**Name:** Aldaine Clarke

**Course:** Front End Development (AMBER)

**Project Title:** Pediatric Hospital of Jamaica.

**Due Date:** September 9, 2022

**Table of Contents**

[Product Overview 3](#_Toc113650925)

[Product Objective 4](#_Toc113650926)

[Roadmap 5](#_Toc113650927)

[Features to Implement: 5](#_Toc113650928)

[Requirements 7](#_Toc113650929)

[Functional Requirements 7](#_Toc113650930)

[Non-functional Requirements 7](#_Toc113650931)

[User Manual 8](#_Toc113650932)

[Getting Started 8](#_Toc113650933)

[Managing Account Information. 10](#_Toc113650934)

[Creating an Appointment with account 11](#_Toc113650935)

[Find and view doctor’s information 12](#_Toc113650936)

[System Administrator Manual 15](#_Toc113650937)

[Getting Started 15](#_Toc113650938)

[Managing Appointments 16](#_Toc113650939)

[API Documentation 17](#_Toc113650940)

[API Endpoints 17](#_Toc113650941)

[users/ 17](#_Toc113650942)

[appointments/ 27](#_Toc113650943)

[doctors/ 30](#_Toc113650944)

[patients/ 34](#_Toc113650945)

[records/ 37](#_Toc113650946)

[Data Models 41](#_Toc113650947)

[References 44](#_Toc113650948)

# Product Overview

The requirement was to create an application that allows a pediatric medical care centre to properly manage the patients and staff that they have available.

The **Minimum Viable Product (MVP)** of the system is that, it should allow users to create an appointment, and to see the doctors available. The hospital staff should be able to manage all the appointments that are made, and should also be able to create an appointment, in the case where the user decides to call in to set appointment. To ensure authorization on who can manage the system, there will be a login system for the admin section of the application.

The vision of this project is to create a Management System to assist medical care facilities with features that will expedite the processes of its regular day to day operation. Some of these features are the management of Doctors, Inventory, Patients, Medical Records.

# Product Objective

The objective of the application as it stands is to provide as an expeditor for the medical care processes. The objectives of the product are:

* To reduce the time it takes for user to make an appointment: - It takes a long amount of time to make an appointment at a health care facility, and even though you can make an appointment over the phone, it takes some time to get through to the medical care professionals.
* To allow users to compare medical professionals to ensure they get the best care possible.

# Roadmap

The major limitation for this ambitious project, is the time frame that was required to complete the project. While the base requirement for this project may be simple, as it should be for a project of such a small time frame, I couldn’t neglect the possibility of the project scaling up.

Additionally, I have considered the possibility of this being a necessary application for a medical facility, as I have been in a situation where I have been inconvenienced because of the medical care facility’s processes. It is because of these reasons that I have decided to take this route and do this ambitious project.

I plan to continue working on the project even after submission to have a working prototype as soon as possible.

## Features to Implement:

* Allow the user to make a request for medical record from inside their account.
* Alter the doctor’s status whenever the doctor has an appointment or is seeing a patient.
* Validate all form inputs to ensure that the data being sent from the forms are the correct data.

# Requirements

### Functional Requirements

* The system should send appointment information when form is submitted from the home page.
* When user clicks sign up, login or update to submit form data, the system should approve request if the data is valid. If data is not valid then a message should be displayed to the user about the reason for failure.
* When a specialty is selected, the system should filter the doctors available to only show the doctors that are in that department.
* When user visits doctor profile page, once they select the option to set appointment, they should be redirected to the appointment creation page, and the doctor’s information should already populate the form.
* Should not allow unauthenticated users to visit protected pages
* Should not allow users to view pages that are protected by admin.
* Admins should not be able to sign up but only created

### Non-functional Requirements

* Pages should not leave users with no way to navigate to another page.
* All functionality should show proper messages to the user, especially if an error occurs. It should fail gracefully.
* The website should be up 24/7. The only downtime of the application should be if there is a schedule or non-schedule maintenance.

# User Manual

### Getting Started

When you visit the PHJ application page, you will be first greeted with the home page. This page gives you a summary of the services offered by the hospital and application. It contains a news section that has live news data and also an area where any user that visits the page can make an appointment to see any of the doctors available in the system.

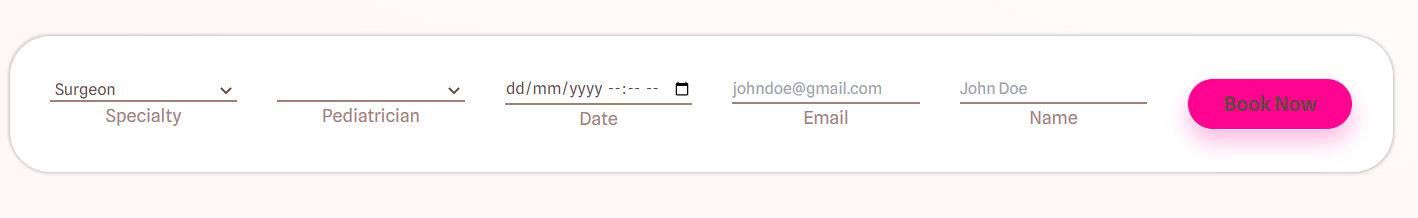


Fig. G-1: shows appointment booking form on home-page

If you want more control over your data and possible do more than just set appointments, you can opt to sign up with the application.

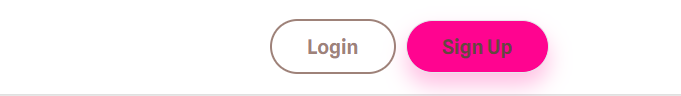


Fig. G-1.2: shows signup and login option on home-page

When you click on sign up this will bring you to the sign up page. This page is where we collect the basic information about you the user.

