|  |  |
| --- | --- |
| Student Name: | Kisan Rai |
| Student Number: | C0910925 |
| Assignment name: | MERN Final Assignment |
| Instructor name: | Prof. Terry D.Silva |
| Submission Date | 03rd -Dec-2024 |

A blue text on a black background

Description automatically generated

CSD-3102 Full Stack JavaScript

Table of Contents

[1. Introduction 1](#_Toc183722899)

[2. Project Objectives 1](#_Toc183722900)

[3. Technologies Used 2](#_Toc183722901)

[3.1 Frontend: 2](#_Toc183722902)

[ React.js: 2](#_Toc183722903)

[ Axios: 2](#_Toc183722904)

[ ag-Grid: 2](#_Toc183722905)

[ Bootstrap/W3CSS 2](#_Toc183722906)

[ Evergreen UI Components: 2](#_Toc183722907)

[3.2. Backend: 2](#_Toc183722908)

[ Node.js: 2](#_Toc183722909)

[ Express.js: 2](#_Toc183722910)

[ MongoDB: 2](#_Toc183722911)

[ Mongoose: 2](#_Toc183722912)

[ Login.js: 3](#_Toc183722913)

[4. Application Features 3](#_Toc183722914)

[4.1. User Authentication (Login Page) 3](#_Toc183722915)

[4.2. User Management (Add, Update, Delete) 3](#_Toc183722916)

[4.3. View Users (Data Grid) 3](#_Toc183722917)

[4. Responsive UI 4](#_Toc183722918)

[5. Backend Architecture 4](#_Toc183722919)

[6. Populating MongoDB Database with Random Data 4](#_Toc183722920)

[7. Challenges Faced 5](#_Toc183722921)

[8. Conclusion 6](#_Toc183722922)

[9. Program Instructions: 7](#_Toc183722923)

[10 To run the Project 8](#_Toc183722924)

[10.1 Backend server 8](#_Toc183722925)

[10.2 Fronted server 10](#_Toc183722926)

[11. Login Page 11](#_Toc183722927)

[12. View User page 12](#_Toc183722928)

[13. Add user page 13](#_Toc183722929)

[14. User is added into view user AG grid page. 14](#_Toc183722930)

[15. Rose User Detail information page. 14](#_Toc183722931)

[16. Updating Rose’s information in updateUser page 15](#_Toc183722932)

[17. user information updated from Rose to JISSO information in viewUser page inside the AG grid. 16](#_Toc183722933)

[18. Deleting user information 16](#_Toc183722934)

[17](#_Toc183722935)

[19. MongoDB Atlas 18](#_Toc183722936)

[20. Populating data 19](#_Toc183722937)

# Acknowledgements

I would like to express my sincere gratitude to my professor, Terry D. Silva, for his invaluable support and guidance throughout this project. I deeply appreciate the resources, materials, and extra information provided, which greatly contributed to my understanding and development of the project. Professor Silva’s continuous encouragement and thoughtful feedback helped me navigate challenges and successfully complete this assignment. His expertise and willingness to assist me at every step were instrumental in my learning experience. Thank you for your dedication and support.

# Introduction

The purpose of this project is to build a comprehensive full-stack web application using the **MERN stack**, which includes **MongoDB**, **Express.js**, **React.js**, and **Node.js**. This application is designed to manage user information, providing features for adding, viewing, updating, and deleting user data. As part of the Full Stack JavaScript course for Fall 2024, this project aims to showcase the integration of both front-end and back-end technologies to create a functional and user-friendly application.

The project serves as a practical demonstration of my ability to apply JavaScript in building a complete web application that adheres to real-world business needs. This project involves working with databases, building RESTful APIs, and creating a user interface that is interactive and responsive. In addition to technical implementation, the project also provides an opportunity to focus on best practices, code structure, and documentation.

# 2. Project Objectives

The primary goal of this project is to develop a **MERN** stack application that satisfies several key requirements:

* **User Data Management**: The application should allow users to add, view, update, and delete personal information through a web interface.
* **Front-end Development**: Utilize **React.js** to build a dynamic and responsive interface for interacting with the backend.
* **Back-end Development**: Use **Express.js** to handle HTTP requests and **MongoDB** to store user data.
* **Authentication**: Implement user login functionality to authenticate users before allowing access to sensitive data.
* **Data Display**: Integrate a data grid, using tools such as **ag-Grid**, to present user information in a tabular format.
* **Styling**: Use **Bootstrap/W3CSS** to ensure the application is visually appealing and responsive across different devices.

# 3. Technologies Used

## 3.1 Frontend:

* React.js: The core JavaScript library used to build the user interface of the web application. React's component-based architecture allows for the creation of reusable components, making it easier to manage the UI.
* Axios: A promise-based HTTP client used to send asynchronous requests from the React frontend to the Express backend. It was used to fetch and send data to the server for CRUD operations.
* ag-Grid: A powerful grid library that displays user data in a sortable, filterable, and interactive table format. It supports features like pagination and data sorting out of the box.
* Bootstrap/W3CSS: Used for styling the user interface. Bootstrap provides responsive grid layouts and components like forms, buttons, and navigation bars, while W3CSS was used for additional visual styling to ensure a clean and modern look.
* Evergreen UI Components: Optional third-party React components used to enhance the user experience, providing pre-designed elements for forms, buttons, and dialogs.

## 3.2. Backend:

* Node.js: A JavaScript runtime environment used to build the backend of the application. Node.js allows for asynchronous processing, making it ideal for handling concurrent requests in web applications.
* Express.js: A minimal and flexible web application framework for Node.js that provides a set of tools to build APIs. Express was used to handle the routing, manage HTTP requests, and interact with the MongoDB database.
* MongoDB: A NoSQL database used to store user data in a flexible JSON format. MongoDB was chosen because of its ability to scale and handle large volumes of unstructured data efficiently.
* Mongoose: An Object Data Modeling (ODM) library for MongoDB and Node.js. Mongoose was used to define the schema for the user data and interact with the database.
* Login.js: A library used to hash passwords securely, providing an extra layer of protection for sensitive user information.

# 4. Application Features

## 4.1. User Authentication (Login Page)

The login page serves as the entry point to the application. It requires users to authenticate themselves using a username {‘**test@estexample.com’**} and password{‘**password123**’}. Once authenticated, users can access the application’s core features such as adding, updating, and viewing user data. This feature ensures that only authorized individuals can modify user information.

## 4.2. User Management (Add, Update, Delete)

The core functionality of the application revolves around user management. There are three key actions:

* **Add User**: Users can fill in a form with their personal details such as first name, last name, date of birth, email, phone number, address, and user notes. This data is sent to the backend where a new record is created in the MongoDB database.
* **Update User**: Users can select an existing record to view and update. The form is pre-populated with the current details, and changes can be saved by submitting the updated form.
* **Delete User**: Users can delete a user’s data from the system. This action permanently removes the record from the database.

## 4.3. View Users (Data Grid)

The application displays a list of all users stored in the MongoDB database. This data is presented in a table format using **ag-Grid**. The grid allows users to:

* Sort and visualize records.
* View details of each user by selecting a record.
* Navigate between pages of data, making it easier to manage large amounts of user information.

# 4. Responsive UI

The application is fully responsive, meaning that it adapts to different screen sizes. This is achieved using **Bootstrap/W3CSS**, which ensures that the UI elements adjust accordingly whether the application is accessed on a desktop, tablet, or mobile device.

# 5. Backend Architecture

The backend of the application is responsible for handling HTTP requests from the frontend. It provides the necessary API endpoints for performing CRUD operations on user data. The backend architecture follows the **Model-View-Controller (MVC)** pattern, where:

* **Model**: Defines the structure of the user data in MongoDB.
* **View**: Serves as the client-side UI for interacting with the user data.
* **Controller**: Handles the logic for performing CRUD operations, such as adding, updating, and deleting users.

The server is set up using **Express.js**, and the database connection is established with **MongoDB** through **Mongoose**. The API endpoints interact with the database to retrieve, insert, update, or delete user records.

# 6. Populating MongoDB Database with Random Data

As part of the project requirements, the MongoDB database needed to be populated with at least 10 random records of user data. To ensure that the application would function with actual data, I utilized a method to automatically populate the database with placeholder data using a data generation tool.

