UNIVERSITY of BRADFORD

Large Scale Data Driven Applications	Interactive Dashboard Documentation	Version 1.0
Sameer Uddin 20004135		

Contents

Purpose

The purpose of this coursework is to implement the learnings from the first part of this coursework into an interactive dashboard that can be used by the clinic to gain valuable insight into their patients, appointment history and how this may affect no-shows for appointments. The dashboard should help the clinic gain insight into various factors that can affect no-shows by factors such as patient demographic, medical conditions, reminder effectiveness and appointment history.

The interactive dashboard will visualise data by using graphs (D3.js) and give the clinic filtering options to further explore what contributes to no-show rates. The graphs and information provided should also help the clinic identify what is causing no-show rates and should give them ideas on what can be done to reduce these.

The ultimate objective of this dashboard is to help reduce no-shows which should improve the overall quality of service that can be provided by the clinic. This will be achieved by increasing the patients turn out rate and reducing the amount of time that is wasted. This dashboard will provide the clinic and its staff with key information to increase their efficiency.

Design

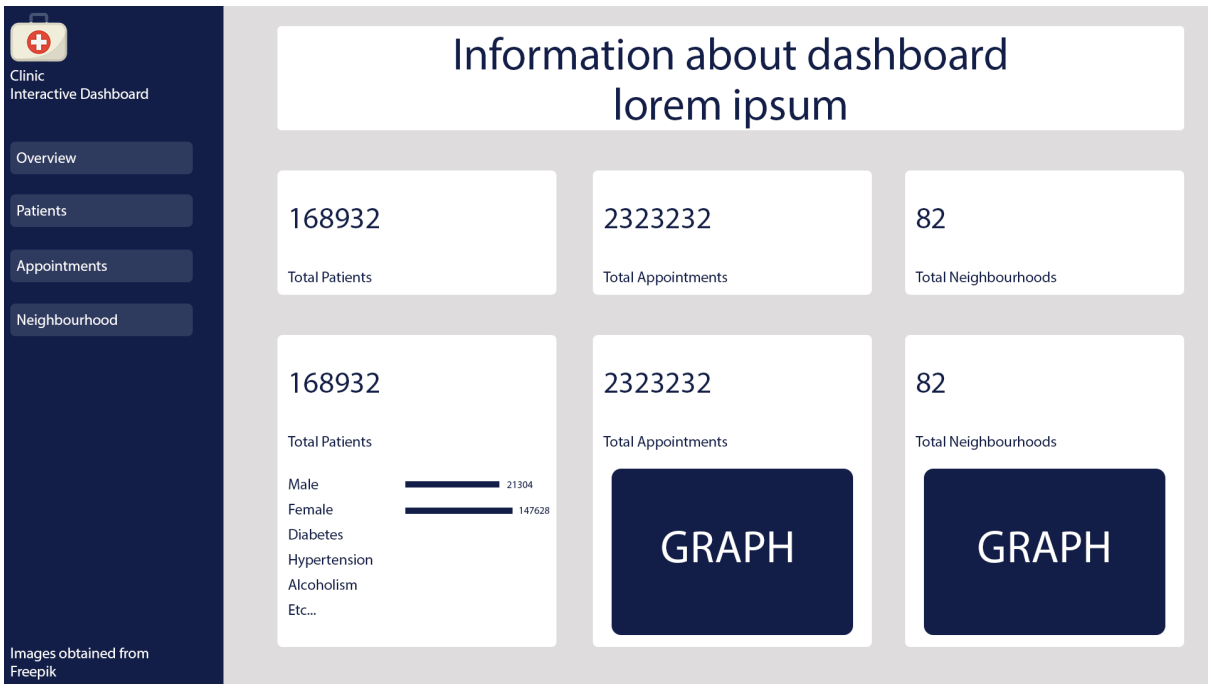


Figure 1 - Homepage Design

The landing page or the home page will act as an overview of the database. It will provide insight into the data. This page will offer quick access to important metrics and include graphs that users can view to get a quick overview of the database.

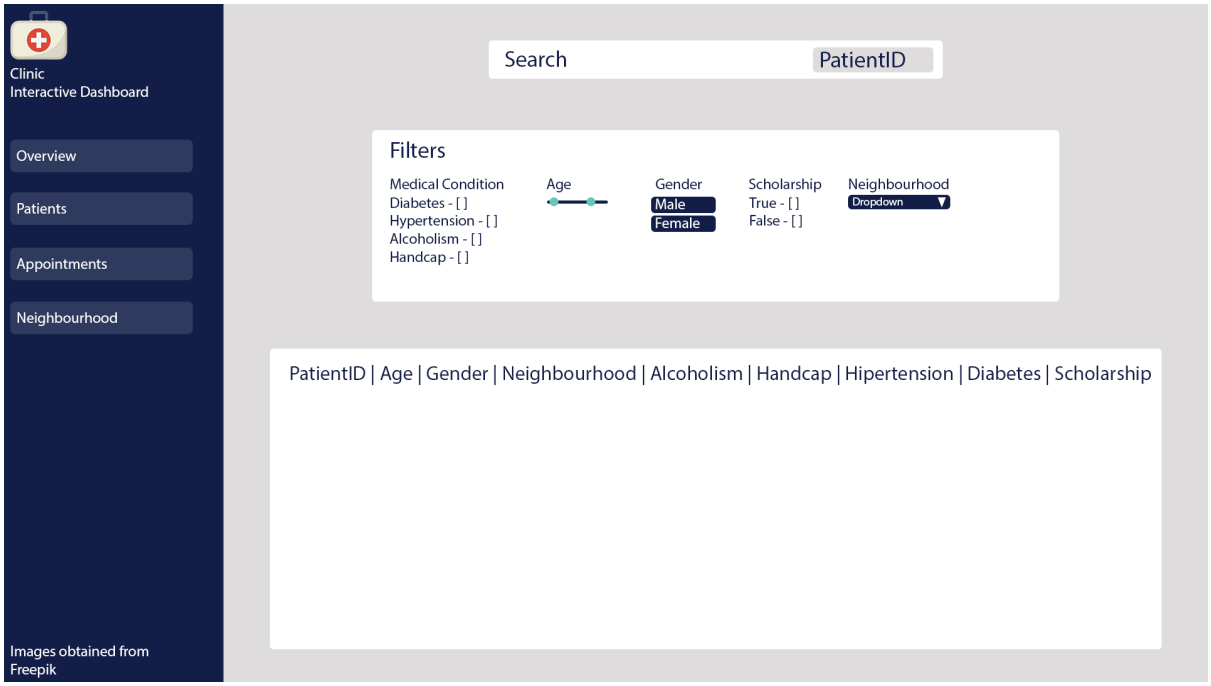


Figure 2 - Patients Page

This page will provide information on the patients in the database. This page will not provide information on appointment history. This page will allow the clinic to search for patients based on their ID (as name is not available). They can also filter the table by other categories such as medical conditions, age, gender, scholarship and neighbourhood.

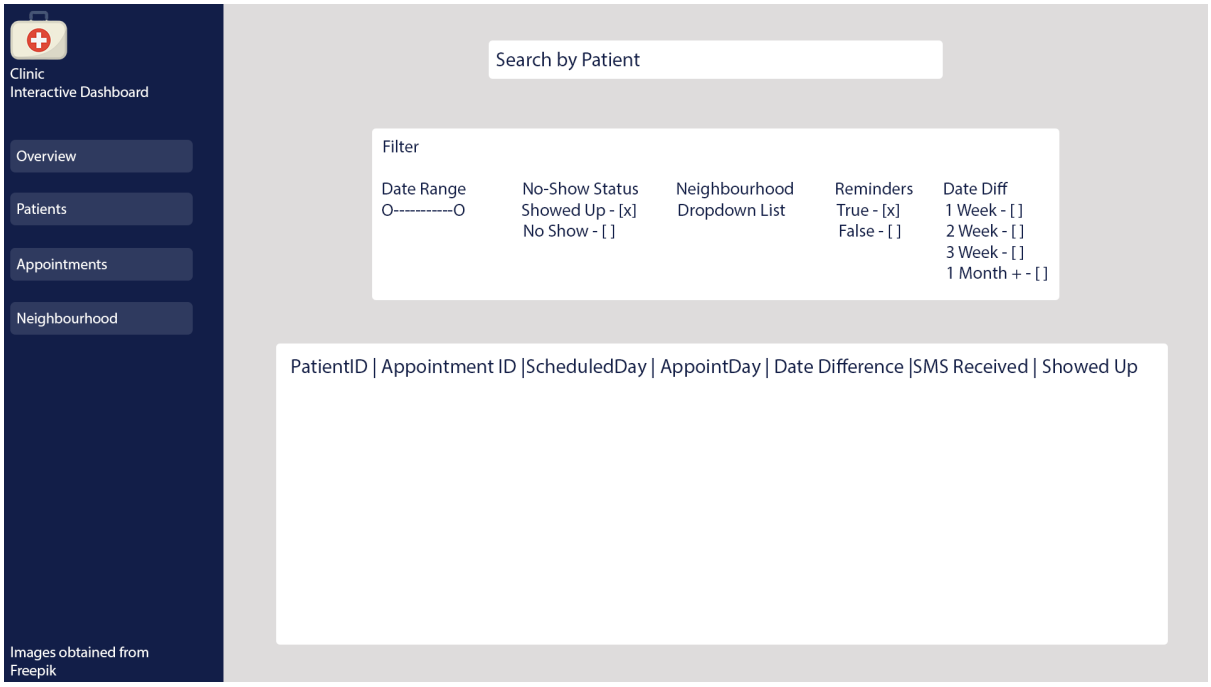


Figure 3 - Appointments Page

This page will provide information on appointments. The table will have key information on appointments such as the patients ID, appointment ID and other information like key dates. Based on the design the user should be able to filter by date range, show status, neighbourhood, reminders and the date difference between booking date and scheduled date.

