**CONSULT DOCTOR ONLINE**

**A Project Report**

*Submitted in partial fulfilment for the award of the degree*

*of*

**Master of Science**

***in***

**Information Technology**

*by*

**PAYYAVULA KALYANI**

**15MIN0241**

*Under the guidance of*

**Dr./Prof. Prasanna Santhanam**

**VIT University**

****



**School of Information Technology and Engineering**

August, 2019

****

**School of Information Technology and Engineering**

**DECLARATION BY THE CANDIDATE**

I hereby declare that the thesis entitled **“CONSULT DOCTOR ONLINE”** submitted by me to Vellore Institute of Technology University Vellore, in partial fulfillment of the requirement for the award of the degree of **Master of Science** in **Information Technology** is a record of bonafide project work carried out by me under the supervision of **Prof. Prasanna Santhanam**. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

**Place**: Bangalore Kalyani Payyavula

**Date**:17/08/2019 s**ignature of the Candidate**

****

**School of Information Technology and Engineering**

**BONAFIDE CERTIFICATE**

This is to certify that the project work entitled **“CONSULT DOCTOR ONLINE”** by **Payyavula Kalyani(15MIN0241),** to Vellore Institute of Technology University, Vellore, in partial fulfillment of the requirement for the award of the degree of **Master of Science** in **Information Technology**, is a project bonafide work carried out by him/her under my supervision. The project fulfills the requirement as per the regulations of this Institute and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this Institute or any other Institute or University.

**Prof. Prasanna Santhanam**

**Internal Supervisor**

**VIT University**

**Internal Examiner(s) External Examiner(s)**

**ABSTRACT:**

Recent studies shows 85% of doctors use smart phones and medical apps. A survey in 2015 suggest that 17% of world population use online doctor appointment. It also suggest that, This can exponentially increase up to 64%. Online appointment helps patients to better connect with their personal doctors. Using online appointment system will help both doctor and patient to check the history when required, This will help to avoid maintaining physical records.

Legacy applications use different technologies in front end and back end. Maintaining such applications needs team with diverse skills. Writing applications in MERN stack will help a team with JavaScript skills to develop and maintain entire stack. This will help us to deliver online appointment system with low cost.

Building applications in MERN stack, Will also help us to deliver mobile friendly applications with very less efforts. Thus using this technologies will help us to deliver both web and mobile applications in JavaScript.

**OBJECTIVE**

This application helps operational convenience and Smoothen the process of booking doctor appointment. This application eliminates the need for paper work to book doctor appointment.

**SCOPE**

The scope of the project include,

* Patient profile – signup.
* Allow user to login.
* Display available slot timings for doctor.
* Option to Book appointment.
* Option to cancel the appointment.

**SOFTWARE / HARDWARE REQUIREMENTS**

The below software/technology will be used for this project

* Front end development: HTML5, CSS3, JavaScript, ReactJS, MaterialUI
* Backend development: NodeJS, MongoDB, ExpressJS
* Unit testing : Jest & Enzyme, End to End testing : TestCafe
* End to End testing : TestCafe
* Version Control : Git

**ACKNOWLEDGEMENT**

Foremost, I would like to express my sincere and heartfelt gratitude to my project guide **Prof. Prasanna Santhanam** for his valuable help and guidance. I am thankful to him for the encouragement he gave me in completing the project.

I am also thankful to the all the professors of VIT University, for imparting their valuable knowledge on software development. It’s their guidance and knowledge which helped me to solve real world problems with software application. This application gave me a strong hands on experience on various stages of software development like requirements gathering, design, implementation and testing.

Lastly, I would like to express my deep appreciation towards my classmates for providing me moral support and encouragement.

Kalyani Payyavula

**(15MIN0241)**

Table of Contents

[LIST OF TABLES 9](#_Toc16787588)

[List of Figures 9](#_Toc16787589)

[LIST OF Abbreviations 9](#_Toc16787590)

[1. INTRODUCTION 10](#_Toc16787591)

[1.1 PURPOSE 10](#_Toc16787592)

[1.2 SCOPE 11](#_Toc16787593)

[1.3 REFERENCES 11](#_Toc16787594)

[1.4 DEFINITIONS, ACRONYMS AND ABBREVATIONS 12](#_Toc16787595)

[1.5 USER CHARACTERISTICS 12](#_Toc16787596)

[1.6 GENERAL CONSTRAINTS 13](#_Toc16787597)

[1.7 ASSUMPTIONS AND DEPENDENCIES 13](#_Toc16787598)

[2 Literature Survey 13](#_Toc16787599)

[2.1 PRODUCT PERSPECTIVE 14](#_Toc16787600)

[2.2 PRODUCT FUNCTIONS 15](#_Toc16787601)

[2.3 GENERAL CONSTRAINTS 15](#_Toc16787602)

[2.4 EXTERNAL INTERFACE REQUIREMENTS 16](#_Toc16787603)

[2.4.1 User Interfaces 16](#_Toc16787604)

[2.4.2 Hardware Interfaces 16](#_Toc16787605)

[2.4.3 Software Interfaces 16](#_Toc16787606)

[2.4.4 Communications Interfaces 16](#_Toc16787607)

[2.5 FUNCTIONAL REQUIREMENTS 17](#_Toc16787608)

[2.5.1 Home Page Requirements: 17](#_Toc16787609)

[2.5.2 New user registration Requirements: 17](#_Toc16787610)

[2.5.3 Doctor Availability Requirements: 17](#_Toc16787611)

[2.5.4 Book Appointment Requirements: 18](#_Toc16787612)

[2.5.5 Check History Requirements: 18](#_Toc16787613)

[2.5.6 Profile Management Requirements: 19](#_Toc16787614)

[2.5.7 Doctor Requirements: 19](#_Toc16787615)

[3 Detailed Design of the system 20](#_Toc16787616)

[3.1 System Architecture 20](#_Toc16787617)

[3.1.1 Single-Page Applications: 20](#_Toc16787618)

[3.1.2 MERN Stack: 21](#_Toc16787619)

[3.1.3 React with Flux Architecture: 23](#_Toc16787620)

[3.1.4 High level Design of Applicaiton 25](#_Toc16787621)

[3.2 DESIGN RATIONALE 26](#_Toc16787622)

[3.2.1 Benefits of React JavaScript Library: 26](#_Toc16787623)

[3.2.2 Benefits of Node.js 27](#_Toc16787624)

[3.3 Sequence diagrams 30](#_Toc16787625)

[3.4 Entity-Relationship diagram 32](#_Toc16787626)

[3.5 Data Flow diagram 33](#_Toc16787627)

[4 Implementation of System 34](#_Toc16787628)

[4.1 HOME PAGE 34](#_Toc16787629)

[4.2 BOOK APPOINTMENT SCREEN 35](#_Toc16787630)

[4.3 LOGIN SCREEN 37](#_Toc16787631)

[4.4 PATIENT HOME SCREEN 38](#_Toc16787632)

[4.5 APPOINTMENT DETAILS 41](#_Toc16787633)

[4.6 REGISTER A NEW USER 43](#_Toc16787634)

[4.7 HISTORY OF THE APPOINTMENTS 44](#_Toc16787635)

[4.8 Reusable Components 47](#_Toc16787636)

[4.8.1 Application Header 47](#_Toc16787637)

[4.8.2 Application Footer 48](#_Toc16787638)

[4.8.3 Menu Layout 48](#_Toc16787639)

[4.8.4 Page Layout 49](#_Toc16787640)

[4.8.5 Application Notification 50](#_Toc16787641)

[4.8.6 Appointment Grid 51](#_Toc16787642)

[4.9 Server 55](#_Toc16787643)

[4.9.1 Express JS Interface 55](#_Toc16787644)

[4.9.2 Home Screen Interface 56](#_Toc16787645)

[4.10 Screen Shots 57](#_Toc16787646)

[4.10.1 Home Page 57](#_Toc16787647)

[4.10.2 Book Appointment Page 58](#_Toc16787648)

[4.10.3 Appointment Details Page 60](#_Toc16787649)

[4.10.4 Login Page 60](#_Toc16787650)

[4.10.5 History Page 61](#_Toc16787651)

[4.10.6 Profile Page 61](#_Toc16787652)

[4.10.7 Register New User 62](#_Toc16787653)

[5 Results & Disucssions 62](#_Toc16787654)

[5.1 TEST PLAN 62](#_Toc16787655)

[5.1.1 Purpose 62](#_Toc16787656)

[5.1.2 Scope 62](#_Toc16787657)

[5.1.3 Functions to be tested 63](#_Toc16787658)

[*5.2* TESTING PROCESS OVERVIEW 63](#_Toc16787659)

[5.2.1 Preparing Test Cases: 63](#_Toc16787660)

[5.2.2 Preparing Test Matrix: 63](#_Toc16787661)

[5.2.3 Reviewing test cases and matrix: 63](#_Toc16787662)

[5.2.4 Creating Test Data: 63](#_Toc16787663)

[5.2.5 Executing Test Cases: 64](#_Toc16787664)

[5.2.6 Defect Logging and Reporting: 64](#_Toc16787665)

[5.2.7 Retesting and Regression Testing: 64](#_Toc16787666)

[5.2.8 Deployment/Delivery: 64](#_Toc16787667)

[5.3 TEST STRATEGY 65](#_Toc16787668)

[5.3.1 Black box testing: 65](#_Toc16787669)

[5.3.2 GUI Testing: 65](#_Toc16787670)

[5.3.3 Functional Testing: 65](#_Toc16787671)

[5.3.4 User acceptance testing: 65](#_Toc16787672)

[5.4 REVIEWS 66](#_Toc16787673)

[5.5 Change Request: 66](#_Toc16787674)

[5.6 Defect Reporting: 66](#_Toc16787675)

[5.6.1 Entry Criteria 66](#_Toc16787676)

[5.6.2 Suspension Criteria 66](#_Toc16787677)

[5.6.3 Exit Criteria 67](#_Toc16787678)

[5.6.4 Risk 67](#_Toc16787679)

[5.7 Unit Test: 67](#_Toc16787680)

[5.8 Integration Testing 68](#_Toc16787681)

[5.9 System Testing 70](#_Toc16787682)

[5.10 TEST CYCLES and TEST REPORTS 71](#_Toc16787683)

[6 Conclusion and Future Work 72](#_Toc16787684)

LIST OF TABLES

|  |  |  |
| --- | --- | --- |
| **Table No** | **Title** | **Page No** |
| 1 | Unit Test | 48 |
| 2 | Integration Test | 49 |
| 3 | System Testing | 51 |

List of Figures

|  |  |  |
| --- | --- | --- |
| **Figure No** | **Title** | **Page No** |
| 1 | Home page | 39 |
| 2 | Book Appointment page | 40 |
| 3 | Appointment details page | 42 |
| 4 | Login page | 42 |
| 5 | History page | 43 |
| 6 | Profile page | 43 |

LIST OF Abbreviations

|  |  |
| --- | --- |
| **ACRONYM** | **EXPANSION** |
| FR | Functional Requirements |
| NFR | Non Functional Requirements |
| NURR | New User Registration Requirements |
| DAR | Doctor Availability Requirements |
| BAR | Book Appointment Requirements. |
| PM | Profile Management |

# INTRODUCTION

As the population continues to grow, so does the need for healthcare services. Exponential increase in the number of patients, creates new challenges for facility administrators and staff. Legacy processes and procedures that clinics followed may no longer be effective in handling a raise in new patients. One process affected almost every patient and doctor from an increase in patients is appointment-scheduling. This seemingly simple task can quickly become burdensome and challenging to staff members. Most of the clinics and doctors tend to do scheduling appointment over the phone which consumes most of the time and resources. In the next few years most of the individuals prefer to do common tasks online instead of picking up the phone and calling doctor.

Few benefits of online appointment are Time Savings, Monetary Saving and 24 -hour convenience. Let’s take simple example of clinic which does 100 appointments daily on phone. Each appointment call can take an average of 4 minutes which result in 400 minutes or seven hours. One staff member is dedicated to take appointments. Both time and expense required for this staff member can be saved. Sending automated remainders to patients helps to decrease the number of ‘no-shows’. This can directly impact on the revenue of the clinic. Scheduling appointments over the phone usually requires an individual to phone in during office hours. This can cause inconvenience for most patients, as they too are working at this time. An online scheduling system allows for 24-hour scheduling, not just during normal facility of office hours.

Facilities or Clinics with younger patients and clients can have an easy transition to online-appointment, as this group is typically internet savvy and actually prefer to schedule appointments online. On the other hand, hospitals with large number of senior patients and clients would not have an that easy transition to online appointment. However, few surveys show that Internet users in the age group of 50-64 grew exponentially day by day. In order to address the group which doesn’t use internet, clinics may continue to do phone appointment for few more years.

## PURPOSE

In my observation almost, all clinics or doctors are trying to solve the same problem, allow patients to book appointments round the clock. I would like to provide a common solution which can solve the problem of all patients and doctors. Provide a configurable application which can be configured for a particular clinic or doctor. Allow administrator to add/delete doctors to the clinic. Patients should be able to book appointment right from the mobile browser or desktop browser. Scheduling appointment online can benefit everyone involved in the scheduling process, as administrators and staff can conduct their task more efficiently and accurately, while customers and clients have the ability to book their appointments and reservations quickly and more conveniently.

## SCOPE

This document will cover design aspects on allowing users to book doctor appointments online. This covers both functional and non-functional aspects of the use case. In the functional scope it covers about the use case of patient, doctor and clinic admin. It covers in detail about how the application allows users to register, login and book appointments. It also covers about how users can check the history of the appointments. To make the use case complete this talk about how users can cancel/edit the appointments which are already in place.

In the non-functional aspects of the use case this document talks about how to avoid hackers to make appointment. This covers on how the system should be build robust so that it won’t execute the scripts injected by hackers. This also covers requirements on system performance, portability and saving user identity.

This document doesn’t speak about the design or technical aspects of the implementation. For more information on design, implementation and testing, I would recommend to refer respective documents.

## REFERENCES

|  |  |
| --- | --- |
| Reference |  |
| Document location | <https://github.com/KalyaniPayyavula/consult_doctor/tree/master/documents> |
| Implementation & Testing documents | <https://github.com/KalyaniPayyavula/consult_doctor/tree/master/documents> |
| Apollo Appointment | <https://www.askapollo.com/physical-appointment/> |
| Manipal Appointment | <https://www.manipalhospitals.com/appointments> |
| Sakra Hospital Appointment | <https://www.sakraworldhospital.com/request-appointment.php> |
| Online appointments | <https://www.appointmentplus.com/> |
| Single Page Application | <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx> |
| MERN Stack | <https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack> |
| React with Fluz Architecture | <https://dzone.com/articles/a-detailed-study-of-flux-the-reactjs-application-a> |
| Express | <https://expressjs.com/> |
| MongoDB | https://www.mongodb.com/ |

## DEFINITIONS, ACRONYMS AND ABBREVATIONS

FR – Functional Requirements

NFR – Non Functional Requirements

NURR – New User Registration Requirements

DAR – Doctor Availability Requirements

BAR – Book Appointment Requirements.

