los **services (lógica de negocio)** están claramente diferenciados.

# **7. Seguridad**

El sistema implementa medidas de seguridad básicas:

* **Autenticación:** mediante tokens JWT.
* **Cifrado:** contraseñas encriptadas con bcrypt.
* **Validación:** entradas verificadas con Pydantic para evitar inyecciones.
* **Comunicación:** planificado HTTPS en producción.

# **8. Conclusión**

La arquitectura de StudyForge permite ofrecer a los estudiantes una **herramienta moderna, modular y escalable**. Gracias a su separación en capas, el sistema puede crecer con nuevas funcionalidades (como text-to-speech, integración con LMS o microservicios especializados en IA) sin comprometer la estabilidad del producto.

Con esta base técnica, el proyecto se encuentra preparado para avanzar a la siguiente fase de desarrollo y despliegue.