**Blackhole Infiverse Assignment**

**MERN-NLP-Project**

**Overview**

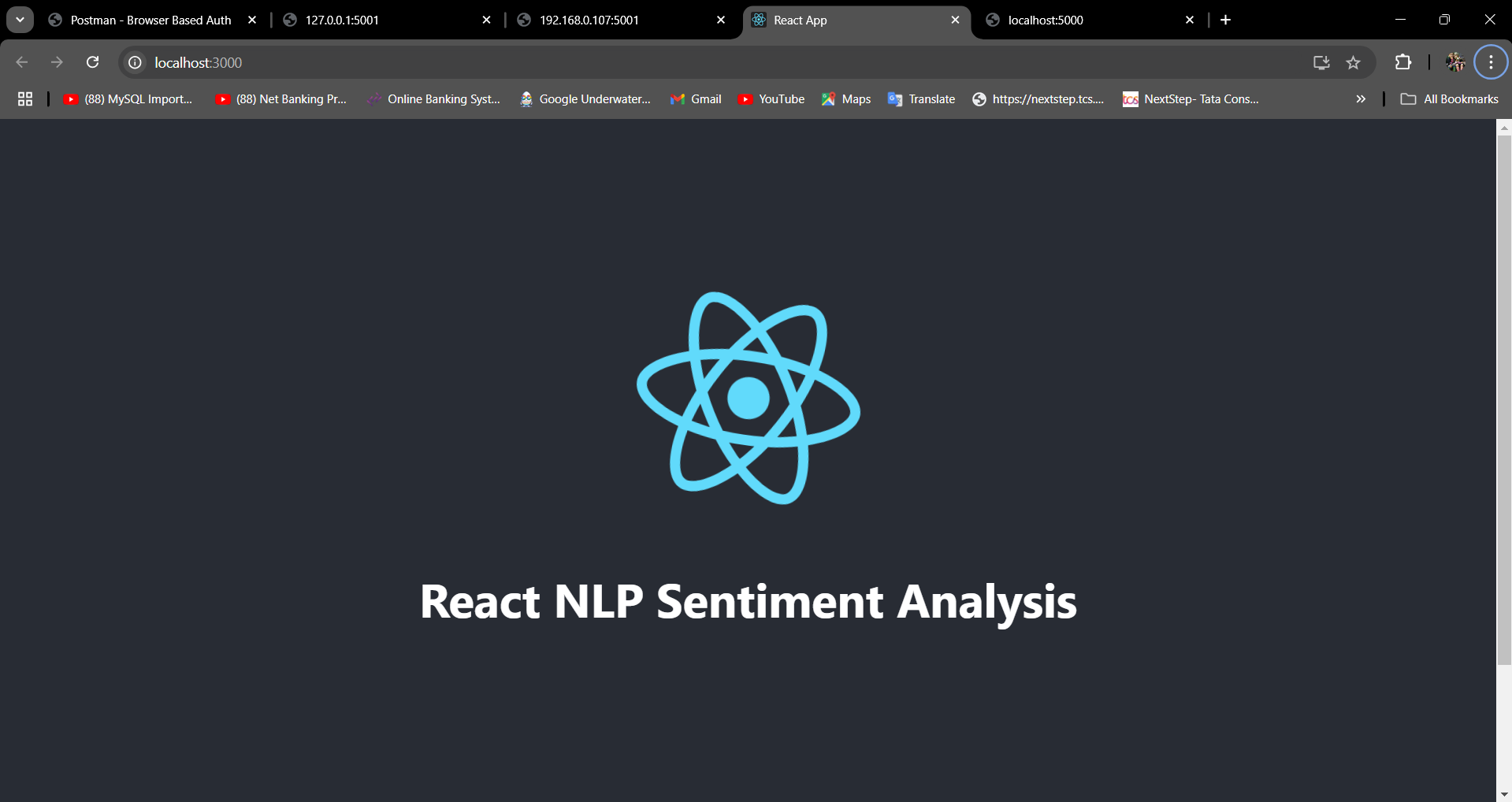
This project is a **web-based application** developed using the **MERN** stack (MongoDB, Express.js, React.js, and Node.js), combined with a **Flask-based NLP API** for sentiment analysis. The website enables users to input text and analyze its sentiment (positive or negative) by utilizing a pre-trained model hosted in a Flask API. The results of the sentiment analysis are then stored in a MongoDB database for future reference.