PM – Profile Management

CH – Check History

## USER CHARACTERISTICS

There are many eventual users who will use my application. The general characteristics of the eventual users of the product that will affect the specific requirements are as below:

1. **New Patient:** New Patient is someone who is approaching doctor or facility for the first time.

Functionalities:

* Register himself with all personal details.
* Check doctor availability and book appointment.
* Check history of appointments.
* Edit/ delete appointments already made.
* Edit his personal profile.
* Take appointment for his/her family member.

1. **Doctor:**  User authorized by admin as doctor.

Functionalities:

* Updates his/her availability.
* Can have access to patient details who has taken appointment.
* Mark appointment as completed/handled.
* Give future appointments for patients.
* Ability to cancel appointments on emergency.

1. **Admin:**  User authorized to add doctor profile and generate access credentials for doctors.

Functionalities:

* Updates list of doctors available in the clinic.
* Has permissions to add/remove doctors.
* Controls appointment remainders.
* Maintains the system and sole responsible for maintaining appointments.

## GENERAL CONSTRAINTS

While the actual implementation of an online scheduling system is typically seamless and relatively simple, there are considerations that medical, healthcare and wellness facilities should keep in mind when transitioning to Web-based booking. They include:

Optional or required? One question administrator should answer is whether or not to make online scheduling a requirement. Requiring that all appointments be made online can certainly free up staff responsibilities and schedules, but it can also be a hindrance to those without easy access to the Internet or who prefer to schedule their appointments over the phone. Many facilities give their patients and clients the option of booking online, which typically brings good results.

## ASSUMPTIONS AND DEPENDENCIES

* Each facility has a technical person who can manage the application.
* Admin will not remove/add list of doctors on his own wish.
* This application needs user with internet connection.
* Doctors don’t make sudden surprise to patients by cancelling the appointments.
* Patients who don’t make the appointments on time should be given opportunity in the next available slot.

# Literature Survey

The appointment-scheduling process, historically viewed as a necessary burden in medical offices, healthcare facilities and wellness centers, can be completely automated through an inefficient online scheduling software program. The benefits of implementing this technology touch everyone involved in the scheduling process, as administrators and staff can conduct their tasks more efficiently and accurately, while customers and clients have the ability to book their appointments and reservations quickly and more conveniently.

The Benefits of an Online Scheduling System:

Commonly referred to by such names as online scheduling software, online booking applications and online scheduler, an online scheduling system is a Web-based application that allows individuals to conveniently and securely book their appointments and reservations online through any Web-connected device, such as a computer, laptop, smartphone or tablet. They typically access the online scheduling system through a “Book Now” button found on a Web site or page, or from a URL provided to them by the medial, healthcare or wellness facility. Once a date and time are selected, the system will automatically confirm the booking and instantly record it within the system, without any staff action needed.

In addition to online scheduling, online scheduling systems also come equipped with other beneficial features like automated e-mail and text message reminders, which the system sends out to patients and booked individuals on a specific date prior to their scheduled appointment; recording and record-keeping capabilities that make it quick and simple to access data associated with a specific appointment; and repeat patient reminders, which the system sends out automatically when a specified amount of time has expired between appointments.

24-Hour Convenience:

Scheduling appointments over the phone usually requires an individual to phone in during office hours, as few facilities offer round-the-clock phone booking. This is an inconvenience for most patients, as they too are working at this time. Additionally, many individuals prefer to schedule their appointments online rather than over the phone. An online scheduling system allows for 24-hour scheduling, not just during normal facility or office hours.

Time savings:

Staff spends less time on the phone booking and managing appointments, thereby freeing up their schedule for more important and pressing tasks. Booking individuals also save time, as they no longer have to commit a part of their busy schedule to calling their medical, healthcare or wellness provider (or remain on hold, which adds minutes to the scheduling process).

As an example, let’s look at a large medical facility that typically schedules approximately 100 appointments daily. Each appointment call is fielded by an administrative support staffer, who spends an average of four minutes on the phone. This equates to an average of 400 minute or almost seven hours of time spend each day just to booking appointments over the phone.

That’s time savings just from scheduling appointments alone. Other tasks automated by an online scheduling system, such as automated appointment reminders, add additional time savings to daily operations.

## PRODUCT PERSPECTIVE

This could have an effect on the success of an online scheduling system whose goal is to provide online scheduling. Facilities, centers and practices with younger patients and clients may have an easy transition, as this group is typically Internet savvy and actually prefer to schedule appointments online. On the other end of the spectrum are sites with a large number of senior patients and clients. HIPAA compliance. Given the online interaction and transfer of information between a facility and its patients, compliance of Health Insurance Portability and Accountability Act (HIPAA) provisions is an important consideration when utilizing an online scheduling system.

## PRODUCT FUNCTIONS

There are many eventual users who will use my application. The general characteristics of the eventual users of the product that will affect the specific requirements are as below:

1. **New Patient:** New Patient is someone who is approaching doctor or facility for the first time.

Functionalities:

* Register himself with all personal details.
* Check doctor availability and book appointment.
* Check history of appointments.
* Edit/ delete appointments already made.
* Edit his personal profile.
* Take appointment for his/her family member.

1. **Doctor:**  User authorized by admin as doctor.

Functionalities:

* Updates his/her availability.
* Can have access to patient details who has taken appointment.
* Mark appointment as completed/handled.
* Give future appointments for patients.
* Ability to cancel appointments on emergency.

1. **Admin:**  User authorized to add doctor profile and generate access credentials for doctors.

Functionalities:

* Updates list of doctors available in the clinic.
* Has permissions to add/remove doctors.
* Controls appointment remainders.
* Maintains the system and sole responsible for maintaining appointments.

## GENERAL CONSTRAINTS

While the actual implementation of an online scheduling system is typically seamless and relatively simple, there are considerations that medical, healthcare and wellness facilities should keep in mind when transitioning to Web-based booking. They include:

Optional or required? One question administrator should answer is whether or not to make online scheduling a requirement. Requiring that all appointments be made online can certainly free up staff responsibilities and schedules, but it can also be a hindrance to those without easy access to the Internet or who prefer to schedule their appointments over the phone. Many facilities give their patients and clients the option of booking online, which typically brings good results.

## EXTERNAL INTERFACE REQUIREMENTS

### User Interfaces

* System shall provide browser interface to use the system.
* Application shall launch successfully in all popular browsers like IE, Google Chrome, Safari and Firefox.
* System shall also allow users to use Mobile browsers.
* System shall work fine with Android and iOS default browsers.

### Hardware Interfaces

* Application shall not take any input from hardware.
* Application shall save data to database only on valid session.
* Application shall not allow any external hardware to read user data.
* This system doesn’t read/write data to any hardware.

### Software Interfaces

* System shall not accept data from any external software.
* System shall write data to database server.
* System shall read data form database server after successful authentication.

### Communications Interfaces

* System shall communicate with database to store user details and doctor details.
* User details are only available to doctor with read access.
* User is the only interface which inserts data into the system.
* Doctor can only read the data of user, he cannot modify it.

## FUNCTIONAL REQUIREMENTS

### Home Page Requirements:

|  |  |
| --- | --- |
| Requirement ID | Requirement |
| FR-HomePage-1 | System shall display list of doctors and their specialization. |
| FR-HomePage-2 | List of doctors available in a clinic can be edited / modified. |
| FR-HomePage-3 | Home page should include doctor timings, doctor qualifications. |
| FR-HomePage-4 | Home page shall display a link to check availability of particular doctor. |
| FR-HomePage-5 | Home page shall display an option to register new user. |
| FR-HomePage-6 | Home page shall allow already registered user to login. |
| FR-HomePage-7 | Home page shall allow Guest user to check his Appointments. |

### New user registration Requirements:

|  |  |
| --- | --- |
| Requirement ID | Requirement |
| FR-NURR-1 | System shall allow user to register with his personal details so that he can use them later to book appointment. |
| FR-NURR-2 | System shall ask for following details.   * User Id \* * Password\* * Confirmation password\* * Email\* * Mobile Number\* * Address: Line 1, Line 2, Pin Code   Fields marked with \* are mandatory. |
| FR-NURR-3 | System shall validate user provided details and provide a confirmation message to the user. |

### Doctor Availability Requirements:

|  |  |
| --- | --- |
| Requirement ID | Requirement |
| FR-DAR-1 | System shall display list of available dates with green color. |
| FR-DAR-2 | On selecting a date with green color, system shall display an overlay to the user with list of available Slots. |
| FR-DAR-3 | System shall allow user to check doctor availability for next 3 months. |
| FR-DAR-4 | If doctor is not available for any reason system shall display such reason on the screen. |

### Book Appointment Requirements:

|  |  |
| --- | --- |
| Requirement ID | Requirement |
| FR-BAR-1 | System shall allow user to pick an available slot and proceed with appointment booking. |
| FR-BAR-2 | System shall block a slot for a user and doesn’t show that slot for any other user. |
| FR-BAR-3 | System shall allow user to book appointment even without login as guest user. |
| FR-BAR-4 | Guest user will be asked to enter details like  Name\*  Mobile number\*  Email id\*  Problem description\*  Address:  Fields marked with \* are mandatory. |
| FR-BAR-5 | System shall allow registered user to login and book appointment. In this use case System shall pick user details directly. |
| FR-BAR-6 | System shall display an option to select if the appointment is for his/her family member. |
| FR-BAR-7 | When user selects family member option he will be asked to enter the details of family member. |
| FR-BAR-8 | Family details shall be saved by the system so that they will not be asked to enter again. |
| FR-BAR-9 | Each slot shall be 15 mins. This should be configurable by doctor. |
| FR-BAR-10 | Each user can book a maximum of 2 appointments per doctor. |

### Check History Requirements:

|  |  |
| --- | --- |
| Requirement ID | Requirement |
| FR-CH-1 | System shall allow registered user to check his history of appointments. |
| FR-CH-2 | If an appointment is in future user will be allowed to edit/cancel that appointment. |
| FR-CH-3 | When user cancel an appointment, it will be made available to other users for use. |
| FR-CH-4 | If user wishes to edit the time of appointment user shall be presented with available slots in the same page. |
| FR-CH-5 | When user cancel appointment, free the slot. |

### Profile Management Requirements:

|  |  |
| --- | --- |
| Requirement ID | Requirement |
| FR-PM-1 | System shall allow user to check his profile. |
| FR-PM-2 | System shall allow user to edit his profile if user feels so. |
| FR-PM-3 | System shall allow user to add his family members details. |
| FR-PM-4 | User can never change his user id once registered. |

### Doctor Requirements:

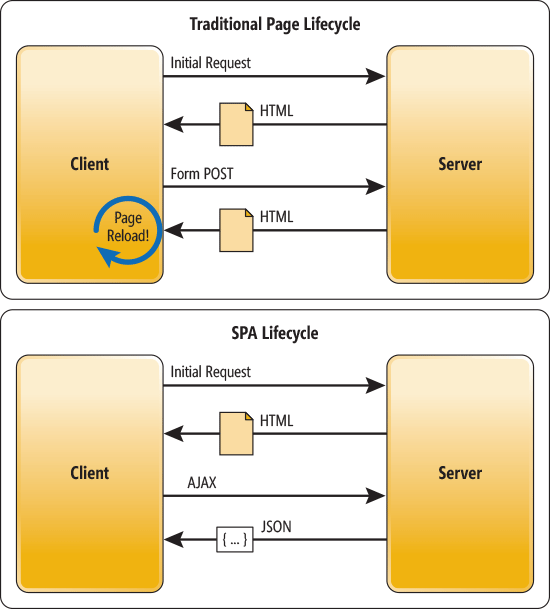
|  |  |
| --- | --- |
| Requirement ID | Requirement |
| FR-DR-1 | System shall allow admin to add doctor to clinic. |
| FR-DR-2 | Once doctor is added to clinic he/she will be provided login credentials with which they are allowed to enter their availability in the clinic. |
| FR-DR-3 | System shall allow doctor to check his filled slots for the next 3 months. |
| FR-DR-4 | System shall allow doctor to check the profile of the patient. |
| FR-DR-5 | System shall allow doctor to enter notes about the patient findings. |
| FR-DR-6 | System shall allow doctor to check previous visits of the patient and his findings. |

# Detailed Design of the system

## System Architecture

### Single-Page Applications:

Single-Page Applications (SPAs) are Web apps that load a single HTML page and dynamically update that page as the user interacts with the app.



In a traditional Web app, every time the app calls the server, the server renders a new HTML page. This triggers a page refresh in the browser. In an SPA, after the first page loads, all interaction with the server happens through AJAX calls. These AJAX calls return data—not markup—usually in JSON format. The app uses the JSON data to update the page dynamically, without reloading the page.

One benefit of SPAs is obvious: Applications are more fluid and responsive, without the jarring effect of reloading and re-rendering the page. Another benefit might be less obvious and it concerns how you architect a Web app. Sending the app data as JSON creates a separation between the presentation (HTML markup) and application logic (AJAX requests plus JSON responses).

In a pure SPA, all UI interaction occurs on the client side, through JavaScript and CSS. After the initial page load, the server acts purely as a service layer. The client just needs to know what HTTP requests to send. It doesn’t care how the server implements things on the back end.

With this architecture, the client and the service are independent. You could replace the entire back end that runs the service, and as long as you don’t change the API, you won’t break the client. The reverse is also true—you can replace the entire client app without changing the service layer. For example, you might write a native mobile client that consumes the service.

### MERN Stack:

I would like to choose MERN stack architecture for implementing this application. Here is the high-level architecture of MERN depicting how each component are connected.

#### **MongoDB: A cross-platform document database**

[MongoDB](https://www.mongodb.com/) is a NoSQL (non-relational) document-oriented database.

While conventional relational databases have a typical schema design based on columns and tables, MongoDB is schema-less. Data is stored in flexible documents with a JSON (JavaScript Object Notation)-based query language. The content, size, and number of fields in the documents can differ from one to the next. This means that the data structure to be changed over time.

MongoDB is known for being flexible and easy to scale. You can see multiple examples of [real-life MongoDB applications here](https://www.mongodb.com/use-cases).

#### **Express: A back-end web application framework**

[Express](https://expressjs.com/) is a web application framework for Node.js, another MERN component. Instead of writing full web server code by hand on Node.js directly, developers use Express to simplify the task of writing server code. There’s no need to repeat the same code over and over, as you would with the Node.js HTTP module.

The Express framework is designed for building robust web applications and APIs. It’s known for its fast speed and minimalist structure, with many features available as plugins.

#### **React: A JavaScript library for building user interfaces**

[React](https://reactjs.org/) was originally created by a software engineer at Facebook, and was later open-sourced. It is maintained by Facebook, as well as a community of development companies and individual developers.

The React library can be used for creating views rendered in HTML. React views are declarative. This means that developers don’t have to worry about managing the effects of changes in the view’s state (the object that determines how components behave) or changes in the data.