**Data Population Process:**

For the population of random data, I used one of the following tools:

* **JSONPlaceholder**: A free API that provides fake data for testing and prototyping. This API was used to generate random user records that contained fields such as first name, last name, date of birth, address, phone number, email, and notes.
* **Faker.js**: This tool was used to generate random data directly in the backend to insert into MongoDB. Faker.js allows for the creation of realistic-looking names, addresses, dates, and other user-related information.
* **PopulateData.json**: To streamline the process, I created a **populateData.json** file in the backend directory. This file contained a set of random user data entries in JSON format. The file could be used to populate the MongoDB database when the application was initialized or when a manual trigger was required to populate the database with dummy data.

# 7. Challenges Faced

**1. Data Synchronization Between Frontend and Backend**

One of the challenges was ensuring that the data between the frontend and backend remained in sync, particularly when adding, updating, or deleting user records. This was resolved by using **Axios** to send requests to the backend and using **React’s state management** (via useState and useEffect) to keep the frontend updated in real-time.

**2. Managing Complex Forms in React**

Another challenge was managing the state of complex forms in React. Each form (for adding or updating a user) contained multiple fields, and ensuring that the data entered was correctly captured, validated, and sent to the backend was a significant challenge. React’s controlled components and state management via **useState** hooks helped in organizing and handling form data effectively.

**3. Designing the User Interface**

Creating a user-friendly interface that was both visually appealing and easy to navigate was an important aspect of the project. While building the UI, I focused on clear navigation, easy form submission, and responsiveness across devices. I used **Bootstrap** to help with styling and layout, but also customized it to fit the unique requirements of the project.

# 8. Conclusion

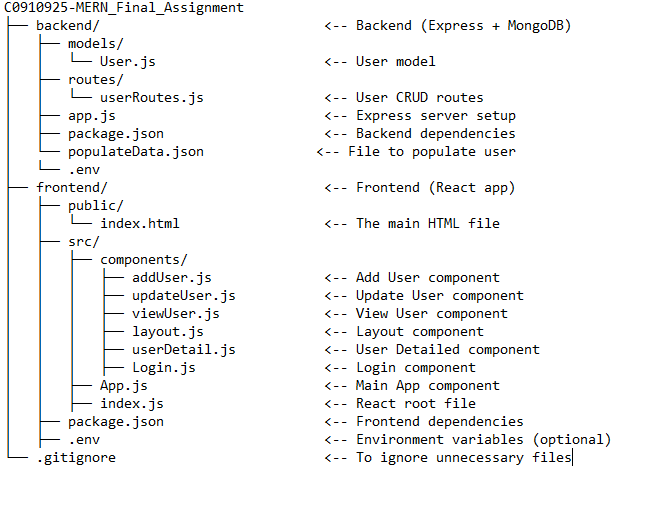
In conclusion, this full-stack JavaScript project successfully meets the requirements and demonstrates the integration of various technologies to create a functional web application. The application allows users to manage their data efficiently with features for adding, viewing, updating, and deleting user records. By using **React.js** for the frontend and **Express.js** with **MongoDB** for the backend, the project showcases the power of the **MERN stack** in building modern web applications.

This project helped me enhance my understanding of full-stack development, particularly in areas such as asynchronous programming, database management, and UI design. Despite facing challenges such as data synchronization, form management, and user authentication, I was able to find effective solutions using the tools and libraries available within the JavaScript ecosystem.

The application is fully functional, user-friendly, and easily extendable, making it a strong example of my capability to develop full-stack applications. This project not only fulfilled the assignment requirements but also provided valuable learning experiences that I can apply in future development projects.

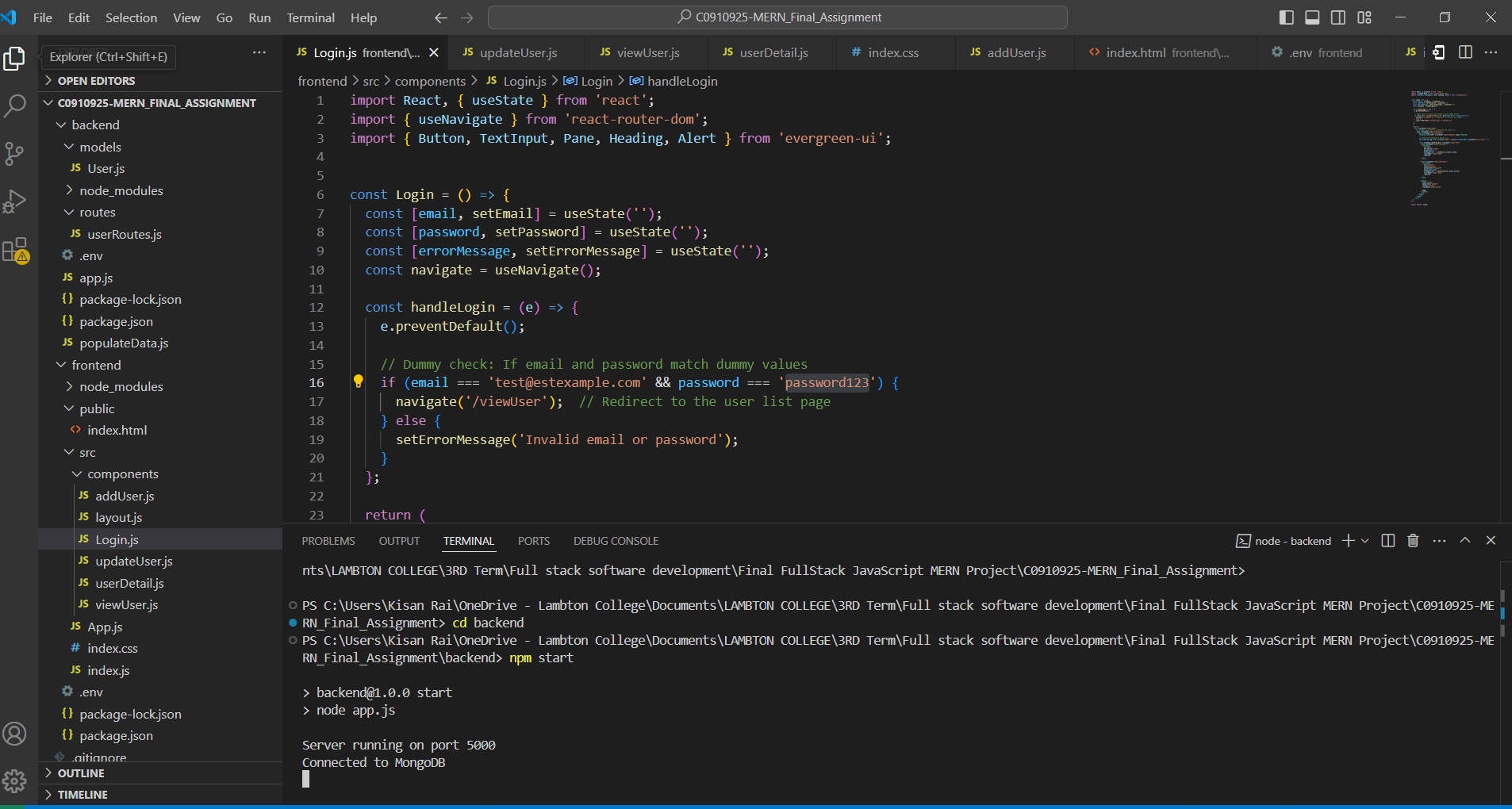
# 9. Program Instructions:

Here is what my entire project directory looks like, with the project divided into two parts: **backend** and **frontend**.



