News Feed

▼ 1. Introduction

Welcome to the documentation for the News Feed Personalization and Subscription System project.

▼ Overview of the platform

The Personalized RAG News Feed Subscription Platform is an innovative application designed to revolutionize the way users consume news by leveraging advanced artificial intelligence technologies. This platform integrates LangChain, Qdrant, and OpenAI to deliver highly personalized news feeds tailored to individual interests and preferences, ensuring that users receive content that is relevant, engaging, and up-to-date.

▼ Purpose and target audience

> The primary purpose of the News Feed Personalization and Subscription System is to enhance the user experience in consuming news by delivering highly personalized and relevant content. This is achieved through the integration of advanced artificial intelligence technologies, such as Qdrant and OpenAI, which allow the system to understand user interests and curate news feeds tailored to those interests. The system aims to:

1. **Deliver Personalized Content:** Provide users with news articles that match their specific interests and preferences, ensuring that the content they receive is engaging and pertinent to their needs.
2. **Automate News Delivery:** Allow users to subscribe to personalized news updates that are delivered via email at their chosen frequency, making it easy for them to stay informed without actively searching for news.
3. **Leverage AI for Improved Accuracy:** Utilize AI technologies to enhance the accuracy and relevance of the news articles included in the personalized feed, ensuring timely and contextually relevant content.
4. **Streamline Content Curation:** Automate the process of fetching, analyzing, and curating news articles from multiple sources, reducing the effort required to gather comprehensive and diverse news coverage.
5. **Enhance User Engagement:** Increase user engagement by providing a seamless and convenient way to access news that matters to them, fostering a more satisfying and immersive news consumption experience.

> The target audience for the News Feed Personalization and Subscription System includes:

1. **General News Consumers:**
    - Individuals who want to stay updated with the latest news but prefer a tailored approach that filters out irrelevant information and focuses on their specific interests.
2. **Busy Professionals:**

- Professionals who have limited time to browse through multiple news sources and would benefit from a service that curates and delivers relevant news articles directly to their inbox.

3. **Tech-Savvy Users:**
    - Users who are comfortable with using AI-driven applications and appreciate the value of personalized content delivered through advanced technological solutions.
4. **Researchers and Analysts:**
    - Individuals in fields that require constant monitoring of specific topics, trends, or events, who would benefit from a tool that automates and personalizes news curation.
5. **Newsletter Subscribers:**
    - Users who already subscribe to newsletters and are looking for a more personalized and relevant news delivery service.
6. **Media and Content Enthusiasts:**
    - People with a keen interest in consuming a wide range of news articles from various perspectives and sources, ensuring a well-rounded understanding of current events.

▼ Key features

- **Personalized News Feeds**: Users can enter a prompt describing their interests, and the platform uses ~~LangChain~~ and Qdrant to curate a personalized news feed based on these interests. This ensures that users receive content that is relevant and engaging to them.
- **Subscription Management**: Users can subscribe to receive personalized news updates via email at their chosen frequency. This feature allows users to stay informed about the topics they care about without having to actively search for news.
- **Integration with ~~LangChain~~ and Qdrant**: At the core of the platform is the integration of ~~LangChain~~ and Qdrant, which enhance the accuracy and reliability of the news feed by allowing the AI system to fetch and incorporate up-to-date information from various sources. This ensures that the news presented to the user is timely, accurate, and contextually relevant.
- **OpenAI Integration**: The platform utilizes OpenAI's capabilities to further refine the content curation process, ensuring that the content is not only personalized but also curated with care to provide the best possible reading experience.
- **Multiple News Sources**: The platform integrates with multiple news sources to fetch articles regularly. This wide range of sources ensures that users receive a diverse selection of news articles that cover various perspectives and viewpoints.
- **AI-Driven Content Curation**: The combination of ~~LangChain~~, Qdrant, and OpenAI processes user prompts and selects the most relevant news articles for inclusion in the personalized newsletter. This AI-driven curation process ensures that the content is not only personalized but also curated with care to provide the best possible reading experience.
- **Email Notifications**: The platform implements a system to send personalized newsletters to users via email based on their selected frequency. This feature ensures that users receive their news updates in a convenient and timely manner.

▼ 2. Getting Started

▼ Installation prerequisites

> ⚠ TODO: Complete this installation part

To set up and run the News Feed Personalization and Subscription System, you will need to ensure that you have the following prerequisites installed and configured on your system:

1. Operating System

- **Linux**, **macOS**, or **Windows** (with WSL for Linux compatibility).

2. Docker and Docker Compose

- **Docker:** Containerization platform to run and manage the application's services.
  - Installation instructions: Docker Installation Guide
- **Docker Compose:** Tool for defining and running multi-container Docker applications.
  - Installation instructions: Docker Compose Installation Guide

4. Node.js and npm

- **Node.js:** JavaScript runtime environment to run the frontend development server and other Node.js-based tools.
  - Installation instructions: Node.js Installation Guide
- **npm:** Node package manager, included with Node.js, to manage project dependencies.

5.RabbitMQ

- **RabbitMQ:** Message broker to facilitate communication between microservices. The setup for RabbitMQ will be handled within the Docker environment.

6. OpenAI API Key

- **OpenAI API Key:** Required for integrating with OpenAI's capabilities. You will need to sign up for an API key from OpenAI and include it in your environment configuration.