The goal of the project is to create an interactive and user-friendly interface for sentiment analysis using the latest web technologies, providing real-time feedback to users and storing results in the backend for analytics and reporting.

**Features**

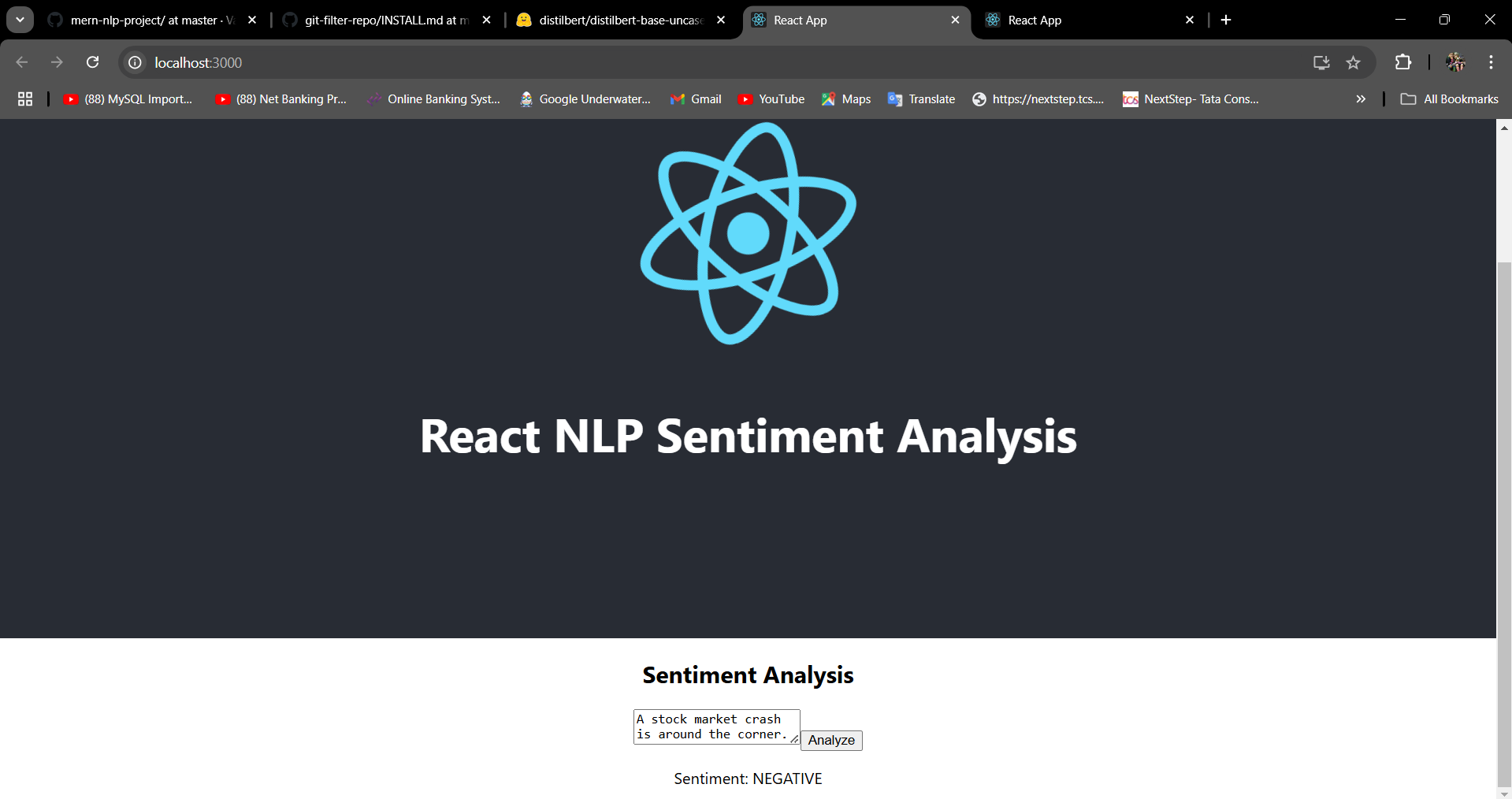
**1. User-Friendly Interface (Frontend)**

* The **frontend** of the website is built using **React.js**, providing a dynamic and responsive user interface.
* Users can easily input text via an input field, and the sentiment of the text is analyzed in real time upon submission.
* The website provides clear visual feedback, displaying whether the text has a **positive** or **negative** sentiment.
* The interface is designed to be clean and intuitive, making it accessible for users with no technical expertise.



