ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

SOFTWARE

INTEGRANTES:

Jheyson Monje 7188

Jonathan Chamorro 7167

CURSO:

Octavo “A”

ASIGNATURA:

Aplicaciones Informáticas II

DOCENTE:

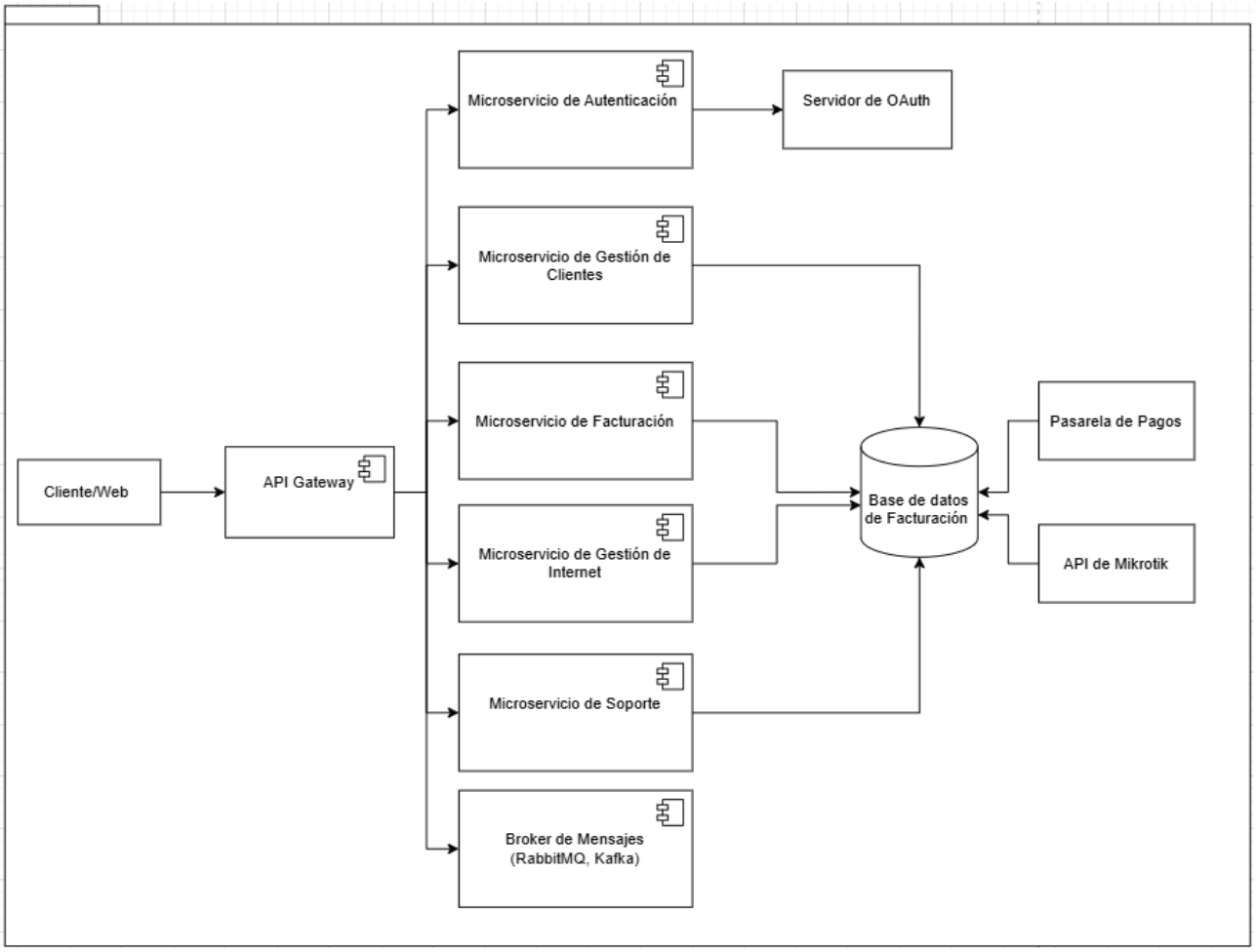
Ing. Julio Santillan

TEMA:

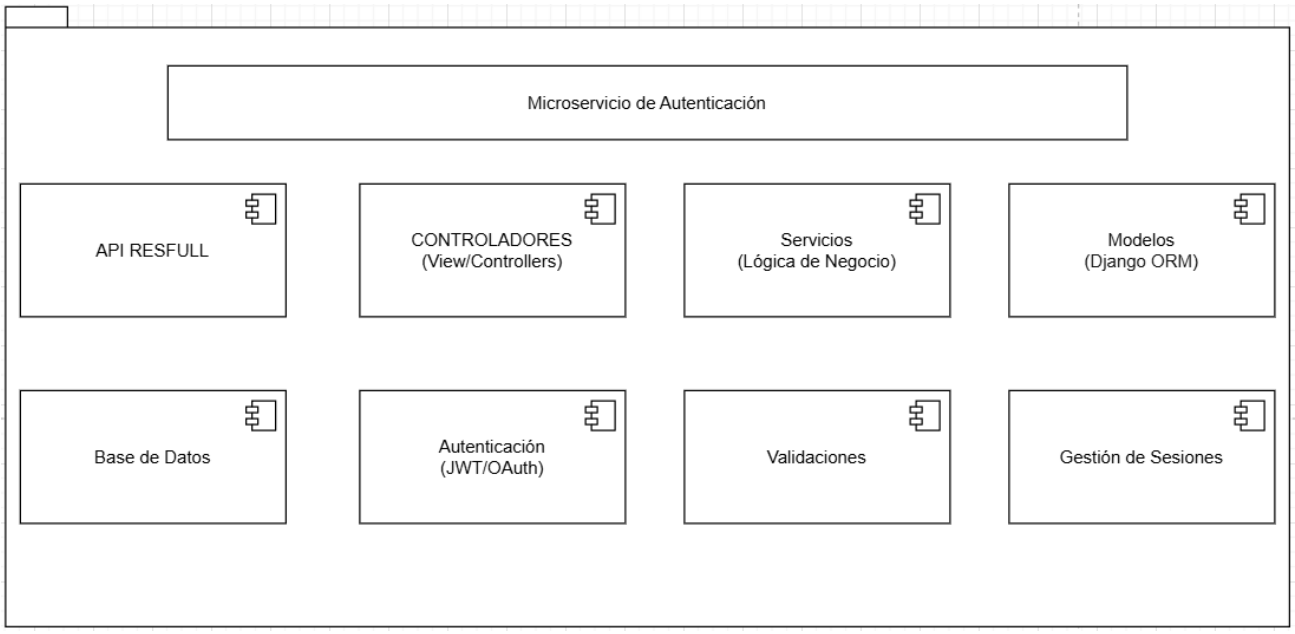
Arquitectura Basada en Microservicios

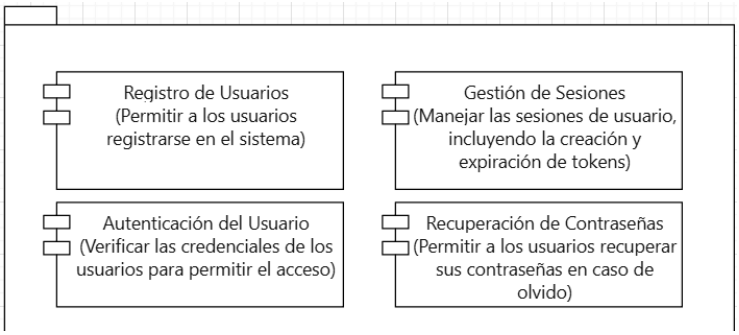
07 octubre 2024 - 28 febrero 202

Arquitectura Basada En Microservicios



Microservicio de Autenticación





Descripción de los Componentes

1. **API RESTful**
   - **Tecnologías**: Las posibles tecnologías que nos pueden ayudar al desarrollo serían Django REST Framework (para Django), Flask-RESTful (para Flask)
   - **Funcionalidad**: Nos va a proporciona endpoints para interactuar con el microservicio, permitiendo operaciones como registro de usuarios, autenticación y recuperación de contraseñas. Estos endpoints son utilizados por el frontend y otros microservicios para gestionar la autenticación y autorización.
2. **Controladores (Views/Controllers)**

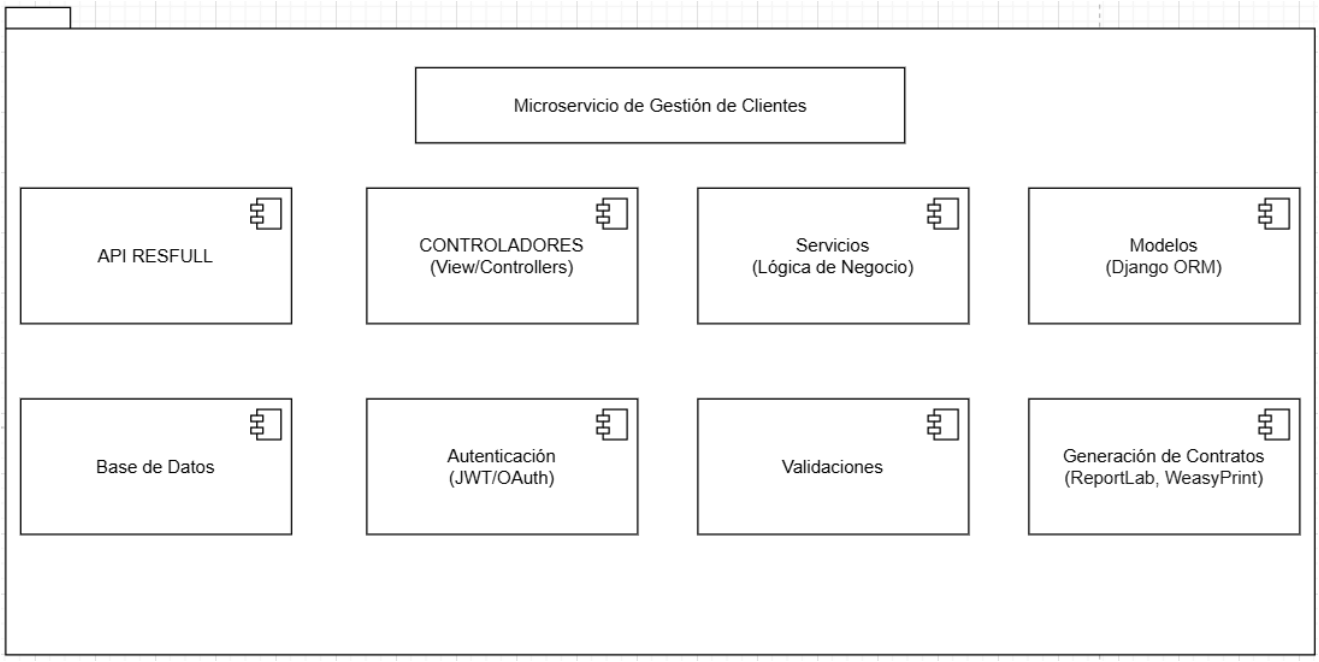
   Funcionalidad: Van a manejar las solicitudes HTTP entrantes, interactúan con los servicios y devuelven las respuestas adecuadas. Los controladores aseguran que las solicitudes de autenticación y gestión de usuarios se procesen correctamente.
3. **Servicios (Business Logic)**

