VIREC PROJECT – NESTQUEST

Problem Statement:

The aim of this application is to provide users an experience where they can browse available properties across the subcontinent and explore more details accordingly. This application was to be developed using visual assists like maps and tables which enhance the user experience. While user can go through the product, under the hood, an RBAC enforced dashboard was to be developed which can control all the CRUD operations including providing users required privileges.
This extensive application was to be developed using a tech stack of our own choice and move with the most secure line of processing providing users with the needful. With an emphasis on security, the application had to be seamless enough for the admin and user to interact with the application with the most ease possible.

Tech Stack Implemented:

At the core of this application lies the JavaScript framework called Next.js. The UI has been implemented using ShadCn which works perfectly with Next.js. The choice of database has been the NoSql database MongoDb. For queries and changes to the database has been implemented using Prisma. A particular attention to detail has been made using Next/auth which prevents any kind of API Tapping or Data Manipulation attacks.

1. **Next.js**: It is a popular JavaScript framework that provides powerful features for building React-based applications. It enables features like server-side rendering, static site generation, and API routes, making it ideal for building scalable web applications. The framework also supports the app directory structure, which enhances routing, modularity, and performance in modern web applications.

2. **ShadCn**: Shadcn is a UI library built to work seamlessly with Next.js. It provides reusable components and design patterns, enabling faster development and consistent UI across the application. Its flexibility and integration with Tailwind CSS make it a great choice for building responsive, modern interfaces.

3. **MongoDB**: MongoDB is a NoSQL database that stores data in a flexible, JSON-like format. It allows for easy scalability and efficient handling of large volumes of unstructured or semi-structured data. MongoDB's flexibility makes it well-suited for applications that require dynamic schema design, such as the one being developed here.

4. **Prisma**: Prisma is a powerful database toolkit that provides an ORM (Object Relational Mapping) layer for working with databases like MongoDB and PostgreSQL.

It simplifies database queries by providing an easy-to-use, type-safe API, reducing the chances of errors and improving developer productivity. Prisma is used to handle data modeling, database migrations, and query building in the backend of this application.

5. **NextAuth.js:** It is flexible authentication library for Next.js applications. It enables secure authentication workflows, including OAuth, email/password login, and social logins. This library ensures that only authorized users have access to certain parts of the application and is crucial for preventing unauthorized access, API tapping, or data manipulation attacks.

Architectural Designs to be noted:

Next.js:
Next.js is the backbone of this application due to its robust feature set and ability to provide both server-side rendering (SSR) and static site generation (SSG). These features are essential for ensuring fast load times and SEO optimization. Furthermore, Next.js supports a file-based routing system that is simple to implement and maintain, allowing developers to focus on business logic rather than configuration.
The framework's integration with React provides a flexible UI structure that pairs well with Shadcn, and the powerful routing system works perfectly with MongoDB-based API routes. Next.js API routes serve as an efficient bridge between the frontend and the database, handling requests securely and efficiently.

ShadCn:
Shadcn was chosen for its ability to quickly build clean, accessible, and customizable UI components. It offers high flexibility, which allows developers to tailor the design to meet specific user needs while maintaining a modern aesthetic. Since it integrates seamlessly with Tailwind CSS (used in the project), it accelerates the styling process and provides consistency across the application.
Shadcn components work harmoniously within the Next.js ecosystem. The design system provided by Shadcn complements the flexibility of Next.js, and the Tailwind integration allows developers to quickly customize and adapt styles. The two tools support a streamlined development process, enabling quick UI iterations.

MongoDB:
MongoDB integrates smoothly with Prisma, which acts as an abstraction layer to streamline interactions with the database. This combination ensures that data models are clean and easily maintainable while optimizing query performance. Additionally, MongoDB's scalability ensures that as the application grows, the database can easily accommodate more data without sacrificing performance.

Prisma:

Prisma works seamlessly with MongoDB through its native MongoDB adapter, allowing for efficient database queries while still providing the benefits of an ORM. It integrates perfectly into the backend of the Next.js application, ensuring clean data management and efficient interaction with the database. Prisma's support for migrations and its type-safe queries ensure long-term maintainability.

