**ITE5315 Assignment 2**

**Kiranjit Kaur(n01702773)**

**Objective**: To explore more on Express Form and Template Engines

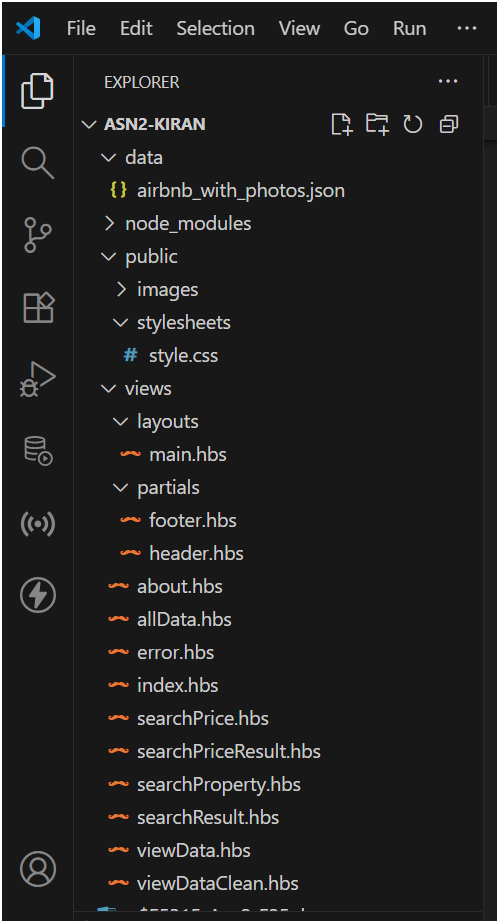
**Complete the following steps and take screenshot of the outcome of each step:**

**Step1: Create new Node/Express app as follow:**

* **Create new folder named “Asn2-yourname”.**
* **Change directory to “Asn2-yourname”:**
* **Open terminal and run npm init**
* **In the terminal run: npm install express --save**
* **In the terminal run: npm install express-handlebars --save**
* **Make sure to have the following project structure:**

|  |  |
| --- | --- |
| **Create a “public” which contains all images and stylesheets**    **Create “views” folder which includes all Handlebars templates. Make sure to add two folders “ partials” and “layouts” inside “views”. Also add “index.hbs” and error.hbs” files.**    **In the “views/layouts”, create a new file:**  **“main.hbs”**    **Create “app.js” in the project root.**    **Make sure to have proper project settings and dependencies in “package.json”** |  |

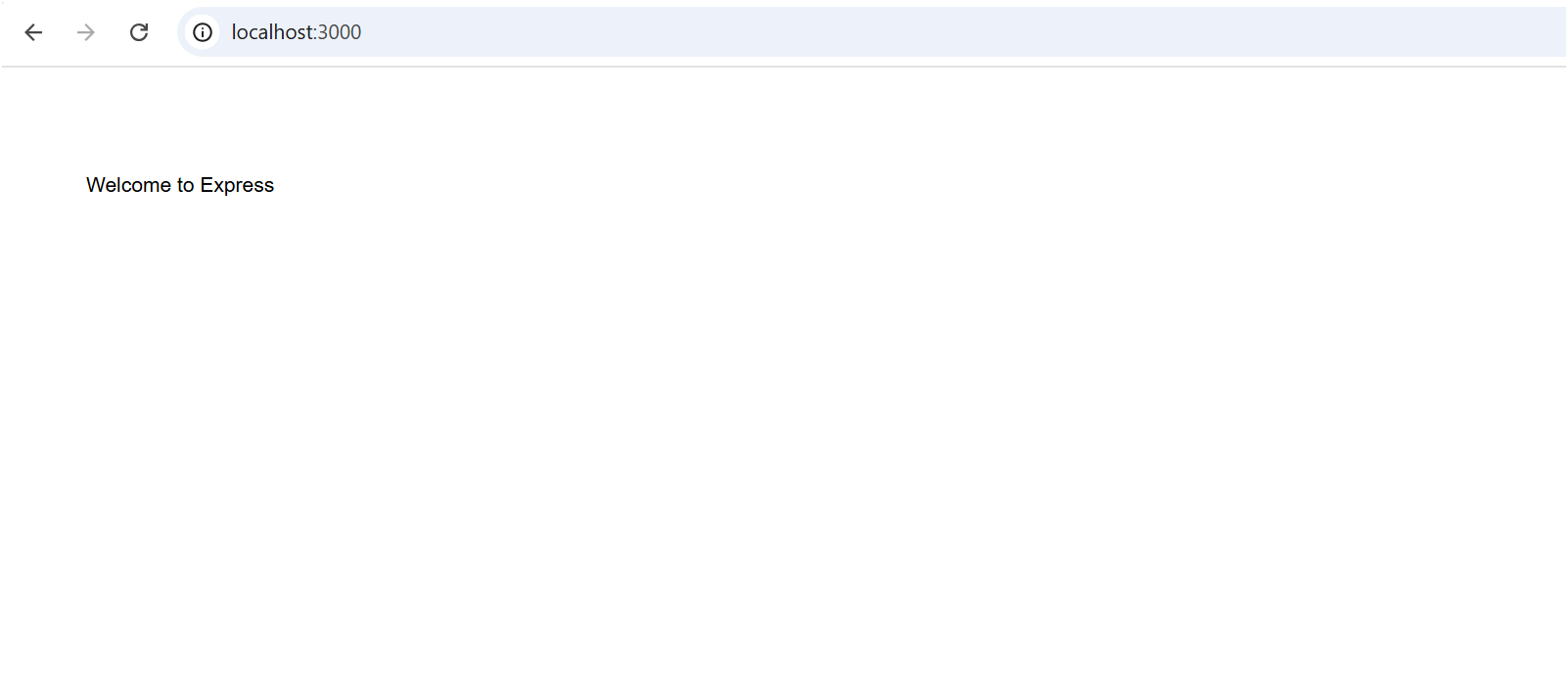
***Note****: It's important to distinguish that we've separated our* ***views*** *folder into three classifications for* ***layouts****,* ***partials****, and* ***pages****, where* ***pages*** *occupy the* ***root /views*** *directory. It's important to keep this distinction as our structure affects how we serve up these templates.*

****

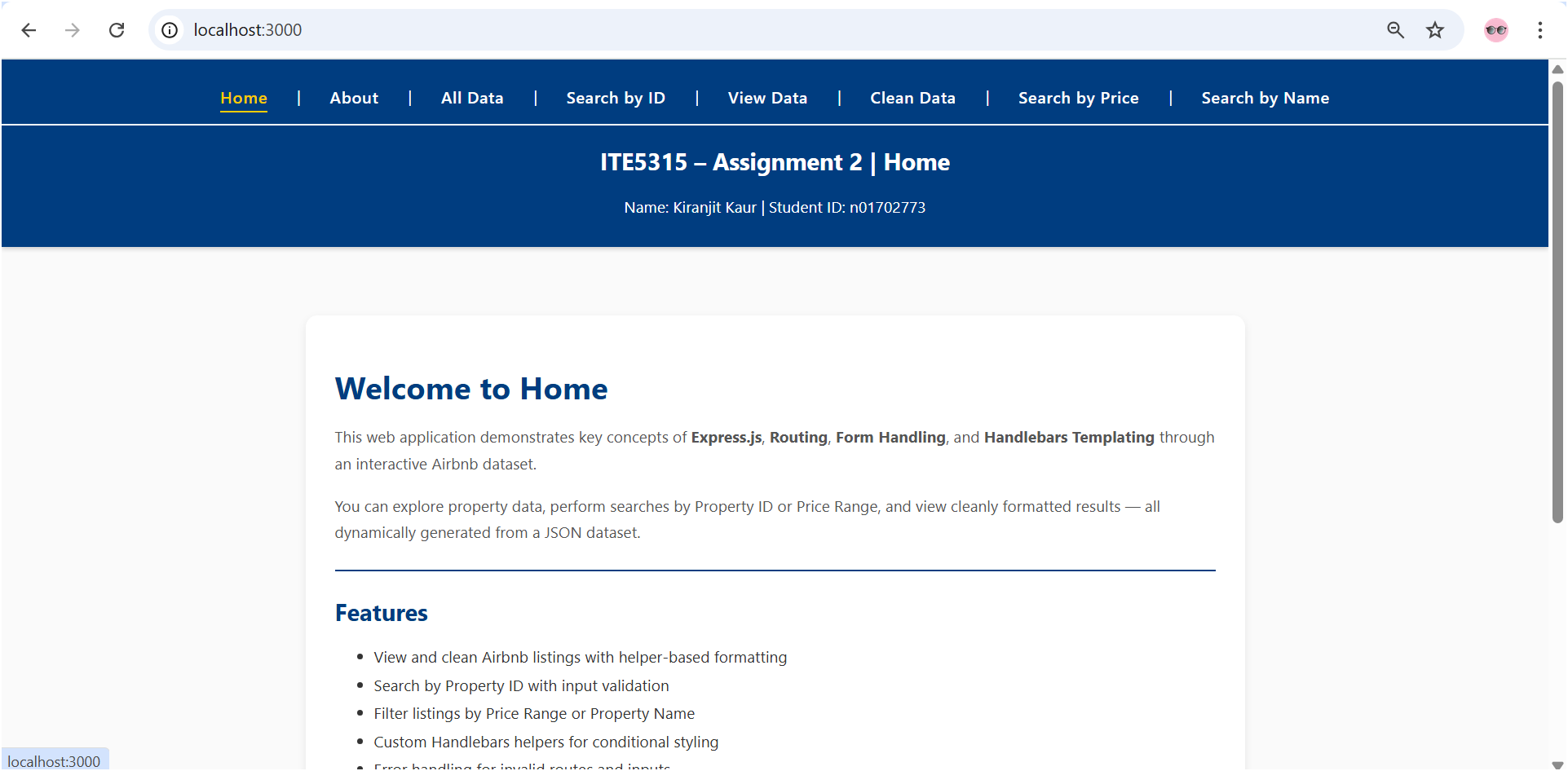
**Step2: Using the attach file (codenippet.txt), copy/paste the related code to “style.css”, “main.hbs”, “index.hbs”, “error.hbs” and “app.js”.**

**Step 3: Run the application and test it using the following routes:**

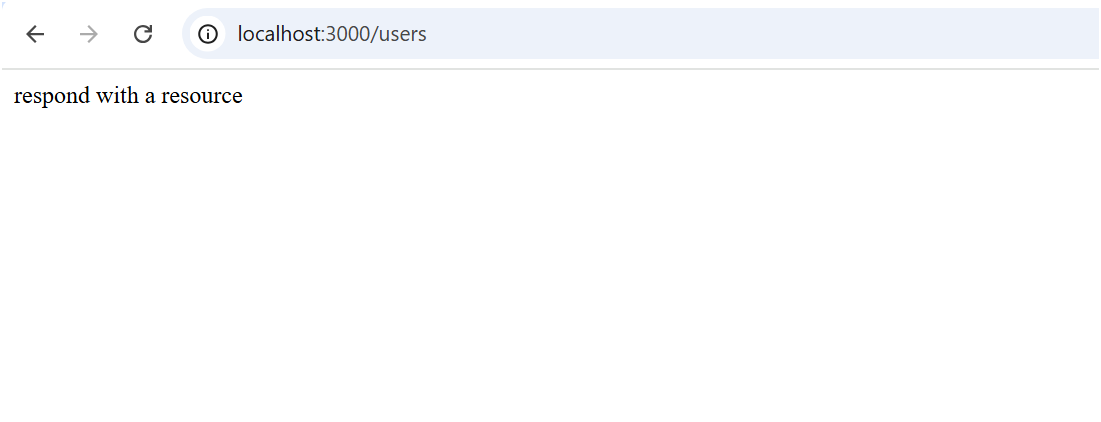
* [**http://localhost:3000/**](http://localhost:3000/)



