Project Overview:

The Effort Estimation Tool aims to provide users with accurate estimates for various tasks based on historical data stored in a MongoDB database. The tool will allow users to login, submit details of estimation required, and utilize a simple formula to aggregate historical data and predict new estimates.

User Stories:

1. User Registration and Authentication:
    - As a user, I want to register for an account so that I can access the Effort Estimation Tool.
    - As a registered user, I want to log in to my account securely.
    - As a user, I want the system to authenticate my credentials before allowing access to the tool.
2. Estimation Submission:
    - As a user, I want to input the details of the task for which I need an estimation.
    - As a user, I want to specify parameters such as complexity, size, and type of task.
    - As a user, I want the option to add additional notes or specifications for the estimation.
3. Estimation Calculation:
    - As a user, I want the system to calculate the estimated effort based on historical data.
    - As a user, I want the estimation to be displayed in a clear and understandable format.
    - As a user, I want the system to provide a confidence level or range for the estimation.
4. Database Interaction:
    - As a user, I want the tool to fetch historical data from the MongoDB database.
    - As a user, I want the system to update the historical data with new estimations for future reference.
5. UI/UX Design:
    - As a user, I want the interface to be intuitive and easy to navigate.
    - As a user, I want the tool to have a responsive design for use on different devices.
    - As a user, I want the interface to provide feedback on successful actions and errors.
6. Testing:
    - As a developer, I want to write unit tests for the backend using pytest.

- As a developer, I want to ensure that the frontend components are thoroughly tested for functionality and responsiveness.

Project Components:

1. Front-End:
    - Develop a user-friendly interface using HTML, CSS, and JavaScript (ES6/TypeScript).
    - Implement AJAX for asynchronous communication with the backend.
    - Utilize Jinja2 for server-side templating to render dynamic content.
2. Back-End:
    - Build the backend using Python with Flask framework.
    - Implement user authentication and authorization.
    - Connect to MongoDB database to fetch and update historical data.
    - Develop APIs to handle estimation submissions and calculations.
3. Database Management:
    - Design the MongoDB database schema to store historical estimation data.
    - Implement CRUD operations for interacting with the database.
    - Ensure data integrity and security measures are in place.
4. Deployment:
    - Containerize the application using Docker for easy deployment and scalability.
    - Set up Continuous Integration/Continuous Deployment (CI/CD) pipelines for automated testing and deployment.
5. Testing:
    - Write comprehensive unit tests for both frontend and backend components using pytest.
    - Conduct integration tests to ensure seamless interaction between different parts of the application.
    - Perform usability testing to gather feedback on the user interface and experience.

Technical Steps Of Implementation

Project Plan: Estimation Tool with Flask and MongoDB

1. User Registration and Authentication

Technical Steps:

- Implement a user registration form using HTML and CSS.
- Utilize the Flask-User or Flask-Security extension to manage user authentication within the Flask backend.
- Securely store user credentials in the MongoDB database.
- Develop robust login functionality, incorporating comprehensive error handling.
- Employ JSON Web Tokens (JWT) for secure authentication token generation and validation.

2. Estimation Submission

Technical Steps:

- Design an estimation submission form with fields for task details, utilizing HTML and CSS.
- Utilize AJAX to transmit form data asynchronously to the Flask backend.
- Implement RESTful API endpoints in Flask to efficiently handle estimation submissions.
- Validate and sanitize input data rigorously to prevent injection attacks.
- Store estimation details reliably in the MongoDB database.

3. Estimation Calculation

Technical Steps:

- Retrieve historical estimation data from the MongoDB database.
- Analyze this historical data to identify patterns and trends.
- Develop a streamlined formula or algorithm to calculate new estimates based on historical data and user-provided input parameters.
- Implement the estimation calculation logic within the Flask backend.
- Return the calculated estimate, along with a confidence level/range, to the frontend for clear display.

4. Database Interaction

Technical Steps:

- Establish a connection to the MongoDB database using the Flask-PyMongo extension.
- Define database models/schema to effectively store historical estimation data.
- Implement comprehensive CRUD (Create, Read, Update, Delete) operations for seamless interaction with the database.
- Ensure robust error handling and data validation to maintain data integrity.
- Consistently update historical data with new estimations after each submission.

5. UI/UX Design

Technical Steps:

- Design responsive UI components, leveraging HTML/CSS frameworks like Bootstrap or Tailwind CSS.
- Employ JavaScript (ES6/TypeScript) to enhance the frontend with interactivity and dynamic content.
- Implement client-side form validation for an improved user experience.
- Utilize AJAX for smooth data retrieval and updating, eliminating the need for full page reloads.
- Thoroughly test the UI across various devices and screen sizes to guarantee optimal responsiveness.

6. Testing

Technical Steps:

- Write thorough unit tests for backend APIs using pytest.
- Mock database interactions in unit tests to isolate components effectively.
- Implement integration tests to verify the seamless interaction between frontend and backend components.
- Utilize testing libraries like Selenium or Cypress for comprehensive end-to-end testing of UI interactions.
- Automate the testing process and integrate it with a CI/CD pipeline for continuous validation.