▼ Setting up the development environment

▼ Running the platform locally

▼ 3. Architecture

▼ High-Level architecture overview

The combination of an ~~API Gateway~~ and a reverse proxy offers several advantages:

- ~~**Load Balancing and High Availability**: Nginx can distribute incoming requests across multiple instances of your API Gateway, improving responsiveness and availability.~~
- **SSL Termination**: Nginx can handle SSL/TLS encryption and decryption, offloading this task from your API Gateway and reducing its computational load.
- **Static Content Serving**: Nginx can efficiently serve static assets, reducing the load on your API Gateway and improving response times for clients.
- **Centralized Authentication and Authorization**: By placing Nginx in front of your API Gateway, you can centralize authentication and authorization checks, reducing the complexity of your API Gateway and improving security.

▼ Microservice architecture

To organize this system effectively, we can apply Domain-Driven Design (DDD) principles to identify the core domains and subdomains, which will guide us in defining the micro-services architecture.

Core Domains and Subdomains

1. **newsfeed-service**

- **Purpose:** Store, maintain, and update news content.
- **Technologies:** MongoDB, Qdrant

2. **user-service**
    - **Purpose:** Handle user authentication and store user information.
    - **Technologies:** Redis (for user auth whitelist), MongoDB
3. **subscription-service**
    - **Purpose:** Manage subscriptions and subscribers, schedule cron jobs for periodic updates.
    - **Technologies:** MongoDB
4. **notification-service**
    - **Purpose:** Send personalized email updates to subscribers.
    - **Technologies:** NodeMailer, RabbitMQ
5. **content-service**
    - **Purpose:** Communicate with the OpenAI API to process user prompts and find relevant news articles.
    - **Technologies:** OpenAI API, Qdrant FastEmbed

5. Search and Recommendation

- **Subdomain**: Search, Personalized Recommendations, RAG Model Implementation
- **Microservices**: Search Service, Recommendation Service

6. Data Storage and Indexing

- **Subdomain**: Data Storage, Indexing, Full-Text Search
- **Microservices**: Data Storage Service, Indexing Service

▼ Technology stack

Frontend

- **React:** A JavaScript library for building user interfaces, used for developing the responsive and interactive frontend of the application.
- **TypeScript:** A statically typed superset of JavaScript, providing type safety and enhanced developer experience.
- **MUI (Material-UI):** A React component library that implements Google's Material Design, used for building a consistent and aesthetically pleasing user interface.
- **Tailwind CSS:** A utility-first CSS framework for rapidly building custom designs, providing flexibility in styling components.
- **Redux:** A state management library for managing and centralizing application state.

Backend

- **Node.js:** A JavaScript runtime environment used for building scalable network applications, serving as the foundation for the backend services.
- **Express.js:** A minimal and flexible Node.js web application framework used to build the APIs.

Databases and Data Stores

- **MongoDB:** Primary permanent database for storing user information, news articles, and subscription details.
- **Qdrant:** Vector database for storing and querying vectorized news content to find relevant articles for subscribers.

- **Redis:** In-memory data structure store used for user authentication token whitelisting.

Message Broker

- **RabbitMQ:** A message broker used to facilitate communication between microservices, ensuring reliable message delivery.

Reverse Proxy and Load Balancer

- **Nginx:** Used as a reverse proxy to handle HTTP requests and direct them to the appropriate microservice, also providing load balancing capabilities.

Containerization and Orchestration

- **Docker:** Containerization platform to run and manage the application's services in isolated environments.
- **Docker Compose:** Tool for defining and running multi-container Docker applications, used to manage the entire stack including backend services, databases, message broker, and the frontend.

AI and Machine Learning

- **OpenAI API:** Utilized for processing user prompts and enhancing the personalization of news articles.
- **Qdrant FastEmbed:** Service for embedding news content, aiding in the efficient search and relevance matching of articles.

Email Service

- **NodeMailer:** A module for Node.js applications to send emails, used in the `notification-service` to send personalized newsletters.

Development and Documentation

- **Git:** Version control system for managing the project's source code.
- **npm:** Node package manager for managing project dependencies and scripts.
- **JSdoc:** A documentation generator for JavaScript, used to produce API documentation from inline comments.

4. Development Guide

Development environment setup

Contributing to the project

Code style and conventions

5. API Reference

Overview of the API endpoints

POST /auth/subscribe

Description:

This endpoint allows a new user to subscribe by creating an account.

Request Body:

```
{
  "email": "user@example.com",
  "password": "yourpassword",
  "frequency": "daily",
  "prompt": "Your interests prompt",
  "fingerPrint": "unique_device_fingerprint"
}
```

Validations:

- `email` must be a valid email address and unique.
- `password` must be provided.
- `frequency` must be one of `daily`, `weekly`, or `monthly`.
- `prompt` must be between 10 and 255 characters and not empty.

Responses:

- `201 Created` - Subscription successful.
- `400 Bad Request` - Validation error or missing fields.
- ~~`409 Conflict` - Email already exists.~~

Middlewares:

- `signupValidation` - Validates the request body.

Controller:

- `signup` - Handles the subscription process.

▼ POST /auth/login

Description:
This endpoint allows a user to log in.