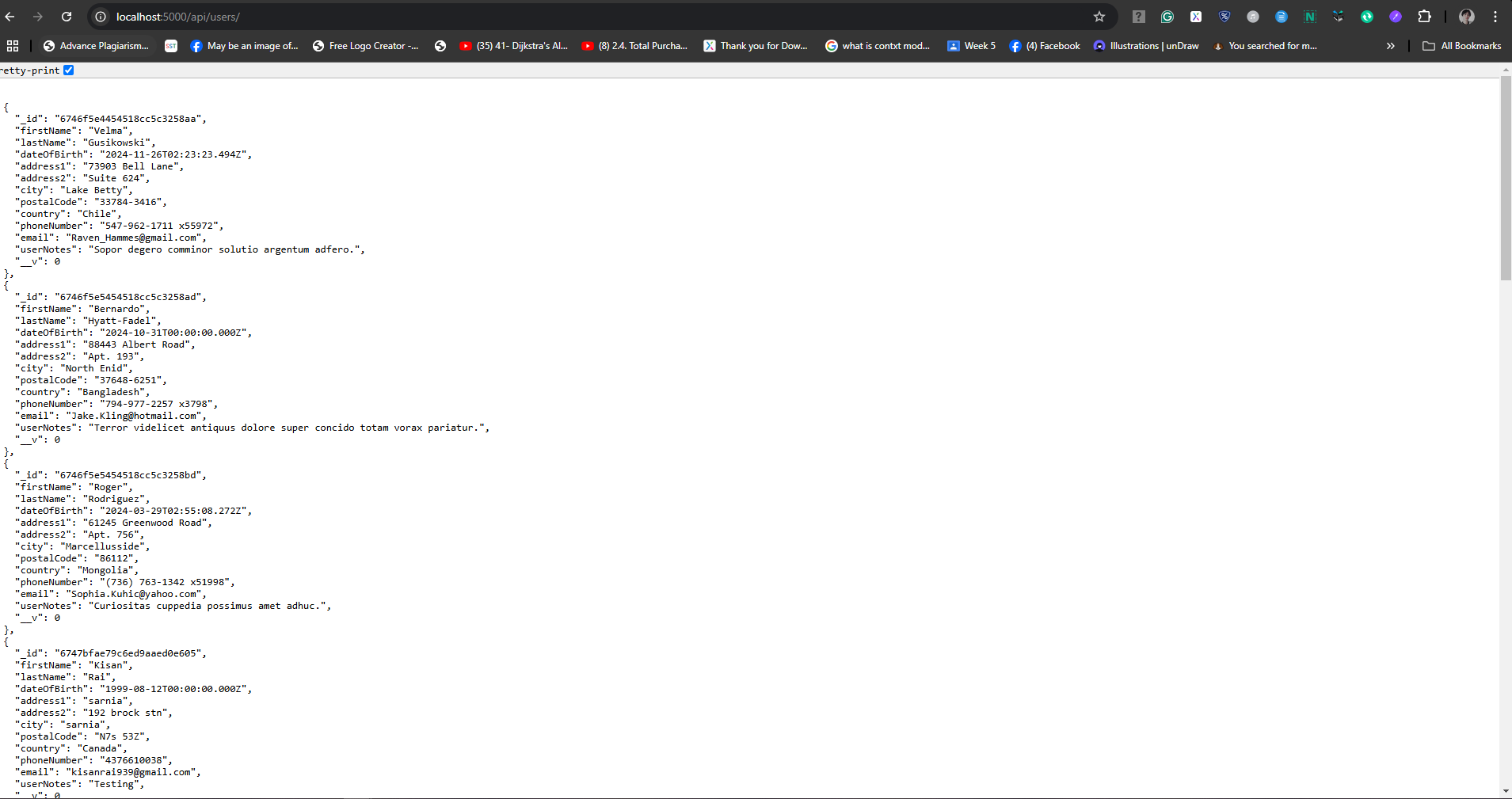
# 10 To run the Project

## 10.1 Backend server

To run the backend server, first open the terminal and navigate to the backend directory by typing **[cd backend]** and hitting Enter. This will change the directory to the backend folder. Then, type **[npm start]** to run the backend server.

To check the backend server and view the user information, use the following URL to see all the populated user data in JSON format:

<http://localhost:5000/api/users>

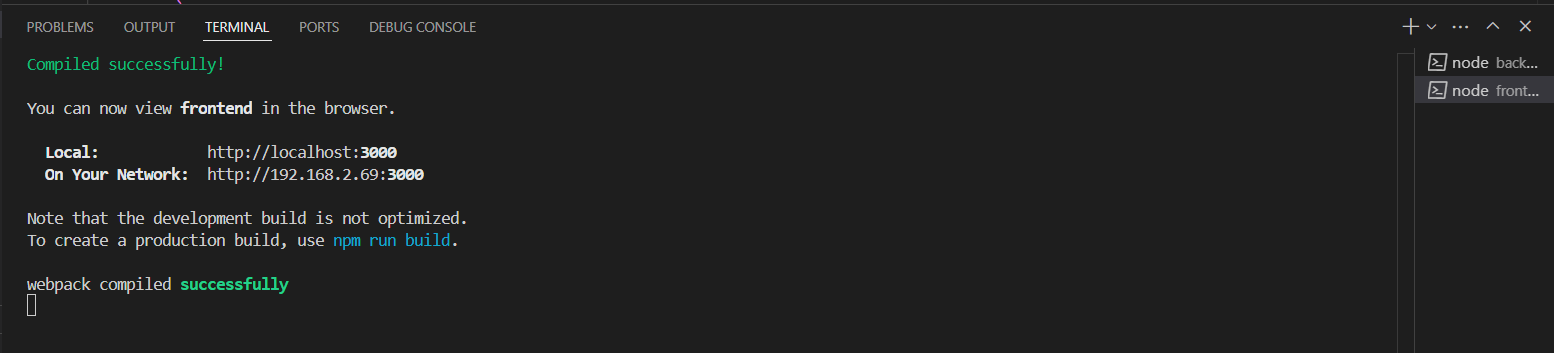


As we can see the user information in JSON format from the backend server in the screenshot above, you can view specific user information by using the user ID. For example, to see the details of a specific user, use the following URL:

<http://localhost:5000/api/users/674900dd0f375e1fbe7aea90>

## 10.2 Fronted server

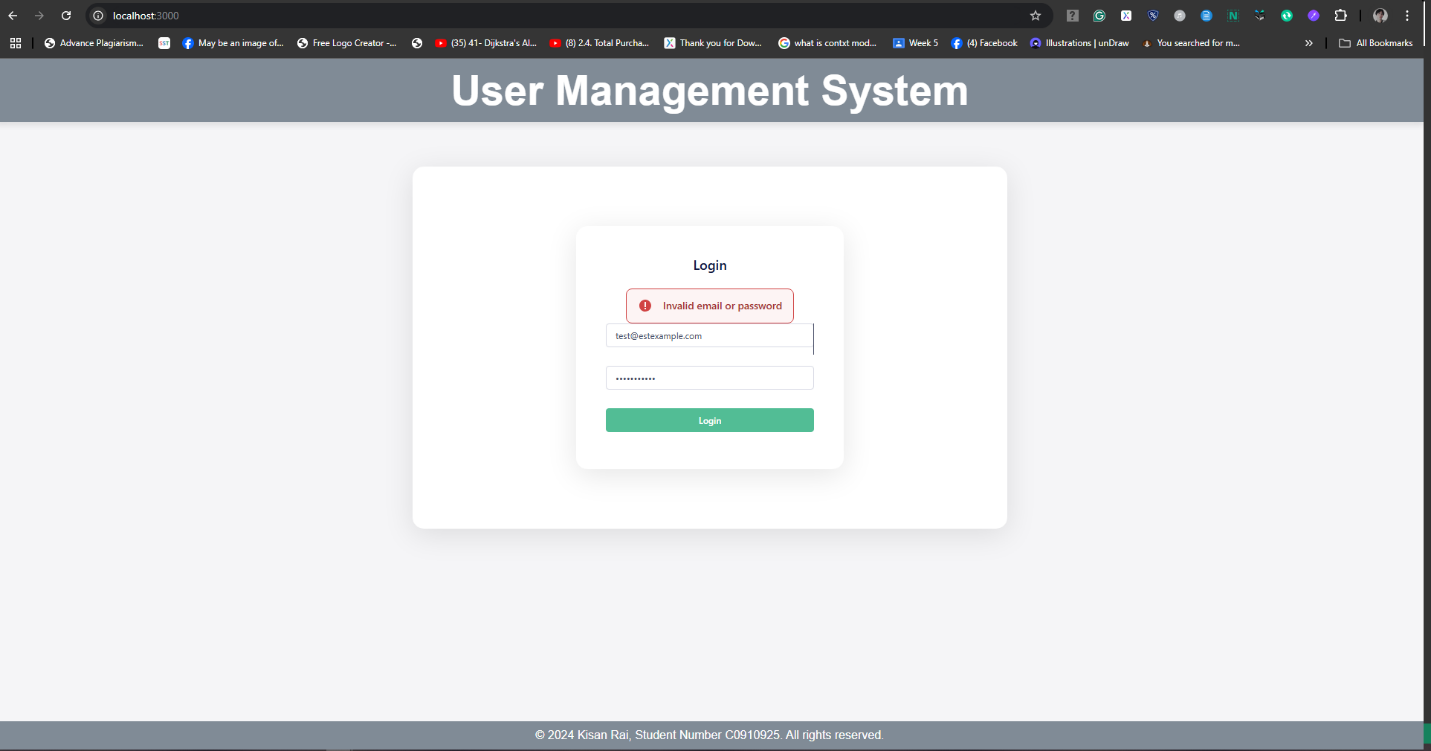
To run the frontend server, open another terminal and navigate to the frontend directory by typing **[cd fronted]** and hitting Enter. This will change the directory to the front-end folder. Then, type **[npm start]** to run the fronted reactJs server

