ResumeReady

Final Project Technical Report

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

Team Members:
Rejinthala Mani Raj - sai143@iastate.edu
Kenny Jia Hui Leong - kennyljh@iastate.edu

May 11, 2025

1. Introduction

Primarily, the idea for this project originated from our own interests and needs. However, during the course of development, we explored several notable web applications offering similar services such as Resume.com, ResumeNerd, and others. These platforms provided valuable insights and influenced some of our design decisions, particularly in user interface layout and template handling. Our project is a resume builder web application that allows users to efficiently create professional, visually appealing, and ATS-friendly resumes. It reduces the complexity typically associated with resume formatting by offering structured form inputs, predefined templates, and real-time previews. The goal is to enable users of all categories, including students, professionals, and job seekers to generate high-quality resumes in less time, with minimal effort, and complete control over their content and presentation.

2. Project Description

The major features of our application include:

- **User Authentication:** Signup, login, OTP-based email verification using nodemailer.
- **Resume Creation Interface:**
    - Left panel: Form inputs for various resume sections.
    - Right panel: Real-time preview of resume based on selected template and input.
- **Dashboard:** Personalized homepage displaying the user's saved resumes, templates, and assets.
- **Template and Asset Library:** Users can browse, filter, and select templates and resume sections.
- **Resume Export Options:** Users can download their resume as **PDF or PNG**.
- **Account Management:** Users can update their email, password, or delete their account.

3. File and Folder Architecture

Backend/

```
├archived data/
    ├ coms3190.assets.json
    ├ coms3190.assetsmetadata.json
    ├ coms3190.services.json
    ├ coms3190.subscriptions.json
    └ coms3190.templates.json
├routes/
    ├ assets.js
    ├ assetsmetadata.js
    ├ genOTP.js
    ├ resumes.js
    ├ services.js
    ├ subscriptions.js
    ├ templates.js
    └ users.js
└ server.js
```

Documents/

```
├ Final Project/
    ├ Wireframes/
    └ IP21_Final_Proposal.pdf
└ Mini-assignments/
    ├ Mini-assignment-1/
    └ Mini-assignment-2/
```

Frontend/

```
src/
├ assets/
|  ├ images/
|  |  ├ Assets/
|  |  ├ Home/
|  |  ├ icons/
|  |  ├ landing/
|  |  ├ logo/
|  |  ├ mock dashboard/
|  |  ├ old/
|  |  ├ profilePic/
|  |  ├ services/
|  |  ├ subscriptions/
|  |  ├ templates/
|  |  └ .gitkeep
|  └ .gitkeep
├ components/
|  ├ .gitkeep
|  ├ AboutUs.jsx
|  ├ Assets.jsx
|  ├ AssetSelect.jsx
|  ├ AuthContext.jsx
|  ├ Boilerplate.jsx
|  ├ CEmail.jsx
|  ├ CPassword.jsx
|  ├ Dashboard.jsx
|  ├ Delete.jsx
|  ├ Landing.jsx
|  ├ Login.jsx
|  ├ MyAssets.jsx
|  ├ MyResumes.jsx
|  ├ MyTemplates.jsx
|  ├ ProcessHelper.jsx
|  ├ Protected.jsx
|  ├ ResumeCreation.jsx
|  ├ Services.jsx
|  ├ SideBar.jsx
|  ├ SignUp.jsx
|  ├ Subscriptions.jsx
|  ├ Templates.jsx
|  ├ TemplateSelect.jsx
|  ├ UserVerify.jsx
|  └ Utils.jsx
├ data/
|  ├ .gitkeep
|  ├ AssetProducts.js
|  ├ LandingInfo.js
|  ├ MockAssets.js
|  ├ MockCategorySubcategory.js
|  ├ MockResumes.js
|  ├ MockTemplates.js
|  ├ ServiceProducts.js
```

```
|  ├ SubscriptionProducts.js
|  └ TemplateProducts.js
├ App.jsx
├ index.css
└ main.jsx
```

4. Code Explanation and Logic Flow

4.1. Frontend–Backend Communication

The frontend uses an asynchronous function to send HTTP requests to the backend using routes that are specified beforehand. For example, when the user clicks “Save Resume” button, the asynchronous function collects data from the state variable, stringifies it to JSON before putting it in the request body, then places it in the request body (with a specified method: “POST”) to be sent along with the HTTP request to the backend (e.g. localhost:8080/resume). When the backend receives it, the type of method (e.g. POST) determines which mapping is called on. For the POST mapping, the saved resume is retrieved from the request body, along with the userid and resumeid from the query parameters. The userid and resumeid is checked against the mongodb database to check for duplicates, if there is a duplicate, a 409 conflict response will be sent back, otherwise the resume is saved and a 200 OK response is sent back.

