# CyMatch

Final Project Technical Report

SE/COM S 3190 – Construction of User Interfaces
Spring 2025

Team Members:
Noah Wons - wons123@iastate.edu
Alexander Ganekov - aganekov@iastate.edu

May 11, 2025

## 1. Introduction

CyMatch is a full-stack web application designed to help students find internship or job opportunities based on quick preferences, similar to a swipe-based UI. The project aims to make job discovery more engaging and modern by combining a clean UI with backend functionality for saving job data, user profiles, and session logic. The platform is designed for college students and early-career individuals seeking entry-level positions. It addresses the need for simple, intuitive user interfaces that streamline job exploration while still supporting full CRUD operations behind the scenes. CyMatch is an original project developed specifically for SE/COM S 3190, but is inspired by the way Tinder works, with the left/right swiping mechanics.

## 2. Project Description

CyMatch is a single-page web application built with React that allows users to explore job opportunities through a swipeable interface. The application includes several key views: a login/register screen for authentication, a home page where users can swipe through job cards, a saved jobs page to review previously liked positions, a profile page for entering contact and resume information, and an about page describing the course and team members. Users can register and log in, save or dismiss jobs using swipe buttons, and view or edit their personal profile. All job and user data is managed through a Node.js and Express backend, with MongoDB handling persistent storage. The frontend is styled using Tailwind CSS for a clean, responsive layout, and all page views are managed through React hooks and state-based conditional rendering.

## 3. File and Folder Architecture

CyMatch is built as a full-stack MERN application using React for the frontend, Express and Node.js for the backend, and MongoDB as the database. The application is structured into a frontend folder for all React components and a backend folder containing API routes, models, and server configuration. On the client side, components are organized into modular files for swipeable cards, profile forms, login/register pages, and navigation. The backend exposes RESTful endpoints for user registration, login, and job data operations. Communication between the frontend and backend occurs through asynchronous fetch calls, allowing the application to retrieve and modify data in real time. The system is deployed locally for development, with both the frontend and backend running on separate ports. Tailwind CSS is used for UI styling, and application state is managed using React's built-in hooks like useState and useEffect.

## 4. Code Explanation and Logic Flow

### 4.1. Frontend–Backend Communication

The frontend communicates with the backend using asynchronous fetch calls to specific API routes. When a user logs in, registers, saves a job, or submits their profile, the frontend sends a GET or POST request to the backend server running on localhost port 3000. These requests are defined inside the services folder on the frontend, where reusable API functions handle the communication. The backend is structured with route files such as users.js, which receive incoming requests and pass them to the correct database functions. This separation of concerns keeps the frontend focused on rendering and interaction, while the backend handles data processing and storage.

## 4.2. React Component Structure

The React components are organized in the components folder within the src directory. Each major feature of the application has its own component file, such as login, register, card deck, swipeable card, profile, saved jobs, tip banner, and about. The main file is App.jsx, which uses React hooks to manage the application state. The app uses conditional rendering to control which view is displayed, based on the current state. Components are modular and reusable, and the structure allows for clear separation between individual views and shared UI elements.

## 4.3. Database Interaction

The backend uses MongoDB to store user and job data. Mongoose is used to define database models and handle queries. The user model is defined in user.js and includes fields such as name, email, and resume. The users.js route file handles incoming requests to create or retrieve user data by calling functions like save and find from the Mongoose library. The database connection is initialized in server.js using mongoose.connect, allowing the backend to store and retrieve data as needed. This setup allows the application to support persistent user profiles and saved jobs..

## 4.4. Code Snippets

This snippet shows how the application switches between views like home, saved jobs, profile, and login based on the view state.

```
{view === ’home’ ? (
  <CardDeck savedJobs={savedJobs} setSavedJobs={setSavedJobs} />
) : view === ’saved’ ? (
  <SavedJobs />
) : view === ’profile’ ? (
  <Profile />
) : view === ’about’ ? (
  <About />
) : view === ’login’ ? (
  <Login setLogin={setLogin} setView={setView} />
) : view === ’register’ ? (
  <Register setLogin={setLogin} setView={setView} />
) : null}
```

This conditional rendering ensures that only one component is displayed at a time based on the user's current location in the app.

