INTRODUCCIÓN

El presente manual técnico tiene como objetivo proporcionar una descripción detallada de la arquitectura, componentes y funcionamiento interno del sistema desarrollado. La aplicación está desarrollada en Python utilizando el microframework Flask, y cuenta con una arquitectura modular que favorece la mantenibilidad y la reutilización de código. Para el almacenamiento de datos se emplea Microsoft SQL Server, permitiendo consultas complejas, integridad referencial y un alto rendimiento. La capa frontend está construida con tecnologías estándar del desarrollo web, como HTML5, CSS3, JavaScript y el framework Bootstrap, brindando una experiencia de usuario amigable y adaptable a distintos dispositivos.

El despliegue de la aplicación se realiza en la plataforma Render, la cual automatiza el proceso de entrega continua desde el repositorio GitHub y garantiza disponibilidad en la nube. A lo largo de este documento se detallan los aspectos técnicos clave del sistema, incluyendo la estructura del proyecto, la configuración del entorno, los modelos de base de datos, la lógica de negocio y las rutas implementadas, con el fin de facilitar el mantenimiento, la escalabilidad y futuras integraciones.

OBJETIVOS DEL MANUAL

Objetivo general

Proveer una guía técnica para facilitar la implementación, capacitación y mantenimiento del sistema.