4.2. React Component Structure

The component at the highest level would be App.jsx. All of the routing, toast container, and page authentication are declared and managed here. Going down a level, we have components: AboutUs.jsx, Assets.jsx, Dashboard.jsx, Landing.jsx, Login.jsx, ResumeCreation.jsx, Services, SignUp.jsx, Subscriptions.jsx, Templates.jsx that represent the “main” pages of our website. Components called by these main components, called in-between pages, are: AssetsSelect.jsx, CEmail.jsx, CPassword.jsx, Delete.jsx, MyAssets.jsx, MyResumes.jsx, MyTemplates.jsx, SideBar.jsx, and UserVerify.jsx. For utility components, we have: AuthContext.jsx, Boilerplate.jsx, ProcessHelper.jsx, Protected.jsx, and Utils.jsx.

4.3. Database Interaction

The database used is mongodb, and is mainly communicated with using functions such as findOne, find, deleteOne, updateOne, and insertOne. We have a number of collections, each storing unique

data such as users or resumes. For each of these collections, we have separate routers such as user.js and resume.js, each containing all necessary mappings/routes for CURDL operations. The main "manager" of these different routes is server.js, where the database is declared, necessary modules are imported, and all different routes listed.

4.4. Code Snippets

Authentication Context:

```
import { createContext, useContext, useState } from 'react';

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [authenticated, setAuthenticated] = useState(false);

  const login = () => setAuthenticated(true);
  const logout = () => setAuthenticated(false);

  return (
    <AuthContext.Provider value={{ authenticated, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

Post Request to send OTP

```
router.post("/", async (req, res) => {
 const {email} = req.body;
 const otp = genOtp();
 const data={};
 try{
    await transporter.sendMail({
        from: `"ResumeReady" <resumebuilder577@gmail.com>`,
        to: email,
        subject: "ResumeReady verification OTP",
        html:
        `...
        `
    });
    console.log(`OTP for ${email}: ${otp}`);
    data.OTP=otp;
    res.status(200);
    res.send(data);
 }catch(err){
   console.error(err);
   res.status(500);
   res.send(data);
 }
});
```

```
const handleSendOtp = async (e) => {
  e.preventDefault();
  const res = await fetch("http://localhost:8080/sendOtp", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ email }),
  });
  const data = await res.json();
  if (data.OTP) setGenOtp(data.OTP);
  else alert("Failed to send OTP, try again!");
};
```

Libraries/Modules integration

```
const transporter= nodemailer.createTransport({
  service:"Gmail",
  auth:{
     user: "resumebuilder577@gmail.com",
     pass: "ghvlvrrqgzssrudr",
  },
});
```

```
const handleDownloadPNG = async () => {

    const masterContainer = document.getElementById('fixed-size-resume-master-container');

    if (!masterContainer){
        console.log("Resume master container not found");
        return;
    }

    const canvas = await html2canvas(masterContainer, {
        backgroundColor: null,
        scale: 2,
    });

    // convert image to base64-encode
    const dataURL = canvas.toDataURL('image/png');

    const link = document.createElement('a');
    link.href = dataURL;
    link.download = 'resume' + getCurrentDateTime() + '.png';
    link.click();
    toast.success("Resume PNG created");
};
```

```
const handleDownloadPDF = async () => {

    const masterContainer = document.getElementById('fixed-size-resume-master-container');

    if (!masterContainer){
        console.log("Resume master container not found");
        return;
    }

    const canvas = await html2canvas(masterContainer, {
        backgroundColor: null,
        scale: 2,
    });

    // convert image to base64-encode
    const dataURL = canvas.toDataURL('image/png');

    const pdf = new jsPDF({
        orientation: "portrait",
        unit: "px",
        format: [canvas.width, canvas.height]
    });

