**EUROPEAN UNIVERSITY OF LEFKE**

FACULTY OF ENGINEERING

TRNC HOME SPOTTER

By

**VICTOR MAYOWA ADEDAYO**

**20140273**

A dissertation submitted in partial fulfilments for the

degree of

Bachelor of Science

In

Computer Engineering

20th December 2023

Supervisor: Asst. Prof. Dr. Cem Kalyoncu

### Abstract

**The main aim of this project is to build up a real estate web application for the TRNC (Turkish Republic of Northern Cyprus). The major outlook behind project this project is to develop a web application which makes apartments and residential buildings easier to access in the TRNC. Real estate agents in the TRNC can be challenging on individuals as there is no preciseness in services rendered. Therefore, this project addresses a web application which provides its users access to detailed information on properties, including prices, pictures, locations, and a comment section.**

**Housing being one of the most essential needs of life can play a crucial role when it comes to selecting the right choice,**

**Table Of Contents**

[Abstract 1](#_Toc1401205619)

[Table Of Contents 1](#_Toc898991817)

[1.Introduction 2](#_Toc1636914662)

[1.1 Problem definition 2](#_Toc886789351)

[1.2 Goals 2](#_Toc2008716941)

[1.3 Purpose of this project 3](#_Toc706923684)

[1.4 Benefits of this project 3](#_Toc1787400184)

[1.5 Literature Survey 3](#_Toc925414728)

[2. Development Methodology 4](#_Toc612362393)

[2.1 Background Information 4](#_Toc1644273553)

[2.1.1 software used 4](#_Toc582374546)

[2.1.2 6](#_Toc1936559807)

[3. Modules 6](#_Toc632085764)

[4.1 Property Listings 6](#_Toc1829940289)

[4.1.1 Advanced Search 6](#_Toc1492782185)

[4.3 User Dashboard 7](#_Toc1825365719)

[4.5 Security Procedures 8](#_Toc583911076)

[5. Risk Analysis 8](#_Toc1673359855)

[6. Ethics 9](#_Toc760987013)

[7. Conclusion 9](#_Toc1281929454)

[7.1 Benefits 10](#_Toc1363579509)

[Benefit to Users 10](#_Toc1176350445)

[Benefits to me 10](#_Toc1961721331)

[7.2 Future Works 11](#_Toc1361227848)

[8. References 11](#_Toc731706417)

# 

# 1.Introduction

## 1.1 Problem definition

There is usually a great distress when it comes to buying a house/acquiring a residence. This project seeks to ease the home searching process in the TRNC. In the scope of the TRNC, buying a house can be incredibly challenging. The TRNC has been described for decades as a prominent landscape for tourism and infrastructural expansion which also dates to a long history of foreign influence and hegemony, from ownership by the Ottoman Empire in the 16th century to the British Empire occupation in the 18th century which continued till the mid 90’s where the island gained its independence, the island at that time occupied by Greek and Turkish Cypriots got separated by the Turkish military into the Greek-speaking south and Turkish-speaking north as a result of aggravations in inter-communal hostilities. The Turkish Republic of Northern Cyprus became a republic in 1983, and it is recognised solely by Turkey, this history plays a labyrinthine role in the country’s property market. Over the years, the landscape of the TRNC has drawn in an influx of tourists, investors, students, and other purposed-filled entrants which is shown to have scaled the infrastructural development in the country, from schools to estates, hotels, local business stores etc. This also led to an increase in demand of real estate agents who help buyers find suitable properties, negotiate and seal deals, provide credible information to buyers and market properties for sellers to seek potential buyers. This had been demonstrating remarkable functionality prior to recent peculiar events involving real estate agents misguiding buyers in property advertisement and negotiations, misinformation of such gravity that they could also qualify as criminal offences as they often cause the buyer into making a wrong decision. The significant issues posed to buyers are stated below:

* Limited information accessibility: buyers could only know as much as they were being told by the agents thereby hindering access to make informed decisions.
* Unreliable search processes: information regarding buildings, rooms and prices are inefficient due to lack of precision in information retrieval.
* Communication barriers: often there is no direct communication link between the buyer and the seller, which is a challenge to most buyers.