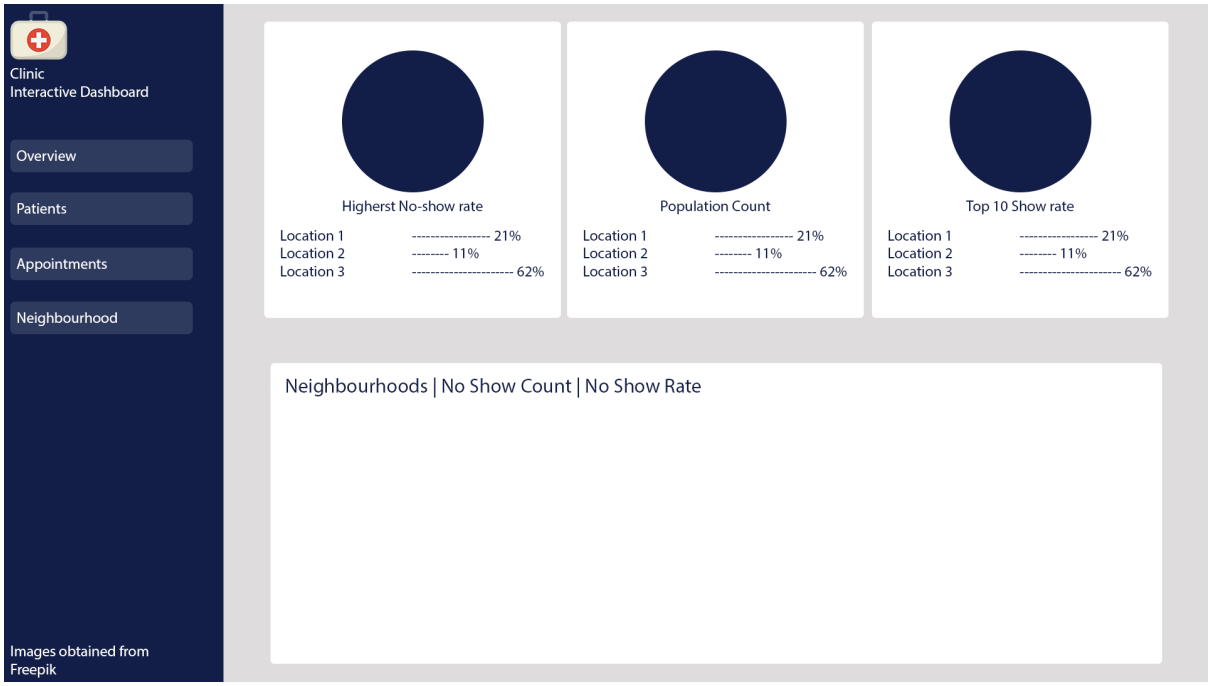


Figure 4 - Neighbourhood Page

This page will provide information on neighbourhoods. This page should have a table with all the lists of neighbourhoods and provide key information based on these neighbourhoods. This page will display metrics such as the neighbourhoods with the highest no-show rates, population of each neighbourhood and the neighbourhoods that have the lowest no-show rates.

Implementation

The implementation of the dashboard is done through the use of frontend languages such as:

- HTML
- CSS
- Bootstrap
- JavaScript

Database connections and filtering data is handled by PHP. PHP is used in this project to connect to the database and retrieve the information from the database which can then be used in the frontend of the application.

The database used in for this project is MongoDB. This database stores the information for both the patients and appointments. Each page utilises each table to fetch and populate tables with information. The overview page uses both databases to summarise key information that were queried in the first part of the coursework.

Figure 5 - Database Schema

Frontend

HTML and some basic CSS is used to structure the pages. The basic CSS used is used to align some items and change colours of components.

Bootstrap was utilised in this project as the grid system is extremely useful and a lot of components will be added to this page. The overview contains a lot of graphs and the grid system ensures that graphs do not overlap each other. The patients and appointments pages have a row for filters and a row for the table that will be generated. The grid system also made it easier to implement the navigation bar to the left side and have the right side of the page entirely dedicated to the contents of the given page.

JavaScript was another important part of this project as it allows for interactivity. It also handles user activities such as resetting the filters, applying the data, populating the data in the table by using PHP scripts and generating the dynamic graphs for the overview page. JavaScript also enables us to use pagination which will reduce the load on the database. This database is large (appointments contains over 100,000 records) and pagination allows the load to be spread over pages.

Backend

The backend is handled by using PHP. Some essential files include the “db_connection.php” file as this file is used to connect to the remote database.

```php
<?php
require 'vendor/autoload.php'; // include Composer's autoloader

use MongoDB\Client; // Import the MongoDB Client class

class Database // Create a Database class to handle database
connections. Used in most other files
{
    public $database; // Create a public variable to store the database
connection

    public function __construct() // Create a constructor method to
connect to the database
    {
        $uri =
"mongodb+srv://su042002:r0m2ZGaYscBSPW9r@cluster0.wkif8.mongodb.net/";
// MongoDB connection URI
        $client = new Client($uri); // Create a new MongoDB client
        $this->database = $client->Coursework1; // Select the database
to use
    } etc…
```

This file also contains some methods used to filter data from the database whether it is simply counting documents returned or using the aggregation pipeline function of MongoDB.

```php
public function getTotalNeighbourhoods() // Method to get the total
number of neighbourhoods in the database
{
    $patients = $this->database->Patients;

    $result = $patients->aggregate([ // Use aggregation to group by
Neighbourhood and count the number of groups
        [
            '$group' => [
                '_id' => '$Neighbourhood' // Group by Neighbourhood
field
            ]
        ],
        [
            '$count' => 'TotalNeighbourhoods' // Count the number of
groups
        ]
    ]);
```

Other key files include “filter_patients.php” and “filter_appointments.php” as these files are used by the front end to fetch filtered data that is return as JSON data. They also check if certain filters are empty or if they contain values to enable the user use a mixture of filters to fetch information.

Database

The database used is MongoDB. This database uses embedded design as designed in part 1 of this coursework. The two tables' patients and appointments contain all the data needed for the dashboard. The connection is created using via Composer to allow to enable the MongoDB PHP library.

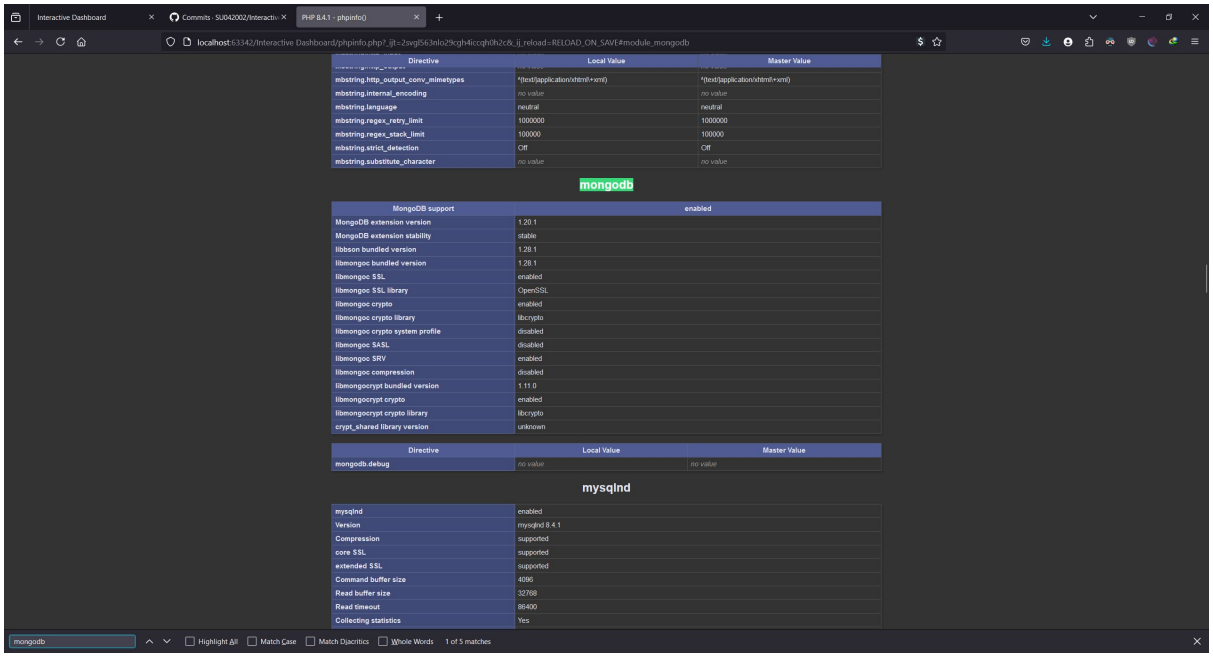


Figure 6 - MongoDB Library

Interactions

The user interacts with the front end of the application by using the filters:

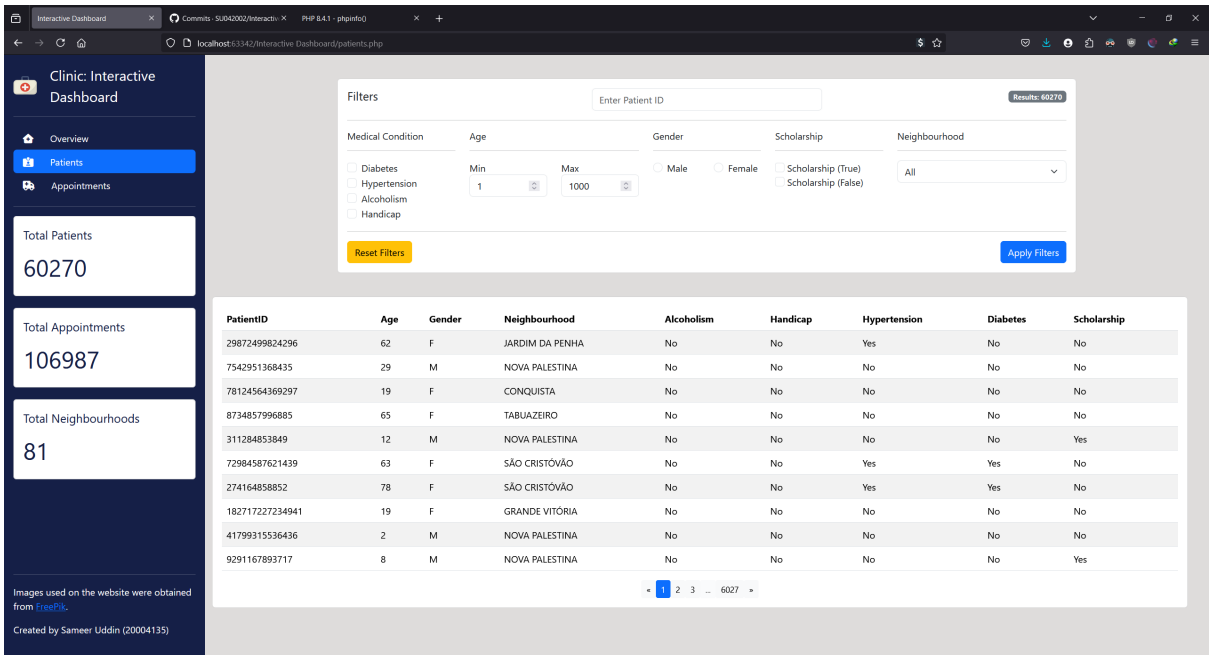


Figure 7 - Patients Dashboard

When the webpage originally loads the JavaScript files loads all the data (limit 10) with the help of pagination. This prevents all the data from being loaded at once as this would use too much ram and slow down the application. When the user applies the filters JavaScript sends a fetch request to the corresponding PHP filter file. This fetch request is processed against the database and the data is returned as JSON data that can be used by the JavaScript files. The data is then displayed on the frontend by updating the HTML tables by utilising their ID.

