

DR. HARISINGH GOUR UNIVERSITY, SAGAR

DEPARTMENT OF COMPUTER SCIENCE AND APPLICATION



PROJECT REPORT

MASTER OF COMPUTER APPLICATION

2020-2021

TOPIC

HOSPITAL BED BOOKING SYSTEM

Submitted To

Prof. Diwakar Shukla (Head Of Department)

Guidance By:

Dr. Dharmesh Shrivastava

Submitted By:

Shrikant Dwivedi

Y18271032

MCA VI Sem

COMPANY CERTIFICATE

ARITE InfoSolutions Pvt Ltd

Regd. Office: 58-B, Pocket A
Sukhdev Vihar, New Delhi 110025
Tel 91-11-41074692
Email info@arite.in

Date: 11 June 2021

CERTIFICATE

This is to certify that **Mr. Shrikant Dwivedi** bearing the Registration No. **Y18271032** from Department of Computer Science and Applications, Dr. Harisingh Gour Vishwavidyalaya, Sagar (M.P.), student of MCA has successfully completed the Industrial Training as part of the course curriculum from our organization.

He done the project entitled **Hospital Bed Booking Management System** during the period from 20 January 2021 to 10 June 2021 under the guidance and supervision of **Dr. Dharmesh Shrivastava**.

He has completed project well within the time frame. He is sincere, hardworking and his conduct during period was good.

We wish his success in future endeavor.

R Singh
Rashmi Singh
(Manager-HR)

DECLARATION

I hereby declare that the work contained in this project entitled "**Hospital Bed Booking System**" towards the partial fulfillment of the requirements for the award of the Degree of Master of Computer Applications in Department of Computer Science and Application, Dr. Harisingh Gour Vishwavidyalaya, Sagar (M.P.), of my own work and performed by me. It is an original and authentic record.

I have not submitted the matter embodied in the project for the award of any other degree or diploma to any other institute or University.



Signature of Student

Name: Shrikant Dwivedi

Registration No: Y18271032

ACKNOWLEDGEMENT

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many Faculties. We would like to extend my sincere thanks to all of them. We highly indebted to **Dr. Harisingh Gour University, Sagar University** for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project. We would like to express my gratitude towards my parents & member of their kind co-operation and encouragement which help us in completion of this project. Last but not least, many thanks go to the head of the project, **Prof. Diwakar Shukla (Head of Department)**, whose have invested his full effort in guiding the team in achieving the goal. We have to appreciate the guidance given by other supervisor as well as the panels especially in our project presentation that has improved our presentation skills thanks to their comment and advises. We would like to express our special gratitude and thanks to all above mentioned people for giving us such attention and time. Our thanks and appreciations also go to our colleague in developing the project and people who have willingly helped us out with their abilities.



Shrikant Dwivedi

Roll No: Y18271032

INDEX

Company Certificate	I
Declaration	II
Acknowledgement	III
CHAPTER 1 INTRODUCTION	1
1.1 Introduction	2
1.2 Overview	4
1.3 Objectives	5
1.4 Existing System	7
1.5 Proposed System	8
CHAPTER 2 TECHNOLOGY DESCRIPTION	9
2.1 Technology Used	10
2.2 Tools Used	26
CHAPTER 3 SYSTEM REQUIREMENTS	38
3.1 Hardware Requirement	39
3.2 Software Requirement	39
CHAPTER 4 SYSTEM ANALYSIS	44
4.1 Introduction	45
4.2 Requirement and analysis specifications.....	45
4.3 Technical Feasibility	48
4.4 Operational Feasibility	49

4.5 Economic Feasibility	50
CHAPTER 5 SYSTEM DESIGN	52
5.1 Introduction	53
5.2 Architecture Design	54
CHAPTER 6 METHODOLOGY	58
6.1 Methodology Flow	59
6.2 Front End Development	63
6.3 Database Development	83
5.3.1 ER Diagram	84
6.4 Backend Development	85
CHAPTER 7 SCREENSHOTS & OUTPUT SCREEN.....	104
CHAPTER 8 CONCLUSION & FUTURE SCOPES	109
8.1 Conclusion	110
CHAPTER 9 BIBLOGRAPHY	112

CHAPTER 1

INTRODUCTION

1.1 Introduction:

“Hospital Bed Booking System” Hospital bed management systems became a pivotal practice after the National Health Service’s statement rule on patient’s emergency trolley waiting hours. After an accident, inconvenient waiting hours in hospital waiting halls due to improper operational capacity planning and control over managing, allotting, and maintaining beds may turn frustrating when emergency care is required. The lack of real-time data analysis provides a disadvantage resulting in patient overcrowding and malefactions in inpatient bed management services during emergencies.

