**Conduct initial testing to ensure data integration is successful and user interactions are smooth**

**Data Integration**

**Backend Setup**: Using a backend framework (Express.js with Node.js), creating the RESTful APIs to handle data transmission between the web application and MongoDB.

**Install Dependencies**: Using the npm to install necessary packages like mongoose, express, etc.

**API Development:** Developing the endpoints for data upload, retrieval, update, and deletion.

**Database Connection:** Useing Mongoose (for Node.js) to connect your application with MongoDB.

**Define Models:** Based on the database schema that we have define Mongoose models to interact with the MongoDB database.

**Data Upload Feature:** Implementing a feature in the web application to input data to be stored in MongoDB.

**Form Handling:** Createing forms in the frontend for data input.

**API Consumption**: Using AJAX, Fetch API to send data from the web application to your backend.

Reflect the Changes in Graphs

**Retrieve Data:** Implementing API endpoints to fetch the data required for the graphs.

**Graph Generation**: Utilizing the javaScript library like Chart.js, D3.js libraries to generate graphs based on the data retrieved from MongoDB.

**Dynamic Update:** Ensure the graph generation logic is dynamic, allowing for real-time updates when new data is added or existing data is modified.

**Testing**

**Functionality Testing:** Testing all of the features related to data input, retrieval, and graph updating to ensure they work as expected.

**Integration Testing:** Focusing on the integration points, especially how the web application communicates with MongoDB and how data changes reflect in graphs.

**User Interaction Testing:** Pay attention to the user interface and experience. Ensure that data can be added smoothly and graph updates are intuitive and clear to the user.