Instead of relying on templates to automate the creation of repetitive HTML or DOM (Document Object Model) elements, React uses a full-featured programming language (JavaScript) to construct repetitive or conditional DOM elements.

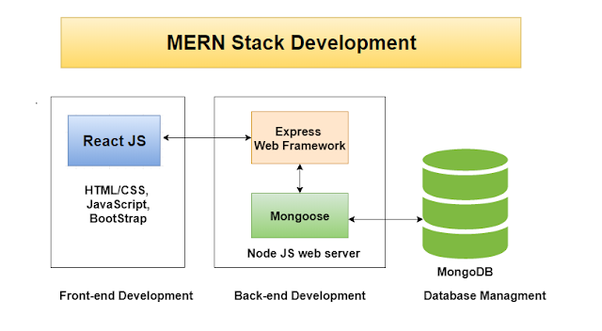
With React, the same code can run on both the server and the browser.

React anchors the MERN stack. In a way, it’s the defining feature of the stack. It’s the one component that differentiates MERN from MEAN, another popular JavaScript stack that uses AngularJS (a front-end web application framework) instead of the React library.

#### Node.js: A cross-platform JavaScript runtime environment

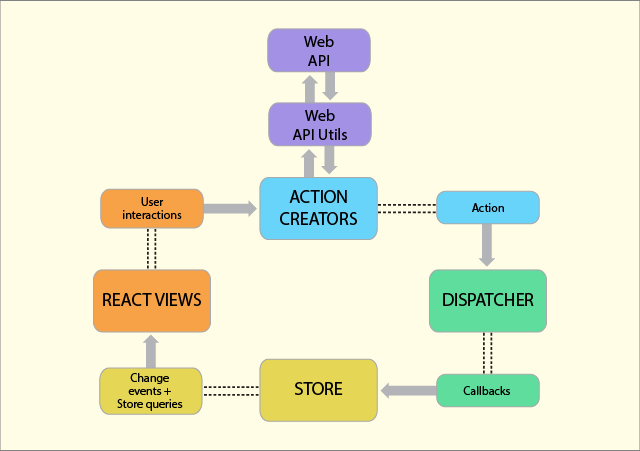
[Node.js](https://nodejs.org/en/) was initially built for Google Chrome, and later open-sourced by Google in 2008. It is built on Chrome’s V8 JavaScript engine. It’s designed to build scalable network applications, and can execute JavaScript code outside of a browser.

Node.js works without an enclosing HTML page, instead using its own module system based on CommonJS, to put together multiple JavaScript files.



### React with Flux Architecture:

The components in Flux's architecture interact more like an EventBus and less like an MVC. As mentioned earlier, Flux is not actually a library or a framework, it is a new kind of architecture that Facebook created to work with React. Hence the main function of Flux is to complement React and promote Unidirectional Data Flow.



In a typical Flux architecture, you will find the following components:

Actions - Helpers that pass data to the Dispatcher.

Dispatcher - Receives these Actions and broadcasts payloads to registered callbacks.

Stores - Act as containers for application state and logic. The real work in the application is done in the Stores. The Stores are registered to listen in on the actions of the Dispatcher and update the Views according to these actions.

Controller Views - React Components grab the state from the stores and then pass it down to the child components.

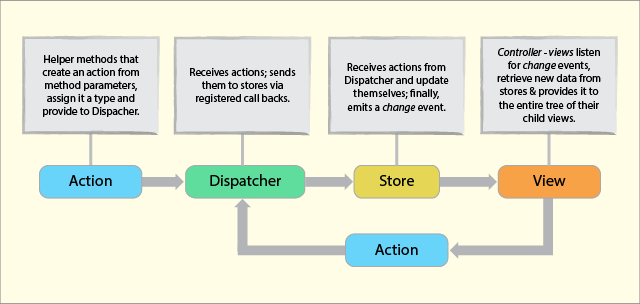
The Controllers in the MVC and Flux are different. In Flux, the Controllers are Controller-Views and are found at the very top of the hierarchy. Views are React components. All the functionality is usually found in the Store. The Store is where all the work is done and tells the Dispatcher which events/actions it is listening for.

When an event happens, the Dispatcher would send the “payload” to the Store that is registered to listen for that particular action. Now it is up to the Store to update the View, which in turn triggers an action. The action to that will occur is also predetermined, like name, the type of action, and so on.

The View propagates the Action through a central Dispatcher and this will be sent to various Stores. These Stores contain an application’s business logic and other data. It updates all the Views. It works best with React’s programming style and the Store sends updates without the need to provided detailed code on how to transition views between states.

This proves that the Flux pattern follows a unidirectional data flow. The Action, Dispatcher, Store, and View are independent nodes with specific inputs and outputs. The data flows through the Dispatcher, the central hub, which in turn manages all the data. The Dispatcher acts as a registry with registered callbacks that the Stores respond to. Stores will emit a change which will be picked by the Controller-Views.

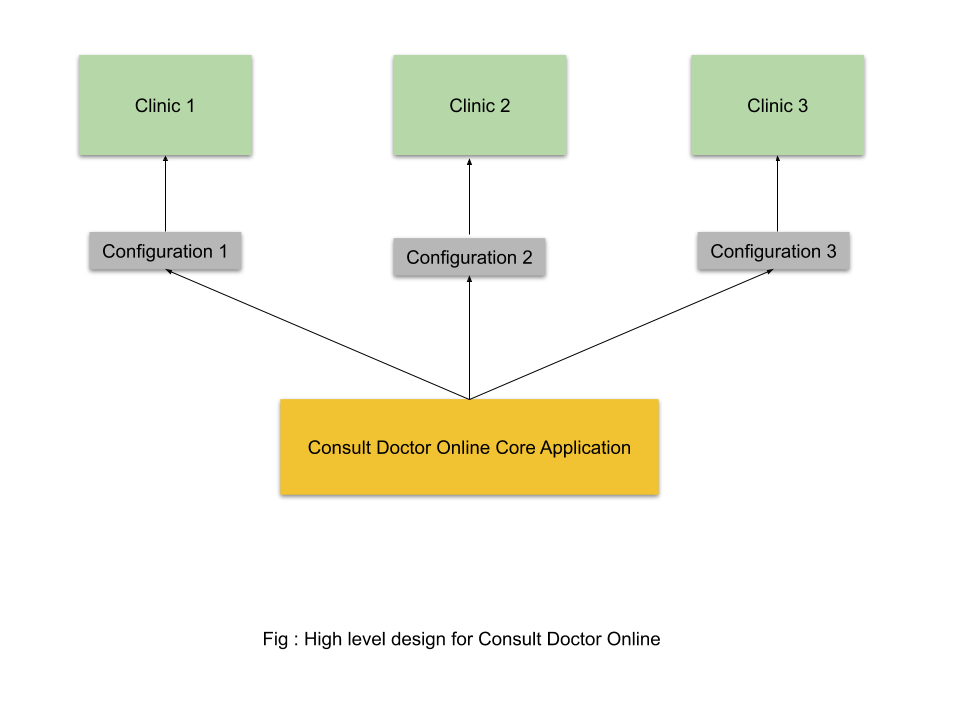
This is what happens when a View is propagated in the system:



This proves that there are no two-way bindings, the structure is akin to functional relative programming, and more something like flow-based programming. The dependencies that occur in the stores are kept in a strict hierarchy, while the Dispatcher handles the updates. This structure also solves the problems that come naturally with two-way binding. To create a Dispatcher, you need to bring the Dispatcher from the Flux. You can handle this by using libraries such as Dispatcher.js.

### High level Design of Applicaiton

Main Intention behind this application is to develop a single application which can be configured for a particular clinic. In this way we can save time and effort to develop application for an individual Clinic or facility.



Configurations can be any JSON file which can load the doctor information. Deployments can be taken care in such a way that each clinic can own an application as if it is developed for them. In this way we can take care of security.

## DESIGN RATIONALE

### Benefits of React JavaScript Library:

#### **Better User Experience**

Unlike other JavaScript frameworks, React uses the Virtual DOM – the abstract form of Real DOM. This makes it easier for the React app developers to update changes performed by the users in the application without affecting the other parts of the interface.

This results in building a highly dynamic UI with the exquisite user experience.

#### **Time-Saving**

In the case of React, the app development companies can reuse the code components at distinct levels at any point of time. Besides, the components are isolated to each other and changes in one does not affect the other, which makes it easier to manage the updates.

This makes the React app development easier, time-saving, and efficient for developers.

#### **Quick Development**

React allows the developers to reuse the existing code and apply hot reloading into the process. This approach not only improves the app performance but also accelerates the development speed.

#### **Faster Testing**

React extensively uses Redux which cut down the hassle of storing and managing component states in large-sized and complex applications with enormous dynamic elements.

It helps the developers to add application state in a single object and empower every component of the app to access the application state without involving child components or using callback. This makes it easier to test the application and log data changes, along with the use of hot reloading and other such tools.

#### **Code Stability with One-directional data binding**

ReactJS let the developers work directly with the components and employ downward data binding to ensure that the parent entities do not get affected by the changes of child entities. This approach makes the code stable and supports the idea of development in the future.

### Benefits of Node.js

#### **Node.js offers an Easy Scalability**

One of the key advantages of Node.js is that developers find it easy to scale the applications in horizontal as well as the vertical directions. The applications can be scaled in horizontal manner by the addition of additional nodes to the existing system.

Moreover, Node.js also offers you the option of adding extra resources to the single nodes during the vertical scaling of the application. So, it is highly scalable and provides better option than other JavaScript servers.

#### **Easy to Learn**

Since JavaScript is one of the most popular programming languages, most of the front-end developers have a good grasp over it.

It becomes much easier for them to start using the Node.js at the backend. It is easier to learn Node.js and consumes less time to work with it.

#### **Node.js is used as a Single Programming Language**

Node.js offers the developers the luxury of writing the server-side applications in the JavaScript. This allows the [Node.js developers](https://www.mindinventory.com/hire-node-js-developers.php) to write both the front-end as well as the back-end web application in JavaScript using a runtime environment.

And they don’t need to use any other server-side programming language. It also makes the deployment of the web applications simpler because almost all the web browsers support JavaScript.

#### **The Benefit of Fullstack JS**

Node.js has been regarded as a full-stack JavaScript for serving both the client and the server-side applications.

Therefore, the advantage is that you don’t have to hire separate developers for backend as well as the front-end development. It saves both your valuable money and time.

#### **Known for Offering High Performance**

It has been mentioned earlier that Node.js interprets the JavaScript code via Google’s V8 JavaScript engine. This engine complies the JavaScript code directly into the machine code. This makes it easier and faster to implement the code in a effective manner.

The speed of the code execution also enhanced by runtime environment as it supports the non-blocking I/O operations.

#### **The Support of Large and Active Community**

Node.js is blessed to have a large and active community of developers who keep on continuously contributing towards its further development and improvement.

In fact, the groups of developers are well supported by the JavaScript programmers providing ready-made and easy solutions and codes in GitHub. It is expected that the developers will initiate many further developers in the future.

#### **The Advantage of Caching**

The open-source runtime environment of the Node.js also provides the facility of caching single modules. Whenever there is any request for the first module, it gets cached in the application memory.

The developers don’t have to re-execute the codes as caching allows applications to load the web pages faster and responds more swiftly to the user.

#### **Offers the Freedom to Develop Apps**

Another advantage that Node.js offers to the developers is the freedom to develop the apps and software.

This is one essential feature, which remains absent in Ruby on Rails imposing certain guidelines. You can begin everything from the scratch while developing applications.

#### **Getting Support for Commonly Used Tools**

With Node.js, the developers can get an extended support for the various commonly used tools. Let’s take an example. Suppose, you want to test the source code of Node.js application; you can do so by using the Jasmin and other such unit-testing tools.

Similarly, if you want to identify and install the project dependencies, you can make use of npm, a powerful package manager. You can use grunt for task running of the project.

#### **Handles the Requests Simultaneously**

Since the Node.js is providing the option of non-blocking I/O systems, it relatively helps you to process several requests concurrently.

The system can handle the concurrent request handling efficiently better than others including Ruby or Python. The incoming requests get lined up and are executed quickly and systematically.

#### **Node.js is Highly Extensible**

The Node.js is known to be highly extensible, which means that you can customize and further extend Node.js as per their requirements.

You can also make use of JSON to provide the scope for exchange of data between the web server and the client. It also is facilitated with built-in APIs for developing HTTP, TCP, and DNS etc. servers.

## Sequence diagrams

This application provides a responsive web design. Responsive web design is an approach whereby a designer creates a web page that “responds to” or resizes itself depending on the type of device it is being seen through.  That could be an oversized desktop computer monitor, a laptop or devices with small screens such as [smartphones](https://smallbiztrends.com/2018/05/get-ready-for-the-mobile-first-index.html) and tablets.

Responsive Web design has become an essential tool for anyone with a digital presence. With the growth of smartphones, tablets and other mobile computing devices, more people are using smaller-screens to view web pages.

Let’s take a traditional “fixed” website.  When viewed on a desktop computer, for instance, the website might show three columns. But when you view that same layout on a smaller tablet, it might force you to scroll horizontally, something users don’t like. Or elements might be hidden from view or look distorted.  The impact is also complicated by the fact that many tablets can be viewed either in portrait orientation or turned sideways for landscape view.