This backend route handles the registration of a new user by receiving data from the frontend and saving it to MongoDB using Mongoose.

```
router.post('/', async (req, res) => {
  try {
    const newUser = new User(req.body);
    const savedUser = await newUser.save();
    res.status(201).json(savedUser);
  } catch (err) {
    res.status(500).json(err);
  }
});
```

This route enables basic user creation and persistence in the database.

---

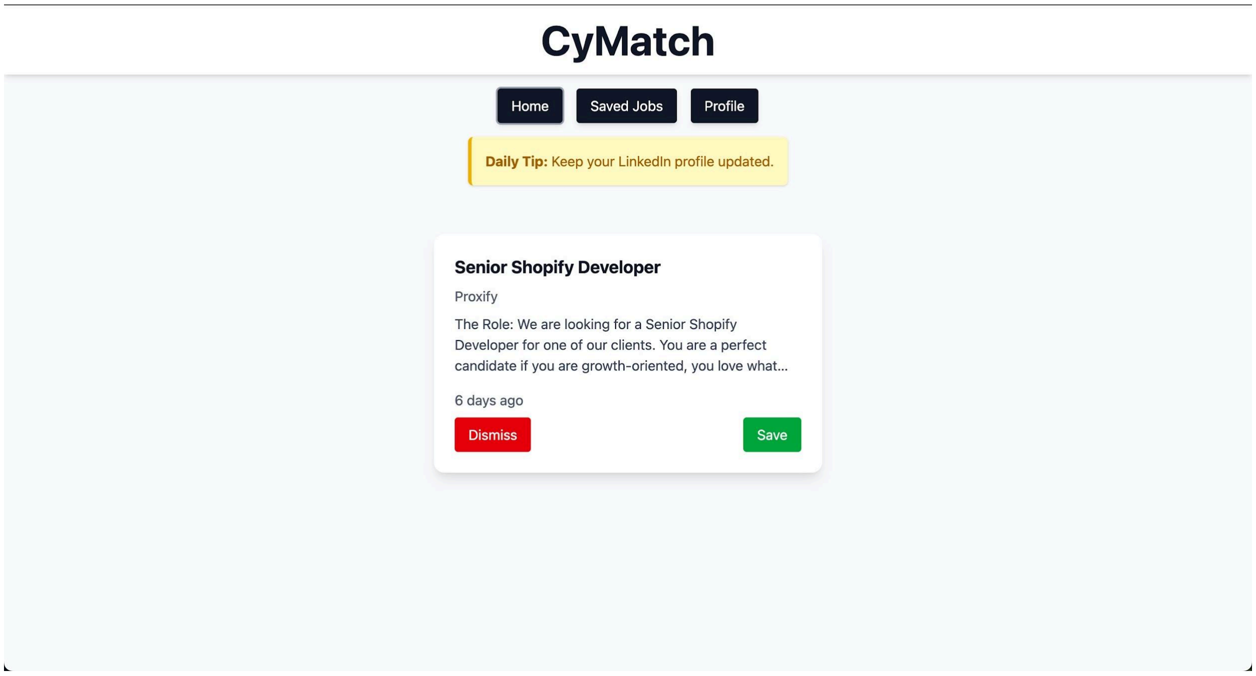
This function sends a POST request from the frontend to the backend to save a job. It abstracts the fetch logic so UI components can easily reuse it.

```
export const saveJob = async (id) => {
  await fetch('http://localhost:3000/jobs/save', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ jobId: id }),
  });
};
```

This modular function helps maintain clean separation between frontend UI and backend communication logic.

