MongoDB Integration Plan for Student Prediction Project

Shahzad Rizwan and Hammad Nawaz

May 17, 2025

1 Introduction

We are working on a project called "Design and Development of a Hybrid Deep Learning Model for Predicting Student Academic Outcomes Using Open University Dataset." The project uses a web app built with Flask to predict if a student will pass, fail, withdraw, or get a distinction based on inputs like study credits and quiz attempts. Now, we need to add MongoDB, a database, to store information like predictions, how long users take to fill the form, their location, and more. This plan explains how we will set up MongoDB with five tables (called collections in MongoDB) and how we will connect it to our project.

2 Why MongoDB?

MongoDB is a simple database that stores data in a flexible way, like JSON files. It's good for our project because:

- It can handle different types of data, like numbers, text, and lists.
- It's easy to set up and connect to our Flask app.
- It lets us link data across tables using a unique ID.

We will create five collections to store different pieces of information when a user clicks the "Predict" button on our prediction page.

3 MongoDB Collections (Tables)

We will create a database called `student_prediction_db` with five collections. Each collection is like a table that holds specific data. Below, we describe each one, what it stores, and an example in simple words.

3.1 Collection 1: Predictions

- **Purpose**: Saves the prediction result when a user clicks "Predict" (e.g., Pass, Fail).

- **Data Stored**:
  - `session_id`: A unique ID to link this prediction to other data (like a ticket number).
  - `prediction`: The result (e.g., "Pass" or "Fail").
  - `confidence`: How sure the model is (e.g., 0.92 means 92% sure).
  - `input_features`: All user inputs (e.g., study credits, quiz attempts).
  - `timestamp`: When the prediction was made (e.g., May 17, 2025, 7:52 PM).
- **Example**: If a user enters their study credits and gets a prediction of "Pass" with 92% confidence, we save:
  - Session ID: "session_12345"
  - Prediction: "Pass"
  - Confidence: 0.92
  - Inputs: {study credits: 60, quiz attempts: 5, etc.}
  - Time: "2025-05-17 19:52"

3.2 Collection 2: Form Interactions

- **Purpose**: Tracks how long a user took to fill the prediction form and if they made errors.
- **Data Stored**:
  - `session_id`: Links to the same session as the prediction.
  - `form_fill_time`: Time in seconds to fill the form (e.g., 45 seconds).
  - `submission_timestamp`: When the form was submitted.
  - `form_errors`: Any mistakes (e.g., "Study credits must be a number").
- **Example**: If a user takes 45 seconds to fill the form with no errors, we save:
  - Session ID: "session_12345"
  - Form Fill Time: 45 seconds
  - Submission Time: "2025-05-17 19:52"
  - Errors: []

3.3 Collection 3: User Locations

- **Purpose**: Saves where the user is based on their internet address (IP).
- **Data Stored**:
  - `session_id`: Links to the session.
  - `ip_address`: User's internet address (e.g., 203.0.113.1).

  - `country`: Country (e.g., Pakistan).
  - `city`: City (e.g., Lahore).
  - `latitude`: Map coordinate (e.g., 31.5497).
  - `longitude`: Map coordinate (e.g., 74.3436).
  - `timestamp`: When we got the location.
- **Example**: For a user in Lahore, we save:
  - Session ID: "session_12345"
  - IP Address: "203.0.113.1"
  - Country: "Pakistan"
  - City: "Lahore"
  - Latitude: 31.5497
  - Longitude: 74.3436
  - Time: "2025-05-17 19:52"

3.4 Collection 4: Sessions

- **Purpose**: Keeps track of each user's visit to our app.
- **Data Stored**:
  - `session_id`: Unique ID for the visit.
  - `start_timestamp`: When the user started using the app.
  - `user_agent`: Browser info (e.g., Chrome on Windows).
- **Example**: For a user visiting our app, we save:
  - Session ID: "session_12345"
  - Start Time: "2025-05-17 19:50"
  - Browser: "Chrome on Windows"