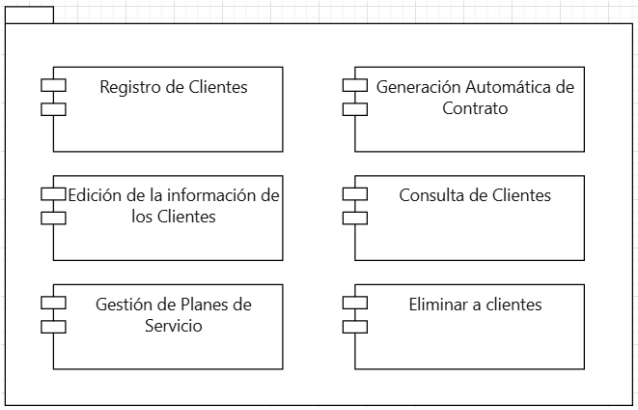
   Funcionalidad: Contienen la lógica de negocio del microservicio, como la gestión de usuarios, la verificación de credenciales y la generación de tokens de sesión. Los servicios encapsulan la lógica compleja y las reglas de negocio, facilitando su reutilización y mantenimiento.
4. **Modelos**
   - **Tecnologías**: Django ORM (para Django), SQLAlchemy (para Flask)
   - **Funcionalidad**: Definen la estructura de los datos y las relaciones entre ellos, y proporcionan una capa de abstracción para interactuar con la base de datos. Los modelos representan las entidades del dominio, como usuarios y sesiones, y permiten realizar operaciones sobre la base de datos de manera sencilla.
5. **Base de Datos**
   - **Tecnologías**: PostgreSQL
   - **Funcionalidad**: Almacena la información de los usuarios, incluyendo datos de registro y credenciales. La base de datos es el repositorio central de datos, asegurando la persistencia y consistencia de la información.
6. **Autenticación**
   - **Tecnologías**: JWT (JSON Web Tokens), OAuth2

- **Funcionalidad**: Gestiona la autenticación y autorización de los usuarios que interactúan con el microservicio. La autenticación asegura que solo usuarios autorizados puedan acceder a las funcionalidades del sistema.

7. **Validación**
   - **Tecnologías**: Cerberus, Marshmallow
   - **Funcionalidad**: Valida los datos de entrada para asegurar que cumplen con los requisitos antes de procesarlos. La validación previene errores y asegura la integridad de los datos.
8. **Gestión de Sesiones**
   - **Tecnologías**: Redis
   - **Funcionalidad**: Maneja las sesiones de usuario, incluyendo la creación y expiración de tokens de acceso. Redis se utiliza para almacenar las sesiones de manera eficiente y rápida.

Microservicio de Gestión de Clientes





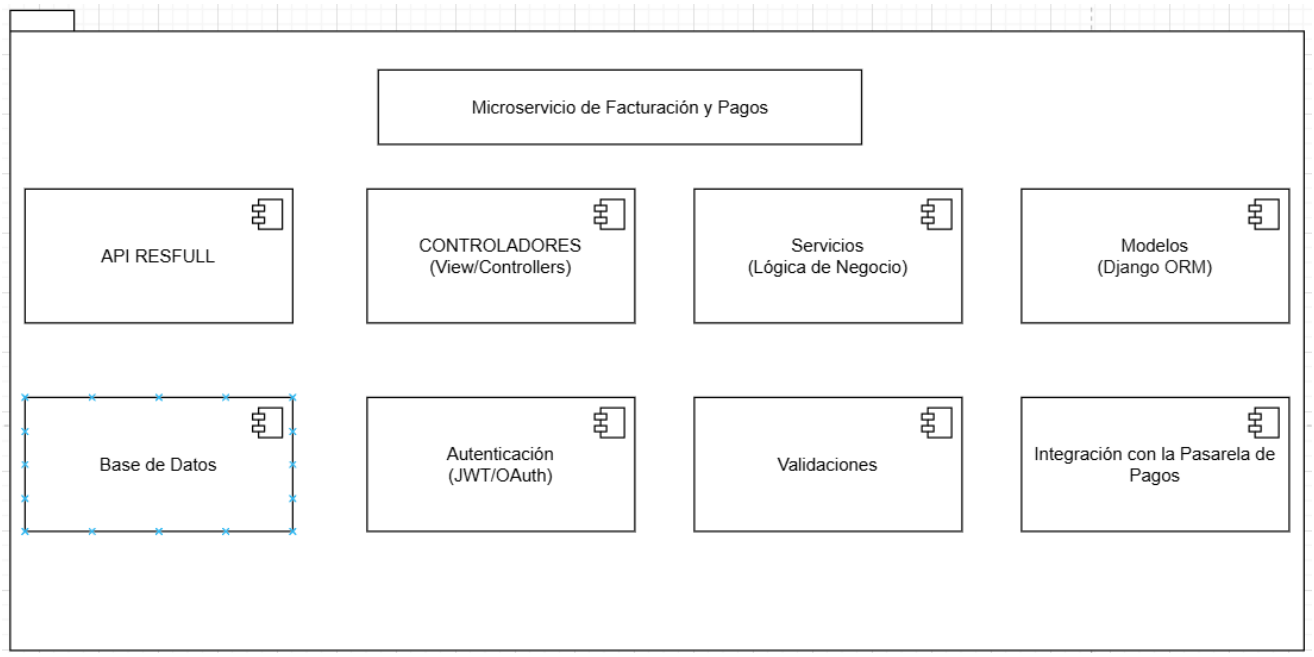
Descripción de los Componentes

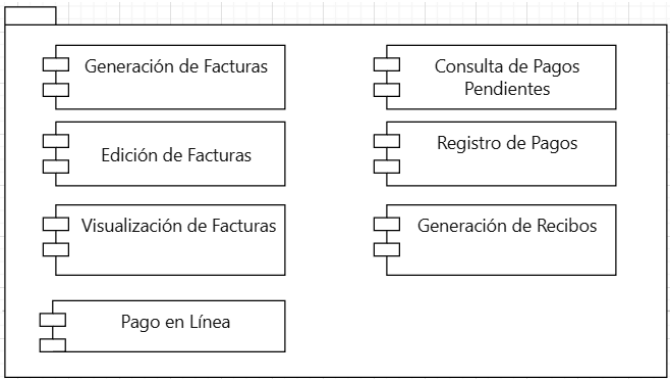
1. **API RESTful**
   - **Tecnologías**: Django REST Framework (para Django), Flask-RESTful (para Flask)
   - **Funcionalidad**: Proporciona endpoints para interactuar con el microservicio, permitiendo operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los clientes. Estos endpoints son utilizados por el frontend y otros microservicios para realizar operaciones sobre los datos de clientes.
2. **Controladores (Views/Controllers)**
   - **Funcionalidad**: Manejan las solicitudes HTTP entrantes, interactúan con los servicios y devuelven las respuestas adecuadas. Los controladores actúan como intermediarios entre la API RESTful y la lógica de negocio, asegurando que las solicitudes se procesen correctamente.
3. **Servicios (Business Logic)**

