CSE 341 Final project Proposal

# General Info

Karim Valenzuela Gonzalez

Caitlin Evans

Personal Pet Tracker

Contents:

Contents

[General Info](#_gjdgxs) 1

[Application Info](#_oxlmg6u5hyfx) 2

[What will the API do?](#_1fob9te) 2

[How will your API utilize a login system?](#_tyjcwt) 2

[What database will you use?](#_2et92p0) 2

[How will the data be stored in your database?](#_2s8eyo1) 2

[How would a frontend be able to manage authentication state based on the data you provide?](#_3dy6vkm) 2

[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?](#_1t3h5sf) 3

[What file structure and program architecture will you use for this project (how will you organize your node project)? Why?](#_4d34og8) 3

[What are potential stretch challenges that you could implement to go above and beyond?](#_17dp8vu) 5

[**API Endpoint Planning**](#_mral43ww1pvt)5

[Project Scheduling and Delegation](#_1ksv4uv) 6

[How will you divide up work in your team to ensure the following tasks all get completed?](#_3rdcrjn) 6

[Potential Risks and Risk Mitigation Techniques](#_26in1rg) 7

[What are the risks involved with you being able to finish this project in a timely manner?](#_lnxbz9) 7

[How will you mitigate or overcome these risks?](#_35nkun2) 7

# Application Info

## What will the API do?

Our API is designed to help pet owners securely track important details about their animals, including general info, medical history, and upcoming appointments. Since many people have more than one pet—often across different species—it can be a lot to keep track of. We wanted something practical, scalable, and personal, especially in case of emergencies. Each user will only see their own pet data, and the system is flexible enough to work for households with one or multiple pets, across different needs and breeds.

## How will your API utilize a login system?

We’ll be using **Google OAuth 2.0** so each user can log in with their personal email. This keeps data secure and separated by user ID, ensuring that one owner can’t see another’s pets. It also allows us to use the app in our real lives without needing to manage manual logins.

## What database will you use?

We'll use **MongoDB Atlas** for our database, hosted in the cloud so both team members can collaborate without local conflicts. I think this will be easiest because that was the focus of the class this block.

## How will the data be stored in your database?

Each document in our collections will be tied to a userId from the authenticated user. We'll have at least four collections: Users, Pets, MedicalRecords, and AppointmentHistory. We’ll also make sure at least one collection includes 7 or more fields, though from trying to create basic outlines we may have more than one to meet this criteria.

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

Even though this project is focused on backend development, we’re planning to include a simple **dashboard** view for testing purposes.

The dashboard will:

* Display the current user's email and role after OAuth login
* Show different buttons based on user role (similar to our HP project)
* Provide buttons to:
  + View Pets (user-specific JSON)
  + View Medical Records (also filtered by userId)
  + View Appointments
  + Access Swagger API Docs
  + Logout

If a user tries to access data they’re not authorized to see, like another user’s pets or a restricted endpoint (like a petid or userid), we’ll return a 403 Forbidden response with a simple HTML page and a link to return to the dashboard.

This helps us test our **protected routes** and show how user-specific access works—even without a full frontend.

## 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?

We'll secure any data that is linked to a specific user—especially pet medical records and appointment information. Each pet is tied to a userId, so only the owner can access or modify their data.

We'll demonstrate security in a few ways:

* All CRUD routes for pets, medical records, and appointments will be protected by OAuth-based middleware.
* We'll use **type validation and express-validator** to prevent incorrect or unsafe data—like accidentally passing a boolean into a string field, or a number where a name should go.
* Sensitive values like our OAuth client secrets and MongoDB URI will be stored in a .env file so they aren’t committed to GitHub.
* For users who try to access data they shouldn't (e.g., a professor trying to access headmaster-only routes like in the HP dashboard), we’ll show a simple HTML message with a “Return to Dashboard” link to give clear feedback while keeping things secure and user-friendly.

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

This is hard to articulate outside of a .md file but I would base it similarly to week 3/ week 4 project we completed.