## 5. Web View Screenshots and Annotations

Screenshot 1: Homepage



**Daily Tip:** Keep your LinkedIn profile updated.

**Senior Shopify Developer**

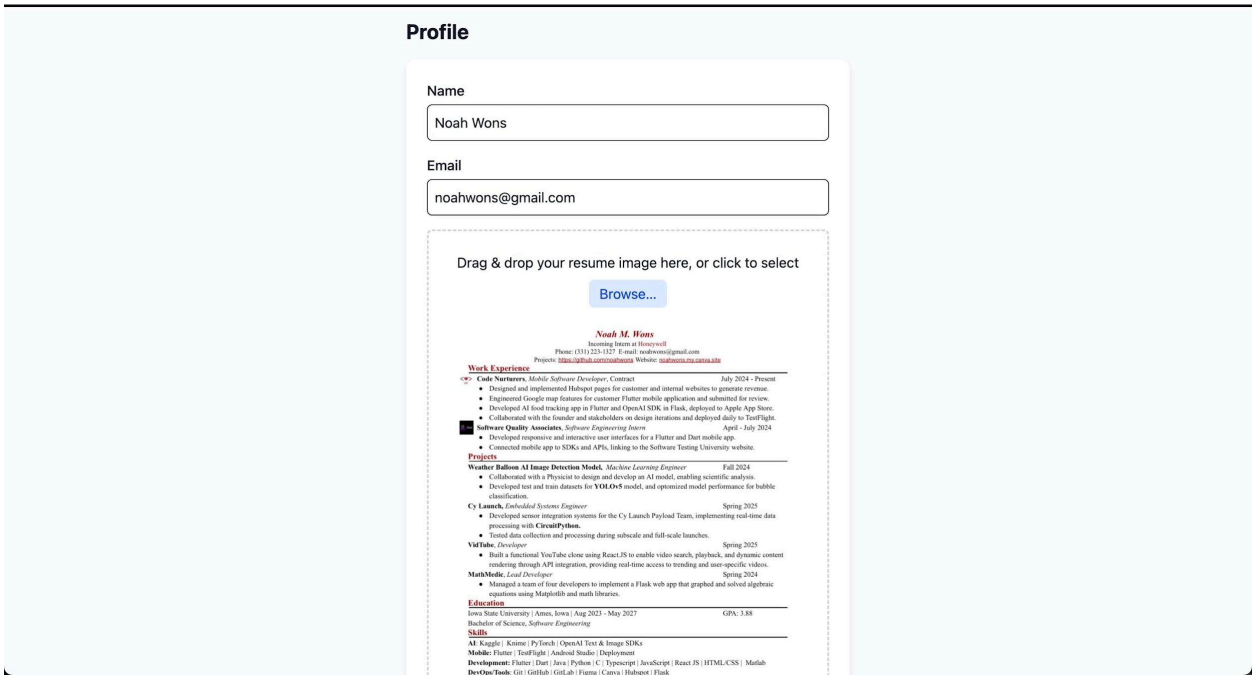
Proxify

The Role: We are looking for a Senior Shopify Developer for one of our clients. You are a perfect candidate if you are growth-oriented, you love what...

6 days ago

This is the home screen of the application where users can view and interact with job cards. Users can dismiss a job by clicking “Dismiss” or save a job by clicking “Save.” Each card shows the job title, company, and a brief description. Once a job is saved or dismissed, the next job card is revealed in the stack. The daily tip banner is also visible at the top.

SS 2: Profile



**Profile**

Name: Noah Wons

Email: noahwons@gmail.com

Drag & drop your resume image here, or click to select

Browse...

*Noah M. Wons*
Incoming Intern at Honeywell
Phone: (331) 223-1327 E-mail: noahwons@gmail.com
Projects: https://github.com/noahwons Website: noahwons.my.canva.site

**Work Experience**

**Code Nurturers**, *Mobile Software Developer*, Contract — July 2024 - Present
- Designed and implemented Hubspot pages for customer and internal websites to generate revenue.
- Engineered Google map features for customer Flutter mobile application and submitted for review.
- Developed AI food tracking app in Flutter and OpenAI SDK in Flask, deployed to Apple App Store.
- Collaborated with the founder and stakeholders on design iterations and deployed daily to TestFlight.

**Software Quality Associates**, *Software Engineering Intern* — April - July 2024
- Developed responsive and interactive user interfaces for a Flutter and Dart mobile app.
- Connected mobile app to SDKs and APIs, linking to the Software Testing University website.

**Projects**

**Weather Balloon AI Image Detection Model,** *Machine Learning Engineer* — Fall 2024
- Collaborated with a Physicist to design and develop an AI model, enabling scientific analysis.
- Developed test and train datasets for **YOLOv5** model, and optomized model performance for bubble classification.

**Cy Launch,** *Embedded Systems Engineer* — Spring 2025
- Developed sensor integration systems for the Cy Launch Payload Team, implementing real-time data processing with **CircuitPython.**
- Tested data collection and processing during subscale and full-scale launches.