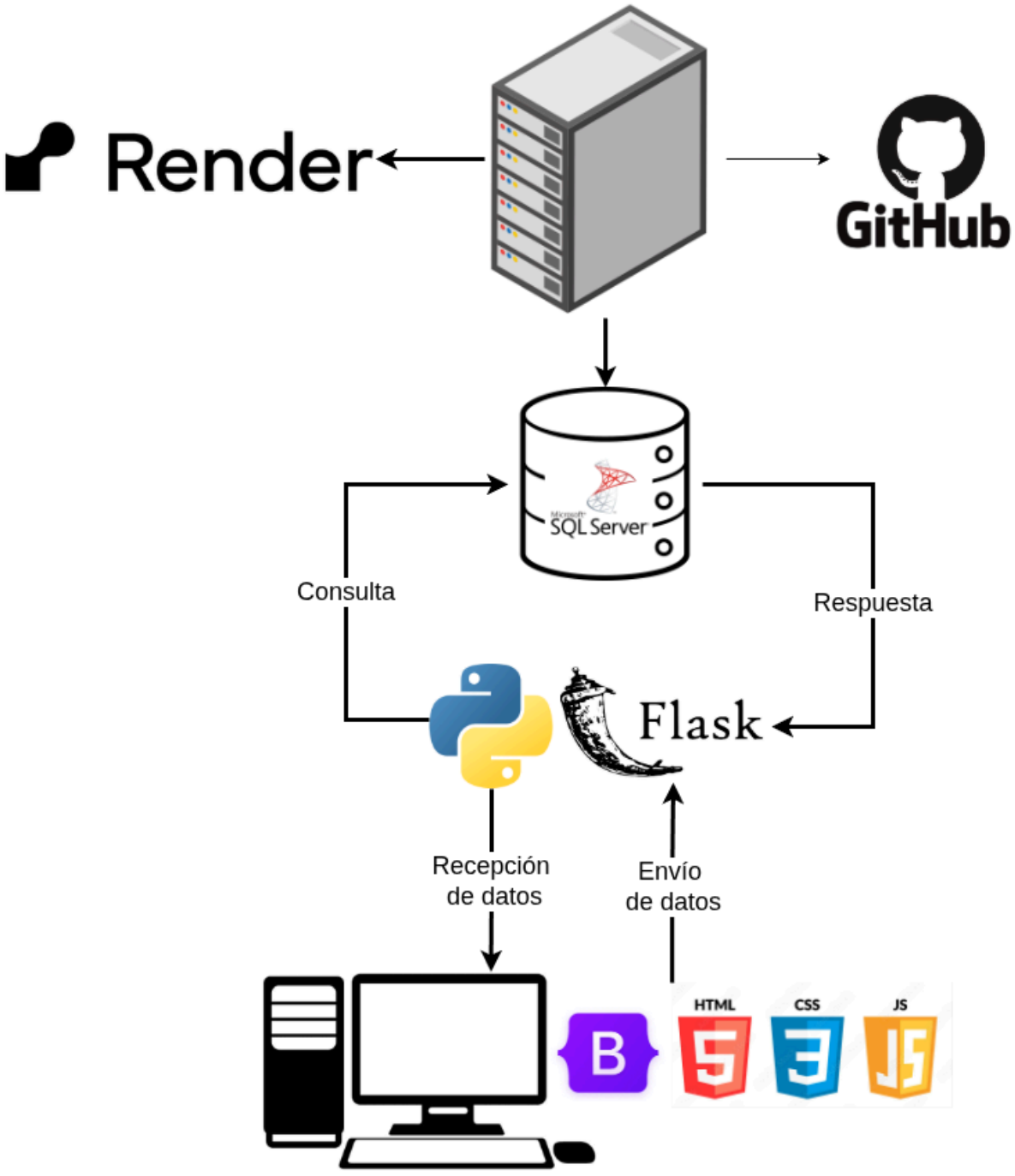
Objetivos específicos

- Ofrecer una guía técnica que detalle la arquitectura, base de datos, procesos y aspectos clave para la compresión efectiva y sin problemas.
- Brindar instrucciones claras de la instalación y configuración inicial del sistema, así como la configuración de un entorno de desarrollo.

CAPÍTULO 1: DISEÑO DEL SISTEMA

Dentro del sistema, las acciones ejecutadas dependen de la estrecha interacción entre una serie de componentes interconectados, estos componentes operan en conjunto para garantizar un flujo eficiente de la información a través del sistema. En la figura 1.1 se tiene una representación visual de la comunicación entre los distintos componentes del sistema, la cual permite comprender la forma en que trabajan en conjunto para lograr una gestión efectiva de la información.

Figura 1.1: Diagrama de componentes.



Fuente: Los autores, con la herramienta draw.io.

El diagrama de componentes ilustra la arquitectura de un sistema web desarrollado con Flask (Python) y desplegado en Render, que interactúa con una base de datos SQL Server. Este sistema permite a los usuarios interactuar con una interfaz web construida con tecnologías del lado cliente como HTML, CSS, JavaScript y Bootstrap, y se comunica con el backend a través de Flask para procesar y consultar datos almacenados en la base de datos. A continuación se describen los componentes:

1. **Render**
- IaaS en la nube donde se despliega el servidor Flask.
- Automatiza el despliegue continuo desde **GitHub**.
- Maneja conexiones entrantes desde Internet y ejecuta la lógica del servidor.

2. **GitHub**
- Repositorio remoto del proyecto.
- Render extrae el código desde GitHub para realizar despliegues automáticos.
- Facilita la colaboración y control de versiones del código.

3. **Servidor de Aplicación**
- Representado como una torre de servidores.
- Aloja la API creada con Flask.
- Gestiona las conexiones entre el cliente (navegador web) y la base de datos.

4. **SQL Server**
- Sistema de gestión de base de datos (SGBD) que almacena la información estructurada.
- El servidor Flask realiza consultas a esta base y recibe respuestas con los datos necesarios.

5. **Flask (Python)**
- Microframework que actúa como backend o API del sistema.
- Recibe peticiones HTTP del cliente, consulta la base de datos y devuelve respuestas.
- Puede enviar y recibir datos en formato JSON u otros.

6. **Frontend (HTML, CSS, JS, Bootstrap)**
- Interfaz visual que utilizan los usuarios finales.
- Desarrollada con HTML5, CSS3, JavaScript y Bootstrap para diseño responsive.
- Realiza peticiones a Flask (por ejemplo, mediante fetch o axios) y muestra los datos al usuario.

7. **Usuario final (Cliente)**

- Utiliza un navegador para acceder al sistema.
- Interactúa con formularios, botones y otros elementos de la interfaz para enviar y recibir datos.