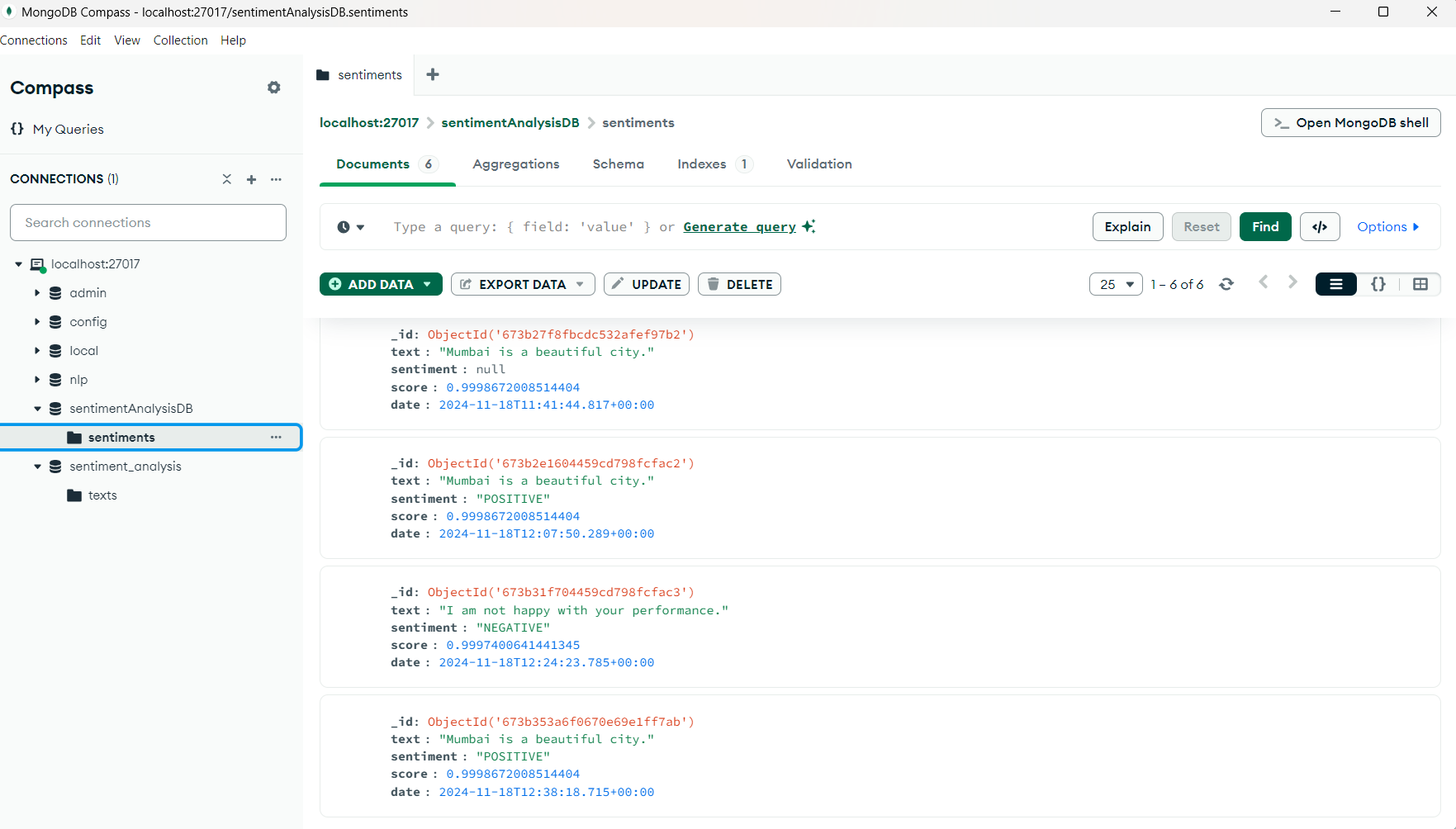
**2. Real-Time Sentiment Analysis**

* Upon text submission, the website communicates with the **Flask-based NLP API**, which processes the text and returns the sentiment result.
* The sentiment analysis is powered by a **pre-trained model** from **Hugging Face**, specifically fine-tuned for sentiment classification on the SST-2 dataset.
* The backend handles requests asynchronously, ensuring that the website provides real-time results without any delays.