**VidTube**, *Developer* — Spring 2025
- Built a functional YouTube clone using React.JS to enable video search, playback, and dynamic content rendering through API integration, providing real-time access to trending and user-specific videos.

**MathMedic**, *Lead Developer* — Spring 2024
- Managed a team of four developers to implement a Flask web app that graphed and solved algebraic equations using Matplotlib and math libraries.

**Education**

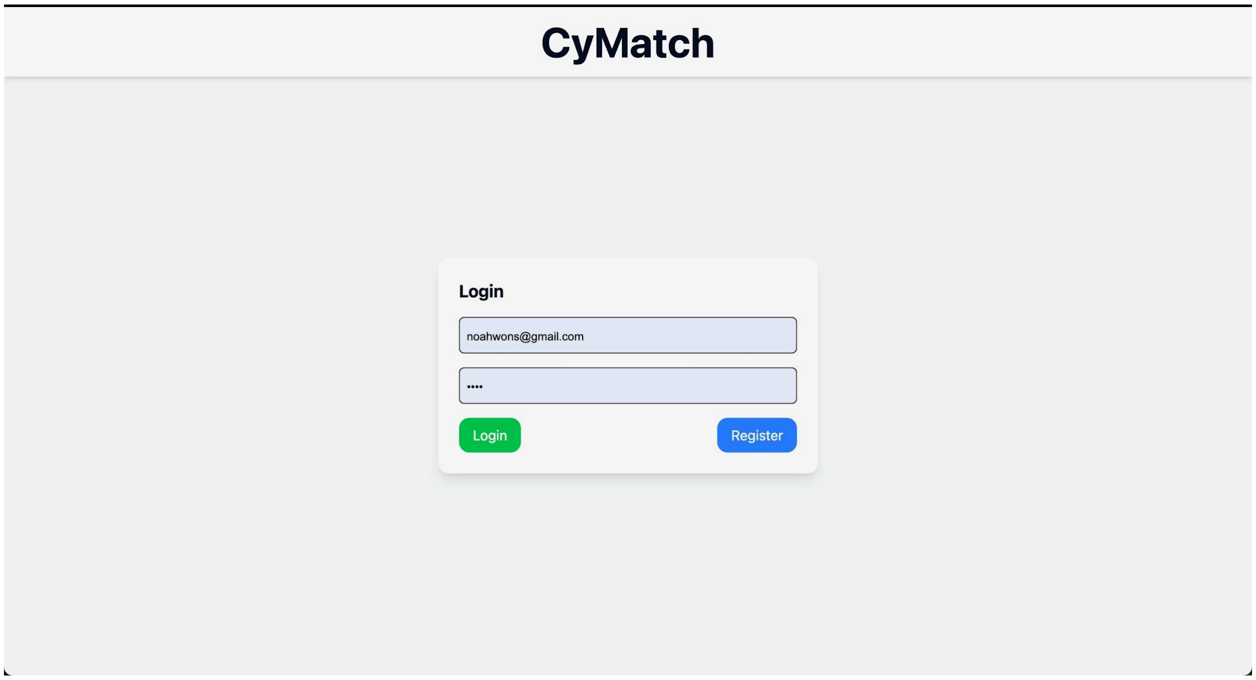
Iowa State University | Ames, Iowa | Aug 2023 - May 2027 — GPA: 3.88
Bachelor of Science, *Software Engineering*

**Skills**

**AI**: Kaggle | Knime | PyTorch | OpenAI Text & Image SDKs
**Mobile:** Flutter | TestFlight | Android Studio | Deployment
**Development:** Flutter | Dart | Java | Python | C | Typescript | JavaScript | React JS | HTML/CSS | Matlab
**DevOps/Tools**: Git | GitHub | GitLab | Figma | Canva | Hubspot | Flask

This page allows users to enter and update their profile information including name, email, and resume details. Once submitted, the profile is saved (future versions will save this to the database). The user interface uses styled input fields and includes a submit button. This is where users customize their presence in the app.