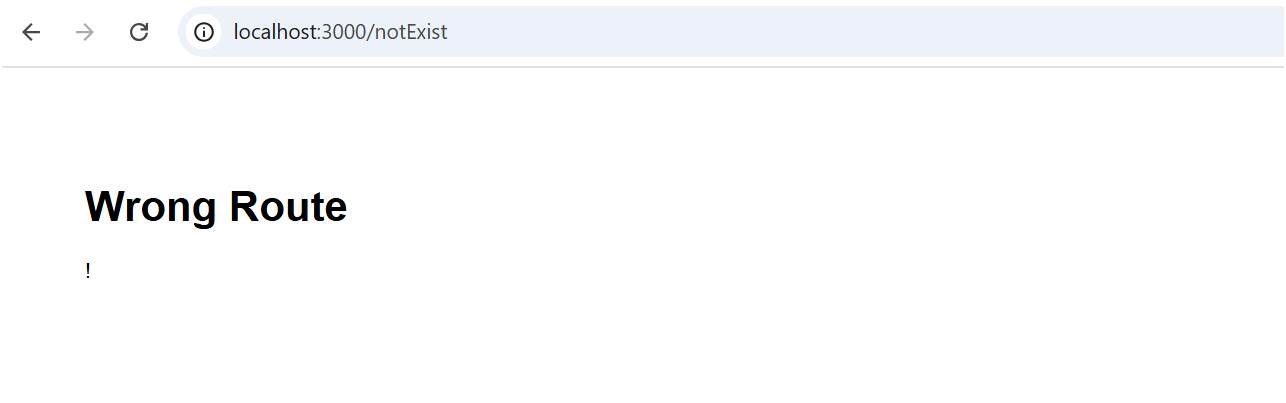
After applying all the styles, it looks like:

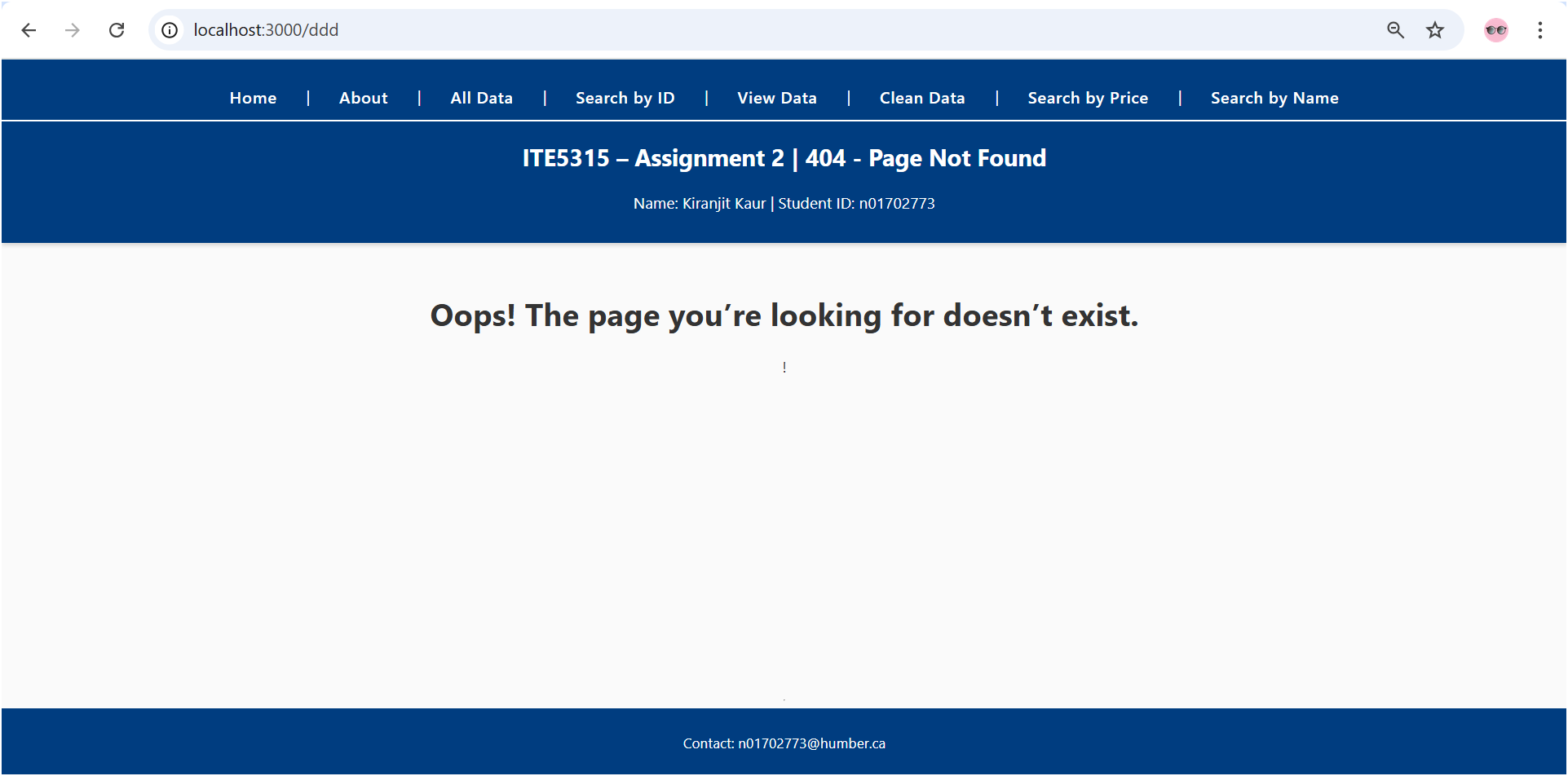


* [**http://localhost:3000/users**](http://localhost:3000/users)



* [**http://localhost:3000/notExist**](http://localhost:3000/notExist)





**Step 4: Base on your observation, answer the following questions:**

1. **What is the role of “main.hbs”? How does it being used/called in the app?**

main.hbs is like the frame of a webpage. It defines the common structure that all pages share, such as:

The <head> section with the title and CSS files

The <body> tag

Other pages like index.hbs or error.hbs don’t need to repeat this structure, they just provide the unique content for that page, which gets inserted into the {{{body}}} placeholder in main.hbs.

It is like a picture frame: the frame (main.hbs) stays the same and the pictures (index.hbs, error.hbs) go inside it.

1. **What is the role of “index.hbs” and “error.hbs”? How do these files being used/called in the app?**

**index.hbs**: This is the content users see when they visit the homepage (/). For example, a welcome message.

**error.hbs**: This shows a message when users go to a route that doesn’t exist, like /notExist.

Both of these files are loaded using res.render() in app.js, which tells Express to combine

them with main.hbs and display the full page.

1. **In “app.js”, what is the role of the following line? What happen if you commend it out?**

app.use(express.static(path.join(\_\_dirname, 'public')));

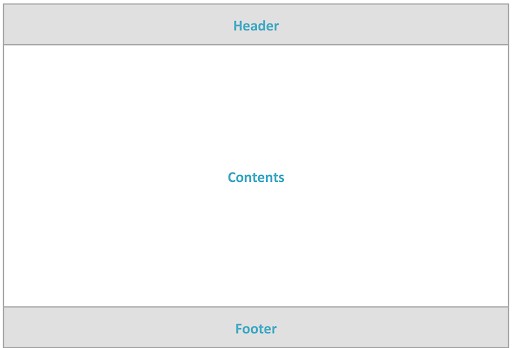
This line tells express where to find static files like CSS, JavaScript, or images.For instance, style.css is in /public/stylesheets/. If this line is missing, the browser can’t load CSS, so pages would look plain and unstyled.

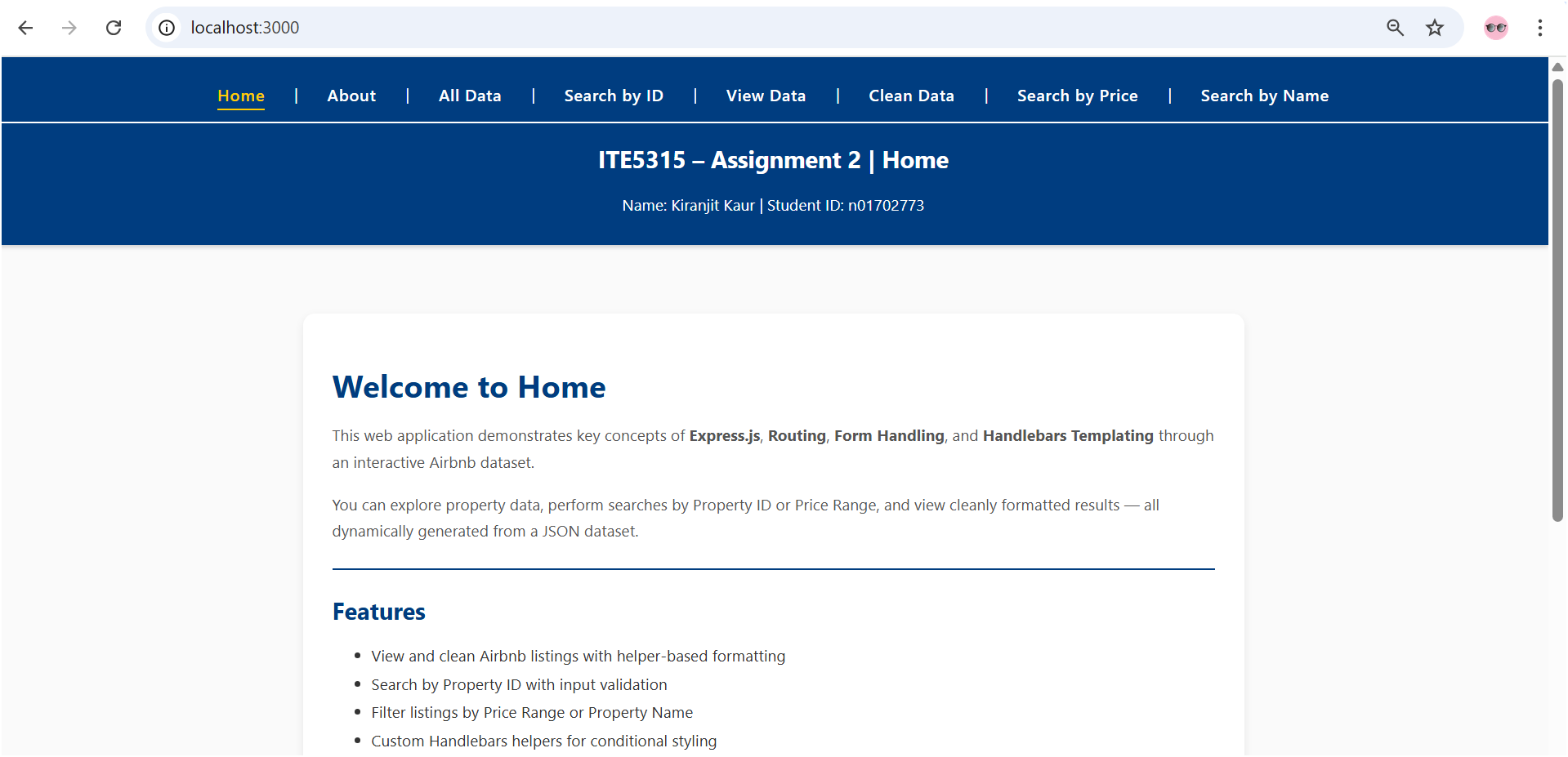
1. **Try to add proper comments to each line/block in app.js**

**Step 5**: **Modify the layout of the page and add “header”, “content” and “footer” as follow.**

* **Add your name/std\_id to the header**
* **Add your email address to the footer**
* **The other dynamic contents/messages should be displayed in the “content” area.**
* **Use your creativity to add proper css style/format. Add**

