**1. Project Outline: Windsurf Customer Service Complaint Agent**

**Overview:** The Windsurf Customer Service Complaint Agent is an AI-powered web application designed to streamline the processing of customer complaint letters for a windsurfing business. By leveraging advanced natural language processing capabilities, the system aims to enhance customer service efficiency and improve communication quality.

**Key Features:**

* **Text Processing Modes:** The application offers two primary modes of text processing:
  + **Rewrite (Brief Herschrijven):** This mode maintains the core message of the complaint while improving its professional tone, logical structure, and ensuring no information is invented. It concludes with a standardized closing: "Met Vriendelijke Groeten."
  + **Response (Antwoord Genereren):** This mode generates a response that demonstrates understanding, addresses all points of the complaint, provides background information, and suggests prevention measures. It maintains an empathetic, professional tone and concludes with the same standardized closing.
* **File Handling:** The application supports various document formats, including .txt, .doc, .docx, and .pdf, allowing users to upload complaint letters in their preferred format.
* **User Interface:** A user-friendly interface built with React and Material-UI components enables easy file uploads, text processing, and selection of processing types via radio buttons.
* **Security and Compliance:** The system implements rate limiting, input validation, and file upload size restrictions (5MB) to ensure security and compliance.

**Purpose and Benefits:** The Windsurf Customer Service Complaint Agent aims to assist customer service representatives by automating the initial stages of complaint handling. By providing rewritten or generated responses, the system saves time, reduces human error, and ensures consistent communication standards. This ultimately leads to enhanced customer satisfaction and operational efficiency for the windsurfing business.

**2. Tech Stack and Package Documentation**

**Tech Stack:**

* **Backend:**
  + **Node.js:** A JavaScript runtime environment that executes server-side code.
  + **Express:** A web application framework for Node.js, used for building the RESTful API.
  + **MongoDB:** A NoSQL database for potential future data storage.
  + **OpenAI API (GPT-4):** Used for natural language processing to rewrite and generate responses to customer complaints.
* **Frontend:**
  + **React:** A JavaScript library for building user interfaces.
  + **Material-UI:** A React component library for implementing Google's Material Design.
  + **Axios:** A promise-based HTTP client for making API requests.

**Package Documentation:**

* **Backend Packages:**
  + **Express:** Provides a robust set of features for web and mobile applications. Used for routing and handling HTTP requests.
  + **Mongoose:** A MongoDB object modeling tool designed to work in an asynchronous environment.
  + **Multer:** A middleware for handling multipart/form-data, used for file uploads.
  + **OpenAI (v4.20.1):** Provides access to OpenAI's language models for text processing.
  + **pdf-parse:** A library for extracting text from PDF files.
  + **mammoth:** Converts .docx files into plain HTML and text.
* **Frontend Packages:**
  + **React (18.2.0):** Enables the building of interactive UIs with components.
  + **Material-UI:** Provides pre-designed components to create a responsive UI.
  + **Axios:** Facilitates communication between the frontend and backend by making HTTP requests.

**Development and Deployment:**

* **Version Control:** Git and GitHub are used for version control and collaboration.
* **Deployment:** Render.com is used for deploying the application, with specific build and start commands to ensure proper execution.

**3. Schema Design**

**Overview:** While the current implementation of the Windsurf Customer Service Complaint Agent does not actively utilize MongoDB for data storage, the schema design is prepared for potential future integration. This design focuses on storing customer complaints and their associated metadata.

**Schema Components:**

1. **Complaint Schema:**
   * **complaintId (String):** A unique identifier for each complaint.
   * **customerName (String):** The name of the customer who submitted the complaint.
   * **customerEmail (String):** The email address of the customer.
   * **complaintText (String):** The original text of the complaint.
   * **processedText (String):** The rewritten or generated response text.
   * **processingType (String):** Indicates whether the text was rewritten or a response was generated (e.g., "Rewrite", "Response").
   * **submissionDate (Date):** The date and time when the complaint was submitted.
   * **status (String):** The current status of the complaint (e.g., "Pending", "Processed").
2. **User Schema (For Future Authentication):**
   * **userId (String):** A unique identifier for each user.
   * **username (String):** The username for login purposes.
   * **passwordHash (String):** A hashed version of the user's password for secure authentication.
   * **role (String):** The role of the user (e.g., "Admin", "Customer Service").

**Database Design Considerations:**

* **Scalability:** The schema is designed to accommodate a growing number of complaints and users, ensuring efficient data retrieval and storage.
* **Security:** Sensitive information, such as passwords, is stored securely using hashing techniques.
* **Flexibility:** The schema allows for easy expansion, enabling additional fields or collections as new features are implemented.

**4. Detailed App Flow**

**Overview:** The Windsurf Customer Service Complaint Agent is designed to provide a seamless user experience from complaint submission to text processing. The following outlines the detailed flow of the application, highlighting key interactions between the frontend, backend, and external services.