Request Body:

```
{
  "email": "user@example.com",
  "password": "yourpassword",
  "fingerPrint": "unique_device_fingerprint"
}
```

Validations:

- `email` must be a valid email address.
- `password` must be provided.

Responses:

- `200 OK` - Login successful.
- `400 Bad Request` - Validation error or missing fields.
- `404 Not Found` - User not found.
- `401 Unauthorized` - Invalid email or password.

Middlewares:

- `signinValidation` - Validates the request body.
- `isUserExist` - Checks if the user exists.

Controller:

- `login` - Handles the login process.

▼ Post /auth/logout

Description:
This endpoint allows a user to log out.

```
{
  "fingerPrint": "unique_device_fingerprint"
}
```

Responses:

- `200 OK` - Logout successful.
- `401 Unauthorized` - User is not authenticated.

Middlewares:

- `authenticate` - Checks if the user is authenticated.

Controller:

- `logout` - Handles the logout process.

▼ PUT /auth/change

Description:
This endpoint allows the authenticated user to change their subscription details.

Request Body:

```
{
  "frequency": "weekly",
  "prompt": "Updated interests prompt",
  "fingerPrint": "unique_device_fingerprint"
}
```

Validations:

- `frequency` must be one of `daily`, `weekly`, or `monthly`.
- `prompt` must be between 10 and 255 characters and not empty.
- `fingerPrint` must be provided and unique for the user's device.

Responses:

- `200 OK` - Subscription details updated successfully.
- `400 Bad Request` - Validation error or missing fields.
- `401 Unauthorized` - User is not authenticated.

Middlewares:

- `authenticate` - Checks if the user is authenticated.
- `changeSubscriptionValidation` - Validates the request body.
- `isUserExist` - Checks if the user exists.

Controller:

- `changeSubscription` - Handles the subscription change process.

▼ POST /auth/whoisme

Description:

This endpoint returns the authenticated user's information.

Request Body:

```
{
  "fingerPrint": "unique_device_fingerprint"
}
```

Responses:

- `200 OK` - Returns user information.
- `401 Unauthorized` - User is not authenticated.
- `429 Too Many Requests` - Rate limiting.

Middlewares:

- `authenticate` - Checks if the user is authenticated.
- `isUserExist` - Checks if the user exists.

Controller:

- `whoIsMe` - Returns the user's information.

GET /api/news

⚠ TODO: Change GET to POST method!

Description:
This endpoint returns a list of news articles related to the authenticated user's interests.

Query Parameters (Optional):

- `page` (Number) - The page number for pagination.

Request Body:

```
{
  "email": "user@example.com",
  "fingerPrint": "unique_device_fingerprint"
}
```

Responses:

- `200 OK` - Returns a list of related news articles.
  - Sample Response:

```
{
  "message": "Success",
  "error": [],
  "data": {
    "news": [
      {
        "title": "News Title",
        "description": "Brief description of the news article",
        "content": "Full content of the news article",
        "category": "Category",
        "author": "Author",
        "keywords": "Keywords associated with the article",
        "country": "Country of origin",
        "language": "Language of the article",
        "source": "Source of the article",
        "url": "URL of the article",
        "publishedAt": "Publication date",
        "createdAt": "Creation date"
      }
      // More articles...
    ],
    "lastNewsSummerized": "Summary of the last batch of news articles"
  }
}
```

- `404 Not Found` - Subscriber with the given email not found.
    - Sample Response:

    ```
    {
      "message": "Subscriber not found",
      "error": ["Email not found"],
      "data": null
    }
    ```

- `500 Internal Server Error` - Internal server error.
    - Sample Response:

    ```
    {
      "message": "Internal Server Error",
      "error": ["Failed to read news"],
      "data": null
    }
    ```

Middlewares:

- `authenticate` - Checks if the user is authenticated.

▼ GET /api/news/count

⚠ TODO: Change GET to POST method!

Description:
This endpoint returns the count of news articles related to the authenticated user's interests.

Request Body:

```
{
  "email": "user@example.com",
  "fingerPrint": "unique_device_fingerprint"
}
```

Responses:

- `200 OK` - Returns the count of related news articles.
    - Sample Response:

    ```
    {
      "message": "Success",
      "error": [],
      "data": {
        "count": 50
      }
    }
    ```

- `404 Not Found` - Subscriber with the given email not found.
- `500 Internal Server Error` - Internal server error.

Middlewares:

- `authenticate` - Checks if the user is authenticated.

▼ Authentication and authorization

This platform uses JWT (JSON Web Token) for authentication and Redis for maintaining a whitelist of valid tokens. This ensures secure access to the API endpoints and allows for efficient management of user sessions.

Authentication Process:

1. **User Login:**
    - The user provides their email and password.
    - If the credentials are valid, the server generates a JWT.
    - The JWT is added to the Redis whitelist with an expiration time.
    - The token is returned to the client for use in subsequent requests.
2. **Token Validation:**
    - For each request to a protected endpoint, the JWT provided by the client is validated.
    - The server checks if the token is in the Redis whitelist.
    - If the token is valid and in the whitelist, the request is authorized.
    - If the token is invalid or expired, the request is denied.

Authorization Process:

- User-specific actions are protected by middleware that verifies the user's identity and permissions using the JWT and Redis whitelist.

