**📘 Project Documentation – Polling App (FastAPI + MongoDB + NewsAPI)**

## Project Documentation

Submitted By : Jeevitha Anisetty

Organization: Zennial Pro private limited

Date : 26-08-202

**📚 Table of Contents**

1. **Introduction**
   * 1.1 Project Overview
   * 1.2 Key Objectives
2. **Technology Stack**
   * 2.1 Backend
   * 2.2 Database
   * 2.3 Authentication
   * 2.4 External API
3. **System Architecture**
   * 3.1 Overview
   * 3.2 Modules
4. **Authentication**
   * 4.1 JWT Token Flow
   * 4.2 Token Structure
   * 4.3 Example Headers
5. **Polling Logic & News Fetching**
   * 5.1 Poll Creation Flow
   * 5.2 Scheduling
   * 5.3 Sample Poll Document
6. **MongoDB Data Models**
   * 6.1 User Model
   * 6.2 Poll Model
7. **API Endpoints**
   * 7.1 Auth Endpoints
   * 7.2 Poll Endpoints
8. **Security Considerations**
   * Hashing password

**Chapter 1: Introduction**

**1.1 Project Overview**

This section provides a thorough description of the Polling Application. It outlines the primary goal of the project: to automatically generate polls every 48 hours based on news headlines fetched from NewsAPI. This project is a **modern, automated polling application** built using **FastAPI** for the backend and **MongoDB** for data storage. The unique feature of this application is its integration with **NewsAPI**, which allows it to **automatically fetch the latest news headlines every 48 hours** and generate poll questions based on titles. This Polling API application using fastAPI and MongoDB provides modern way to create and manage online polls.FastAPI ensures high performance and secure endpoints for user authentication, poll creation, voting, and result retrieval. With features like JWT-based authentication and one-vote-per-user rules, this system combines scalability and flexibility, making it suitable for real-time polling in web or mobile application.

Users can **register and log in using JWT-based authentication**, and then participate in polls by casting their votes. Each poll is time-bound and expires after 48 hours. The backend ensures that users can only vote once per poll and that all sensitive operations are protected through secure authentication and authorization mechanisms.

A background scheduler ensures timely fetching of news and generation of fresh polls, maintaining continuous user engagement with real-world events. The system is designed to be scalable, secure, and extensible for future features like custom poll options, analytics, and real-time voting

**1.2 Key Objectives**

A breakdown of the application’s main objectives:

* Automate poll creation using trending news.
* Enable users to vote securely.
* Maintain system security using JWT.
* Implement a scalable, asynchronous API using FastAPI.
* Store and manage poll data efficiently using MongoDB

**Chapter 2: Technology Stack**

The technology stack refers to the combination of programming languages, frameworks, databases, and tools used to build and run the application. Here’s a breakdown of the key technologies likely involved:

**2.1 Backend – FastAPI**

FastAPI was chosen as the backend framework. It is chosen for its asynchronous capabilities, speed, and automatic OpenAPI documentation.

Authenticating users securely (JWT-based)

Fetching external news data via an API

Interacting with MongoDB asynchronously

Managing timed tasks (like polling every 48 hours)

**2.2 Database – MongoDB**

Details the use of MongoDB for storing documents such as user accounts and polls. Introduces the motor async driver for interacting with MongoDB in an asynchronous environment.

MongoDB is **highly scalable** and can handle:

* Thousands of poll documents
* Rapidly growing vote counts
* High-concurrency reads and writes

**2.3 Authentication – JWT**

The JWT-based authentication mechanism,explains how access tokens are generated, decoded, validated, and used for user identification and authorization. It makes FastAPI app becomes lighter and faster — great for scalability and performance.

JWT allows **stateless authentication** — meaning the server doesn’t need to store any session data.