    pdf.addImage(dataURL, 'PNG', 0, 0, canvas.width, canvas.height);
    pdf.save('resume' + getCurrentDateTime() + '.pdf');
    toast.success("Resume PDF created");
}
```

Database Interaction:

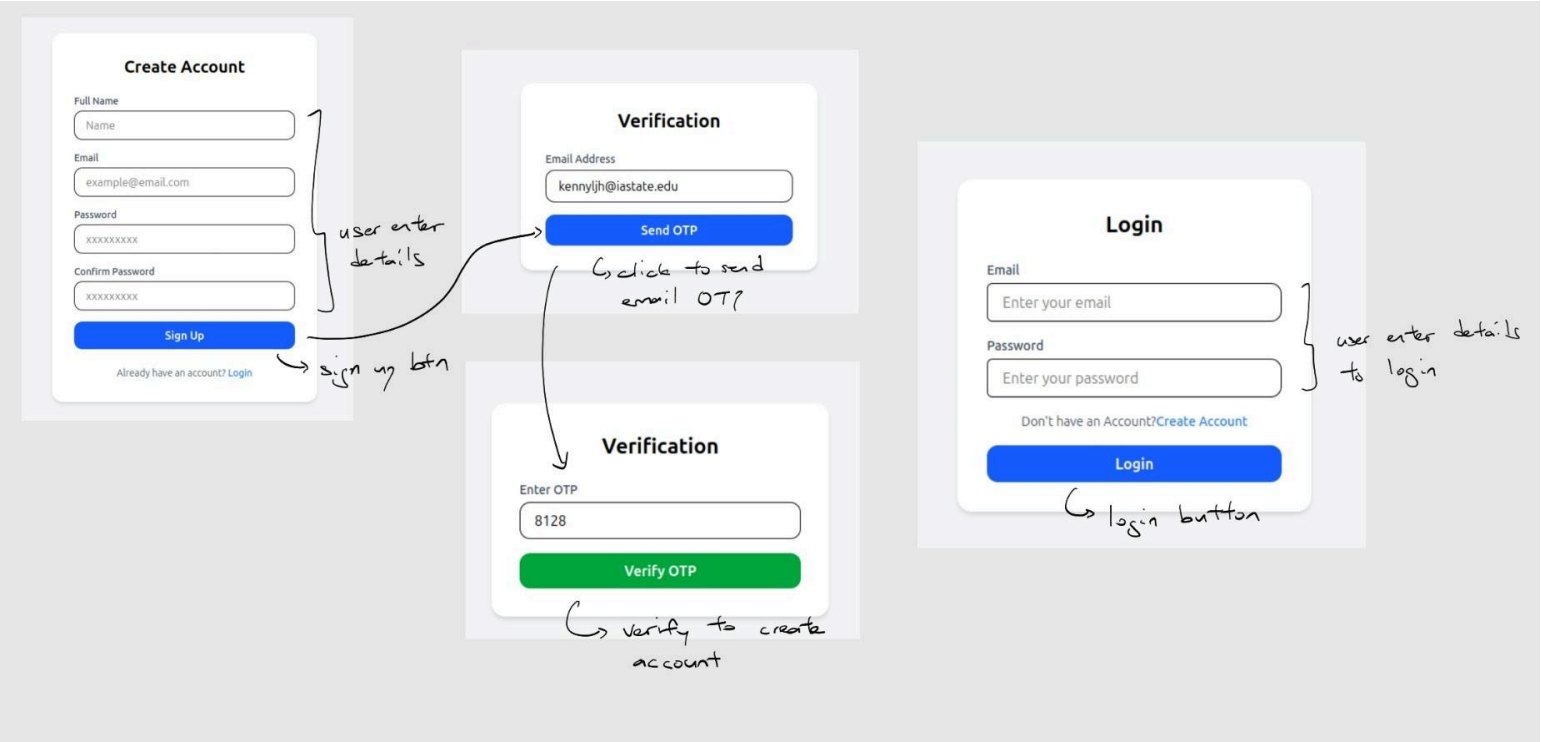
```
const {MongoClient} = require("mongodb");
const url = "mongodb://localhost:27017";
const dbName = "coms3190"
const client = new MongoClient(url);
```