JWT Structure:

- **Header:** Specifies the algorithm used for signing the token.
- **Payload:** Contains the user's information and any additional claims.
- **Signature:** Ensures the token's integrity and authenticity.

Endpoints Requiring Authentication:

- All endpoints listed under the `/auth` and `/api/news` routes require the user to be authenticated.

▼ API_KEYS

service	API_KEY
openai	sk-proj-c1g3JENJXrQBSTC7yLPWT3BlbkFJG8HqlcPQRW9sB6FgussR
newsdata.io	pub_433503da424996c1637d79153613a966ae842
newsapi.org	461c65e7852b4d3ebc1b43ea688858a5
gnews	2d5260b305beb6d93a75cf0be78c95f9

▼ 6. Data Models

▼ News

Field	Type	Description	Constraints
title	String	Title of the news article	Required
description	String	Description of the news article	Required
content	String	Full content of the news article	Required
category	String	Category of the news article	Optional, default: "General"
author	String	Author of the news article	Optional, default: "Anonymous"

keywords	String	Keywords associated with the article	Optional, default: ""
country	String	Country of origin	Optional, default: "Unknown"
language	String	Language of the article	Optional, default: "English"
source	String	Source of the article	Optional, default: "Unknown"
url	String	URL of the article	Optional, default: ""
publishedAt	Date	Date the article was published	Optional, default: Date.now()
createdAt	Date	Timestamp of article creation	Optional, default: Date.now(), expires based on configuration

▼ User

Field	Type	Description	Constraints
email	String	User's email address	Required, unique, indexed, valid email, 10-50 characters
password	String	User's password	Required, bcrypt hashed (60 characters), non-empty
frequency	String	Frequency of subscription	Required, enum (daily, weekly, monthly)
prompt	String	Prompt for subscription	Required, 10-255 characters, non-empty
createdAt	Date	Timestamp of user creation	Automatically generated
updatedAt	Date	Timestamp of last update	Automatically generated

▼ Subscriber

Field	Type	Description	Constraints
email	String	Subscriber's email address	Required
frequency	String	Frequency of subscription	Required
prompt	String	User's original prompt	Required
parsedPrompt	String	Parsed version of the user's prompt	Required
embeddingParsedPrompt	String	Embedded version of the parsed prompt	Required
lastNewsSummerized	String	Summarized version of the last sent news	Optional, default: ''
lastRelatedNewsIDs	Array<String>	Array of IDs of the last related news articles	Optional, default: []
updatedAt	Date	Timestamp of the last update	Optional, default: Date.now
createdAt	Date	Timestamp of subscriber creation	Optional, default: Date.now

▼ Keywords

Field	Type	Description	Constraints
title	String	The keyword title	Required, unique, indexed
createdAt	Date	Timestamp of creation	Required, default: Date.now

▼ 7. Functional Requirements

▼ User registration and login

▼ Subscription management

▼ Content curation and aggregation

▼ Personalized news feed generation

▼ Email notifications

▼ 8. Non-Functional Requirements

▼ Performance and scalability

▼ QDrant

Improving the performance of Qdrant can be approached from several angles:

1. **Optimize your data**: Make sure your data is clean and well-structured. Remove any unnecessary data points. The less data Qdrant has to process, the faster it can perform operations.
2. **Use efficient vector representations**: The efficiency of Qdrant heavily depends on the vector representations you use. Try to use compact and meaningful vector representations. High-dimensional vectors can slow down the search operation.
3. **Tune Qdrant parameters**: Qdrant has several parameters that you can tune to optimize its performance for your specific use case. For example, you can adjust the number of trees in the HNSW index, or the maximum number of points in a leaf node of the tree.
4. **Scale horizontally**: If you have a lot of data, you might want to consider scaling Qdrant horizontally by running multiple instances of Qdrant and distributing your data across them. This can help to distribute the load and make your application more resilient.
5. **Use efficient queries**: Make sure your queries are as efficient as possible. Avoid unnecessary complex queries. The simpler the query, the faster Qdrant can process it.
6. **Hardware considerations**: Qdrant's performance can also be influenced by the hardware it's running on. Faster CPUs, more memory, and SSDs can all contribute to better performance.

```
import axios from 'axios';
import * as Bluebird from 'bluebird';

class QDrantController {
  constructor(private _url: string) {}

  async addDocument(collection: string, documentPayload: any) {
    const response = await axios.post(`${this._url}/collections/${collection}/points`
    return response.data;
  }

  async findRelevantDocuments(collection: string, vector: number[], top: number) {
    const response = await axios.post(`${this._url}/collections/${collection}/points/
      vector,
      top,
    });
    return response.data;
  }

  async dropCollection(collection: string) {
    const response = await axios.delete(`${this._url}/collections/${collection}`);
    return response.data;
  }

  async addDocumentsInBatch(collection: string, documents: any[], batchSize: number)
    const batches = Array(Math.ceil(documents.length / batchSize)).fill([]).map((_, i
    await Bluebird.map(batches, batch => this.addDocument(collection, batch), { concu
  }
}
```