7. Deployment Using CI/CD (Jenkins and Docker)

Technical Steps:

- Set up a Jenkins server to facilitate continuous integration and deployment.
- Configure Jenkins jobs to pull source code from your chosen version control system (e.g., GitHub).
- Employ Docker to containerize both the Flask application and the MongoDB database.
- Create Dockerfiles to define the application and database images meticulously.
- Configure the Jenkins pipeline to build Docker images and push them to a Docker registry.
- Implement automated tests within the Jenkins pipeline to validate the application rigorously.
- Deploy the Docker images to a container orchestration platform like Kubernetes or Docker Swarm to achieve scalability and high availability

Estimation Tool Project Submission Criteria

To ensure the originality and quality of your work, please adhere to the following submission guidelines:

1. Submission Components

- **Presentation Deck:** A clear and concise presentation (e.g., PowerPoint, Google Slides) summarizing your project approach, implementation details, key findings, and results.
- **Solution Code:** Well-structured and commented source code for the entire application (Flask backend, frontend code, database scripts).
- **Screenshots of Output:** Visual evidence of the application's functionality, including user registration/login, estimation submission, calculation results, and any error handling.
- **README File:** A comprehensive document outlining installation instructions, dependencies, how to run the application, and any additional notes.

2. Code Originality

- All code must be **original work**. Plagiarism is strictly prohibited.
- If you have used external libraries or code snippets, clearly **cite the sources** and explain their purpose in your code comments.
- We will utilize plagiarism detection tools to verify the originality of your code.

3. Presentation and Documentation

- Your presentation deck should be well-organized, visually appealing, and convey the key aspects of your project effectively.
- Code comments should be clear, concise, and explain the logic behind your implementation.
- The README file should provide all necessary instructions for others to easily set up and run your application.

4. Functionality and Completeness

- Your application should implement all the required features outlined in the project specifications: user registration/authentication, estimation submission, calculation, and database interaction.
- Ensure that your application functions correctly, handles errors gracefully, and provides a smooth user experience.

5. Evaluation Criteria

Your submission will be evaluated based on the following criteria:

- **Originality and Plagiarism Prevention:**
  - Absence of plagiarized code
  - Proper citation of external sources
- **Functionality:**
  - Correct implementation of all required features
  - Error handling and user experience
- **Code Quality:**
  - Clean, well-structured code
  - Thorough and informative code comments
- **Presentation and Documentation:**

- Clear and concise presentation deck
- Informative README file

Important Notes:

- Any instance of plagiarism will result in disqualification.
- We encourage you to use version control (e.g., Git) to track changes in your code.
- Submit your project components in a single compressed file (e.g., ZIP).

Category	Technical Step	Evaluation Criteria	Points
User Registration (15 Points)	Implement registration form (HTML/CSS)	Form elements, styling, user-friendliness, responsiveness, adherence to best practices	5
	Flask-User/Flask-Security integration	Correct setup, secure password hashing, appropriate configuration, error handling	5
	MongoDB user data storage	Proper schema design, secure storage of credentials, data validation	5
Estimation Submission (20 Points)	Design submission form (HTML/CSS)	Form elements, layout, clarity of instructions, responsive design	5
	AJAX data submission	Asynchronous communication, error handling, data validation	5
	RESTful API endpoints (Flask)	Correct HTTP verbs, proper routing, appropriate status codes, input validation, data sanitization	5
	MongoDB estimation data storage	Appropriate schema design, data validation, efficient storage of submitted estimations	5
Estimation Calculation (20 Points)	Historical data retrieval (MongoDB)	Correct query construction, efficient data retrieval, error handling	4
	Historical data analysis	Identification of patterns/trends, statistical analysis, use of appropriate algorithms	8
	Estimation algorithm/formula development	Clear logic, accuracy of calculations, consideration of relevant factors, flexibility to accommodate different input parameters	8
Database Interaction (15 Points)	Flask-PyMongo setup	Proper connection, error handling	3
	Database schema design	Clear and efficient schema, data relationships, adherence to best practices	5
	CRUD operations implementation	Create, read, update, and delete operations, data validation, error handling	7
UI/UX Design (10 Points)	Responsive UI with HTML/CSS framework	Use of Bootstrap/Tailwind CSS, responsive design, adherence to design principles	4

	JavaScript interactivity (ES6/TypeScript)	Dynamic content, client-side validation, user-friendly interactions	3
	AJAX for smooth updates	Asynchronous data updates, minimal page reloads, good user experience	3
Testing (10 Points)	Unit testing (pytest)	Backend API testing, mocking database interactions, test coverage	5
	Integration/End-to-end testing (Selenium/Cypress)	Testing of frontend-backend interactions, UI interactions, overall application flow	5
Deployment (10 Points)	Jenkins setup and configuration	Correct Jenkins server setup, job configuration, integration with version control (e.g., GitHub)	3
	Dockerization (Flask app and MongoDB)	Creation of Dockerfiles, successful build and push of Docker images to a registry	3
	Deployment to Kubernetes/Docker Swarm	Successful deployment, service availability, scalability, load balancing, application monitoring	4