Monitoring and managing inpatient-outpatient capacity & patient inflows can be challenging tasks during emergencies. Unplanned inventory, the inability of beds, and other scarce resources may lead to poor operations, and that affects the hospital’s goodwill. The data integrated robust bed management systems improve healthcare services by optimizing operations. The **digitized data-driven hospital bed management system** provides an advantage with the accessibility of real-time data. It includes predictive planning, altering beds during emergencies, improved individual patient care, efficiency in diagnosis, and systematized operations.

This is the system where the patient/user book their bed for the patient. In Hospital lots of people gets treatment, they need beds for the rest so need to book the bed for the patient. This system will help management by making their work easy. They can easily analyze which hospital near patients is empty and also show the beds, wards and doctors details so the new patient can use it. The application is implemented in Reactjs , GraphQL, Nodejs, Expressjs and Mongodb. Continuously, it consists of one main models of users, hospital and admin side.

User needs to login to the system with the username and password. In this system employee which is admin can add the new patients with their name, gender, age, blood group, phone number.

Admin can view all the patients that have booked the bed for them. To get the bed employee needs to assign for the bed. They can even see all the beds that are available. Eventually, they can add a new bed too.

User/ patient can check the status of beds in hospital s. suppose there is an emergency, and then user can check the nearest hospital from his current location and also check the wards, beds, doctors available in the particular hospital and easily book the bed in hospital. User can also select the bed type (General/Private) and also select the ward where he wants to admit the patient. Patient can see the hospitals on a map and also can find the way of hospital using this system; he is able to see the details of hospital like how many (Gen/ Private) beds available in particular ward of selected hospital. This bed management software helps in getting the record of hospital bed status. With various information regarding the bed occupancies, this software aids the healthcare centers in optimizing bed management. With a detailed report on the occupancy of beds in each ward, it helps the management in reducing the bed turnover time. The hospital staff gets the details of each patient in the software itself which eases the tasks of doctors and nurses. This also helps in keeping a track of each patient's treatment and health condition thereby resulting in patient satisfaction regarding hospital services.

This bed management software empowers the hospital management team to serve the patients in a better way. There are decision support tools in the software which enables the staff to predict and decide on the management of the beds in various wards so that empty beds don't go unnoticed. This hospital bed management software eases the task of bed assignment and transfer. Right from the patient's admission to a hospital, till the discharge, one can get all the details about the patient.

With a graphical representation of wards and a list of the beds, it becomes easy for the bed management staff to carry out the bed allocation process. Information is updated in real-time, with separate color codes for bed availability and the estimated time for patient discharge. This helps the staff

in getting away from the manual counting process. The management can know the status of the beds which are classified on the basis of location, treatment, and type. Patients don't have to wait longer to get admitted as the staff now has the details of bed availability at a few clicks. The process of patient admission and discharge is made less overwhelming.

1.2 Overview:

The hospital bed management system is a fusion and integration of various systems, which perform and fulfill distinct operations at the basic level of hospital management. The systems include EHR (electronic health records), ADT system (admission, discharge, and transfer system), emergency department information system, and BMS (bed management system). Utilizing the mentioned technologies and their significant insights help in determining a flowchart of hospital operations that can better the existing bed management system. The optimized hospital information helps in better operations during addressing accidental & emergency cases, medical assessment cases, and patients transferred from other hospitals for better treatment.

ADT System: The admission, discharge, and transfer systems are popularly known as ADT that digitally records information regarding patient status, medication details, allotted bed duration, etc. The ADT system is a digital data analytical tool that uses data analysis to predict and provide real-time information on patient demographic data. The accessibility of analyzing this information can provide an advantage in understanding each bed status in a particular ward for better hospital operations. During an A&E (accident or emergency) situation, the time in allotting and shifting reserved patients to vacant beds can be reduced by announcing the list of available beds in different wards. It helps in taking preventive action against emergencies in a hospital.