▼ Security considerations

▼ Error handling and logging

▼ 9. Deployment

▼ Deployment process

▼ Environment setup

▼ General .env file setup

▼ Message Broker

Environment Variable	Service	Explanation	Example
BROKER_PROTOCOL	All Service	Message broker connection protocol	amqp
BROKER_HOST	All Service	Message broker hostname or IP address	rabbitmq
BROKER_PORT	All Service	Message broker port number	5672
BROKER_DEFAULT_VHOST	All Service	Default virtual host for the message broker	notificationVhost
BROKER_DEFAULT_USERNAME	All Service	Default username for the message broker	guest
BROKER_DEFAULT_PASSWORD	All Service	Default password for the message broker	guest

▼ Mongo DB

Environment Variable	Explanation	Example
MONGO_PROTOCOL	MongoDB connection protocol	mongodb
MONGO_HOST	MongoDB hostname or IP address	mongo
MONGO_PORT	MongoDB port number	27017
MONGO_USERNAME	MongoDB username (if applicable)	
MONGO_PASSWORD	MongoDB password (if applicable)	
MONGO_NAME	MongoDB database name	newsfeed

⛔ MONGO_NAME can be override for each container in docker-compose file, to separate each service DB.

▼ Redis

Environment Variable	Explanation	Example
REDIS_PROTOCOL	Redis connection protocol	redis
REDIS_HOST	Redis hostname or IP address	redis
REDIS_PORT	Redis port number	6379
REDIS_USERNAME	Redis username (if applicable)	guest
REDIS_PASSWORD	Redis password (if applicable)	guest
REDIS_DB	Redis database index	0

⛔ REDIS_DB can be override for each container in docker-compose file, to separate each service caching DB.

▼ Whitelist

Environment Variable	Explanation	Example
WHITELIST_PROTOCOL	Whitelist connection protocol	redis
WHITELIST_HOST	Whitelist hostname or IP address	redis
WHITELIST_PORT	Whitelist port number	6379
WHITELIST_USERNAME	Whitelist username (if applicable)	guest
WHITELIST_PASSWORD	Whitelist password (if applicable)	guest
WHITELIST_DB	Whitelist database index	1

⛔ Whitelist use redis as it core; so, notice that WHITELIST_DB not combined with another Redis DB

▼ Authentication

Environment Variable	Explanation	Example
ACCESS_TOKEN_SECRET	Secret key for signing JWT access tokens	"94ff6d0f434222dfbc53f8fc85d6aaa1e3e653f827660db41a3433ce
ACCESS_TOKEN_EXPIRES_IN	Expiration time for access tokens	"1h"
ACCESS_TOKEN_ALGORITHM	Algorithm used for signing access tokens	HS256
REFRESH_TOKEN_SECRET	Secret key for signing JWT refresh tokens	"1cc68b7e835fdb9188f2ec039ac38fa82d5cc6f948d31ecb1287f9f6e
REFRESH_TOKEN_EXPIRES_IN	Expiration time for refresh tokens	"7d"
REFRESH_TOKEN_ALGORITHM	Algorithm used for signing refresh tokens	HS256

▼ Logging

Environment Variable	Explanation	Example
LOG_LEVEL	Logging verbosity level	debug
LOG_DIR	Directory where log files are stored	./logs

Example:

```
# General Configuration
NODE_ENV=development
LOG_LEVEL=debug
LOG_DIR=./logs

# Authentication Tokens
ACCESS_TOKEN_SECRET="94ff6d0f434222dfbc53f8fc85d6aaa1e3e653f827660db41a3433ce7fe69795
ACCESS_TOKEN_EXPIRES_IN="1h"
ACCESS_TOKEN_ALGORITHM="HS256"

REFRESH_TOKEN_SECRET="1cc68b7e835fdb9188f2ec039ac38fa82d5cc6f948d31ecb1287f9f6e22cd9c
REFRESH_TOKEN_EXPIRES_IN="7d"
REFRESH_TOKEN_ALGORITHM="HS256"

# MongoDB Configuration
MONGO_PROTOCOL=mongodb
MONGO_HOST=mongo
MONGO_PORT=27017
MONGO_USERNAME=
MONGO_PASSWORD=

# Redis Configuration
REDIS_PROTOCOL=redis
REDIS_HOST=redis
REDIS_PORT=6379

# Broker Configuration
BROKER_PROTOCOL=amqp
BROKER_HOST=rabbitmq
BROKER_PORT=5672
BROKER_DEFAULT_VHOST="notificationVhost"
BROKER_DEFAULT_USERNAME="guest"
BROKER_DEFAULT_PASSWORD="guest"

# CORS Configuration
CORS_ORIGIN=http://localhost:3000

# Service-Specific Configuration
# User Service
APP_PORT_USER=8000
MONGO_NAME_USER=newsfeed
REDIS_DB_USER=1

# Subscription Service
APP_PORT_SUBSCRIPTION=8004
MONGO_NAME_SUBSCRIPTION=userdb
REDIS_USERNAME_SUBSCRIPTION=guest
REDIS_PASSWORD_SUBSCRIPTION=guest
REDIS_DB_SUBSCRIPTION=2
WHITELIST_PROTOCOL_SUBSCRIPTION=redis
```

```
WHITELIST_HOST_SUBSCRIPTION=redis
WHITELIST_PORT_SUBSCRIPTION=6379
WHITELIST_USERNAME_SUBSCRIPTION=guest
WHITELIST_PASSWORD_SUBSCRIPTION=guest
WHITELIST_DB_SUBSCRIPTION=1
PARSE_PROMPT_SUBSCRIPTION=0

# Notification Service
APP_PORT_NOTIFICATION=8003
EMAIL_SERVICE_NOTIFICATION=gmail
EMAIL_USERNAME_NOTIFICATION=""
EMAIL_PASSWORD_NOTIFICATION=""

# Newsfeed Service
APP_PORT_NEWSFEED=8005
MONGO_DB_NEWSFEED=newsFeedService
MONGO_USERNAME_NEWSFEED=
MONGO_PASSWORD_NEWSFEED=
NEWS_EXPIRE_NEWSFEED=2592000
NEWSDATA_API_KEY_NEWSFEED="pub_433503da424996c1637d79153613a966ae842"
NEWSDATA_PATH_NEWSFEED="https://newsdata.io/api/1"
NEWSDATA_ENDPOINT_NEWSFEED="news"
NEWSDATA_MAX_NEWSFEED=50
NEWSDATA_ACTIVE_NEWSFEED=1
NEWSAPI_API_KEY_NEWSFEED="461c65e7852b4d3ebc1b43ea688858a5"
NEWSAPI_PATH_NEWSFEED="https://newsapi.org/v2"
NEWSAPI_ENDPOINT_NEWSFEED="top-headlines"
NEWSAPI_MAX_NEWSFEED=50
NEWSAPI_ACTIVE_NEWSFEED=1
GNEWS_API_KEY_NEWSFEED="2d5260b305beb6d93a75cf0be78c95f9"
GNEWS_PATH_NEWSFEED="https://gnews.io/api/v4"
GNEWS_ENDPOINT_NEWSFEED="top-headlines"
GNEWS_MAX_NEWSFEED=50
GNEWS_ACTIVE_NEWSFEED=1
QDRANT_PROTOCOL_NEWSFEED=http
QDRANT_HOST_NEWSFEED=qdrant
QDRANT_PORT_NEWSFEED=6333
RESET_NEWS_NEWSFEED=1

# Content Service
APP_PORT_CONTENT=8002

# OpenAI Configuration
OPENAI_API_KEY="sk-proj-c1g3JENJXrQBSTC7yLPWT3BlbkFJG8HqIcPQRW9sB6FgussR"
```