This project aims to develop an extensive web application that allows users to search for properties with ease in the TRNC. The application will provide detailed insights to buildings, locations, rooms, pricing, contacts, and community feedback. This project will leverage React in designing its front end and Node backend from a user-friendly interface to backend features like property listings and engaging with property owners through a feedback configuration. Goals

This project aims to tackle the preceding intricate challenges. The goals of this project are highlighted below:

* The use of JavaScript combined with other utilities to create comprehensive property search web application which allows users to search for apartments and residential buildings with respect to preference such as location and available rooms.
* Provide users with thorough information on each property including rooms, map locations, residence image, and pricing details.
* Synchronising direct communication between the buyers/users and the sellers/property owners reducing the need for third party facilitators.
* Implement a community engagement system enabling buyers/users to share their experience, provide feedback on properties and ask questions prompting a collaborative community.
* Design a comprehensible user-friendly platform ensuring a cohesive and delightful experience for users of different technical proficiency.

## 1.3 Purpose of this project

This project plays a vital role in the reforming the real estate experience in the TRNC. The purpose of this project is to:

* Provide property seekers with an effective means to make informed decisions when searching for properties.
* Ease the property search process by minimizing the scale of residences to a single application.
* Bridge communication gaps between users and property owners by eliminating intermediaries and facilitating a better transaction.

## 1.4 Benefits of this project

* Enhanced user experience as search process becomes much more efficient and enjoyable.
* Public/societal liaison as people come together to share, insights and experiences about properties and information regarding locations.
* Promoting the growth of the TRNC real estate market as it attracts both buyers and sellers.

## 1.5 Literature Survey

This project defines a web app for the TRNC that allows users to find apartments and residential buildings, view the buildings, rooms, location, price, contact of the owner, and comments from other people. The implementation of web applications in finding properties has been progressively utilised over the years. In [1], a novel architecture is proposed for home rental applications, which is implemented on the concept of smart contracts. This (smart contract) operating as the autonomously executable protocols leveraged the transparency, the immutability, and the security inherent in blockchain technology. Advocating for the transformative impact of smart contracts, the project posited that the enhancement of the efficiency, reliability, and trust in home rentals. Through the navigation of intermediaries, cuts in transaction costs, and cinching stringent enforcements of its statutory obligations, the project strives to simplify the rental process. The tangible proof of this lies in the prototype of a home rental web application. Built up on the Ethereum blockchain using Solidity to exhibit the project’s viability. A major contrast distinguishing this project from mine is the strategic choice of a blockchain-based approach focused on delivering heightened security, reliability, and trustworthiness while mine underlines simplicity, economical, speed and compatibility. Another implementation could be seen in [2], this project aimed to effect a comprehensive user study aimed at accessing the usability and user satisfaction with functionalities including property search, booking and review features. Its technical framework constituents are Django used as its development stack, PostgreSQL, and Google maps API for geospatial integration. Its methodology consisted of a flexible approach combining quantitative analysis of data including evaluation principles measured on efficiency, learnability, and satisfaction, and qualitative analysis of data displaying data collection techniques such as questionnaires, interviews, and surveys. Following a thesis test, the nuanced difference between this project in comparison to mine was its focus as it centred on web application evaluation contrasting with my project's primary focus on development. This distinction serves as an insight for my project as its features are drawn from thoroughly examined user needs and preferences. In [3], A tutorial guide unfolding the creative pattern of developing a home rental web application in lens view to MySQL and JavaScript, two pivotal technologies in the web development landscape complexly leading, planning, coding, testing, and designing phases. Encompassing crucial elements such as database design, web page aesthetics and meticulous testing, the project serves as a meticulous guide, bolstered by downloadable source code and insightful screenshots, the project not only aims to impart essential web development skills but also to kindle the flames of creativity in the coder to begin their own web app development endeavours.

# 2. Development Methodology

# 2.1 Background Information

## 2.1.1 Software used

* **Javascript:**

this is widely used JavaScript library used which will be used in the project to build its dynamic user interfaces. Its virtual DOM makes it suitable for designing interactive and responsible web applications.

* **Node.js:**

It is widely used in backend development as it creates a runtime environment where users can run JavaScript code without a web browser, compatible with database operations it interacts with systems and processes requests from client-side application.

* **Visual Studio Code:**

It is an extensively used code editor efficient in building GUI applications and working with APIs development as it also provides a benevolent interface for implementing code as it is compatible with the project's language.

* **PostgreSQL/Vite:**

A sophisticated open-source relational database management system standardly used in applications like web and mobile, for managing, retrieving, and storing data as it draws its functionalities from the commonly known SQL (Structured Query Language).