EHR System: An electronic health records system is a digital tabulation of patients' paper charts that provide real-time instant access to individual patient history for authorized users. The data stored in EHR is in an unstructured format. Hence with this data, the hospital operations management can divide and calculate the no.of beds available for occupancy in every ward and the reserved time and date of regular patients who often visit the hospital for checkups.

HBMS System: The hospital bed management system provides real-time data access to staff and senior management regarding vacant, preoccupied, reserved information of beds through digital dashboard. The HBMS is systematically interconnected with other systems like ADT and EHR for obtaining, resourcing, and analyzing real-time data.

1.3 Objectives: The project “Hospital Bed Booking System” is aimed to develop to maintain day-to-day status of bed in hospitals, list of wards, list of doctors etc.

- To computerize all details regarding patient details and hospital details.
- Scheduling the appointment of patient with doctors to make it convenient for both.
- Scheduling the service of specialized doctors and emergency properly so that facilities provided by hospital are fully utilized in effective manner.
- If an accident or any critical conditions or cases comes then people can search hospital which is the nearest hospital are there and can generate a request for bed in hospital.
- Patient can choose the bed type and also can see the doctors and wards available in the hospital.

- Patient can also check how many beds are available in particular ward of a hospital, where he wants to admit the patient.
- Develop a database for the project.
- It's providing a user friendly web interface for patient and hospital owner to admit/discharge a patient and also some more features.
- Have sufficient memory space to store the data.
- Have some extra features where government can get the information about BPL beds in the hospitals.

All this work is done manually by the receptionist and other operational staff and lot of papers are needed to be handled and taken care of. Doctors have to remember various medicines available for diagnosis and sometimes miss better alternatives as they can't remember them at that time.

The lack of adequate numbers of hospital beds to accommodate the injured is a main problem in public hospitals. For control of occupancy of bed, we design a dynamic system that announces status of bed when it changes with admission or discharge of a patient. This system provide a wide network in country for bed management, especially for ICU and CCU beds that help us to distribute injured patient in the hospitals. I design and implement a national-wide bed management system for management cross hospital referral patient in crowded hospital that hasn't enough bed for admission patient. Also this system use for accommodate victims in mass-casualty incidents.

The Hospital Management System can be entered using a username and password. It is accessible either by an administrator or receptionist. Only they can add data into the database. The data can be retrieved easily. The interface is very user-friendly. The data are well protected for personal use and makes the data processing very fast.

1.4 Existing System:

In the current system the data required is maintained in records. They are to be updated according to the requirements of the users. It takes time to search for the required query. All the details regarding the hospital and its patients are hard to maintain. The work will be more, so the system needs more number of crew to fulfill the requirements. There may be a chance of failure since it is manual. A one fault of the system may lead to inconvenience and also causes a vast destruction. So these faults make the system less efficient and performance of the system is very slow. Hence, there should be a system to overcome all these defaults and provide the users with more facilities.

In the current system if the user was suffering from any pain or etc he/she has no idea how to control the pain and suffering. Just they will be no idea for them and they become sicker and died more sooner And to know the availability for the treatment they have to go to hospital but mostly the government hospital doesn't give more facilities to the patient as the patients want from the doctors. But in the case of the private hospital the patients has to pay more fares for the treatment and they do more delays in the case of the treatment they will be more formalities to be fulfill by the patients which take lot of time waste.

1.5 Proposed System:

In the proposed system everything is computerized. The system provides all the details regarding the Hospital, doctors, patients, bed numbers, and fares also and so on. The user can search required data easily with no time. A very less number of staff is required to handle the system.

The patients need not wait for a long time to fulfill his requirement. There is no chance of any failure in the system, which improves the performance of the system and also increases the efficiency of the system.

Though this system is very beneficial a minor failures in the server or else the computer leads a major loss of data.

CHAPTER 2

TECHNOLOGY DESCRIPTION

2.1 Technology Used

It describes the technology required during development of Design and Implementation of Online Buying, Selling and Bidding Application. In the development of the proposed system, following tools and technologies are used. “MERN stack”, It comprises advanced technologies used to develop both the server-side and the client-side of a web application in a JavaScript environment. The components of the MEAN stack include the MongoDB Database, Express.js (a web framework), ReactJS (a front-end framework), and the Node.js runtime environment. Taking control of the MEAN stack and familiarizing different JavaScript technologies during the process will help you in becoming a full-stack JavaScript developer. JavaScript’s sphere of influence has dramatically grown over the years and with that growth, there’s an ongoing desire to keep up with the latest trends in programming. New technologies have emerged and existing technologies have been rewritten from the ground up.