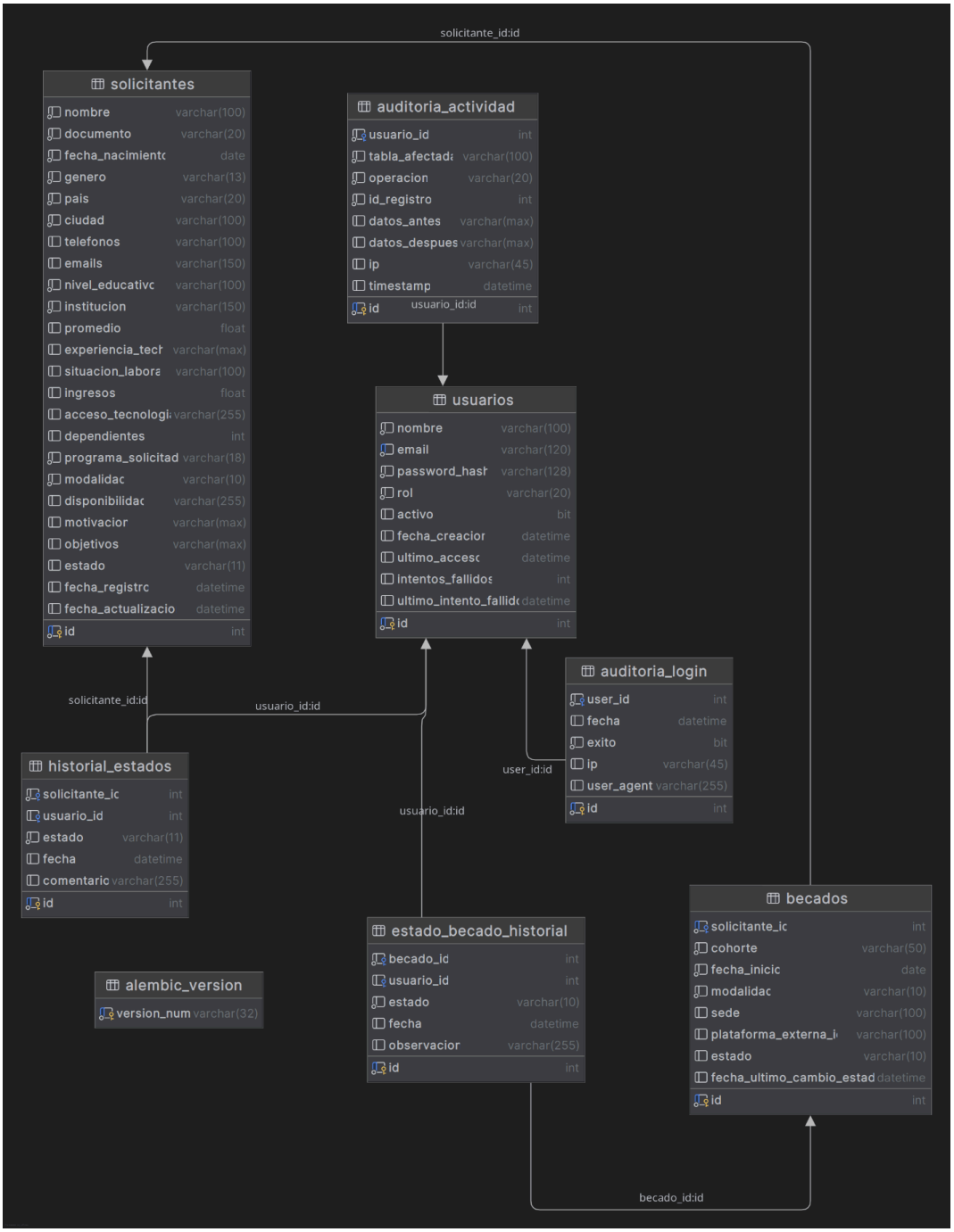
Flujo general

1. El usuario interactúa con la interfaz web.
2. El frontend envía datos al backend (Flask) a través de HTTP.
3. Flask recibe los datos y realiza una consulta al servidor SQL Server.
4. SQL Server responde con los datos solicitados.
5. Flask procesa la respuesta y la devuelve al cliente.
6. El cliente muestra los datos al usuario en la interfaz.