   Funcionalidad: Contienen la lógica de negocio del microservicio, como la gestión de clientes, la asignación de planes de servicio y la generación de contratos. Los servicios encapsulan la lógica compleja y las reglas de negocio, facilitando su reutilización y mantenimiento.
4. **Modelos**
   - **Tecnologías**: Django ORM (para Django), SQLAlchemy (para Flask)
   - **Funcionalidad**: Definen la estructura de los datos y las relaciones entre ellos, y proporcionan una capa de abstracción para interactuar con la base de datos. Los modelos representan las entidades del dominio, como clientes y contratos, y permiten realizar operaciones sobre la base de datos de manera sencilla.
5. **Base de Datos**
   - **Tecnologías**: PostgreSQL
   - **Funcionalidad**: Almacena la información de los clientes, incluyendo datos personales, contratos y planes de servicio. La base de datos es el repositorio central de datos, asegurando la persistencia y consistencia de la información.
6. **Autenticación**
   - **Tecnologías**: JWT (JSON Web Tokens), OAuth2
   - **Funcionalidad**: Gestiona la autenticación y autorización de los usuarios que interactúan con el microservicio. La autenticación asegura que solo usuarios autorizados puedan acceder a las funcionalidades del sistema.
7. **Validación**
   - **Tecnologías**: Cerberus, Marshmallow
   - **Funcionalidad**: Valida los datos de entrada para asegurar que cumplen con los requisitos antes de procesarlos. La validación previene errores y asegura la integridad de los datos.

8. **Generación de Contratos**
    - **Tecnologías**: ReportLab, WeasyPrint
    - **Funcionalidad**: Genera contratos en formato PDF cuando se registra o modifica un cliente. Esta funcionalidad automatiza la creación de documentos legales, asegurando que los contratos estén siempre actualizados y disponibles.

Microservicio de Facturación y Pagos





Descripción de los Componentes

1. **API RESTful**
    - **Tecnologías**: Django REST Framework (para Django), Flask-RESTful (para Flask)
    - **Funcionalidad**: Proporciona endpoints para interactuar con el microservicio, permitiendo operaciones CRUD sobre las facturas y pagos. Estos endpoints son utilizados por el frontend y otros microservicios para gestionar la facturación y los pagos.
2. **Controladores (Views/Controllers)**

    Funcionalidad: Manejan las solicitudes HTTP entrantes, interactúan con los servicios y devuelven las respuestas adecuadas. Los controladores aseguran que las solicitudes de facturación y pagos se procesen correctamente.

3. **Servicios (Business Logic)**

   Funcionalidad: Contienen la lógica de negocio del microservicio, como la generación de facturas, la gestión de pagos y la integración con pasarelas de pago. Los servicios encapsulan la lógica compleja y las reglas de negocio, facilitando su reutilización y mantenimiento.

4. **Modelos**

   - **Tecnologías**: Django ORM (para Django), SQLAlchemy (para Flask)
   - **Funcionalidad**: Definen la estructura de los datos y las relaciones entre ellos, y proporcionan una capa de abstracción para interactuar con la base de datos. Los modelos representan las entidades del dominio, como facturas y pagos, y permiten realizar operaciones sobre la base de datos de manera sencilla.

5. **Base de Datos**

   - **Tecnologías**: PostgreSQL
   - **Funcionalidad**: Almacena la información de las facturas y pagos, incluyendo detalles de los montos, fechas de vencimiento y estados de pago. La base de datos es el repositorio central de datos, asegurando la persistencia y consistencia de la información.

6. **Autenticación**

   - **Tecnologías**: JWT (JSON Web Tokens), OAuth2
   - **Funcionalidad**: Gestiona la autenticación y autorización de los usuarios que interactúan con el microservicio. La autenticación asegura que solo usuarios autorizados puedan acceder a las funcionalidades del sistema.

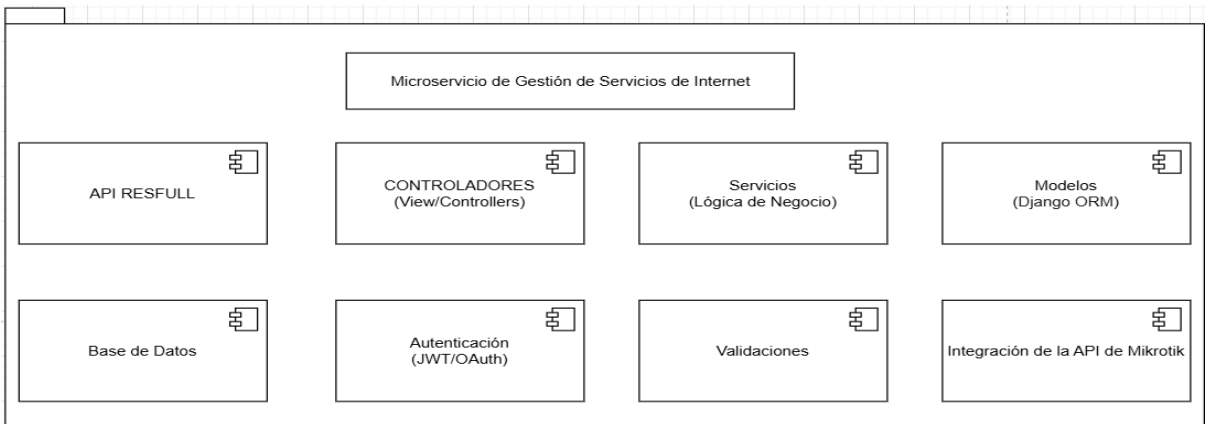
7. **Validación**

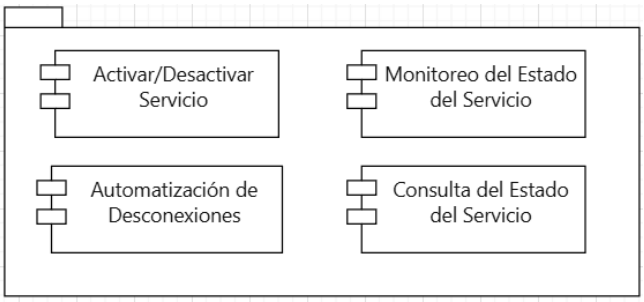
   - **Tecnologías**: Cerberus, Marshmallow
   - **Funcionalidad**: Valida los datos de entrada para asegurar que cumplen con los requisitos antes de procesarlos. La validación previene errores y asegura la integridad de los datos.

