SISTEMA DE CAJAS DE AHORROS

Introduccion

El sistema de Caja de Ahorros busca ofrecer una solución financiera confiable y accesible. Su propósito principal es ayudar a los usuarios a gestionar sus ahorros, facilitando los depósitos y retiros, así como la solicitud de créditos. Su desarrollo se realizó en base a tecnologías actuales y se han seguido buenas prácticas de ingeniería de software, con la intención de ofrecer una experiencia de usuario óptima. Además, el sistema integra distintos módulos y asegura un rendimiento escalable en el tiempo.

Arquitectura del sistema

La arquitectura de este sistema sigue una estructura cliente-servidor, con el frontend y el backend operando como componentes independientes pero interconectados. La interacción del usuario es gestionada por el frontend, mientras que el backend supervisa la lógica de negocio y la persistencia de datos. Se comunican a través de una API RESTful que aplica el patrón de arquitectura Modelo-Vista-Controlador (MVC)para delinear las preocupaciones del sistema y facilitar el mantenimiento.

Backend API

El backend del sistema fue creado usando FastAPI, el cual se considera uno de los mejores frameworks de Python. Este se caracteriza por su velocidad, usabilidad y su excelente capacidad para validar datos de forma automática. FastAPI se eligió debido a sus capacidades excepcionales para crear APIs RESTful eficientes y escalables. A continuación, se detallan las partes más importantes del backend:

- **Estructura de directorios del backend**: El backend se organiza de manera modular para asegurar la facilidad de mantenimiento y escalabilidad

- **main.py**: Archivo principal donde se inicia la aplicación FastAPI, se establece la conexión con la base de datos y se configuran las rutas más relevantes.

- **controllers**: Esta carpeta contiene los controladores que se encargan de definir las rutas y patrones HTTP. Cada ruta en el sistema está configurada para ejecutar acciones como registrar un usuario, depositar fondos, entre otros

- **models**: En este ítem, se definen las entidades del sistema, por ejemplo, Socio, Crédito y Transacciones. Los modelos definidos en esta carpeta coinciden con las colecciones en la base de datos MongoDB.

- **repositories**: Este ítem contiene los elementos con la base de datos. En este caso, la base de datos almacena los créditos, transacciones y los usuarios, este ítem está a cargo de las solicitudes y respuestas de estos datos.

- schemas: Describe las estructuras de datos que el sistema utiliza para verificar la exactitud de la información cuando se recibe o se envía. Los esquemas se basan en Pydantic, que proporciona el beneficio de automatizar la validación de datos para los datos del backend.

- **services**: Esta carpeta contiene la lógica de negocio del sistema. Por ejemplo, aquí es donde se aplican las reglas de negocio para las operaciones de depósito y retiro, como no permitir retiros que excedan el saldo disponible.

- **tests**: Contiene las pruebas automatizadas que verifican el correcto funcionamiento de cada componente del sistema. Las pruebas abarcan desde rutas de API hasta funciones comerciales específicas.

Rutas de API: Las rutas establecidas en el backend permiten que el frontend se interfase con el sistema. Incluyen:

- **POST /auth/register**: Registra nuevos usuarios. Recibe su nombre, correo electrónico y contraseña.

- **POST /auth/login**: Autentica a los usuarios registrados y emite un token JWT para la sesión.

- POST /ahorros/depositar: Permite a un usuario hacer un depósito en su cuenta de ahorros.

- **POST /ahorros/retirar**: Permite a un usuario hacer un retiro de su cuenta de ahorros.

- **POST /creditos/solicitar**: Permite a un usuario solicitar un préstamo.

- GET /creditos/historial/{socio_id}: Devuelve el historial crediticio del usuario identificado por el **socio_id**.

Frontend: Apliacion web

La interfaz de la aplicación se maneja en React. Esta es una popular biblioteca de JavaScript para el desarrollo de UI. Se seleccionó React para este proyecto por su beneficio de construir componentes modulares. Procedo a señalar los puntos del frontend en React**.**

-Estructura de directorios del frontend: se organizan todos los componentes.

-**src**: En este directorio se guardan todos los archivos del proyecto. Existen diferentes carpetas para sus funcionalidades, como la de components para los componentes de UI o hooks/para la reutilizable.

-**public**: Se encarga de almacenar el **index.html** que es el archivo que lanza la aplicación React.

-**package.json**: Aquí se encuentra la definición de las dependencias del proyecto, los scripts para construirlo y ejecutarlo.