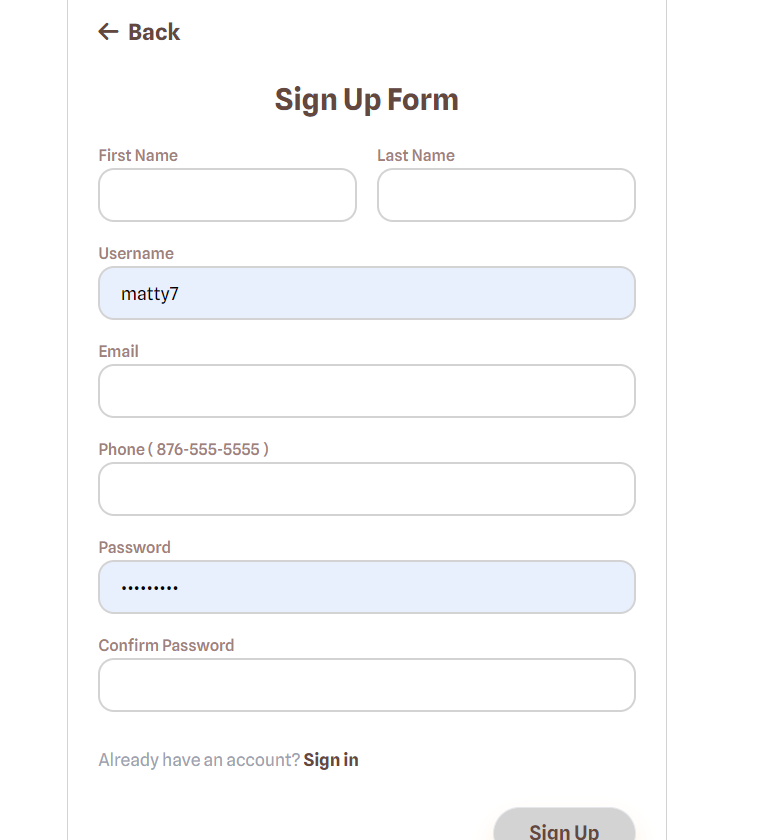


Fig. G-1.3: shows signup page.

Even from the signup page you are able to navigate to the login page if you already have an account, and the same can be said and done from the login page.

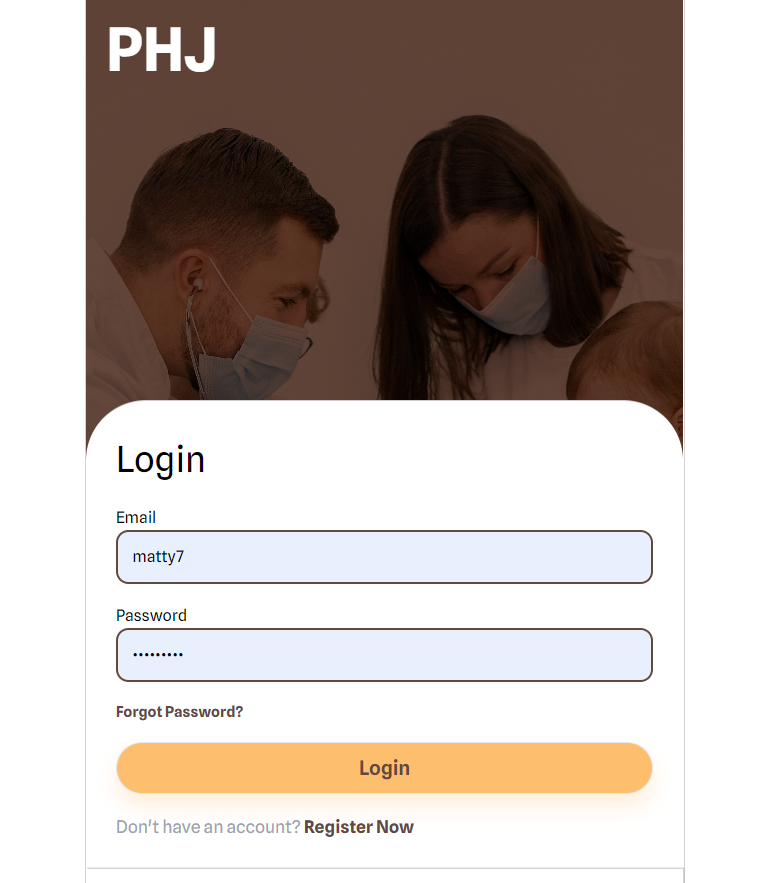


Fig. G-1.4: shows login page design

### Managing Account Information.

PHJ makes it easy to sign up and manage your data. To sign up for an account we only require information that you would need to contact you. With this, there is no need to worry about any additional costs when using our application. You may also choose not to provide some information if you feel it to be too invasive.

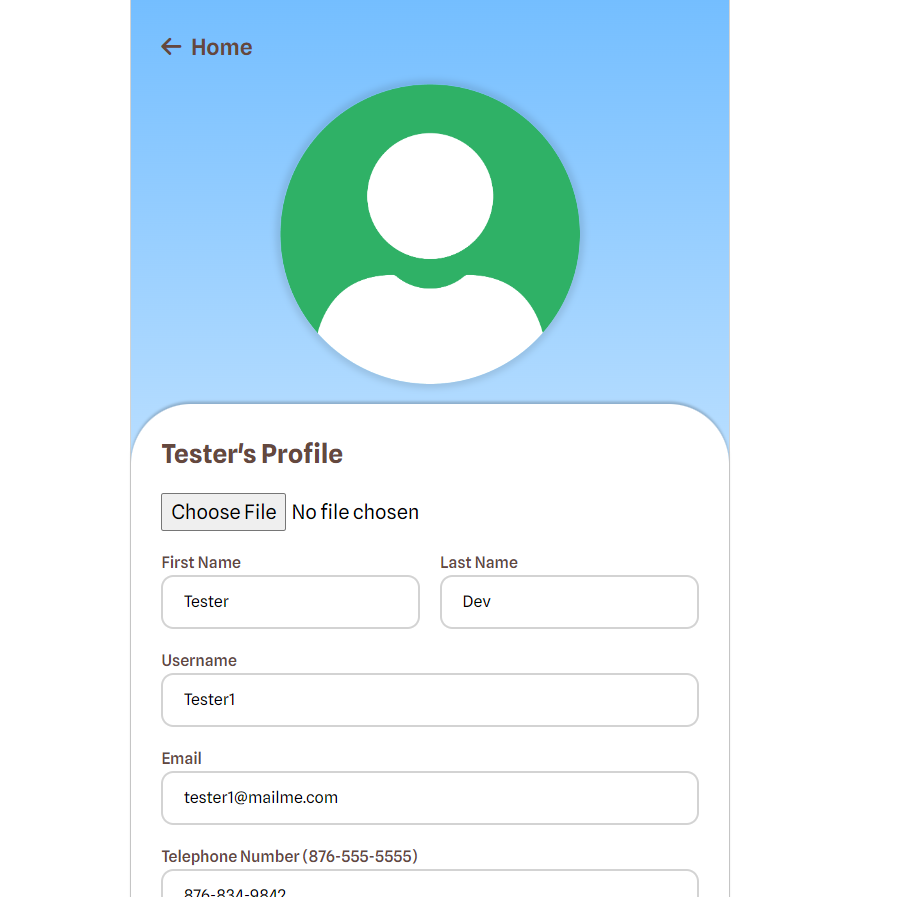


Fig. G-1.5: shows the profile page which allows you to update user info.

Additionally, you are able to add information to create a profile for yourself. This is useful when creating an appointment, if you do not wish to type out the same information every time you wish to create an appointment.