When the user applies a filter the JavaScript file sends a fetch request:

```
fetch('filter_patients.php', { // Fetch the filter_patients.php file
    method: 'POST', // Use the POST method
    headers: { 'Content-Type': 'application/json' }, // Set the
Content-Type header
    body: JSON.stringify(filters) // Set the body to the filters object
as a JSON string
})
    .then(response => response.json()) // Parse the response as JSON
    .then(data => { // Handle the parsed data
        console.log('Filtered data:', data); // Log the data to the
console
        displayPatients(data.data); // Call the displayPatients
function with the data
        updatePaginationControls(data.totalCount, data.currentPage,
data.limit); // Call the updatePaginationControls function with the
total count, current page, and limit
        document.getElementById('resultCount').textContent = `Results:
${data.totalCount}`; // Set the result count text content
    })
    .catch(error => { // Handle any errors
        console.error('Error:', error); // Log the error to the console
        alert('An error occurred while applying filters.'); // Show an
alert to the user
    });
```

The PHP file queries the database:

```
// Age Range Filter
if (!empty($filters['AgeMin']) && !empty($filters['AgeMax'])) { // Age
Range is not empty
    $query['Age'] = ['$gte' => intval($filters['AgeMin']), '$lte' =>
intval($filters['AgeMax'])]; // Add the Age Range to the query
}
```

The data is then sending back the data based on the filters applied and returns it in JSON format:

```php
// Execute the query
$result = $patients->find($query, ['skip' => $skip, 'limit' =>
$limit])->toArray(); // Get the records based on the query and limit
the records per page
$totalCount = $patients->countDocuments($query); // Get the total count
of records based on the query

echo json_encode([ // Return the result as JSON
    'data' => $result, // Records
    'totalCount' => $totalCount, // Total count of records
    'currentPage' => $page, // Current page
    'limit' => $limit // Limit of records per page
]);
```

The dashboard has been designed to allow the users to gain insight into patient and appointment insights by applying filters to the data. The overview page provided quick access along with graphs of common enquiries.

Filtering Data and Interpreting Data

Overview Page

The main page of the application is where all the charts and visualisations are stored. This page was created as a quick way to interpret interesting data. The charts are dynamically updated when the database is refreshed. A lot of these charts were created from questions that were asked in part 1 of the coursework. This page can be furthered developed to include more charts that will allow users to quickly answer any frequently asked questions or to query interesting information.

This page includes total patients' information, total appointments, total neighbourhoods and other information such as no-show rates.

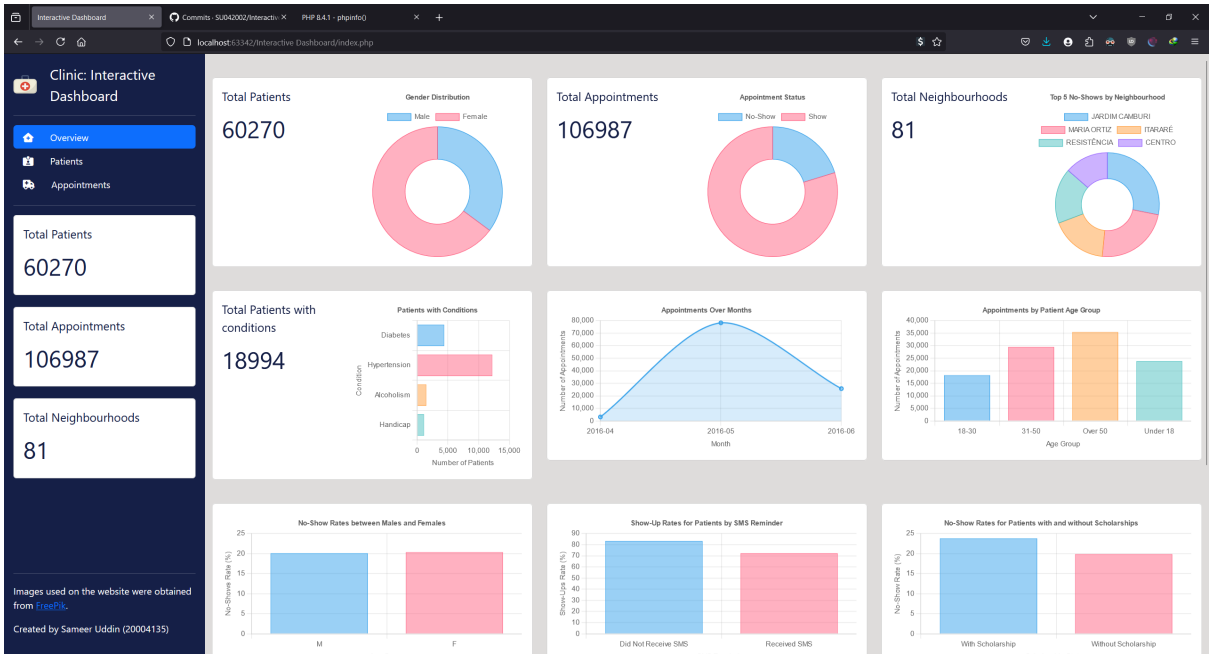


Figure 8 - Overview Page

Patients Page

The patients page includes many filters to allow the user to gain insight into patient information. The result badge in the top right of the filter panel shows the total amount of patients in the database. The user can individually search for users in the database using their Patient ID. They can also filter by using the other criteria provided such as medical condition and age. One example of a query would be searching for users that are under 18 with diabetes in the neighbourhood “Bela Vista”:

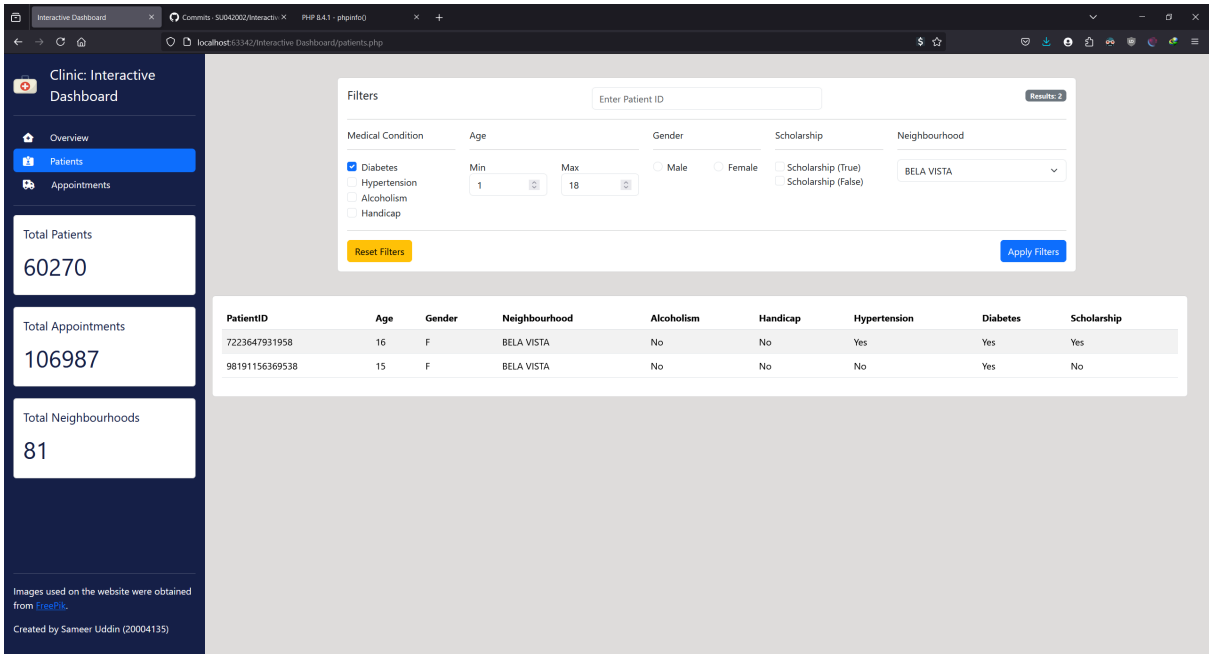


Figure 9 - Patients Page

The table is populated when the user applies the filter. In this case the query returned 2 patients who fall under this category.

Appointments Page

The same logic from the patient’s page is applied to the appointments page. The user can apply queries to the appointments page to gain insight into the data. The database on load contains 106987 rows of data. The user can filter by appointment date, showed up, SMS received, date difference, patient ID and appointment ID. This table does not contain information such as patient neighbourhood as this data can be queried in the patient’s page. The default range for appointment date is currently set for 2016-04 to 2017-07 to encompass all appointments (information obtained from overview page):

Figure 10 - Appointments over Months

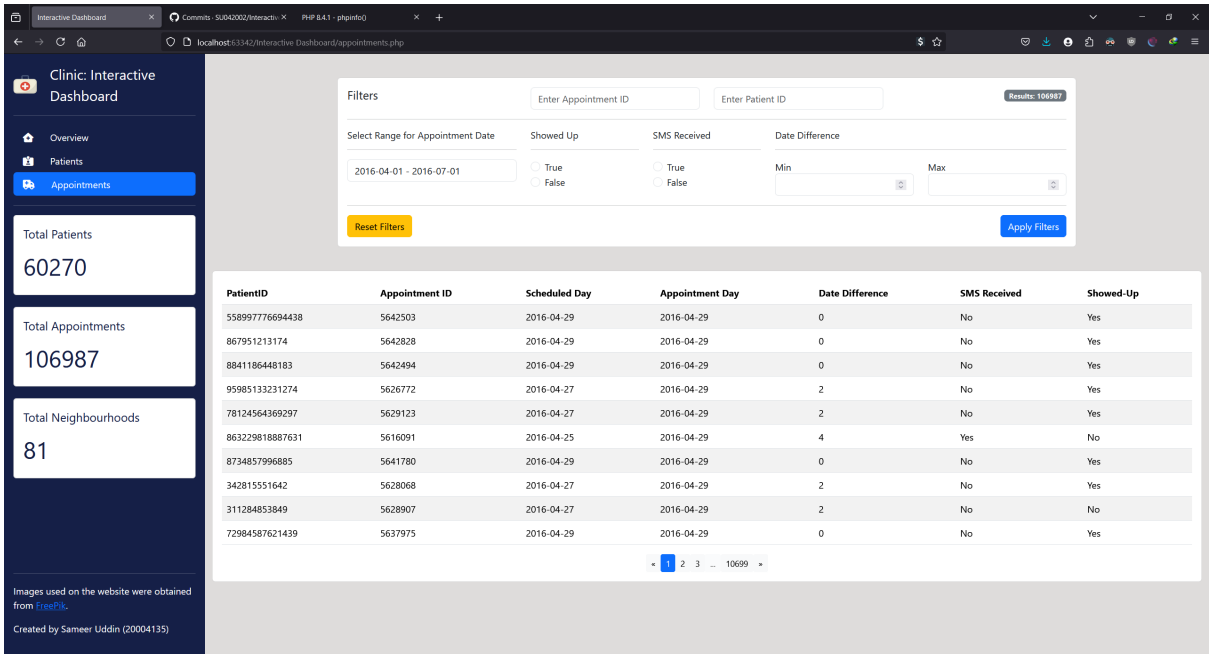


Figure 11 - Appointments Page

Neighbourhoods Page

The original design had another page for neighbourhood but I got rid of this page as it was repetitive. Neighbourhood information can be interpreted from the patient's page. The overview page also contains graphs and some data about neighbourhoods such as how many neighbourhoods there are. The patient's database also has distinct rows of patients which means information such as how many patients are from a neighbourhood can be found through the patients' page.