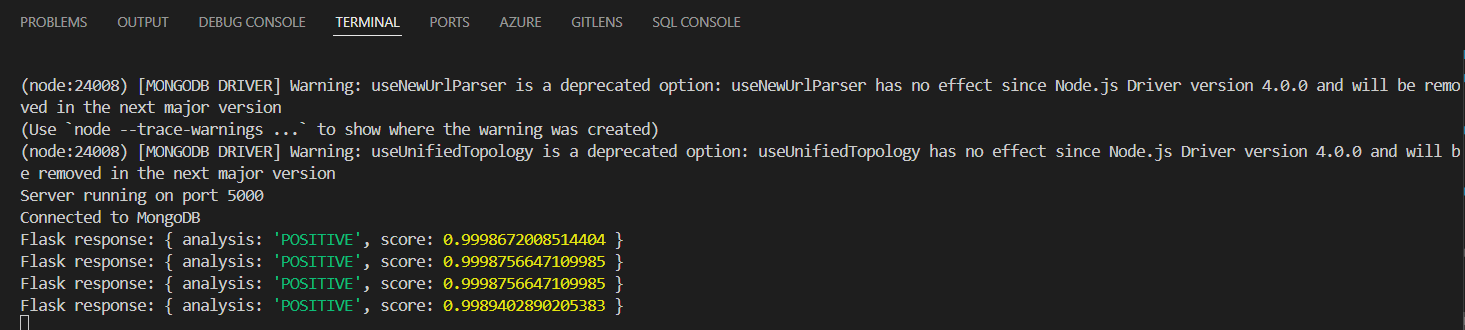
**3. Sentiment Storage in MongoDB**

* Once sentiment analysis is completed, the result is stored in a **MongoDB database**, providing the website with the ability to store a history of analyzed texts.
* The database stores both the original text input and the sentiment result (positive or negative), which can be useful for analytics and data-driven insights in future improvements.
* MongoDB is chosen for its flexibility and scalability, allowing the website to handle large amounts of data efficiently.



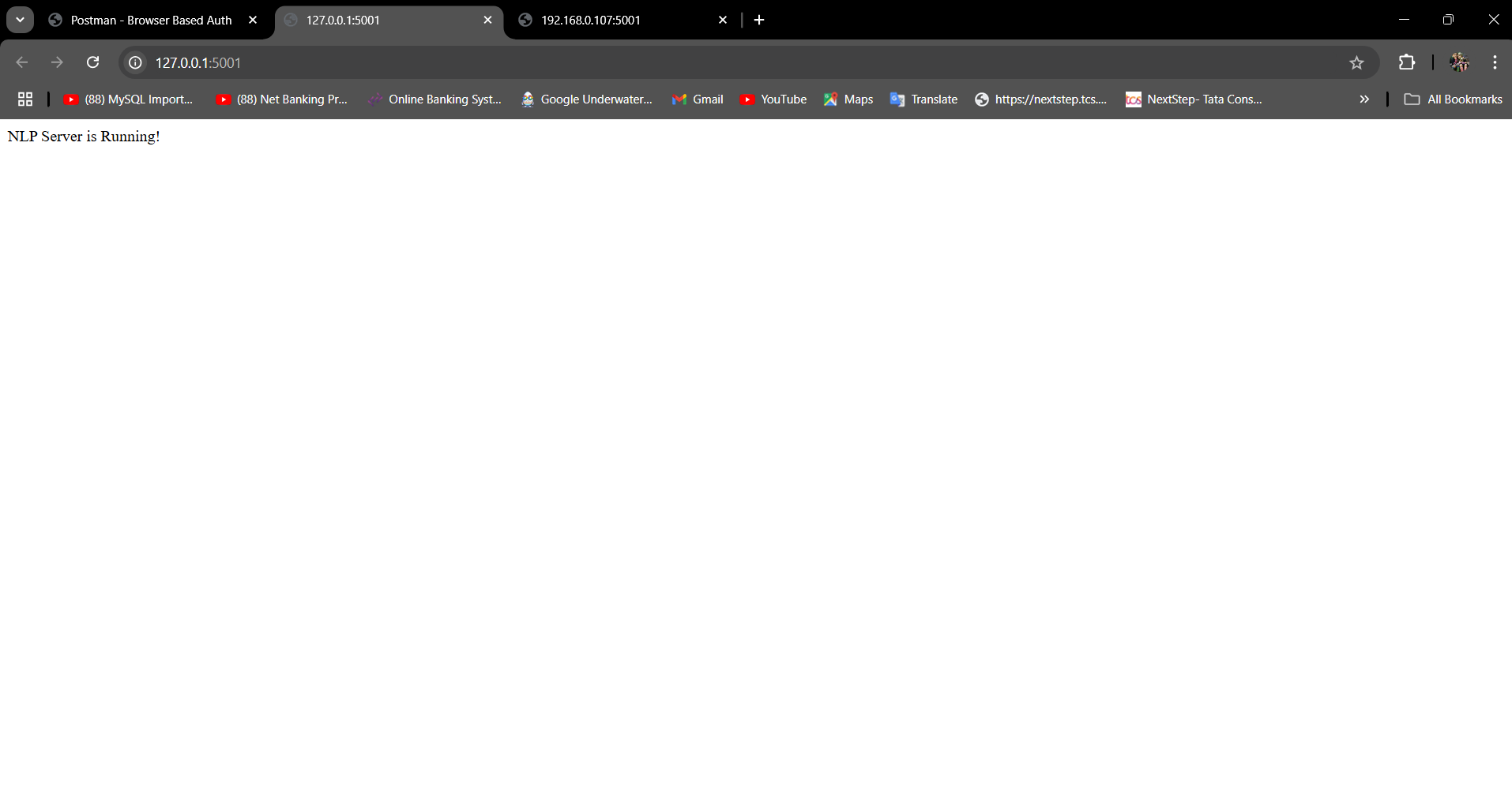
**4. Secure and Scalable Backend (Node.js & Express)**