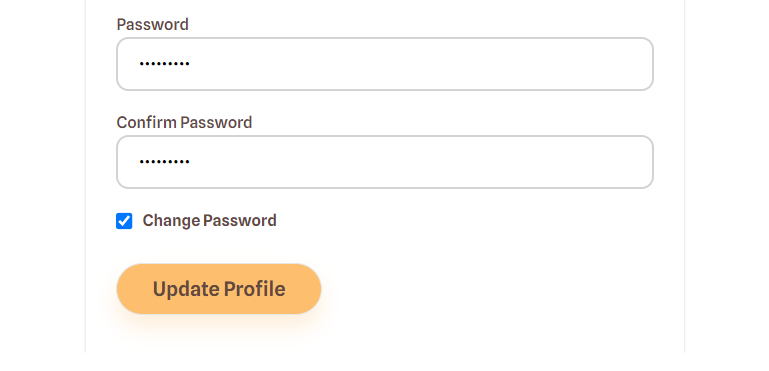


Fig. G-1.6: shows the change password option on the profile page.

By toggling the change password option in the profile page, you are able to change your account password.

### Creating an Appointment with account

Having an account makes it super easy to create appointments. Unlike the appointment creation form, the form available to the registered users contains more details to ensure that you, the user, are able to properly manage all user appointments.

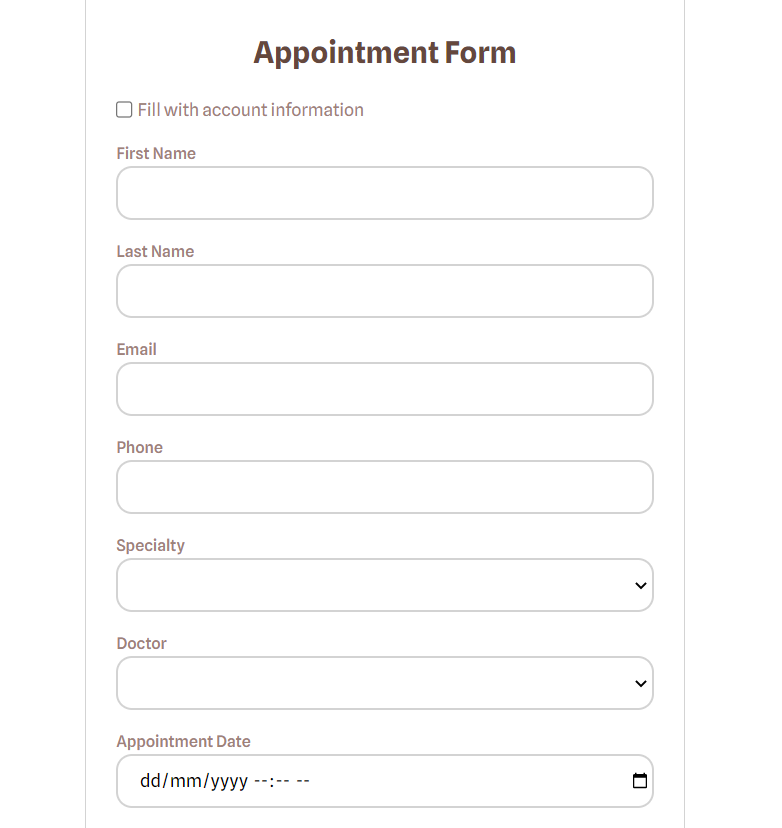


Fig. G-1.7: shows the appointment form available to registered users.

There is also an auto fill feature that is available. This helps to easily sign up the form, since that data that you will type is almost also the same data we have in our database for you.

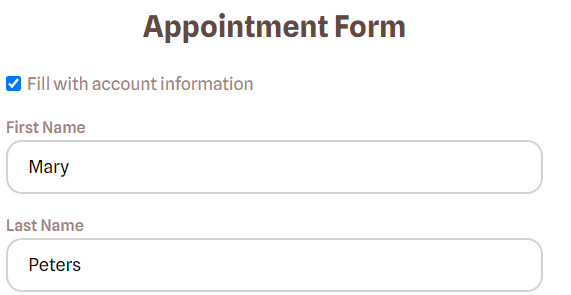


Fig. G-1.8: shows checkbox to determine if you wish to auto fill appointment form with account info.

### Find and view doctor’s information

The application also provides the feature which allows any you to check find doctors that are registered within the system. You are also able to view the doctors’ information as they are documented in the system. This helps with making more informed decisions as to which doctor you wish to make an appointment with.

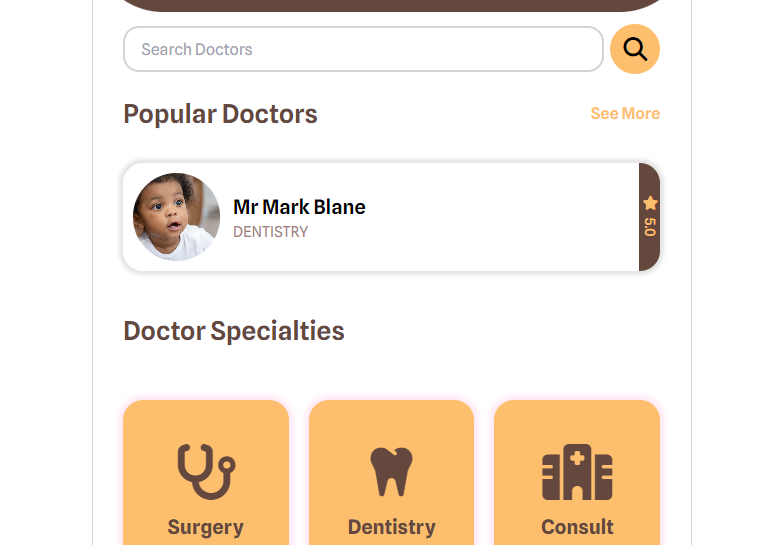


Fig. G-1.9: shows doctors page which hosts doctors and also their specialties/ department.

By selecting the doctor from the doctors’ page, you are able to view the doctors profile information. There is also a button which allows the user to simply set an appointment with the doctor from the doctor’s detail page.

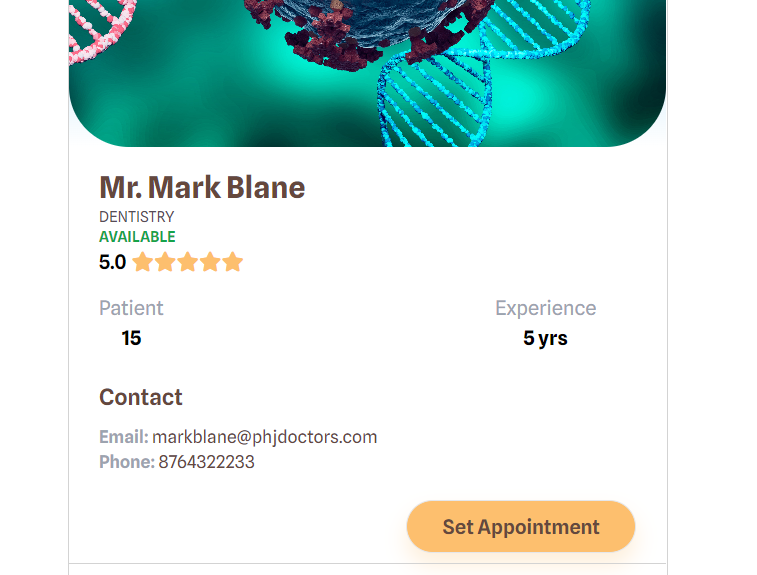


Fig. G-1.9: Doctors’ detail or profile page.