Testing

Overview Page

Test Case	Test Scenario	Expected Results	Actual Results	Screenshot (hyperlink)
Total patients count displaying correctly on reload of page	Reload the page and see if the total count is correct compared to the original Excel sheet. This will be done by selecting all the rows and checking the count in Excel.	The total count should be 60270.	The total count is 60270.	Figure 12 - Testing Total Patients
Total appointments count displaying correctly on reload of page	Reload the page and see if the total count is correct compared to the original Excel sheet used to build the database. This will be done by selecting all the rows and checking the count in Excel.	The total count should be 106987.	The total count is 106987.	Figure 13 - Testing Total Appointments
Total neighbourhood count displaying correctly on reload of page	Reload the page and see if the total count is correct compared to the original Excel sheet. Duplicate records will be removed by using Excel's remove duplicate function.	There should be 81 distinct neighbourhoods.	There are 81 neighbourhoods. The Excel sheet says 82 unique values because it includes the string "Neighbourhood".	Figure 14 - Testing Neighbourhood Count

Is total patients with Conditions graph generated correctly	Reload the page and see what values the graph produces.	Diabetes = 4416 Hypertension = 12, 242 Alcoholism = 1506 Handicap = 1132	The expected results are displayed in the application.	Figure 15 - Testing Diabetes Count
Is total appointments over months generated correctly	Reload the page and see what values the graph produces. Filter the months using Excel and count all the selected values.	April = 3104 May = 78202 June = 25681	The expected results are displayed correctly in the line graph.	Figure 16 - Testing Appointments over Months

The values being displayed are correct and the graphs generated are accurate when compared with the original data. The values and charts are generated using similar methods so they are also accurate and are displaying results as intended.

Patients Page

Test Case	Test Scenario	Expected Results	Actual Results	Screenshot (hyperlink)
Patients filter for diabetes	Apply diabetes filter to the data and see if first page and last page are users with diabetes. Another confirmation can be obtained from the patients table testing, according to the database there are 4416 patients with diabetes. 4416 results should be obtained from the database.	Only patients with diabetes should be displayed. 4416 results should be obtained after only applying the diabetes filter (all neighbourhoods selected).	First and last page only contain patients with diabetes. 4416 results were also obtained.	Figure 17 - Testing Diabetes Filter

Gender filter for patients	The overview page was confirmed to be working and that page has a chart for gender distribution. There are 21189 males and 39081 females.	I will select females and there should be 39081 results obtained in total.	The search query returned 39081 results.	Figure 18 - Testing Gender Filter
Age filter for patients that are 1	I will set the age between 1 and 1 which should only return results for patients that are 1.	All the results should return users that are 1 and nothing else.	The query returned patients that are only 1.	Figure 19 - Testing Age Filter (Patients Aged 1)
Age filter for patients that are 1 and 2	I will set the age between 1 and 2 which should return more results than the original query above.	All the results should return users are 1 and 2 and nothing else.	The query returned patients that are only 1 and 2.	Figure 20 - Testing Age Filter (Patients Aged 1 & 2)
Age filter for patients that should not exist	I will set the age value to something ridiculous like 200 to see if any results are returned.	There should be no results and the table should be empty.	The query returned no patients.	Figure 21 - Testing Age Filter (Aged 200)
Selecting multiple medical conditions	I will set two conditions so that patients with multiple conditions are only returned.	The table should only be filled with patients with diabetes and hypertension. Both conditions should be true to return results.	Only patients with hypertension and diabetes were returned.	Figure 22 - Testing Multiple Conditions
Selecting multiple filters of different categories	I will set a medical condition and an age filter to find patients aged 1 to 12 with diabetes.	The table should only be filled with users aged 1-12 with diabetes.	11 results were returned and all of them met both requirements.	Figure 23 - Testing Multiple Conditions of Different Categories

Searching for patient using PatientID	I will search for a specific patient using their PatientID.	The table should only have 1 patient because there are no duplicate patient entries in the database.	1 result was returned with the correct PatientID.	Figure 24 - Testing PatientID Search
Searching for patient by neighbourhood	I will search for patients in only 1 neighbourhood. For this test I will only select “Aeroporto”.	The table should return 7 values as that is how many there are for that location in the Excel sheet.	7 results were returned.	Figure 25 - Testing Neighbourhood Search
Pagination	Pressing the button should switch pages.	The table should be updated with new results.	The table changes values when the next button is clicked.	
Reset Filters Button	Set all the values to something and then press reset to see if it is updated.	All the values should be reset to their default values such as empty string or something else like 1 for age.	All the values were reset correctly.	Figure 26 - Testing Reset Filter
Apply Filters Button	Select filters and press apply.	The table should be updated according to the filters applied.	The filter button works correctly.	
Real time updates	Selecting filters and then press apply and then repeat the process with different filters to see if the table is updated. I will select male first and then filter. Then select females.	The results should change once the 1st filter applied. Then the table should update when the 2nd filter is applied.	The table updates properly after every update to the filter.	

Invalid Input	Type in none existent patient filter and see if anything is returned. A string of characters will be sent instead of int.	Nothing should be returned. An error prompt could be provided for invalid input.	No data is returned but there is no prompt for an invalid input that is checked before querying the database	Figure 27 - Testing String Search

Appointments Page

Test Case	Test Scenario	Expected Results	Actual Results	Screenshot (hyperlink)
Search for appointments with default appointment day date filter	The default values are set to encompass all the values in the database.	The total amount of appointments returned should be 106987.	106987 results were returned.	Figure 28 - Testing Appointments Date
Search for appointments for duration of 1 month	For the month of April there were 3104 appointments according to the graph in the overview page. The date range will be applied in the appointments page for April.	The database should return 3104 rows of data.	The table has 3104 rows of data.	Figure 29 - Testing Appointments for April

The logic and methods for this page are very similar to the patient's page. That page was tested and works which also means that this page functions as intended. The only problem is that there is no data validation on the frontend before the database is queried.

