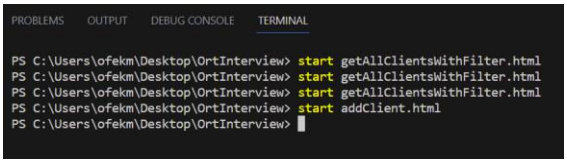
How to run the program

The entire system needs to be activated by running the program in C# (Swagger will open), which will confirm that the backend is running.

To activate the frontend, there are several options available:

- The first option is to choose one of the HTML files in the addClient, deleteClient, filterClients, getAllClients, getAllClientsWithFilter, or getIpGeoInformation folder. Opening any of these files will launch an HTML page, allowing us to start using the project. Each page provides navigation to the other pages.
- The second option is to open the project folder in Visual Studio Code and use the Terminal. By typing 'start' followed by the desired HTML file name (e.g., start addClient) and pressing Enter, the frontend of the project will open.
  Example of the terminal:



- The data base save in Microsoft SQL Server Management studio

assumptions

- The assumption was to create a generic site so that if we want to add another option, an additional tab will be added. Each option would have its own separate page on the site. However, if I had built the site for an end user and not for developers, all the options that could be performed on the site would be on the same page. For example, there would be a table of customers, and next to each customer, there would be a delete button. Clicking on the button would delete the corresponding row.
- The assumption was to keep the customer's phone number within quotation marks as it appeared in the link. I added all the records of the 50 customers at once using a text file in Microsoft SQL Server Management.
- The assumption was that the front-end tests were not conducted on the back-end because both the front-end and back-end were developed by the same person (myself), ensuring the correctness of the information reaching the back-end. However, if the front-end and back-end code were developed in different locations, by different companies or departments, it would be necessary to verify the input in the back-end as well to ensure its accuracy.
- The assumption regarding the page that displays the geographic information of each IP is that the GetGeoInformationOnClientsIP operation utilizes a separate query that accesses SQL and retrieves only the clients' IPs. This approach avoids using the getAllClients function, which fetches all the data on every client, as it is resource-intensive and unnecessary, preventing the transfer of unnecessary fields.
- The assumption is that when adding or deleting a client, we access the SQL again instead of relying on the existing information we have about the database. This is necessary because there might have been changes in the database between the operations.

The assumption is that when it's necessary to display all existing customers in the database, we access the SQL every time. This is because there could be operations such as deletions or additions in between, resulting in changes to the database. If we were to save all the customers in a variable and not access the SQL repeatedly, we wouldn't have the updated information.
If it is known that there were no operations in between and only the information needs to be displayed, it would be possible to save the table and display it each time, accessing the SQL only once.

- The assumption is that the ID serves as a unique key and identifier for each customer. To check if a customer exists, we verify if the provided ID matches an existing customer. Similarly, for deleting a customer, we simply obtain the ID from the user and proceed with deletion.
  Furthermore, the assumption is that the ID alone is sufficient for identification, without relying on the cell phone number. This is because there is a possibility of multiple customers sharing the same cell phone number, such as a parent and child, where the child may have the parent's cell phone. As a result, the same cell phone number can be associated with multiple customers.
  Moreover, each person has a unique identity card, and it is ensured that no two individuals can possess the same id.
- The assumption is that the user knows the IP. When creating a new customer, this assumption is made to avoid the need for developing a mechanism to retrieve everyone's IP.
- The assumption is that when adding a customer to the system, all fields, including Name, ID, IP, and Phone, are mandatory and required.
- The assumption is that when there are more than 45 requests in the API, indicating multiple requests, the corresponding error message '429 Too Many Requests' will be displayed on the screen.

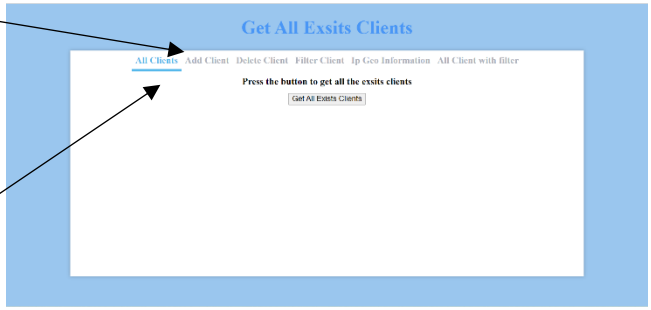
Example of screens

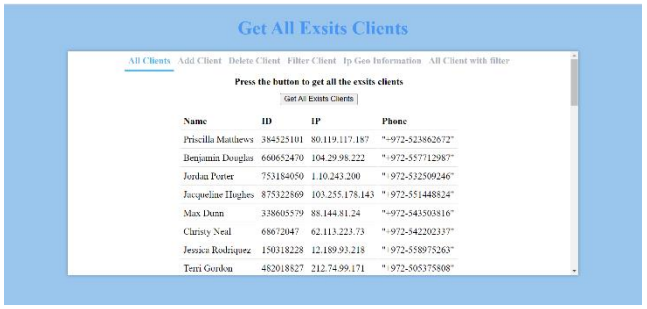
All exception errors returned from the backend are handled and displayed according to the client's specifications.

On the above page, when you click the button, a table displaying all existing customers will appear.