***(hint: you need to redesign main.hbs and style.css*)**





**Step 6: Any idea how to utilize “Partial” Templates in this app? Explain your answer**

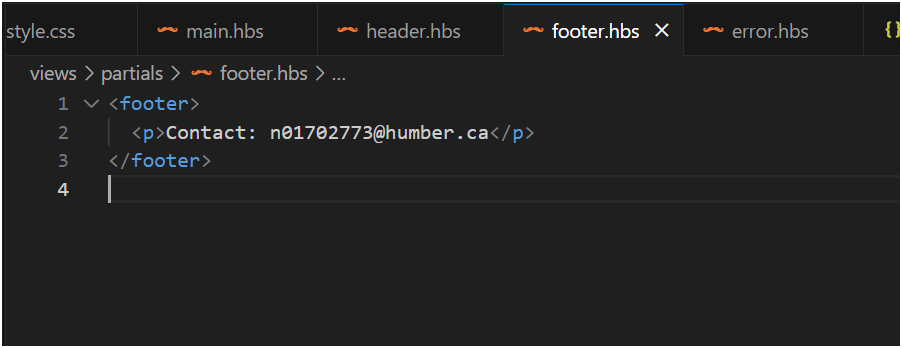
* **Bonus: implement your idea!**

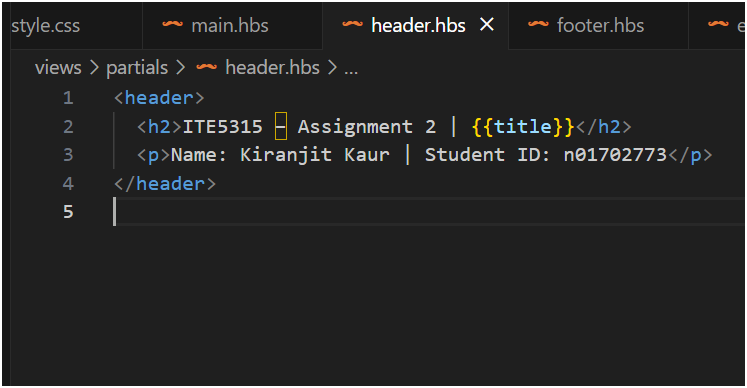
In handlebars, partials are reusable small template files — for example, a header, footer, or navbar that appears on many pages.  
Instead of repeating the same HTML in every .hbs file, we can define it once as a partial and include it anywhere with:

{{> partialName}}

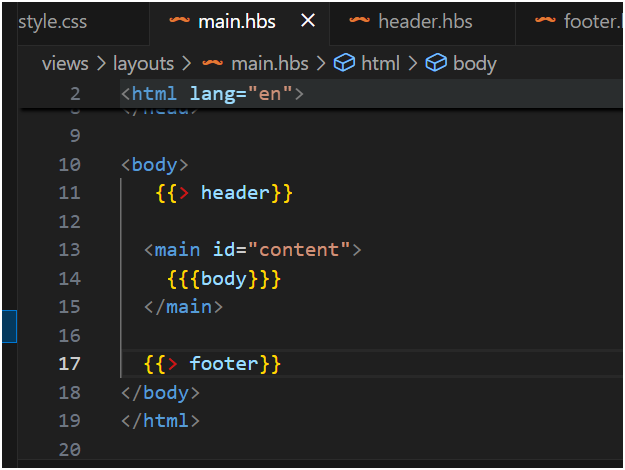
For instance:

Putting headers and footers into the partials folder





And changing the main.hbs:



Telling handlebars where partials live in app.js:

****

**Step 7: Add the Assignment1 code into this template. So the assignment1 form and output should be displayed in the content area of the layout.**

* **For each A1 route, render a dedicated page under views/ using res.render o For each of the route in assignment 1 (like /allData, /alldata/invoiceID/{index},**

**/search/ produceLine/, /search/invoiceID/), you need to create a related “page” in “/views” folder. So, when the “app.get” of the related route is called, you gonna use the proper “res.render(‘page’, {data})”. (*hint: use similar ideas like index.hbs*)**

**Moreover, you also should redesign the way that you generate output in**

**Assingment2 by displaying the output using Handlebars *expression/helpers*.**

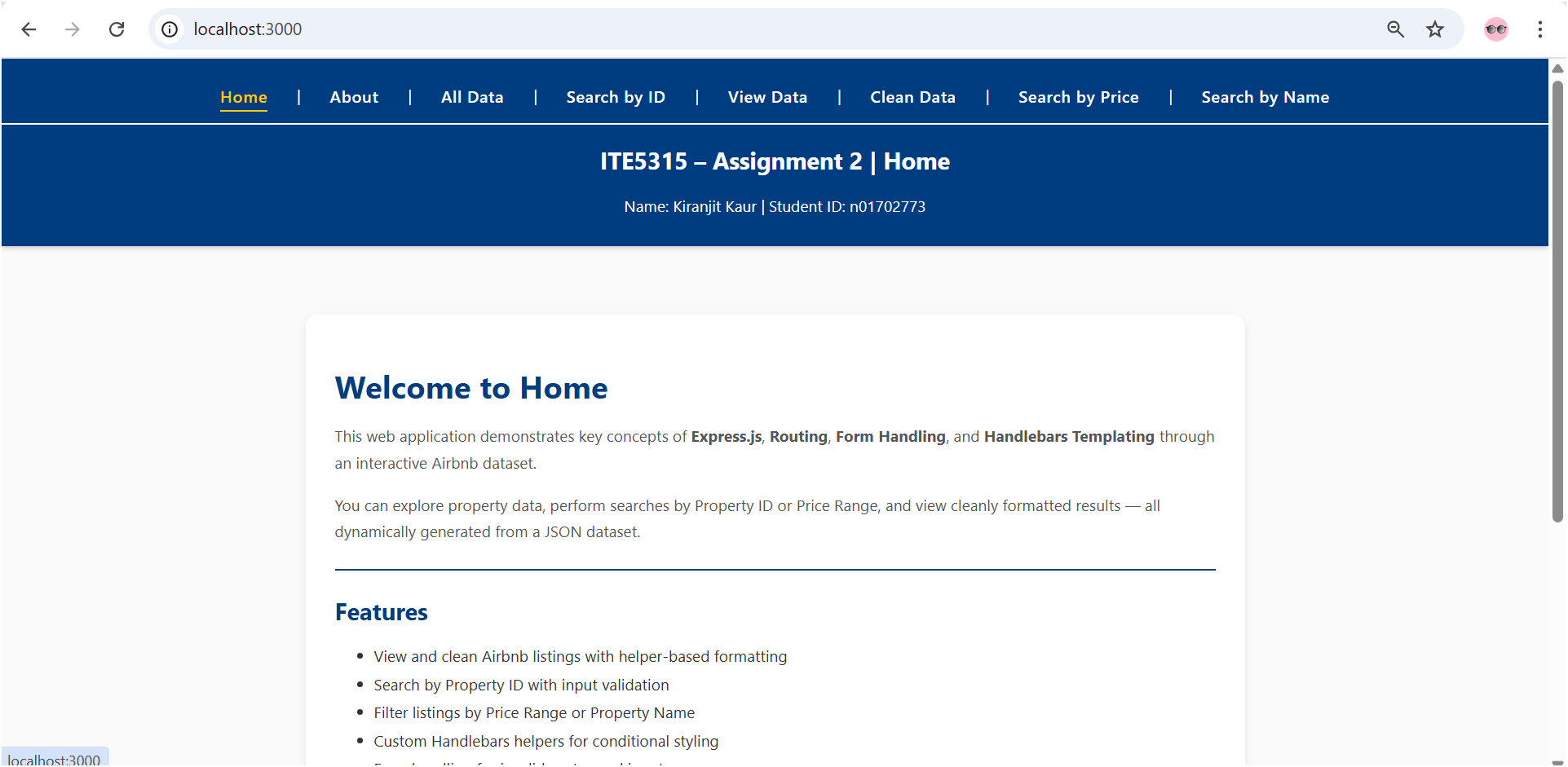
* **For the forms in (Assignment1: step7 and 8), add express-form-validator and define/apply the following validation rules o ensure the form-filed is not empty, & Sanitize inputs o ensure user provide numeric data in property\_id form\_field, & Sanitize inputs**

**➔ Also make sure your search results include the property image**

In this step, I integrated the code from Assignment 1 into the new Express + Handlebars layout.  
Each route from Assignment 1 (such as /allData and /search/PropertyID) was modified to use res.render() to display data within the main.hbs layout.  
The output was redesigned using Handlebars expressions and helpers, and the form validation was implemented using express-validator.

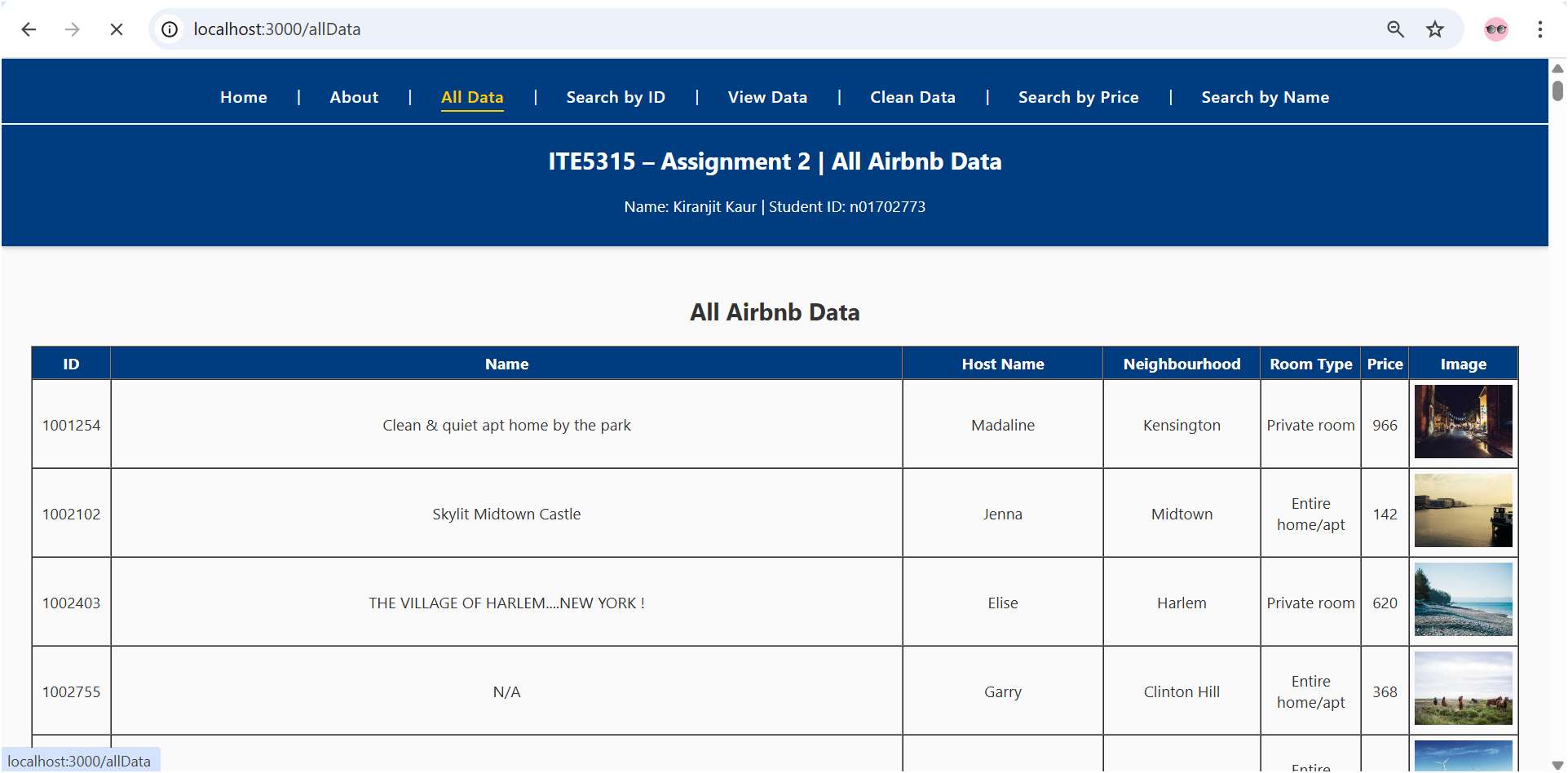
**Home Route (/)**

The home route (/) is the main entry point of the web application. It displays the homepage content within the common main.hbs layout using the res.render("index", { title: "Home" }) function. I have added more css styling and make the home page page looking good by adding the content as well.