Testing Screenshots

Overview Page

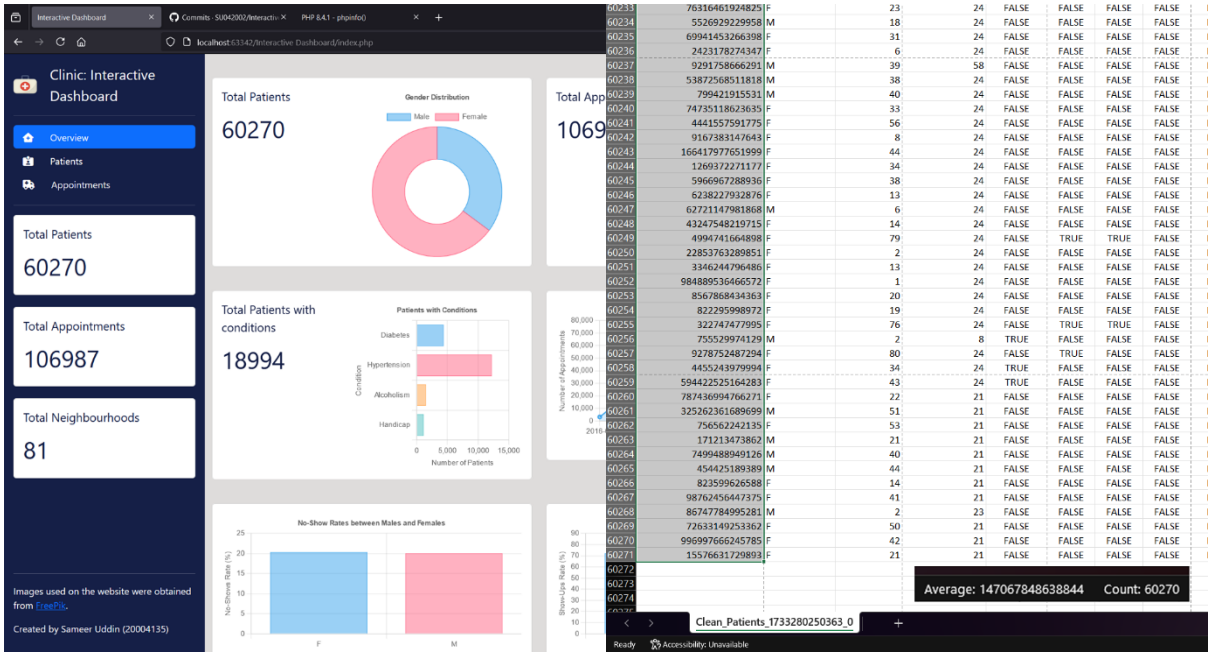


Figure 12 - Testing Total Patients

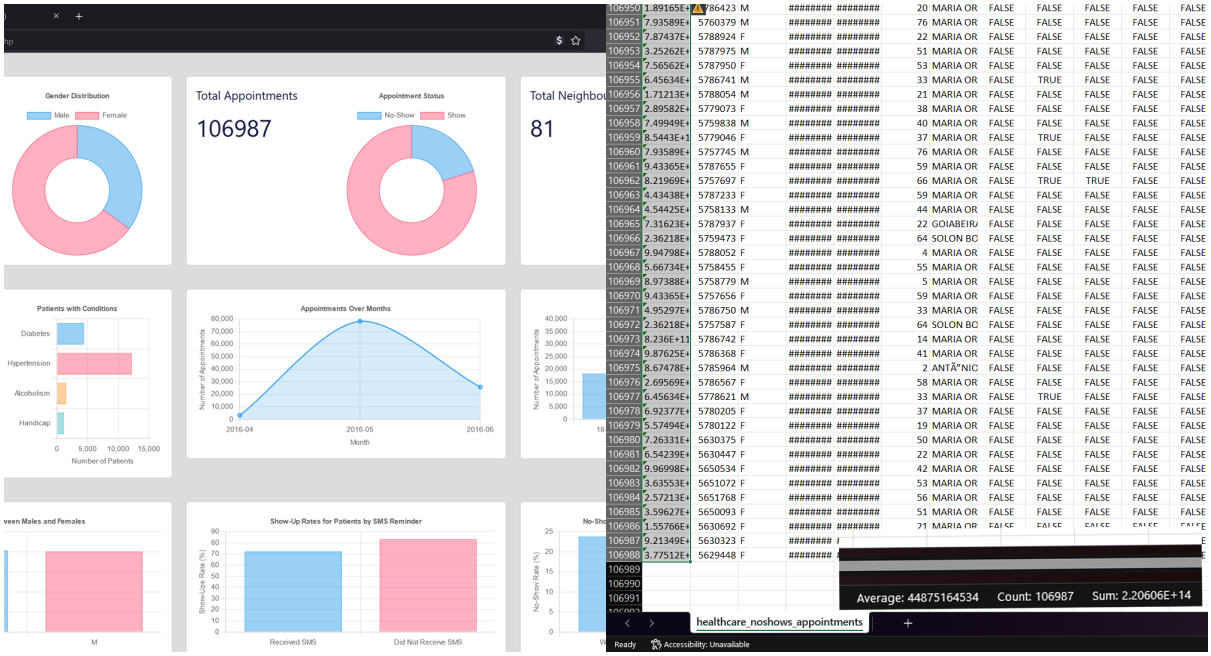


Figure 13 - Testing Total Appointments

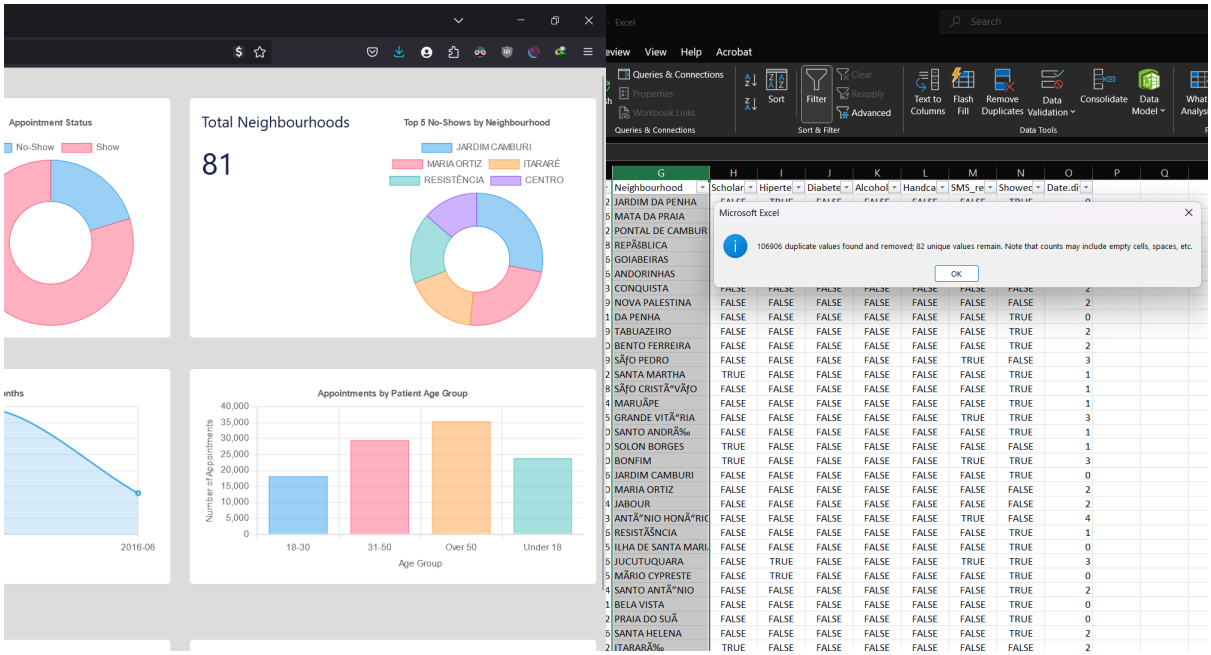


Figure 14 - Testing Neighbourhood Count

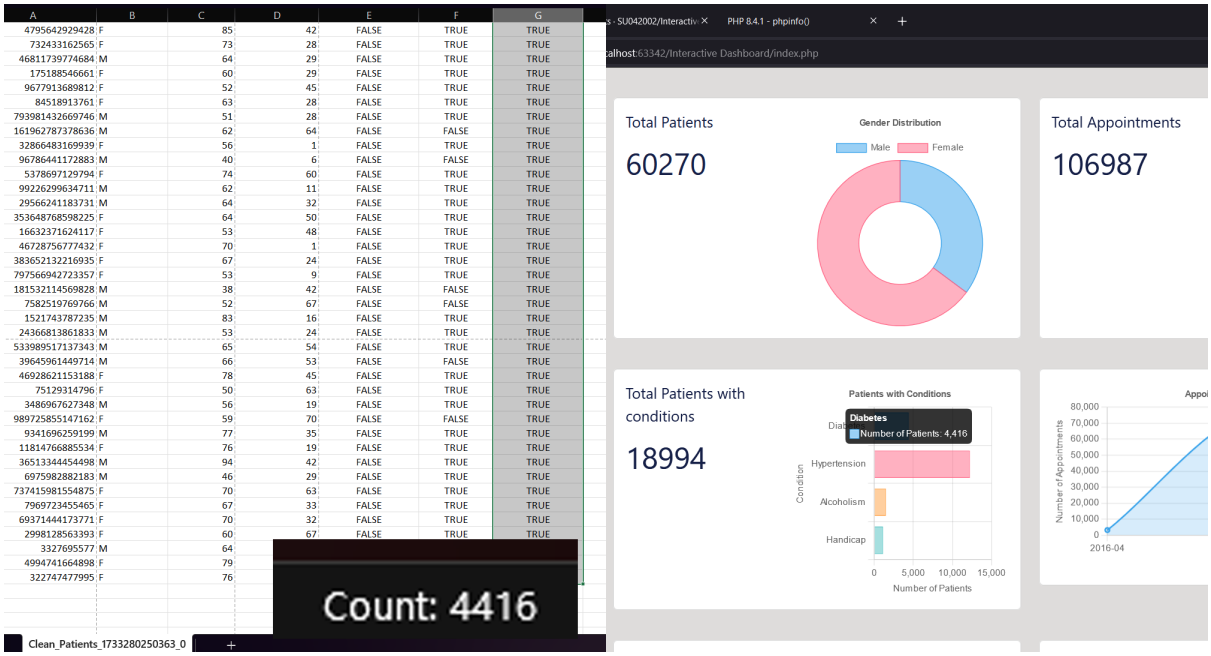


Figure 15 - Testing Diabetes Count

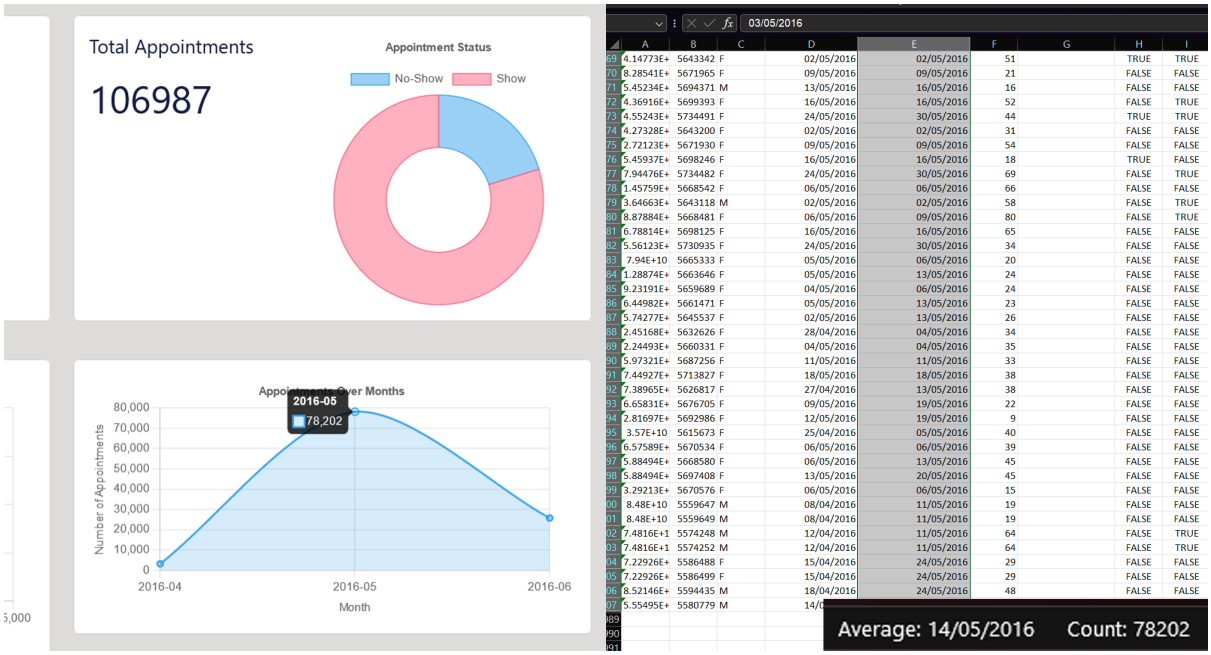


Figure 16 - Testing Appointments over Months