SS3: Login

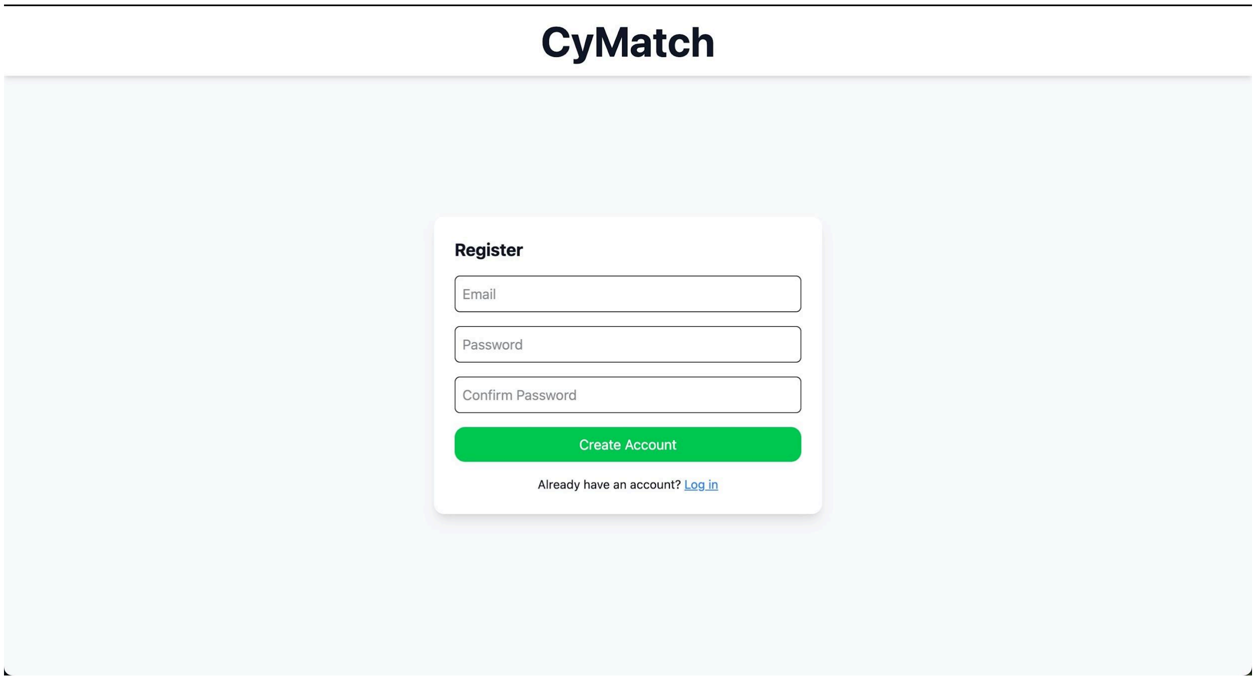


CyMatch

Login

noahwons@gmail.com

••••



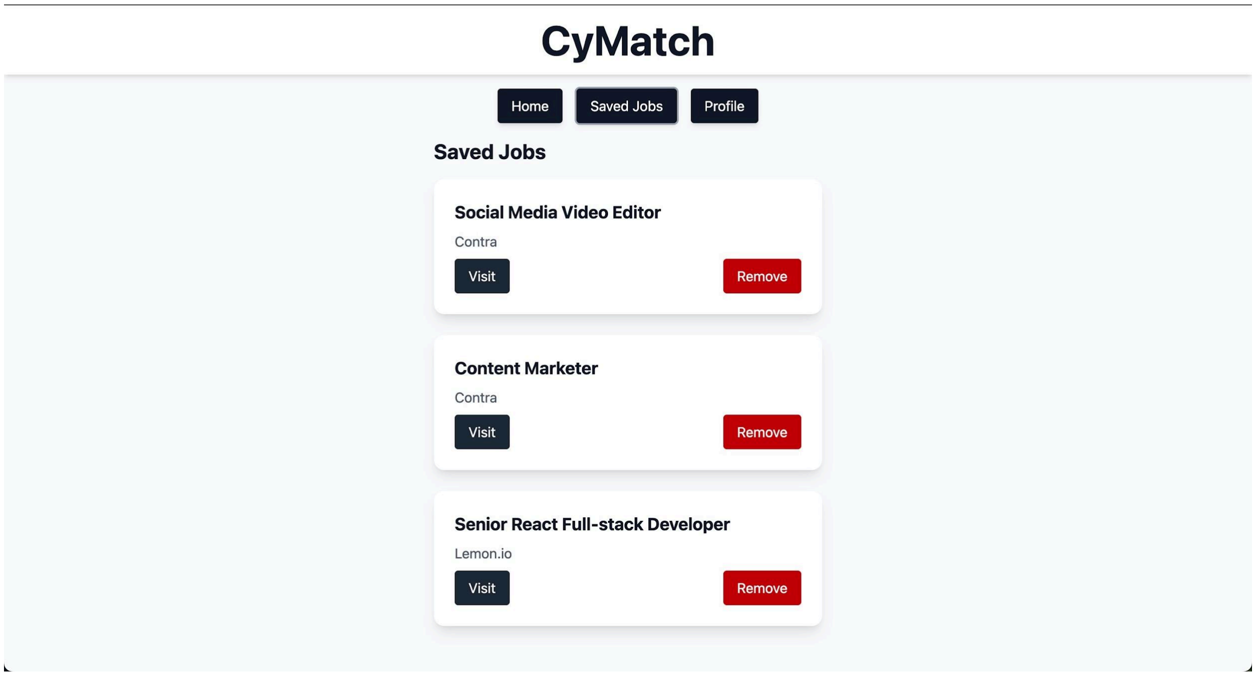
CyMatch

Register

Already have an account? Log in

This is the login screen shown when the user first accesses the app. The user can either enter their credentials to log in or click “Register” to create a new account. Upon successful login, the app transitions to the main view and unlocks navigation. This page uses React hooks to manage form input and updates the login state.

SS4: Saved Jobs



This view displays all the jobs a user has saved by swiping right on the home page. Each saved job includes the title, company, and job description. This page helps users keep track of positions they are interested in. The data is pulled from the savedJobs state and rendered dynamically.

## 6. Installation and Setup Instructions

To run the CyMatch application locally, follow the steps below to install and start both the frontend and backend. The application uses Node.js, Express, MongoDB, and React. To run the CyMatch application locally, users must have Node.js, npm, and MongoDB installed. After cloning the project repository from GitLab, the user should navigate to the backend folder and run npm install to install all required dependencies. A .env file must be created inside the backend directory with the following environment variables: MONGO_URL set to mongodb://localhost:27017/cymatch, and PORT set to 3000. Once the environment variables are configured, the backend can be started by running node server.js, which launches the Express server on localhost port 3000.
Next, the user should navigate to the frontend folder and run npm install to install frontend dependencies. After installation, the frontend can be started with the command npm run dev, which starts the Vite development server on localhost port 5173. The application will then be accessible in the browser at http://localhost:5173, where users can register, log in, swipe through job cards, view saved jobs, and complete their profile.