CAPÍTULO 2: BASE DE DATOS Y DICCIONARIO DE DATOS

La base de datos del sistema constituye un componente fundamental que permite el almacenamiento estructurado, seguro y eficiente de la información relacionada con los usuarios, solicitantes, beneficiarios (becados), registros de auditoría y demás entidades del sistema. Esta base de datos fue diseñada siguiendo principios de normalización y relaciones bien definidas para asegurar la integridad, consistencia y escalabilidad de los datos. En la figura 2.1 se muestra el diagrama de la base de datos:

Figura 2.1: Diagrama de la base de datos.



Fuente: Los autores, con la herramienta DataGrip-JetBrains.

2.1 Diccionario de datos

El diccionario de datos cumple un rol esencial como recurso de consulta, que facilita la identificación y comprensión de las entidades y atributos presentes en las tablas de la base de datos. A continuación se describe cada una de las tablas que conforman la base de datos.

2.1.1 Tabla alembic_version

Tabla 2.1: Diccionario de datos alembic_version.

Campo	Tipo	Nulo	Descripción
version_num	varchar(32)	No	Número de versión de migración

Fuente: Los autores.

2.1.2 Tabla solicitantes

Tabla 2.2: Diccionario de datos solicitantes.

Campo	Tipo	Nulo	Descripción
id	int (IDENTITY)	No	Identificador único
nombre	varchar(100)	No	Nombre completo del solicitante
documento	varchar(20)	No	Documento de identidad
fecha_nacimiento	date	No	Fecha de nacimiento
genero	varchar(13)	No	Género
pais	varchar(20)	No	País de residencia
ciudad	varchar(100)	No	Ciudad de residencia
telefonos	varchar(100)	Sí	Teléfonos de contacto
emails	varchar(150)	Sí	Correos electrónicos
nivel_educativo	varchar(100)	No	Nivel educativo alcanzado
institucion	varchar(150)	No	Institución educativa
promedio	float	Sí	Promedio académico
experiencia_tech	varchar(max)	Sí	Experiencia previa en tecnología
situacion_laboral	varchar(100)	Sí	Situación laboral actual
ingresos	float	Sí	Ingresos mensuales
acceso_tecnologia	varchar(255)	Sí	Recursos tecnológicos

			disponibles
dependientes	int	Sí	Número de dependientes económicos
programa_solicitado	varchar(18)	No	Programa educativo al que aplica
modalidad	varchar(10)	No	Modalidad del programa
disponibilidad	varchar(255)	Sí	Horarios disponibles
motivacion	varchar(max)	Sí	Motivación para aplicar
objetivos	varchar(max)	Sí	Objetivos personales/profesionales
estado	varchar(11)	Sí	Estado de la solicitud
fecha_registro	datetime	Sí	Fecha de creación del registro
fecha_actualizacion	datetime	Sí	Fecha de última actualización

Fuente: Los autores.

2.1.3 Tabla becados

Tabla 2.3: Diccionario de datos becados.

Campo	Tipo	Nulo	Descripción
id	int (IDENTITY)	No	Identificador único
solicitante_id	int	No	FK al solicitante (único)
cohorte	varchar(50)	No	Cohorte asignada
fecha_inicio	date	No	Fecha de inicio del programa
modalidad	varchar(10)	No	Modalidad del programa
sede	varchar(100)	Sí	Sede física o virtual
plataforma_externa_id	varchar(100)	Sí	ID en plataforma externa
estado	varchar(10)	Sí	Estado actual del becado
fecha_ultimo_cambio_estado	datetime	Sí	Fecha del último cambio de estado

Fuente: Los autores.

2.1.4 Tabla usuarios

Tabla 2.4: Diccionario de datos usuarios.

Campo	Tipo	Nulo	Descripción
id	int (IDENTITY)	No	Identificador único del usuario
nombre	varchar(100)	No	Nombre del usuario
email	varchar(120)	No	Correo electrónico (único)
password_hash	varchar(128)	No	Contraseña cifrada
rol	varchar(20)	No	Rol del usuario
activo	bit	Sí	Si el usuario está activo o no
fecha_creacion	datetime	Sí	Fecha de creación del usuario
ultimo_acceso	datetime	Sí	Última vez que accedió
intentos_fallidos	int	Sí	Número de intentos fallidos de acceso
ultimo_intento_fallido	datetime	Sí	Fecha del último intento fallido

Fuente: Los autores.