▼ Docker-compose

Services

1. **application**
    - **container_name**: The name of the container is `application`.
    - **image**: Uses the `newsfeed-application` image.
    - **build**: Builds from the `./frontend/newsfeed` directory.
    - **ports**: Maps port 3000 on the host to port 3000 in the container.
    - **depends_on**: Depends on `userService` and `subscriptionService` to be running.
    - **networks**: Connects to `app_network`.

2. **userService**
    - **container_name**: The name of the container is `userService`.
    - **image**: Uses the `user-service` image.
    - **build**: Builds from the `./backend/userService` directory.
    - **ports**: Maps port 8000 on the host to port 8000 in the container.
    - **depends_on**: Depends on `mongo`, `redis`, and `rabbitmq` to be running.
    - **volumes**: Mounts `./modules` and `./command` directories into the container.
    - **env_file**: Loads environment variables from `.env`.
    - **environment**: Overrides some environment variables.
    - **networks**: Connects to `app_network`.
    - **command**: Executes a shell command to wait for `rabbitmq` and then runs `node dist/server.js`.
3. **newsfeedService**
    - **container_name**: The name of the container is `newsfeed`.
    - **image**: Uses the `newsfeed-service` image.
    - **build**: Builds from the `./backend/newsFeedService` directory.
    - **ports**: Maps port 8005 on the host to port 8005 in the container.
    - **depends_on**: Depends on `rabbitmq`, `qdrant`, and `mongo` to be running.
    - **volumes**: Mounts `./modules` and `./command` directories into the container.
    - **env_file**: Loads environment variables from `.env`.
    - **environment**: Overrides some environment variables.
    - **networks**: Connects to `app_network`.
    - **command**: Executes a shell command to wait for `rabbitmq` and then runs `node dist/server.js`.
4. **contentService**
    - **container_name**: The name of the container is `content`.
    - **image**: Uses the `content-service` image.
    - **build**: Builds from the `./backend/contentService` directory.
    - **ports**: Maps port 8002 on the host to port 8002 in the container.
    - **depends_on**: Depends on `rabbitmq` to be running.
    - **volumes**: Mounts `./modules` and `./command` directories into the container.
    - **env_file**: Loads environment variables from `.env`.
    - **environment**: Overrides some environment variables.
    - **networks**: Connects to `app_network`.
    - **command**: Executes a shell command to wait for `rabbitmq` and then runs `node dist/server.js`.
5. **notificationService**
    - **container_name**: The name of the container is `notification`.
    - **image**: Uses the `notification-service` image.
    - **build**: Builds from the `./backend/notificationService` directory.
    - **ports**: Maps port 8003 on the host to port 8003 in the container.
    - **depends_on**: Depends on `rabbitmq` to be running.
    - **volumes**: Mounts `./modules` and `./command` directories into the container.
    - **env_file**: Loads environment variables from `.env`.

- **environment**: Overrides some environment variables.
- **networks**: Connects to `app_network`.
- **command**: Executes a shell command to wait for `rabbitmq` and then runs `node dist/server.js`.