* **React.js:**

this is widely used JavaScript library used which will be used in the project to build its dynamic user interfaces. Its virtual DOM makes it suitable for designing interactive and responsible web applications.

* **MongoDB:**

this is widely used JavaScript library used which will be used in the project to build its dynamic user interfaces. Its virtual DOM makes it suitable for designing interactive and responsible web applications.

* **MongoDB:**

this is widely used JavaScript library used which will be used in the project to build its dynamic user interfaces. Its virtual DOM makes it suitable for designing interactive and responsible web applications.

* **Google Firebase:**

this is widely used JavaScript library used which will be used in the project to build its dynamic user interfaces. Its virtual DOM makes it suitable for designing interactive and responsible web applications.

* **Express:**

this is widely used JavaScript library used which will be used in the project to build its dynamic user interfaces. Its virtual DOM makes it suitable for designing interactive and responsible web applications.

* **HTML :**

this is widely used JavaScript library used which will be used in the project to build its dynamic user interfaces. Its virtual DOM makes it suitable for designing interactive and responsible web applications.

* **Tailwind CSS:**

this is widely used JavaScript library used which will be used in the project to build its dynamic user interfaces. Its virtual DOM makes it suitable for designing interactive and responsible web applications.

* **Figma:**

Unlike other frameworks, this is a design tool that enables great visually attractive and interactive patterns for websites and mobile applications. It will be used to simplify the design process in this project.

* **Git:**

Distributive version control system used in tracking codebase modifications and a community with other users. Functions as a code repository in this project.

## 2.1.2 Dependencies Installation

After opening the trnc-home-spotter-system folder in my visual studio code terminal, I created a client folder with Vite using the command “npm create vite@latest client”. This command created the client folder in my home-spotter folder alongside its vite dependencies and a request to select the framework to be implemented for which the React framework was selected to be used in this application, following that a request came up to select the variant I would be using in the application for which I selected javaScript + SWC which is basically the normal javaScript configuration but with SWC(Super-fast Web Compiler) which simply optimises the compilation of javaScript code to make it faster and smaller. Prior to this installation, I had installed node.js, which is the runtime environment for this application and by the initialization code npm i in my client folder, the necessary dependencies were installed in the folder. Next, I followed the installation process for Tailwind CSS with Vite via their documentation on google, the steps followed can be seen in <https://tailwindcss.com/docs/guides/vite> . Following that I installed “ES7 + React/Redux/React-Native snippets” from my extensions in visual studio code which is a unique combination of moder javascript(ECMAScript 7), React which serves as the user interface library and Redux which is generally used in state management. The “prettier” extension was also included to ensure all codes were formatted to structure; tailwind CSS IntelliSense was also installed to suggest styling format. Following this, I cloned my bitbucket address using git clone <http://gmells@bitbucket.org/eul_comp/trnc-home-spotter-system.git> for initializing the git repository, added the files using “git add .”, and committed he files with a message saying “git commit –m “install tailwind CSS and react” “. Following that I used the git remote add origin <https://trnc-home-spotter-system@bitbucket.org/eul_comp/trnc-home-spotter.git> to create the remote repository followed by “git add .” to add the files and then commit with the message “git commit –m “initial commit” and finally “git push –u origin master” to set the branch to master. In the package.json file, the type was also set to module to call dependencies better (e.g import express from “express”). In the package.json, I also set in the scripts “dev”: “nodemon api/index.js” so that npm run dev also runs the backend. And “start”: “node api/index.js” was also included in the scripts for the render platform running the backend. The .env file was also ignored in the backend in order not to make the password stored there visible to other users.

**Client Folder Installations**

* Installed react-router-dom using “npm i react-router-dom".
* Installed jsonwebtoken using “mpm i jsonwebtoken”.
* Installed redux toolkit and react-redux using “npm i @reduxjs/toolkit react-redux

**Backend Installations**

* Initialised the backend (root – trnc home spotter systems) with npm init –y
* We use npm i express to install express which was the framework used in creating the backend Api's
* npm i nodemon was also installed to automatically update changes in the backend
* npm i mongoose installed mongoose in the backend to use mongoDB as its database
* npm i dotenv was also installed in the backend
* npm i bcryptJs was installed also, used to hash the password
* npm i redux-persist
* npm i firebase

## 2.2 Client Side

The client side of this application also known as the frontend defines the imperative files and folders that make up the visible part of this application which the user gets to interact with. The clients side consists of three main folders which are the node\_modules(containing most of the installation dependencies and files that ensure the right functioning of the application), the public folder(containing a favicon.ico file which is actually the favourite icon file that displays an image in file links to the application), and the src folder consisting of all the major files and folders that make up this application. A breakdown of the frontend would be expounded as follows:

### 2.2.1 Assets

### This houses the images folder which contains the home image which is the image that appears at the top of webpages when the trnc app is launched.