2.1.5 Tabla auditoria_actividad

Tabla 2.5: Diccionario de datos auditoria_actividad.

Campo	Tipo	Nulo	Descripción
id	int (IDENTITY)	No	Identificador único
usuario_id	int	No	FK a usuarios
tabla_afectada	varchar(100)	No	Nombre de la tabla afectada
operacion	varchar(20)	No	Tipo de operación (INSERT, UPDATE, DELETE, etc.)
id_registro	int	No	ID del registro afectado
datos_antes	varchar(max)	Sí	Datos antes de la operación
datos_despues	varchar(max)	Sí	Datos después de la operación
ip	varchar(45)	Sí	Dirección IP

timestamp	datetime	Sí	Fecha y hora del evento

Fuente: Los autores.

2.1.6 Tabla auditoria_login

Tabla 2.6: Diccionario de datos auditoria_login.

Campo	Tipo	Nulo	Descripción
id	int (IDENTITY)	No	Identificador único
user_id	int	No	FK a usuarios
fecha	datetime	Sí	Fecha y hora del intento
exito	bit	No	Resultado del intento
ip	varchar(45)	Sí	Dirección IP
user_agent	varchar(255)	Sí	Información del navegador/cliente

Fuente: Los autores.

2.1.7 Tabla estado_becado_historial

Tabla 2.7: Diccionario de datos estado_becado_historial.

Campo	Tipo	Nulo	Descripción
id	int (IDENTITY)	No	Identificador único
becado_id	int	No	FK a becados
usuario_id	int	Sí	FK a usuarios
estado	varchar(10)	No	Estado del becado
fecha	datetime	Sí	Fecha del cambio de estado
observacion	varchar(255)	Sí	Observaciones registradas

Fuente: Los autores.

2.1.8 Tabla historial_estados

Tabla 2.8: Diccionario de datos historial_estados.

Campo	Tipo	Nulo	Descripción
id	int (IDENTITY)	No	Identificador único
solicitante_id	int	No	FK a solicitantes