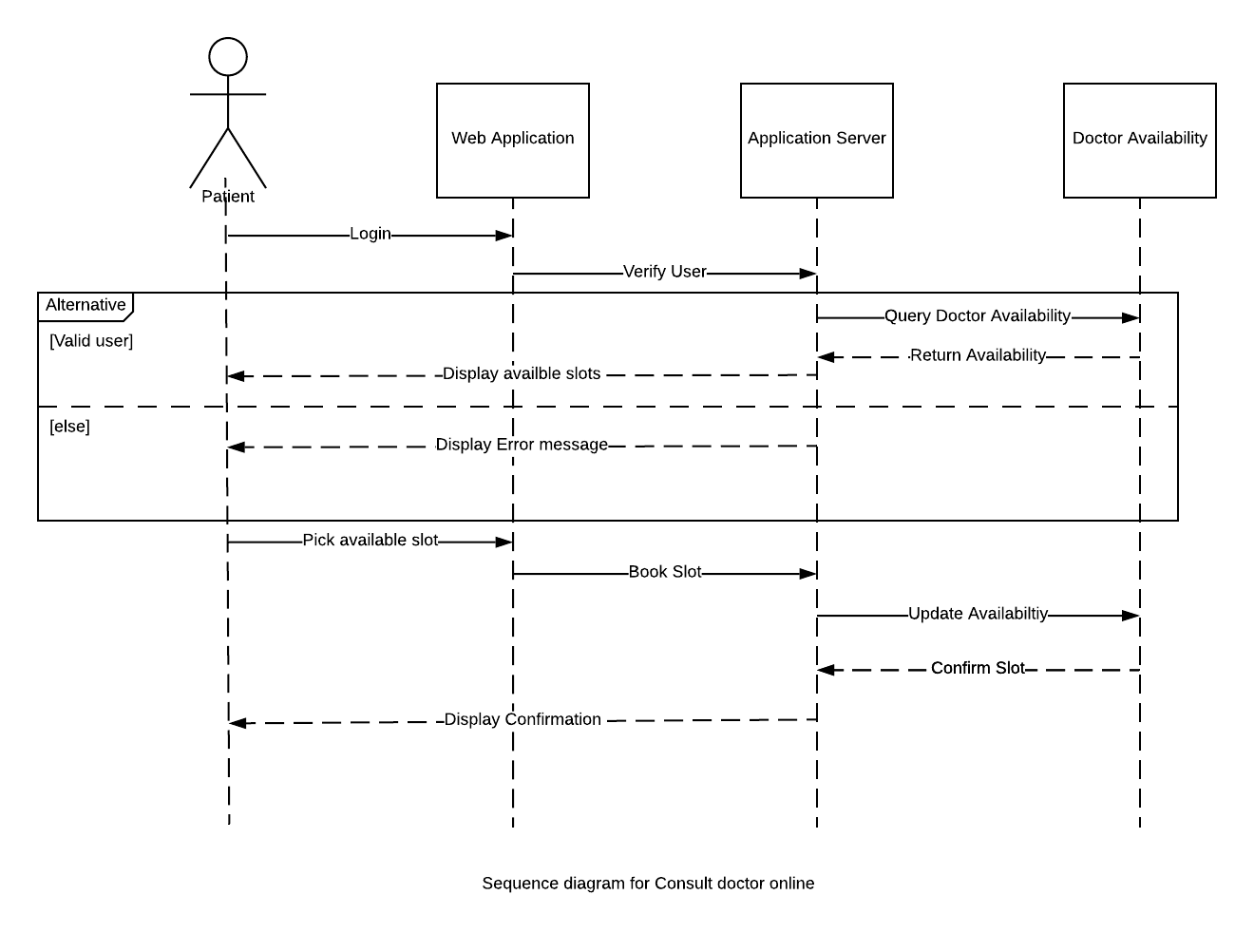
On a tiny smartphone screen, websites can be even more challenging to see. Large images may “break” the layout. Sites can be slow to load on smartphones if they are graphics heavy.

However, if a site uses responsive design, the tablet version might automatically adjust to display just two columns. That way, the content is readable and easy to navigate. On a smartphone, the content might appear as a single column, perhaps stacked vertically.  Or possibly the user would have the ability to swipe over to view other columns.  Images will resize instead of distorting the layout or getting cut off.

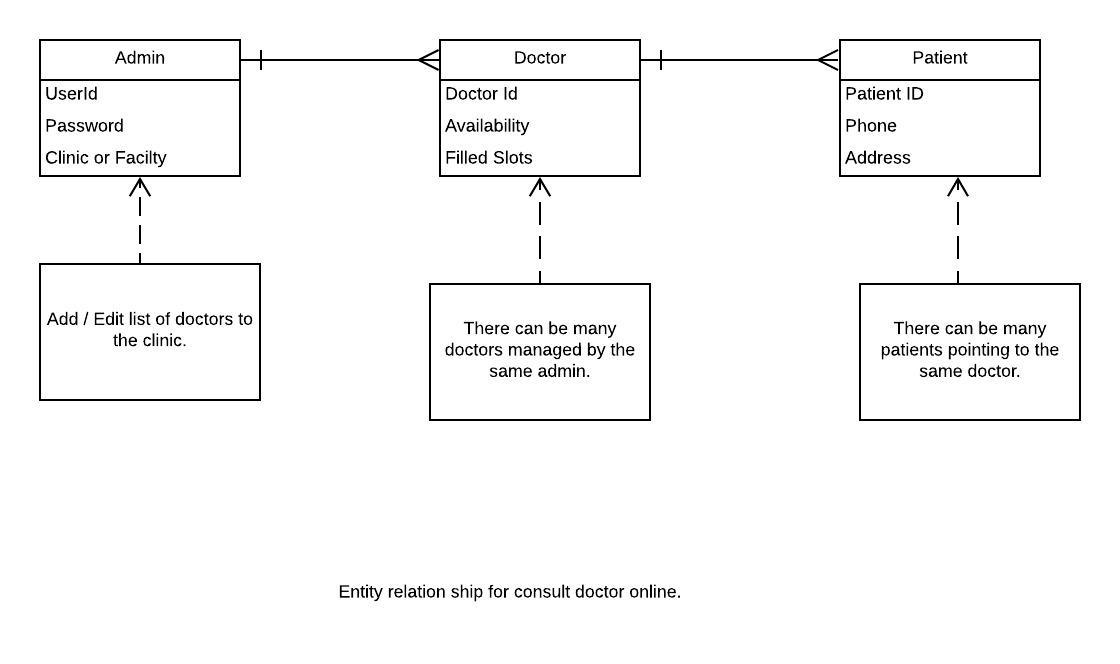
The point is: with responsive design, the website automatically adjusts based on the device the viewer sees it in.

**User interacts with the system**

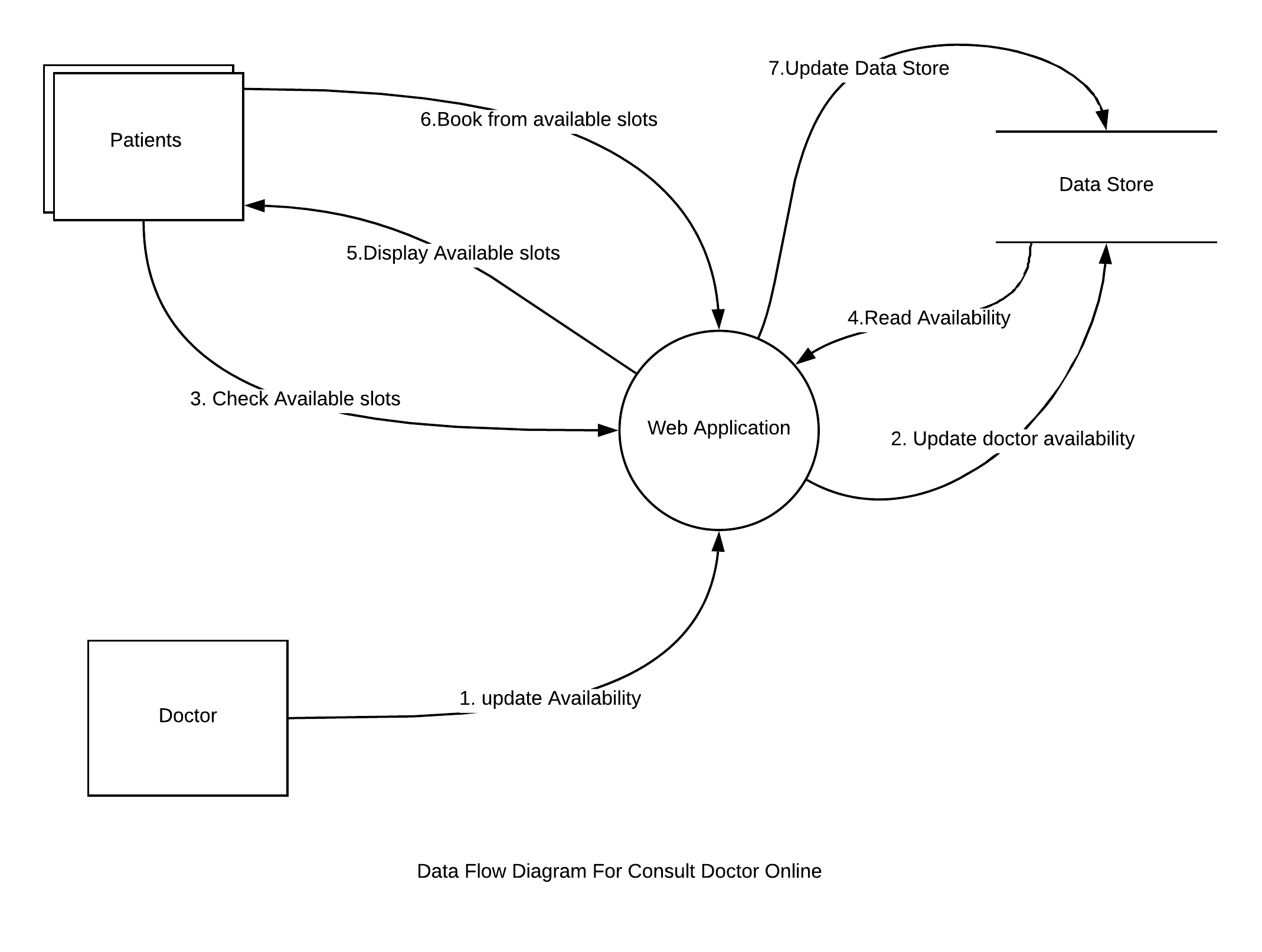
The sequence diagram shows how the user can book appointment with doctor online.



## Entity-Relationship diagram



## Data Flow diagram



# Implementation of System

## HOME PAGE

import React, { useEffect, useState } from "react";

import { withRouter } from "react-router";

import { useDispatch, useSelector } from "react-redux";

import \* as types from "../actions/action\_types";

import {

AppHeader,

AppFooter,

PageLayout,

TwoColumnLayout,

DoctorPhoto,

DoctorProfile

} from "../app\_components";

import LoginLayer from "./LoginLayer";

const HomeScreenContent = withRouter(props => {

const { history, doctorList } = props;

const bookAppointment = doctorId => {

history.push(`/BookAppointment/${doctorId}`);

};

return (

<React.Fragment>

{doctorList.map(doctorData => (

<TwoColumnLayout

dataTestId="doctor-homescreen"

key={doctorData.id}

leftColumn={

<DoctorPhoto

doctorData={doctorData}

bookAppointment={bookAppointment}

isButtonRequired={true}

/>

}

rightColumn={<DoctorProfile doctorData={doctorData} />}

/>

))}

</React.Fragment>

);

});

const HomeScreen = props => {

const doctorList = useSelector(state => state["doctorReducer"].doctorList);

const dispatch = useDispatch();

const [showLogin, setShowLogin] = useState(false);

useEffect(() => {

fetch("/api/app")

.then(resp => resp.json())

.then(resp => {

dispatch({ type: types.SET\_DOCTOR\_DETAILS, doctorList: resp.data });

});

}, [dispatch]);

const loginHandler = () => {

setShowLogin(true);

};

const onClose = () => {

setShowLogin(false);

};

const onSubmit = () => {

props.history.push("/PatientHomeScreen");

};

return (

<React.Fragment>

<PageLayout

header={

<AppHeader

actionTitle="Login"

isActionRequired={true}

actionHandler={loginHandler}

title="Consult Doctor Online"

/>

}

content={<HomeScreenContent doctorList={doctorList} />}

footer={<AppFooter />}

/>

{showLogin && <LoginLayer onClose={onClose} onSubmit={onSubmit} />}

</React.Fragment>

);

};

export default HomeScreen;

## BOOK APPOINTMENT SCREEN

import React from "react";

import { Grommet } from "grommet";

import AppTheme from "../css/app\_theme";

import { useSelector } from "react-redux";

import { withRouter } from "react-router";

import {

AppHeader,

AppFooter,

AppointmentGrid,

PageLayout,

TwoColumnLayout,

DoctorPhoto

} from "../app\_components";

const AppointmentContent = withRouter(({ doctorId }) => {

const doctorList = useSelector(state => state["doctorReducer"].doctorList);

const filterArray = doctorList.filter(obj => obj.id === Number(doctorId));

const doctorData = filterArray[0];

return (

<React.Fragment>

<TwoColumnLayout

dataTestId="book-appscreen"

leftColumn={<DoctorPhoto doctorData={doctorData} />}

rightColumn={<AppointmentGrid />}

/>

</React.Fragment>

);

});

function BookAppointmentScreen(props) {

const { match } = props;

const handleCancel = () => {

props.history.push("/");

};

return (

<Grommet theme={AppTheme}>

<PageLayout

header={

<AppHeader

title="Book Appointment"

isActionRequired={true}

actionTitle="Cancel"

actionHandler={handleCancel}

/>

}

content={<AppointmentContent doctorId={match.params.doctorId} />}

footer={<AppFooter />}

/>

</Grommet>

);

}

export default BookAppointmentScreen;

## LOGIN SCREEN

import React, { useState } from "react";

import {

MelodyBox,

MelodyHeading,

MelodyButton,

MelodyLayer,

MelodyFormField,

MelodyTextInput

} from "../basic\_components";

import { Link } from "react-router-dom";

import { AppNotification } from "../app\_components";

import { Close } from "grommet-icons";

const LoginLayer = ({ onClose, onSubmit }) => {

const [userName, setUserName] = useState("");

const [password, setPassword] = useState("");

const [showMsg, setShowMsg] = useState(false);

const [notifMsg, setNotifMsg] = useState("");

const closeMsg = () => {

setShowMsg(false);

};

const preSubmit = (userId, pwd) => {

if (userId.length === 0 || pwd.length === 0) {

//show notification

setNotifMsg("username or password can't be empty!");

setShowMsg(true);

} else {

onSubmit(userId, pwd);

}

};

return (

<React.Fragment>

<MelodyLayer

position="right"

full="vertical"

modal

onClickOutside={() => onClose()}

onEsc={() => onClose()}

>

<MelodyBox fill="vertical" overflow="auto" width="medium" pad="medium">

<MelodyBox flex={false} direction="row" justify="between">

<MelodyHeading level={2} margin="none">

Login

</MelodyHeading>

<MelodyButton icon={<Close />} onClick={() => onClose()} />

</MelodyBox>

<MelodyBox flex="grow" overflow="auto" pad={{ vertical: "medium" }}>

<MelodyFormField label="User Name">

<MelodyTextInput onInput={e => setUserName(e.target.value)} />

</MelodyFormField>

<MelodyFormField label="Password">

<MelodyTextInput

type="password"

onInput={e => setPassword(e.target.value)}

/>

</MelodyFormField>

<MelodyButton

type="submit"

label="Submit"

onClick={() => preSubmit(userName, password)}

primary

/>

</MelodyBox>

</MelodyBox>

<MelodyBox flex={false} as="footer" align="start" pad="medium">

<MelodyBox>

<Link to="/register">New User? Register Now.</Link>

</MelodyBox>

</MelodyBox>

</MelodyLayer>

{showMsg && <AppNotification text={notifMsg} onClose={closeMsg} />}

</React.Fragment>

);

};

export default LoginLayer;

## PATIENT HOME SCREEN

import React, { useState } from "react";

import { MelodyText, MelodyBox, MelodyButton } from "../basic\_components";

import { MenuLayout, AppFooter } from "../app\_components";

import ProfileScreen from "./ProfileScreen";

import HistoryScreen from "./HistoryScreen";

import BookAppointmentScreen from "./BookAppointmentScreen";

import { AppHeader } from "../app\_components";

import { withRouter } from "react-router";

import { Route } from "react-router-dom";

//sidebar

const Sidebar = withRouter(props => {

const [menuSelected, setMenuSelected] = useState(0);

const menuItemHandler = name => {

let route = "/";

switch (name) {

case "Book Appointment":

route = "BookAppointment";

setMenuSelected(0);

break;

case "Check History":

route = "CheckHistory";

setMenuSelected(1);

break;

case "Edit Profile":

route = "EditProfile";

setMenuSelected(2);

break;

default:

route = "/";

}

props.history.push(`/PatientHomeScreen/${route}`);

};

return (

<React.Fragment>

{["Book Appointment", "Check History", "Edit Profile"].map(

(name, index) => (

<MelodyButton

key={name}

hoverIndicator

onClick={() => menuItemHandler(name)}

>

<MelodyBox

elevation="medium"

pad={{ horizontal: "medium", vertical: "small" }}

background={index === menuSelected ? "#d1215c" : undefined}

>

<MelodyText>{name}</MelodyText>

</MelodyBox>

</MelodyButton>

)

)}

</React.Fragment>

);

});

//content

const Content = () => {

return (

<React.Fragment>

<Route

path={`/PatientHomeScreen/BookAppointment`}

component={BookAppointmentScreen}

/>

<Route

path={`/PatientHomeScreen/CheckHistory`}

component={HistoryScreen}

/>

<Route

path={`/PatientHomeScreen/EditProfile`}

component={ProfileScreen}

/>

</React.Fragment>

);

};

function PatientHomeScreen(props) {

const logoutHandler = () => {

props.history.push("/");

};

return (

<MenuLayout

header={

<AppHeader

title="Welcome John!!"

actionTitle="Logout"

isActionRequired="true"

actionHandler={logoutHandler}

/>

}

sidebar={<Sidebar />}

content={<Content />}

footer={<AppFooter />}

/>

);

}

export default PatientHomeScreen;

## APPOINTMENT DETAILS

//This screen will help to register a new user.