3.5 Collection 5: User Inputs

- **Purpose**: Saves the user's input data for later analysis (e.g., to improve our model).
- **Data Stored**:
  - `session_id`: Links to the session.
  - `input_features`: All user inputs (same as in predictions).
  - `timestamp`: When the inputs were saved.
- **Example**: For a user's inputs, we save:
  - Session ID: "session_12345"
  - Inputs: {study credits: 60, quiz attempts: 5, etc.}
  - Time: "2025-05-17 19:52"

4 How Collections Are Connected

We use the `session_id` to link all five collections. For example:

- When a user clicks "Predict," we create a unique `session_id` (like "session_12345").
- We save the prediction, form time, location, session, and inputs with the same `session_id`.
- This lets us find all data for one prediction by searching for `session_id`.

It's a simple connection: one session links to one record in each collection.

5 How We Will Add MongoDB to Our Project

Here's our step-by-step plan to add MongoDB to our Flask app:

5.1 Step 1: Set Up MongoDB

- We will use MongoDB Atlas, a free online service, to avoid installing anything on our computer.
- Sign up at mongodb.com, create a database called `student_prediction_db`, and get a connection link (like a web address for the database).
- Create the five collections: `predictions`, `form_interactions`, `user_locations`, `sessions`, and `user_inputs`.

5.2 Step 2: Update Our Flask App

- In our main app file (app . py), we will add a connection to MongoDB using the connection link from Atlas.
- When a user visits the prediction page and clicks "Predict," we will:
  - Create a unique `session_id` for the user's visit.
  - Save the session details (like browser and time) in the `sessions` collection.
  - Save the user's inputs (like study credits) in the `user_inputs` collection.
  - Get the user's location from their internet address using a free service (like ipapi.co) and save it in `user_locations`.
  - Save how long they took to fill the form and any errors in `form_interactions`.
  - Save the prediction result (like "Pass" with 92% confidence) in `predictions`.
- All these will use the same `session_id` to keep them connected.

5.3 Step 3: Update the Prediction Form

- In our JavaScript file (`scripts.js`), we will add a timer to measure how long a user takes to fill the prediction form.
- When the user clicks "Predict," we will send the time and any form errors (like wrong numbers) to the app along with the form data.
- This way, we can save the time in the `form_interactions` collection.

5.4 Step 4: Test Everything

- Run our Flask app and go to the prediction page.
- Fill the form, click "Predict," and check if:
  - A new tab opens with the prediction result (like before).
  - All five collections in MongoDB have new records with the same `session_id`.
  - The form fill time looks correct (e.g., 45 seconds).
  - The location shows the right city and country (or "Unknown" if the service fails).
- If something doesn't work, check the app logs to find the problem (e.g., MongoDB not connecting).

5.5 Step 5: Keep It Safe

- Store the MongoDB connection link in a separate file (like `.env`) so it's not in our code.
- Make sure only our app can access the database by setting up a password in MongoDB Atlas.

6 What We Need to Do

- **Muhammad Sadiq**: Set up MongoDB Atlas, create the database and collections, and share the connection link with Hammad.
- **Hammad KHan**: Update `app.py` to connect to MongoDB and save data in all five collections when "Predict" is clicked.
- **Both**: Update `scripts.js` to add the form timer and test the prediction page together.
- **Dr. Chen Tee (Supervisor)**: Review our plan and suggest any changes.
- **Dr Shehzad Rizwan**: Check if we need extra data in the collections for analysis.

7 Conclusion

By adding MongoDB with five collections, we can save important data like predictions, form fill times, and user locations. This will help us track how users interact with our app and improve our model later. The `session_id` will keep everything connected simply. We will use MongoDB Atlas for easy setup, update our app to save data, and test it to make sure it works. This plan keeps our project organized and meets our goal of storing prediction data.