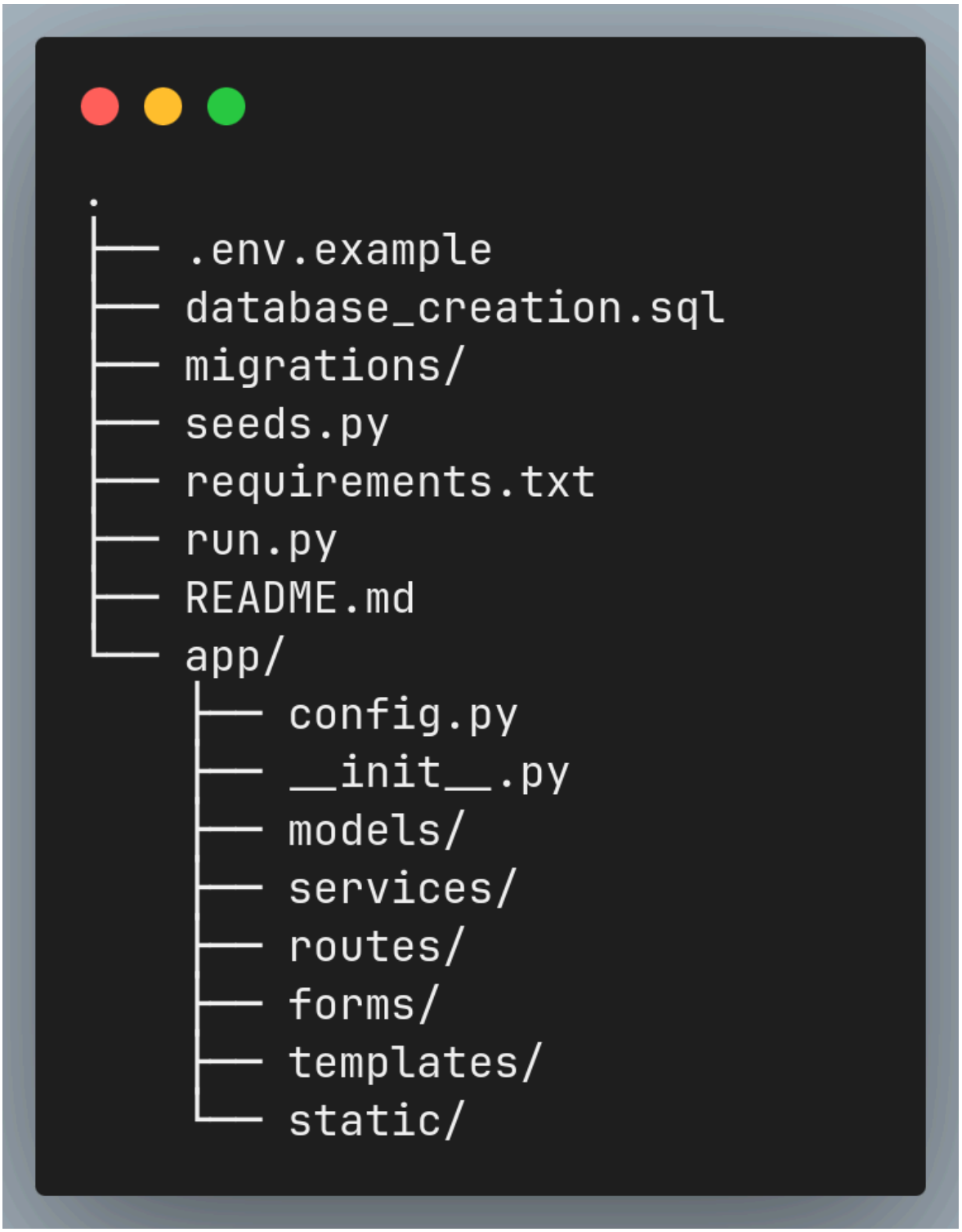
usuario_id	int	Sí	FK a usuarios
estado	varchar(11)	No	Estado registrado
fecha	datetime	Sí	Fecha de cambio
comentario	varchar(255)	Sí	Comentarios adicionales

Fuente: Los autores.

CAPÍTULO 3: ESTRUCTURA DE CARPETAS DEL PROYECTO

El proyecto está organizado de forma modular siguiendo las buenas prácticas del patrón Factory + Blueprints en Flask, lo cual permite una mejor mantenibilidad, escalabilidad y separación de responsabilidades. A continuación se describe cada uno de los archivos y carpetas principales:

Figura 3.1: Estructura del proyecto.



Fuente: Los autores.

3.1 Nivel raíz (/)

Tabla 3.1: Descripción de estructura de proyecto nivel raiz.

Archivo / Carpeta	Descripción
.env.example	Archivo de ejemplo que muestra las variables de entorno necesarias para el proyecto (como URL de la base de datos, claves secretas, etc.).
database_creation.sql	Script SQL que contiene las instrucciones para crear la base de datos, tablas, índices, restricciones y relaciones en SQL Server.
migrations/	Carpeta generada por Alembic que contiene las versiones de migraciones de esquema de base de datos. Permite rastrear y aplicar cambios estructurales.
seeds.py	Script para poblar la base de datos con datos iniciales como roles, usuario administrador, etc.
requirements.txt	Lista de dependencias del proyecto en Python. Este archivo es utilizado por pip para instalar todo lo necesario.
run.py	Punto de entrada de la aplicación Flask. Ejecuta la instancia de la app creada por create_app().
README.md	Archivo de documentación inicial del proyecto. Contiene información general, instrucciones de instalación y uso.

Fuente: Los autores.

3.2 app/ – Núcleo de la aplicación

Tabla 3.2: Descripción de estructura de proyecto núcleo de la aplicación.