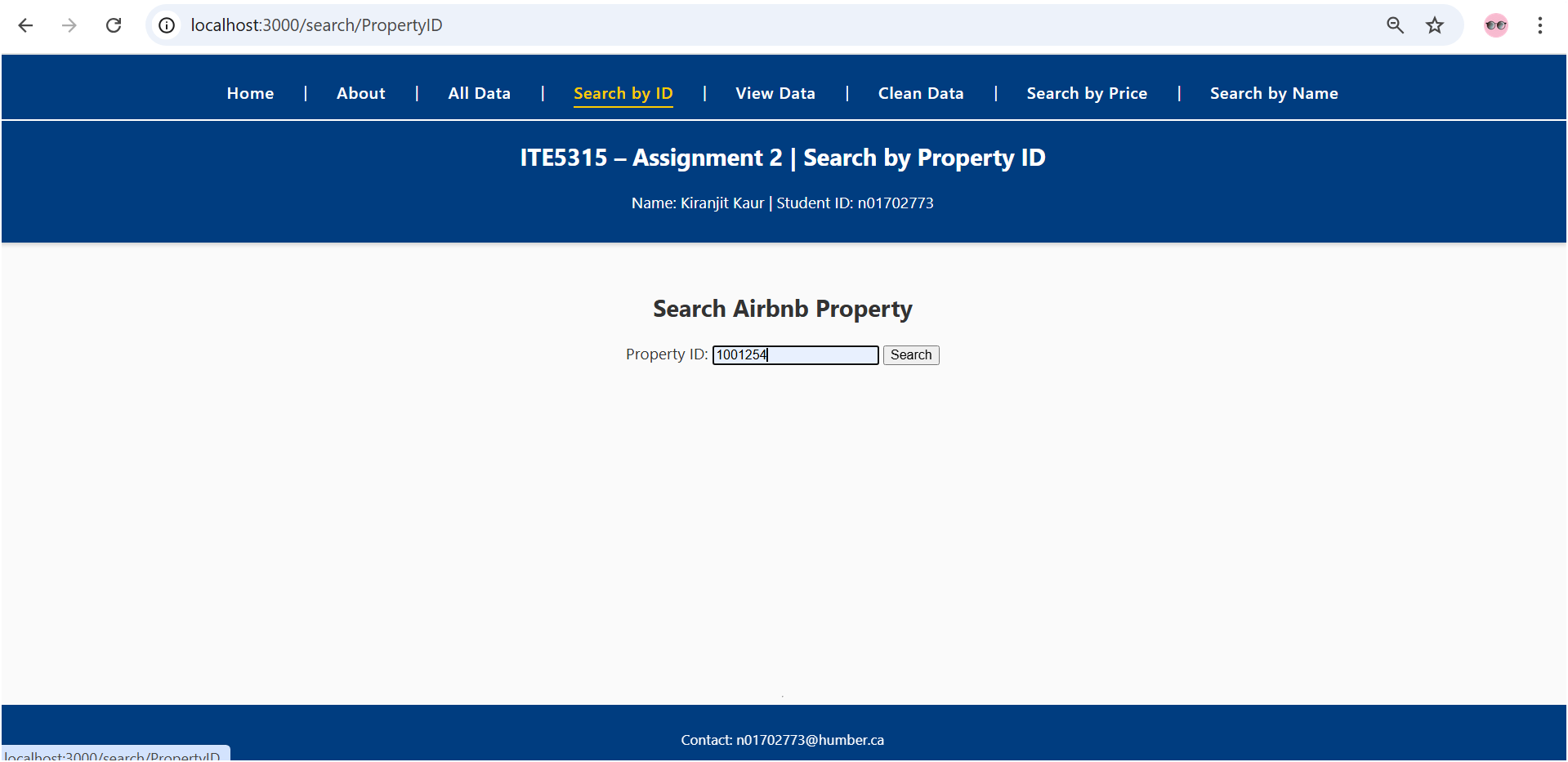
**/allData**

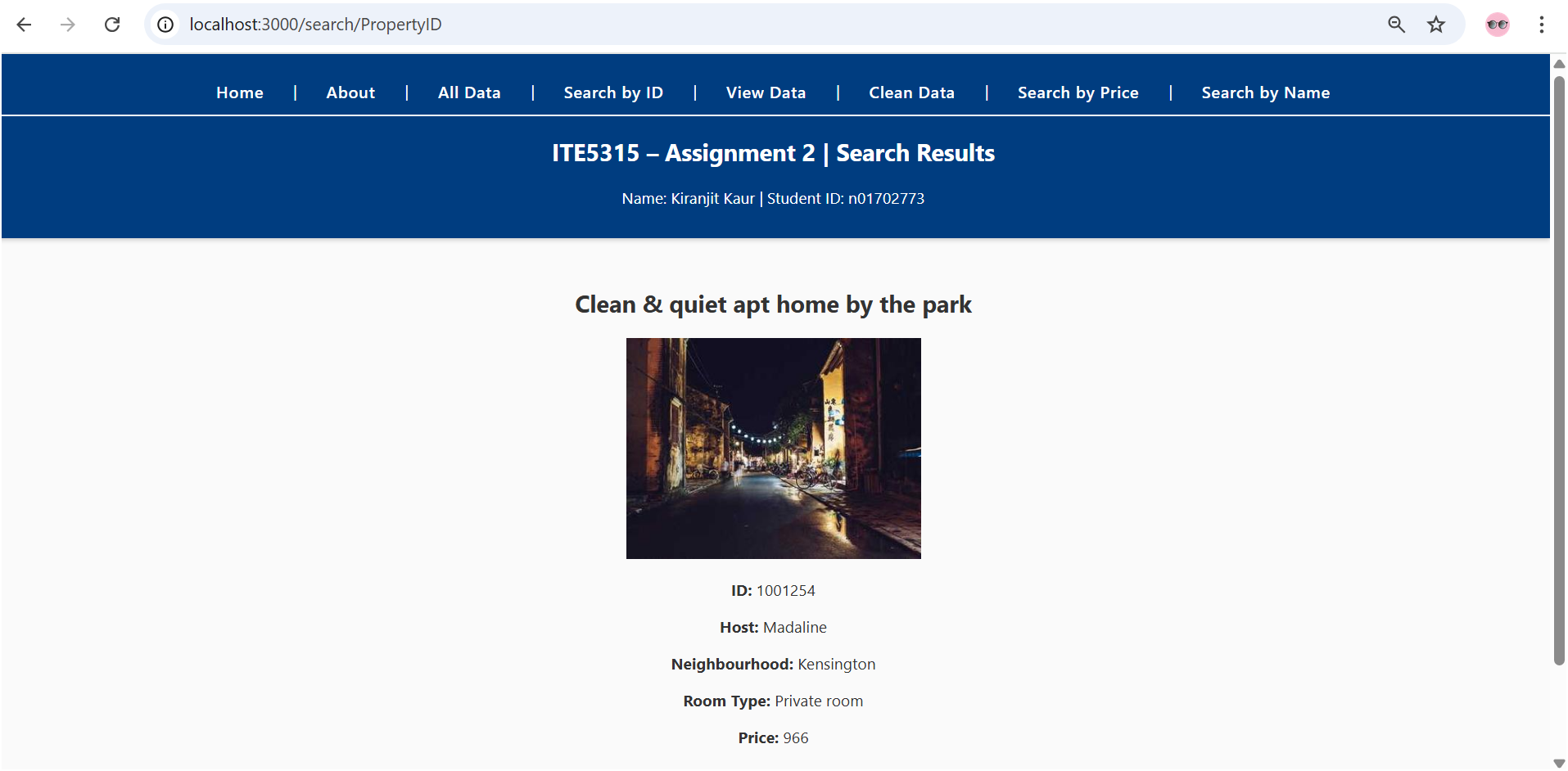
The /allData page now displays all Airbnb data in a formatted table and includes property images. This is unfiltered data.