6. **subscriptionService**
   - **container_name**: The name of the container is `subscriptionService`.
   - **image**: Uses the `subscription-service` image.
   - **build**: Builds from the `./backend/subscriptionService` directory.
   - **ports**: Maps port 8004 on the host to port 8004 in the container.
   - **depends_on**: Depends on `rabbitmq`, `redis`, and `mongo` to be running.
   - **volumes**: Mounts `./modules` and `./command` directories into the container.
   - **env_file**: Loads environment variables from `.env`.
   - **environment**: Overrides some environment variables.
   - **networks**: Connects to `app_network`.
   - **command**: Executes a shell command to wait for `rabbitmq` and then runs `node dist/server.js`.
7. **nginx**
   - **container_name**: The name of the container is `nginx`.
   - **image**: Uses the `nginx:latest` image.
   - **ports**: Maps port 80 on the host to port 80 in the container.
   - **volumes**: Mounts `./nginx.conf` into the container at `/etc/nginx/nginx.conf`.
   - **networks**: Connects to `app_network`.
8. **mongo**
   - **container_name**: The name of the container is `mongo`.
   - **image**: Uses the `mongo:latest` image.
   - **ports**: Maps port 27017 on the host to port 27017 in the container.
   - **networks**: Connects to `app_network`.
9. **rabbitmq**
   - **container_name**: The name of the container is `rabbitmq`.
   - **image**: Uses the `rabbitmq:management` image.
   - **ports**: Maps port 15672 on the host to port 15672 in the container, and port 5672 on the host to port 5672 in the container.
   - **networks**: Connects to `app_network`.
10. **redis**
    - **container_name**: The name of the container is `redis`.
    - **image**: Uses the `redis:latest` image.
    - **ports**: Maps port 6379 on the host to port 6379 in the container.
    - **networks**: Connects to `app_network`.
11. **qdrant**
    - **container_name**: The name of the container is `qdrant`.
    - **image**: Uses the `qdrant/qdrant` image.
    - **ports**: Maps port 6333 on the host to port 6333 in the container.
    - **volumes**: Mounts `./qdrant_data` into the container at `/qdrant/storage`.
    - **networks**: Connects to `app_network`.

Volumes

- Defines persistent storage volumes:
  - `qdrant_data` for Qdrant storage.
  - `nginx.conf` for the Nginx configuration.
  - `modules` for shared modules.
  - `command` for shared command scripts.

Networks

- **app_network**: Defines a network called `app_network` using the `bridge` driver for inter-container communication.

Key Points

- **Dependencies**: Services like `userService`, `newsfeedService`, `contentService`, `notificationService`, and `subscriptionService` rely on MongoDB, Redis, and RabbitMQ.
- **Environment Variables**: Uses environment variables from a shared `.env` file but overrides specific values for each service as needed.
- **Command Execution**: Uses custom shell commands to ensure dependencies are up before starting the main application.
- **Network Configuration**: All services are connected to a common `app_network` for seamless communication.

This configuration ensures that all services are correctly set up, can communicate with each other, and wait for their dependencies to be ready before starting.