### 2.2.2 Pages

### 2.2.2.1 About.jsx

### 

### 2.2.2.2 CreateProperty.jsx

### 

### 2.2.2.3 Home.jsx

### 2.2.2.4 Profile.jsx

### 

### 2.2.2.5 Property.jsx

### 

### 2.2.2.6 Search.jsx

### 

### 2.2.2.7 SignIn.jsx

### In the sign in page, I created an h1 tag saying sign in alongside two input tags taking in the user email and password, alongside a Sign In button and an the google authentication component. I had also imported the reducers, signInStart, signInSuccess and signInFailure from the resux userSlice file. When a value is entered the handleChange function holds that information whilst accepting newer inputs and setting the change as the value of each id targeted. The handleSubmit function, triggered when the user clicks submit firstly prevents the page reload default, then the useDispatch() function is activated using the dispatch constant to call the signInStart reducer, the form info passed as a string to the json is stored in a constant named data.

### 

### 2.2.2.8 SignUp.jsx

### In the sign up page, I created an h1 tag saying sign up alongside three other input tags taking in the username, email and password parameter with an onChange attribute that adds whatever information is given to the parameter to the as the new value as e.target.id locates the id of what is being changed(could be the email for instance) and sets its value to whatever is written in the input. The spread operator ...formData keeps the input probably email could have been written first or username but it holds the previous information whilst adding a new one. A paragraph tag was also added provided there the user already has an account with a span element reading “Sign in” wrapped in a link tag that directs the user to the sign in page where they could just log in with their email and password. In the asynchronous handle submit function, when the user submits the form, the default behaviour of forms reloading the page is cancelled using e.preventDefault(), from there loading is set to true which shows loading on the screen whist the formInfo from the body of the screen is being passed as a json string to a data variable and sent to the database. If the data is not passed successfully, the setError function would pass the error message and the loading effect would stop else if it is successful, the loading effect would also stop, the error would be set to null, and the user would be navigate to the sign-in page using the useNavigate function

### 2.2.2.9 UpdateProperty.jsx

### 

### 2.2.2.10 ViewProperties.jsx

### 2.2.3.1 Action.jsx

### 

### 2.2.3.2 Comments.jsx

### 

### 2.2.3.3 Contact.jsx

### 2.2.3.4 Header.jsx

### In the header component, the return statement preceded a header tag as the outer container, in it was the nav tag which had the global styling for the header page, succeeding the nav bar a div containing the logo component imported from logo.jsx. Another div with a class of lg:hidden(on larger screens this div would be hidden) was then set containing a button with an onClick functionality which calls the toggle mobile menu which sets the not(!) of isMobileMenuOpen which was already given the state of false thereby setting it to true. Inside that button is a span div with a class of sr-only(screen readers only for visually impaired people), a svg card containing the design for the main menu button and finally a closing tag to the button and div. Next is a div containing the rest of the of the navigation buttons set to hidden in smaller screens ensuring it is only visible in larger screens, starting here is a form element containing an input element passed a value of searchTerm and a button reading search. For anything the user searches, the onChange attribute sets the new input from the user to the new searchTerm therefore whatever the user searches for becomes the new input value and when the search button is clicked, the onSubmit button is activated triggering the handleSubmit button which passes the query string of the URL to the object URLSearchParams and then held in a constant called searchParams. The value of the searchTerm is then passed to the searchParams using searchParams.set(“searchTerm”, searchTerm) and finally the navigate function, from the useNavigate hook is called to navigate to to the new search term if it exists in the database. After the from icon comes the anchor tag to the Home, about & viewProperties page alognside the respective link tags to their pages. Lastly in the larger screen size is a condition asking if currentUser is true, currentUser state is true if the user has signed up already, if the user is signed in, the last div in the nav bar would read “profile” else if the user has not signed up, it would read sign in, preventing the user from having access to custom features in the application.