* When a user logs in, the server generates a token.
* This token is sent to the client and stored (e.g., in localStorage or cookies).
* The client includes the token in the header of every request

**2.4 External API – NewsAPI**

Introduces the NewsAPI integration for fetching the latest news headlines. **NewsAPI** is a RESTful API that provides **real-time news headlines** from thousands of trusted sources like: BBC News, CNN, Bloomberg, Reuters etc.

It lets you search for current or historical news by: topic , source, country, language, keywords etc.

NewsAPI provides exactly the kind of data needed:

* Up-to-date news titles
* Authoritative sources
* Concise headlines perfect for poll questions

**Chapter 3: System Architecture**

**3.1 Overview**

A high-level view of how the system components interact. Describes the relationships between the user, FastAPI endpoints, MongoDB database, and external services like NewsAPI. The system connects users to a polling platform through FastAPI endpoints, which handle requests like registration, login, and voting. FastAPI communicates with a MongoDB database to store user info, polls, and votes. Meanwhile, an external service, NewsAPI, regularly supplies fresh news headlines that the system automatically turns into new poll questions. This setup ensures users get up-to-date polls while keeping the system responsive, secure, and easy to maintain.

**3.2 Modules**

In-depth explanation of major application modules:

* **Authentication Module**: Handles user registration, login, and JWT validation.
* **Polling Module**: Responsible for poll creation, listing, and deletion.
* **Voting Module**: Manages vote casting and tallying.
* **News Fetcher(services)**: Communicates with NewsAPI.
* **Database Layer**: Handles interactions with MongoDB

**3.3 Project-Structure:**

polling\_app/

│

├── app/

│ ├── core/ # Core configuration and authentication

│ │ ├── auth.py # JWT creation & verification

│ │ └── config.py # Environment settings (via pydantic/BaseSettings)

│

│ ├── database/

│ │ └── db.py # MongoDB Motor connection and client setup

│

│ ├── models/

│ │ ├── user.py # User Pydantic & Mongo schema

│ │ └── polls.py # Poll schema (question, options, timestamps)

│

│ ├── routes/

│ │ ├── user\_routes.py # Routes: /auth/register, /auth/login

│ │ └── poll\_routes.py # Routes: /polls/, /vote, etc.

│

│ ├── services/

│ │ ├── services.py # Business logic: fetch news, create polls, voting

│

│ ├── utils/

│ │ ├── decorators.py # Route protection decorators (e.g., @login\_required)

│ │ ├── hashing.py # Password hashing/verification (bcrypt)

│ │ └── logger.py # Custom logger for file logging

│

├── logs/

│ └── polling\_app.log # App logs (info, error, requests)

│

├── main.py # App entry point (FastAPI instance, route inclusion)

├── requirements.txt # Python dependencies

├── postman\_collection.json # Postman collection for testing API

Finally, the **Project Structure** outlines the organization of the codebase:

* The **app** directory contains all core components:
  + **core** for configuration and authentication logic (JWT handling and environment settings).
  + **database** for MongoDB connection setup.
  + **models** for defining data schemas for users and polls.
  + **routes** to define API endpoints for user and poll-related operations.
  + **services** containing the business logic such as fetching news, creating polls, and managing votes.
  + **utils** with helpers like route protection decorators, password hashing utilities, and logging.
* Additional files and folders include:
  + logs for application logging.
  + main.py as the FastAPI app entry point.
  + requirements.txt listing project dependencies.
  + A postman\_collection.json file for API testing.

This architecture promotes modularity, clarity, and separation of concerns, making the application easier to maintain and extend.

**Chapter 4: Authentication**

This explains how JWT (JSON Web Token) is used for user authentication in a web application. The process begins when a user registers and their information is stored securely. Upon logging in, the server verifies their credentials and issues a JWT token. This token is then stored on the client side and sent along with every request to protected routes using the HTTP Authorization header in the format: Bearer <token>. The server checks the token’s validity and expiration before granting access.