Patients Page

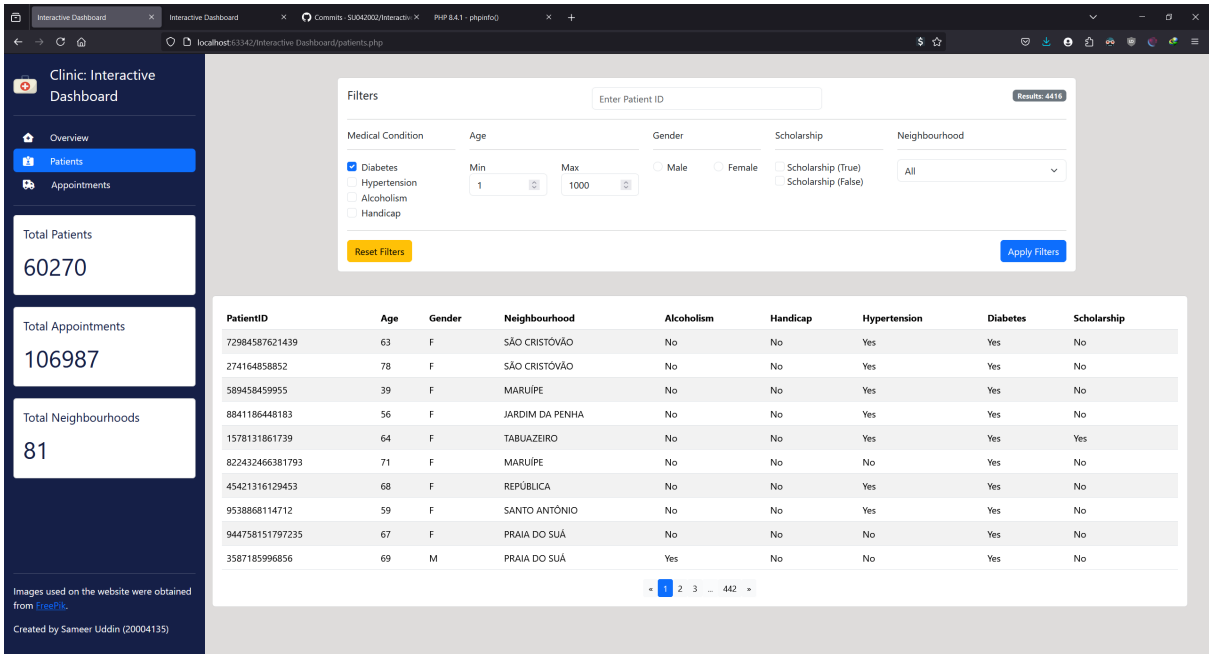


Figure 17 - Testing Diabetes Filter

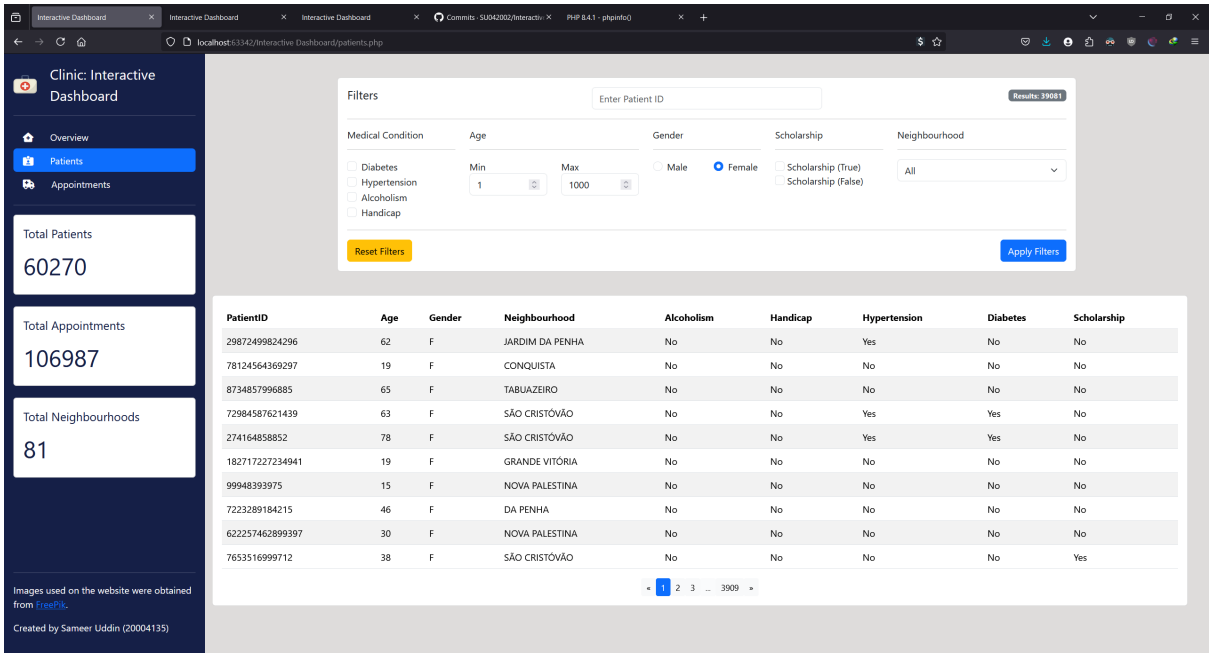


Figure 18 - Testing Gender Filter

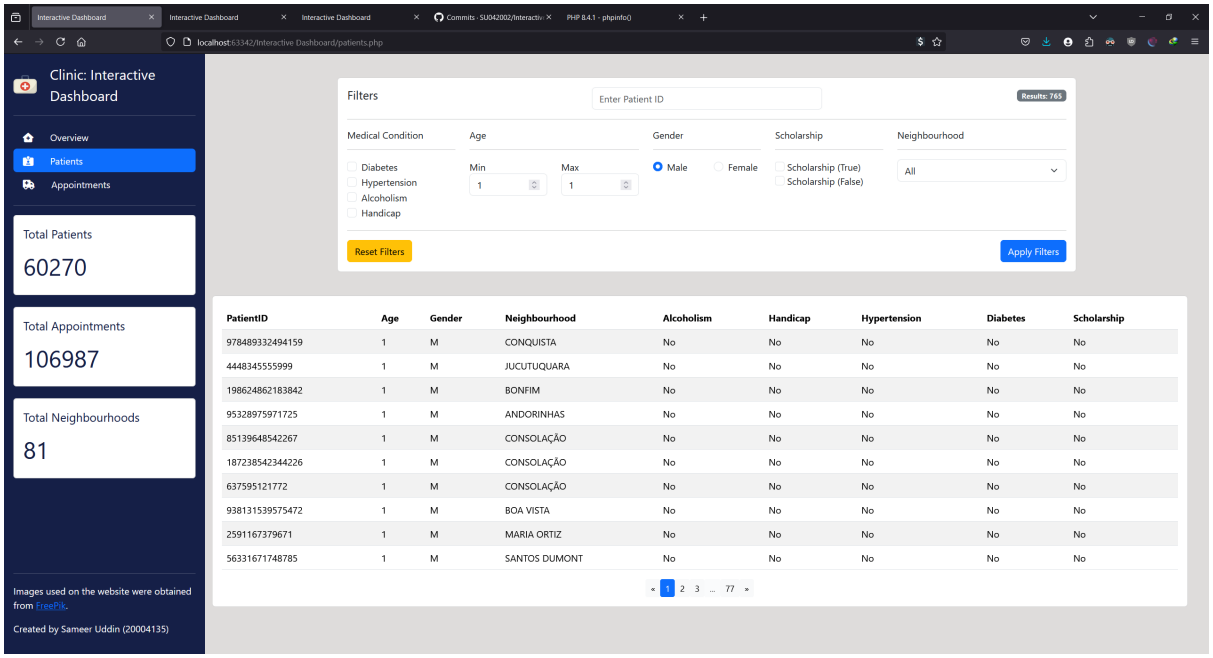


Figure 19 - Testing Age Filter (Patients Aged 1)

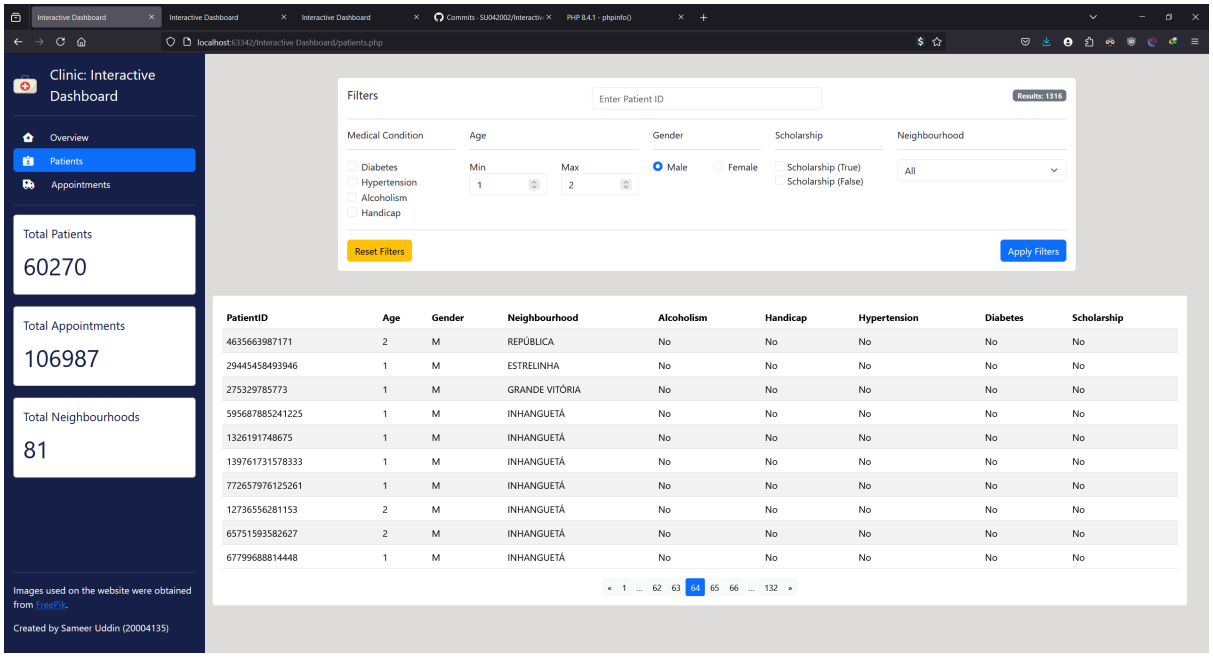


Figure 20 - Testing Age Filter (Patients Aged 1 & 2)

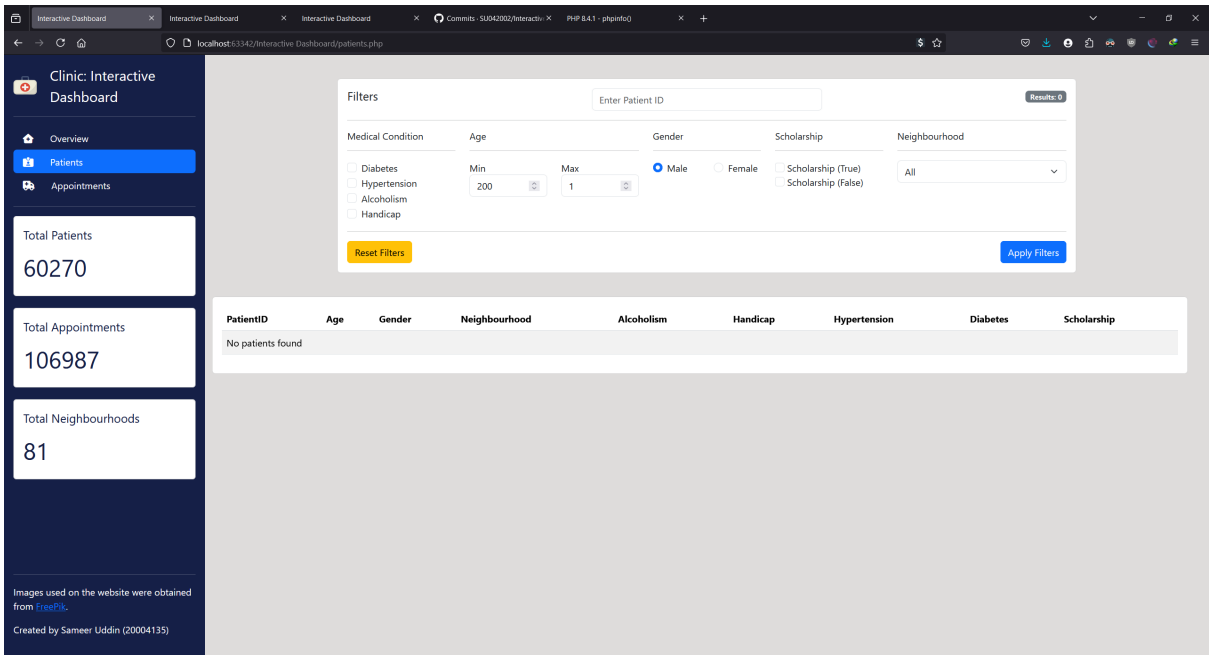


Figure 21 - Testing Age Filter (Aged 200)