# System Administrator Manual

### Getting Started

Administrators of the system have much more control over who can use the application. Their interface is different from the user, such that there is no option to sign up from the admin page, if you do not already have an account.

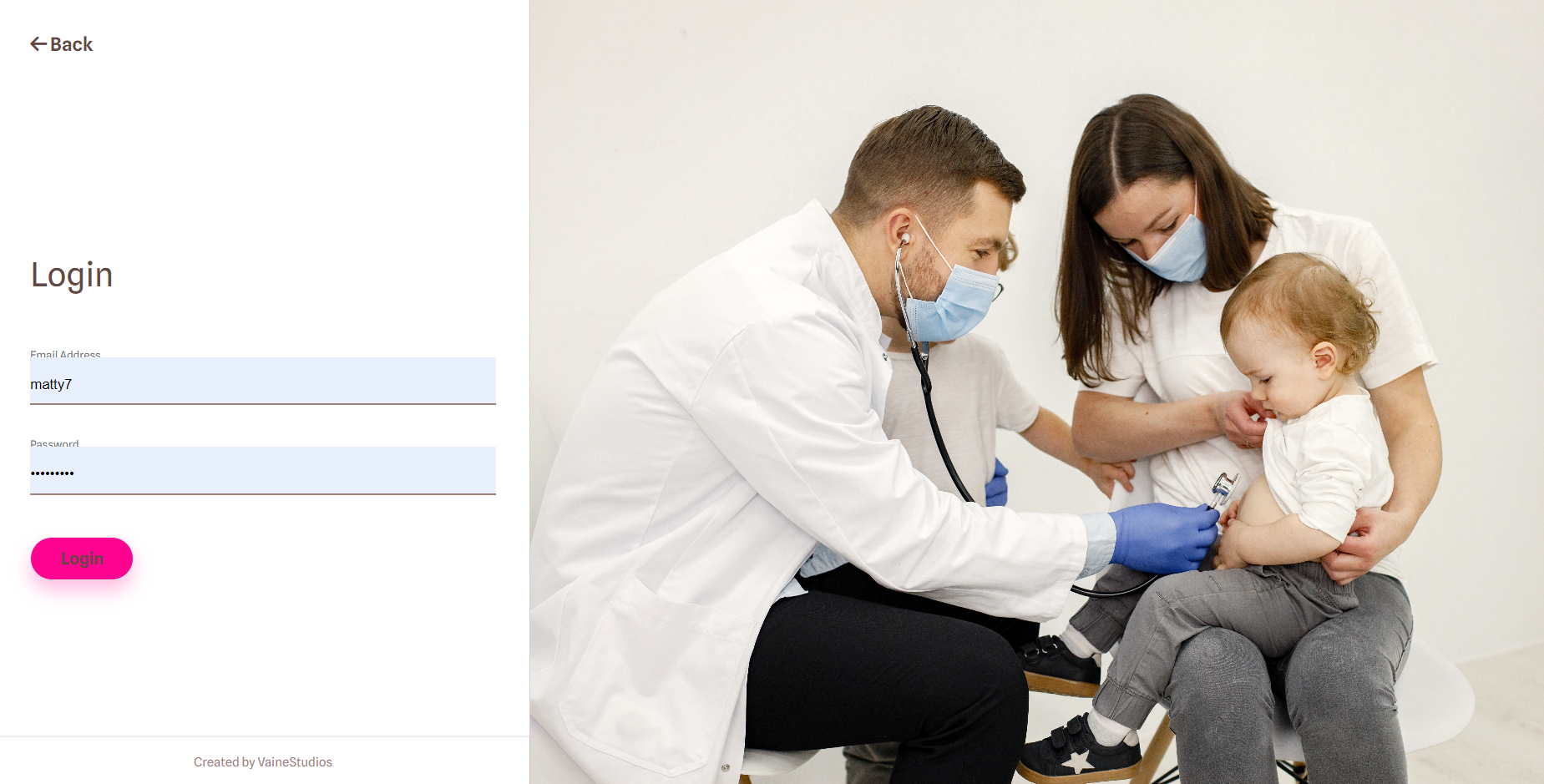


Fig A-1.1: Shows the admin page of the web application.

Any doctor, at this moment, would be able to log into the system. For this stage of the application, a doctor is already created which would allow you to log in to begin using the system:

Email: [admin@phjdoctors.com](mailto:admin@phjdoctors.com)

Password: admin123

### Managing Appointments

Once the doctor logs into the system with the information, then he will be greeted by the dashboard. It is with on this page that you will be able to manage appointment and other features of the project.