/config

└── auth.js // OAuth config

/controllers

└── petController.js

└── medicalController.js

└── appointmentController.js

/middleware

└── authMiddleware.js // Route protection by userId and role

/models

└── petModel.js

└── medicalModel.js

└── appointmentModel.js

└── userModel.js

/routes

└── authRoutes.js

└── petRoutes.js

└── medicalRoutes.js

└── appointmentRoutes.js

/swagger

└── swagger.json // API Docs

.env // Sensitive config that would not be published to GitHub (OAuth, DB URI, etc.)

server.js // Main entry point

.gitignore // ignotr the .env and any other sensitive file we do not want published to github

README.md // main document to users who want to

It is important to note that we’re using .env to safely store secrets like our Google OAuth keys and database credentials. The reason we like this layout is because it separates concerns really clearly and helps when we’re debugging or working asynchronously. It is important that we create a .gitignore so the .env and any other chosen sensitive file is not published to github.

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

* Creating a sign up / enrollment page possibly so if we shared the app with a friend or family they could create their own pet profile and add the data using the dashboard links or api etc.
* Allow users to export pet medical records as PDF
* Filter for upcoming vs. past appointments

# API Endpoint Planning

**I am unsure if we would need an endpoint for users since we want to keep that data separated and only accessed by the user themselves but here are the main endoints based on the other 3 collections**

**USERS**

* GET /users – list all pets owned by the logged-in user
* GET /users/:id
* POST /users
* PUT /users/:id
* DELETE /users/:id
* **Any ID would have to be mapped to check the current user ID so User A cannot GET / POST / PUT / DELETE data for an ID that not associated to them\*\***

**Pets**

* GET /pets – list all pets owned by the logged-in user
* GET /pets/:id
* POST /pets
* PUT /pets/:id
* DELETE /pets/:id

**MedicalRecords**

* GET /medical/:petId
* POST /medical/:petId
* PUT /medical/:recordId
* DELETE /medical/:recordId

**Appointments**

* GET /appointments/:petId
* POST /appointments/:petId
* PUT /appointments/:apptId
* DELETE /appointments/:apptId

# Project Scheduling and Delegation

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

Because we are working in separate timezones and working asynchronously this is a rough schedule we are still trying to figure out as well as divvying tasks but

|  |  |
| --- | --- |
| **Week** | ***Task*** |
| Week 04 | • *Finalize Project Proposal and submit* |
| Week 05 | • Set up GitHub repo  • Deploy starter code to Render  • Add Google OAuth login  • Swagger setup available at route /api-docs  • Create Mongoose models for User, Pet, MedicalRecord, and Appointment |
| Week 06 | *• Build CRUD routes for all collections  • Add middleware to protect routes by userId  • Validate POST and PUT data  • Start writing unit tests for GET routes* |
| Week 07 | *• Finish all testing  • Finalize and polish Swagger documentation  • Record and edit video presentation (via Teams + Clipchamp)  • Push final code to GitHub and Render* |

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

We're both pet owners and excited about this project, so we're aiming to collaborate as equally as possible but we're working asynchronously due to time zone differences, so we’ll use GitHub Issues and commit tags to keep track of tasks. We should be putting all code changes in branches and creating PRs for review and the other person should approve and merge the code after testing.

Main Tasks:  
Caitlin:

* Create Repo
* Create MongoDB collection set up (some generic pets we can alter and edit after OAUTH implemented)
* Google OAuth setup and auth middleware / add to render env
* Swagger documentation set up / update routes as they are created
* USER (specifically connecting IDs?)

Karim:

* Pet Routes (Caitlin can help / take if USER and ID completed)\*
* MedicalRecords Routes
* AppointmentHistory Routes
* Type Validations / Unit testing for GET routes
* Create ReadMe.md

We'll communicate over Teams to plan tasks and merge pull requests as we go.

# Potential Risks and Risk Mitigation Techniques

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

* Time zone differences
* Miscommunication on overlapping tasks
* Scope creep if we try to overbuild

## How will you mitigate or overcome these risks?

* Stick to main plan first, then stretch goals if we complete early
* Use Teams for communication and Notion files (published) for async planning
* Schedule 1–2 weely async checkins to stay communicating on progress
* Use clear commit messages and frequent pushes through branches and PRs