2.1.1 ReactJS:

ReactJS is JavaScript library used for building reusable UI components. According to React official documentation, following is the definition –

React is a library for building composable user interfaces. It encourages the creation of reusable UI components, which present data that changes over time. Lots of people use React as the V in MVC. React abstracts away the DOM from you, offering a simpler programming model and better performance. React can also render on the server using Node, and it can power native apps using React Native. React implements one-way reactive data flow, which reduces the boilerplate and is easier to reason about than traditional data binding.

React Features

- **JSX** – JSX is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.
- **Components** – React is all about components. You need to think of everything as a component. This will help you maintain the code when working on larger scale projects.
- **Unidirectional data flow and Flux** – React implements one-way data flow which makes it easy to reason about your app. Flux is a pattern that helps keeping your data unidirectional.
- **License** – React is licensed under the Facebook Inc. Documentation is licensed under CC BY 4.0.

React Advantages

- Uses virtual DOM which is a JavaScript object. This will improve apps performance, since JavaScript virtual DOM is faster than the regular DOM.
- Can be used on client and server side as well as with other frameworks.
- Component and data patterns improve readability, which helps to maintain larger apps.

React Limitations

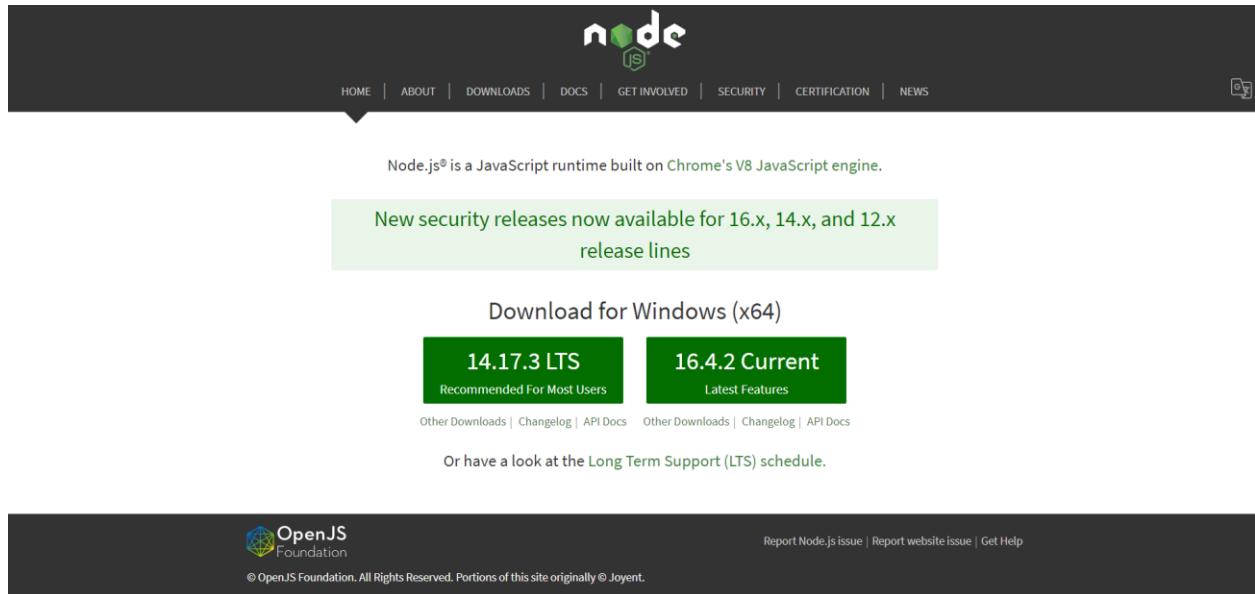
- Covers only the view layer of the app, hence you still need to choose other technologies to get a complete tooling set for development.
- Uses inline templating and JSX, which might seem awkward to some developers.

2.1.1.1 Environment Setup

In this chapter, we will show you how to set up an environment for successful React development. Notice that there are many steps involved but this will help speed up the development process later. We will need **NodeJS**, so if you don't have it installed, check the link from the following.