### In small screen sizes which is mobile, the condition for seeing its nav bar is when the isMobileMenuOpen state reads true and it’s triggered by the onclick functionality but was hidden in larger screens as set in the div the top of the return statement. So, in the smaller screen this button appears and when clicked, the ToggleMobileMenu is set to true and the nav bar is shown in a at the side of the screen. Another button is set with a close icon and an onClick attribute is set with a closeMobileMenu function that closes the mobile menu and the rest of the nav bar is displayed in the same format. The two useEffect hooks in here perform two different functions. The first one has a handleResize constant sets menu to close in larger screens while the second one handles the search functionality which was explained earlier.

### 

### 2.2.3.5 Logo.jsx

### 2.2.3 Components

### 2.2.3.1 Action.jsx

### 

### 2.2.3.2 Comments.jsx

### 

### 2.2.3.3 Contact.jsx

### 2.2.3.4 Header.jsx

### In the header component, the return statement preceded a header tag as the outer container, in it was the nav tag which had the global styling for the header page, succeeding the nav bar a div containing the logo component imported from logo.jsx. Another div with a class of lg:hidden(on larger screens this div would be hidden) was then set containing a button with an onClick functionality which calls the toggle mobile menu which sets the not(!) of isMobileMenuOpen which was already given the state of false thereby setting it to true. Inside that button is a span div with a class of sr-only(screen readers only for visually impaired people), a svg card containing the design for the main menu button and finally a closing tag to the button and div. Next is a div containing the rest of the of the navigation buttons set to hidden in smaller screens ensuring it is only visible in larger screens, starting here is a form element containing an input element passed a value of searchTerm and a button reading search. For anything the user searches, the onChange attribute sets the new input from the user to the new searchTerm therefore whatever the user searches for becomes the new input value and when the search button is clicked, the onSubmit button is activated triggering the handleSubmit button which passes the query string of the URL to the object URLSearchParams and then held in a constant called searchParams. The value of the searchTerm is then passed to the searchParams using searchParams.set(“searchTerm”, searchTerm) and finally the navigate function, from the useNavigate hook is called to navigate to to the new search term if it exists in the database. After the from icon comes the anchor tag to the Home, about & viewProperties page alognside the respective link tags to their pages. Lastly in the larger screen size is a condition asking if currentUser is true, currentUser state is true if the user has signed up already, if the user is signed in, the last div in the nav bar would read “profile” else if the user has not signed up, it would read sign in, preventing the user from having access to custom features in the application.

### In small screen sizes which is mobile, the condition for seeing its nav bar is when the isMobileMenuOpen state reads true and it’s triggered by the onclick functionality but was hidden in larger screens as set in the div the top of the return statement. So, in the smaller screen this button appears and when clicked, the ToggleMobileMenu is set to true and the nav bar is shown in a at the side of the screen. Another button is set with a close icon and an onClick attribute is set with a closeMobileMenu function that closes the mobile menu and the rest of the nav bar is displayed in the same format. The two useEffect hooks in here perform two different functions. The first one has a handleResize constant sets menu to close in larger screens while the second one handles the search functionality which was explained earlier.

### 

### 2.2.3.5 Logo.jsx

### In the logo menu, letters in the alphabet were passed as a string to a constant called letters and the target text which was TRNCHS (trnc home spotter system) was set also;

### 2.2.3.6 OAuth.jsx

### In the authentication component, I returned a button saying log in with google with the type set to button so that it is not submitted along with the form elements it is used in both the sign in and sign up page when the form is submitted. It is only activated when the button is clicked and the onClick attribute is activated. In the handleGoogleClick function, I set up firebase for the google authentication then I pass the export app from firebase.js to the OAuth component, following that I set up the google authentication provider from the firebase/auth module and the getAuth from firebase/auth also, the app from the firebase configuration was passed into the getAuth, so firebase knows where to get the authentication from. We use the signInWithPopup function from firebase/auth to pass in the auth variable containing the getAuth for the imported app and the provider which holds the google authentication provider itself

### 2.2.3.7 Privacy.jsx

### 2.2.3.8 PropertyCard.jsx

### 

### 2.2.4 Redux

### 2.2.5 App.jsx

### In the App.jsx file, I imported BrowserRouter, Routes & Route from “react-router-dom" which allowed me to set up a return <BrowserRouter> statement containing the <Routes> and individual <route> component. In doing this, I was able to setup routes to the Home, About, SignnIn, SignUp, Search, Property, Privacy, Profile, createProperty, viewProperties & Update property page. The header component was added just after the BrowserRouter ensuring its visibility on all pages.