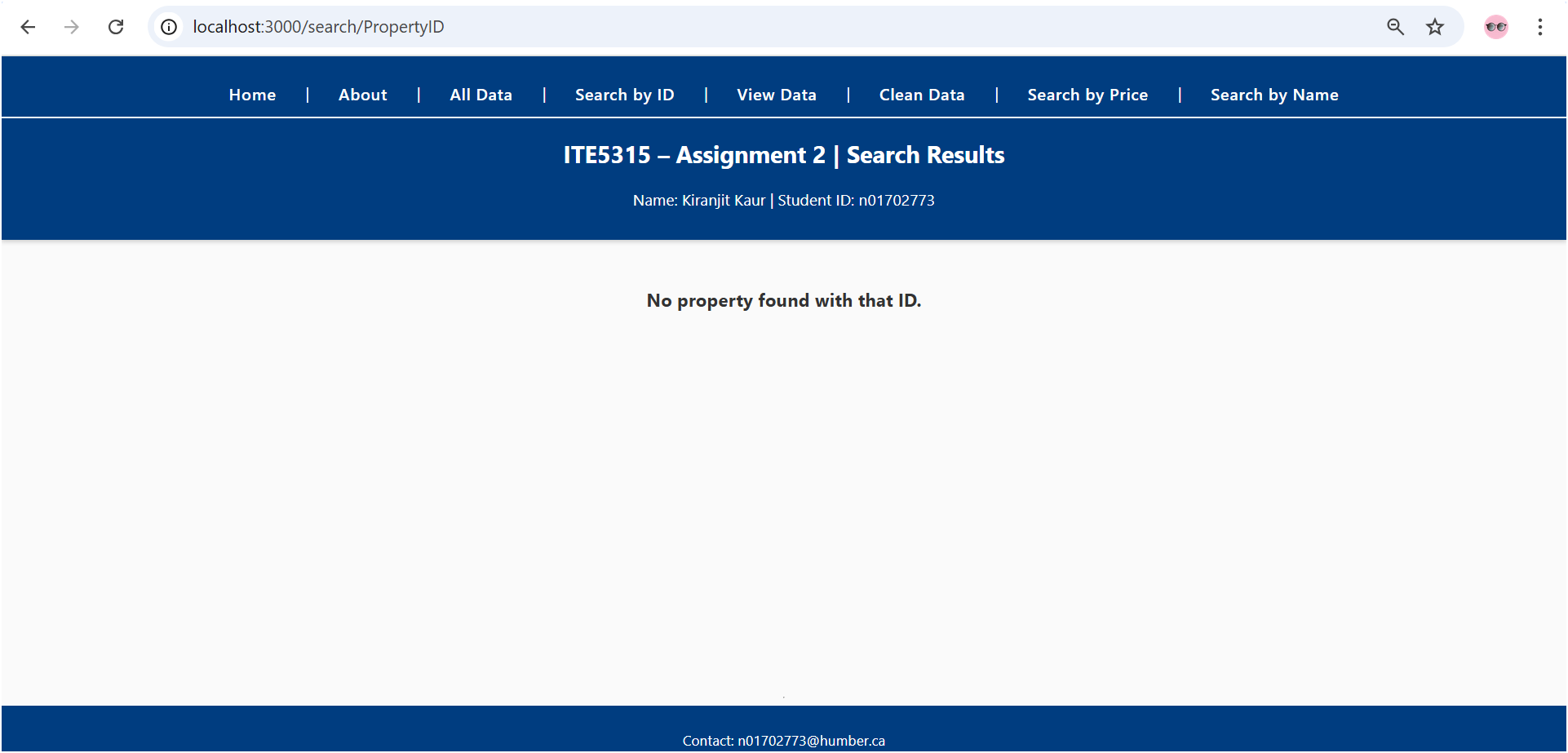


**/search/PropertyID)**

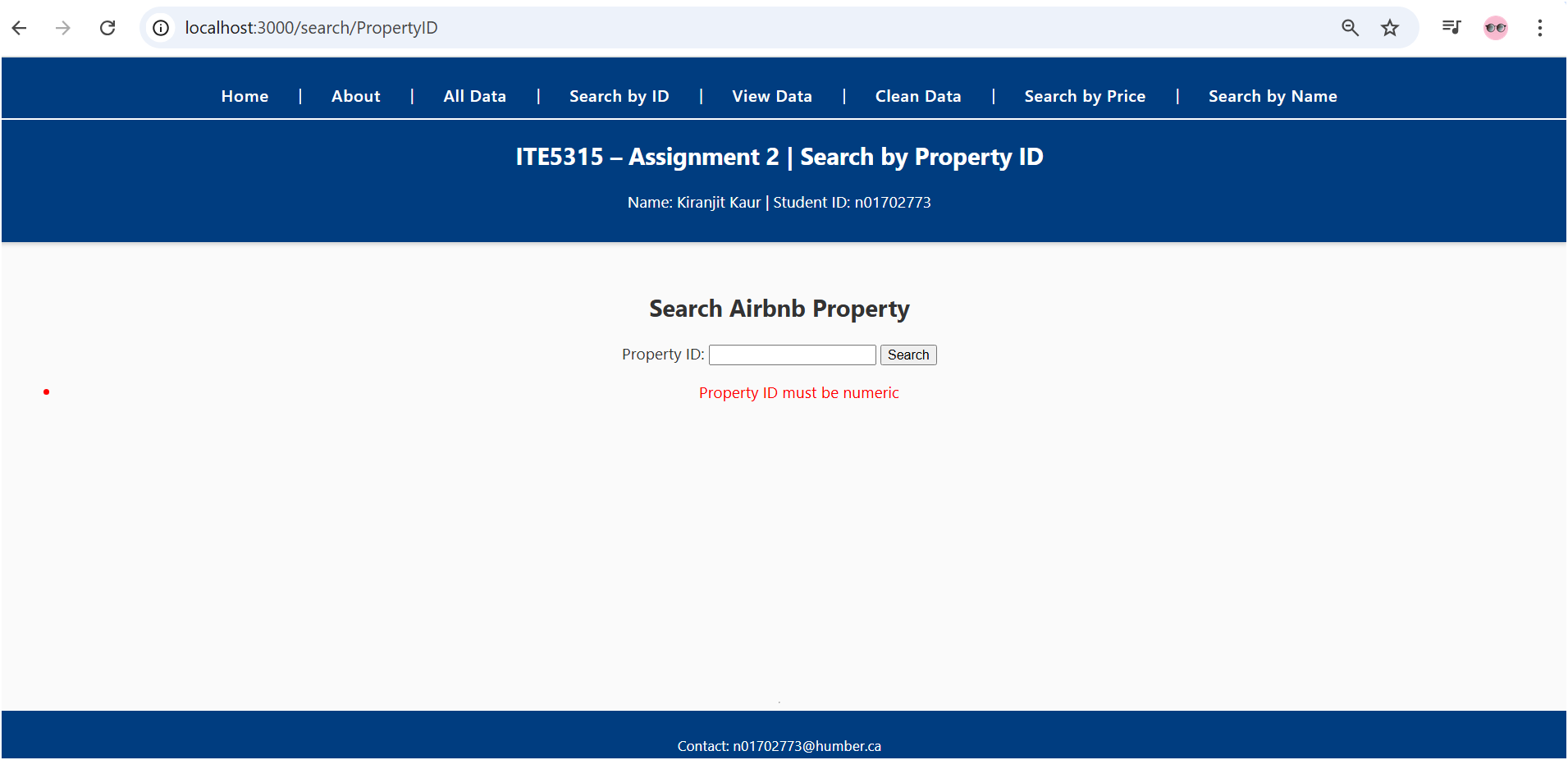
The search route allows users to look up Airbnb listings by their unique property ID. It is implemented in two parts — a GET route that displays the search form, and a POST route that handles form submission and validation. Using the express-validator middleware, the form ensures that the property ID field is not empty, only accepts numeric input, and sanitizes the data before processing. When a valid property ID is entered, the matching record (including its image) is displayed on the searchResult.hbs page. If no result is found, a “Not Found” message is shown. This route showcases how form validation and dynamic rendering are handled securely in Express.





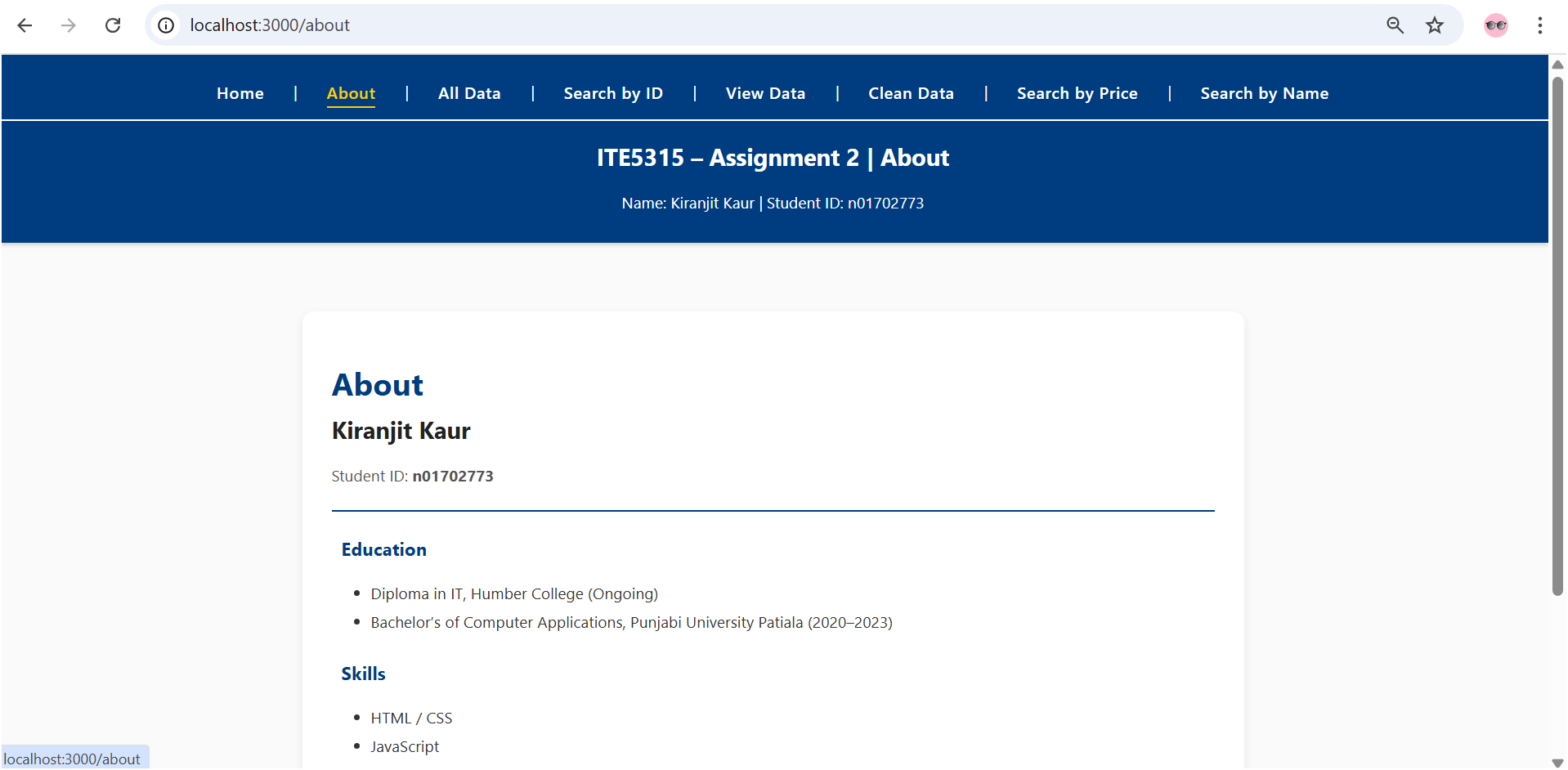


Sanitization is also working here:



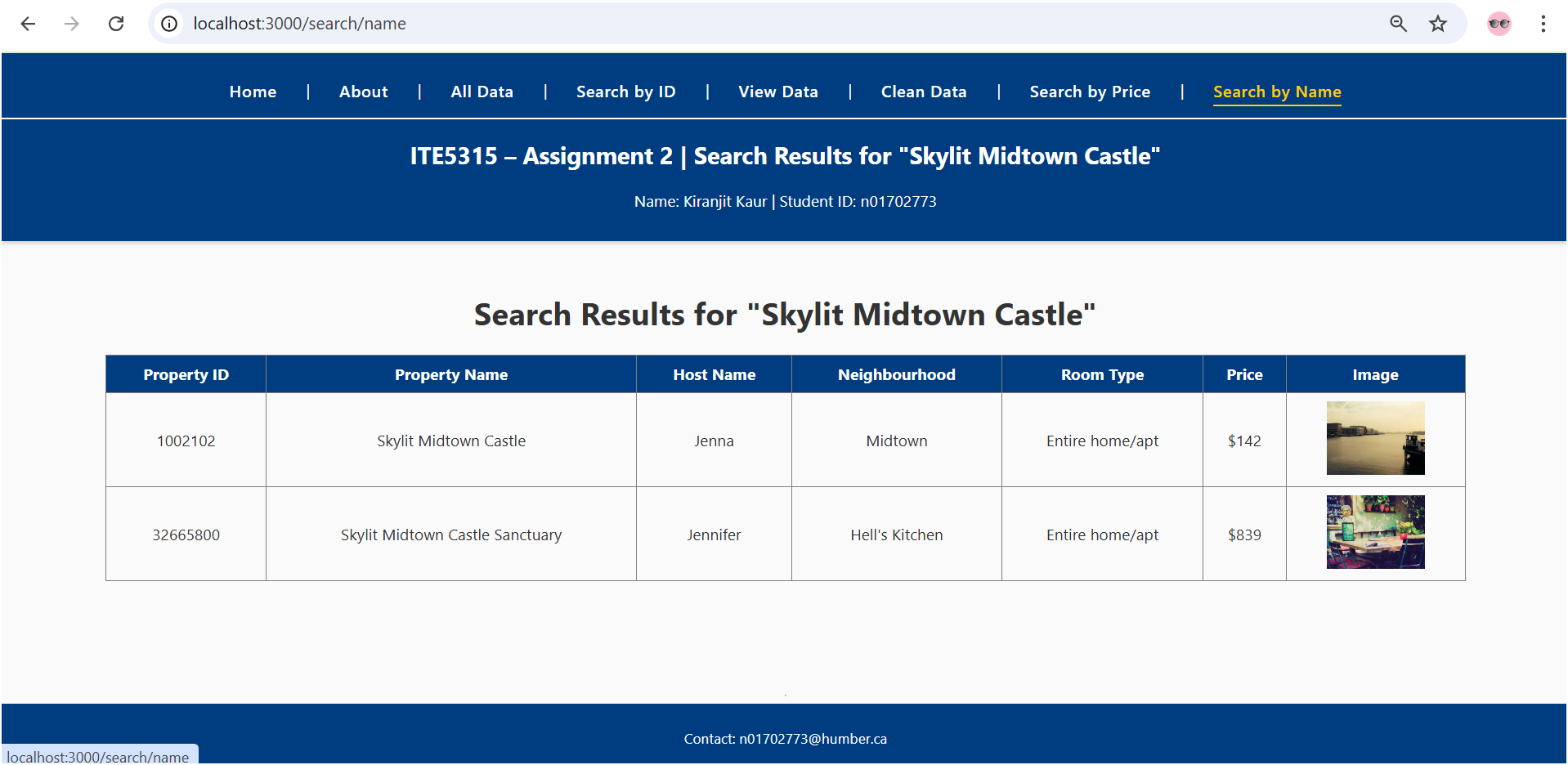
**/about**

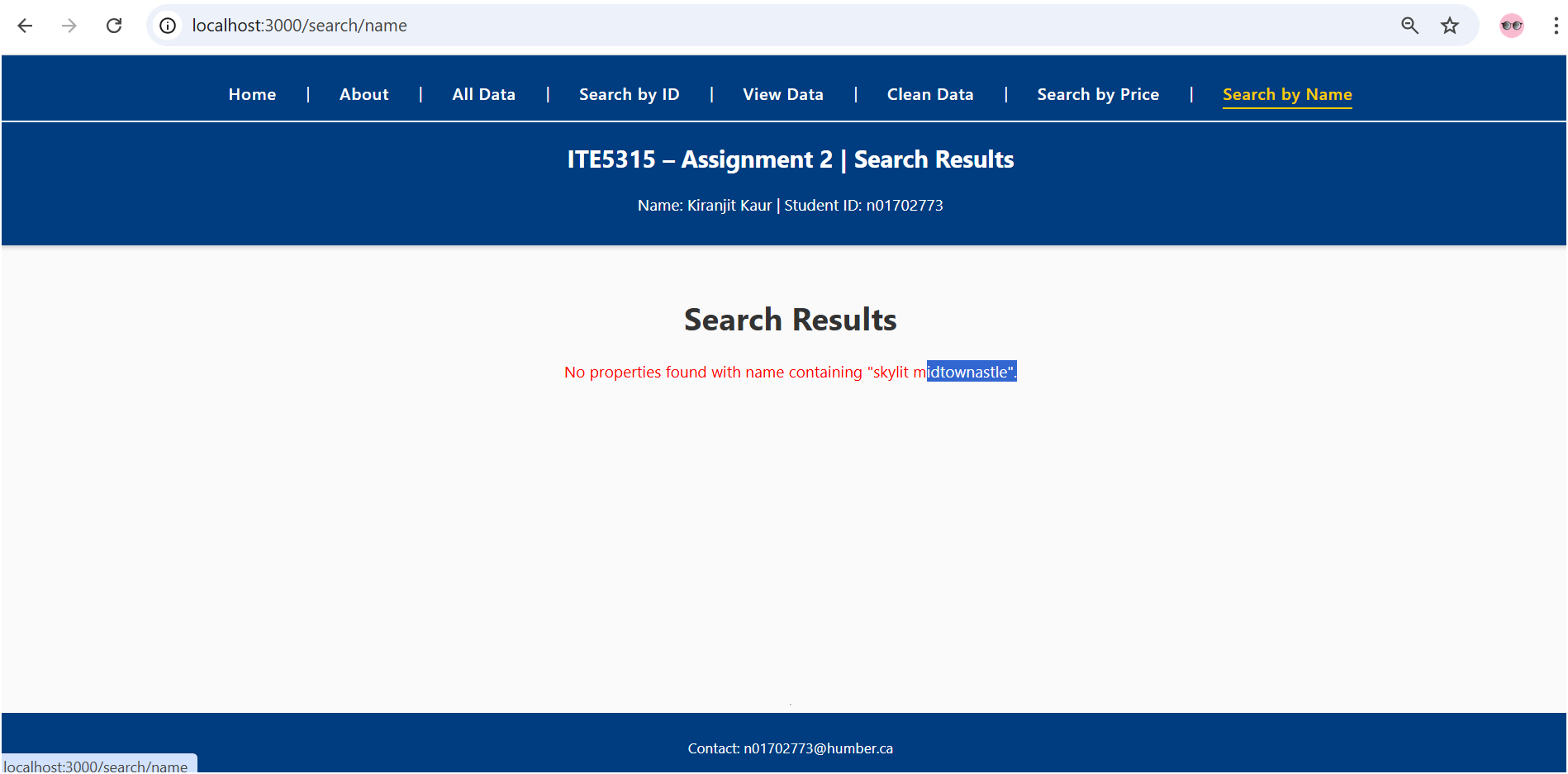
The about route (/about) is rendered using the about.hbs view and the shared layout, maintaining a consistent look with other pages. This page describes my resume as in assignment 1.



**/search/name**

allows users to search Airbnb property listings by name. It displays a form where the user can enter part or full property name and after submitting, the server filters the data to show only the matching properties. The results are displayed neatly in a table using the Handlebars template engine.





**Step 8: Design a new route “/viewData” to display all Airbnb info in html table output using handlebars helpers/expression.**

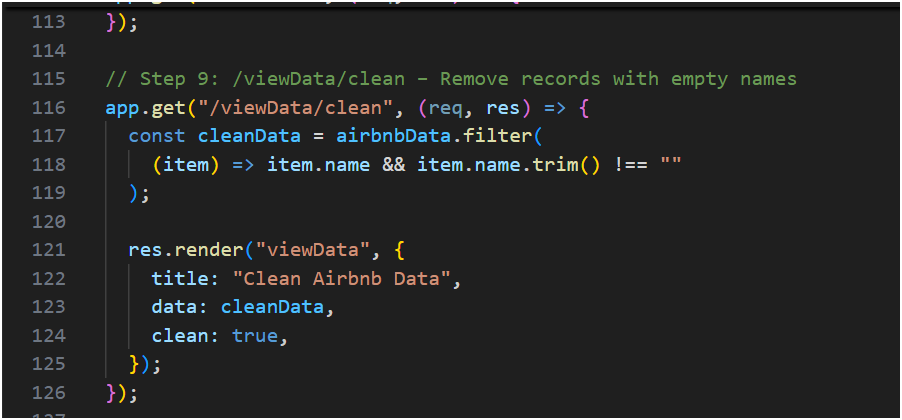
**/viewData**

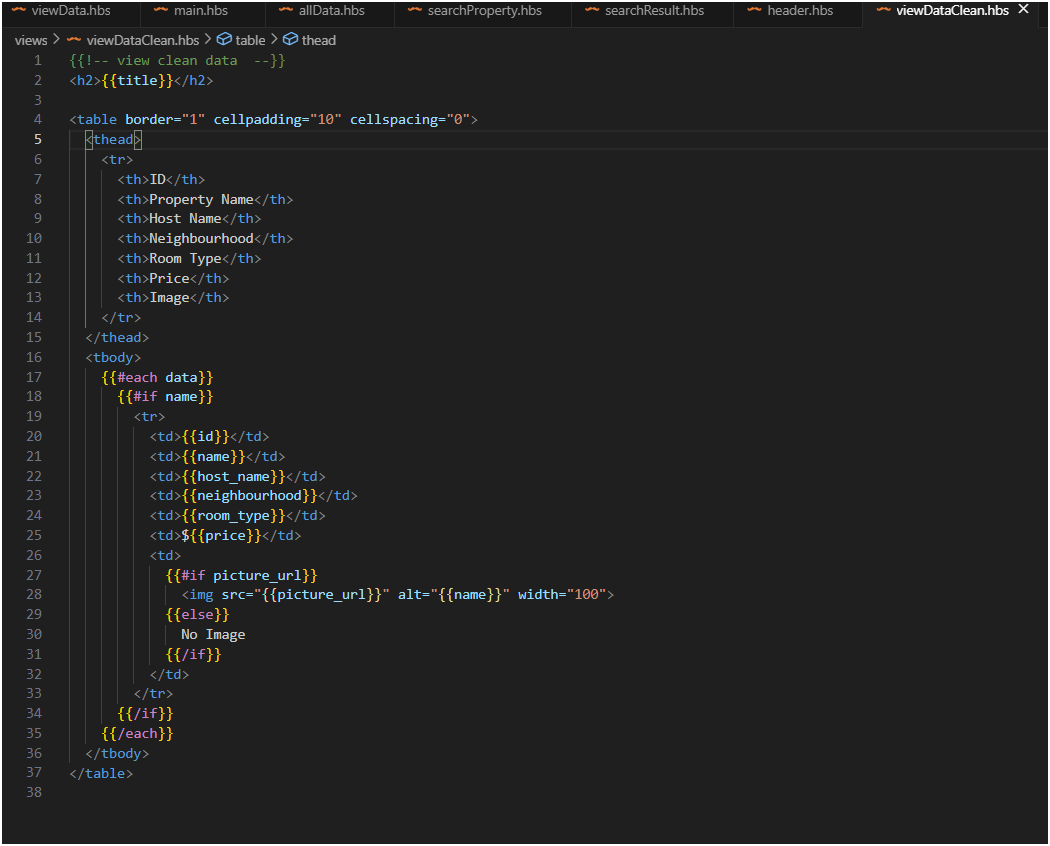
In this step, I created a new route /viewData to display all Airbnb information using Handlebars helpers and expressions.  
Although this page also lists all data in a table, its purpose is to demonstrate the use of Handlebars helpers such as {{#each}}, {{#if}}, and the custom helper {{emptyName}}.  
This step proves that new dynamic pages can be created independently using the templating system introduced in Step 7.