**App Flow:**

1. **User Interface:**
   * **File Upload:** Users can upload complaint documents in supported formats (.txt, .doc, .docx, .pdf) through a simple drag-and-drop interface or file selector.
   * **Processing Type Selection:** Users choose between "Rewrite" and "Response" options using radio buttons to specify the desired text processing mode.
2. **Frontend to Backend Communication:**
   * **Form Submission:** Upon selecting a file and processing type, the user submits the form, triggering an HTTP POST request to the backend server using Axios.
   * **Data Transmission:** The selected file and processing type are transmitted as multipart/form-data to the backend endpoint.
3. **Backend Processing:**
   * **File Handling:** The backend uses Multer to handle file uploads and pdf-parse or mammoth to extract text from the uploaded documents.
   * **Text Processing:** The extracted text is sent to the OpenAI API, where the chosen processing mode (Rewrite or Response) is applied using GPT-4. The API returns the processed text.
   * **Response Construction:** The backend constructs a response containing the processed text and any relevant metadata.
4. **Backend to Frontend Communication:**
   * **Response Delivery:** The processed text and metadata are sent back to the frontend as a JSON response.
   * **Display:** The frontend updates the user interface to display the processed text, allowing users to review and download the results.
5. **Error Handling and Security:**
   * **Input Validation:** The backend validates the uploaded file type and size, ensuring compliance with security standards.
   * **Error Feedback:** Any errors encountered during processing are communicated back to the frontend, providing users with clear feedback and guidance.

**Additional Features:**

* **Rate Limiting:** Implemented to prevent abuse and ensure fair usage of the application.
* **Standardized Closings:** Both processing modes conclude with "Met Vriendelijke Groeten" to maintain a consistent and professional tone.

**5. Points to Learn: Resolved Errors and Solutions**

**Overview:** Throughout the development of the Windsurf Customer Service Complaint Agent, several challenges were encountered and resolved. These experiences offer valuable learning opportunities for future projects.

**Key Points and Solutions:**

1. **Frontend Folder as a Git Submodule:**
   * **Issue:** The **frontend** folder was mistakenly tracked as a Git submodule, causing deployment issues and preventing access to its contents on GitHub.
   * **Solution:** The **.git** directory within the **frontend** folder was deleted, and the folder was re-added to the main repository as regular files. This resolved the submodule issue and ensured proper tracking of frontend files.
2. **Hidden Files Not Deleted Locally:**
   * **Issue:** Hidden files, such as **.git** directories, were not visible in the file explorer, leading to confusion during troubleshooting.
   * **Solution:** Enabling the display of hidden files in the file explorer allowed for the identification and deletion of unnecessary **.git** folders, preventing Git from treating directories as submodules.
3. **Deployment Build Command Errors:**
   * **Issue:** Incorrect build commands in the Render.com configuration led to failed deployments.
   * **Solution:** The build command was updated to ensure all dependencies were installed and the React app was built correctly. The command was structured as follows:

Code

CopyInsert

npm install && cd frontend && npm install && npm run build && cd ..

1. **File Upload and Processing Errors:**
   * **Issue:** Errors occurred when uploading unsupported file types or exceeding size limits.
   * **Solution:** Input validation middleware was implemented to enforce file type and size restrictions, providing users with clear error messages and preventing processing failures.
2. **API Key Management:**
   * **Issue:** Exposing API keys in the codebase posed a security risk.
   * **Solution:** Environment variables were used to securely manage API keys, ensuring they were not hardcoded or exposed in the repository. For projects, the OpenAI API key should be stored securely and accessed via environment variables to prevent unauthorized access and maintain security.

**Lessons Learned:**

* **Version Control Best Practices:** Ensure all directories are properly tracked and avoid submodule configurations unless intentional.
* **Visibility of Hidden Files:** Regularly check for hidden files to prevent unexpected behavior in version control systems.
* **Deployment Configuration:** Carefully structure build and start commands to match the application's architecture and dependencies.
* **Security Considerations:** Use environment variables and input validation to protect sensitive information and maintain application integrity.

**Step-by-Step Process for Adding Features**

**1. Planning and Design:**

* **Identify the Feature:** Clearly define the new feature you want to add, including its purpose and expected outcome.
* **Design the Solution:** Plan how the feature will integrate with the existing architecture. Consider any necessary changes to the backend, frontend, or database schema.

**2. Set Up a Development Environment:**

* **Clone the Repository:** Ensure you have the latest version of the codebase by cloning the GitHub repository to your local machine.
* **Create a New Branch:** Use Git to create a new branch for the feature development. This isolates your changes and prevents conflicts with the main codebase.

bash

CopyInsert in Terminal

git checkout -b feature/your-feature-name

**3. Implement the Feature:**

* **Backend Changes:** If applicable, modify the backend code to support the new feature. This may involve adding new endpoints, modifying existing logic, or updating the database schema.
* **Frontend Changes:** Update the frontend to include any new UI components or interactions required for the feature.
* **API Integration:** If the feature involves external APIs, ensure proper integration and error handling.