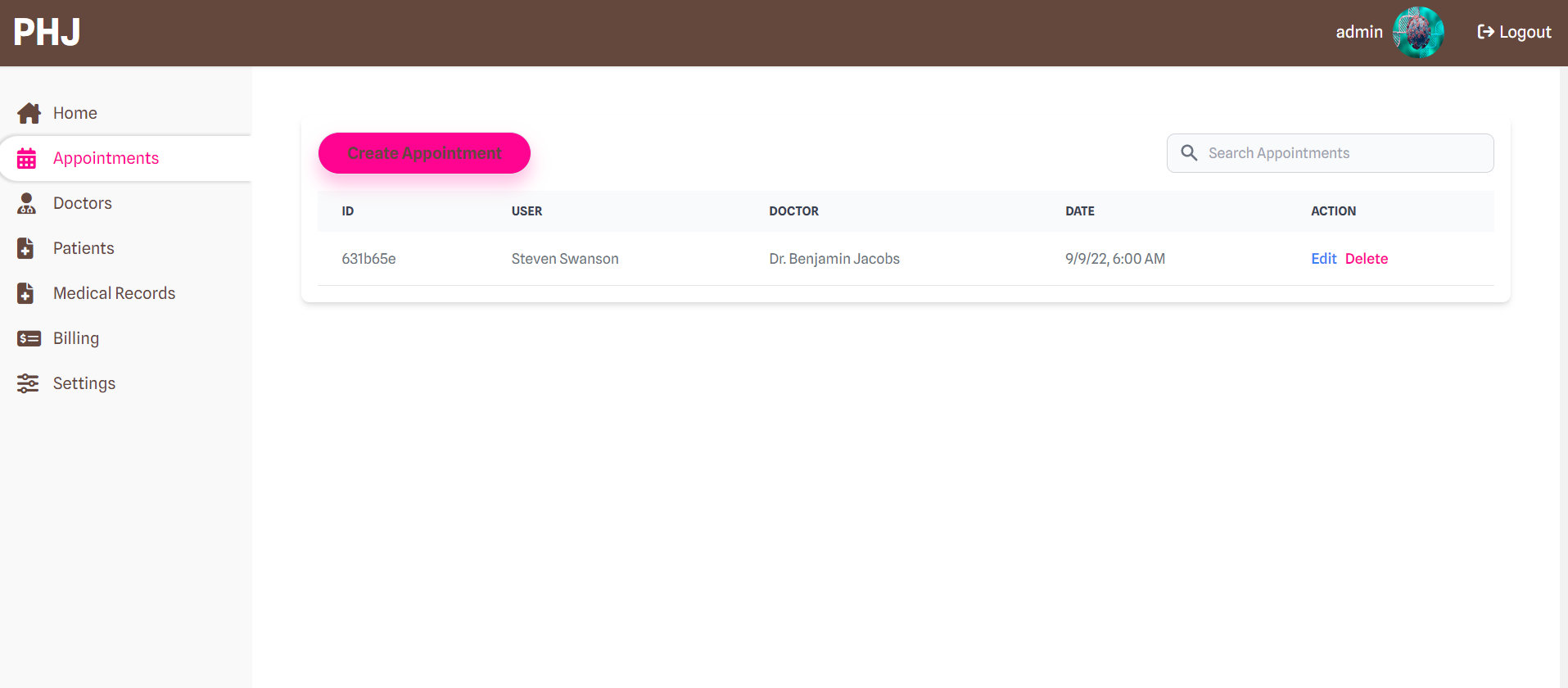


Fig A-1.2: Show the list of appointments already made in the system

Doctors may also be able to create appointment on the behalf of the customers when they call in. They would just need to provide basic contact information to ensure the appointment is set successfully.

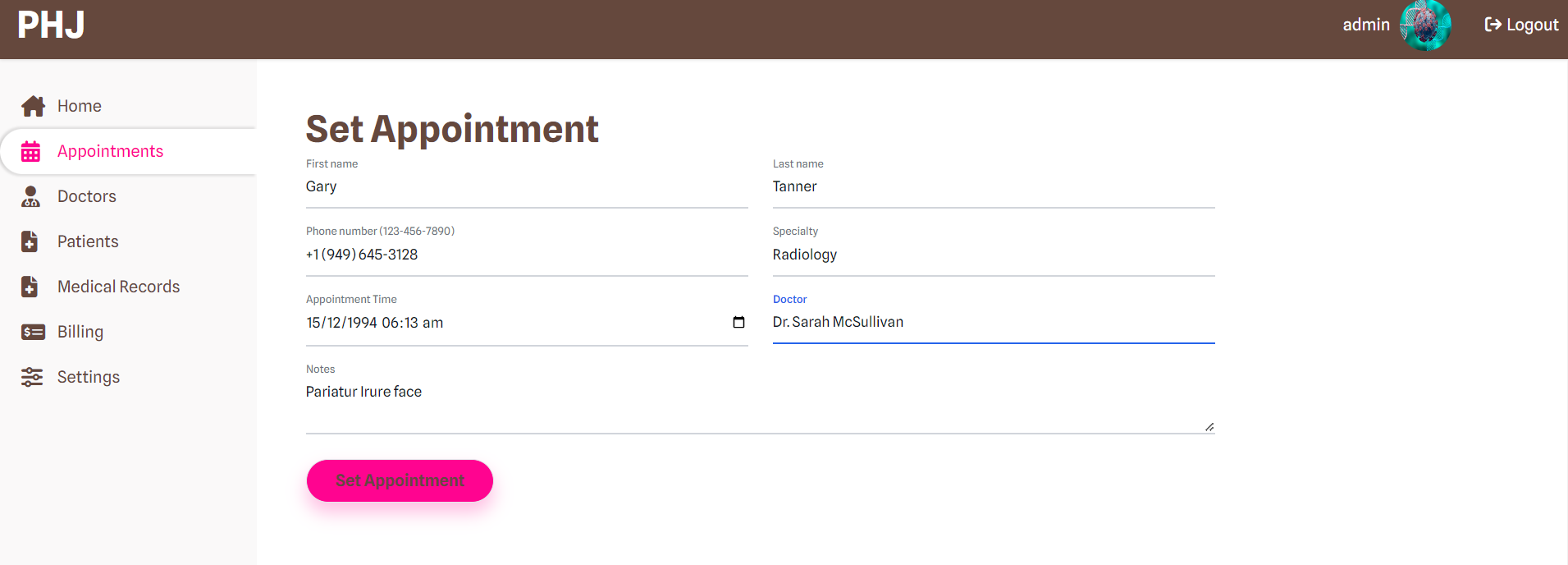


Fig A-1.3 Show how to set appointments on the doctor’s end.

# API Documentation

The PHJ API is organized around the **REST.** Our API has predictable URLs, and even accepts form-encoded request bodies. Each resource returns a JSON-encoded response and also standard HTTP response codes, authentication and verbs.

**BASE URL =** [**https://phj-app.herokuapp.com/api/vi/**](https://phj-app.herokuapp.com/api/vi/)

Once you visit the BASE URL, you will receive a JSON response with the list of routes and resources that the API manages.

## API Endpoints

#### users/

Manages users that are registered to the web application.

POST

users/

A POST request made to the users’/ resource route will create a user with the data that is passed in the body of the request.

**Parameters**

*body: Partial<user>;*

**Response**

The response returns a JSON with content type *application/json* along with a status code. The user is returned wrapped in a JSON response if created. If no data is passed, then the API will respond with a Bad Request (HTTP 400).

**Response Example**

{

    "status": "Success",

    "message": "Successfully Created User",

    "data": {

        "fname": "Carlton",

        "lname": "Banks",

        "email": "carltonb@mailme.com",

        "phone": "876-789-8433",

        "username": "carlton",

        "\_id": "631ad4af0517320ed557bc95",

        "createdAt": "2022-09-09T05:52:47.561Z",

        "updatedAt": "2022-09-09T05:52:47.561Z",

        "\_\_v": 0

    }

}

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| Status | String | Describe the status of the response sent. Can be ‘Success’ or “Failed”. |
| message | String | Short description of operation that was perform or error message if there is an error |
| data | Object | An object which contains the user that was created |
| fname | String | The first name of the user |
| lname | String | The last name of the user |
| email | String | The user’s email address |
| phone | Number | The user’s telephone number |
| username | String | The user’s username |
| address | Object | An Object containing the address information. |
| street | String | A part of address object and has user’s street address |
| city | String | A part of address object and has user’s city address |
| parish | String | A part of address object and has user’s parish address |

GET

users/

A GET requests made to the users’/ resource route will retrieve all users from the database. There are no parameters for this GET request.

