CSE 341 Final Project Proposal

General Info

Diogo Rangel Dos Santos

Paula Jessica Ferreira

Luis Eleazar Lizano Rojas

David Aniefioko Uda

Kaden Robert Brown

Iyen Samuel Evbosaru

Personal Record Keeper API

Contents:

Contents

Application Info

What will the API do?

The API will be a **simple RESTful service** for managing a list of **personal records** (like contacts, quick notes, or client lists). It must implement all **CRUD operations (Create, Read, Update, Delete)** for these records.

How will your API utilize a login system?

It will use a Session-based login system (cookies). The API will have a POST /user/login route to validate credentials and start a session, and a GET /user/logout route to end it.

What database will you use?

MongoDB

How will the data be stored in your database?

There will be two main, simple collections:

1. **Users:** Stores username and the **hashed password** (using bcrypt).
2. **Records:** Stores the record data (e.g., firstName, lastName, email), along with an **ownerId** to link the record to the authenticated user.

How would a frontend be able to manage authentication state based on the data you provide?

Upon a successful login (POST /user/login), the API will respond with an HTTP Status 200 OK and set a secure session cookie in the client's browser. The frontend will know the user is authenticated because subsequent requests to protected routes will automatically send this cookie and be accepted by the server.

What pieces of data in your app will need to be secured? How will you demonstrate web security principles in the development of this app?

User Passwords: Must be secured by hashing (bcrypt).

Personal Records Data: All CRUD routes for records must be protected by authentication and authorization (users can only access their own records based on the ownerId field).

1. Password Hashing (bcrypt) on registration and login.
2. Route Protection using authentication middleware to enforce login.
3. Authorization checks to ensure a user can only manipulate their own data.

4. Using Environment Variables for sensitive configuration data.

What file structure and program architecture will you use for this project (how will you organize your node project)? Why?

We will use the MVC (Model-View-Controller) architecture, adapted for an API:

- /controllers: Houses the primary CRUD and API logic.
- /routes: Defines all the URL endpoints.
- /models: Contains the MongoDB schemas.

This structure is modular and keeps the logic organized, which is essential for team development and long-term readability.

What are potential stretch challenges that you could implement to go above and beyond?

1. Implement simple filtering/searching on the GET /record route (e.g., searching by first name).
2. Add more robust data validation for email addresses and required fields.

API Endpoint Planning

For this section, you'll plan out what API endpoints you'll need for your project.

POST	/user/register Create a new user account.
POST	/user/login Start a session (authenticate).
GET	/user/logout End the session.
POST	/record CREATE: Add a new personal record.
GET	/record READ: Return all records for the logged-in user.
GET	/record/{id} READ: Return a single specific record.
PUT	/record/{id} UPDATE: Modify an existing record.
DELETE	/record/{id} DELETE: Remove a record.

Project Scheduling and Delegation

Plan out what tasks will get completed with each lesson remaining in the semester (Only edit highlighted text).

Week 04 Tasks	***Finalized and Approved*.**
Week 05 Tasks	● *Node.js Project Creation* ● ***MongoDB Setup*** ● ***API DOCUMENTATION** completed and available at route* ● */api-docs*
Week 06 Tasks	**Implement CRUD - READ and CREATE**: Focus on **GET /record** (all and single) and **POST /record**. Test data persistence. • Implement CRUD - UPDATE and DELETE. Implement • ***Login/Logout** and **Protect all CRUD routes** with authentication middleware.* • *Final Code Review* ***Testing**, and Preparation of the **Video***
Week 07 Tasks	• ***Send the Presentation and assignment** (Demonstrate all routes functioning and MongoDB data modification)*

How will you divide up work in your team to ensure the following tasks all get completed?

Role/Team Member	**Primary Focus**	**Tasks to be Completed**	
Team Member Diogo Rangel (Infrastructure & Documentation)	Project Setup & API Documentation	1. **Node.js project creation**. 2. **Create git repo** and share with group.	3. **MongoDB setup** and Model creation. 4. **API Swagger documentation** for all API routes
Team Member Diogo Rangel (Data/Records CRUD)	Core Application Functionality	1. Implement **HTTP GET** (all and single) routes for /record. 2. Implement **HTTP POST** (Create) route for /record.	3. Implement **HTTP PUT** (Update) route for /record. 4. Implement **HTTP DELETE** route for /record.

Team Member Diogo Rangel (Security & Finalization)	User Authentication & Presentation	1. Implement all User Authentication routes (/user/register, /user/login, /user/logout). 2. Implement authentication/authorization middleware to protect CRUD routes.	3. Finalize code and coordinate the the **Video presentation.**

Potential Risks and Risk Mitigation Techniques

What are the risks involved with you being able to finish this project in a timely manner?

The primary risks for a team completing a project quickly are focused on **scope, complexity, and coordination**:

1. **Complexity of Authentication:** Implementing a secure and functional login/session system often takes longer than anticipated.
2. **Unclear Endpoint Requirements:** Misunderstanding what data each API route needs, leading to rework after the initial coding.
3. **Merge Conflicts & Version Control Issues:** Having team members work on the same critical files simultaneously, causing delays when trying to combine code.
4. **Deployment Issues:** Unforeseen problems when pushing the working application to a hosting service like Heroku.

How will you mitigate or overcome these risks?

Risk	Mitigation Technique	
Complexity of Authentication	**Focus on Basics First:** Use a standard, simple session or JWT implementation (whichever is standard for the course). Complete the entire	**CRUD (Create, Read, Update, Delete)** functionality first, and then add authentication as a final layer
Unclear Endpoint Requirements	**Mandatory Swagger Documentation (Week 05):** Ensure the **API Swagger documentation is completed first** and verified by the team. This document serves as the "contract" for what data each route expects and returns, providing a clear blueprint for coding.	
Merge Conflicts & Version Control Issues	**Strict Delegation:** Enforce the work division based on **entities**	

	(e.g., Team B is the only one touching the /record controller files). Use **separate Git branches** for new features, and perform code reviews before merging into the main branch.	
Deployment Issues		**Early Deployment:** Team A should create the Git repository and ensure the project can be **pushed to Heroku** with a basic "Hello World" placeholder in Week 05. This verifies the hosting setup early, isolating deployment issues from coding bugs.