* The **backend** of the website is built using **Node.js** with **Express.js**, which provides a robust, scalable, and secure platform for handling user requests.
* The backend handles the logic of sending text to the Flask API for sentiment analysis, as well as interacting with the MongoDB database for storing results.
* Express.js also provides routing capabilities, allowing smooth communication between the frontend and the API.



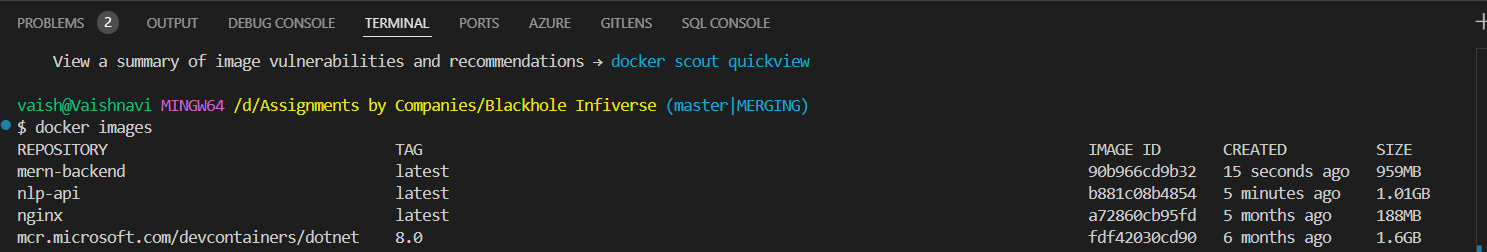
**5. Flask-based Sentiment Analysis API**