NodeJS and NPM

NodeJS is the platform needed for the ReactJS development. Checkout our [NodeJS Environment Setup.](#)



After successfully installing NodeJS, we can start installing React upon it using npm. You can install ReactJS in two ways

- Using webpack and babel.
- Using the **create-react-app** command.

Installing ReactJS using webpack and babel

Webpack is a module bundler (manages and loads independent modules). It takes dependent modules and compiles them to a single (file) bundle. You can use this bundle while developing apps using command line or, by configuring it using webpack.config file.

Babel is a JavaScript compiler and transpiler. It is used to convert one source code to other. Using this you will be able to use the new ES6 features in your code where, babel converts it into plain old ES5 which can be run on all browsers.

Step 1 - Create the Root Folder

Create a folder with name **reactApp** on the desktop to install all the required files, using the **mkdir** command.

```
C:\Users\username\Desktop>mkdir reactApp  
C:\Users\username\Desktop>cd reactApp
```

To create any module, it is required to generate the **package.json** file. Therefore, after Creating the folder, we need to create a **package.json** file. To do so you need to run the **npm init** command from the command prompt.

```
C:\Users\username\Desktop\reactApp>npm init
```

This command asks information about the module such as packagename, description, author etc. you can skip these using the **-y** option.

```
C:\Users\username\Desktop\reactApp>npm init -y  
Wrote to C:\reactApp\package.json:  
{  
  "name": "reactApp",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\"Error: no test specified\\" && exit 1"  
  },  
  "keywords": []  
}
```

```
    "author": "",  
    "license": "ISC"  
}
```

Step 2 - install React and react dom

Since our main task is to install ReactJS, install it, and its dom packages, using **install react** and **react-dom** commands of npm respectively. You can add the packages we install, to **package.json** file using the **--save** option.

```
C:\Users\Tutorialspoint\Desktop\reactApp>npm install react --save  
C:\Users\Tutorialspoint\Desktop\reactApp>npm install react-dom --  
save
```

Or, you can install all of them in single command as –

```
C:\Users\username\Desktop\reactApp>npm install react react-dom --  
save
```

Step 3 - Install webpack

Since we are using webpack to generate bundler install webpack, webpack-dev-server and webpack-cli.

```
C:\Users\username\Desktop\reactApp>npm install webpack --save  
C:\Users\username\Desktop\reactApp>npm install webpack-dev-server -  
--save  
C:\Users\username\Desktop\reactApp>npm install webpack-cli --save
```

Or, you can install all of them in single command as –

```
C:\Users\username\Desktop\reactApp>npm install webpack webpack-dev-  
server webpack-cli --save
```

Step 4 - Install babel

Install babel, and its plugins babel-core, babel-loader, babel-preset-env, babel-preset-react and, html-webpack-plugin

```
C:\Users\username\Desktop\reactApp>npm install babel-core --save-  
dev  
C:\Users\username\Desktop\reactApp>npm install babel-loader --save-  
dev  
C:\Users\username\Desktop\reactApp>npm install babel-preset-env --  
save-dev  
C:\Users\username\Desktop\reactApp>npm install babel-preset-react -  
--save-dev
```

```
C:\Users\username\Desktop\reactApp>npm install html-webpack-plugin  
--save-dev
```

Or, you can install all of them in single command as –

```
C:\Users\username\Desktop\reactApp>npm install babel-core babel-  
loader babel-preset-env  
babel-preset-react html-webpack-plugin --save-dev
```

Step 5 - Create the Files

To complete the installation, we need to create certain files namely, index.html, App.js, main.js, webpack.config.js and, **.babelrc**. You can create these files manually or, using **command prompt**.

```
C:\Users\username\Desktop\reactApp>type nul > index.html  
C:\Users\username\Desktop\reactApp>type nul > App.js  
C:\Users\username\Desktop\reactApp>type nul > main.js  
C:\Users\username\Desktop\reactApp>type nul > webpack.config.js  
C:\Users\username\Desktop\reactApp>type nul > .babelrc
```

Step 6 - Set Compiler, Server and Loaders

Open **webpack-config.js** file and add the following code. We are setting webpack entry point to be main.js. Output path is the place where bundled app will be served. We are also setting the development server to **8001** port. You can choose any port you want.