### 2.2.6 firebase.js

### In the firebase file, I followed the setup from firebase.google.com and copied the setup into this file, for extra security, I also hid the api key in the .env file by setting up a variable called “VITE\_FIREBASE\_API\_KEY” which was then imported into the firbase.js file and passed in as the value of the api key.

### 2.2.6 Main.jsx

## 2.3 Server Side/Api

The server side of this application prominently referred to as the backend defines the imperative files and folders that make up the non-visible part of this application. It consists of the functionalities and tools that ensures that the front end runs fine and work properly. In this sector, a breakdown of the backend would be expounded as follows:

### 2.3.1 Controllers

### 2.3.1.1 auth.controller.js

### In this authentication controller, the User model is imported from model, then the functions are defined.

### In the signup function, we get the username password and email form the request body and then we pass it to the database using the model and save it, since this might take some time we use the async/await function to making the signup asynchronous as it has wait for a function to be executed before moving to the next. Since the password could being seen in the database was a threat to user’s privacy, bcryptJs was imported to hash the password.

### In the signin function, the email and password are passed in from the screen body then a function called findOne is used to search if the email entered matches with any in the database User model, if none exists, the errorHandler function returns a 404 status code with the message User not found. Also, the bcrypt password for that email is compared with the password entered by the user. If the password is not correct, an errorHandler function is called passing in a 401 statusCode error and a message saying “Invalid login details!”. If the user email and password are valid, a const called token is created using the jwt(jsonwebtoken).sign to create a secured token for the user to get signed in. The validUser’s(user found in the database) is mixed with a secret key called JWT\_SECRET in my .env file for security. Finally, the password is separated from the rest of the data the user gets to see and then, the response is sent passing the cookie for the browser, containing a string called access\_token with the value of the token constant and its httpOnly attribute set to true allowing only the server to have access to the cookie, tightening security by disenabling third party access to the cookie.

### 

### This was the first route test for the sign-up page, after inputting the username, email and password, the response generated a “201” saying the user was created successfully.

### 

### In the database in mongoDB, the password could be clearly seen provided anyone had access to the storage which was a potential risk for the users

### 

### A route test was also performed in insomnia to ensure the sign in functionality operates normally.

### 2.3.1.2 user.controller.js

### In this user controller,

### 2.3.1.3 property.controller.js

### 2.3.1.4 comment.controller.js

### 2.3.2 Models

### 2.3.2.1 user.model.js

### Firstly, the database was activated by importing mongoose form “mongoose” dependency, then a userSchema/table was made using “new mongoose.Schema” and it took in username and password which are strings, required also and unique so no two users would ever have the same username or password. A password input was also set to be a string and require but didn’t have to be unique allowing to users to have the same password. Finally, an avatar table and stored as a string and given a default profile image string which would be passed to each user provided, they haven’t uploaded an image. The timestamps property was also set to true thereby storing the time of user creation and the time of user update. Then the model was created using the userSchema using mongoose.model(). And passed to a User variable which is then exported as the default.

### 2.3.2.2 property.model.js

### 2.3.2.2 comment.model.js

### 2.3.3 Routes

### 2.3.3.1 auth.route.js

### In this authentication route, the express package was first imported, then the middleware was defined by passing a const router to express.Router(), three post requests were made for the signin, signup and google functionality plus one get request for the signout functionality. All these functionalities had been imported from the auth.controller.js which houses all the functions for the route requests and finally I exported the default function route.

### 2.3.3.2 user.route.js

### In this user route, the express package was first imported then I activated the express middleware using express.Router() and passed it to my router constant. Following that I passed a get request to the test route calling the test function from the user controller file. All the other controller function were exported in this manner although a post request was sent in the case of update user, and a delete request in the case of delete user. Finally, I exported the default router.

### 2.3.3.3 property.route.js

### 2.3.3.4 comment.route.js

### 2.3.4 Utility

### 2.3.4.1 error.js

### In the error.js file, and errorHandler is defined takes whatever error statusCode and message given to it, and crates a new error from the function new Error() and returns the error.