The JWT itself consists of three parts: a header, a payload, and a signature. The header specifies the algorithm and token type, the payload carries user-related data like the username and an expiration time (exp), and the signature ensures the token’s integrity. If the token is expired or tampered with, access is denied. To maintain a secure session, strategies like token expiration and refresh tokens are used to re-authenticate users without requiring them to log in repeatedly.

**4.1 JWT Token Flow**

A detailed explanation of the JWT authentication workflow:

* Registering a new user.
* Logging in to generate a JWT.
* Validating tokens on protected routes.
* Token expiration and refresh strategies.

**4.2 Token Structure**

Describes the content and encoding of JWT tokens:

* **Header**: Algorithm & type.
* **Payload**: Contains username, and expiration (exp).
* **Signature**: Ensures integrity.

**4.3 Example Headers**

Shows how tokens are attached to HTTP requests:

Authorization: Bearer <token>

**Chapter 5: Polling Logic & News Fetching**

This chapter describes how the system creates polls based on news headlines. It starts with a timer that regularly fetches the latest news from NewsAPI. Each news headline is converted into a poll question, which is then saved in the database with default answer options such as “Agree” and “Disagree.” Each poll records when it was created and is set to expire automatically after 48 hours.

This also provides an example of a poll’s data structure in MongoDB, showing how the question, available options with vote counts, and timestamps for creation and expiration are stored.

**5.1 Poll Creation Flow**

Step-by-step breakdown:

1. Fetches news headlines from NewsAPI.
2. Each headline is turned into a poll question.
3. Polls are saved with default options like “Agree” and “Disagree”.
4. Polls include timestamps for creation and automatic expiration after 48 hours

**5.2 Sample Poll Document**

Shows the MongoDB document structure for a poll, including:

* Question (from the news title)
* Options and votes
* Created and expiration timestamps

**Chapter 6: MongoDB Data Models**

Together, these models form the backbone of the application’s data storage:

* The **User Model** ensures secure authentication and access control.
* The **Poll Model** keeps track of poll content, voting options, vote counts, timing, and voter participation.

Using schemas in MongoDB with these fields helps maintain data integrity, enforce business rules (like single voting), and support features such as automatic poll expiration.

**6.1 User Model**

Describes the user schema:

* Username : A unique identifier for each user, usually a string. This is what users use to log in or are identified by in the system.
* hashed\_password : Instead of storing plain-text passwords (which is a big security risk), passwords are hashed using a strong hashing algorithm like **bcrypt**. Hashing transforms the original password into a fixed-length string that is practically impossible to reverse. This protects user credentials even if the database is compromised
* role : This field determines the user’s permissions within the application. Common roles might include:
  + **user** (default role with normal permissions)
  + **admin** (with elevated privileges like managing polls or users)

Roles allow the system to enforce access control and security policies by checking the user’s role before performing certain actions.

* Optional roles (e.g., admin)

Security measures like bcrypt hashing are discussed.

**6.2 Poll Model**

Describes the poll schema:

* Question : A string containing the poll question, often generated from news headlines in this application.
* Options : An array (list) of objects where each object contains:
  + The **option** text (e.g., “Agree”, “Disagree”)
  + The **votes** count for that option

This structure makes it easy to tally votes per option and display resu

* Created\_at and Expires\_at : Timestamps to track when the poll was created and when it will automatically expire. This allows the system to close polls after a certain period (like 48 hours), preventing voting after the deadline.
* Voters : A list of user IDs who have already voted in the poll. This prevents a user from voting multiple times, ensuring fairness. When a user votes, their ID is added to this list, and the system checks it before accepting new votes from that user.

**Chapter 7: API Endpoints**

Chapter 7 defines a clear, role-based API system:

* **Authentication endpoints** let users securely register and log in, receiving JWT tokens for session management.
* **Poll endpoints** enable users to view active polls, vote securely, and allow admins to manage polls directly.
* Security is enforced both by input validation (e.g., email format) and authorization checks (JWT for voting, admin roles for poll management).
* The system ensures fairness by preventing multiple votes per user and managing poll lifetimes with expiration times.