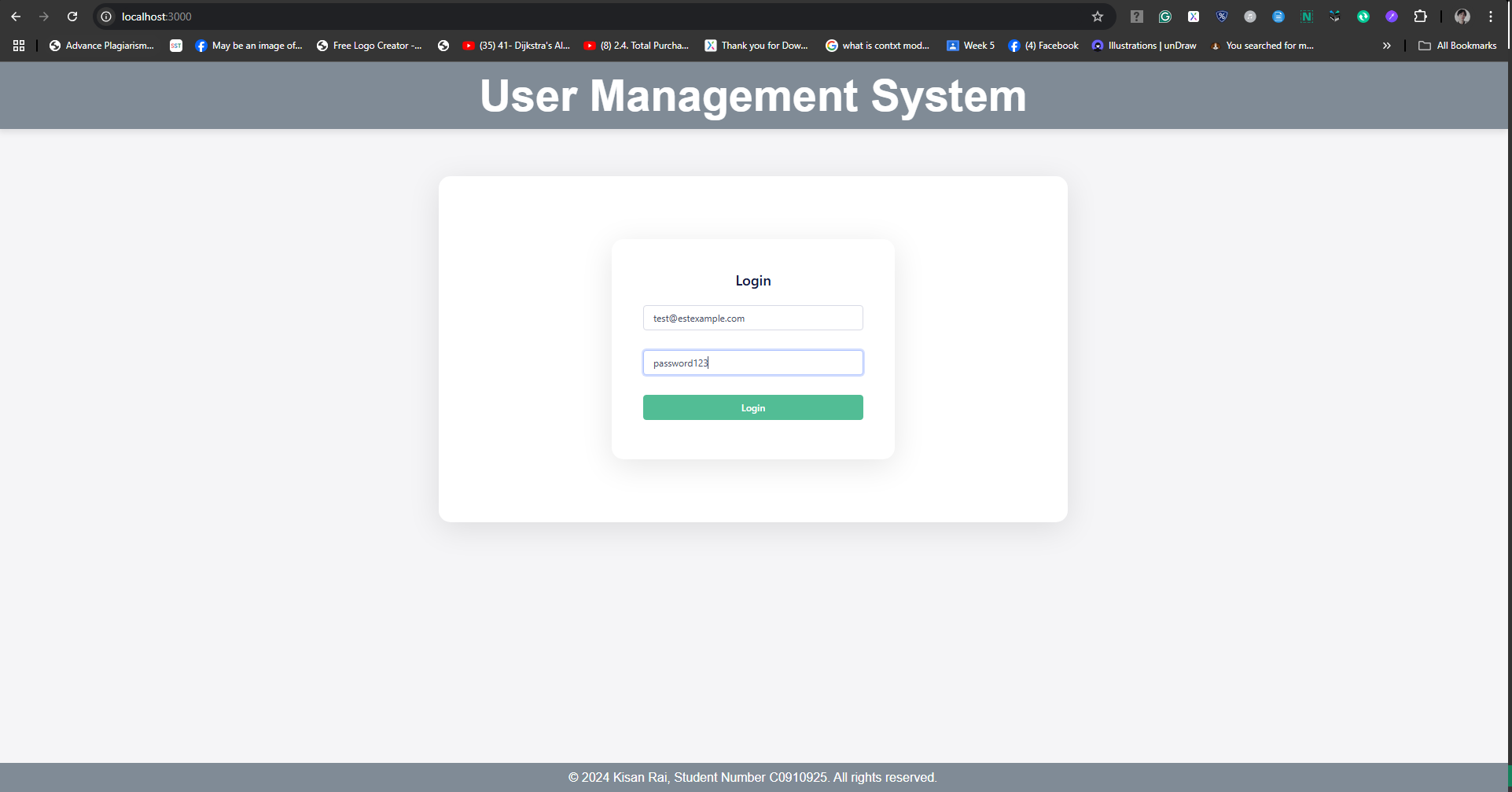


It will open in your browser at **http://localhost:3000/**, which is the default login page of my application.

# 11. Login Page

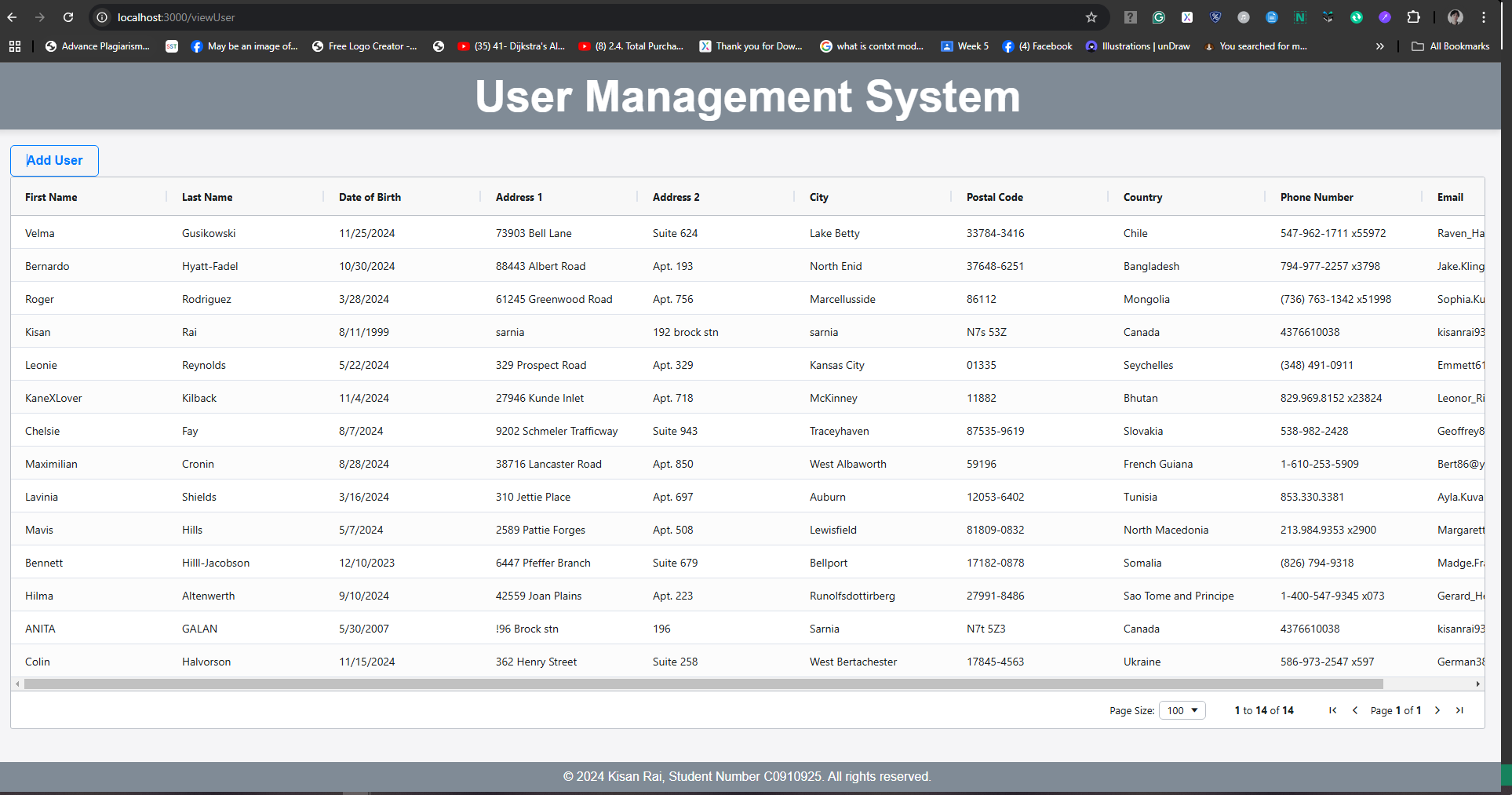
Use the dummy email**: test@estexample.com** and password: **password123** for login credentials. If the credentials are incorrect, an error message will pop up. If the login is successful, it will redirect you to the "View User" page.