### 2.3.5 index.js

### After importing express and mongoose, I created a mongoDB account for my database and connected it to my application using mongoose.connect(process.env.MONGODB) the variable was passed in from the .env file containing the actual string passed in. This was done as a security measure to prevent unwanted access to the database. The dotenv dependency was also imported as an environmental variable could not be used by default. All the routes were also imported into index.js and the app variable which was already set to express() was set as the middleware. app.use() containe d each of the routers from their various routes and their names were changed from router in their individual route's importation to a more specific name to know where each route is coming from but the route path itself did not change ensuring that there is an issue in the code. Finally, a middleware was created for the error code to pass the error status code and an error message if any from any of the routes.

### 2.4 redux

### 2.4.1 user

### In the userSlice.jsx, the createSlice function is imported, the initial state was also set starting currentUser, error and loading were set to null, null & false respectively. The createSlice is then activated and takes in reducers to to routes signInStart, signInSuccess, signInFailure, updateUserStart, updateUserFailure, updateUserSuccess, deleteUserStart, deleteUserFailure, signOutUserStart, signOutUserFailure & signOutUserSuccess. A rootReducer constant was also setup , then another persistedReducer was setup to take in the peristConfig and combine that with rootReducer. In the store the reducer is given the persistedReducer and then exported as a constant called persistor. In the

### Firstly, I imported the configureStore from “@reduxjs/toolkit as directed redux toolkit official documentation, then I configured the stored to get the default middle ware and set that to its value, after that was setting the serializableCheck in the middleware to false to prevent getting an error as the middleware isn’t serialized. MMore and the functions of this reducers where oeiedotr than haflfgokgjed eadcg =

### 2.5 Main.jsx

### This file runs the App.jsx file which basically contains every route in the application. A provider component wrapped around; the Provider component was imported from react-redux alongside the store for storage and the prersistGate functionality which is passes in the persistor from its files .

### 2.9 Vite.config.js

### In the Vite file, a define configuration function is written connecting a server to the client folder with a proxy setup which target the backend route <http://localhost:300> whenever /api is called.

# 3. Modules

## Property Listings

This depicts the main aspect in the applications. Users can search for residential building and apartments effortlessly, some of its key features include:

* + 1. Advanced Search

Users can become more informed when searching for apartments and residential buildings filtering through barriers such as location, price range and other utilities.

**4.1.2 Comprehensive** **Property Description**

Each property provides an elaborate description of its constituents, including rooms, pricing details, location on a map and exquisite images of the property.

**4.2 User Profile**

This submodule creates a personalised space improving users' performance by allowing a unique space or users to manage their preferences, saved listings, and accessibilities.

**4.2.1 Profile Creation**

Distinctions between buyers and sellers in profile accessibilities and controls, while both users can save favourite listings, sellers can actually modify their properties.

## 4.3 User Dashboard

This is a personalized space where users can customise their preference and get informed with the community.

**4.3.1 Unique Dashboard**

Users' customisation to individual satisfaction.

**4.3.2 Community Interaction**

Comment section accessibility creating an informed society as individuals ask questions, provide responses, and share encounters.

**4.4 Testing and Quality Assurance module**

This ensures the validity and verification of the application by testing its functionalities and reliability. Its features include;

**4.4.1 Unification Testing**

Ensuring all modules are functionally in line with each other.

**4.4.2 Unit Testing**

Components are examined individually to ensure their viability

**4.4.3 Verification testing**

External systems are used to ensure the applications functionality is correct.

## 4.5 Security Procedures

These are measures put in place to enforce user data is protected and the applications is secured.

**4.5.1 Secure Authentication**

Security measured enforced are user login information including a username and password and sellers require an image authentication with a valid identification document.

**4.5.2 Accessibility Controls**

Accessibility restrictions to some of the functionalities dependent on user roles e.g buyer, seller, admin etc.

**4.5.3 Consistent Security Audits**

Conducting regular security checks to identify potential security threats and implement measures to counter them.

