Scaledex

Road - 1, Gulshan - 1, Dhaka

PeerLink

Mar 20, 2025

Overview

This document provides an explanation of the microservices architecture for the PeerLink system. Each microservice is built using Django and operates independently but shares the same database, handling specific functionalities of the platform. The services communicate using message brokers, WebSockets, and API calls..

Goals and Objectives

The primary goal of the PeerLink system is to create a scalable, modular, and efficient microservices-based platform that facilitates seamless user interaction across multiple services, including social networking, live sessions, private chats, AI-driven conversations, file storage, e-commerce, and secure payments. By leveraging Django and Django REST Framework (DRF), the system ensures maintainability, security, and performance.

1. **Modular Architecture:** Each microservice operates independently, enhancing scalability and fault tolerance.
2. **Seamless Integration:** Efficient inter-service communication via APIs, WebSockets, and message queues.
3. **Security & Authentication:** Enforce robust authentication (JWT/OAuth) and access control mechanisms.
4. **Performance Optimization:** Utilize caching (Redis), asynchronous processing (Celery), and efficient database queries.

5. **User-Centric Design:** Provide real-time interactions, AI-driven chat, and streamlined e-commerce workflows.

1. User Control Microservice

Manages user authentication, profile management, and access control.

Key Features

- User Registration and Login (JWT/OAuth authentication)
- User Profile Management (Edit profile, change password, etc.)
- Role-based access control (Admin, User, Moderator)
- Email and phone verification
- User activity tracking

Technology Stack

- Django
- Django REST Framework (DRF)
- PostgreSQL
- Redis (for session management)
- Celery (for background tasks)

Data Flow and Functions

1. **User Registration (`register_user`)**
   - Validates user input and creates a new user.
   - Send verification email/SMS.
2. **User Login (`login_user`)**
   - Verifies credentials and generates JWT token.
3. **Profile Update (`update_profile`)**
   - Updates user details.
4. **Access Control (`check_access`)**

  - Determines user permissions.

API Endpoints

- `POST /api/register/` - Register a new user
- `POST /api/login/` - Authenticate user and return JWT token
- `GET /api/profile/{username}/` - Fetch user profile
- `PUT /api/profile/{username}/` - Update user profile
- `POST /api/logout/` - Log out user

Inter-Service Communication

- Sends user authentication data to **Social Control**, **Live Session**, and **E-Commerce** microservices.
- Verifies user access for **Drive Control** and **Payment** microservices.

2. Social Control Microservice

Handles social interactions such as posts, comments, likes, and followers.

Key Features

- Create, edit, and delete posts
- Commenting and replying to posts
- Like and share functionality
- Follow/unfollow users
- User feed generation

Technology Stack

- Django
- Django REST Framework (DRF)
- PostgreSQL

- Redis (for caching feeds)
- Django Channels (for real-time updates)

Data Flow and Functions

1. **Create Post (`create_post`)**
    - Stores new posts in the database.
    - Notifies followers.
2. **Comment on Post (`add_comment`)**
    - Saves comments and updates posts.
3. **Like Post (`like_post`)**
    - Updates like count and sends notifications.
4. **Follow User (`follow_user`)**
    - Updates follower list.

API Endpoints

- `POST /api/posts/` - Create a new post
- `GET /api/posts/` - Retrieve all posts
- `POST /api/posts/{id}/comment/` - Add a comment to a post
- `POST /api/posts/{id}/like/` - Like a post
- `POST /api/follow/{user_id}/` - Follow a user

Inter-Service Communication

- Fetches user details from **User Control** microservice.
- Notifies **Live Session** and **Private Chat** about social interactions.

3. Live Session Microservice

Facilitates real-time audio and video communication between users.

Key Features

- One-on-one and group audio / video calls
- Live streaming
- Screen sharing
- Session recording

Technology Stack

- Django
- Django REST Framework (DRF)
- WebRTC
- Redis (for session storage)
- Django Channels (for WebSockets)

Data Flow and Functions

1. **Start Session (`start_session`)**
   - Initiates WebRTC connection.
2. **End Session (`end_session`)**
   - Terminates connection and logs details.
3. **Send Stream (`send_stream`)**
   - Handles video/audio data transfer.

API Endpoints:

- `POST /api/session/start/` - Start a new live session
- `POST /api/session/end/` - End a live session
- `GET /api/session/{id}/` - Retrieve session details

Inter-Service Communication

- Validates user authentication via **User Control**.

- Shares session updates with **Private Chat** for messaging support.

4. Private/Group Chat Microservice

Enables private and group messaging features.