8. **Integración de Pasarela de Pago**

   - **Tecnologías**: Stripe, PayPal
   - **Funcionalidad**: Procesa pagos en línea y actualiza el estado de las facturas. La integración con pasarelas de pago como paypal

Microservicio de Gestión de Servicios de Internet





Descripción de los Componentes

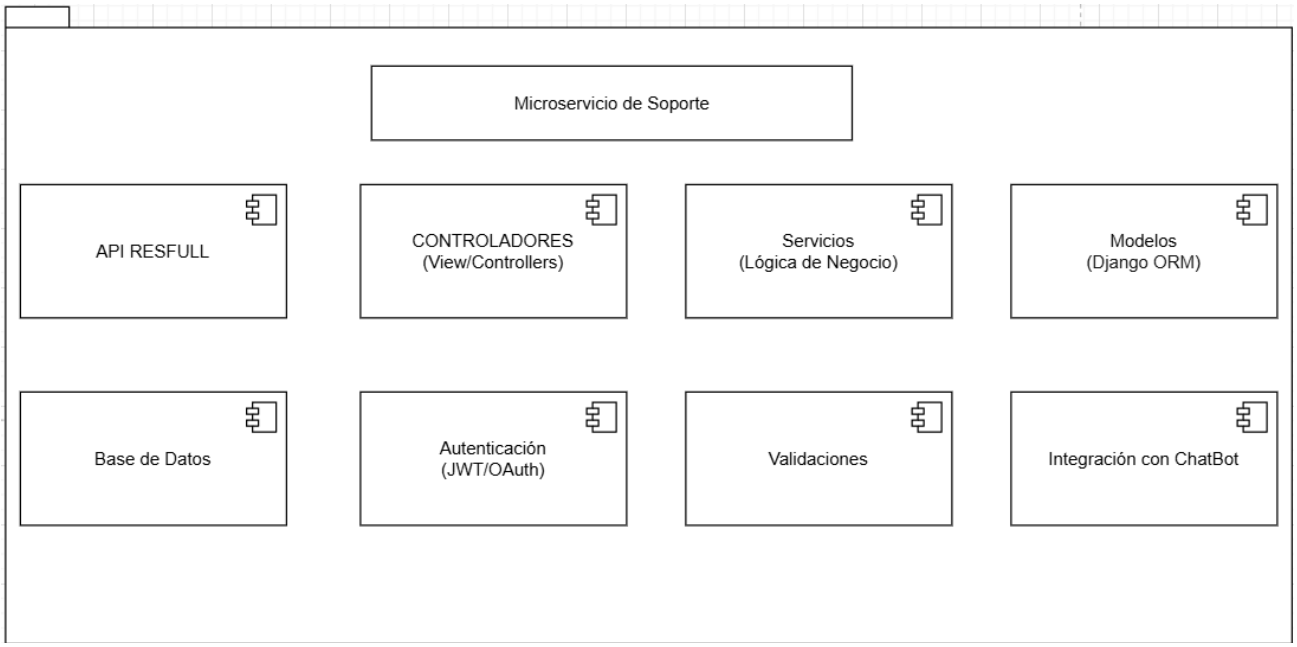
1. **API RESTful**
   - **Tecnologías**: FastAPI, Flask-RESTful, Django
   - **Funcionalidad**: Proporciona endpoints para interactuar con el microservicio, permitiendo operaciones CRUD sobre los servicios de internet. Estos endpoints son utilizados por el frontend y otros microservicios para gestionar los servicios de internet.
2. **Controladores (Views/Controllers)**

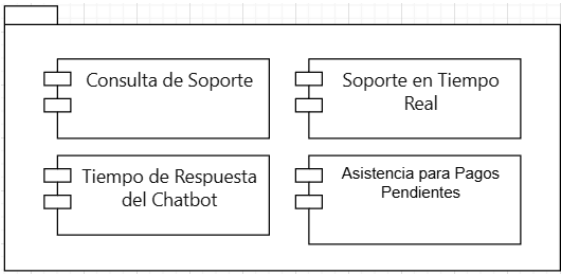
   Funcionalidad: Manejan las solicitudes HTTP entrantes, interactúan con los servicios y devuelven las respuestas adecuadas. Los controladores aseguran que las solicitudes de activación, desactivación y monitoreo de servicios se procesen correctamente.
3. **Servicios (Business Logic)**

   Funcionalidad: Contienen la lógica de negocio del microservicio, como la activación y desactivación de servicios de internet, y la automatización de desconexiones. Los servicios encapsulan la lógica compleja y las reglas de negocio, facilitando su reutilización y mantenimiento.
4. **Modelos**
   - **Tecnologías**: SQLAlchemy
   - **Funcionalidad**: Definen la estructura de los datos y las relaciones entre ellos, y proporcionan una capa de abstracción para interactuar con la base de datos. Los modelos representan las entidades del dominio, como los servicios de internet, y permiten realizar operaciones sobre la base de datos de manera sencilla.
5. **Base de Datos**
   - **Tecnologías**: PostgreSQL
   - **Funcionalidad**: Almacena la información de los servicios de internet, incluyendo detalles de activación, desactivación y monitoreo. La base de datos es el repositorio central de datos, asegurando la persistencia y consistencia de la información.
6. **Autenticación**
   - **Tecnologías**: JWT (JSON Web Tokens), OAuth2
   - **Funcionalidad**: Gestiona la autenticación y autorización de los usuarios que interactúan con el microservicio. La autenticación asegura que solo usuarios autorizados puedan acceder a las funcionalidades del sistema.