**Response**

The response of the request would be an array of the users in the system. If there are no users presently in the system, then the API returns an empty array.

**Response Example**

{

    "status": "Success",

    "message": "Successfully retrieved",

    "data": [

        {

            "\_id": "631a0a99164ad08a66832fc8",

            "fname": "Mary",

            "lname": "Peters",

            "email": "marypeters@mailme.com",

            "phone": "874-849-5323",

            "username": "mary",

"address": {

                "street": " ",

                "city": " ",

                "parish": " "

            },

            "\_\_v": 0

        }

    ]

}

**Response Data**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| Status | String | Describe the status of the response sent. Can be ‘Success’ or “Failed”. |
| message | String | Short description of operation that was perform or error message if there is an error |
| data | Array | An array which contains all the users currently in the database. |
| fname | String | The first name of the user |
| lname | String | The last name of the user |
| email | String | The user’s email address |
| phone | Number | The user’s telephone number |
| username | String | The user’s username |
| address | Object | An Object containing the address information. |
| street | String | A part of address object and has user’s street address |
| city | String | A part of address object and has user’s city address |
| parish | String | A part of address object and has user’s parish address |

users/{id}

GET

A GET requests made to this route returns a single element or user from the database.

**Parameters**

The parameter for this request needs to be the id which represents the user that should be found in the database.

*id: string;*

**Response**

The response of the request would be a JSON response with the data being the user that was found that matches the id that was passed as a parameter to the route.

If there was an error, the API will return a message telling the user the error that occurred.

**Response Example**

{

    "status": "Success",

    "message": "Successfully retrieved",

    "data": {

        "address": {

            "street": " ",

            "city": " ",

            "parish": " "

        },

        "\_id": "6318e0010aa7540cdd124c3f",

        "fname": "Tester",

        "lname": "Dev",

        "email": "tester1@mailme.com",

        "phone": "876-834-9842",

        "username": "Tester1",

        "\_\_v": 0

    }

}

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| Status | String | Describe the status of the response sent. Can be ‘Success’ or “Failed”. |
| message | String | Short description of operation that was perform or error message if there is an error |
| data | object | An object containing the user that matched the id passed in URL |
| fname | String | The first name of the user |
| lname | String | The last name of the user |
| email | String | The user’s email address |
| phone | Number | The user’s telephone number |
| username | String | The user’s username |
| address | Object | An Object containing the address information. |
| street | String | A part of address object and has user’s street address |
| city | String | A part of address object and has user’s city address |
| parish | String | A part of address object and has user’s parish address |

PATCH

users/{id}

A PATCH requests made to this route finds a single element or user from the database, and updates the user with the data that is passed in the request.

**Parameters**

The parameter for this request needs to be the id which represents the user that should be found in the database, along with the data that should be used to update the user found. The request will only update the data passed.

*id: string;*

*body: Partial<user>;*

**Response**

The response of the request would be a JSON response with the data being the user that was found and updated.

If there was an error, the API will return a message telling the user the error that occurred. If the id is invalid, mongoose with throw an error. However, if the data passed to the PATCH request is empty, the API will respond with a **bad request (HTTP 400)**.

**Response Example**

{

    "status": "Success",

    "message": "Successfully updated",

    "data": {

        "address": {

            "street": "King's Street ",

            "city": "Kingston ",

            "parish": "Kingston "

        },

        "\_id": "6318e0010aa7540cdd124c3f",

        "fname": "Tester",

        "lname": "Dev",

        "email": "tester1@mailme.com",

        "phone": "876-834-9842",

        "username": "Tester Dev",

        "\_\_v": 0

    }

}

**Response Data**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| Status | String | Describe the status of the response sent. Can be ‘Success’ or “Failed”. |
| message | String | Short description of operation that was perform or error message if there is an error |
| data | Object | An object which represent the updated information of the user that was updated. |
| fname | String | The first name of the user |
| lname | String | The last name of the user |
| email | String | The user’s email address |
| phone | Number | The user’s telephone number |
| username | String | The user’s username |
| address | Object | An Object containing the address information. |
| street | String | A part of address object and has user’s street address |
| city | String | A part of address object and has user’s city address |
| parish | String | A part of address object and has user’s parish address |

DELETE

users/{id}

A DELETE requests made to this route delete a single element or user from the database that matches the id passed.

**Parameters**

The parameter for this request needs to be the id which represents the user that should be found and removed from the database.

*id: string;*

**Response**

The response of the request would be a JSON response with the data being the user that was found and deleted.

If the id passed is an invalid ObjectId, mongoose will return a message telling the user the error that occurred. Also, if the id passed does not find the user to delete, the API responds that the user does not exist.

**Response Example**

{

    "status": "Success",

    "message": "Successfully Deleted",

    "data": {

        "\_id": "631adb244444cc9a445d54af",

        "fname": "Carlton",

        "lname": "Banks",

        "email": "carltonb@mailme.com",

        "phone": "876-789-8433",

        "username": "carlton"

        "\_\_v": 0

    }

}

**Response Data**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| Status | String | Describe the status of the response sent. Can be ‘Success’ or “Failed”. |
| message | String | Short description of operation that was perform or error message if there is an error |
| data | Object | An object which represent the updated information of the user that was deleted. |
| fname | String | The first name of the user |
| lname | String | The last name of the user |
| email | String | The user’s email address |
| phone | Number | The user’s telephone number |
| username | String | The user’s username |

POST

users/login

A post request made to this route will allow the user to create a JWT token to personalize a session with the user.

**Parameters**

*body: Partial<user>: Specifically, email and password.*

**Response**

The response of the request would be a JSON response with the data being the user that was found, as well as a hashed token which represents the user that was found.

If the email is not found in the database. The API responds with Not Found (HTTP 404).

If the password does not match, the API responds with bad request (HTTP 400).

**Response Example**

{

    "status": "Success",

    "message": "Successfully Logged in",

    "data": {

        "user": {

            "\_id": "631ae3914395d43389911e6c",

            "fname": "Carlton",

            "lname": "Banks",

            "email": "carltonb@mailme.com",

            "phone": "876-789-8433",

            "username": "carlton",

            "createdAt": "2022-09-09T06:56:17.247Z",

            "\_\_v": 0

        },

        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2MzFhZTM5MTQzOTVkNDMzODk5MTFlNmMiLCJ1c2VybmFtZSI6ImNhcmx0b24iLCJlbWFpbCI6ImNhcmx0b25iQG1haWxtZS5jb20iLCJyb2xlIjoidXNlciIsImlhdCI6MTY2MjcwNjc5MywiZXhwIjoxNjYyNzEwMzkzfQ.sp\_Byy8f8s31yY2tZl85e5daZB5zkNcTxA1xWIXcI-o"

    }

}

**Response Data**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| Status | String | Describe the status of the response sent. Can be ‘Success’ or “Failed”. |
| message | String | Short description of operation that was perform or error message if there is an error |
| data | Object | An object which contains the user as well as the token if user is successfully authenticated |
| token | String | A signed JWT token that is created upon successfully authenticating user info. |
| user | Object | Represents the user information that matched the logged in user. |
| fname | String | The first name of the user |
| lname | String | The last name of the user |
| email | String | The user’s email address |
| phone | Number | The user’s telephone number |
| username | String | The user’s username |

### appointments/

Manages appointments created by both the users and administrator.