Key Features

- One-on-one chat
- Group chat
- Message delivery and read receipts
- Multimedia file sharing
- Notifications

Technology Stack

- Django
- Django REST Framework (DRF)
- PostgreSQL
- Django Channels (for WebSockets)
- Redis (for message queues)

Data Flow and Functions

1. **Send Message (`send_message`)**
   - Stores messages and updates recipients in real-time.
2. **Read Receipt (`update_read_status`)**
   - Updates message status.
3. **Create Group (`create_group`)**
   - Establishes group chat.

API Endpoints:

- `POST /api/chat/send/` - Send a new message

- `GET /api/chat/messages/` - Retrieve chat messages
- `POST /api/chat/group/` - Create a group chat

Inter-Service Communication:

- Verifies users via **User Control** microservice.
- Shared chat logs with **Chat with AI** for NLP-based responses.
- Links conversations to **Live Session** for real-time support.

5. Chat with AI Microservice

Provides AI-driven chat functionality using NLP models.

Key Features:

- AI-powered responses
- Sentiment analysis
- Predefined chatbot interactions
- User behavior learning

Technology Stack

- Django
- Django REST Framework (DRF)
- LangChain
- LangGraph
- Mem0.ai
- Together.ai
- OpenAI API or Rasa for NLP
- Redis (for session storage)

- Celery (for async AI processing)

Data Flow and Functions

1. **Analyze Text (`process_text`)**
    - Runs NLP model.
2. **Generate Response (`generate_response`)**
    - Creates an AI response and sends it to the user.

API Endpoints:

- `POST /api/ai/analyze/` - Analyze text and generate AI response

Inter-Service Communication

- Receives chat inputs from **Private Chat** microservice.
- Logs AI interactions in **User Control** for personalized experiences.

6. Drive Control Microservice

Manages cloud storage and file sharing features.

Key Features

- File uploads and downloads
- File version control
- Access control (private, public, restricted)
- Storage analytics

Technology Stack

- Django
- Django REST Framework (DRF)
- PostgreSQL
- AWS S3 or MinIO (for storage)

- Redis (for caching)

Data Flow and Functions

1. **Upload File (`upload_file`)**
   - Stores file and metadata.
2. **Download File (`download_file`)**
   - Retrieves file based on access control.

API Endpoints

- `POST /api/files/upload/` - Upload a new file
- `GET /api/files/{id}/` - Download a file

Inter-Service Communication

- Verifies user permissions via **User Control**.
- Links files to **E-Commerce** for digital product sales.

7. E-Commerce Microservice

Handles product listings, orders, and cart functionality.

Key Features

- Product management (add, update, delete)
- Shopping cart and checkout
- Order processing
- User reviews and ratings

Technology Stack

- Django

- Django REST Framework (DRF)
- PostgreSQL
- AWS S3 or MinIO (for storage)
- Redis (for caching)

Data Flow and Functions

1. **Add to Cart (`add_to_cart`)**
   - Adds items to the user's cart.
2. **Checkout (`checkout`)**
   - Process order.

API Endpoints

- `POST /api/products/` - Add a new product
- `GET /api/products/` - Retrieve all products
- `POST /api/cart/add/` - Add item to cart
- `POST /api/order/checkout/` - Process checkout

Inter-Service Communication:

- Connects with **Payment Microservice** for transactions.
- Fetches digital products from **Drive Control** for download links.

8. Payment Microservice

Handles financial transactions and payment processing.

Key Features

- Secure payment gateway integration (Stripe, PayPal, etc.)
- Subscription management

- Refund processing

Technology Stack

- Django
- Django REST Framework (DRF)
- PostgreSQL
- AWS S3 or MinIO (for storage)
- Redis (for caching)

Data Flow and Functions

1. **Process Payment (`process_payment`)**
   - Validates payment details and completes transactions.
2. **Issue Refund (`issue_refund`)**
   - Handles refund processing.

API Endpoints:

- `POST /api/payment/process/` - Process a payment
- `POST /api/payment/refund/` - Issue a refund
- `GET /api/payment/status/` - Retrieve transaction status

Inter-Service Communication

- Receives payment requests from **E-Commerce**.
- Sends transaction confirmation to **User Control** for records.

Conclusion

The PeerLink system leverages Django, Django REST Framework (DRF), and other modern technologies to create an efficient, modular, and scalable microservices architecture. Each service is designed to operate independently while seamlessly integrating with other components. By implementing robust authentication, efficient data flow, and optimized inter-service communication, the system ensures a smooth and secure user experience. This documentation serves as a blueprint for development and future enhancements

Author

MD. Golam Mostofa
AI and Full Stack Dev
Date: Mar 20, 2025

Co-Author