7. **Validación**
   - **Tecnologías**: Cerberus, Marshmallow
   - **Funcionalidad**: Valida los datos de entrada para asegurar que cumplen con los requisitos antes de procesarlos. La validación previene errores y asegura la integridad de los datos.
8. **Integración de API de Mikrotik**
   - **Tecnologías**: RouterOS API
   - **Funcionalidad**: Gestiona la activación y desactivación de los servicios de internet. La integración con la API de Mikrotik permite controlar los dispositivos de red y asegurar que los servicios se gestionen correctamente.

Microservicio de Soporte





Descripción de los Componentes y Tecnologías

1. **API RESTful**
   - **Tecnologías**: Django REST Framework (para Django), Flask-RESTful (para Flask), FastAPI
   - **Funcionalidad**: Proporciona endpoints para interactuar con el microservicio, permitiendo operaciones CRUD (Crear, Leer, Actualizar, Eliminar).
2. **Controladores (Views/Controllers)**

   Funcionalidad: Manejan las solicitudes HTTP entrantes, interactúan con los servicios y devuelven las respuestas adecuadas.

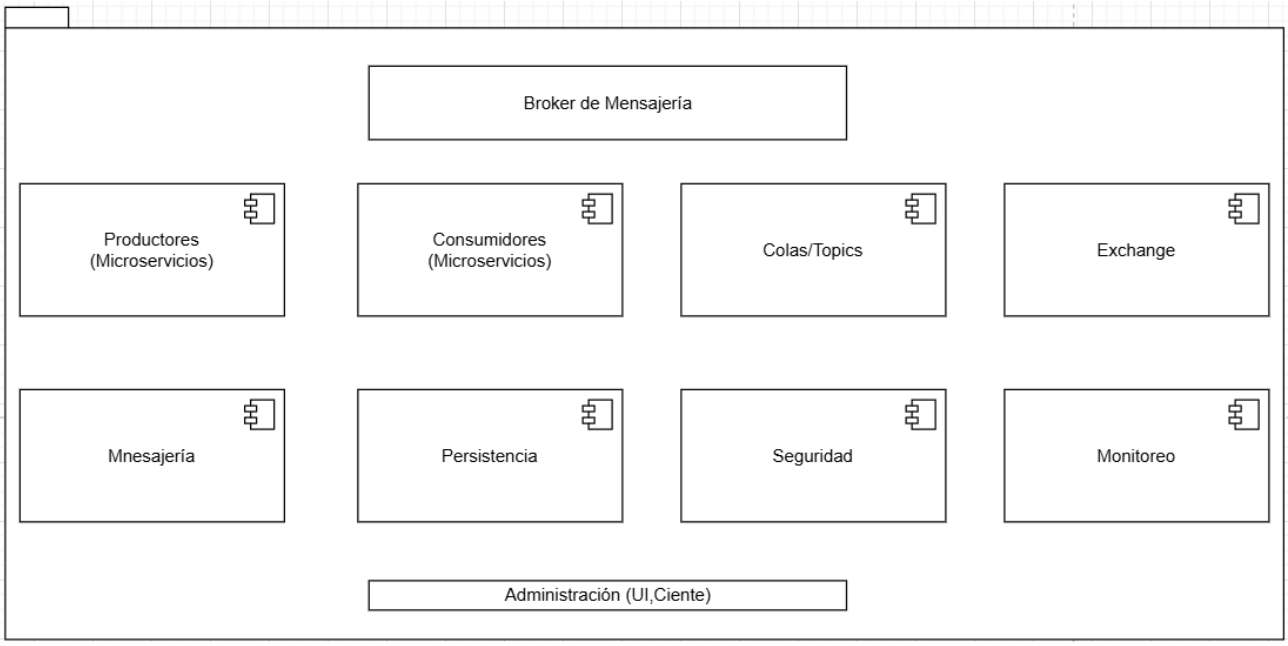
3. **Servicios (Business Logic)**

   Funcionalidad: Contienen la lógica de negocio del microservicio, como la gestión de clientes, la asignación de planes de servicio, la generación de contratos, etc.

4. **Modelos**
   - **Tecnologías**: Django ORM (para Django), SQLAlchemy (para Flask y FastAPI)
   - **Funcionalidad**: Definen la estructura de los datos y las relaciones entre ellos, y proporcionan una capa de abstracción para interactuar con la base de datos.
5. **Base de Datos**
   - **Tecnologías**: PostgreSQL, MongoDB
   - **Funcionalidad**: Almacena la información relevante para cada microservicio, como datos de clientes, facturación, servicios de internet y soporte.
6. **Autenticación**
   - **Tecnologías**: JWT (JSON Web Tokens), OAuth2
   - **Funcionalidad**: Gestiona la autenticación y autorización de los usuarios que interactúan con el microservicio.
7. **Validación**
   - **Tecnologías**: Cerberus, Marshmallow
   - **Funcionalidad**: Valida los datos de entrada para asegurar que cumplen con los requisitos antes de procesarlos.
8. **Chatbot**

Funcionalidad: Encargado de resolver las inquietudes de los clientes darles asistencia en tiempo real

Broker de Mensajería



Descripción de los Componentes

1. **Productores**
   - **Tecnologías**: Microservicios que envían mensajes al Broker de Mensajes.
   - **Funcionalidad**: Los productores son microservicios que generan mensajes y los envían al Broker de Mensajes para ser procesados por otros servicios. Estos mensajes pueden contener datos en formatos como JSON, Avro o Protobuf.
2. **Consumidores**
   - **Tecnologías**: Microservicios que reciben mensajes del Broker de Mensajes.
   - **Funcionalidad**: Los consumidores son microservicios que reciben y procesan mensajes del Broker de Mensajes. Estos servicios suscriben a colas o topics específicos para recibir los mensajes relevantes.
3. **Colas/Topics**
   - **Tecnologías**: RabbitMQ (colas), Kafka (topics)
   - **Funcionalidad**: Las colas en RabbitMQ y los topics en Kafka son estructuras que almacenan los mensajes hasta que son consumidos. Las colas aseguran que los mensajes se entreguen en el orden en que fueron enviados, mientras que los topics permiten la publicación y suscripción de mensajes.
4. **Exchanges**
   - **Tecnologías**: RabbitMQ, Kafka
   - **Funcionalidad**: Los exchanges en RabbitMQ y los mecanismos de publicación en Kafka determinan cómo se enrutan los mensajes a las colas o topics. Los exchanges pueden ser de diferentes tipos, como direct, fanout, topic o headers, cada uno con su propia lógica de enrutamiento.
5. **Mensajes**
   - **Tecnologías**: JSON, Avro, Protobuf
   - **Funcionalidad**: Los mensajes son las unidades de datos que se envían entre los microservicios a través del Broker de Mensajes. Pueden estar en diferentes formatos, como JSON para simplicidad, Avro para esquemas definidos y Protobuf para eficiencia.
6. **Persistencia**
   - **Tecnologías**: Disco, Base de Datos
   - **Funcionalidad**: La persistencia asegura que los mensajes se almacenen de manera duradera hasta que sean procesados. Esto puede incluir almacenamiento en disco o en una base de datos, dependiendo de la configuración del Broker de Mensajes.
7. **Seguridad**
   - **Tecnologías**: TLS, ACLs (Listas de Control de Acceso)
   - **Funcionalidad**: La seguridad incluye la encriptación de los mensajes en tránsito mediante TLS y la gestión de permisos mediante ACLs para asegurar que solo los servicios autorizados puedan enviar y recibir mensajes.