-**vite.config.js**: archivo del proyecto donde se encuentra la configuración de **Vite**, el bundler utilizado para construir la aplicación de forma ágil y optimizada.

-Interacción con el backend: se obtiene la información haciendo uso de **Axios** o **fetch** a las rutas del backend. Se reciben los datos, se almacenan en el estado de la aplicación y se presentan al usuario de forma dinámica.

-**Componentes Frontend**: La sección orientada al usuario está compuesta por diferentes componentes habilitados para la interacción del usuario e incluye registro, inicio de sesión y gestión de cuentas de ahorro y préstamos. Estos componentes son dinámicos, lo que significa que la interfaz de usuario se actualiza automáticamente según los cambios que ocurren en los datos.

Base de datos MongoDB

El sistema hace uso de MongoDB, una base de datos NoSQL, para almacenar datos de usuarios, transacciones, así como créditos. Se decidió por MongoDB, dado que poseía una capacidad de manejo de datos no estructurados voluminosos, de una manera ágil y no rígida. Se detallará la interacción con la base de datos.

- **Modelo de datos**: En MongoDB, la información se organiza en colecciones. Por ejemplo, la información de los usuarios se halla en la colección **socios** y los créditos y transacciones se almacenan en sus respectivas colecciones. En la colección **socios**, cada documento corresponde a un usuario con atributos de nombre, email, y saldo.

- **Operaciones CRUD**: El backend ejecuta operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para manipular la información en la base de datos. Por ejemplo, en el caso de la recarga de saldo, el saldo se actualiza en la colección socios y la transacción se registra en la colección transacciones

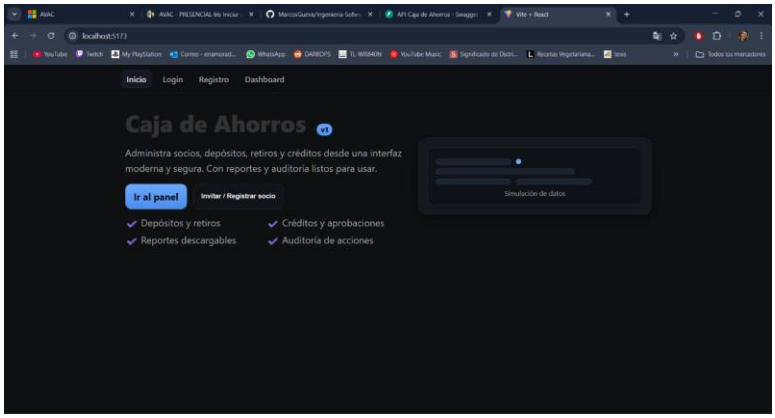
- **Escalabilidad**: MongoDB soporta una escalabilidad horizontal, por lo cual maneja eficientemente el volumen de datos. Esto es muy importante para el caso de un sistema que en un futuro podría crecer.