//Takes all user details.

import React, { useState } from "react";

import { withRouter } from "react-router";

import { useSelector } from "react-redux";

import { PageLayout, AppHeader, AppFooter } from "../app\_components";

import {

MelodyButton,

MelodyForm,

MelodyFormField,

MelodyBox,

MelodyHeading,

MelodyText

} from "../basic\_components";

import { AppNotification } from "../app\_components";

const RegisterForm = withRouter(props => {

const appointmentDetails = useSelector(

state => state["appointmentReducer"].appointment

);

const onClose = () => {

props.history.push("/");

};

const bookAppointment = () => {

setShowMsg(true);

};

let [showMsg, setShowMsg] = useState(false);

return (

<MelodyBox align="center" justify="center">

<MelodyHeading>User Details</MelodyHeading>

<MelodyBox>

<MelodyBox gap="medium">

<MelodyText weight="bold">{`Date of Appointment :${

appointmentDetails.selDate

} `}</MelodyText>

<MelodyText weight="bold">{`Time of Appointment :${

appointmentDetails.selTime

} `}</MelodyText>

</MelodyBox>

<MelodyBox width="medium" margin={{ top: "medium" }}>

<MelodyForm>

<MelodyFormField label="Name" name="name" required={true} />

<MelodyFormField label="Mobile No" name="mobileNo" />

<MelodyFormField label="Mail Id" name="mailId" />

<MelodyFormField label="Problem Description" name="pr-descr" />

<MelodyFormField label="Tests Done" name="test-done" />

<MelodyButton label="Book Appointment" onClick={bookAppointment} />

</MelodyForm>

</MelodyBox>

</MelodyBox>

{showMsg && (

<AppNotification

text="You reserved the slot successfully"

onClose={onClose}

/>

)}

</MelodyBox>

);

});

/\*

This screen follows page Layout.

Uses AppHeader and AppFooter along with the component.

\*/

function AppointmentDetails(props) {

const handleCancel = () => {

props.history.push("/");

};

return (

<React.Fragment>

<PageLayout

header={

<AppHeader

actionTitle="Cancel"

isActionRequired={true}

actionHandler={handleCancel}

title="Consult Doctor Online"

/>

}

content={<RegisterForm />}

footer={<AppFooter />}

/>

</React.Fragment>

);

}

export default AppointmentDetails;

## REGISTER A NEW USER

//This screen will help to register a new user.

//Takes all user details.

import React from "react";

import { PageLayout, AppHeader, AppFooter } from "../app\_components";

import {

MelodyButton,

MelodyForm,

MelodyFormField,

MelodyBox,

MelodyHeading

} from "../basic\_components";

import { withRouter } from "react-router";

const RegisterForm = withRouter(props => {

const registerUser = () => {

props.history.push("/");

};

return (

<MelodyBox align="center" justify="center">

<MelodyHeading>Register User</MelodyHeading>

<MelodyBox>

<MelodyBox width="medium">

<MelodyForm>

<MelodyFormField label="Name" name="name" required={true} />

<MelodyFormField label="Mobile No" name="mobileNo" />

<MelodyFormField label="Mail Id" name="mailId" />

<MelodyFormField label="Password" name="password" type="password" />

<MelodyFormField

label="Confirm Password"

name="confirmPassword"

type="password"

/>

<MelodyFormField label="Line1" name="addr-line1" />

<MelodyFormField label="Line2" name="addr-line2" />

<MelodyFormField label="City" name="city" />

<MelodyFormField label="State" name="state" />

<MelodyFormField label="ZipCode" name="zip-code" />

<MelodyButton label="Register" onClick={registerUser} />

</MelodyForm>

</MelodyBox>

</MelodyBox>

</MelodyBox>

);

});

/\*

This screen follows page Layout.

Uses AppHeader and AppFooter along with the component.

\*/

function RegisterScreen(props) {

const handleCancel = () => {

props.history.push("/");

};

return (

<React.Fragment>

<PageLayout

header={

<AppHeader

actionTitle="Cancel"

isActionRequired={true}

actionHandler={handleCancel}

title="Consult Doctor Online"

/>

}

content={<RegisterForm />}

footer={<AppFooter />}

/>

</React.Fragment>

);

}

export default RegisterScreen;

## HISTORY OF THE APPOINTMENTS

import React from "react";

import {

MelodyHeading,

MelodyBox,

MelodyDataTable,

MelodyText,

MelodyButton

} from "../basic\_components";

import { dateWithMonth } from "../utils/date-utils";

const timeNow = new Date().getTime();

let dayInMS = 24 \* 60 \* 60 \* 1000;

const dateAddition = days => dateWithMonth(new Date(timeNow + days \* dayInMS));

let futureDates = [dateAddition(1), dateAddition(2), dateAddition(3)];

let pastDates = [dateAddition(-3), dateAddition(-2), dateAddition(-1)];

const futureAppointmentHeadings = [

"Docotor Name",

"Time Slot",

"Cancel Appointment"

];

const pastAppointmentHeadings = ["Docotor Name", "Time Slot", "Status"];

const futureData = [

{

doctorName: "Krishna Prasad",

timeSlot: `${futureDates[0]} 1:30 to 1:45PM`,

cancel: { label: "Cancel", handler: () => {} }

},

{

doctorName: "Aruna",

timeSlot: `${futureDates[1]} 2:00 to 2:15PM`,

cancel: { label: "Cancel", handler: () => {} }

},

{

doctorName: "Krishna Prasad",

timeSlot: `${futureDates[2]} 2:30 to 2:45PM`,

cancel: { label: "Cancel", handler: () => {} }

},

{

doctorName: "Aruna",

timeSlot: `${futureDates[2]} 2:45 to 3:00PM`,

cancel: { label: "Cancel", handler: () => {} }

}

];

const pastData = [

{

doctorName: "Krishna Prasad",

timeSlot: `${pastDates[0]} 2:30 to 2:45PM`,

cancel: { label: "Attended", isPast: true }

},

{

doctorName: "Aruna",

timeSlot: `${pastDates[1]} 2:45 to 3:00PM`,

cancel: { label: "Attended", isPast: true }

},

{

doctorName: "Krishna Prasad",

timeSlot: `${pastDates[2]} 1:30 to 1:45PM`,

cancel: { label: "Attended", isPast: true }

},

{

doctorName: "Aruna",

timeSlot: `${pastDates[2]} 1:45 to 2:00PM`,

cancel: { label: "Cancelled", isPast: true }

},

{

doctorName: "Aruna",

timeSlot: `${pastDates[1]} 2:45 to 3:00PM`,

cancel: { label: "Attended", isPast: true }

},

{

doctorName: "Krishna Prasad",

timeSlot: `${pastDates[2]} 1:30 to 1:45PM`,

cancel: { label: "Attended", isPast: true }

}

];

const AppointmentTable = ({ headings, data }) => {

return (

<MelodyDataTable

columns={[

{

property: "doctorName",

header: <MelodyText>{headings[0]}</MelodyText>,

render: data => <MelodyText>{data.doctorName}</MelodyText>

},

{

property: "timeSlot",

header: <MelodyText>{headings[1]}</MelodyText>,

render: data => <MelodyText>{data.timeSlot}</MelodyText>

},

{

property: "cancel",

header: <MelodyText>{headings[2]}</MelodyText>,

render: data => {

if (data.cancel.isPast) {

return <MelodyText>{data.cancel.label}</MelodyText>;

} else {

return (

<MelodyButton

label={data.cancel.label}

onClick={data.handleCancel}

/>

);

}

}

}

]}

data={data}

/>

);

};

function HistoryScreen() {

return (

<MelodyBox>

<MelodyHeading size="small">Future Appointments:</MelodyHeading>

<MelodyBox border={{ color: "gray" }} elevation="small" round="medium">

<AppointmentTable

headings={futureAppointmentHeadings}

data={futureData}

/>

</MelodyBox>

<MelodyHeading size="small">Past Appointments:</MelodyHeading>

<MelodyBox border={{ color: "gray" }} elevation="small" round="medium">

<AppointmentTable headings={pastAppointmentHeadings} data={pastData} />

</MelodyBox>

</MelodyBox>

);

}

export default HistoryScreen;

## Reusable Components

### Application Header

import React from "react";

import { MelodyHeading, MelodyBox, MelodyButton } from "../../basic\_components";

import PropTypes from "prop-types";

const AppHeader = ({ isActionRequired, title, actionTitle, actionHandler }) => {

return (

<MelodyBox

direction="row"

flex="true"

border={{ side: "bottom", color: "gray" }}

>

<MelodyBox align="center" justify="center" flex={{ grow: 2 }}>

<MelodyHeading>{title}</MelodyHeading>

</MelodyBox>

{isActionRequired && (

<MelodyBox flex="true" align="center" justify="center">

<MelodyButton label={actionTitle} onClick={actionHandler} />

</MelodyBox>

)}

</MelodyBox>

);

};

AppHeader.propTypes = {

isActionRequired: PropTypes.bool,

title: PropTypes.string.isRequired

};

AppHeader.defaultProps = {

isActionRequired: false

};

export default AppHeader;

### Application Footer

import React from "react";

import { MelodyBox, MelodyText } from "../../basic\_components";

const AppFooter = () => {

return (

<MelodyBox gap="small" margin="small">

<MelodyText> Copyright © 2019-2020 Kalyani Payyavula </MelodyText>

<MelodyText>

Project submitted for partial fulfilment for the award of the degree.

</MelodyText>

<MelodyText> Contact : kalyanichow8@gmail.com </MelodyText>

</MelodyBox>

);

};

export default AppFooter;

### Menu Layout

import React from "react";

import { MelodyGrid, MelodyBox } from "../../basic\_components";

const MenuLayout = ({ header, sidebar, content, footer }) => {

return (

<>

<MelodyGrid

rows={["auto", "flex", "auto"]}

columns={["auto", "flex", "auto"]}

areas={[

{ name: "header", start: [0, 0], end: [1, 0] },

{ name: "sidebar", start: [0, 1], end: [0, 1] },

{ name: "main", start: [1, 1], end: [1, 1] },

{ name: "footer", start: [0, 2], end: [1, 2] }

]}

>

<MelodyBox

gridArea="header"

direction="row"

align="center"

justify="between"

background="gray"

>

{header}

</MelodyBox>

<MelodyBox

gridArea="sidebar"

background="gray"

width="small"

border={{ color: "dark-2" }}

>

{sidebar}

</MelodyBox>

<MelodyBox gridArea="main" pad="small">

{content}

</MelodyBox>

<MelodyBox as="footer" gridArea="footer" pad="small" background="black">

{footer}

</MelodyBox>

</MelodyGrid>

</>

);

};

export default MenuLayout;

### Page Layout

import React from "react";

import { MelodyBox } from "../../basic\_components";

import PropTypes from "prop-types";

const PageLayout = ({ header, content, footer, dataTestId }) => {

return (

<MelodyBox direction="column" data-testid={`${dataTestId}-page-layout`}>

<MelodyBox border={{ side: "bottom", color: "gray" }}>{header}</MelodyBox>

<MelodyBox pad="small" background="light-2">

{content}

</MelodyBox>

<MelodyBox background="gray">{footer}</MelodyBox>

</MelodyBox>

);

};

PageLayout.propTypes = {

header: PropTypes.node,

content: PropTypes.node,

footer: PropTypes.node,

dataTestId: PropTypes.string

};

export default PageLayout;

### Application Notification

import React from "react";

import {

MelodyBox,

MelodyButton,

MelodyText,

MelodyLayer

} from "../../basic\_components";

import { FormClose, StatusGood } from "grommet-icons";

import PropTypes from "prop-types";

const AppNotification = ({ text, onClose }) => {

return (

<MelodyLayer

position="top"

modal={false}

margin={{ vertical: "medium", horizontal: "small" }}

onEsc={() => onClose}

responsive={false}

plain

>

<MelodyBox

align="center"

direction="row"

gap="small"

justify="between"

round="medium"

elevation="medium"

pad={{ vertical: "xsmall", horizontal: "small" }}

background="status-ok"

>

<MelodyBox align="center" direction="row" gap="xsmall">

<StatusGood />

<MelodyText>{text}</MelodyText>

</MelodyBox>

<MelodyButton icon={<FormClose />} onClick={() => onClose()} plain />

</MelodyBox>

</MelodyLayer>

);

};

AppNotification.propTypes = {

text: PropTypes.string.isRequired

};

export default AppNotification;

### Appointment Grid

import React, { useState } from "react";

import { withRouter } from "react-router";

import { useDispatch } from "react-redux";

import {

MelodyBox,

MelodyButton,

MelodyHeading,

MelodyDataTable

} from "../../basic\_components";

import { dateWithMonth } from "../../utils/date-utils";

import { Next, Previous } from "grommet-icons";

import { AppNotification } from "../../app\_components";

import { ConfirmationDialog } from "../../app\_components";

import \* as types from "../../actions/action\_types";

const timeNow = new Date().getTime();

let dayInMS = 24 \* 60 \* 60 \* 1000;

const dateAddition = days => dateWithMonth(new Date(timeNow + days \* dayInMS));

let datesArray = [dateAddition(1), dateAddition(2), dateAddition(3)];

const SlotColumn = ({ slots, bookSlot, date }) => {

return (

<MelodyBox>

{slots.map(timeSlot => (

<MelodyButton

elevation="small"

label={timeSlot.slot}

color={timeSlot.booked ? "#d1215c" : undefined}

disable={timeSlot.booked}

onClick={() => bookSlot(timeSlot, date)}

/>

))}

</MelodyBox>

);