8. **Monitoreo**
   - **Tecnologías**: Prometheus, Grafana
   - **Funcionalidad**: El monitoreo permite supervisar el rendimiento y el estado del Broker de Mensajes. Herramientas como Prometheus y Grafana se utilizan para recopilar y visualizar métricas, ayudando a detectar y resolver problemas rápidamente.
9. **Administración**
   - **Tecnologías**: UI (Interfaz de Usuario), CLI (Interfaz de Línea de Comandos)
   - **Funcionalidad**: La administración incluye herramientas para gestionar y configurar el Broker de Mensajes. Esto puede hacerse a través de una interfaz de usuario web o mediante comandos en la línea de comandos.

ESTRUCTURA DE CARPETAS PARA EL BACK-END CON DJANGO

plaintext Copiar código

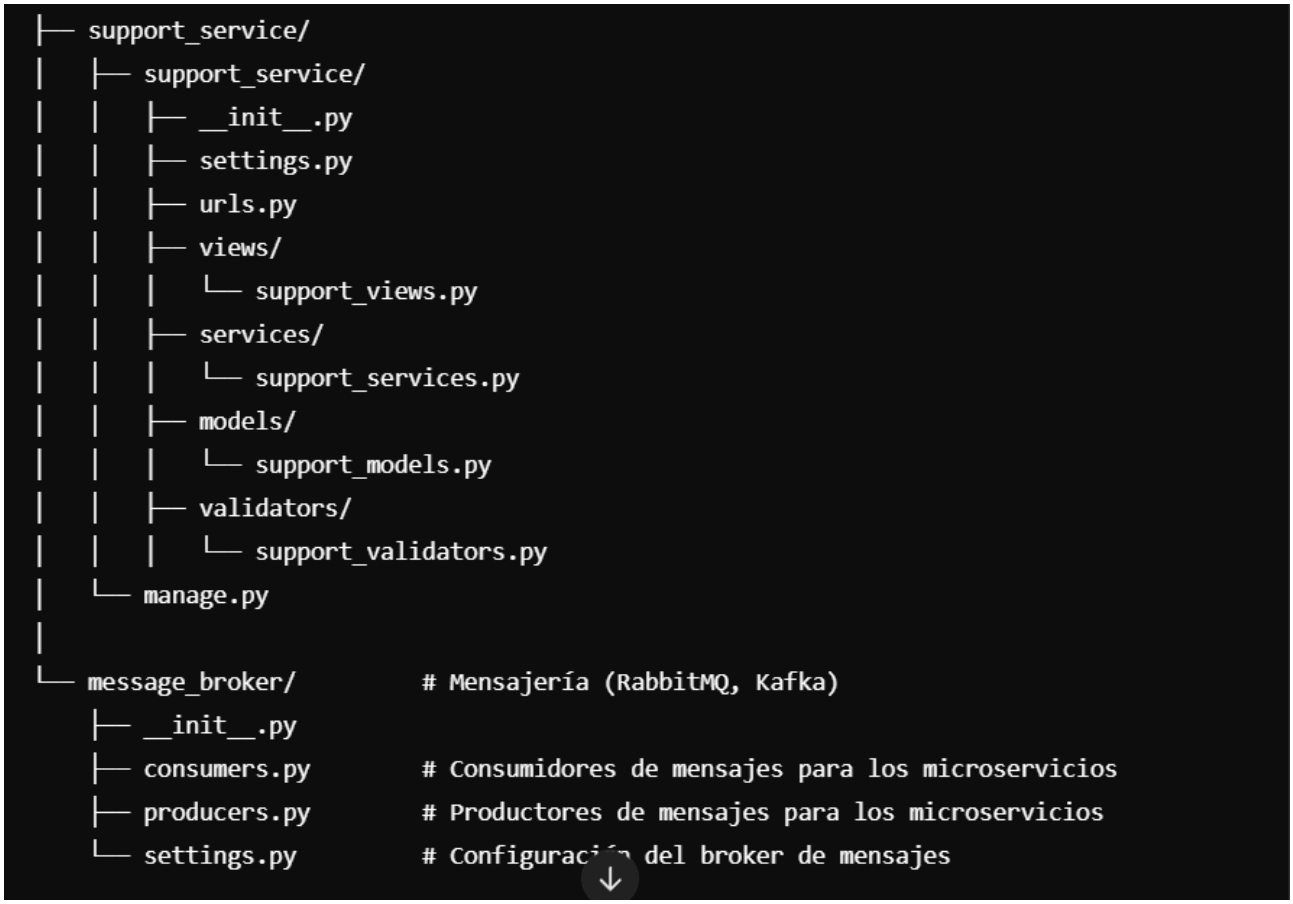
```
project-root/
│
├── api_gateway/
│   ├── __init__.py
│   ├── urls.py             # Rutas del API Gateway que dirigen hacia los microservicios
│   └── settings.py         # Configuración para el API Gateway
│
├── authentication_service/
│   ├── authentication_service/
│   │   ├── __init__.py
│   │   ├── settings.py      # Configuración específica del microservicio de autenticación
│   │   ├── urls.py          # Rutas del microservicio
│   │   ├── views/           # Controladores (View/Controllers)
│   │   │   ├── __init__.py
│   │   │   └── auth_views.py
│   │   ├── services/        # Lógica de negocio
```

```
│   │   ├── services/        # Lógica de negocio
│   │   │   ├── __init__.py
│   │   │   └── auth_services.py
│   │   ├── models/          # Modelos del ORM
│   │   │   ├── __init__.py
│   │   │   └── user_models.py
│   │   ├── validators/      # Validaciones
│   │   │   ├── __init__.py
│   │   │   └── auth_validators.py
│   │   ├── authentication/  # JWT y OAuth
│   │   │   ├── __init__.py
│   │   │   └── jwt_auth.py
│   │   └── session_management/  # Gestión de sesiones
│   │       ├── __init__.py
│   │       └── session_manager.py
│   └── manage.py
│
├── customer_management_service/
│   ├── customer_management_service/
│   │   ├── __init__.py
│   │   ├── settings.py
```

```
├── customer_management_service/
│   ├── customer_management_service/
│   │   ├── __init__.py
│   │   ├── settings.py
│   │   ├── urls.py
│   │   ├── views/
│   │   │   └── customer_views.py
│   │   ├── services/
│   │   │   └── customer_services.py
│   │   ├── models/
│   │   │   └── customer_models.py
│   │   ├── validators/
│   │   │   └── customer_validators.py
│   └── manage.py
│
```

```
├── billing_service/
│   ├── billing_service/
│   │   ├── __init__.py
│   │   ├── settings.py
│   │   ├── urls.py
│   │   ├── views/
│   │   │   └── billing_views.py
│   │   ├── services/
│   │   │   └── billing_services.py
│   │   ├── models/
│   │   │   └── billing_models.py
│   │   ├── validators/
│   │   │   └── billing_validators.py
│   └── manage.py
│
```

```
├── internet_management_service/
│   ├── internet_management_service/
│   │   ├── __init__.py
│   │   ├── settings.py
│   │   ├── urls.py
│   │   ├── views/
│   │   │   └── internet_views.py
│   │   ├── services/
│   │   │   └── internet_services.py
│   │   ├── models/
│   │   │   └── internet_models.py
│   │   ├── validators/
│   │   │   └── internet_validators.py
│   └── manage.py
│
```