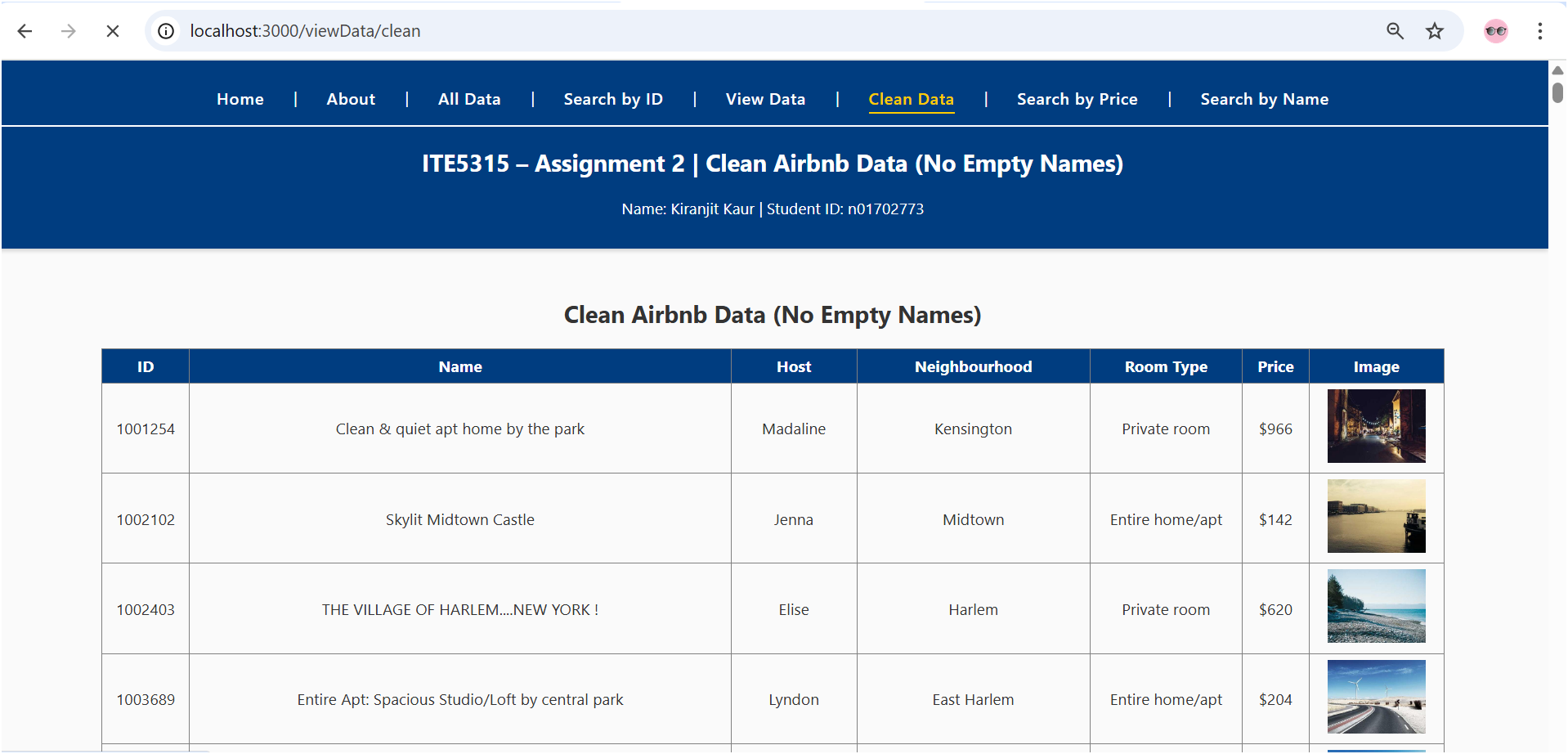
****

Here, n/a is highlighted because of step 10

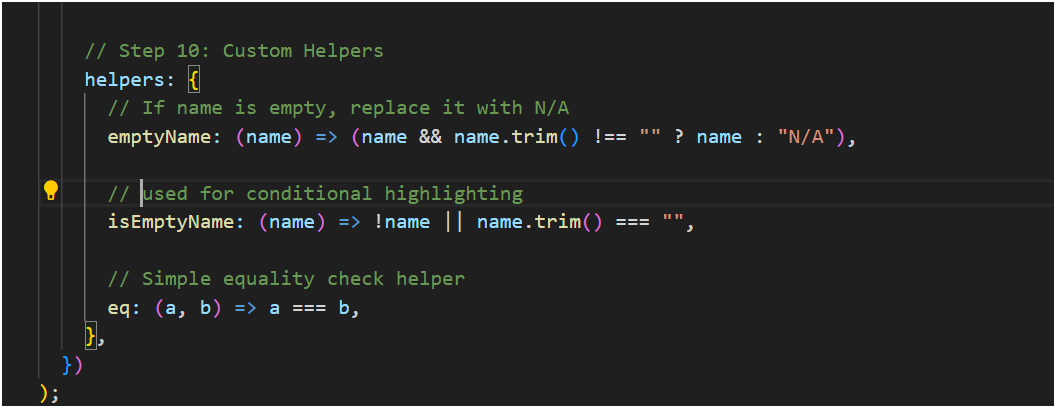
**Step 9: Modify Step 7 (change the endpoint to /viewData/clean) by removing those records which their “name” is empty (*hint: use “#if” helper* ☺)**

****

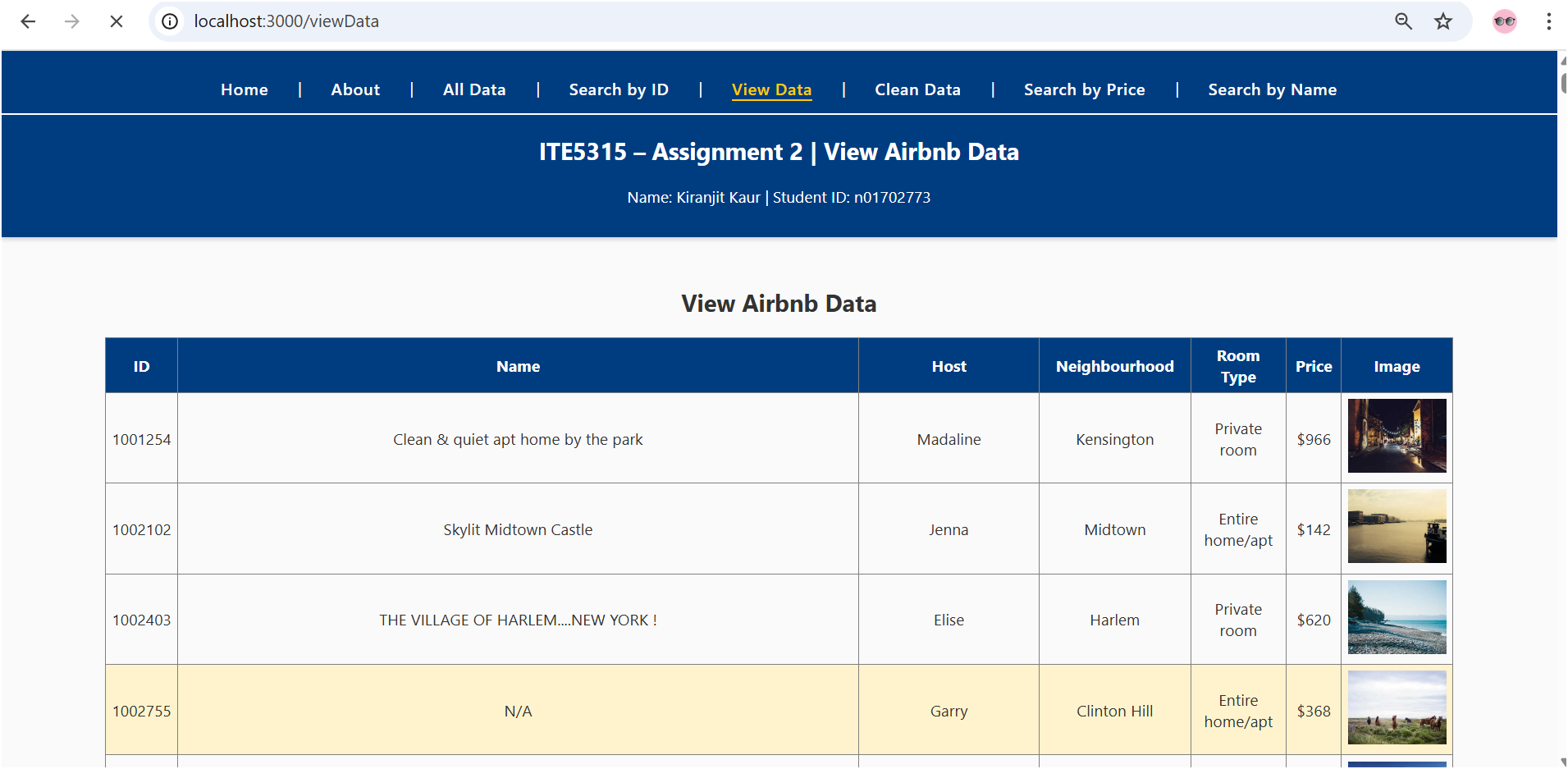
****



**Step 10: Design a custom helper for changing the records with empty “name” to “N/A”. Apply this on Step7&8 in order to display all records but highlight those rows that their “name” is empty!**