};

const AppointmentGrid = props => {

let [dateAt, setDateAt] = useState(3);

let [showMsg, setShowMsg] = useState(false);

let [notifMsg, setNotifMsg] = useState("");

let [getConfirmation, setConfirmation] = useState(false);

let [confirmationMsg, setConfirmationMsg] = useState("");

const dispatch = useDispatch();

const handleNextDate = () => {

datesArray = datesArray.map((elem, index) => {

return dateAddition(index + dateAt + 1);

});

setDateAt(dateAt + 3);

};

const handlePrevDate = () => {

datesArray = datesArray

.map((elem, index) => {

return dateAddition(dateAt - index);

})

.reverse();

setDateAt(dateAt - 3);

};

const onClose = () => {

setShowMsg(false);

};

const closeConfiramtion = () => {

setConfirmation(false);

};

const proceedToBooking = () => {

props.history.push("/AppointmentDetails");

};

const bookSlot = (timeSlot, date) => {

if (timeSlot.booked) {

//Inform user that slot is already booked

setNotifMsg("Slot is already booked!");

setShowMsg(true);

} else {

dispatch({

type: types.SET\_APPOINTMENT\_DETAILS,

appointmentDetails: { selDate: timeSlot.slot, selTime: date }

});

setConfirmationMsg(

`Are you sure you want to book ${timeSlot.slot} slot on ${date}?`

);

setConfirmation(true);

}

};

return (

<MelodyBox>

<MelodyBox align="center" justify="center">

<MelodyHeading size="small">Available Slots</MelodyHeading>

</MelodyBox>

<MelodyBox direction="row" justify="between" align="center">

<MelodyButton

label="Prev 3 days"

onClick={handlePrevDate}

icon={<Previous />}

/>

<MelodyButton

label="Next 3 days"

onClick={handleNextDate}

icon={<Next />}

reverse="true"

/>

</MelodyBox>

<MelodyDataTable

columns={[

{

property: "date",

render: data => (

<MelodyBox

align="center"

elevation="medium"

round="medium"

justify="center"

background="#d1215c"

fill="true"

>

<MelodyHeading>{data.date}</MelodyHeading>

</MelodyBox>

)

},

{

property: "slot1",

render: data => (

<SlotColumn

slots={data.slot1}

bookSlot={bookSlot}

date={data.date}

/>

)

},

{

property: "slot2",

render: data => (

<SlotColumn

slots={data.slot2}

bookSlot={bookSlot}

date={data.date}

/>

)

},

{

property: "slot3",

render: data => (

<SlotColumn

slots={data.slot3}

bookSlot={bookSlot}

date={data.date}

/>

)

}

]}

data={[

{

date: datesArray[0],

slot1: slot1,

slot2: slot2,

slot3: slot3

},

{

date: datesArray[1],

slot1: slot1,

slot2: slot2,

slot3: slot3

},

{

date: datesArray[2],

slot1: slot1,

slot2: slot2,

slot3: slot3

}

]}

/>

{showMsg && <AppNotification text={notifMsg} onClose={onClose} />}

{getConfirmation && (

<ConfirmationDialog

heading="Confirm"

text={confirmationMsg}

primaryButton="Proceed"

secondaryButton="Cancel"

onClose={closeConfiramtion}

primaryButtonAction={proceedToBooking}

/>

)}

</MelodyBox>

);

};

export default withRouter(AppointmentGrid);

## Server

### Express JS Interface

var createError = require("http-errors");

var express = require("express");

var path = require("path");

var cookieParser = require("cookie-parser");

var logger = require("morgan");

var indexRouter = require("./routes/index");

var appRouter = require("./routes/app");

var app = express();

// view engine setup

app.set("views", path.join(\_\_dirname, "views"));

app.set("view engine", "pug");

app.use(logger("dev"));

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

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

app.use("/api", indexRouter);

app.use("/api/app", appRouter);

// catch 404 and forward to error handler

app.use(function(req, res, next) {

next(createError(404));

});

// error handler

app.use(function(err, req, res, next) {

// set locals, only providing error in development

res.locals.message = err.message;

res.locals.error = req.app.get("env") === "development" ? err : {};

// render the error page

res.status(err.status || 500);

res.render("error");

});

module.exports = app;

### Home Screen Interface

var express = require("express");

var router = express.Router();

var payload = require("../models/app");

/\* GET users listing. \*/

router.get("/", function(req, res, next) {

try {

return res.json(payload);

} catch (e) {

console.log(e);

}

});

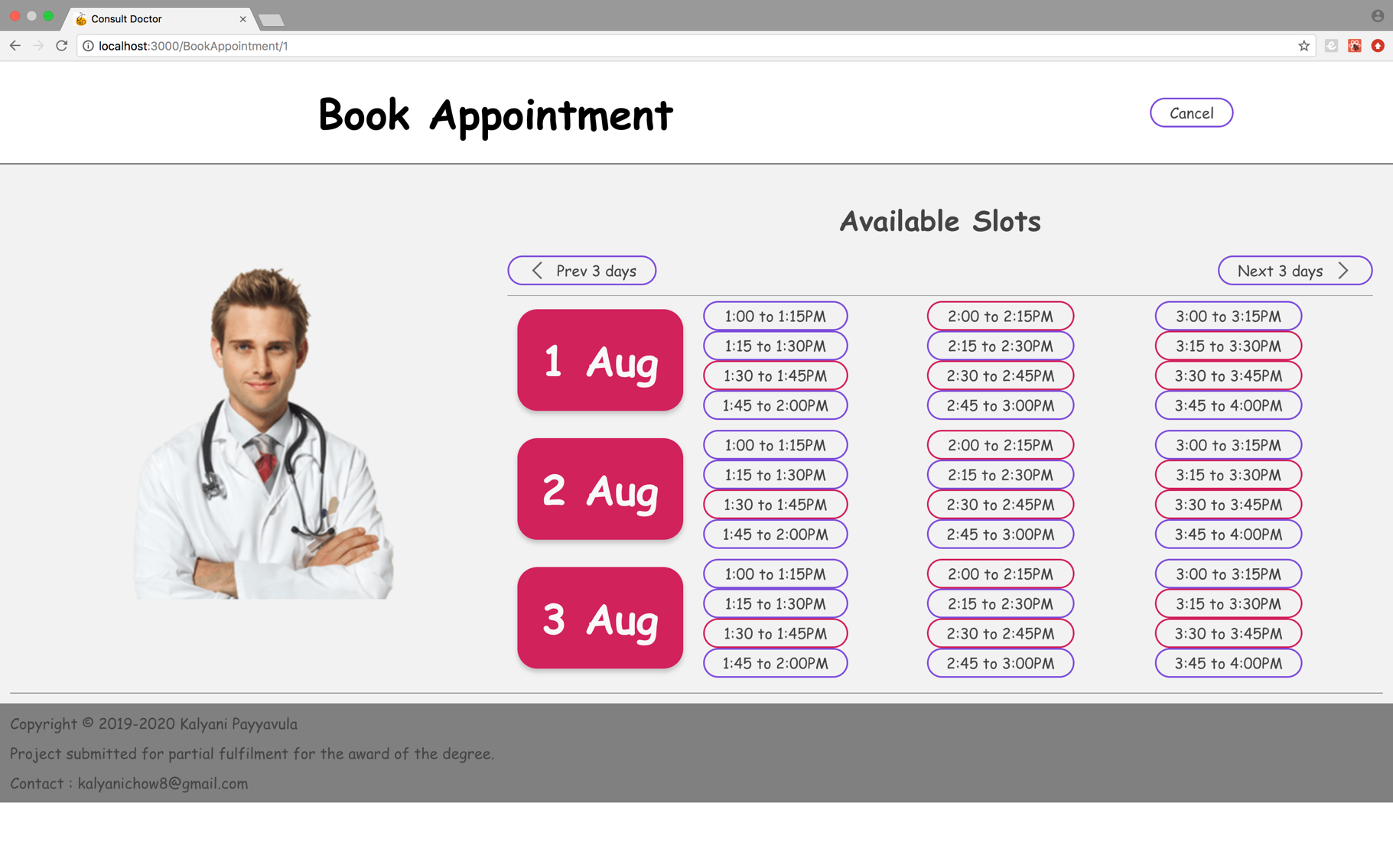
module.exports = router;