├── support_service/
│   ├── support_service/
│   │   ├── __init__.py
│   │   ├── settings.py
│   │   ├── urls.py
│   │   ├── views/
│   │   │   └── support_views.py
│   │   ├── services/
│   │   │   └── support_services.py
│   │   ├── models/
│   │   │   └── support_models.py
│   │   ├── validators/
│   │   │   └── support_validators.py
│   └── manage.py
│
└── message_broker/   # Mensajería (RabbitMQ, Kafka)
    ├── __init__.py
    ├── consumers.py   # Consumidores de mensajes para los microservicios
    ├── producers.py   # Productores de mensajes para los microservicios
    └── settings.py   # Configuración del broker de mensajes

FRONT END

plaintext Copiar código

```
project-root/
└── frontend/
    ├── public/                # Archivos públicos como index.html, imágenes, favicon, e
    │   ├── index.html
    │   ├── favicon.ico
    │   └── assets/            # Carpeta de recursos (imágenes, fuentes, etc.)
    │       └── images/
    ├── src/                   # Carpeta principal del código fuente
    │   ├── api/               # Configuración de servicios API para cada microservicio
    │   │   ├── authApi.js     # API para microservicio de autenticación
    │   │   ├── customerApi.js # API para microservicio de gestión de clientes
    │   │   ├── billingApi.js  # API para microservicio de facturación
    │   │   ├── internetApi.js # API para microservicio de gestión de internet
    │   │   └── supportApi.js  # API para microservicio de soporte
    │   ├── components/        # Componentes reutilizables (botones, formularios, etc.)
    │   │   ├── Button/
    │   │   │   ├── Button.js
    │   │   │   └── Button.css
    │   │   ├── Input/
    │   │   │   ├── Input.js
    │   │   │   └── Input.css
    │   │   └── ...
```


Copiar código
│   │   │   └── Input.css
│   │   └── ...
│   ├── features/   # Funcionalidades específicas de cada módulo
│   │   ├── Auth/   # Módulo de autenticación
│   │   │   ├── components/ # Componentes específicos de autenticación
│   │   │   ├── AuthPage.js
│   │   │   └── authSlice.js # Estado y lógica de autenticación (Redux slice o Context
│   │   ├── Customer/   # Módulo de gestión de clientes
│   │   │   ├── components/
│   │   │   ├── CustomerPage.js
│   │   │   └── customerSlice.js
│   │   ├── Billing/   # Módulo de facturación
│   │   │   ├── components/
│   │   │   ├── BillingPage.js
│   │   │   └── billingSlice.js
│   │   ├── Internet/   # Módulo de gestión de internet
│   │   │   ├── components/
│   │   │   ├── InternetPage.js
│   │   │   └── internetSlice.js
│   │   └── Support/   # Módulo de soporte
│   │   ├── components/
│   │   ├── SupportPage.js
│   │   └── supportSlice.js
│   ├── hooks/   # Hooks personalizados (useAuth, useFetch, etc.)


├── hooks/ # Hooks personalizados (useAuth, useFetch, etc.)
│ ├── useAuth.js
│ ├── useFetch.js
│ └── ...
├── pages/ # Páginas principales de la aplicación
│ ├── HomePage.js
│ ├── LoginPage.js
│ └── NotFoundPage.js
├── routes/ # Rutas y configuración de enrutamiento
│ └── AppRoutes.js
├── store/ # Configuración de Redux o Context para el estado global
│ ├── index.js
│ └── rootReducer.js
├── styles/ # Estilos globales de la aplicación (CSS o SASS)
│ ├── main.css
│ └── variables.css
├── App.js # Componente principal de la aplicación
├── index.js # Punto de entrada de React
└── setupProxy.js # Configuración de proxy para el desarrollo local (opcion
└── package.json