Flow of the App:-

1. **File Upload & Display**: The user uploads a text file, which is displayed in the frontend.
2. **Automatic Annotation**: The backend processes the text using SpaCy, returning annotated text (NER, coreference resolution) to the frontend.
3. **Visualization**: The frontend highlights automatically annotated entities, allowing users to see the annotations.
4. **Manual Annotation**: The user can click on text to manually annotate or adjust the existing annotations.
5. **Real-Time Updates**: Changes are reflected immediately in the text visualization.
6. **Download**: After finalizing the annotations, the user can download the annotated data file.
7. **Training Model**: The manually corrected annotations are used to improve the automatic annotation model over time.

**More detailed Scope of the Project:-  
  
1. File Upload and Text Display (Frontend - HTML, CSS, JavaScript)**

* **Task**: When a user uploads a text file, the entire content of the file should be displayed on the frontend in an editable format, allowing the user to view the text before any processing happens.
* **Technology**: Use an <input type="file"> element to upload the text, and JavaScript to read the file contents and display the text dynamically on the webpage.
* **Visualization**: Use a content-editable <div> or text area to show the file’s content. This will allow easy visualization of the text and future interactive annotations.

**2. Automatic Annotation Process (Backend - Express.js & NLP - Python, SpaCy)**

* **Task**: Once the file is uploaded and displayed, automatically trigger the annotation process in the background using the trained SpaCy model for Named Entity Recognition (NER) and coreference resolution.
* **SpaCy Model**: Initially, you’ll need to train your SpaCy model manually using Assamese data for NER and coreference resolution. Once the model is trained, it will automatically handle these tasks.
* **Backend Handling**: After the text is displayed, send it to the backend, where the SpaCy NLP engine will process the text and return the annotated words/entities (e.g., named entities and coreferences).
* **Visualization**: Return the annotated data to the frontend and highlight or mark the recognized entities within the text (e.g., highlight named entities in different colors).

**3. Interactive Visualization (Frontend - HTML, CSS, JavaScript)**

* **Task**: Allow users to visually see the annotations. For example, if the model annotates a named entity, highlight the word in the text with a different color.
* **Technology**: Use CSS for highlighting different annotations (e.g., named entities in green, coreferences in blue). You can dynamically apply these styles using JavaScript to annotate the text in real-time as the backend processes it.
* **Manual Interaction**: Allow users to click on words or phrases to manually annotate them. Create a simple UI where users can select a type of annotation (e.g., "Named Entity" or "Coreference") and apply it manually if needed.

**4. Manual Annotation (Frontend - HTML, CSS, JavaScript)**

* **Task**: Users should be able to manually annotate any word or phrase in the text if the automatic process misses something or if they want to override an existing annotation.
* **Interactive Text**: Implement a feature where users can highlight text or click on a word to choose an annotation label (e.g., select "Person," "Location," or "Coreference" from a dropdown).
* **Annotation Tracking**: Use JavaScript to track manual annotations and update the text visualization accordingly.

**5. Backend Annotation Management (Backend - Express.js & NLP - Python, SpaCy)**

* **Task**: Send the manually updated annotations back to the backend for integration into the overall dataset. These manual annotations can also be used to improve the training of your SpaCy model.
* **Data Structure**: Store both the automatic and manual annotations in a structured format (e.g., JSON), which includes the text, annotation type, and positions of the annotated entities.
* **Model Training**: Use the manually corrected data to retrain or fine-tune your SpaCy model, allowing it to improve over time and better handle Assamese text in the future.

**6. Real-Time Annotation Updates (Frontend - HTML, CSS, JavaScript)**

* **Task**: As the user makes manual annotations or adjustments, instantly reflect those changes in the text visualization, highlighting the newly annotated entities.
* **Technology**: Use JavaScript event listeners to track user interactions and dynamically update the text display in real-time. For example, if a user selects a word and tags it as a "Person," the corresponding color should update immediately.

**7. Finalized Annotation and Download (Frontend - HTML, CSS & Backend - Express.js)**

* **Task**: Once the user is satisfied with the annotations (both automatic and manual), allow them to download the fully annotated text file.
* **File Generation**: In the backend, generate a file (e.g., .txt, .json, .csv) that includes the original text along with the annotations. The annotations could be embedded in the text (e.g., tags around entities) or stored separately in a structured format.
* **Download Feature**: Add a ‘Download Annotated File’ button on the frontend. When clicked, the annotated data should be sent to the backend, where it’s compiled into a downloadable file, then returned to the frontend for the user to save.