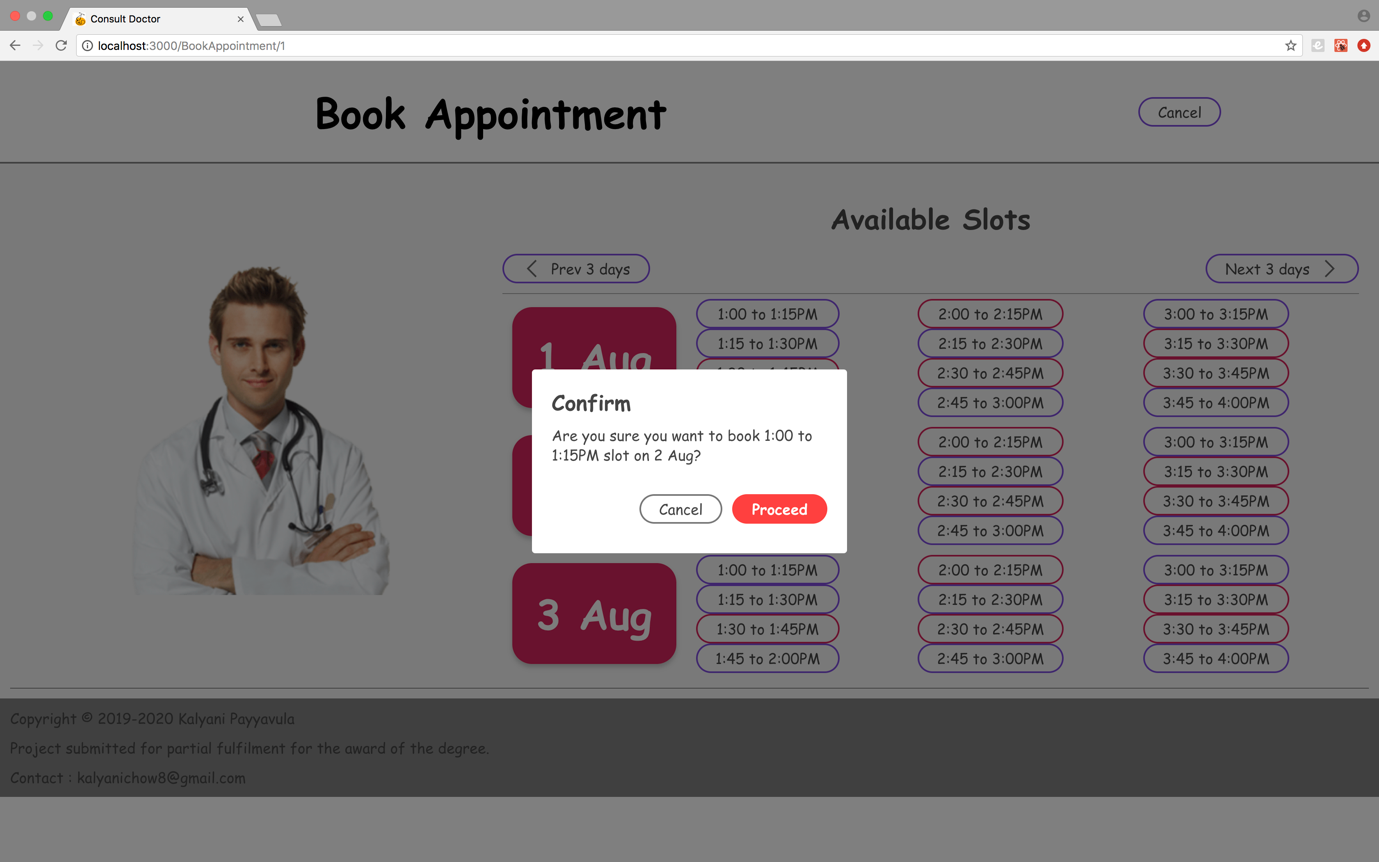
## Screen Shots

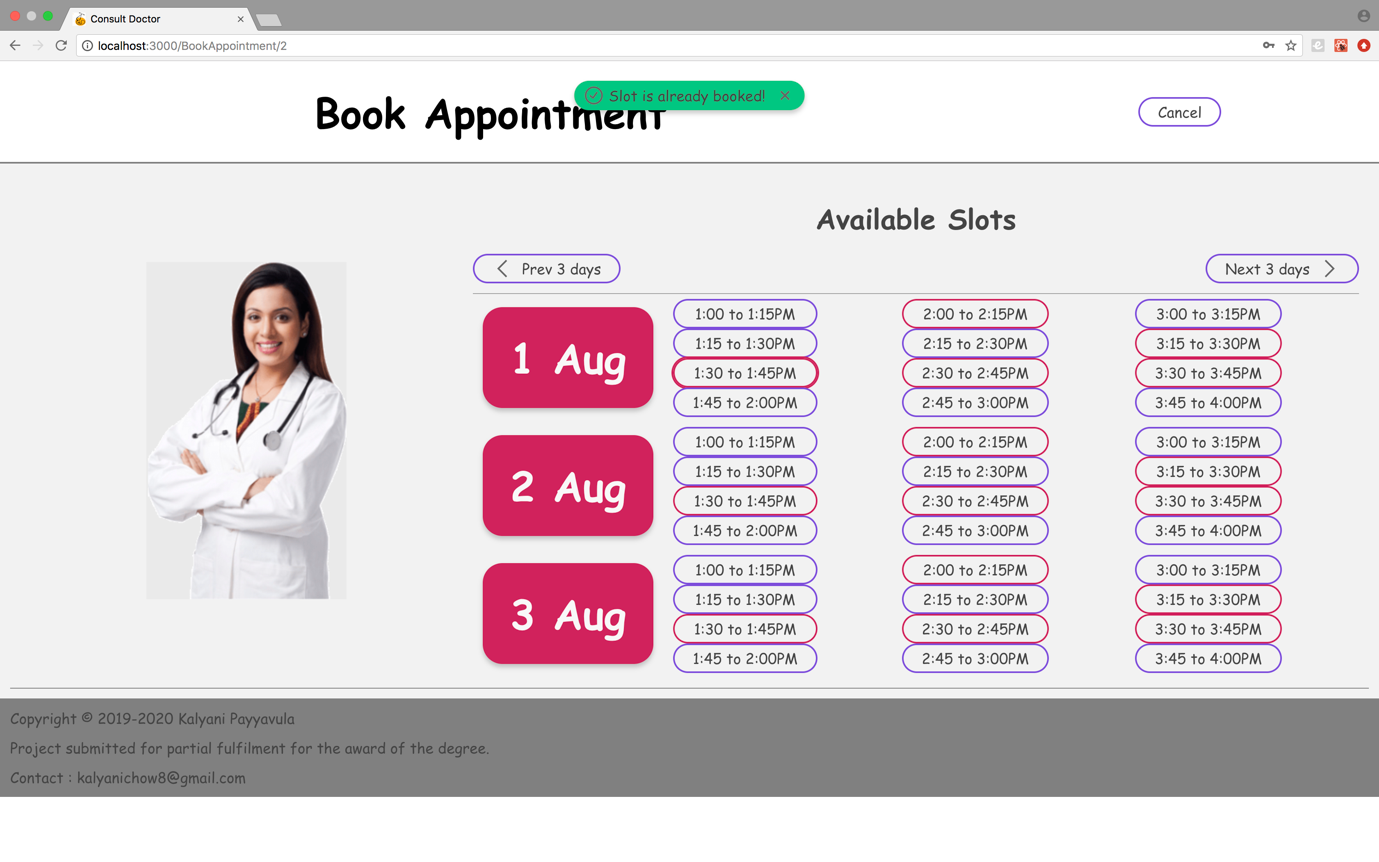
### Home Page



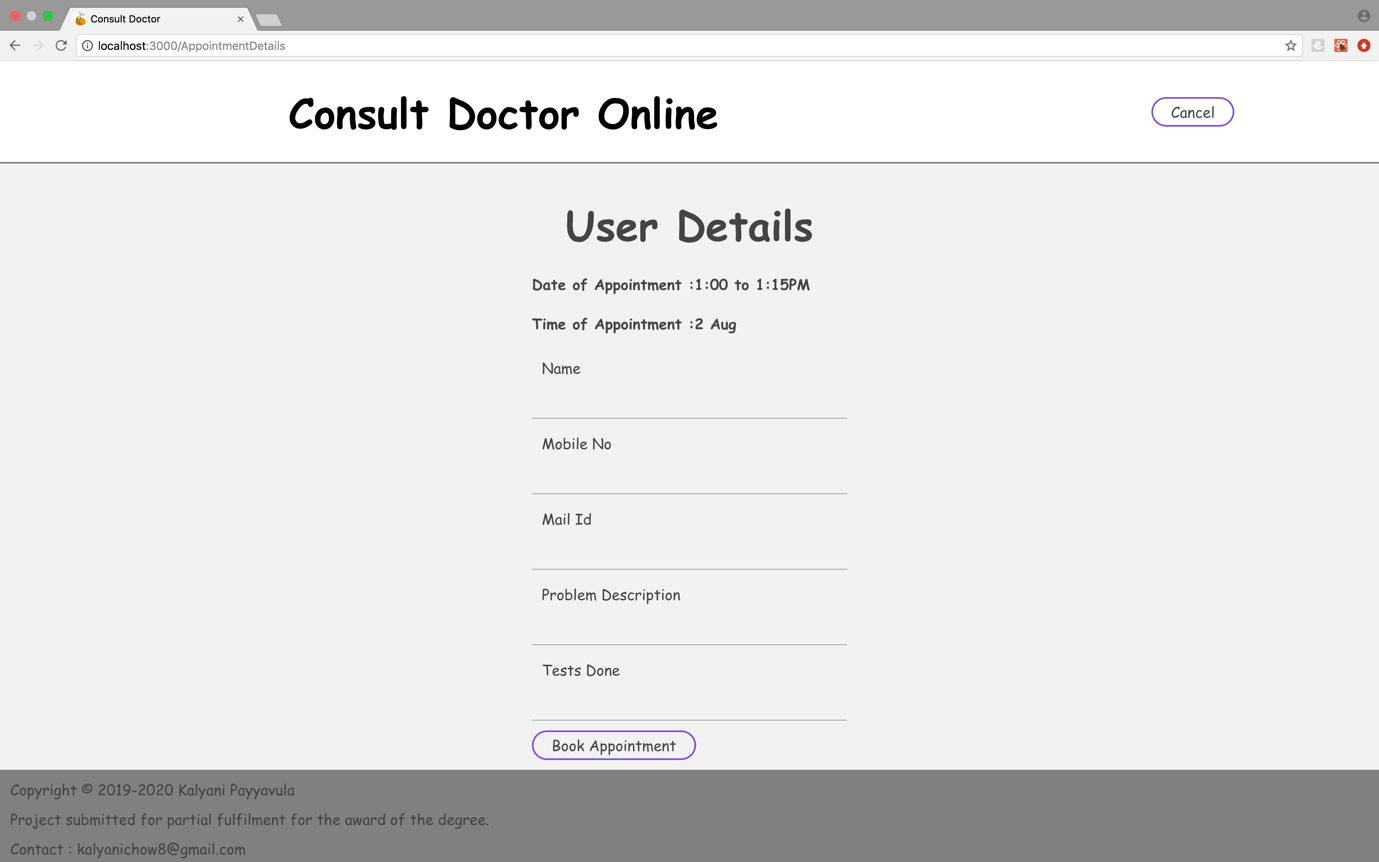
### Book Appointment Page



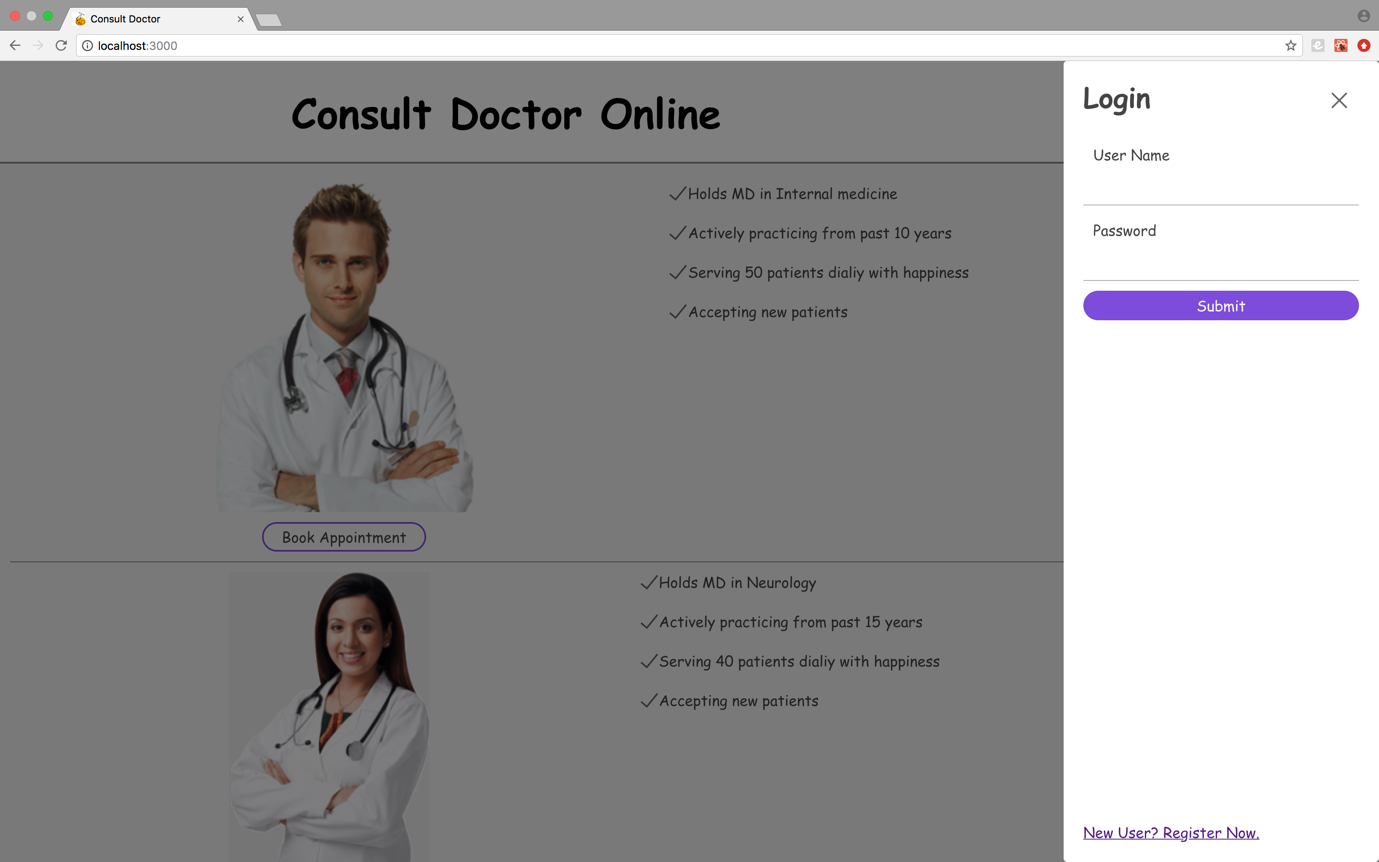




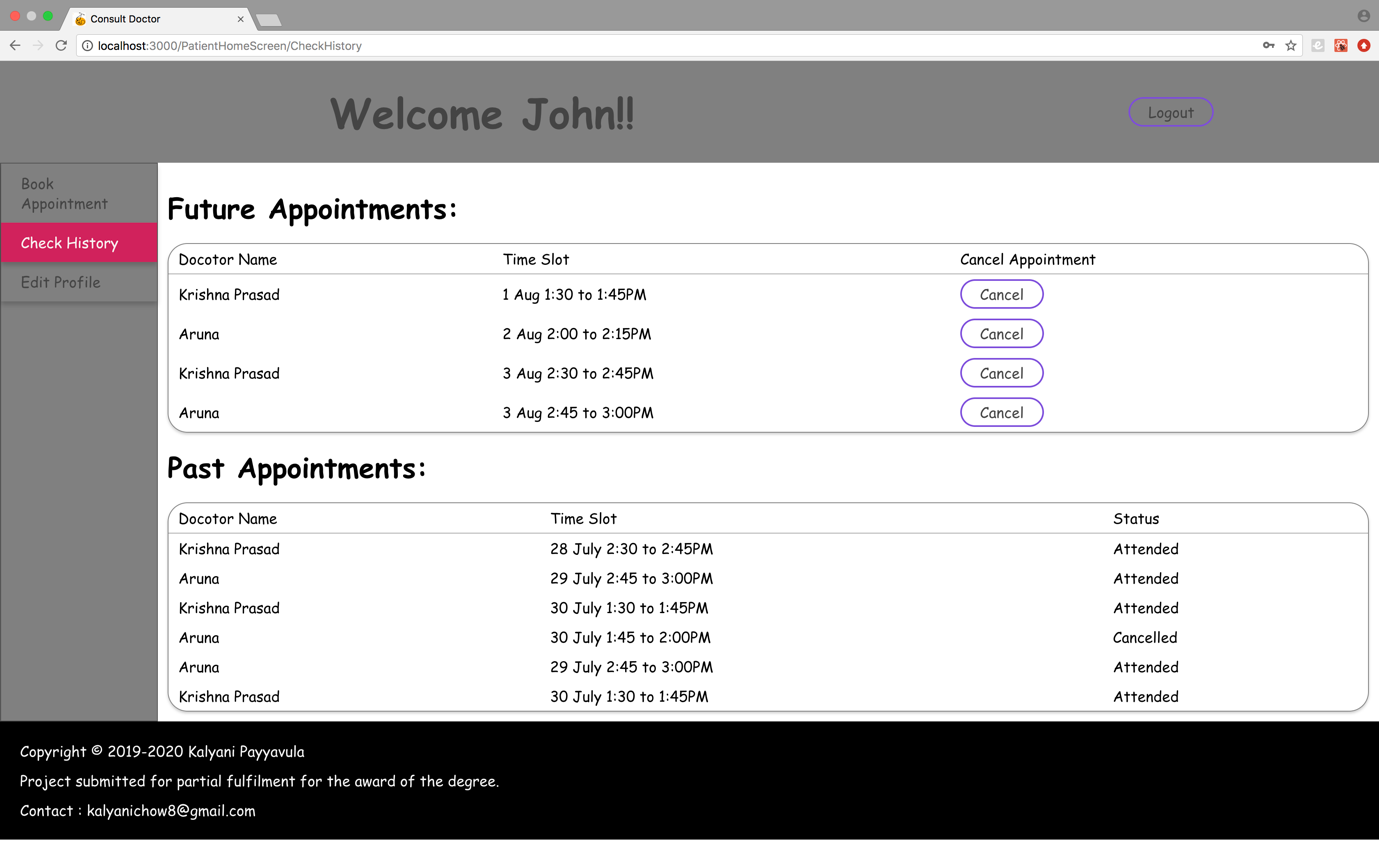
### Appointment Details Page



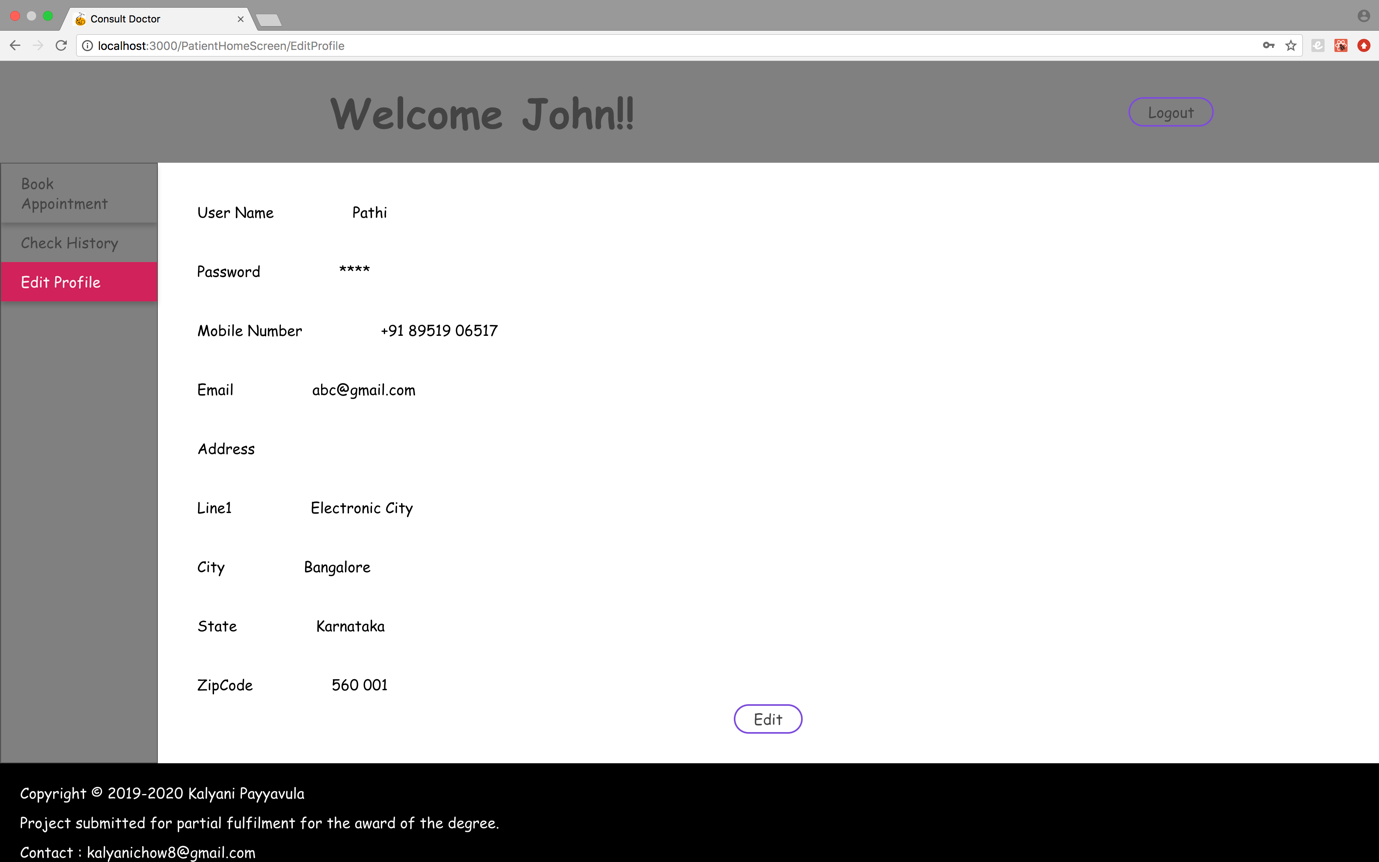
### Login Page



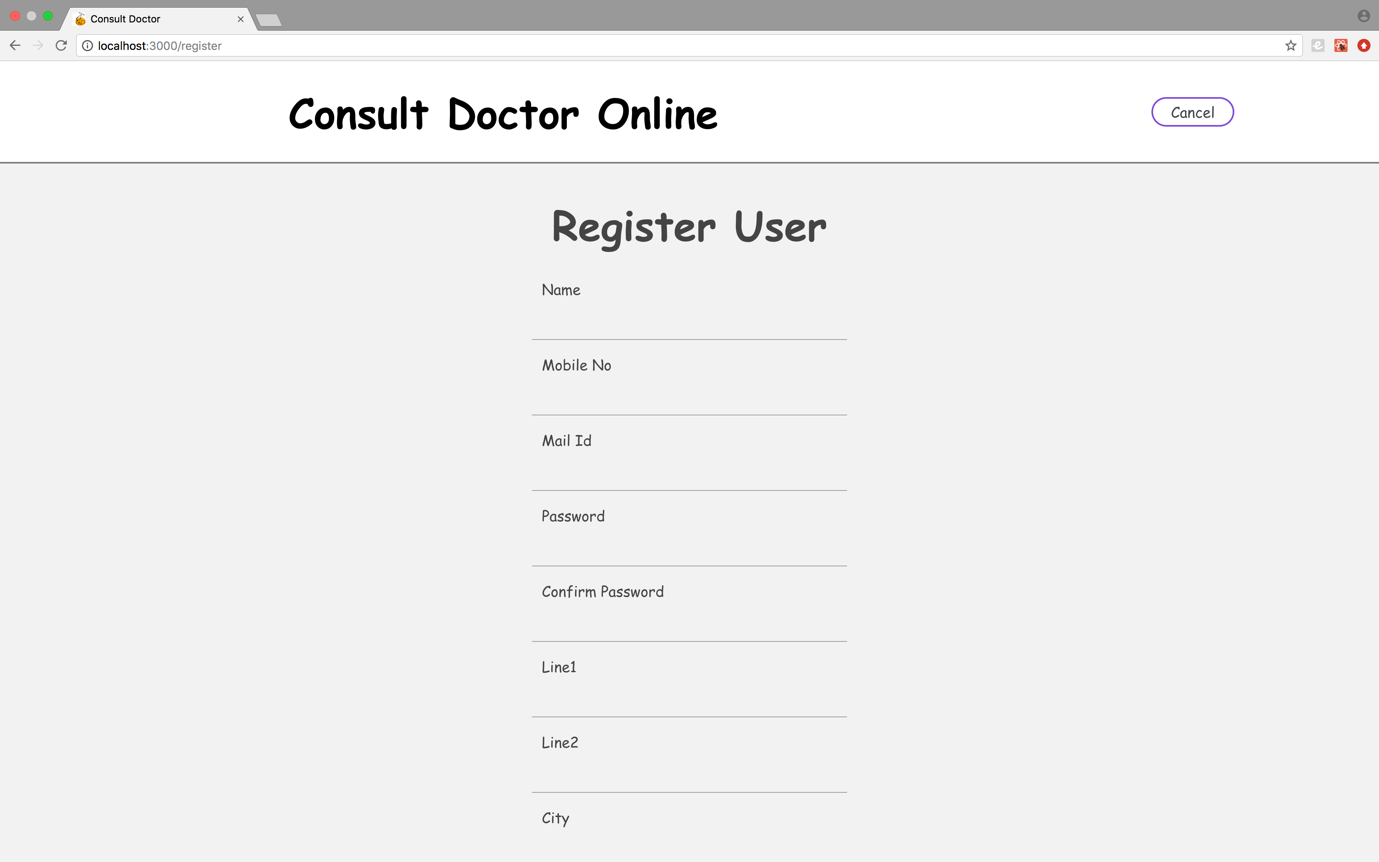
### History Page



### Profile Page



### Register New User



# Results & Disucssions

## TEST PLAN

### Purpose

Testing is the process of detecting errors. Testing performs a very critical role for quality assurance and for ensuring the reliability of software. The results of testing are used later on during maintenance also.

Objective of Test plan is to define the various Testing strategies and testing tools used for complete Testing life cycle of this project.

### Scope

The document mainly targets the GUI testing and validating data in report output as per Requirements Specifications provided by Client.

### Functions to be tested

GUI

Data

Exception scenarios

Compatibility

## TESTING PROCESS OVERVIEW

### Preparing Test Cases:

QA will be preparing test cases based on the requirement specifications. This will cover all scenarios for requirements.

### Preparing Test Matrix:

QA will be preparing test matrix which maps test cases to respective requirement. This will ensure the coverage for requirements.

### Reviewing test cases and matrix:

* Peer review will be conducted for test cases and test matrix by senior QA member in QA team
* In certain cases for e.g. complex requirements, lead's help will be taken for conducting review
* Any comments or suggestions on test cases and test coverage will be provided by reviewer respective Author of Test Case and Test Matrix
* Suggestions or improvements will be re-worked by author and will be send for approval
* Re-worked improvements will be reviewed and approved by reviewer

### Creating Test Data:

Test data will be created by respective QA on client's developments/test site based on scenarios and Test cases.

### Executing Test Cases:

* Test cases will be executed by respective QA on client's development/test site based on designed scenarios, test cases and Test data.
* Test result (Actual Result, Pass/Fail) will updated in test case document

### Defect Logging and Reporting:

QA will be logging the defect/bugs in Bugzilla bug tracking tool found during execution of test cases and will assigned the Bug id generated by Bugzilla to respective test cases document. After this, QA will inform respective developer about the defect/bugs.

### Retesting and Regression Testing:

Retesting for fixed bugs will be done by respective QA once it is resolved by respective developer and bug/defect status will be updated accordingly. In certain cases, regression testing will be done if required.

### Deployment/Delivery:

* Once all bugs/defect reported after complete testing is fixed and no other bugs are found, report will be deployed to client’s test site by developer.
* Once round of testing will be done by QA on client’s test site if required
* Report will be delivered along with sample output by email to respective lead and Report group.
* QA will be submitting the filled hard copy of delivery slip to respective developer.
* Once lead gets the hard copy of delivery slip filled by QA and developer, he will send the report delivery email to client.

## TEST STRATEGY

### Black box testing:

It is some time called behavioral testing or Partition testing. This kind of testing focuses on the functional requirements of the software. It enables one to derive sets of input conditions that that will fully exercise all functional requirements for a program.

### GUI Testing:

GUI testing will includes testing the UI part of report. It covers users Report format, look and feel, error messages, spelling mistakes, GUI guideline violations.

### Functional Testing:

Functional testing is carried out in order to find out unexpected behavior of the report. The characteristic of functional testing are to provide correctness, reliability, testability and accuracy of the report output/data.

### User acceptance testing:

The purpose behind user acceptance testing is to conform that system is developed according to the specified user requirements and is ready for operational use. Acceptance testing is carried out at two levels - Alpha and Beta Testing. User acceptance testing (UAT) will be done at the Client.

## REVIEWS

Reviews will be done on following documents and review report will be prepare for each work products.

* Test cases
* RTM(Requirement Traceability Matrix)

## Change Request:

Change request for report will be handled using following process:

* Understanding the change request and its impact on exiting report functionality
* If the change is major, test cases will be updated
* If the change is minor, test cases will may not be updated
* Retesting and regression testing will be done as per changed request

## Defect Reporting:

Bugs found during static and dynamic testing will be logged in Bugzilla bug tracking tool.

### Entry Criteria

* The whole source code must be unit tested H/W and S/W should be in place
* QA resources have completely understood the requirements
* QA resources have sound knowledge of functionality in Reports
* Reviewed test scenarios, test cases and RTM

### Suspension Criteria

* The build contains many serious defects which seriously or limit testing progress.
* Significant change in requirements suggested by client
* Software/Hardware problems
* Assigned resources are not available when needed by test team.

### Exit Criteria

* No defects over a period of time or less testing efforts
* All the high priority/severity test cases has been executed
* Deliverables are ready
* High severity/ priority bugs are fixed

### Risk

* Delay in delivery of test items might require increased night shift scheduling to meet the delivery date
* Understanding requirements
* Domain and project knowledge

## Unit Test:

Unit testing focuses verification effort on the smallest unit of software design-the software component or module. Using the component – level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented and the step can be conducted in parallel for multiple components.