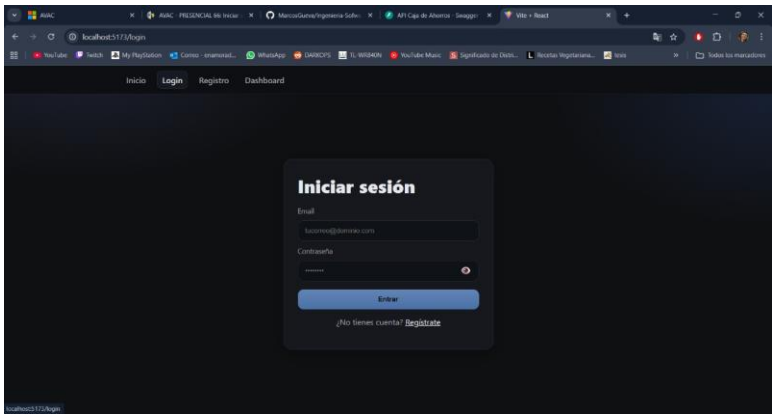
Flujo de integración de API

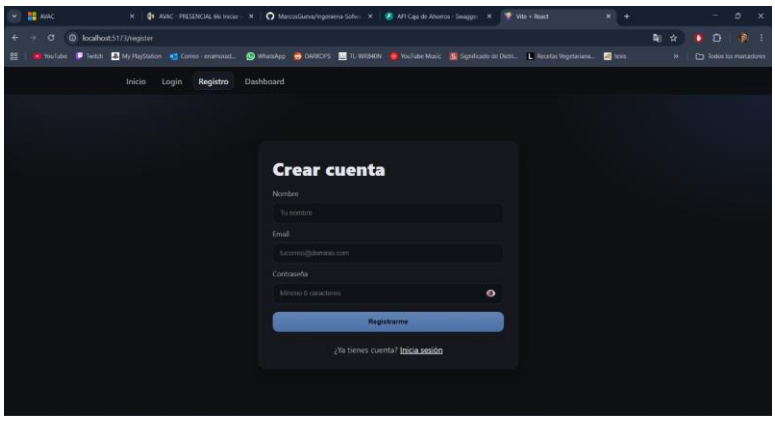
La integración del frontend con el backend sigue el patrón tradicional de API RESTful. A continuación se detalla el flujo de integración:

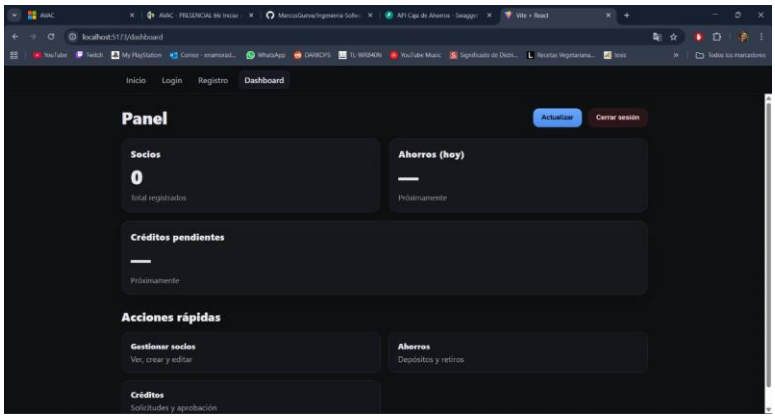
1. **POST /auth/register**: El frontend hace el llamado y el backend se encarga de procesar la solicitud.
2. **Se valida la solicitud:** se valida la información ingresada y, en caso de ser correcto, se efectúa la operación de agregar el nuevo usuario a la base de datos.
3. El backend, una vez realizado el procedimiento, envía una notificación de éxito o falla con base en si la operación se ejecutó de manera correcta o no.
4. El usuario del frontend, de acuerdo a la notificación procesada, se le muestra el resultado de la operación, ya sean mensajes de éxito o error.
5. Para cada operación se debe seguir este procedimiento para cada interacción entre el frontend y el backend y el resto de las operaciones del sistema (depósitos, retiros, créditos).