**8. Continuous Improvement (Training the Model for Automatic Annotation)**

* **Task**: Initially, the model might not perform well on Assamese text for NER and coreference resolution. You'll need to manually annotate some text and use that data to train the SpaCy model.
* **Model Training**: Once you have enough annotated data, use SpaCy’s training pipeline to train a custom model for Assamese text. This model will then automatically annotate text in future interactions.
* **Model Update**: Periodically retrain the model with new manual annotations to improve its performance.

**Difficulty:-**

**1. File Upload & Display (Frontend - HTML, CSS, JavaScript)**

* **Difficulty**: Easy
* **Overview**: Standard file input and text display. You'll pass easily using basic HTML, CSS, and JavaScript.

**2. Automatic Annotation Process (Backend - Express.js & NLP - Python, SpaCy)**

* **Difficulty**: Medium
* **Overview**: Requires integrating Python with Node.js for automatic text annotation using SpaCy. Handling async processing could cause some struggles, but manageable.

**3. Interactive Visualization (Frontend - HTML, CSS, JavaScript)**

* **Difficulty**: Medium
* **Overview**: Applying CSS highlights dynamically to annotated text. Handling complex layouts or large text may pose some challenges but generally feasible with JavaScript.

**4. Manual Annotation (Frontend - HTML, CSS, JavaScript)**

* **Difficulty**: Medium to Hard
* **Overview**: Developing a UI for selecting text and applying manual annotations could be tricky due to complex text interactions and event handling.

**5. Backend Annotation Management (Backend - Express.js & NLP - Python, SpaCy)**

* **Difficulty**: Medium
* **Overview**: Handling data synchronization between manual and automatic annotations, ensuring accuracy in tracking text changes could lead to minor struggles.

**6. Real-Time Annotation Updates (Frontend - HTML, CSS, JavaScript)**

* **Difficulty**: Hard
* **Overview**: Updating text dynamically while keeping track of annotations in real-time can be quite challenging, especially with large text or overlapping annotations.

**7. Finalized Annotation and Download (Frontend - HTML, CSS, Backend - Express.js)**

* **Difficulty**: Medium
* **Overview**: Preparing annotated text for download as a file is straightforward but may get complex depending on the file format and annotation structure.

**8. Training the Model for Automatic Annotation (Python, SpaCy)**

* **Difficulty**: Hard
* **Overview**: Manually annotating data for training and fine-tuning the SpaCy model for Assamese NER and coreference is the hardest part. Expect struggles due to lack of pre-trained models and initial training data.

**Step by Step Approach:-**

**1. Set Up the Project Structure**

* **Task**: Start by setting up your project structure, separating frontend and backend code.
* **Tech Stack**: HTML, CSS, JavaScript (Frontend) | Express.js (Backend)
* **Steps**:
  1. Create a folder structure for the project (e.g., frontend/, backend/).
  2. Initialize a Node.js project in the backend folder with npm init.
  3. Install necessary backend dependencies like express, multer for file uploads.

**2. Build the Frontend (HTML, CSS, JavaScript)**

* **Task**: Build the frontend user interface.
* **Steps**:
  1. Create a file upload form where users can upload Assamese text files.
  2. Display the uploaded text on the screen using JavaScript once the user selects the file.
  3. Ensure the UI is responsive and easy to interact with (using CSS).
* **Goal**: Ensure you can upload a file and see its content displayed on the webpage.

**3. Set Up File Upload Backend (Express.js)**

* **Task**: Create the API in Express.js to handle file uploads from the frontend.
* **Steps**:
  1. Set up an API endpoint in Express.js that accepts text files using multer.
  2. Store the uploaded file temporarily and read its contents on the backend.
* **Goal**: Ensure files uploaded from the frontend are received correctly by the backend.

**4. Display Text in Frontend After Upload**

* **Task**: Once the file is uploaded, display its content in the frontend using real-time JavaScript.
* **Steps**:
  1. Send the file content back to the frontend.
  2. Dynamically show the text in an editable format (e.g., <div contenteditable> or <textarea>).
* **Goal**: Uploaded text should be visible and editable on the frontend.