POST

appointments/

A POST request made to the appointments/ resource route will create an appointment with the data that is passed in the body of the request.

**Parameters**

*body: Partial<appointment>;*

**Response**

The response returns a JSON with content type *application/json* along with a status code, the data returned will be the appointment that was created.

If there is an error with the data that is passed to the route, the API will return a bad request (HTTP 400).

GET

appointments/

A GET requests made to the appointments/ resource route will retrieve all appointments from the database. There are no parameters for this GET request.

**Response**

The response of the request would be an array of the appointments in the system. If there are no appointments presently in the system, then the API returns an empty array.

GET

appointments?userId={user\_id}

A GET requests made to the appointments/ resource route will retrieve all appointments from the database that has a userId which matches the one passed as a query parameter.

**Parameters**

*user\_id: string; Represents the id of the person that creates the appointment*

**Response**

The response of the request would be an array of the appointments in the system, whose userId property matches the user\_id passed in the URL.

If there are no appointments match, then the API returns an empty array.

If the user\_id passed is not a valid ObjectId, the API return Bad Request (HTTP 400).

appointments/{id}

GET

A GET requests made to this route returns a single Appointment from the database.

**Parameters**

The parameter for this request needs to be the id which represents the appointment that should be found in the database.

*id: string;*

**Response**

The response of the request would be a JSON response with the data being the appointment that was found that matches the id that was passed as a parameter to the route.

If no appointment is found with that id the API returns a Not Found (HTTP 404).

appointments/{id}

PATCH

A PATCH requests made to this route finds a single appointment from the database, and updates the appointment with the data that is passed in the request.

**Parameters**

The parameter for this request needs to be the id which represents the appointment that should be found in the database, along with the data that should be used to update the appointment found. The request will only update the data passed.

*id: string;*

*body: Partial<appointment>;*

**Response**

The response of the request would be a JSON response with the data being the appointment that was found and updated.

If there was an error, the API will return a message telling the appointment the error that occurred. If the id is invalid, mongoose with throw an error. However, if the data passed to the PATCH request is empty, the API will respond with a **bad request (HTTP 400)**.

DELETE

appointments/{id}

A DELETE requests made to this route delete an appointment from the database that matches the id passed.

**Parameters**

The parameter for this request needs to be the id which represents the appointment that should be found and removed from the database.

*id: string;*

**Response**

The response of the request would be a JSON response with the data being the appointment that was found and deleted.

If the id passed is an invalid ObjectId, mongoose will return a message telling the appointment the error that occurred. Also, if the id passed does not find the appointment to delete, the API responds that the appointment does not exist.

### doctors/

Manages users that are registered to the web application.

POST

doctors/

A POST request made to the doctors/ resource route will create a doctor with the data that is passed in the body of the request.

**Parameters**

*body: Partial<doctor>;*

**Response**

The response returns a JSON with content type *application/json*  with the doctor infoalong with a status code.

Whenever there is an error especially if there is an error with the data passed in the route. The API will return a message indicating what the error that occurred was.

If there is no data passed to the route, then the server returns Bad Request (HTTP 400).

GET

doctors/

A GET requests made to the doctors/ resource route will retrieve all doctors from the database. There are no parameters for this GET request.

**Response**

The response of the request would be an array of the doctors in the system. If there are no doctors presently in the system, then the API returns an empty array.

GET

doctors?department={department\_name}

A GET requests made to the doctors/ resource route will retrieve all doctor from the database that has a department which matches the one passed as a query parameter.

**Parameters**

*department\_name: string; Represents the departments in the system.*

**Response**

The response of the request would be an array of the doctors in the system, whose department property matches the *department\_name* passed in the URL.

If there are no doctors that match, then the API returns an empty array.

If the *department\_name* passed is not within an enumerated list then, the API return Bad Request (HTTP 400).

doctors/{id}

GET

A GET requests made to this route returns a single element or doctor from the database.

**Parameters**

The parameter for this request needs to be the id which represents the doctor that should be found in the database.

*id: string;*

**Response**

The response of the request would be a JSON response with the data being the doctor that was found that matches the id that was passed as a parameter to the route.

If there was an error, the API will return a message saying the error that occurred.

PATCH

doctors/{id}

A PATCH requests made to this route finds a single element or doctor from the database, and updates the doctor with the data that is passed in the request.

**Parameters**

The parameter for this request needs to be the id which represents the doctor that should be found in the database, along with the data that should be used to update the doctor found. The request will only update the data passed.

*id: string;*

*body: Partial<doctor>;*

**Response**

The response of the request would be a JSON response with the data being the doctor that was found and updated.

If there was an error, the API will return a message telling the user the error that occurred. If the id is invalid, mongoose with throw an error. However, if the data passed to the PATCH request is empty, the API will respond with a **bad request (HTTP 400)**.

DELETE

doctors/{id}

A DELETE requests made to this route delete a single element or user from the database that matches the id passed.

**Parameters**

The parameter for this request needs to be the id which represents the doctor that should be found and removed from the database.

*id: string;*

**Response**

The response of the request would be a JSON response with the data being the user that was found and deleted.

If the id passed is an invalid ObjectId, mongoose will return a message saying that an error occurred. Also, if the id passed does not find the doctor to delete, the API responds that the doctor does not exist.

POST

doctors/login

A post request made to this route will allow the user to create a JWT token to personalize a session with the doctor.

**Parameters**

*body: Partial<doctor>: Specifically, email and password.*

**Response**

The response of the request would be a JSON response with the data being the doctor that was found, as well as a hashed token which represents the doctor that was found.

If the email is not found in the database. The API responds with Not Found (HTTP 404).

If the password does not match, the API responds with bad request (HTTP 400).

### patients/

Manages patients of the hospitals as they come in for treatment.

POST

patients/

A POST request made to the patients/ resource route will create a patient with the data that is passed in the body of the request.

**Parameters**

*body: Partial<patient>;*

**Response**

The response returns a JSON with content type *application/json* along with a status code and the patient created from the data.

Whenever there is an error especially if there is an error with the data passed in the route. The API will return a message indicating what the error that occurred was. If there is no data passed to the route, then the server returns Bad Request (HTTP 400).