Archivo / Carpeta	Descripción
__init__.py	Contiene la función create_app(), que crea y configura la aplicación Flask con extensiones, rutas, y otras inicializaciones.
config.py	Define las configuraciones del entorno (desarrollo, producción, testing), incluyendo URL de la base de datos, claves secretas, etc.
models/	Contiene todos los modelos de datos definidos con SQLAlchemy (como Usuario, Solicitante, Becado, etc.). Representan las tablas de la base de datos.

services/	Implementa la lógica de negocio del sistema. Separa la lógica de las rutas, facilitando las pruebas y el mantenimiento.
routes/	Carpeta con todos los Blueprints, es decir, los módulos de rutas organizados por funcionalidad (auth.py, solicitantes.py, users.py, etc.).
forms/	Contiene formularios creados con WTForms, usados para validar entradas del usuario como registros, logins, y formularios de solicitud.
templates/	Plantillas HTML basadas en Jinja2. Aquí se define la interfaz visual del sistema (layouts, vistas de usuario, etc.).
static/	Contiene archivos estáticos como hojas de estilo CSS, scripts JavaScript y recursos como imágenes o íconos.

Fuente: Los autores.

CAPÍTULO 4: PREPARAR ENTORNO DE DESARROLLO

4.1 Requisitos

- Python 3.10+
- Microsoft SQL Server (o SQL Server Express).
- Cliente ODBC “ODBC Driver 17 for SQL Server”.
- pip o poetry.
- Entorno virtual para Python.

4.2 Configuración de entorno

1) Copia y renombra el fichero de ejemplo:

Imagen 4.1: Comando para copiar archivo .env.

Fuente: Los autores.

2) Edita .env y define:

Imagen 4.2: Ejemplo para configurar de archivo .env.

Fuente: Los autores.

4.3 Instalación de dependencias

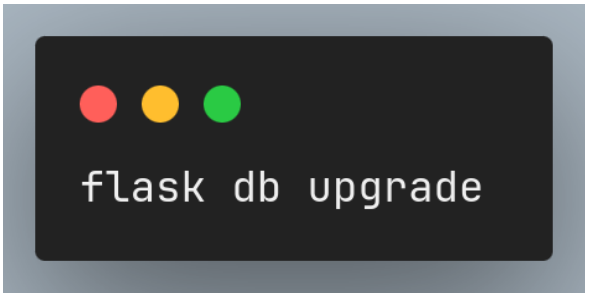
Imagen 4.3: Comando para instalar dependencias.

Fuente: Los autores.

4.4 Migraciones con Alembic

1) Copia y renombra el fichero de ejemplo:

Imagen 4.4: Comando para copiar archivo .env.



Fuente: Los autores.

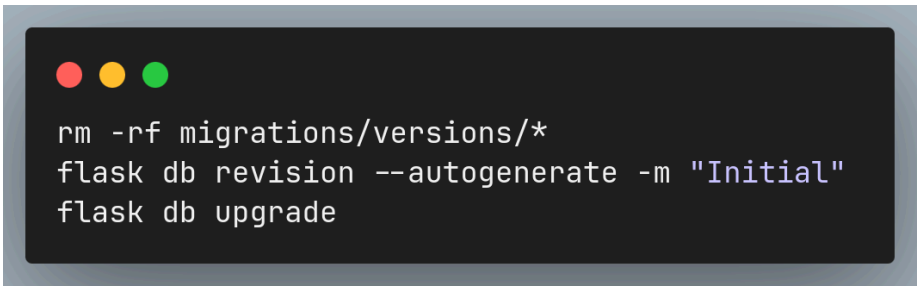
Figura 4.5: Resultado del comando anterior.

```
(.venv) synonimous [00:12:31] [~/Desktop/proyecto-irsi-conia/irsi-system-becas] [main *]
-> % flask db upgrade
INFO  [alembic.runtime.migration] Context impl MSSQLImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
```

Fuente: Los autores.

En caso de error, ejecutar los siguientes comandos que eliminaran los archivos de migración para reemplazarlos por nuevos:

Imagen 4.6: El Comando ejecuta migraciones en caso de error.



Fuente: Los autores.