**4. Testing:**

* **Unit Testing:** Write unit tests for any new functions or modules to ensure they work as expected.
* **Integration Testing:** Test how the new feature interacts with the existing system. This includes testing API requests and responses.
* **User Acceptance Testing:** Verify that the feature meets the requirements and provides a good user experience.

**5. Code Review and Refinement:**

* **Peer Review:** Have another developer review your code for quality, readability, and potential issues.
* **Refactor if Necessary:** Make any necessary changes based on feedback to improve code quality and maintainability.

**6. Commit and Push Changes:**

* **Commit Changes:** Once the feature is complete and tested, commit your changes with a descriptive message.

bash

CopyInsert

git add .

git commit -m "feat: add your-feature-name"

* **Push to GitHub:** Push your branch to the GitHub repository.

bash

CopyInsert in Terminal

git push origin feature/your-feature-name

**7. Create a Pull Request:**

* **Open a Pull Request:** Navigate to your GitHub repository and create a pull request to merge your feature branch into the main branch.
* **Review and Merge:** Once the pull request is approved, merge it into the main branch.

**8. Deployment:**

* **Prepare for Deployment:** Ensure all environment variables and configurations are set up for the new feature.
* **Deploy to Render:** Trigger a deployment on Render.com to update the live application with the new feature.
* **Monitor Deployment:** Check the deployment logs and application functionality to ensure everything works as expected.

**9. Post-Deployment Testing:**

* **Verify Functionality:** Test the deployed application to ensure the new feature is working correctly in the production environment.
* **Monitor for Issues:** Keep an eye on user feedback and application performance to quickly address any issues that arise.

Examples:   
Beste mevrouw A C G Willemsen - Verschuren,

Op 24-08-2024 hebben wij de volgende klacht ontvangen:

Wij vinden het vervelend dat er een reden was voor u om een klacht in te dienen. Daar wij onze dienstverlening continu proberen te verbeteren danken wij u voor uw melding.

Wij hebben van u een klachtmelding ontvangen waarin u aangeeft dat er een boekingsfout is gemaakt voor uw ritreservering van 24-08-2024. U heeft hierdoor geen gebruik kunnen maken van het vervoer.

Voor deze klachtmelding hebben wij de ritgegevens gecontroleerd en het gesprek nageluisterd wat met u gevoerd is op 24-08-2024. Op 24-08-2024 om 10:49 uur neemt u contact op met de reserveringslijn van VPflexvervoer.

In dit gesprek geeft u aan voor dezelfde avond vervoer te willen

Voor de heenrit geeft u aan te willen vertrekken vanuit uw huisadres in Hellevoetsluis en gebracht wil worden naar Hoogvliet met een gewenste ophaaltijd om 19:00 uur. Voor de retourrit geeft u ophaaltijd 22:00 uur door.

De medewerkster neemt de ritgegevens op voor beide ritten en maakt hierbij een fout door de ritten te reserveren op datum 25-08-2024 in plaats van 24-08-2024 zoals u had opgegeven.

Op 24-08-2024 omstreeks 19:17 uur neemt u contact op met de reserveringslijn van VPflexvervoer en geeft u aan dat u nog niet bent opgehaald.

In dit gesprek wordt de fout ontdekt en neemt de medewerkster meteen contact op met de vervoerder om te kijken of u als alsnog zo snel mogelijk opgehaald kan worden.

Helaas heeft de vervoerder geen ruimte meer om uw ritten in te plannen.

Hierdoor heeft u geen gebruik kunnen maken van het vervoer. De foutieve ritten zijn meteen op uw verzoek geannuleerd.

De betreffende medewerkster die de boekingsfout heeft gemaakt voor uw ritten hebben wij inmiddels aangesproken en zij biedt haar welgemeende excuses aan voor de gemaakte fout en dat u hierdoor niet naar het feest kon.

De medewerkster zullen wij de komende periode extra controleren op haar reserveringen zodat herhaling voorkomen kan worden. Wij betreuren dit voorval en de hinder die u hierdoor heeft ondervonden.

Klacht:

Ik ben veel te laat opgehaald en daardoor mijn bezoek gemist.

Ik heb 1uur gewacht op de taxi en die kwam maar niet. Mijn pasnummer is 123456

**Important:**

**MongoDB cluser** - Complaint handling in Mon

User: ejsdebakker

Pass: bip5LZqjqKxYkO9d

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**OPENAI\_API\_KEY**=sk-proj-qNZ6EpmMzBJ9rmgVu2J-faCd04nNZBoOsvKi9\_hSwdMhiWAyAC3belDfTt0IVlvx7sJj\_RYvh8T3BlbkFJYg6WK2wWi\_fZv55Ig3sF-0cjbDlk9ZI5FDCOxxzwW4Anc8N3Ti1uFYz3zIUa7R0V7qNXHbimEA