For a invalid login



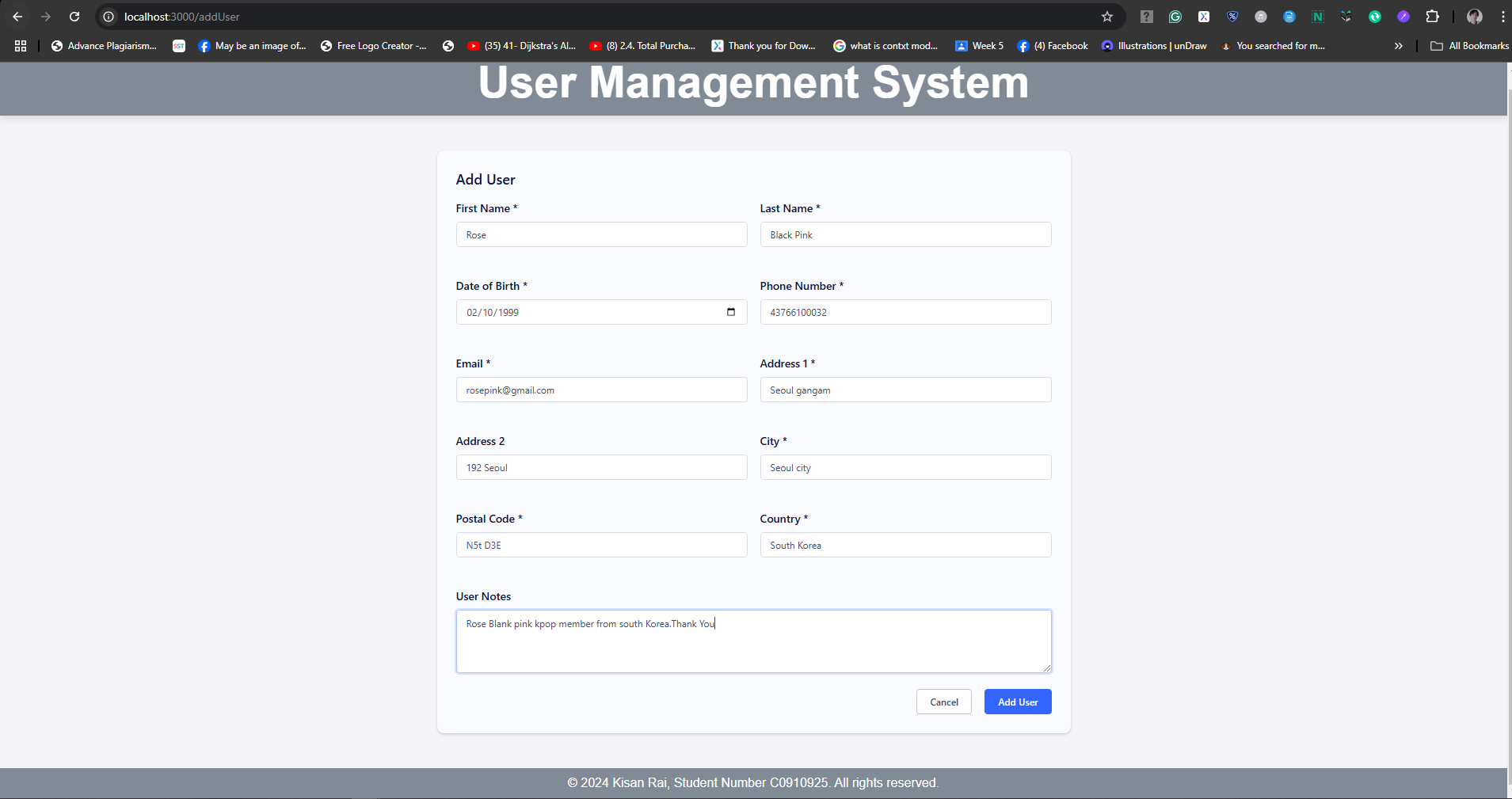
# 12. View User page

Once the login is successful, you will be redirected to the main page, which is the "View User" page. On this page, you will be able to see all the user information displayed in an AG Grid. The data is fetched from MongoDB and shown in the AG Grid on the "View User" page.



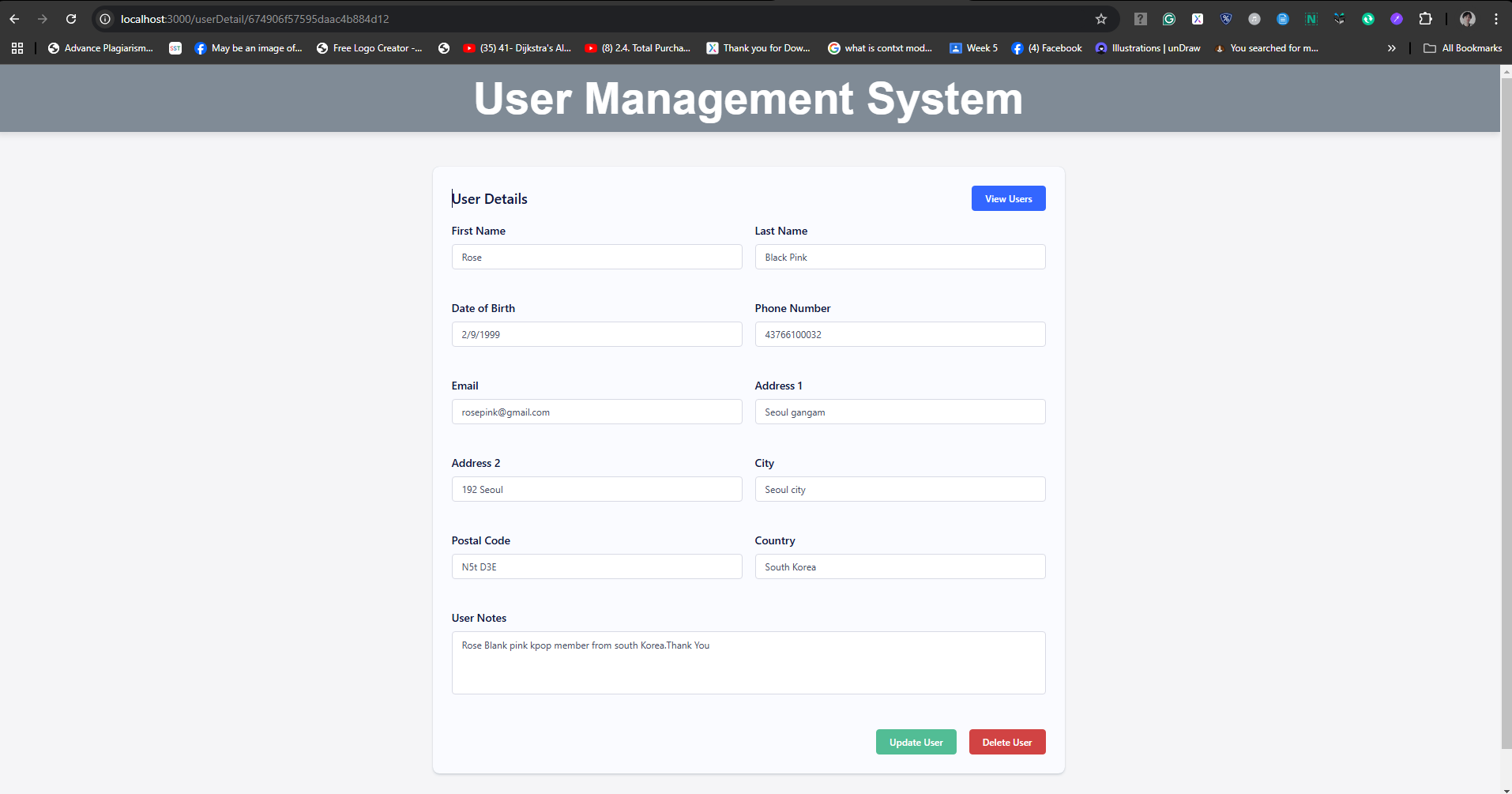
# 13. Add user page

Here, I am adding user information for a user named **Rose** on the "Add User" page.