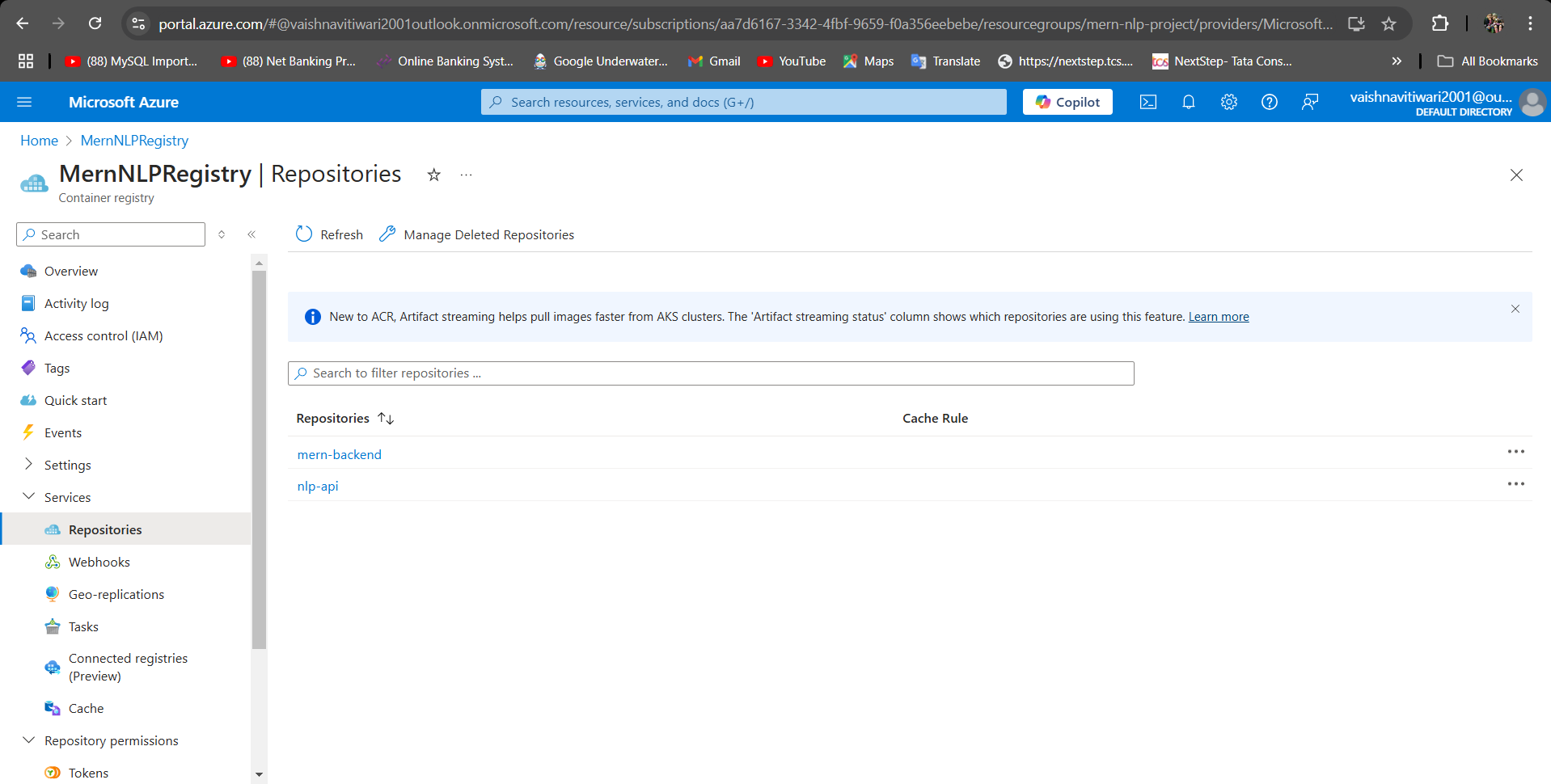
* The sentiment analysis is handled by a **Flask API**, which is a lightweight Python framework. The Flask server communicates with a pre-trained **DistilBERT** model from **Hugging Face**.
* The API takes the user input, processes it, and returns whether the text sentiment is positive or negative, based on the fine-tuned model.
* The Flask API runs on a separate server, allowing the backend to remain lightweight while handling NLP tasks.



**6. Seamless Integration of MERN Stack**

* The project effectively integrates the **MERN stack**, combining MongoDB for database storage, Express.js for handling server-side logic, React.js for building the user interface, and Node.js as the server-side platform.
* The seamless integration between the frontend and backend ensures smooth user interactions and a consistent user experience across the application.
* The frontend communicates with the backend through API calls, ensuring that data flow and user actions are processed in real time.





Vaishnavi’s Task:

To align with our existing tech stack (MERN, Python, Flutter, Figma etc.), you will have to use those technologies for both the DevOps and NLP components

Timeline: 2-3 hour task:

Task Overview:

1. DevOps and Cloud Infrastructure Setup: Set up the backend infrastructure and CI/CD pipeline for deploying an NLP-based web service using the MERN stack, Python, and cloud technologies.

2. NLP Implementation: Build a basic NLP application that processes text input and deploy it to a cloud server (using Python for AI/ML and integrating it with your MERN stack frontend).

Part 1: DevOps and Cloud Infrastructure Setup (1-1.5 hours)

Objective: Set up a cloud infrastructure to deploy a simple NLP model as a service, integrated with a MERN (MongoDB, Express, React, Node.js) backend.