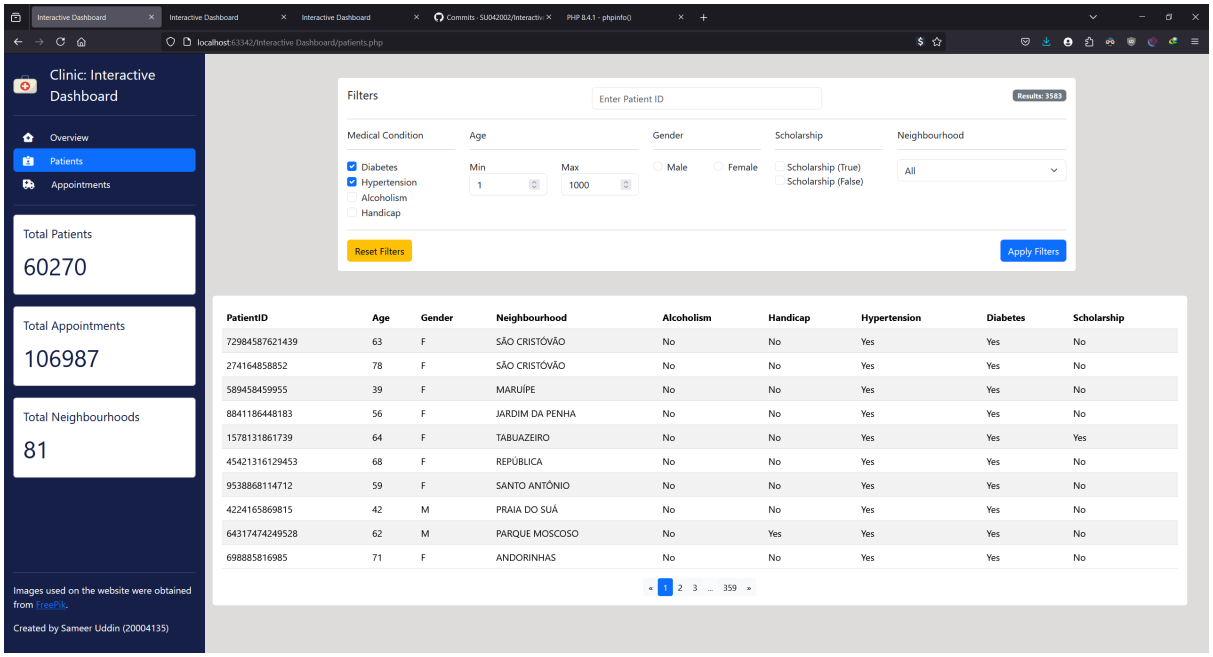


Figure 22 - Testing Multiple Conditions

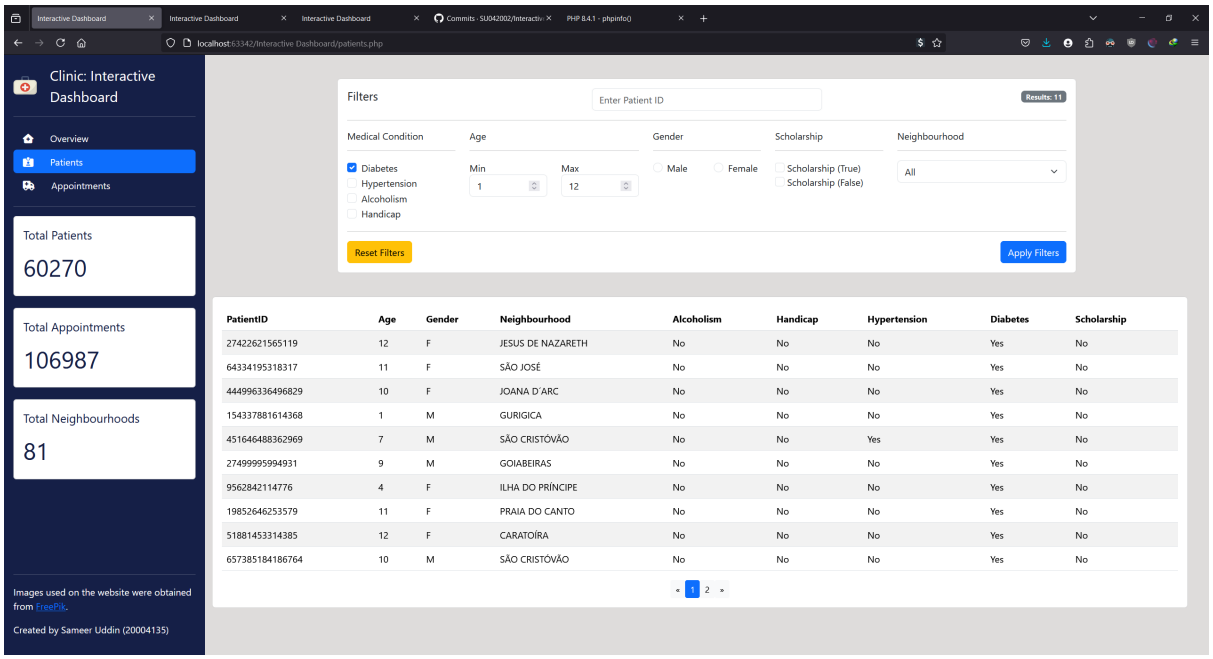


Figure 23 - Testing Multiple Conditions of Different Categories

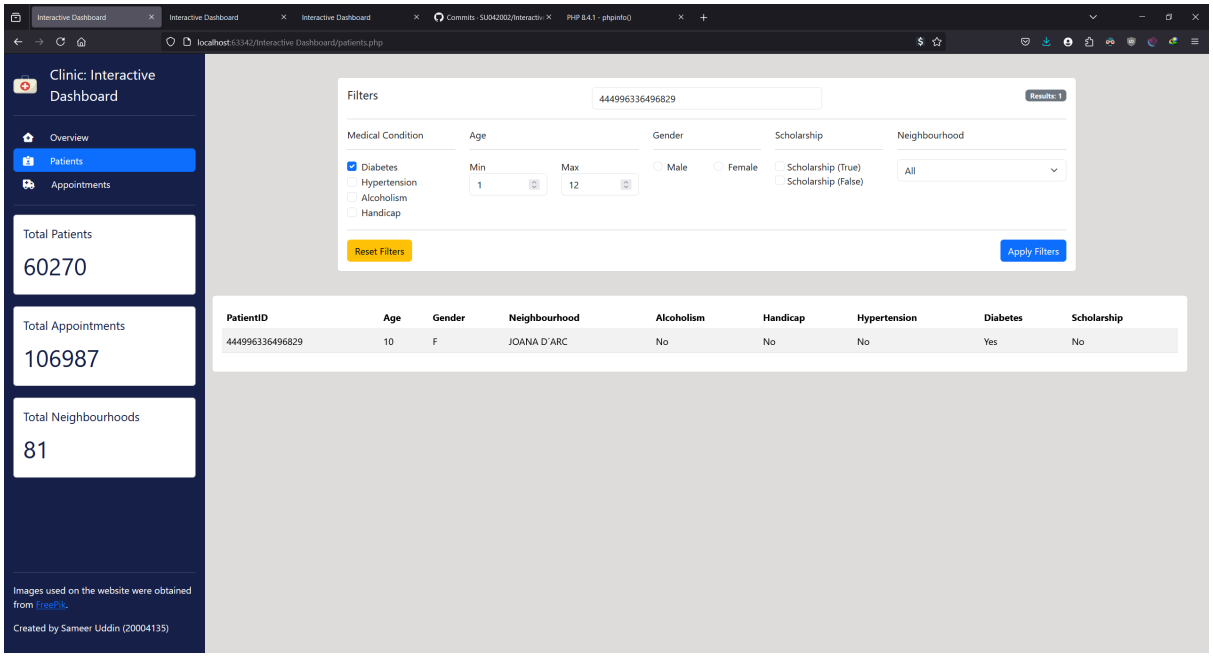


Figure 24 - Testing PatientID Search

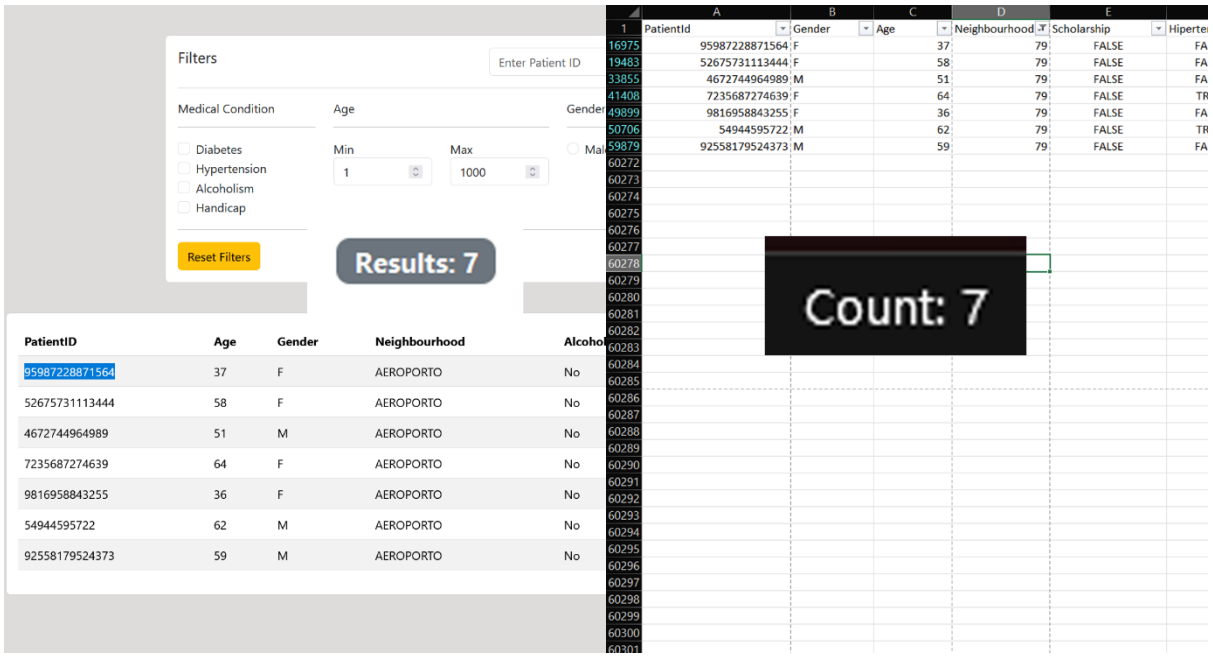
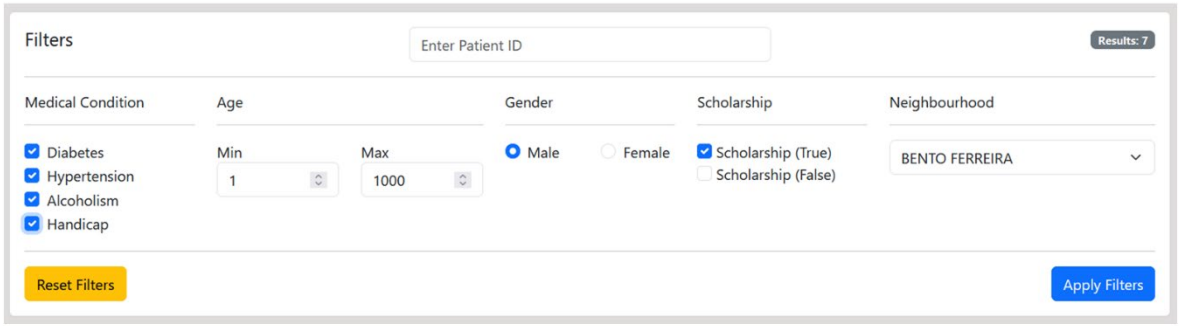


Figure 25 - Testing Neighbourhood Search