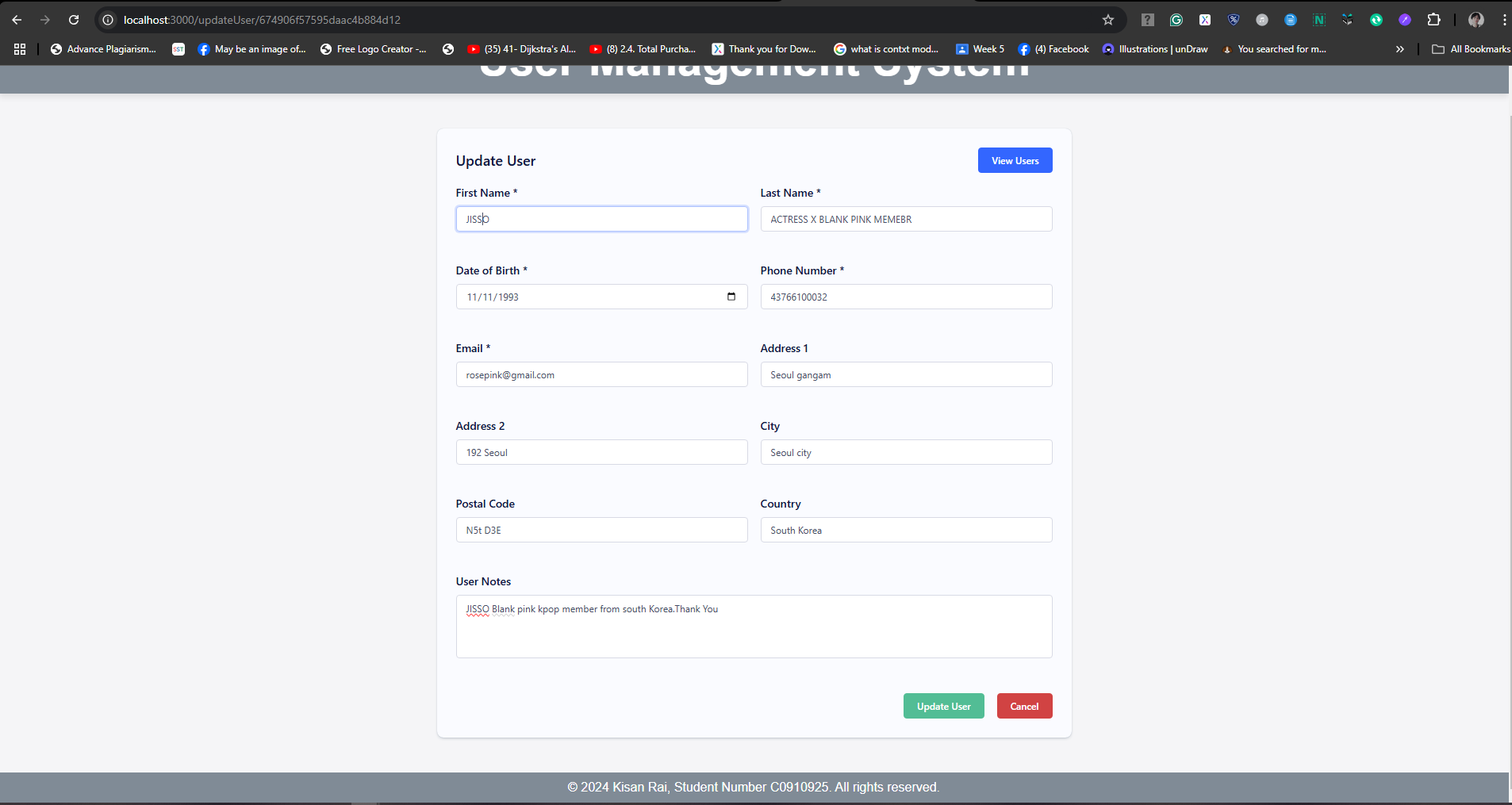


# 14. User is added into view user AG grid page.

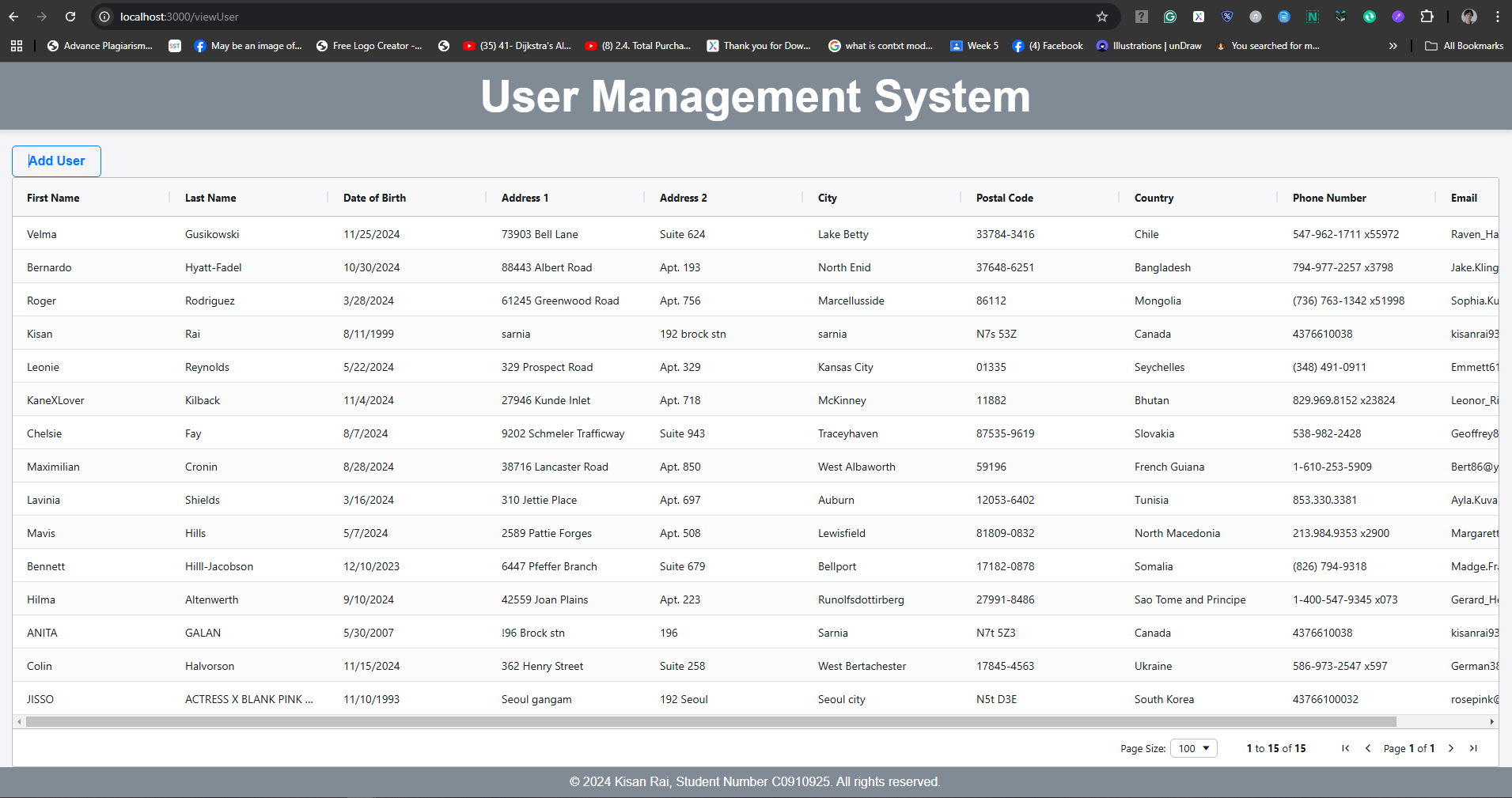
# 15. Rose User Detail information page.