```
await client.connect();
const db = client.db(dbName);
```

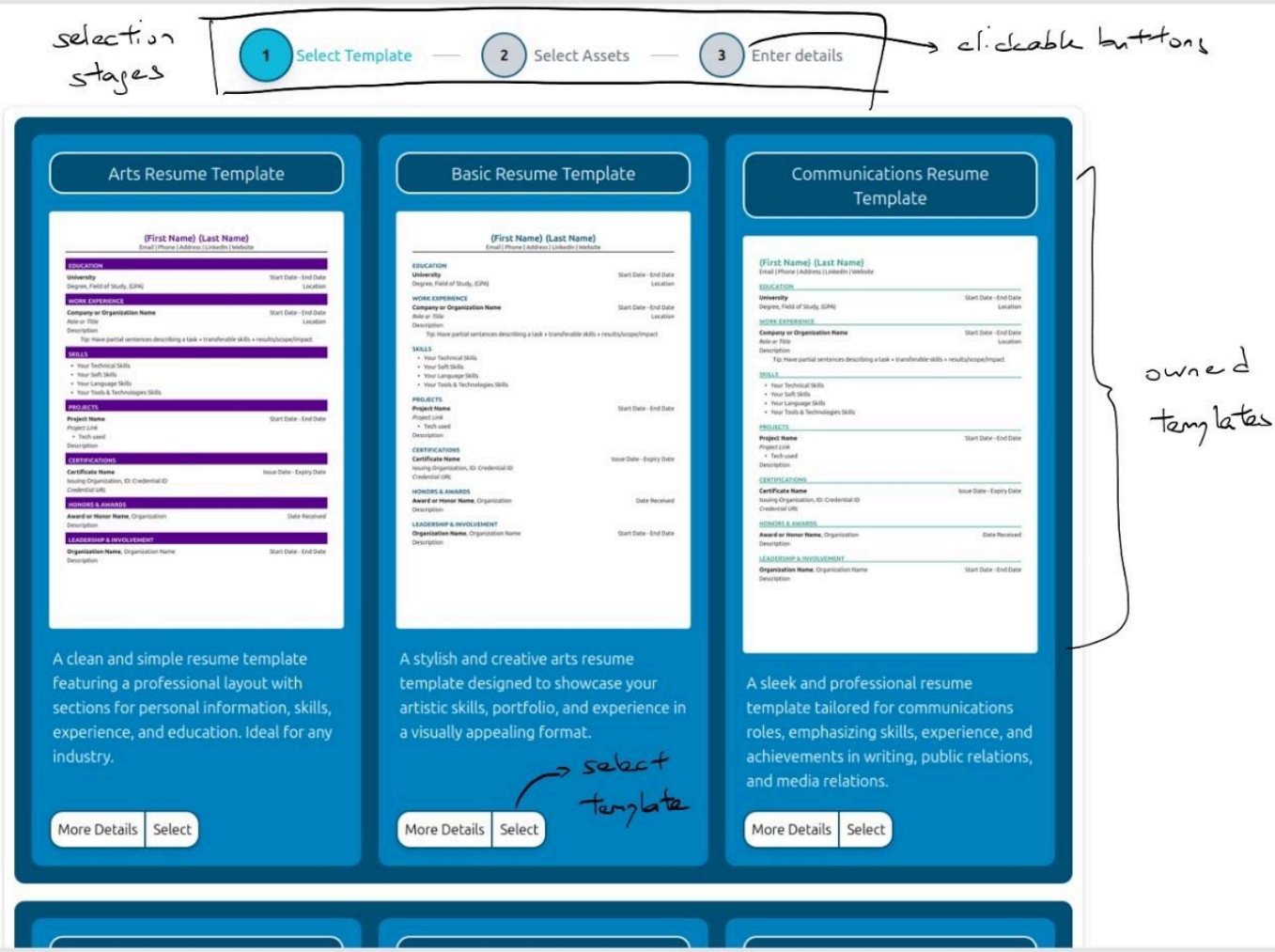
```
const users = await db.collection("users").find({}).limit(100).toArray();
```