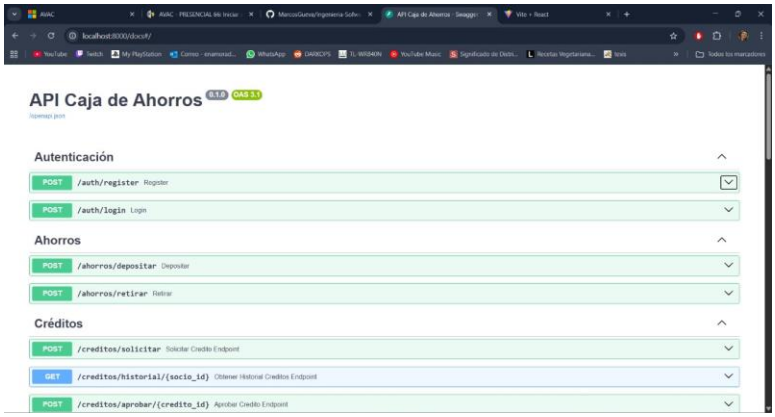
Anexo

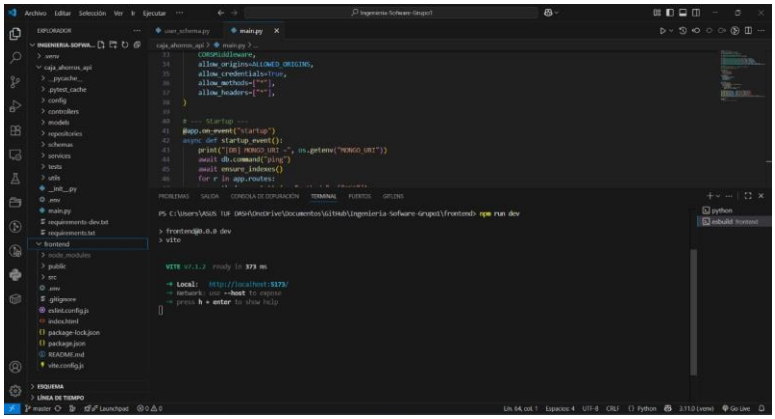

Inicio
Login
Registro
Dashboard
Caja de Ahorros
v1
Administra socios, depósitos, retiros y créditos desde una interfaz moderna y segura. Con reportes y auditoría listos para usar.
Ir al panel
Invitar / Registrar socio
Simulación de datos
Depósitos y retiros
Créditos y aprobaciones
Reportes descargables
Auditoría de acciones

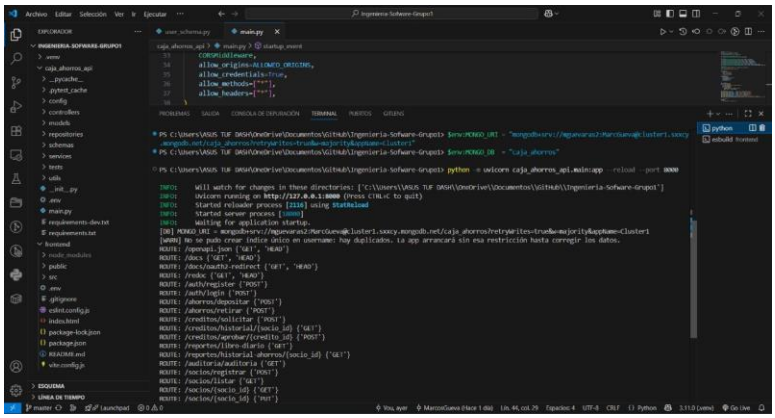

Inicio
Login
Registro
Dashboard
Iniciar sesión
Email
Contraseña
Entrar
¿No tienes cuenta? Regístrate

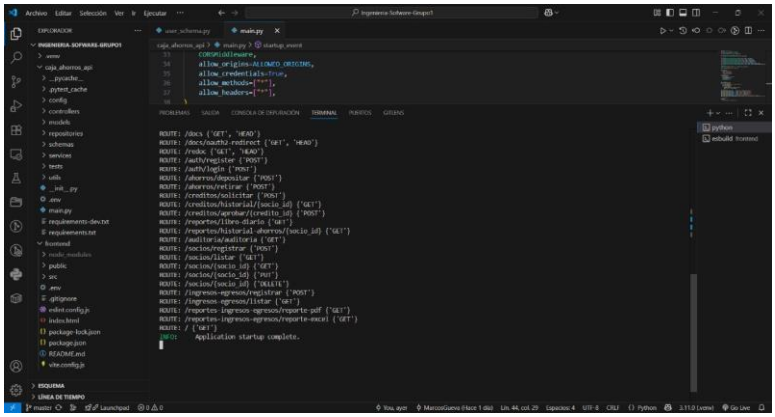

Inicio
Login
Registro
Dashboard
Crear cuenta
Nombre
Email
Contraseña
Registrarme
¿Ya tienes cuenta? Inicia sesión


localhost:5173/dashboard
Inicio
Login
Registro
Dashboard
Panel
Actualizar
Cerrar sesión
Socios
0
Total registrados
Ahorros (hoy)
—
Próximamente
Créditos pendientes
—
Próximamente
Acciones rápidas
Gestionar socios
Ver, crear y editar
Ahorros
Depósitos y retiros
Créditos
Solicitudes y aprobación


localhost:8000/docs#/
API Caja de Ahorros
0.1.0
OAS 3.1
Autenticación
POST /auth/register Register
POST /auth/login Login
Ahorros
POST /ahorros/depositar Depositar
POST /ahorros/retirar Retirar
Créditos
POST /creditos/solicitar Solicitar Credito Endpoint
GET /creditos/historial/{socio_id} Obtener Historial Creditos Endpoint
POST /creditos/aprobar/{credito_id} Aprobar Credito Endpoint


PS C:\Users\ASUS TUF DASH\OneDrive\Documentos\GitHub\Ingenieria-Sofware-Grupo1\frontend> npm run dev
> frontend@0.0.0 dev
> vite
VITE v7.1.2 ready in 373 ms
➜ Local: http://localhost:5173/
➜ Network: use --host to expose
➜ press h + enter to show help