# 16. Updating Rose’s information in updateUser page



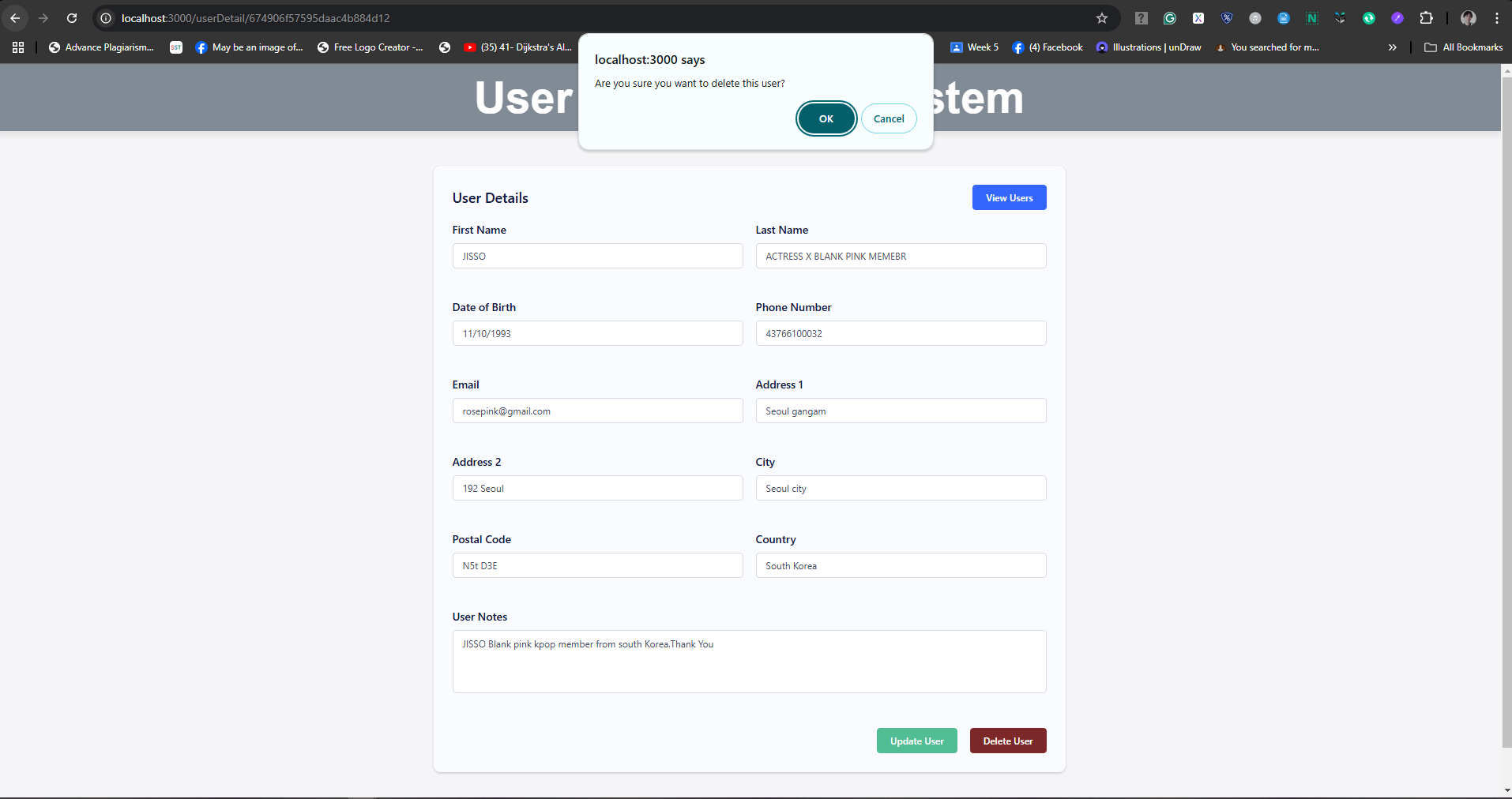
# 17. user information updated from Rose to JISSO information in viewUser page inside the AG grid.



A screenshot of a computer

Description automatically generated

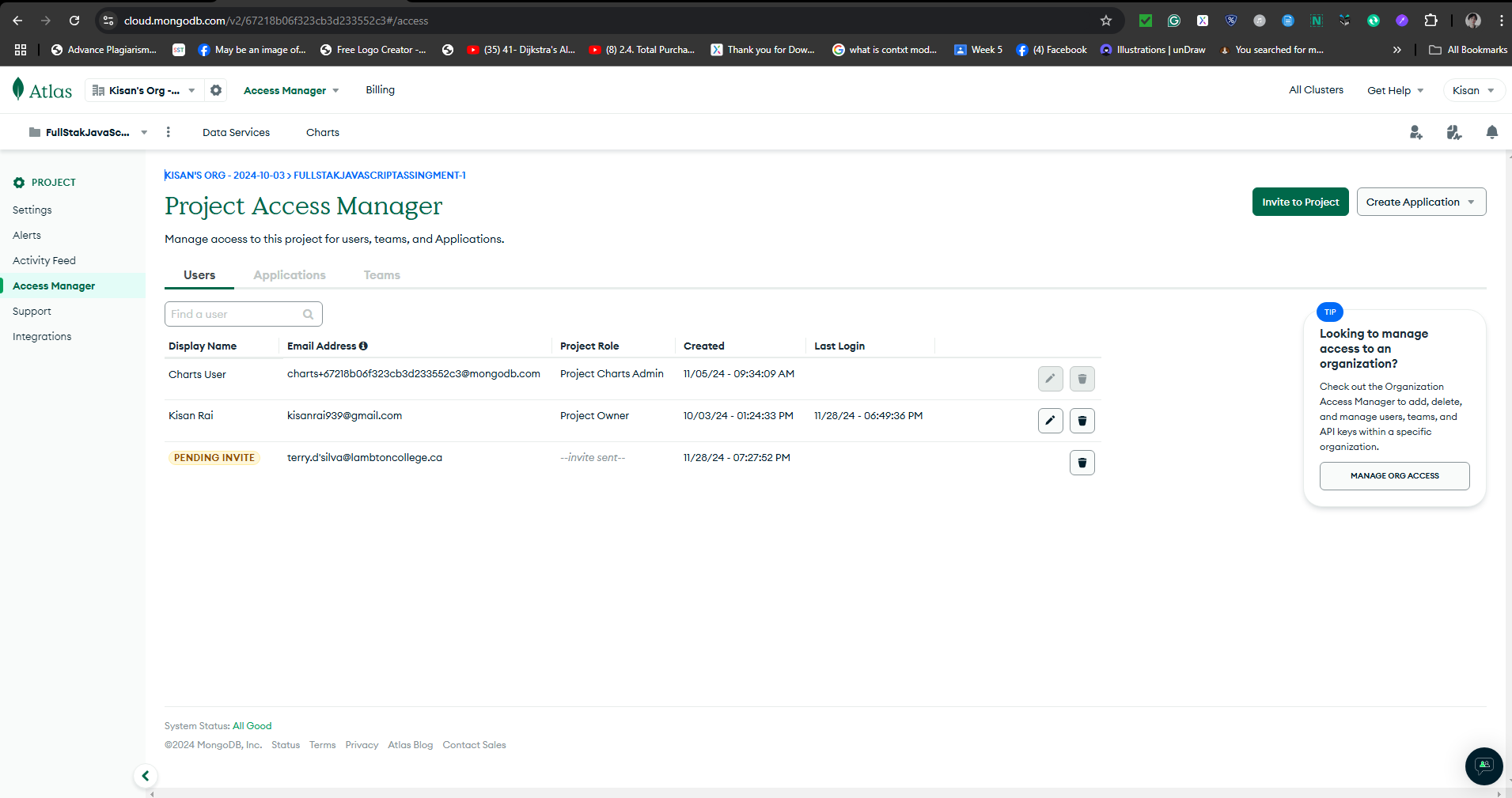
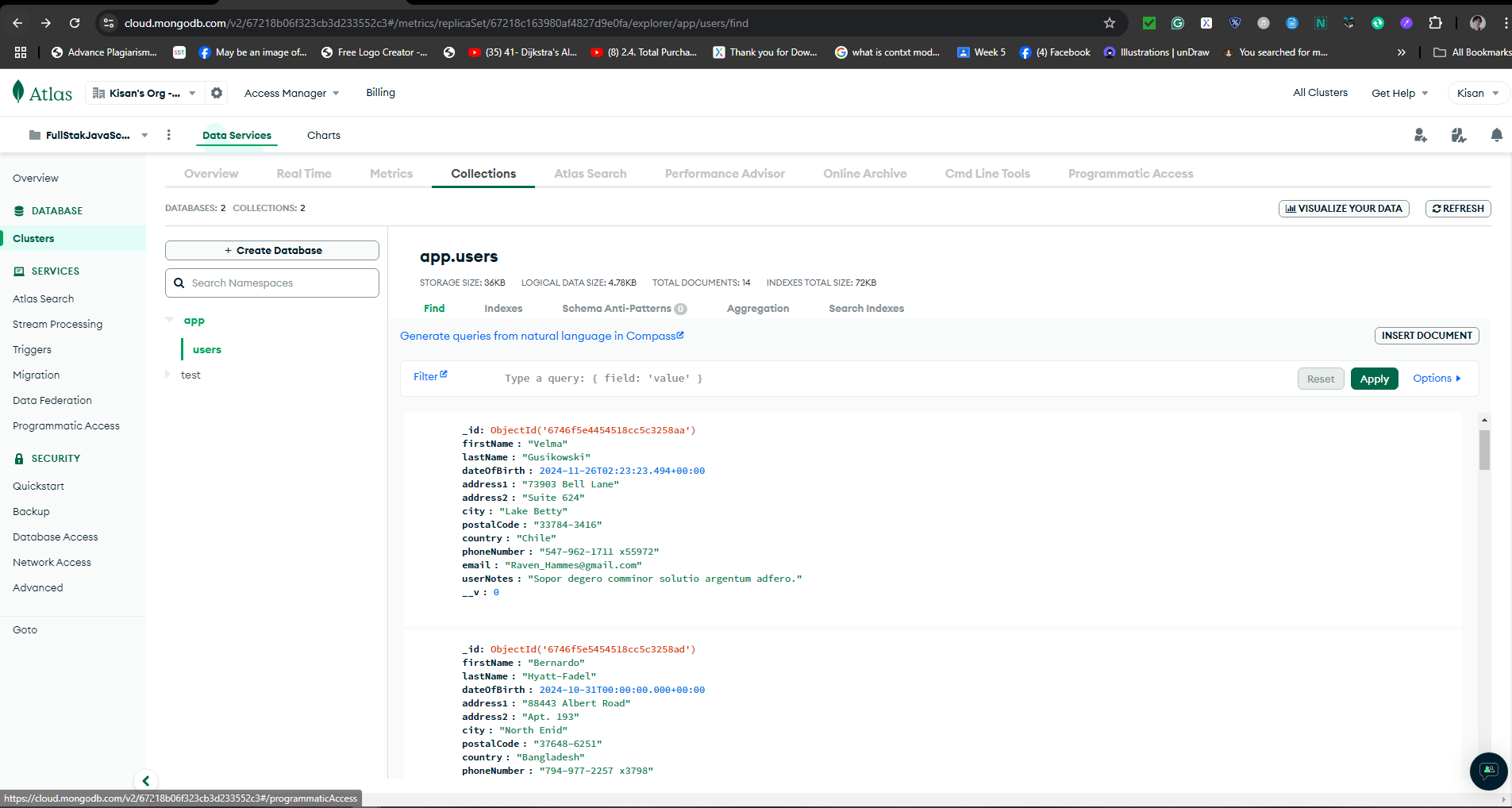
# 18. Deleting user information