**5. Learn and Train the SpaCy Model (Python)**

* **Task**: Train an NER model in SpaCy for Assamese text (this is a challenging task if you’re new to model training).
* **Steps**:
  1. **Learn**: Familiarize yourself with SpaCy’s documentation, especially the NER pipeline. Start with small examples of training SpaCy models.
  2. **Training Data**: Since SpaCy doesn’t have pre-trained models for Assamese, you’ll need annotated Assamese text for NER. Start manually annotating a small dataset using SpaCy’s annotation tools.
  3. **Training**: Train a basic SpaCy model for NER using your manually annotated data.
* **Goal**: Train a simple model for basic Named Entity Recognition (NER) tasks in Assamese.

**6. Automatic Annotation Backend Integration**

* **Task**: Connect your trained SpaCy model with the backend.
* **Steps**:
  1. Use Node.js’s child\_process module to call your Python script that processes text with SpaCy.
  2. Send the uploaded text from the frontend to the backend, run the NLP model, and return the annotated data (e.g., tagged entities).
* **Goal**: Automatically process text with SpaCy and get the annotated entities back to the backend.

**7. Annotated Text Visualization (Frontend - HTML, CSS, JavaScript)**

* **Task**: Highlight the annotated entities in the frontend.
* **Steps**:
  1. Once the annotated text is received from the backend, dynamically highlight the named entities in the text using JavaScript and CSS (e.g., different colors for people, locations, organizations).
  2. Ensure that the user can clearly see which parts of the text have been annotated.
* **Goal**: Display annotations visually with proper highlighting on the frontend.

**8. Add Manual Annotation Feature (Frontend - JavaScript)**

* **Task**: Enable users to manually annotate text.
* **Steps**:
  1. Implement a feature where users can select text and manually apply annotation types (e.g., dropdowns for selecting "Person", "Location", etc.).
  2. Capture these manual annotations and store them in a structured format (e.g., JSON) to send back to the backend.
* **Goal**: Allow users to click on words and manually apply annotations if needed.

**9. Sync Manual Annotations with Backend (Express.js)**

* **Task**: Handle the manual annotations in the backend.
* **Steps**:
  1. Once the user completes manual annotations, send the final annotated text back to the backend.
  2. Combine the automatic and manual annotations to create a final version of the annotated text.
* **Goal**: Merge manual and automatic annotations on the backend.

**10. Download Annotated File (Frontend & Backend)**

* **Task**: Implement the feature to download the fully annotated text file.
* **Steps**:
  1. On the backend, format the annotated text into a downloadable file (e.g., .txt, .json, .csv).
  2. Provide a download link/button on the frontend for the user to save the file.
* **Goal**: Ensure users can download the final annotated text.

**11. Continuous Training and Model Improvement (Python, SpaCy)**

* **Task**: Improve your SpaCy model over time with new data.
* **Steps**:
  1. Use the manual annotations from the users to retrain your SpaCy model periodically.
  2. Gather more training data and improve the accuracy of NER and coreference resolution for Assamese.
* **Goal**: Refine the model for better automatic annotations in future versions.

**Main focus during developing:-**

* **Start Simple**: Focus on getting the basic file upload and text display working (Steps 1-4).
* **Train the Model**: Invest time learning and training your SpaCy model (Step 5). This is the most challenging part.
* **Backend & Annotation Integration**: Connect your backend with the trained model and display the annotations (Steps 6-7).
* **Manual Features & Refinements**: Add manual annotation and real-time visual feedback (Steps 8-9).
* **Finalize and Improve**: Allow users to download their work and improve your model (Steps 10-11).

**Schedule:-**

By Sep 30: Frontend and Backend Setup  
By Oct 8: Automatic Annotation (Train SpaCy Model)  
By Oct 15: Backend Integration with SpaCy  
Oct 30: Frontend Visualization of Annotations  
By Nov 5: Manual Annotation Feature  
By Nov 12: Sync Manual Annotations with Backend  
By Nov 19: Finalize Download Feature  
BY Nov 30: Testing, Bug Fixes, and Model Improvement