```
const templates = await db
    .collection("templates")
    .find({})
    .limit(100)
    .toArray();
```

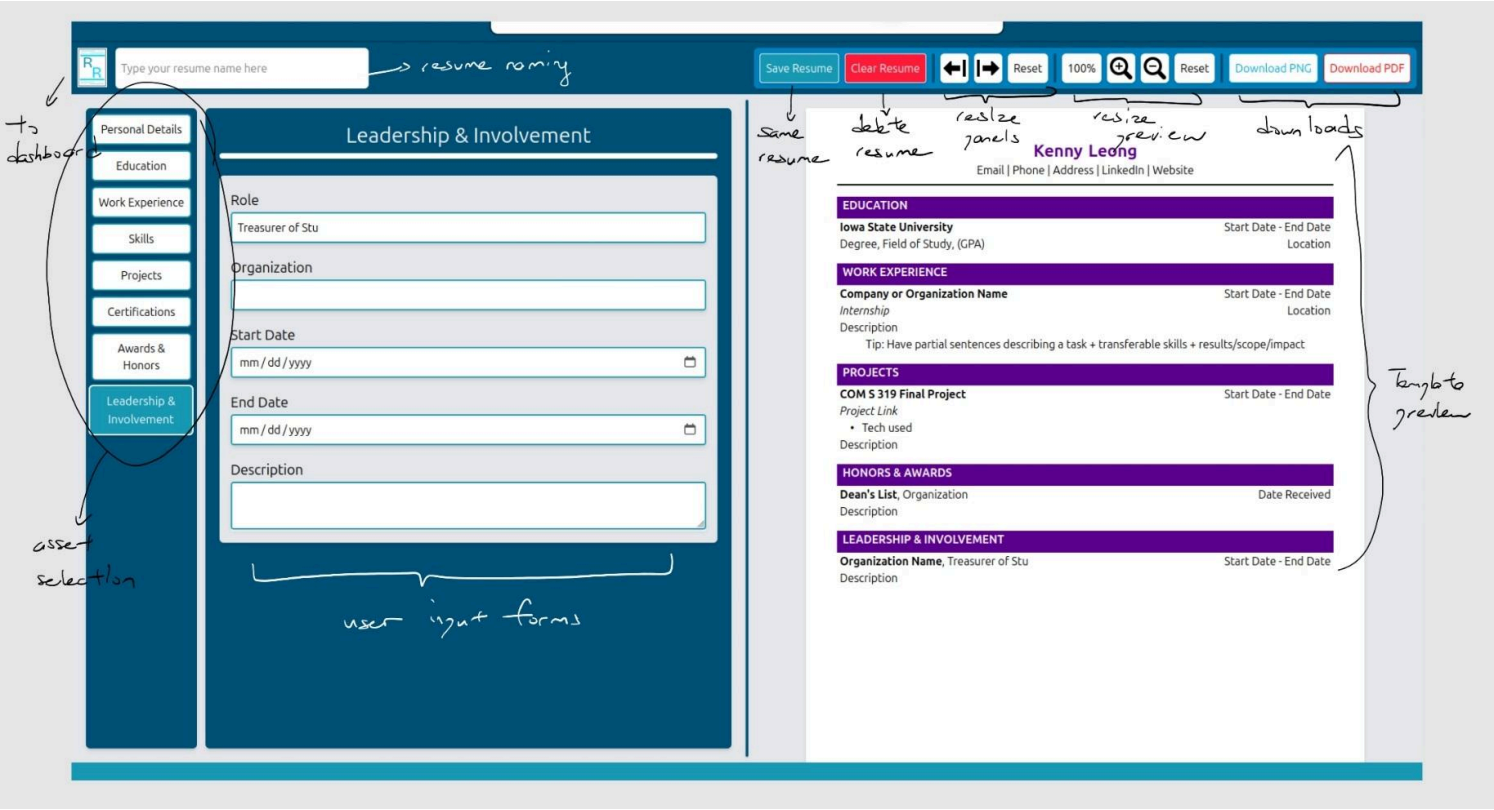
5. Web View Screenshots and Annotations



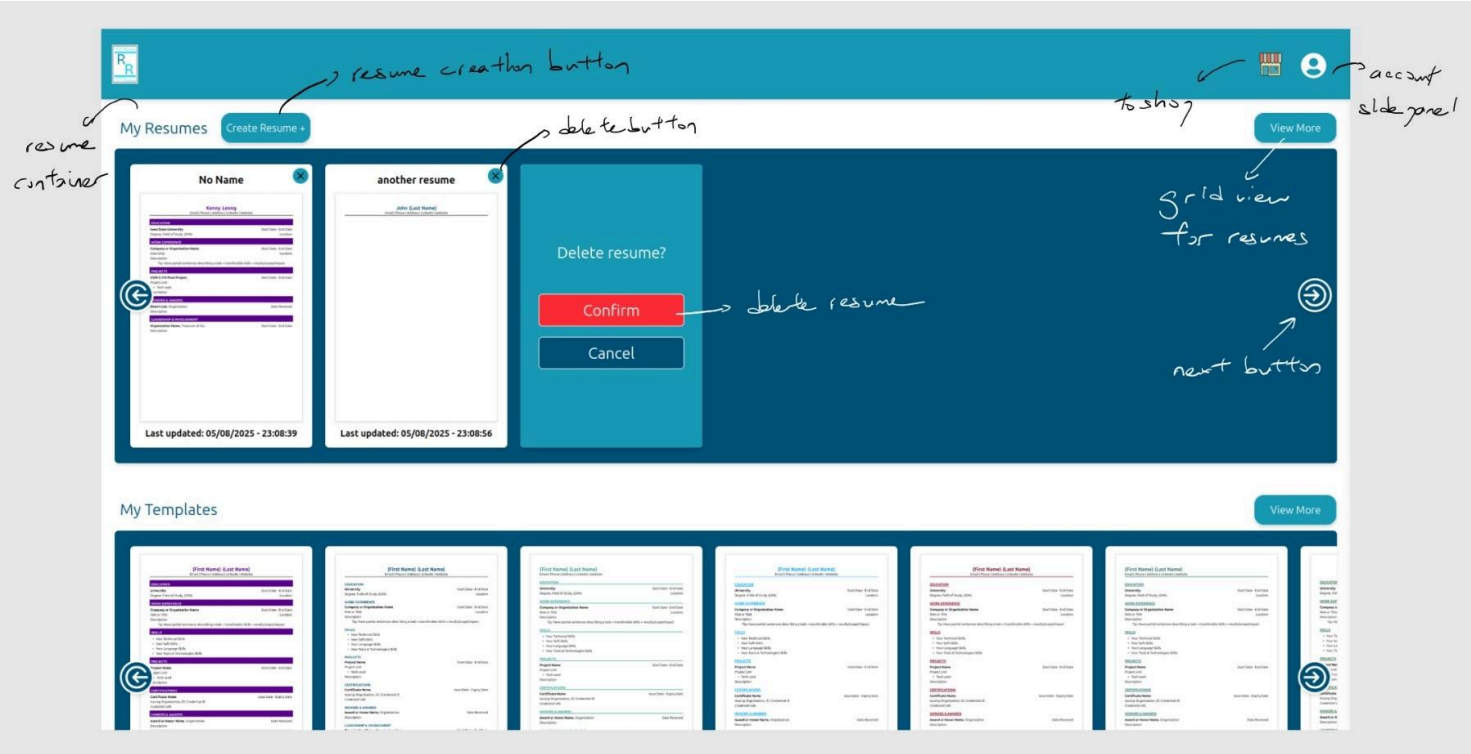
Account sign up & login pipeline



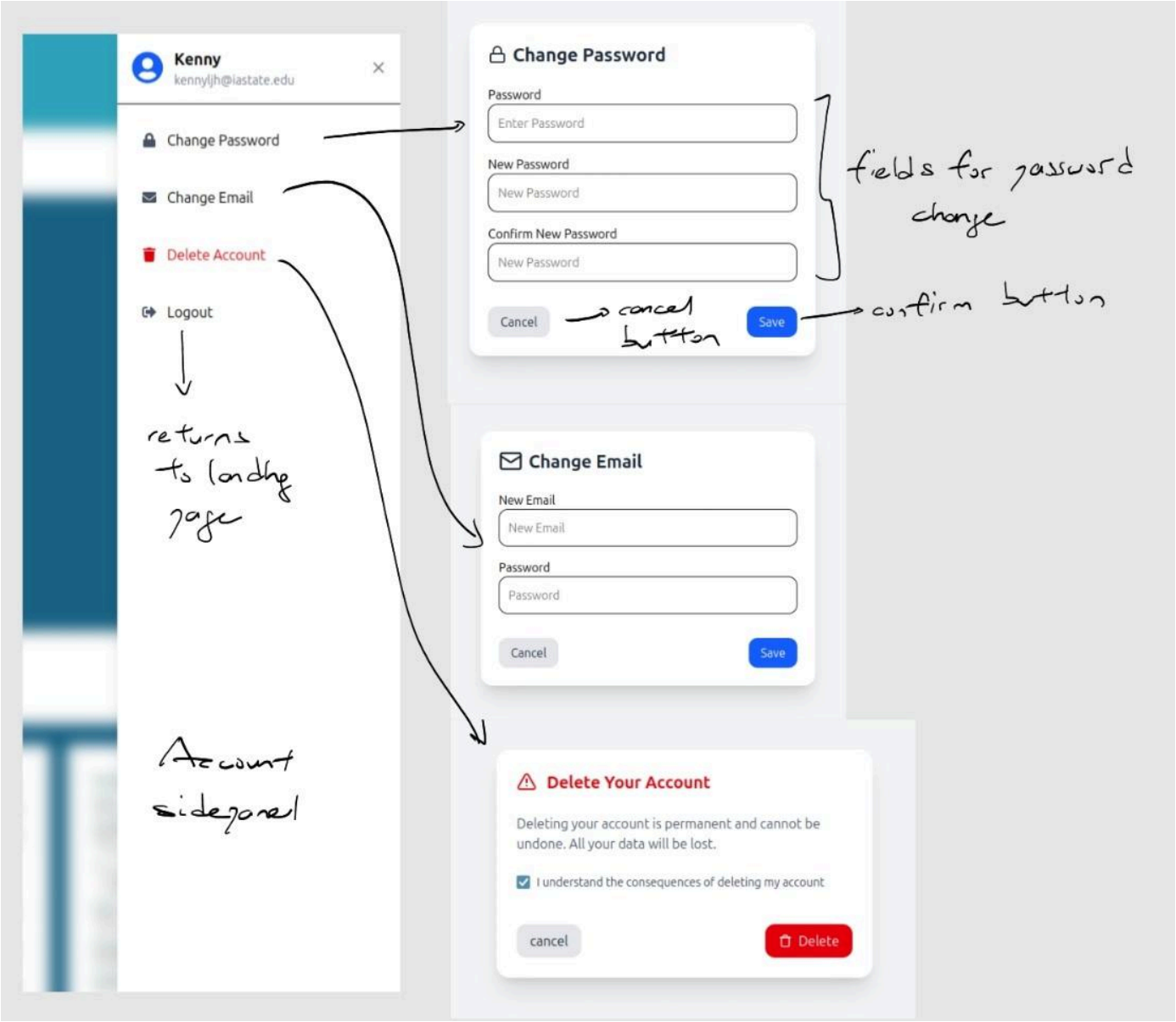
Resume template and assets selection pipeline



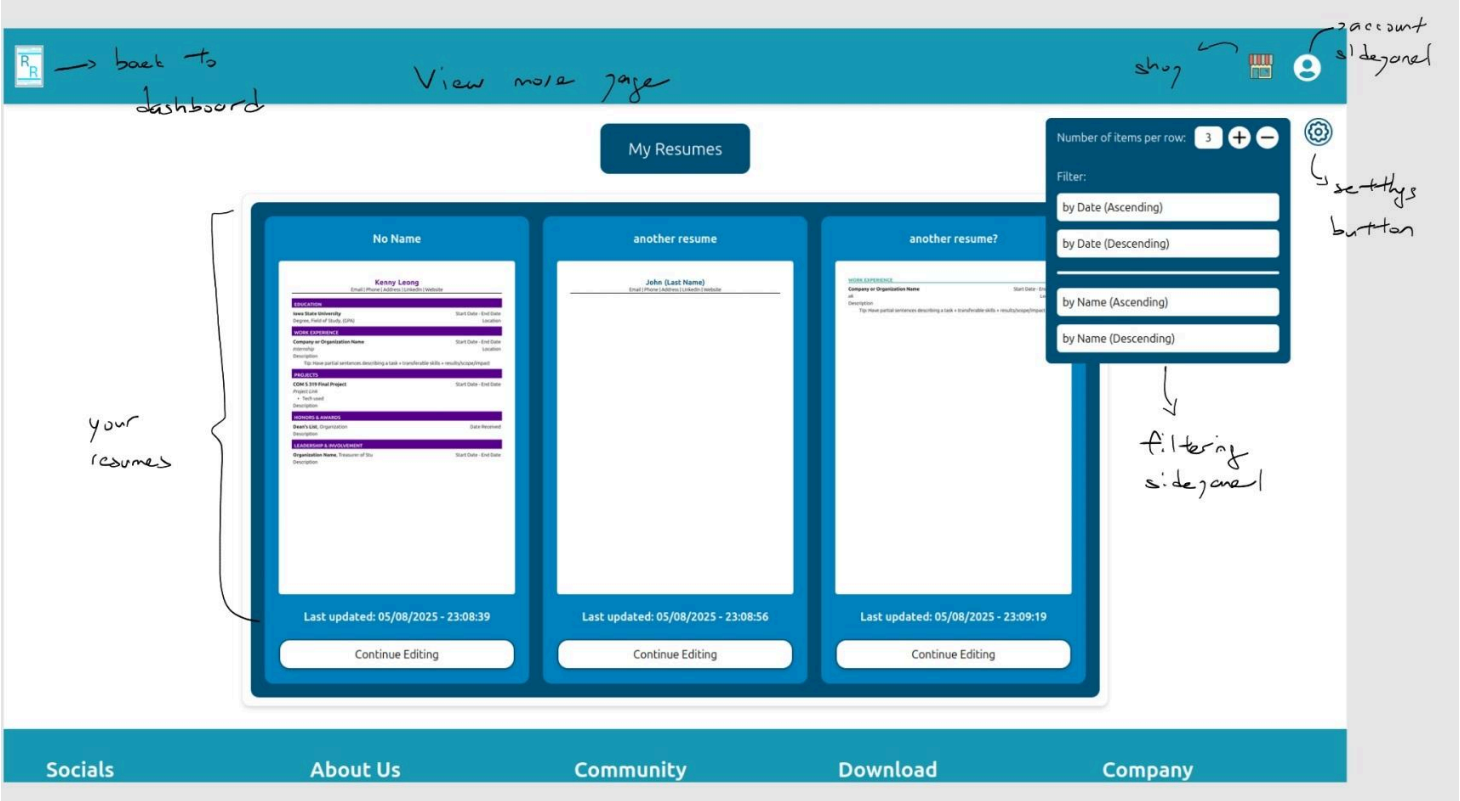
Resume creation page



Dashboard



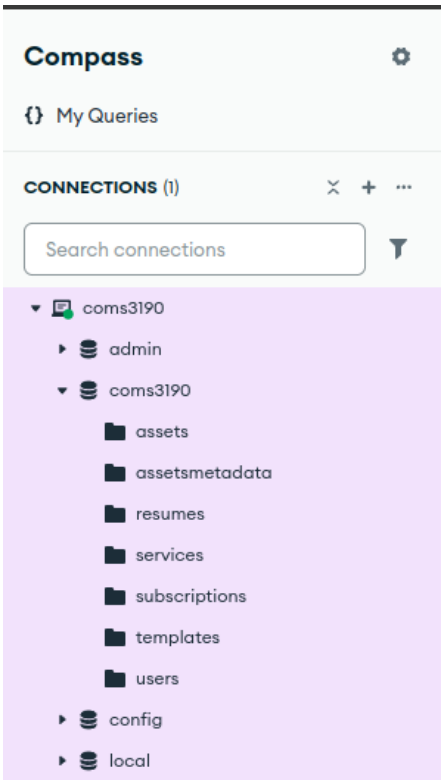
Account side panel popup & account editing options



"View More" page for resumes, templates & assets

6. Installation and Setup Instructions

1) Download a copy of the repository, make sure that the frontend & backend folders are present
2) To set up the frontend:
    a) go to frontend/resume-ready/
    b) run the command: npm install react
    c) run the command: npm run dev
3) To set up the backend:
    a) go to backend/
    b) run the command: npm install
    c) now to set up mongodb:
        i) start up mongodb-compass
        ii) make sure you have the correct structure and all collections:



        iii) go to backend/archived data/