Task:

1. Backend Setup with MERN Stack:

- Create a Node.js/Express server that serves a simple REST API.

- Integrate MongoDB to store user input (text) and corresponding analysis results (sentiment, etc.).

- Set up routes and controllers for:

- Accepting POST requests with text data.

- Storing and retrieving text data and analysis results from the database.

2. NLP Model with Python:

- Use Python to build a simple NLP model (e.g., sentiment analysis) using a pre-trained model from Hugging Face.

- Expose this model via an API endpoint using Flask (Python-based web framework).

- The Python Flask API should:

- Accept text from a request.

- Run the NLP model to analyze sentiment (positive/negative).

- Return the result back to the MERN backend for further use.

3. CI/CD Pipeline Setup:

- Create a GitHub Actions or GitLab CI pipeline to automate the deployment of the app.

- The pipeline should handle:

- Automatic deployment of the MERN backend (Node.js/Express app).

- Automatic deployment of the Python Flask app serving the NLP model.

- Include necessary steps to ensure that the Node.js backend and Python NLP service can communicate, either through a shared API or using Docker for containerized services.

4. Containerization (Optional for Advanced Candidates):

- Containerize the Node.js backend and Python Flask API using Docker.

- Create a Docker Compose file to manage the services (Node.js + Python Flask) and make them easily deployable.

Part 2: NLP Model Development (1-1.5 hours)

Objective: Implement a basic NLP model for sentiment analysis and integrate it with the MERN stack backend.

Task:

1. Text Analysis (NLP Model):

- Choose a pre-trained model from Hugging Face (e.g., DistilBERT or BERT for sentiment analysis) or use spaCy with a simple sentiment analysis pipeline.

- Implement a Python function to:

- Preprocess the input text (tokenization, normalization, etc.).

- Use the model to predict sentiment (positive, negative, or neutral).

- Expose this functionality via a Flask API.

2. Integrating NLP API with MERN Backend:

- Update the Node.js/Express server to send text to the Python Flask API for sentiment analysis.

- The Node.js server should:

- Accept POST requests containing text from the user.

- Send the text to the Python NLP API via HTTP requests (using Axios or Node Fetch).

- Store the text and sentiment result in MongoDB.

- Return the sentiment result to the client (React frontend).

3. Frontend Integration (React):

- Implement a simple React frontend that:

- Allows the user to input text.

- Sends the input to the backend API.

- Displays the sentiment analysis result (positive/negative).

- Ensure the React app works well with the MERN stack API.

4. Testing and Debugging:

- Test the full flow from the frontend (React) to backend (Node.js/Express) to the Python Flask API and back.

- Ensure that the sentiment analysis result is returned and displayed properly in the frontend.

Evaluation Criteria:

1. Technical Competence:

- Proficiency with MERN stack (Node.js, Express, React, MongoDB).

- Ability to integrate Python-based NLP models into the backend services.

- Understanding of DevOps practices, specifically related to CI/CD and cloud deployments.

2. AI/ML Knowledge:

- Ability to use pre-trained NLP models and deploy them via a Python API (Flask).

- Experience in data preprocessing, model inference, and result handling.

3. DevOps Skills:

- Effective use of CI/CD pipelines (GitHub Actions or GitLab CI).

- Experience with Docker for containerizing services (if attempted).

- Deployment of both Node.js and Python Flask apps using cloud or local services.

4. Frontend Skills:

- Proficiency in React for frontend development and API integration.

- Ability to design a user-friendly UI for interacting with the sentiment analysis service.

5. Problem-Solving and Time Management:

- Ability to troubleshoot deployment and integration issues.

- Efficiently managing time to complete all tasks within the 2-3 hour window.

6. Documentation:

- Clear and concise documentation for setting up and running the application, including setup instructions for the development environment, CI/CD pipelines, and deployment steps.

Deliverables:

- GitHub repository containing:

- Node.js/Express backend code.

- Python Flask API for NLP model.

- Docker Compose (if applicable).

- React frontend code.

- CI/CD pipeline configuration (GitHub Actions or GitLab CI).

- Documentation with setup and deployment instructions.

Optional Enhancements (If Time Permits):

- Deploying the project to the cloud (e.g., using Heroku, Azure etc.)

- UI/UX Design: If you are familiar with Figma, you can design a simple UI mockup for the frontend before implementing it in React.