NextAuth.js:
NextAuth.js fits perfectly into the Next.js environment, as it was designed specifically to work with this framework. By managing authentication and sessions at the backend, NextAuth.js integrates seamlessly with Next.js API routes, ensuring that sensitive data remains protected. Its compatibility with Prisma further enhances security by securely managing user data in the database and preventing unauthorized access.

Backend:
Next.js itself provides a lot of functionalities which eliminates the need of a server specifically for Database manipulation or etc. This is the reason, there is not a backend framework implemented for the application. Even in production, Vercel, which allows smooth hosting of Next.js applications, provides a server, which further eliminates the need of a backend.

Database Implementation:

The database is structured to support the application's functionality, with two primary collections: users and properties. These collections are designed to handle user authentication, authorization, and property management within the application.

- **Users Collection:**
  The users collection stores personal information about each user, including details such as their name, email, and role (admin or regular user). The role field is particularly important as it determines the level of access and privileges the user has within the application. Users with the admin role will have access to the administrative features of the portal, while regular users will only have access to property browsing and other non-admin functionalities.
  Sample Entry in Database:

```
{
  "_id": ObjectId("1234567890abcdef"),
  "email": "user@example.com",
  "name": "John Doe",
  "role": "admin", // or "user"
  "password": "hashed_password",
  "createdAt": ISODate("2025-02-09T00:00:00Z"),
  "updatedAt": ISODate("2025-02-09T00:00:00Z")
}
```

- **Properties Collection:**

The properties collection stores details about properties listed on the platform. Each document in this collection contains information such as the property's title, description, price, location, and status. The collection is meant to hold various properties that users can browse, with the ability for admins to add, edit, or remove properties.

Sample Entry in Database:

```
{
  "_id": ObjectId("abcdef1234567890"),
  "title": "Luxury Beachfront Apartment",
  "price": 1200000,
  "location": {
    "latitude": "1921.2929",
    "longitude": "-1.39202",
  },
 "owner": “Bryan Young”,
“city”: “Calgary”,
  "createdAt": ISODate("2025-02-09T00:00:00Z"),
  "updatedAt": ISODate("2025-02-09T00:00:00Z")
}
```

- **Relationship between Users and Properties:**
  In this case, the user's collection does not indicate the ownership of properties directly, but rather it tracks the users responsible for creating or managing the listings. The relationship should reflect the distinction between the creator (the user who adds the property to the platform) and the actual owner of the property.

Project Setup:

There are some requirements which are needed before the setup commands for the project. These are:

1. Node.js
2. MongoDB Compass(optional)

Commands to setup the project:

1. Once the project has been forked, cloned or downloaded on the pc, navigate to the root of the project “property-explorer”.
2. Enter the command on the terminal: **npm i**
3. While the package.json should be able to download all the required packages, there are still some additional commands in case dependencies have not been downloaded properly. You can add them by following the below documentation links:
   a. Mongo install
   b. Shadcn
   c. Auth.js
   d. Prisma
4. Added in the compressed folder are 2 .env files, please add them to the root of the project.
5. You can run the command to run the mongodb service**: mongod**

6. Finally run the command to start the development server**: npm run dev**

There is no express server or anything else which is needed for the application to run. Thanks to Next.js architecture where we do not need anything else for backend as it handles a lot of processes on its own without the requirement of any backend framework.

Project Implementation and Overview:

The project has 2 main aspects: The User Side and The Admin Side. So naturally there had to be 2 separate interfaces which were to be rendered on a condition which was if the user role is that of a **User** or an **Admin**. The user would be able to access the features of the website which features property details and only *read* it. Whereas the admin had the privilege to even modify the content shown on the "User side". To modify the content, the User had to be given the access to perform CRUD Operations through an interface which cannot be accessed by regular users. This would therefore require authentication which would direct the users to their respective places. So, the flow becomes:

Login/Register -> Admin? -> Admin Portal Access and User Website Access
| No?
V
User Website only (No access to the admin portal)

Features of the User Website:
The user should be able to login/register and access the features of the property website where they can interact with the information fetched from the database. The UI includes a table where details of the property are to be listed. This list can be viewed using various filters based on various categories including a search function. To make it more user friendly, UI will include an interactive map as well which can be used to locate the house geographically. User should be able to view details of house clicking on the markers present on the map. These features certainly enhance the User Experience and make it easier for the user to interact with the product.