Imagen 4.7: Resultado del comando anterior en caso de error.

```
(.venv) synonimous [00:13:12] [~/Desktop/proyecto-irsi-conia/irsi-system-becas] [main *]
-> % flask db revision --autogenerate -m "Initial"
INFO  [alembic.runtime.migration] Context impl MSSQLImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.autogenerate.compare] Detected added table 'solicitantes'
INFO  [alembic.autogenerate.compare] Detected added table 'usuarios'
INFO  [alembic.autogenerate.compare] Detected added table 'auditoria_actividad'
INFO  [alembic.autogenerate.compare] Detected added table 'auditoria_login'
INFO  [alembic.autogenerate.compare] Detected added table 'becados'
INFO  [alembic.autogenerate.compare] Detected added table 'historial_estados'
INFO  [alembic.autogenerate.compare] Detected added table 'estado_becado_historial'
  Generating Desktop/proyecto-irsi-conia/irsi-system-becas/migrations/versions/36b81f0d82f1_initial.py ...  done
(.venv) synonimous [00:13:19] [~/Desktop/proyecto-irsi-conia/irsi-system-becas] [main *]
-> % flask db upgrade
INFO  [alembic.runtime.migration] Context impl MSSQLImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade  -> 36b81f0d82f1, Initial
```

Fuente: Los autores.

4.5 Cargar datos de prueba (seeds)

Para poblar las tablas de roles, usuario admin, cohorte demo:

Figura 4.8: Comando ejecutar los seeders.

Fuente: Los autores.

Figura 4.9: Resultado del comando para ejecutar los seeders.

Fuente: Los autores.

Ejecutando este comando se tiene acceso inicial con las siguientes credenciales:

Email: admin@example.com

Password: Admin123!

4.6 Ejecutar la aplicación

Define la variable de entorno y arranca:

Figura 5.10: Configuración de variables de entorno para ejecutar la aplicación.

```
export FLASK_APP=run.py
export FLASK_ENV=development   # hot-reload & debug
flask run
o
python run.py
```

Fuente: Los autores.

Accede en tu navegador a: http://localhost:5000

CAPÍTULO 5: ENDPOINTS DE LA API

Tabla 5.1: Endpoints que maneja la API.

Ruta	Método	Descripción
/login	GET/POST	Iniciar sesión
/register	GET/POST	Registrar usuario
/solicitantes/	GET	Listar solicitantes
/solicitantes/create	GET/POST	Crear nuevo solicitante
/solicitantes/<id>	GET	Ver detalle (historial, documentos…)
/solicitantes/<id>/edit	GET/POST	Editar datos del solicitante
/solicitantes/<id>/change_state	POST	Cambiar estado (modal)
/solicitantes/reaplicaciones	GET	Listar reaplicaciones
/solicitantes/reportes	GET	Reportes gráficos y tablas
/becados/	GET	Listar becados activos
/becados/todos	GET	Listar todos los becados
/becados/convert/<id>	POST	Convertir solicitante a becado
/becados/<id>	GET	Detalle becado (timeline, comunicaciones)
/becados/<id>/change_state	POST	Cambiar estado de becado
/becados/communications/<id>	GET/POST	Panel de comunicaciones

Fuente: Los autores.

CONCLUSIÓN

El desarrollo de este sistema ha sido guiado por principios de modularidad, claridad en el diseño y eficiencia en la gestión de datos, dando como resultado una solución tecnológica capaz de satisfacer las necesidades operativas de registro y seguimiento de solicitantes de forma confiable y segura. Gracias a su arquitectura basada en Flask y su integración con SQL Server, el sistema permite manejar grandes volúmenes de información con rapidez y consistencia.

El despliegue automatizado a través de Render y la integración con GitHub aseguran un ciclo de vida de desarrollo ágil y flexible, facilitando futuras actualizaciones y mejoras. Por otro lado, la organización del código en capas claramente definidas (como modelos, servicios, rutas y formularios) permite que otros desarrolladores comprendan rápidamente la estructura y puedan contribuir de manera efectiva.

Este manual técnico proporciona toda la información necesaria para comprender, ejecutar y dar mantenimiento al sistema. Se espera que sirva como base sólida para futuras expansiones, integraciones con otros servicios y mejoras continuas, asegurando la sostenibilidad del proyecto en el tiempo.