Archivo Editar Selección Ver Ir Ejecutar
EXPLORADOR
INGENIERIA-SOFWARE-GRUPO1
caja_ahorros_api
frontend
ESQUEMA
LÍNEA DE TIEMPO
user_schema.py
main.py
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS GITLENS
PS C:\Users\ASUS TUF DASH\OneDrive\Documentos\GitHub\Ingenieria-Sofware-Grupo1> $env:MONGO_URI = "mongodb+srv://mgaveras2:MarcGueva@cluster1.sxocy.mongodb.net/caja_ahorros?retryWrites=true&w=majority&appName=Cluster1"
PS C:\Users\ASUS TUF DASH\OneDrive\Documentos\GitHub\Ingenieria-Sofware-Grupo1> $env:MONGO_DB = "caja_ahorros"
PS C:\Users\ASUS TUF DASH\OneDrive\Documentos\GitHub\Ingenieria-Sofware-Grupo1> python -m uvicorn caja_ahorros_api.main:app --reload --port 8000
INFO: Will watch for changes in these directories: ['C:\\Users\\ASUS TUF DASH\\OneDrive\\Documentos\\GitHub\\Ingenieria-Sofware-Grupo1']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [2136] using StatReload
INFO: Started server process [19008]
INFO: Waiting for application startup.
[DB] MONGO_URI = mongodb+srv://mgaveras2:MarcGueva@cluster1.sxocy.mongodb.net/caja_ahorros?retryWrites=true&w=majority&appName=Cluster1
[WARN] No se pudo crear índice único en username: hay duplicados. La app arrancará sin esa restricción hasta corregir los datos.
ROUTE: /openapi.json {'GET', 'HEAD'}
ROUTE: /docs {'GET', 'HEAD'}
ROUTE: /docs/oauth2-redirect {'GET', 'HEAD'}
ROUTE: /redoc {'GET', 'HEAD'}
ROUTE: /auth/register {'POST'}
ROUTE: /auth/login {'POST'}
ROUTE: /ahorros/depositar {'POST'}
ROUTE: /ahorros/retirar {'POST'}
ROUTE: /creditos/solicitar {'POST'}
ROUTE: /creditos/historial/{socio_id} {'GET'}
ROUTE: /creditos/aprobar/{credito_id} {'POST'}
ROUTE: /reportes/libro-diario {'GET'}
ROUTE: /reportes/historial-ahorros/{socio_id} {'GET'}
ROUTE: /auditoria/auditoria {'GET'}
ROUTE: /socios/registrar {'POST'}
ROUTE: /socios/listar {'GET'}
ROUTE: /socios/{socio_id} {'GET'}
ROUTE: /socios/{socio_id} {'PUT'}
python
esbuild frontend


Archivo Editar Selección Ver Ir Ejecutar
EXPLORADOR
INGENIERIA-SOFWARE-GRUPO1
caja_ahorros_api
frontend
ESQUEMA
LÍNEA DE TIEMPO
user_schema.py
main.py
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS GITLENS
ROUTE: /docs {'GET', 'HEAD'}
ROUTE: /docs/oauth2-redirect {'GET', 'HEAD'}
ROUTE: /redoc {'GET', 'HEAD'}
ROUTE: /auth/register {'POST'}
ROUTE: /auth/login {'POST'}
ROUTE: /ahorros/depositar {'POST'}
ROUTE: /ahorros/retirar {'POST'}
ROUTE: /creditos/solicitar {'POST'}
ROUTE: /creditos/historial/{socio_id} {'GET'}
ROUTE: /creditos/aprobar/{credito_id} {'POST'}
ROUTE: /reportes/libro-diario {'GET'}
ROUTE: /reportes/historial-ahorros/{socio_id} {'GET'}
ROUTE: /auditoria/auditoria {'GET'}
ROUTE: /socios/registrar {'POST'}
ROUTE: /socios/listar {'GET'}
ROUTE: /socios/{socio_id} {'GET'}
ROUTE: /socios/{socio_id} {'PUT'}
ROUTE: /socios/{socio_id} {'DELETE'}
ROUTE: /ingresos-egresos/registrar {'POST'}
ROUTE: /ingresos-egresos/listar {'GET'}
ROUTE: /reportes-ingresos-egresos/reporte-pdf {'GET'}
ROUTE: /reportes-ingresos-egresos/reporte-excel {'GET'}
ROUTE: / {'GET'}
INFO: Application startup complete.
python
esbuild frontend