The Pages menu appears on each page and provides options for switching between different pages

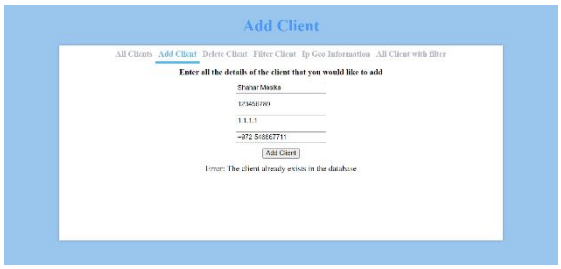
The current page is marked in blue



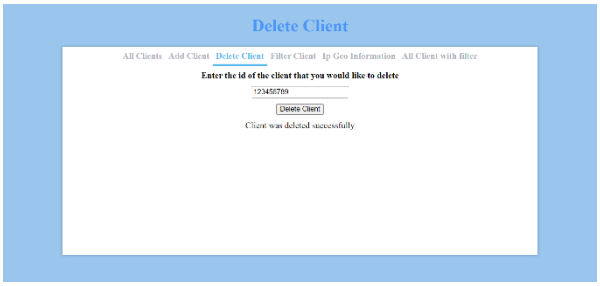


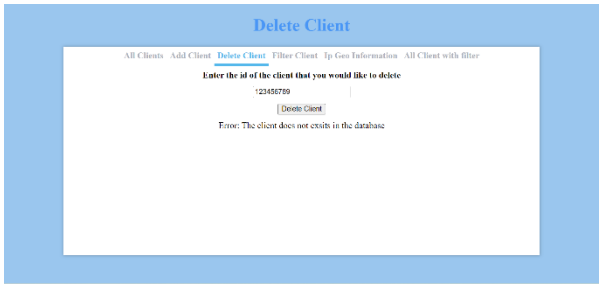
On the above page, when you click the button, the fields are first checked to ensure they like the require. These checks include verifying that the fields are not empty,

the name consists only of letters, the ID field consists only of numbers, the IP field follows the pattern 3digits(0-255).3digits(0-255).3digits(0-255).3digits(0-255), and the phone number is in the format '+972-' followed by 9 digits. If any of the fields are incorrect, an appropriate error message will be displayed. If all the fields pass the checks, the backend then checks if the customer already exists using the provided ID. If the customer exists, an error message will be sent. If the customer does not exist, they will be added to the table.



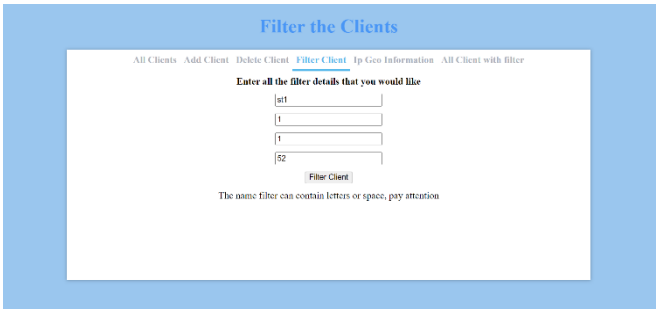
On the above page, when you click the button, the ID field is first checked to ensure it consists only of numbers. If it is incorrect, an appropriate error message will appear. If the ID is correct, the backend then checks if the customer exists. If the customer exists, they will be deleted from the table. If the customer does not exist, an appropriate message will be sent.





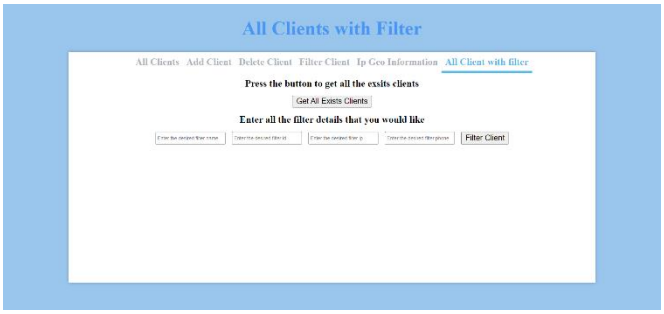
On the above page, when you click the button, a table will appear with the user's desired filter. First, in the frontend, the fields are checked to ensure they contain the appropriate

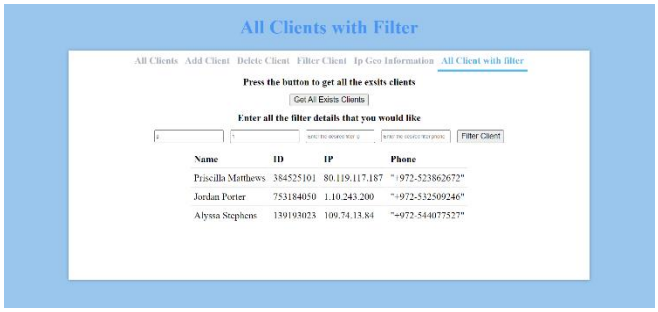
information, such as ensuring the name field only contains letters. Afterward, the backend retrieves the filtered data from the SQL database. The user has the flexibility to filter the fields of their choice without being required to fill in all the fields.



On the above page, when you click the button, the backend first extracts all customers' IPs. Then, an HTTP GET request is made to the website https://ip-api.com/ for each IP to retrieve the corresponding geographic information, which is then presented in a desired table format.

On the above page, when you click the 'Get all clients' button, a table with all existing clients will appear. When you click the 'Filter clients' button, a table will appear based on the user's desired filter. First, in the frontend, the fields are checked to ensure they contain the appropriate information, such as ensuring the name field only contains letters. (e.g., name





contains only letters). Afterward, the backend retrieves the filtered data from the SQL database. The user has the flexibility to filter the fields of their choice without being required to fill in all the fields.