BEFORE



After

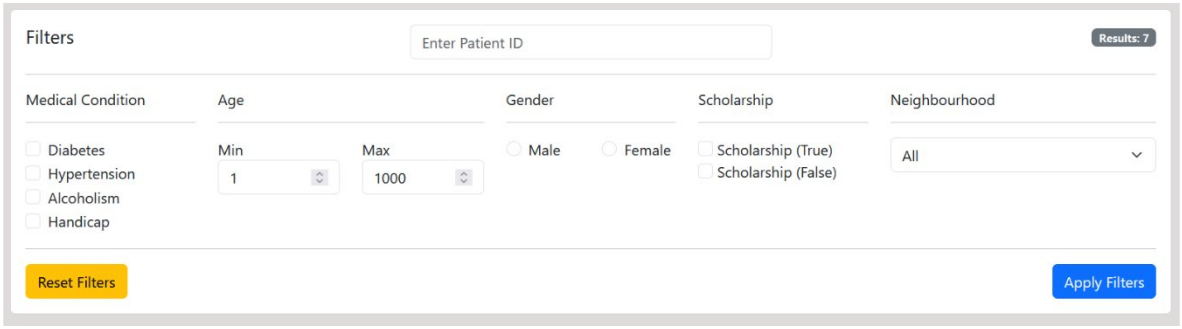


Figure 26 - Testing Reset Filter

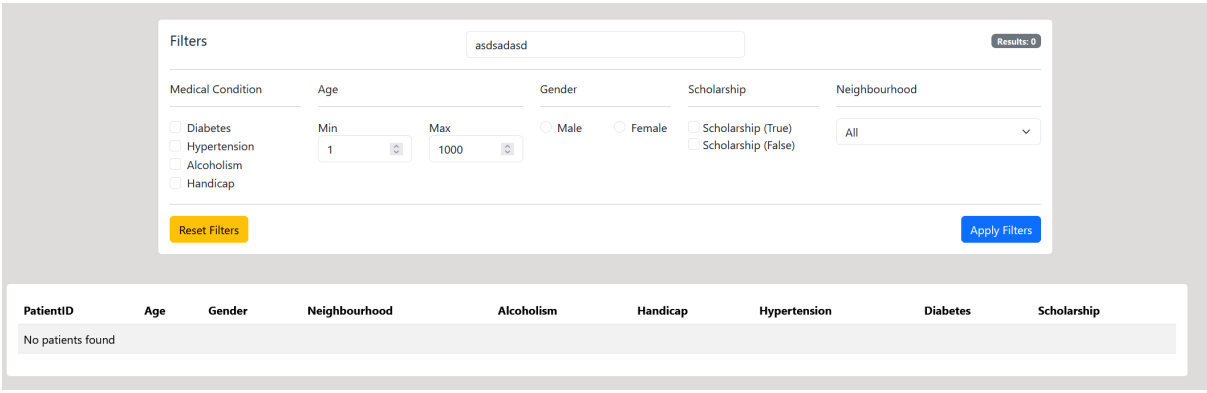


Figure 27 - Testing String Search

Appointments Page

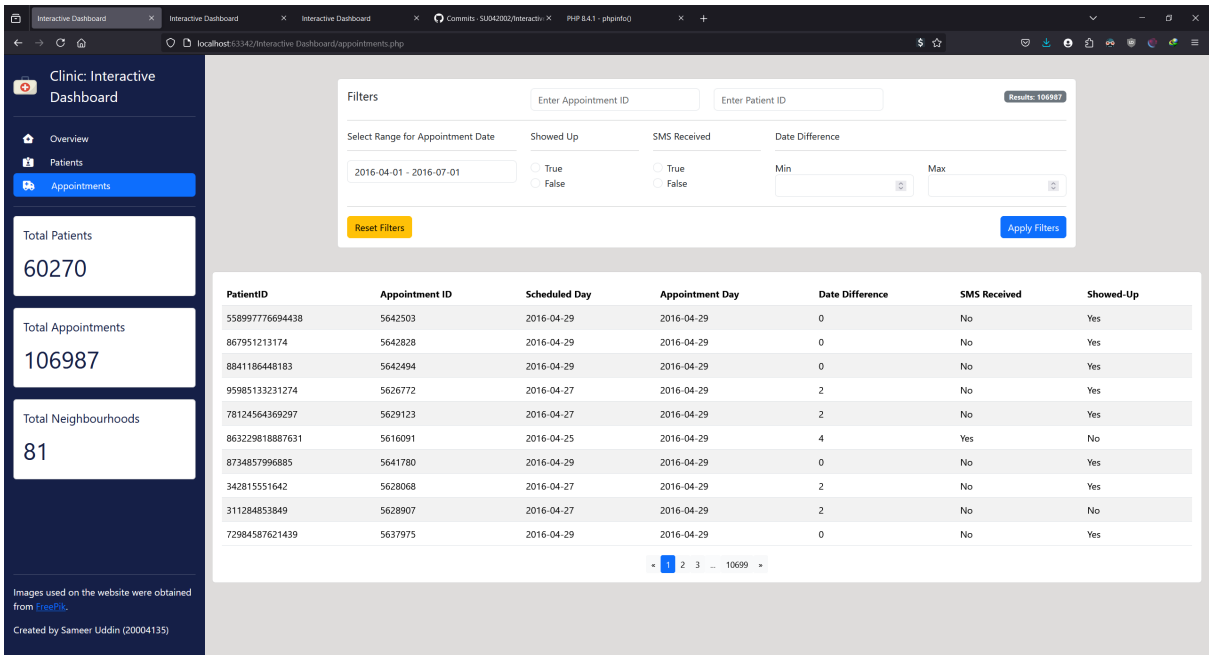


Figure 28 - Testing Appointments Date

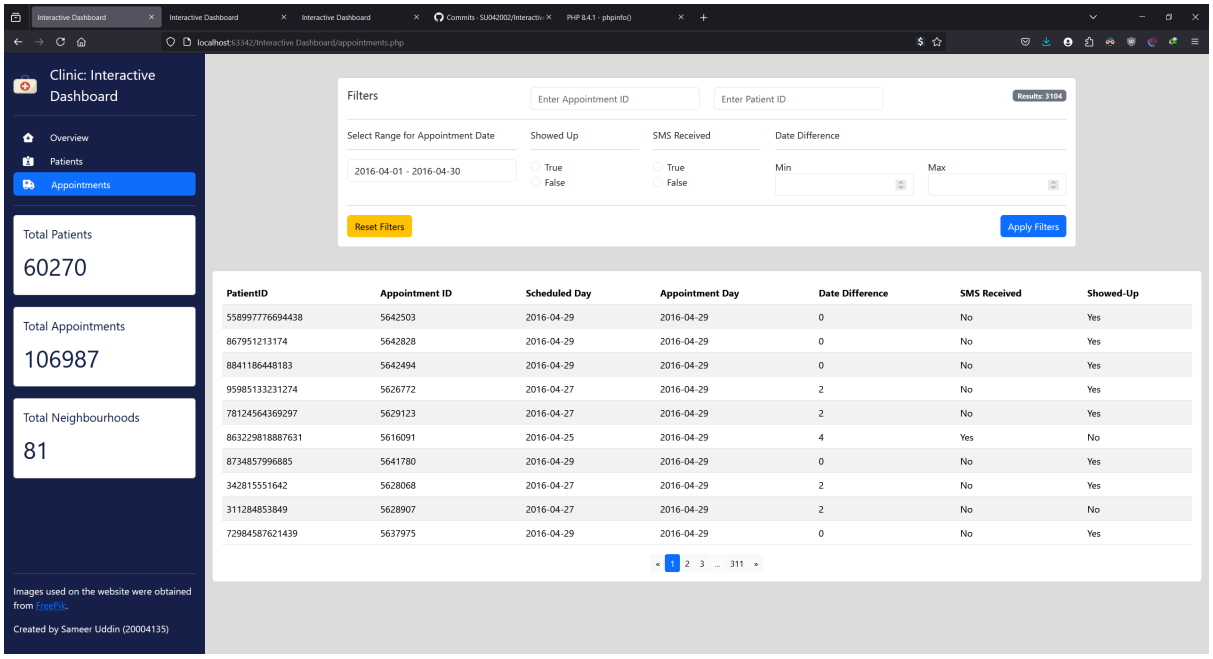


Figure 29 - Testing Appointments for April