# 

# Risk Analysis

In this project, through absolute assessment, monitoring, and mitigating, the potential risks posed to this application are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk | Description | Impact | Likelihood | Mitigation Strategy |
| Technical perplexity | The web application involves compound search algorithms alongside constant data updates affecting the systems performance. | High | Moderate | Consistent code review to identify potential threats and loopholes.  Eliminating complex logic without affecting its functionality. |
| Legal and regulatory compliance | Ignorance regarding the TRNC rules having potential legal threat. | High | Medium | Keeping up with government regulations to ensure compliance and consultation of legal experts to ensure profound knowledge of regulations. |
| User participation | Lesser user engagement as there is difficulty in navigating the application e.g. some functionalities do not work, or the designs are obnoxious. | Medium | Low | Ensuring the user has a unique participation attribute such as a customised dashboard. UX design should be thoroughly tested, and UI design should be user satisfactory |
| Integration  complication | Defects in third party tools e.g. APIs, integrating with the application | Low | Low | Thorough tests should be conducted at each successive progression off the application. Restrain volume of third-party integration enhancing focus to essential functionalities. |

# Ethics

In this project, some ethical accounts and measures considered and put in place are:

* Ensured the integrity of user data by implementing strong measures to secure it preventing unwanted access and trustworthiness [4].
* Promoting and online community, which enforces ethical guidelines to discourage discrimination and any form of crude conduct [5].
* Enforcing techniques to improve the application making it optimal at best in its abilities and ensuring user’s satisfaction [6].
* Users have an insight to the cause for information provided fully understanding their data functionalities ensuring clearness and credibility [7].

These ethical measures aim to create a web application environment that support user interaction and ideas, regard for other users and overall, an interactive digital community.

# Conclusion

In conclusion, the TRNC home spotter project aims to create a web application that helps anyone interested in purchasing a property in the TRNC. Leveraged with JavaScript, this project aims to simplify the search process in the TRNC thereby implementing time management, increased accuracy, creating a social interactive community and an increased rate of user's satisfaction.

## Benefits

This project is beneficiary in bare ways as it not only gives me an advantage, but it also aids the users of the application. This project benefits the uses in the following ways:

Ease in residence search in the TRNC as users can now acquire properties from any location around the world without apprehension of time, preference, or unaffiliated financial impositions from third party sellers.

### **Benefit to Users**

Mental composure as individuals can now have access to credible information regarding residents to be purchased and ensure their criteria is met before purchase is made as they are now more informed.

Establishment of an interactive community as individuals can now connect with other people in their regions and discuss relative matters of interest.

Buyers can now connect directly with sellers and vice versa eliminating communication barriers

### **Benefits to me**

Skill development as I get to advance my web development competence with JavaScript, implementing designs with both interactive and dynamic interfaces, and working with database management systems, improving my overall programming dent.

Improved project management capabilities as I get to research different publications, relative projects and draw out a revised thesis and conclusion.

This project builds my discipline and focus as I will be working on a functional web application, building diverse parts of the project. General improvement can be seen with day-to-day activities improving ethical conduct.

I chose this project cause of three reasons. Firstly, I live in the TRNC myself and the distress in acquiring a residence here is a pending issue which affects a lot of people, particularly international students coming into the country. Secondly, misinformation being spread around by third party agent can be misleading and leads to buyers making uniformed decisions and lastly, the lack of a social community in the real estate sector here is a real setback as it helps to foster credibility, awareness, and unification.

## Future Works

I wish to continue working on this project upon convocation, feedback gotten from users and other innovations would facilitate forward advancements ensuring it is built to user’s satisfaction. Some future considerations are extensions to mobile application, increased community, and dashboard features, and an improved communication channel.

# References

You should write your references which you found at your research and literature survey. You can use APA 6 Format

[1] Chen, J., & Liu, Y. (2016). A smart contract based on blockchain for the rental of apartments. In 206 IEEE international conference on Cloud Computing and Intelligence Systems (CCIS) (pp. 293-296). IEEE <https://doi.org/10.1109/CCIS.201.7819780>

[2] Lee, J., & Lee, J. (2015). Developing and validating a citizen-centric typology for smart city services. Government information Quarterly, 32(4), 453-463. <https://doi.org/10.1016/j.giq.2015.08>

[3] R. K. Patel, “How to design a home renal webapp using PHP and MYSQL,” IEEE Softw., vol. 40, no. 2, pp 10-15, Mar./Apr. 2023

[4] European Data Protection Supervisor. Retrieved from <https://edps.europa.eu/data-protection/our-role-supervisor/publications/privacy-design/principles_en>

[5] Facebook. (n.d.). Community Standards. Retrieved from <https://www.facebook.com/communitystandards/>

[6] Nielsen, J. (1995). 10 usability heuristics for user interface design. Nielsen Norman Group. Retrieved from <https://www.nngroup.com/articles/ten-usability-heuristics/>

[7] Federal Trade Commission. (2016). Cross-device tracking: An FTC staff report. <https://doi.org/10.2139/ssrn.2890502>