CSE 341 Final project Proposal

# General Info

Christopher Edeson

SkillSync – Developer Skill Matching API

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) 2

[What file structure and program architecture will you use for this project (how will you organize your node project)? Why?](#_4d34og8) 2

[What are potential stretch challenges that you could implement to go above and beyond?](#_17dp8vu) 2

[**API Endpoint Planning**](#_mral43ww1pvt) **2**

[Project Scheduling and Delegation](#_1ksv4uv) **3**

[How will you divide up work in your team to ensure the following tasks all get completed?](#_3rdcrjn) 3

[Potential Risks and Risk Mitigation Techniques](#_26in1rg) **4**

[What are the risks involved with you being able to finish this project in a timely manner?](#_lnxbz9) 4

[How will you mitigate or overcome these risks?](#_35nkun2) 4

# Application Info

## What will the API do?

SkillSync is a RESTful API that connects developers with project opportunities based on their skills, experience level, and interests. It allows users to register, build a skill profile, and find compatible projects or collaborators. Project managers can also post projects with specific requirements, and the API matches potential contributors using intelligent search and filtering.

## How will your API utilize a login system?

The API will implement **OAuth 2.0 authentication** using **GitHub login** (via Passport.js). This allows developers to securely sign in using their GitHub accounts. Authenticated users will receive a JWT (JSON Web Token) for secure API access and session persistence.

## What database will you use?

We will use **MongoDB Atlas** for flexible, cloud-hosted document storage that integrates well with Node.js and Mongoose.

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

The database will consist of four main collections:

1. **Users** – stores user profiles, skills, experience, and authentication info.
2. **Projects** – stores project details, required skills, and owner references.
3. **Applications** – connects users to projects they’ve applied to.
4. **Messages** – enables user-to-user communication within project teams.

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

Frontend apps (e.g., React or Vue) can store and manage the JWT in local storage or cookies. The token will be verified on every request to protected routes, and the API will return appropriate 401 Unauthorized or 403 Forbidden responses if the token is missing or invalid.

## 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?

Sensitive data such as passwords, access tokens, and personal details will be encrypted or hashed (bcrypt for passwords, HTTPS for all traffic). Authorization checks will ensure that users can only access or modify their own resources. Input validation (using Joi) and sanitization will prevent injection attacks.

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

We’ll use an **MVC (Model-View-Controller)** structure for scalability and maintainability:

/src

/models

/controllers

/routes

/middleware

/config

/tests

This separation allows cleaner organization, easier testing, and independent updates to each layer.

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

- Add AI-based skill matching (using OpenAI Embeddings or cosine similarity).

- Implement a rating and feedback system for collaborators.

- Add real-time messaging via WebSockets.

- Integrate resume parsing for automated skill extraction.

# API Endpoint Planning

For this section, you’ll plan out what API endpoints you’ll need for your project. If you go to [editor.swagger.io](https://editor.swagger.io/) you’ll see the Pet Store application documentation that they have. This can be a good point of reference because they demonstrate how to have multiple database entities (ie: pet, store, user), and CRUD operations for each with various ways of performing them. For this section of the Final Project Proposal, you will make a list of each api endpoint that will be supplied for each database entity. So, if I was going to create the pet store app, I’d put something like this:

* pet
  + POST /pet
  + PUT /pet
  + GET /pet/findByStatus
  + GET /pet/findByTags
  + GET /pet/{petId}
  + POST /pet/{petId}
  + DELETE /pet/{petId}
  + POST /pet/{petId}/uploadImage
* store
  + GET /store/inventory
  + POST /store/order
  + GET /store/order/{orderId}
  + DELETE /store/order/{orderId}
* user
  + POST /user
  + POST /user/createWithArray
  + POST /user/createWithList
  + GET /user/login
  + GET /user/logout
  + GET /user/{username}
  + PUT /user/{username}
  + DELETE /user/{username}

Thinking about this now will be extremely helpful for you because next week when you have to create the swagger documentation for all of this and publish it to heroku so it is ready for the rest of your project.

**This is mine.**

**API Endpoint Planning**

**Users**

* POST /users/register
* POST /users/login
* GET /users (Admin only)
* GET /users/:id
* PUT /users/:id
* DELETE /users/:id

**Projects**

* POST /projects
* GET /projects
* GET /projects/:id
* PUT /projects/:id
* DELETE /projects/:id

**Applications**

* POST /applications (user applies to a project)
* GET /applications
* GET /applications/:id
* DELETE /applications/:id

**Messages**

* POST /messages
* GET /messages/:conversationId
* DELETE /messages/:id

**Auth**

* GET /auth/github
* GET /auth/github/callback
* GET /auth/logout

# 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 | *Project Proposal* |
| Week 05 Tasks | * *Create Git Repo* * *Push to Heroku* * *API DOCUMENTATION is complete and available at route ‘/api-docs’*   Create GitHub repository, setup Node project, initialize MongoDB Atlas connection, and create Swagger documentation at ‘*/api-docs’* |
| Week 06 Tasks | Implement CRUD operations for Users and Projects, JWT authentication, and error handling |
| Week 07 Tasks | Complete Applications and Messages routes, write unit tests for GET routes, deploy to Render, and record video presentation |

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

* HTTP GET, GET (all, single) Christopher
* HTTP POST Christopher
* HTTP PUT Christopher
* HTTP DELETE Christopher
* Node.js project creation Christopher
* Create git repo and share with group Christopher
* MongoDB setup Christopher
* API Swagger documentation for all API routes Christopher
* Video presentation of node project, all routes functioning, mongoDB data being modified, and API documentation. Christopher

# Potential Risks and Risk Mitigation Techniques

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

- Workload and overburden.

- Delayed API integration or deployment errors.

- Team member inactivity or communication gaps.

- Render deployment issues or environment variable errors.

- Risk of Burnout

## How will you mitigate or overcome these risks?

* Set up Trello board to organize tasks.
* Test deployment early using a staging branch before final submission