webpack.config.js

```
const path = require('path');  
const HtmlWebpackPlugin = require('html-webpack-plugin');  
  
module.exports = {  
  entry: './main.js',  
  output: {  
    path: path.join(__dirname, '/bundle'),  
    filename: 'index_bundle.js'  
  },  
  devServer: {  
    inline: true,  
    port: 8001  
  },
```

```

    module: {
      rules: [
        {
          test: /\.jsx?$/,
          exclude: /node_modules/,
          loader: 'babel-loader',
          query: {
            presets: ['es2015', 'react']
          }
        }
      ]
    },
    plugins:[
      new HtmlWebpackPlugin({
        template: './index.html'
      })
    ]
  }
}

```

Open the **package.json** and delete "test" "echo \"Error: no test specified\" && exit 1" inside "scripts" object. We are deleting this line since we will not do any testing in this tutorial. Let's add the **start** and **build** commands instead.

```

"start": "webpack-dev-server --mode development --open --hot",
"build": "webpack --mode production"

```

Step 7 - index.html

This is just regular HTML. We are setting **div id = "app"** as a root element for our app and adding **index_bundle.js** script, which is our bundled app file.

```

<!DOCTYPE html>
<html lang = "en">
  <head>
    <meta charset = "UTF-8">
    <title>React App</title>
  </head>
  <body>
    <div id = "app"></div>
    <script src = 'index_bundle.js'></script>
  </body>
</html>

```

Step 8 – App.jsx and main.js

This is the first React component. We will explain React components in depth in a subsequent chapter. This component will render **Hello World**.

App.js

```
import React, { Component } from 'react';
class App extends Component{
  render() {
    return(
      <div>
        <h1>Hello World</h1>
      </div>
    );
  }
}
export default App;
```

We need to import this component and render it to our root **App** element, so we can see it in the browser.

main.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.js';

ReactDOM.render(<App />, document.getElementById('app'));
```

Note – Whenever you want to use something, you need to **import** it first. If you want to make the component usable in other parts of the app, you need to **export** it after creation and import it in the file where you want to use it.

Create a file with name **.babelrc** and copy the following content to it.

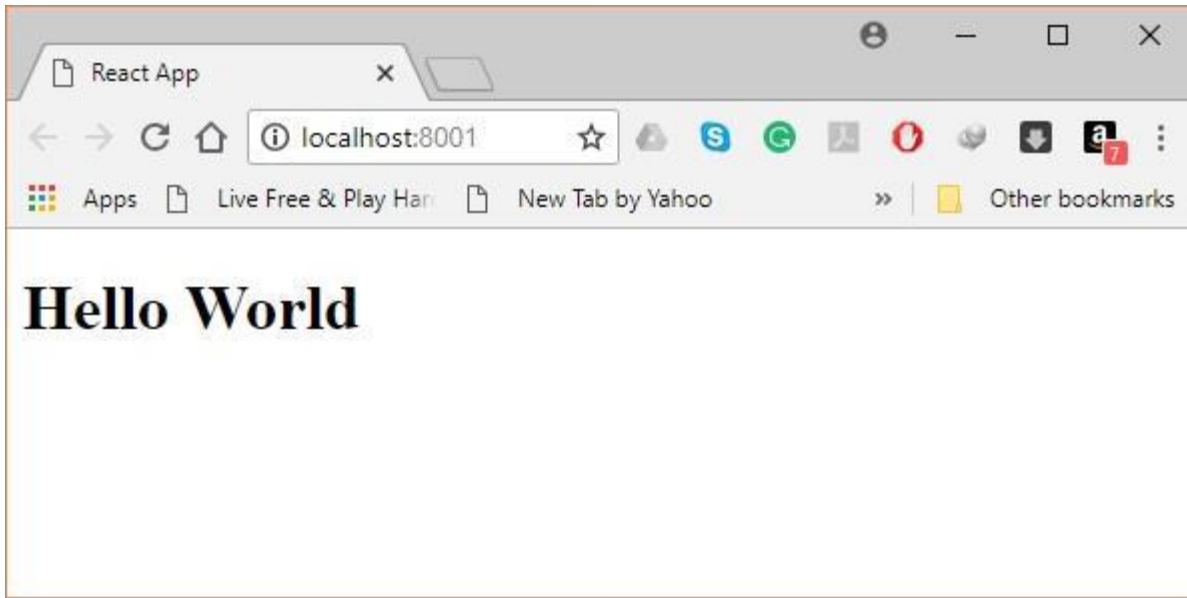
```
{
  "presets": ["env", "react"]
}
```

Step 9 - Running the Server

The setup is complete and we can start the server by running the following command.