**7.1 Auth Endpoints**

**Register** – POST /register  
Accepts username and password, stores hashed password. Accepts email only as username and both password and username should be valid.

* This endpoint handles new user registration.
* It expects the client to send a **username** and **password**.
* The username must be a valid email address, ensuring a proper format.
* The password and username undergo validation checks (e.g., password strength, email format).
* The password is **hashed securely** (e.g., using bcrypt) before being stored in the database to protect user credentials.
* Once the user is registered, their data is saved, and the user can log in.

**Login** – POST /login  
Verifies credentials and returns a JWT token with expiry.

* This endpoint verifies a user’s credentials.
* The client sends the username (email) and password.
* The server compares the submitted password (after hashing) with the stored hash.
* If the credentials match, the server issues a **JWT token** which includes an expiration time.
* The JWT token allows the user to authenticate themselves in future requests without resubmitting credentials.
* The expiration time on the token helps manage session length and security.

**7.2 Poll Endpoints**

**List Polls** – GET /polls  
Returns active polls with 24 hours expire time and latest polls from newsapi generated within last 48 hours.

* Returns a list of **active polls**.
* These include polls that are still valid within a **24-hour expiry window**.
* It also includes the latest polls generated automatically from news headlines fetched via the NewsAPI, covering polls created within the last **48 hours**.
* This endpoint is likely public and does not require authentication, allowing users to view available polls easily.

**Vote on a Poll** – POST /polls/vote  
Requires authentication. Whenever user is authenticated it accepts an option and updates poll voting count and voted user id.

* Requires the user to be **authenticated** via the JWT token.
* The user submits a poll ID and their selected option.
* The server checks that the user has not already voted on that poll (using the voters list in the poll model).
* If eligible, the server updates the vote count for the chosen option and adds the user’s ID to the list of voters to prevent multiple votes.
* This endpoint ensures voting integrity and accurate vote tallying.

**Create Poll (Admin)** – POST /admin/polls  
Allows admin to manually add a poll.

* Restricted to **admin users**.
* Allows administrators to manually create polls, overriding the automatic news-based poll creation.
* Admins can specify the poll question and options.
* This endpoint requires role-based authorization checks to ensure only admins can access it.

**Delete Poll (Admin)** – POST/admin/delete\_poll  
Removes a poll. Secured via admin check.

* Restricted to  **users**.
* Enables admins to delete existing polls from the system.
* Helps maintain poll quality and remove inappropriate or outdated polls.
* Like creation, this action requires secure authorization checks.

**Chapter 8: Security Considerations**

* Storing secrets (e.g., NewsAPI key, JWT secret) in .env. Which is secure and safe.
* Input validation via Pydantic models.
* Password hashing using bcrypt.
* Avoiding duplicate votes by tracking voters. Allowing only one vote per user.

First, it emphasizes the importance of **securely storing sensitive information**, such as the NewsAPI key and JWT secret. These secrets are kept in environment variables (commonly in a .env file) rather than hard-coded in the source code. This approach protects secrets from exposure in version control systems and limits access to authorized environments only.

Next, the chapter highlights **input validation** as a critical defense mechanism. Using **Pydantic models**, all incoming data—like user inputs for registration or poll creation—is strictly validated against predefined schemas. This reduces risks such as injection attacks or malformed data causing unexpected behavior.

For password security, the system uses **bcrypt hashing**. Rather than storing plain-text passwords, bcrypt applies a computationally expensive hashing algorithm, making it practically impossible for attackers to reverse-engineer passwords even if they gain access to the database. This hashing step protects users’ credentials from being compromised.

Finally, to preserve the integrity of poll voting, the system **prevents duplicate votes** by keeping track of users who have already voted on a poll. By maintaining a list of voter IDs, it enforces a strict one-vote-per-user rule, ensuring fair and reliable poll results.

Together, these security measures safeguard both user data and system functionality, helping to build a secure, trustworthy application.