This task will test the your ability to work with both backend development and AI/ML integration while utilizing our stack, allowing you to showcase your skills in a real-world scenario.

--------------------------------------------------------------------------------------------------------------------------------

This is a well-structured task designed to evaluate a candidate's ability to combine DevOps, NLP, and full-stack development skills. Here's a suggested approach to tackle it within the given timeline:

---

### \*\*Plan of Execution\*\*

#### \*\*Part 1: DevOps and Cloud Infrastructure Setup (1-1.5 hours)\*\*

1. \*\*Backend Setup with MERN Stack (20-30 mins):\*\*

- \*\*Create Node.js/Express server:\*\*

- Initialize the project with `npm init`.

- Install dependencies: `express`, `mongoose`, and `body-parser`.

- Define routes:

- \*\*POST `/analyze`:\*\* Accept text input and trigger sentiment analysis.

- \*\*GET `/results`:\*\* Fetch stored analysis results from MongoDB.

- Integrate MongoDB using `mongoose` and define schemas for storing user input and analysis results.

2. \*\*NLP Model with Python Flask (20-30 mins):\*\*

- Set up a Flask app with routes for:

- \*\*POST `/predict`:\*\* Accepts text, processes it using a pre-trained model (e.g., Hugging Face's DistilBERT or spaCy), and returns the sentiment result.

- Use libraries: `transformers`, `torch`, and `flask`.

- Test the Flask API locally with sample requests.

3. \*\*CI/CD Pipeline Setup (30 mins):\*\*

- Configure \*\*GitHub Actions\*\* for:

- Testing and deploying the Node.js backend.

- Building and deploying the Flask API.

- Define separate workflows for:

- \*\*Backend (Node.js):\*\* Install dependencies, run tests, and deploy.

- \*\*Python Flask API:\*\* Set up Python environment, install dependencies, and deploy.

- Use Docker to containerize the services if time permits.

4. \*\*Containerization (Optional, 10-15 mins):\*\*

- Write a `Dockerfile` for each service (Node.js and Flask).

- Combine services using a `docker-compose.yml` file.

---

#### \*\*Part 2: NLP Model Development (1-1.5 hours)\*\*

1. \*\*Text Analysis with NLP Model (30 mins):\*\*

- Load a pre-trained sentiment analysis model from Hugging Face.

- Implement a function for:

- Tokenization, preprocessing, and inference.

- Returning sentiment labels (positive, negative, neutral).

- Integrate this into the Flask API and test using `Postman` or `cURL`.

2. \*\*Backend Integration (20 mins):\*\*

- Update Node.js routes to communicate with the Flask API:

- Use `axios` to send text input to Flask and receive sentiment results.

- Store the text and results in MongoDB.

3. \*\*Frontend Integration (20 mins):\*\*

- Develop a simple React frontend:

- A form for text input and a button to submit the text.

- Display the returned sentiment analysis result.

- Use `axios` or `fetch` to communicate with the Node.js backend.

4. \*\*Testing and Debugging (10-20 mins):\*\*

- End-to-end testing:

- Submit input from React frontend.

- Verify storage in MongoDB and correct sentiment output.

- Debug any issues in communication between services.

---

### \*\*Optional Enhancements\*\*

- Deploy the application to a cloud platform (Azure/Heroku/GCP) for demonstration.

- Design a UI mockup using Figma to align with the task requirements.

- Add Docker Compose for streamlined multi-service deployment.

---

### \*\*Final Deliverables\*\*

1. \*\*Codebase:\*\*

- Organized into directories for backend (Node.js), NLP API (Flask), and frontend (React).

- Docker files and CI/CD configurations.

2. \*\*Documentation:\*\*

- Setup instructions.

- CI/CD pipeline explanation.

- Deployment guide.

3. \*\*Optional:\*\* Deployed app URL or screenshots for demonstration.

This systematic approach ensures task completion within the time limit while maintaining quality and functionality. Would you like help setting up any specific part of this task?

Blackhole Infiverse Assignment

**Install Required Dependencies**

Run the following commands to install necessary packages:

* **Express**: Framework for building the server.
* **Mongoose**: MongoDB ORM.
* **Body-parser**: Parse incoming request bodies.
* **Nodemon**: Automatically restarts the server during development.
* **CORS**: Enable cross-origin resource sharing (useful when integrating with React frontend later).