        iv) import all the data into the appropriate collections as specified by their names. Note: if there isn't any archive data for a collection, that collection can be left empty, it will be filled when the frontend starts.

d) now run the command: nodemon server.js

4) If there are issues such as OTP not sending, kill both front and backend with Ctrl+C and restart with the commands: <npm run dev> <nodemon server.js>

7. Contribution Overview

Kenny Jia Hui Leong	Mani Raj Rejinthala
Real-time resume editor	User login & signup
Dynamic dashboard	Account creation verification using NodeMailer
Ability to download resumes with modules html2canvas and jspdf	Enforced Authentication context for webpages with sensitive and user specific details.
Products pages	Products pages
Ability to save resumes to the database, delete them, and reload from them on the dashboard.	Account modifications, like change email, password, and deletion.
Ported midterm project	Ported midterm project
Filtering panel for owned resumes, templates, and assets.	Selection of resources, like templates and assets, for resume creation.

8. Challenges Faced

The development of the resume creation page was one of the biggest challenges as both authors of this project had no prior experience in developing a creational/creativity tool that dynamically updates according to user input. This first barrier of difficulty was solved by extensively planning out the layout of the creation page, making heavy use of containers and flexboxes, and theorizing how the layout should behave accordingly based on user data or preferences. Canva and Overleaf were heavily referenced to develop a “good” layout that does not hinder the user when it comes to resume creation, but instead promotes creativity and efficiency.

The second barrier of difficulty was providing users the ability to download their resumes. This was solved by researching and making use of node modules html2canvas and jspdf, where we pass the id of the container which has the resume. Once the DOM is loaded, html is then retrieved and translated into a base 64 encoded PNG, and downloaded to the user’s device. The PDF works the same way by retrieving the id container, then translates it to a PDF for download.

The third barrier of difficulty was the need to uniquely associate and identify user inputs to correctly render the preview section. To accomplish this, we came up with the idea of using IDs that follow a predetermined syntax, e.g. PD-FN-1, where PD is the asset (Personal Details), FN is the subasset (First Name), and 1 is the order of rendering. Using this, we were able to correctly identify all user inputs, as well as save them to the database. These unique IDs are later used to reload the resume creation page if users choose to continue their work.

9. Final Reflections

Through this project, we gained hands-on experience and confidence in full-stack development, particularly mastering key technologies like React, Node.js, and Express. We learned how to design scalable architectures by separating concerns across frontend and backend modules, applying principles like low coupling and high cohesion using modular approach. Working with MongoDB, we explored schema design and efficient data retrieval, especially for dynamic content like templates and user resumes. Implementing features like real-time preview, secure authentication with OTP, and PDF/image export deepened our understanding of both user experience and performance optimization. This project not only solidified our technical skills but also taught us how to build production-ready web applications with a focus on usability, scalability, and security.