Example:

```
version: "3.8"

services:
  application:
    container_name: application
    image: newsfeed-application
    build: ./frontend/newsfeed
    ports:
      - "3000:3000"
    depends_on:
      - userService
      - subscriptionService
    networks:
      - app_network

  userService:
    container_name: userService
    image: user-service
    build: ./backend/userService
    ports:
      - "8000:8000"
    depends_on:
      - mongo
      - redis
      - rabbitmq
    volumes:
      - ./modules:/app/modules
      - ./command:/app/command
    env_file:
      - .env
    environment:
      - APP_PORT=${APP_PORT_USER}
      - MONGO_NAME=${MONGO_NAME_USER}
      - REDIS_DB=${REDIS_DB_USER}
    networks:
      - app_network
    command:
      [
        "/bin/sh",
        "-c",
        "./command/waitForIt.sh rabbitmq 5672 && node dist/server.js",
      ]

  newsfeedService:
    container_name: newsfeed
    image: newsfeed-service
    build: ./backend/newsFeedService
    ports:
      - "8005:8005"
    depends_on:
      - rabbitmq
      - qdrant
      - mongo
    volumes:
      - ./modules:/app/modules
      - ./command:/app/command
    env_file:
      - .env
    environment:
      - APP_PORT=${APP_PORT_NEWSFEED}
      - MONGO_DB=${MONGO_DB_NEWSFEED}
      - MONGO_USERNAME=${MONGO_USERNAME_NEWSFEED}
      - MONGO_PASSWORD=${MONGO_PASSWORD_NEWSFEED}
      - NEWS_EXPIRE=${NEWS_EXPIRE_NEWSFEED}
      - NEWSDATA_API_KEY=${NEWSDATA_API_KEY_NEWSFEED}
      - NEWSDATA_PATH=${NEWSDATA_PATH_NEWSFEED}
      - NEWSDATA_ENDPOINT=${NEWSDATA_ENDPOINT_NEWSFEED}
      - NEWSDATA_MAX=${NEWSDATA_MAX_NEWSFEED}
      - NEWSDATA_ACTIVE=${NEWSDATA_ACTIVE_NEWSFEED}
      - NEWSAPI_API_KEY=${NEWSAPI_API_KEY_NEWSFEED}
      - NEWSAPI_PATH=${NEWSAPI_PATH_NEWSFEED}
      - NEWSAPI_ENDPOINT=${NEWSAPI_ENDPOINT_NEWSFEED}
      - NEWSAPI_MAX=${NEWSAPI_MAX_NEWSFEED}
      - NEWSAPI_ACTIVE=${NEWSAPI_ACTIVE_NEWSFEED}
      - GNEWS_API_KEY=${GNEWS_API_KEY_NEWSFEED}
      - GNEWS_PATH=${GNEWS_PATH_NEWSFEED}
      - GNEWS_ENDPOINT=${GNEWS_ENDPOINT_NEWSFEED}
      - GNEWS_MAX=${GNEWS_MAX_NEWSFEED}
      - GNEWS_ACTIVE=${GNEWS_ACTIVE_NEWSFEED}
      - QDRANT_PROTOCOL=${QDRANT_PROTOCOL_NEWSFEED}
      - QDRANT_HOST=${QDRANT_HOST_NEWSFEED}
      - QDRANT_PORT=${QDRANT_PORT_NEWSFEED}
      - RESET_NEWS=${RESET_NEWS_NEWSFEED}
    networks:
      - app_network
    command:
      [
        "/bin/bash",
        "-c",
        "./command/waitForIt.sh rabbitmq 5672 && node dist/server.js",
      ]

  contentService:
    container_name: content
    image: content-service
    build: ./backend/contentService
    ports:
      - "8002:8002"
    depends_on:
      - rabbitmq
    volumes:
      - ./modules:/app/modules
      - ./command:/app/command
    env_file:
      - .env
    environment:
      - APP_PORT=${APP_PORT_CONTENT}
    networks:
      - app_network
    command:
      [
        "/bin/sh",
        "-c",
        "./command/waitForIt.sh rabbitmq 5672 && node dist/server.js",
      ]

  notificationService:
    container_name: notification
    image: notification-service
    build: ./backend/notificationService
    ports:
      - "8003:8003"
    depends_on:
      - rabbitmq
      # - redis # Uncomment this line if you want to use redis for caching notificati
    volumes:
      - ./modules:/app/modules
      - ./command:/app/command
    env_file:
      - .env
    environment:
      - APP_PORT=${APP_PORT_NOTIFICATION}
      - EMAIL_SERVICE=${EMAIL_SERVICE_NOTIFICATION}
      - EMAIL_USERNAME=${EMAIL_USERNAME_NOTIFICATION}
      - EMAIL_PASSWORD=${EMAIL_PASSWORD_NOTIFICATION}
    networks:
      - app_network
    command:
      [
        "/bin/sh",
        "-c",
        "./command/waitForIt.sh rabbitmq 5672 && node dist/server.js",
      ]
```

```yaml
  subscriptionService:
    container_name: subscriptionService
    image: subscription-service
    build: ./backend/subscriptionService
    ports:
      - "8004:8004"
    depends_on:
      - rabbitmq
      - redis
      - mongo
    volumes:
      - ./modules:/app/modules
      - ./command:/app/command
    env_file:
      - .env
    environment:
      - APP_PORT=${APP_PORT_SUBSCRIPTION}
      - MONGO_NAME=${MONGO_NAME_SUBSCRIPTION}
      - REDIS_USERNAME=${REDIS_USERNAME_SUBSCRIPTION}
      - REDIS_PASSWORD=${REDIS_PASSWORD_SUBSCRIPTION}
      - REDIS_DB=${REDIS_DB_SUBSCRIPTION}
      - WHITELIST_PROTOCOL=${WHITELIST_PROTOCOL_SUBSCRIPTION}
      - WHITELIST_HOST=${WHITELIST_HOST_SUBSCRIPTION}
      - WHITELIST_PORT=${WHITELIST_PORT_SUBSCRIPTION}
      - WHITELIST_USERNAME=${WHITELIST_USERNAME_SUBSCRIPTION}
      - WHITELIST_PASSWORD=${WHITELIST_PASSWORD_SUBSCRIPTION}
      - WHITELIST_DB=${WHITELIST_DB_SUBSCRIPTION}
      - PARSE_PROMPT=${PARSE_PROMPT_SUBSCRIPTION}
    networks:
      - app_network
    command:
      [
        "/bin/sh",
        "-c",
        "./command/waitForIt.sh rabbitmq 5672 && node dist/server.js",
      ]

  nginx:
    container_name: nginx
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    networks:
      - app_network

  mongo:
    container_name: mongo
    image: mongo:latest
    ports:
      - "27017:27017"
    networks:
      - app_network

  rabbitmq:
    container_name: rabbitmq
    image: rabbitmq:management
    ports:
      - "15672:15672"
      - "5672:5672"
    networks:
      - app_network

  redis:
    container_name: redis
    image: redis:latest
    ports:
      - "6379:6379"
    networks:
      - app_network

  qdrant:
    container_name: qdrant
    image: qdrant/qdrant
    ports:
      - "6333:6333"
    volumes:
      - ./qdrant_data:/qdrant/storage
    networks:
      - app_network

volumes:
  qdrant_data:
  nginx.conf:
  modules:
  command:

networks:
  app_network:
    driver: bridge
```

▼ Continuous integration and deployment

▼ 10. Troubleshooting

▼ Common issues and solutions

▼ Debugging tips