How was it done?

The UI was created using ShadCn's reusable components. Additional CSS was added using Tailwind CSS. REST API was used to fetch the data from the database and displayed on a table, results of which can be filtered using the filters provided on the top of the table along with a search bar which can be used to filter out results too. There is an addition of interactive map as well which was integrated using **Leaflet** package. The map has all the locations of the houses which have the latitudinal and the longitudinal information available.

The Access to the User website has been given to the 'User' as well as the 'Admin'. Although accessible by all categories of users, property route is still enforced with protection from

malpractices on the routing, more details on which will be given further. You may find screenshots for the UI below:

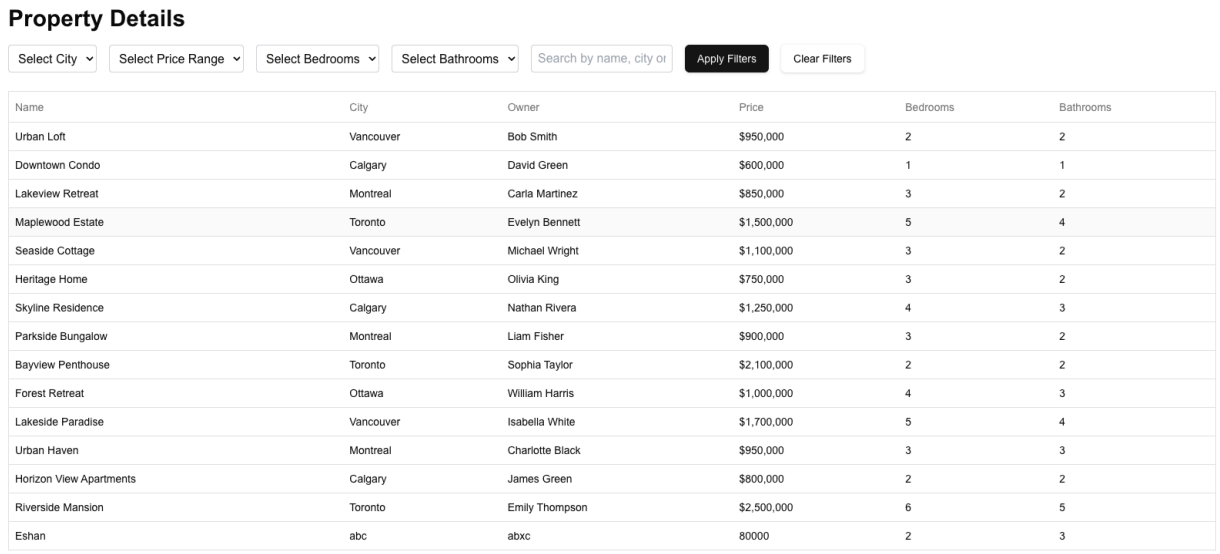

Image.1 Table listing down all the properties

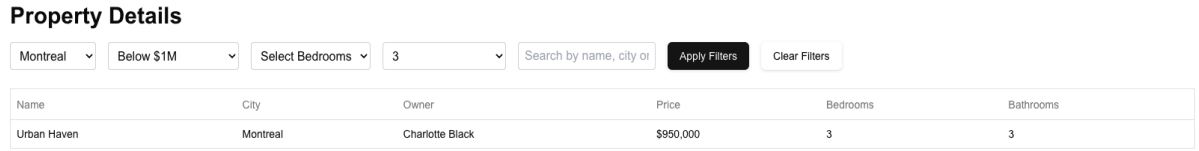

Image.2 Properties listed using filters

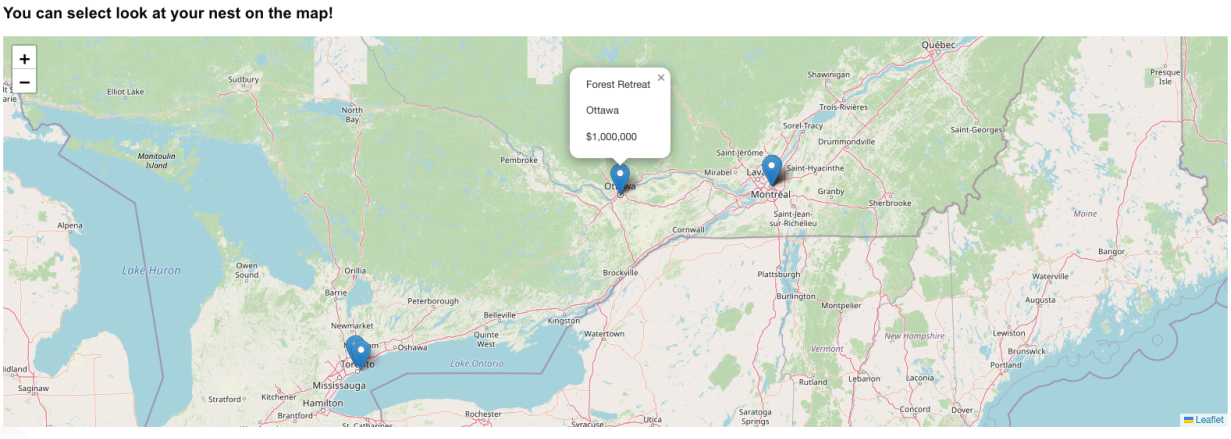

Image.3 Map with property markers which show property details

Features of the Admin Website:

For control on the content displayed on the User Website, an admin portal was required which would be able to perform CRUD Operations. This portal would give the admin the options to give admin control to specific users as well as edit the content on the User Website. Majority number of API calls would be made from this part of the application therefore every network operation for the website had to be protected and made secure.

How was it done?

With the same login process as of the user, only users with the admin privilege would be able to access the website. If someone without admin privilege tries to access the url, they had to be redirected back to user website because they are not authenticated. This was achieved using **NextAuth.js** which provides function to enable and access *sessions* which can be used to determine the user role and reroute accordingly. While access to the admin website was important, protecting API calls was important as well, therefore feature has been implemented where before every API call, authentication is made that the user is indeed an admin and not someone who bypassed the previous check. More details on the security aspect have been provided below. You may find the screenshots for the admin portal below:

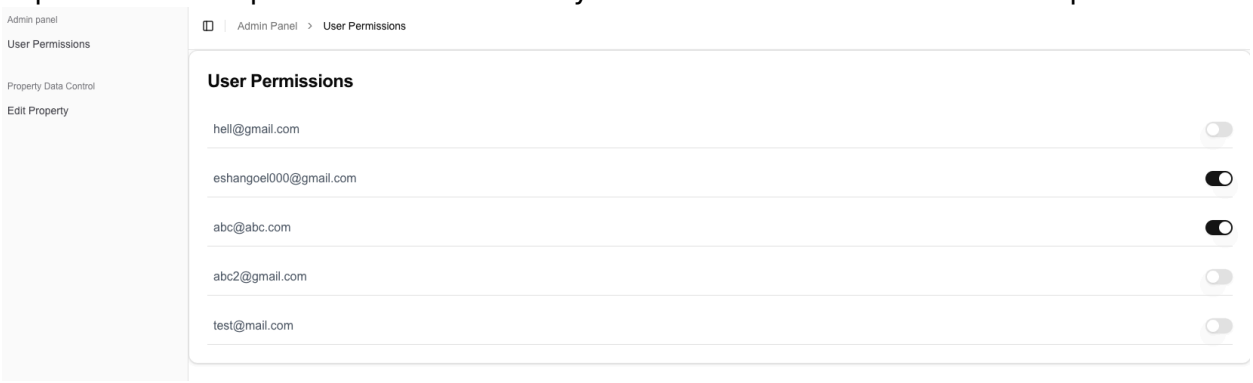

Image.4 Admin portal to change user permissions

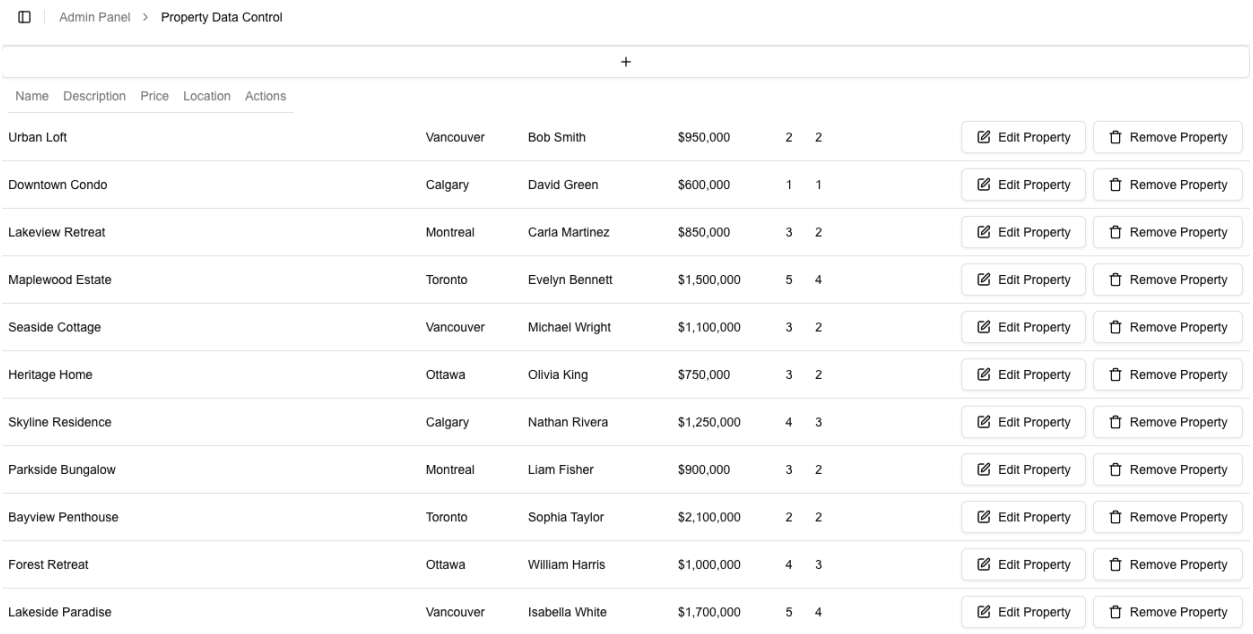

Image.5 Admin portal for performing CRUD Operations

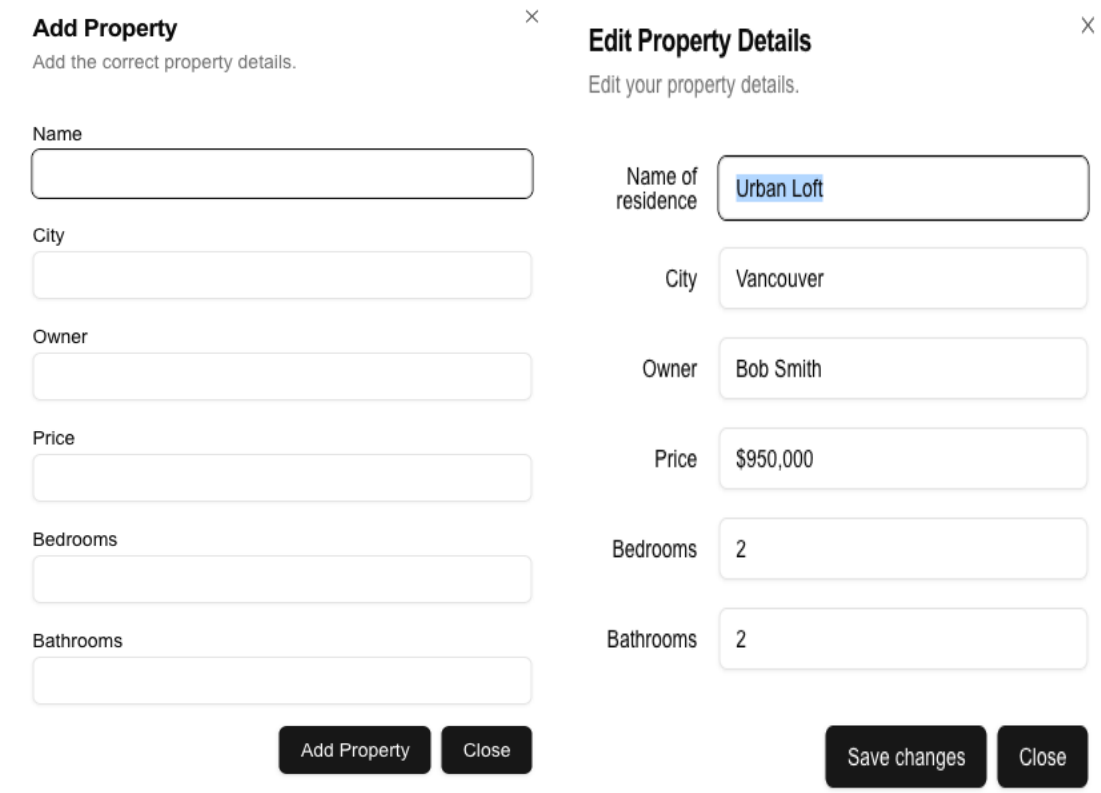

Image.6 Dialog boxes to edit and add properties

Security Implementation:

While basic web application could have been made easily, a heavy emphasis had to be made on the security aspect of the website because of the addition of Role Based Access Control (RBAC). Throughout the report, a heavy emphasis has been made on authentication using **NextAuth.js**. Before we dive into the reason we used this library, we had to list down all the potential ways through which potential invasions or attacks were possible.

- **HTML Injection in the login/register forms:**
  In the login and register forms, attackers could potentially inject malicious HTML or JavaScript code.
- **Password handling through API:**
  Risk of insecure password storage or exposure through APIs.
- **Admin Portal access by a non-admin**
  Non-admin users gaining access to the admin panel.
- **API calls by non-admi.**
  Non-admin users making unauthorized API calls to perform sensitive operations.

All these security issues were addressed in the application. **Regular Expression** was implemented on all the forms to prevent any kind of HTML injection which might affect or make the input fields vulnerable to any attacks. For the issues where user authentication

was of necessity, NextAuth.js was used as it makes the process very convenient for the developer to implement rules which will be followed for the process of authentication. Unless certain conditions are met, user is not allowed to access the admin portal. These conditions have been implemented not just on the login page when accessing the portal, they have been implemented on the API calls as well to prevent any kind of malpractice while performing CRUD operations. Since there are timestamps on several operations, it becomes easier for the admin to pinpoint issues, if created by other admins themselves.
Another feature which has been used in the application is **Prisma** which not only helps create a schema when manipulating data, it also enforces type validation. If the content body of the API does not follow the rules, the request simply does not go through.

All these security measures make the application a very robust and secure website which can recent several attacks. While there are always more things which can be considered regarding web security like encryption of API calls, use of JWT tokens to make it more secure, the additions made for this software truly suffices the needs of the hour.

Future Scopes:

This section aims to address some of the things which would enhance the web experience for the users as well as admins. While most of the additions relate to UI, backend additions aim to make the application more secure.

Frontend:

- User site would have additions of pagination in case there are a lot of entries.
- The filter works with only one option of a type at a time, selecting multiple options would be a better option.
- Every time an item from the table is selected, map can auto locate the house on map.
- Addition of property details like images, accurate addresses and option to bid or inquire would be much better.
- All the operations on the admin portal would become better if an option to “conform selection” were to be added.
- Enhanced RBAC like select admins can *read* and select admins can *read and write*.

Backend:

- Encrypting API calls to protect data in transit.
- Implementing JWT (JSON Web Tokens) for secure and stateless authentication.
- Adding an extra layer of authentication for admin users.