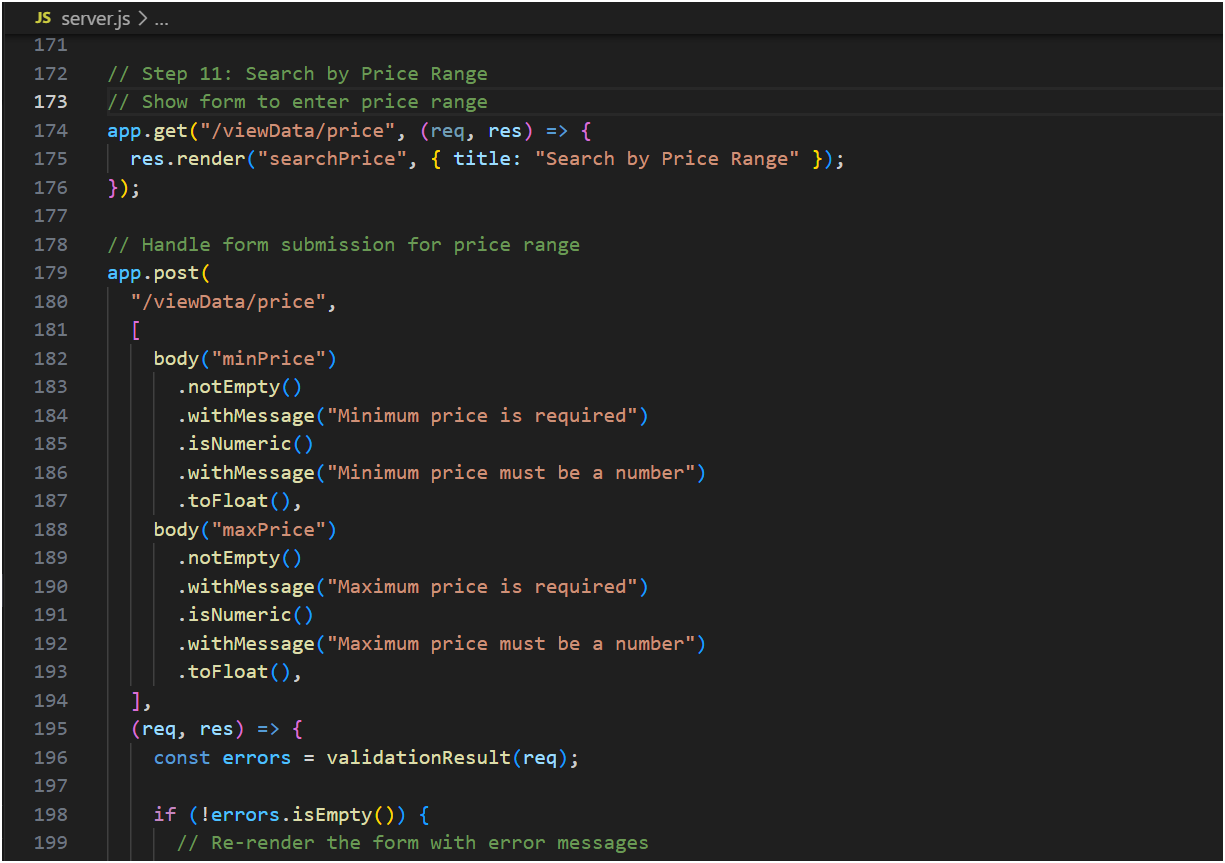


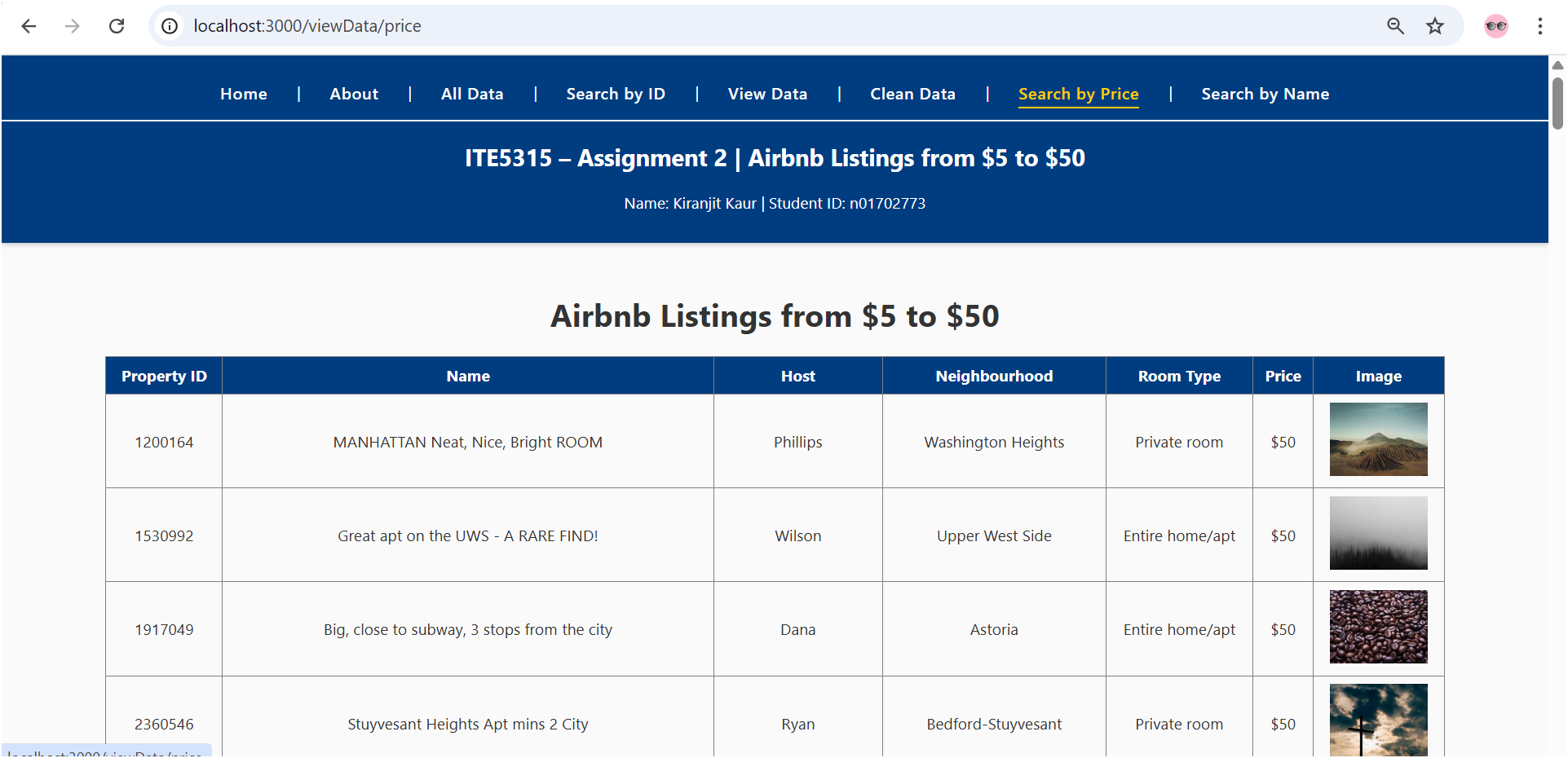


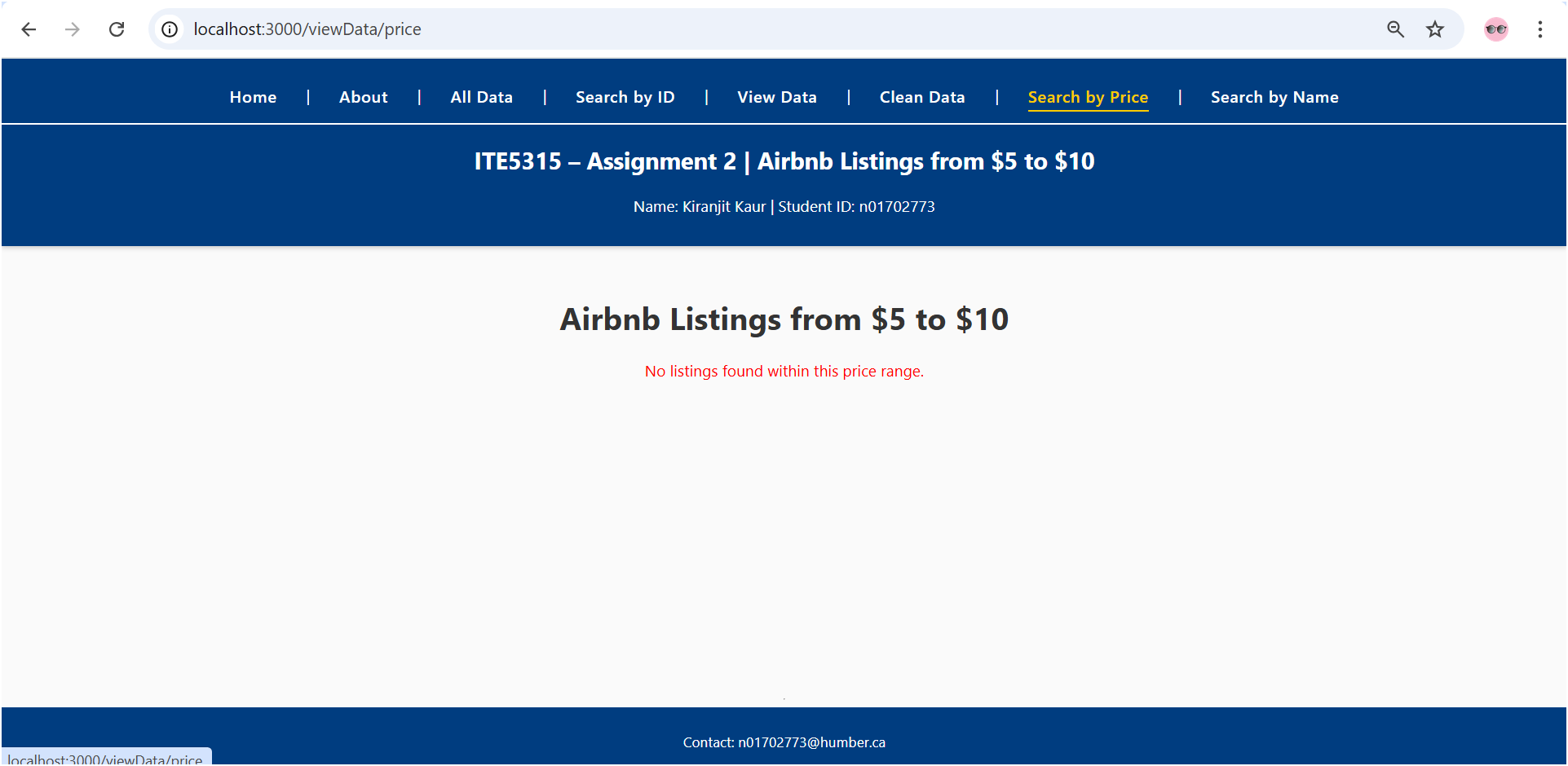


**Step 11: Design a new route (/viewData/price) that opens a new form to get a price range (min and max) from user and return the matching record in that range.**

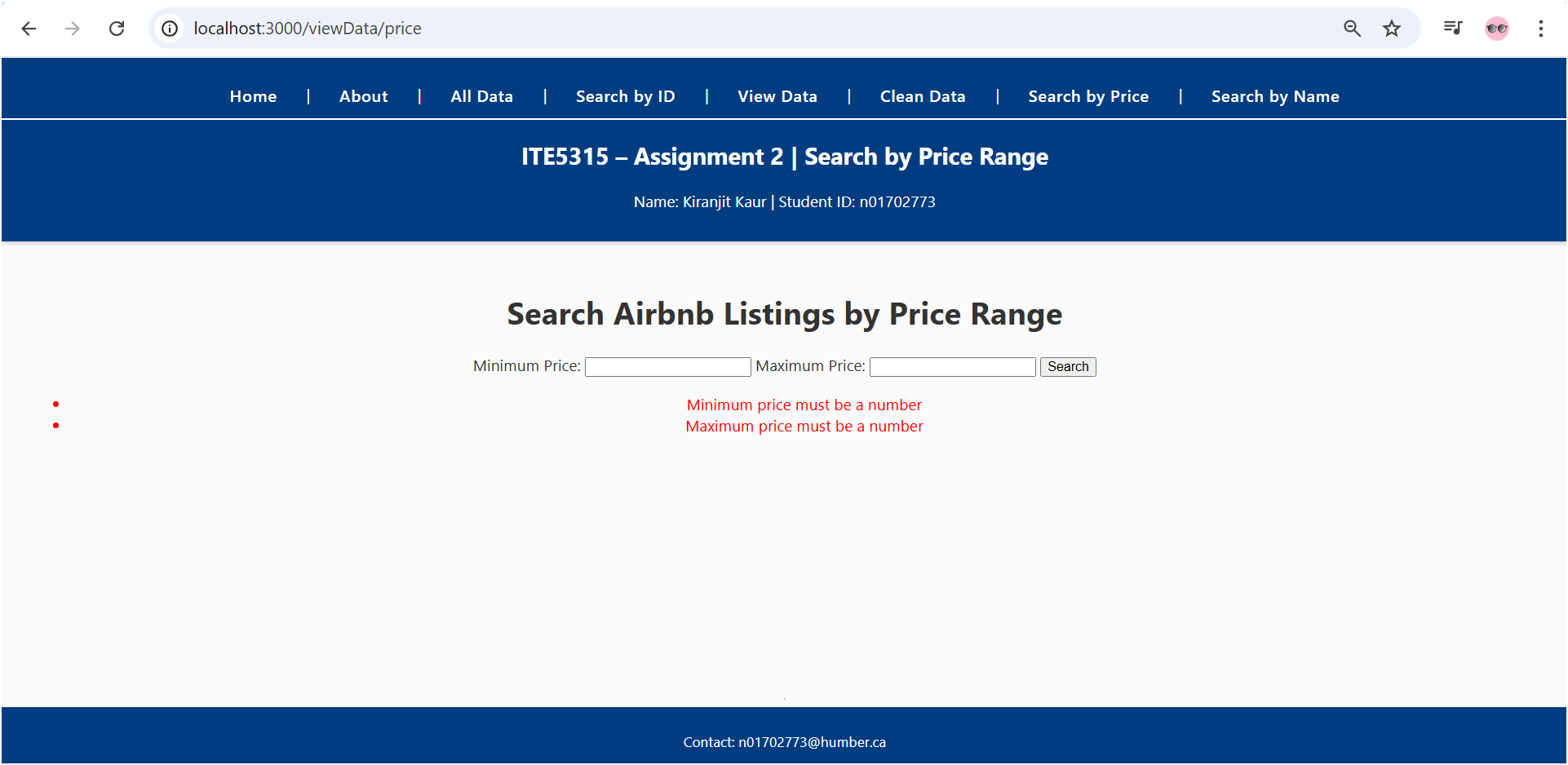
* **To design the form, use proper form validation & sanitization**
* **Use Template Engine to render the output and filter unrelated data (don’t display records outside the given price range)**







Sanitization also works here:



**Step 12: Deploy the app to Vercel.**

* **To allow deployed app being accessible publicly, Turn off protection for Previews o Vercel Dashboard → your Project → Settings → Deployment Protection.**

**o Disable Vercel Authentication (or set Previews to Public).**

* ***Note: you may face some challenges after you deploy the app to load the entire json data. Remember it is better to load the JSON data in async way and in case you are not able to send the entire dataset to the browser (e.g: /viewData), try to display top 100 records!* o I want to see how you solve this challenge**

**Step 13: Answer the following research questions:**

1. **What is “bookmarkable GET searches”? How to apply this in the project? How does this improve UX and Caching?**

Bookmarkable GET search means when we search for something, the search term appears in the **URL**.  
For example: <http://localhost:3000/search/name?keyword=Park>

This is called “bookmarkable” because we can save or share this link, and when someone opens it, they will see the same search results again — no need to type the search again.

**How to apply it in our project:**  
Right now, our form sends data using POST.  
If we use a GET method instead, the search value goes into the URL (like ?keyword=Park).  
Then we can read it in the backend using req.query.keyword and show results.

**How does this improve UX and Caching?**

* Users can bookmark or share their searches easily.
* Browsers can cache GET pages, so the page loads faster if you open it again.
* It feels smoother because users can use back and forward buttons properly.

So basically, GET searches make our site easier to use and faster to reload.

1. **How to you explain this statement: “Vercel is a serverless-first platform”? How does this effect our NodeJS app development/deployment?**

“Serverless-first” means that Vercel doesn’t keep our server running all the time.  
Instead, it runs our code only when someone sends a request and then it stops again.  
It is like having a helper who wakes up only when needed and goes back to sleep after finishing the task.

**How this affects our Node.js app:**

* We don’t need to use app.listen() like in local development — Vercel handles that part.
* Each route (like /search, /about, etc.) becomes a separate function that runs only when someone visits it.
* It automatically handles scaling, speed, and security for us.

Assignment Submission:

* Add the following declaration at the top of .js files

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*

* + ITE5315 – Assignment 2
  + I declare that this assignment is my own work in accordance with Humber Academic Policy. \* No part of this assignment has been copied manually or electronically from any other source \* (including web sites) or distributed to other students.

\*

* + Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Student ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*/

* Compress (.zip) the files in your Visual Studio working directory (this is the folder that you opened in Visual Studio to create your client side code).

Important Note:

* Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.
* **LATE SUBMISSIONS for assignments**. There is a deduction of 10% for Late assignment submissions, and after three days it will grade of zero (0).
* Assignments should be submitted along with a video-recording which contains a detailed walkthrough of solution. Without recording, the assignment can get the maximum of 1/3 of the total.

o In case you are running out of time, you can submit your project on time and submit the video-recording within 24 hours