## 7. Contribution Overview

Alexander Ganekov
- Implemented swipeable job cards (CardDeck, SwipeableCard)
- Created the Saved Jobs view
- Built the Profile page
- Styled frontend using Tailwind CSS
- Implemented About page
- Integrated frontend navigation and view logic

Noah Wons
- Created Login and Register components
- Set up backend routes and Express server
- Connected MongoDB and defined Mongoose models
- Handled backend database integration for user data
- Added API communication between the frontend and the backend
- Structured backend folder and server configuration

## 8. Challenges Faced

One major challenge was setting up the backend connection with MongoDB. Initially, the server could not connect due to a missing environment variable and incorrect URI format. This was resolved by adding a `.env` file with the correct MONGO_URL and ensuring MongoDB was running locally.

Another challenge involved styling conflicts when integrating Tailwind CSS. Some styles were not applied correctly because Tailwind was not properly initialized in the Vite project. We resolved this by manually creating the Tailwind and PostCSS configuration files and importing Tailwind directives in the index.css file.

## 9. Final Reflections

This project helped us gain hands-on experience with full-stack web development, from building a responsive frontend in React to connecting it with a functional Express and MongoDB backend. We learned how to manage state in a single-page application, implement modular components, and structure RESTful API calls. Working through integration challenges and GitLab coordination also improved our ability to debug and collaborate effectively in a team setting. If we were to improve the project, we would focus on enhancing the login system with hashed passwords and tokens, fully persisting saved jobs to the database, and possibly deploying the app for live testing. Overall, the project reinforced the importance of clean architecture, version control, and consistent communication in software development.