GET

patients/

A GET requests made to the patients/ resource route will retrieve all patients from the database. There are no parameters for this GET request.

**Response**

The response of the request would be an array of the patients in the system. If there are no patients presently in the system, then the API returns an empty array.

GET

patients?admit\_doctor={doctor\_id}

A GET requests made to the patients/ resource route will retrieve all patients from the database that has a doctor which matches the one passed as a query parameter.

**Parameters**

*doctor\_id: string; Represents the id of a doctor in the system.*

**Response**

The response of the request would be an array of the patient in the system, whose admit\_doctor property matches the *doctor\_id* passed in the URL.

If there are no doctors that match, then the API returns an empty array.

patients/{id}

GET

A GET requests made to this route returns a single element or patient from the database.

**Parameters**

The parameter for this request needs to be the id which represents the patient that should be found in the database.

*id: string;*

**Response**

The response of the request would be a JSON response with the data being the patient that was found that matches the id that was passed as a parameter to the route.

If there was an error, the API will return a message saying the error that occurred.

PATCH

patients/{id}

A PATCH requests made to this route finds a single element or patient from the database, and updates the patient with the data that is passed in the request.

**Parameters**

The parameter for this request needs to be the id which represents the patient that should be found in the database, along with the data that should be used to update the patient found. The request will only update the data passed.

*id: string;*

*body: Partial<patient>;*

**Response**

The response of the request would be a JSON response with the data being the patient that was found and updated.

If there was an error, the API will return a message telling the user the error that occurred. If the id is invalid, mongoose with throw an error. However, if the data passed to the PATCH request is empty, the API will respond with a **bad request (HTTP 400)**.

DELETE

patients/{id}

A DELETE requests made to this route delete a single element or patient from the database that matches the id passed.

**Parameters**

The parameter for this request needs to be the id which represents the patient that should be found and removed from the database.

*id: string;*

**Response**

The response of the request would be a JSON response with the data being the patient that was found and deleted.

If the id passed is an invalid ObjectId, mongoose will return a message telling the user the error that occurred. Also, if the id passed does not find the doctor to delete, the API responds that the patient does not exist.

### records/

Manages records of patients in the hospitals as they come in for treatment.

POST

records/

A POST request made to the records route will create a patient with the data that is passed in the body of the request.

**Parameters**

*body: Partial<medRecord>;*

**Response**

The response returns a JSON with content type *application/json* along with a status code and the medical record created from the data.

Whenever there is an error especially if there is an error with the data passed in the route. The API will return a message indicating what the error that occurred was. If there is no data passed to the route, then the server returns Bad Request (HTTP 400).

GET

records/

A GET requests made to the records route will retrieve all patients from the database. There are no parameters for this GET request.

**Response**

The response of the request would be an array of the medical records in the system. If there are no medical record presently in the system, then the API returns an empty array.

GET

records?patient={patient\_id}

A GET requests made to the record route will retrieve all medical records from the database that has medical records which matches the one passed as a query parameter.

**Parameters**

*patient\_id: string; Represents the id of a patient in the system.*

**Response**

The response of the request would be an array of the patient in the system, whose patient property matches the *patient\_id* passed in the URL.

If there are no medical records that match, then the API returns an empty array.

records/{id}

GET

A GET requests made to this route returns a single medical record from the database.

**Parameters**

The parameter for this request needs to be the id which represents the medical record that should be found in the database.

*id: string;*

**Response**

The response of the request would be a JSON response with the data being the medical record that was found that matches the id that was passed as a parameter to the route.

If there was an error, the API will return a message saying the error that occurred.

PATCH

records/{id}

A PATCH requests made to this route finds a single element or medical record from the database, and updates the patient with the data that is passed in the request.

**Parameters**

The parameter for this request needs to be the id which represents the medical record that should be found in the database, along with the data that should be used to update the medical record found. The request will only update the data passed.

*id: string;*

*body: Partial<medRec>;*

**Response**

The response of the request would be a JSON response with the data being the medical record that was found and updated.

If there was an error, the API will return a message telling the user the error that occurred. If the id is invalid, mongoose with throw an error. However, if the data passed to the PATCH request is empty, the API will respond with a **bad request (HTTP 400)**.

DELETE

records/{id}

A DELETE requests made to this route delete a single element or medical record from the database that matches the id passed.

**Parameters**

The parameter for this request needs to be the id which represents the medical record that should be found and removed from the database.

*id: string;*

**Response**

The response of the request would be a JSON response with the data being the medical record that was found and deleted.

If the id passed is an invalid ObjectId, mongoose will return a message saying that an error occurred. Also, if the id passed does not find the medical record to delete, the API responds that the medical record does not exist.

### Data Models

The data models that we used for the Pediatric Hospital of Jamaica application are as follows.

#### API Response Model

{

    status: string;

    data: T;

    message?:string;

}

*Where ‘T’ represents the type of data that will be received in the request. This is sent with all response from the API.*

#### User Model

 {

    \_id: string;

    fname: string;

    lname: string;

    email:string;

    phone: string;

    imageUrl: string;

    username: string;

    password: string;

    address:{

        street: string,

        city:string,

        parish: string,

    }

}

#### Appointment Model

{

    \_id: string;

    doctor: string;

    visitStart: Date;

    fname: string;

    lname:string;

    email:string;

    notes:string;

    status: string;

    phone:string;

    userId:string;

}

#### Doctor Model

{

    \_id: string;

    fname: string;

    lname: string,

    title: string,

    imageUrl: string,

    email: string,

    phone: string,

    department: string;

    username: string;

    address: {

        street: string,

        city: string,

        parish: string,

    },

    patientCount: number;

    experience: number;

    status: string;

    password: string,

    rating: number,

    isSuperAdmin: boolean;

}

#### Patients Model

{

    \_id:string;

    fname: string;

    lname: string;

    mname:string;

    email: string;

    DOB: Date;

    patientImage: string;

    guardian\_fname:string;

    guardian\_lname:string;

    guardian\_address:[{

        street: string,

        city:string,

        parish: string

    }];

    phone: string;

    gender:string;

    allergies: string;

    nationality: string;

    admission\_date:Date;

    admit\_doctor: string;

}

#### MedRec Model

{

    patient: string

    complaint: string,

    diagnosis: string,

    comments: [{

        comment: string,

        date: Date

    }],

    prescription: string;

}

# References

Figma Design File: <https://www.figma.com/file/J9rpRBr1pduTkQt7kGlE6G/Children's-Medical-Center?node-id=486%3A319>

Front End Hosted on Firebase: <https://phj-app.web.app/>

Back End Hosted on Heroku: <https://phj-app.herokuapp.com/>

Front End Repository: <https://github.com/aldaineclarke/Pediatric-Hospital-of-Jamaica>

Backend Repository: <https://github.com/aldaineclarke/PHJBackend>