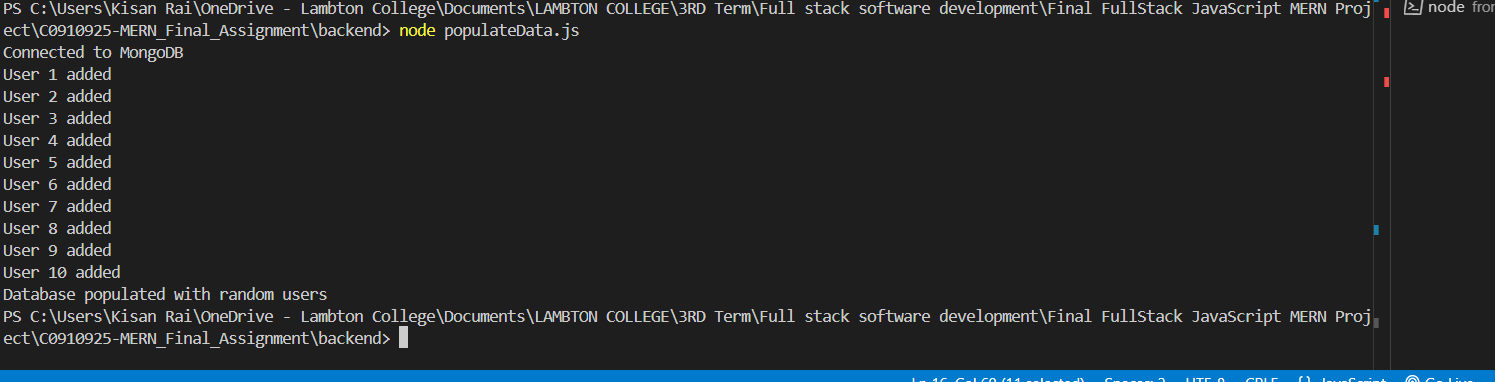
# 

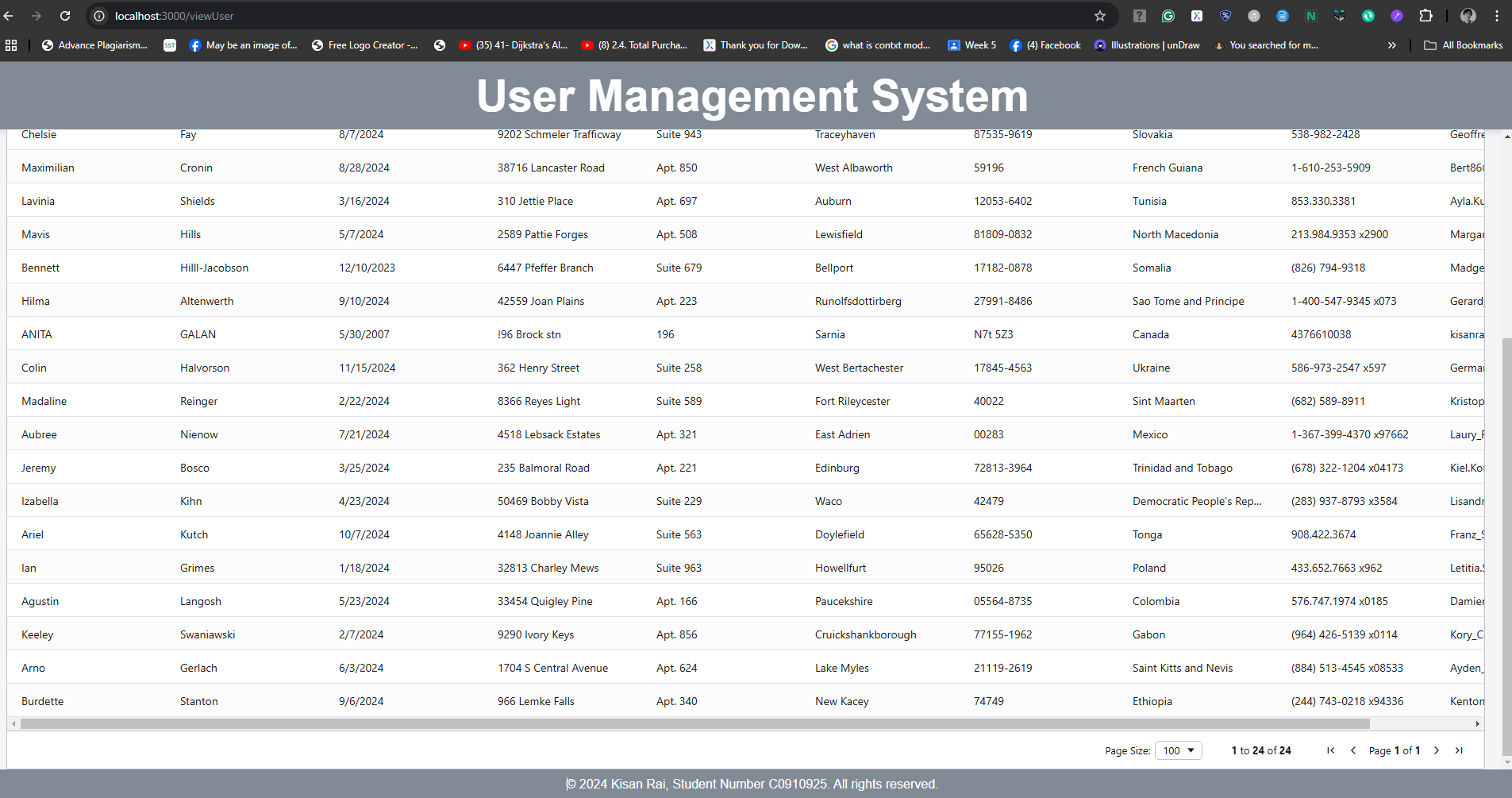
# 19. MongoDB Atlas

I have shared my MongoDB project with you. Please let me know if you are able to access my project database, as you can see in the highlighted red box in the screenshot above.



# 20. Populating data

I have populated 10 random user data into the backend server, which were added as 10 users. This data was fetched into MongoDB Atlas and is now reflected on the "View User" page in the AG Grid.



There were 14 records before, and after populating 10 more records, the total number of records has increased to 24.

Also reflected into MongoDB Atlas