* **Login Module:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No** | **Test Case Description** | **Input** | **Expected Behavior** | **Actual Behavior** | **Test Result** |
| 1 | Can userid be empty? | Null ID | ID cannot be NULL | Warning message "ID can't be NULL" | Pass |
| 2 | Can password be empty? | Null pass­word | Password Can't be NULL | Warning message "password can't be Null" | Pass |
| 3 | Login button is working or not? | Button pressed | Perform login processing | Call user authentication | Pass |
| 4. | Is Login Frame displaying properly? | Invoke Login Frame | All text fields are displayed and are properly aligned | Displays properly as overlay | Pass |

* **Server side Login Module**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No** | **Test Case**  **Description** | **Input** | **Expected Behavior** | **Actual Behavior** | **Test Result** |
| 1 | Is Database Connection establishing? | Connection object is created | Connection establishes | No error during connection was found | Pass |
| 2 | Is able to retrieve Login ID and password from database | Login ID+ Pass-Word | Able to fetch data from Database | No error found during data fetching | Pass |
| 3 | Is able to match Login ID & Password | Login Id +Pass-word | Proper matching | Matching done | Pass |

## Integration Testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design. Incremental integration is the antithesis of the big bang approach.

The program is constructed a teste in small increments, where errors are easier to isolate and correct, interfaces are more likely to be tested completely, and a systematic test approach may be applied.

* **Login Module, History Module and Appointment Module is Integrated**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr No** | **Test Case Description** | **Input** | **Expected Behavior** | **Actual Behavior** | **Test Result** |
| 1 | Is Database Connection establishing? | Connection object is created | Connection establishes | No error during connection was found | Pass |
| 2 | Is able to verify Login ID & Password | Login Id +Pass-word | Valid username and password | Matching done | Pass |
| 3 | Does status of user changes to 'Logout status’? | Logout | Status should change. | No change in  Status | Pass |

##### **Add Appointment Module, Cancel Appointment Module and Check History Module**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr No** | **Test Case Description** | **Input** | **Expected Behavior** | **Actual**  **Behavior** | **Test Result** |
| 1 | Is new Appointment added? | New Timeslot is selected | Appointment details to be displayed.  List of available slots are shown. | User gets confirmation for new appointment. | Pass |
| 2 | Is existing Appointment deleted from database? | topic/options to select | Appointment should be deleted and should not be displayed on the view page | Appointment is deleted and not displayed on the view page. | Pass |
| 3 | Is existing Appointment updated into database? | topic/options to select | Appointment should be updated and should be displayed on the view page | Appointment is updated and displayed on the view page. | Pass |

## System Testing

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.

* **Functional Requirements**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr.**  **No** | **Test Case Description** | **Input** | **Expected Behavior** | **Actual Behavior** | **Test**  **Result** |
| 1 | Can Doctor Login? | Admin Loginld + password | User should Be login | Admin is login only When Login Id & password is valid | Pass |
| 2 | Can Patient Login? | Loginld + password | User should Be login | Employee is login only When Login Id & password is valid | Pass |
| 3 | Can Patient book Appointment | Create the new Appointment | Patient should be able to create the new Appointment | Appointment is successfully taken. | Pass |
| 4. | Can Patient cancel Appointment | Delete the new voting topics | Admin should be able to delete the new topic list for voting | Deleted Appointment will not be displayed | Pass |
| 5 | Can Doctor Update Appointment | List of appointments | Doctor should be able to update appoinment | Appointment records will be displayed | Pass |
| 6. | Can Patient view past appointments | Topics must be displayed on the page until the due date | Employee should be able to view | Page will be displayed with list of topics | Pass |
| 7. | Can doctor mark Appointment as attended | Topics with the options | Doctor should be able to select the select the appointment | Select/Radio button will be clicked | Pass |
| 8 | Can Patient Logout | Click On Logout | Patient should be able to Logout | Logout Message | Pass |

## TEST CYCLES and TEST REPORTS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Total TC** | **Total Passed** | **Total Failed** | **Executed /Valid (in %)** | **Passed /Valid (in %)** |
| Cycle 1 | 12 | 12 | 0 | 100% | 100% |
| Cycle 2 | 11 | 11 | 0 | 100% | 100% |

# Conclusion and Future Work

As the project suggests, it is more than a normal appointment booking system. Patients get a clear picture of each doctor’s profile. Thus, they can have some understanding about the doctor before they actually book appointment. Patients can book appointment round the clock, thus eliminating human intervention. Patients can directly login with their id and cancel the appointment if they are not able make for it. Doctors can also login and check the list of patients who has appointment on that particular date. Doctors can make a note of patients prescription.

In future the application can open also in mobiles and tabs. Patients can get calendar notifications about appointment. They can pay doctor fees online using PayPal and bank gateways. Clinics or Hospitals can totally eliminate front office managing the appointment system. Doctor has a clear picture of appointments reserved per day. He can make a note of prescription given for each patient. In this way a clear history for a patient always exists with the doctor. This will help doctor to suggest a better prescription.