```
C:\Users\username\Desktop\reactApp>npm start
```

It will show the port we need to open in the browser. In our case, it is **http://localhost:8001/**. After we open it, we will see the following output.

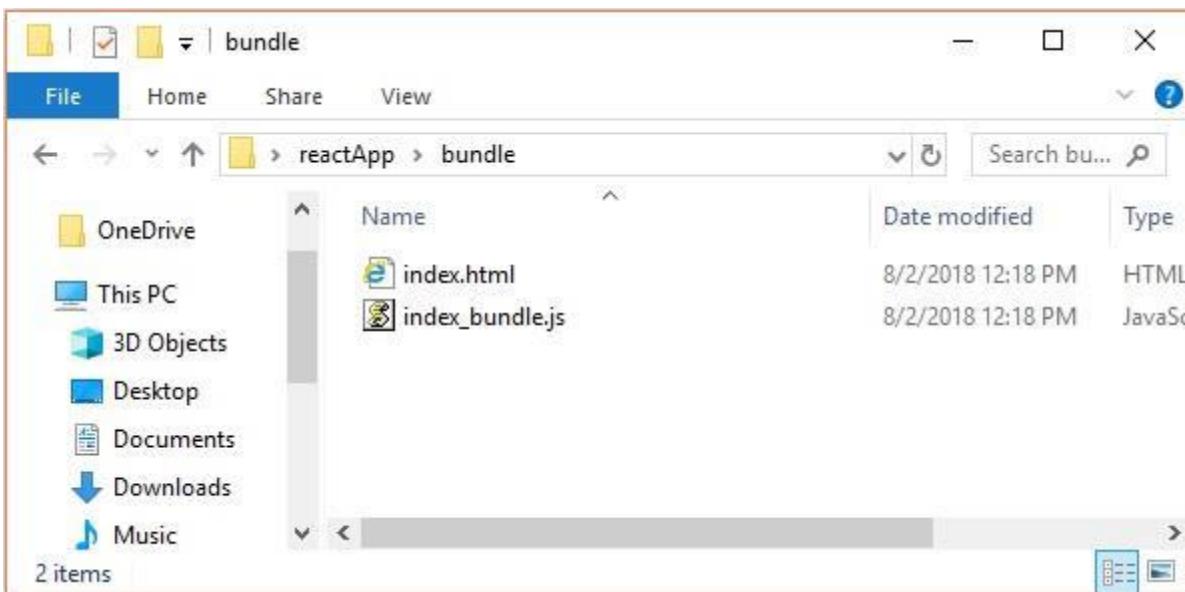


Step 10 - Generating the bundle

Finally, to generate the bundle you need to run the build command in the command prompt as –

```
C:\Users\Tutorialspoint\Desktop\reactApp>npm run build
```

This will generate the bundle in the current folder as shown below.



Using the **create-react-app** command

Instead of using webpack and babel you can install ReactJS more simply by installing **create-react-app**.

Step 1 - install create-react-app

Browse through the desktop and install the Create React App using command prompt as shown below –

```
C:\Users\Tutorialspoint>cd C:\Users\Tutorialspoint\Desktop\  
C:\Users\Tutorialspoint\Desktop>npx create-react-app my-app
```

This will create a folder named my-app on the desktop and installs all the required files in it.

Step 2 - Delete all the source files

Browse through the src folder in the generated my-app folder and remove all the files in it as shown below –

```
C:\Users\Tutorialspoint\Desktop>cd my-app/src  
C:\Users\Tutorialspoint\Desktop\my-app\src>del *  
C:\Users\Tutorialspoint\Desktop\my-app\src\*, Are you sure (Y/N)? y
```

Step 3 - Add files

Add files with names **index.css** and **index.js** in the **src** folder as –

```
C:\Users\Tutorialspoint\Desktop\my-app\src>type nul > index.css  
C:\Users\Tutorialspoint\Desktop\my-app\src>type nul > index.js
```

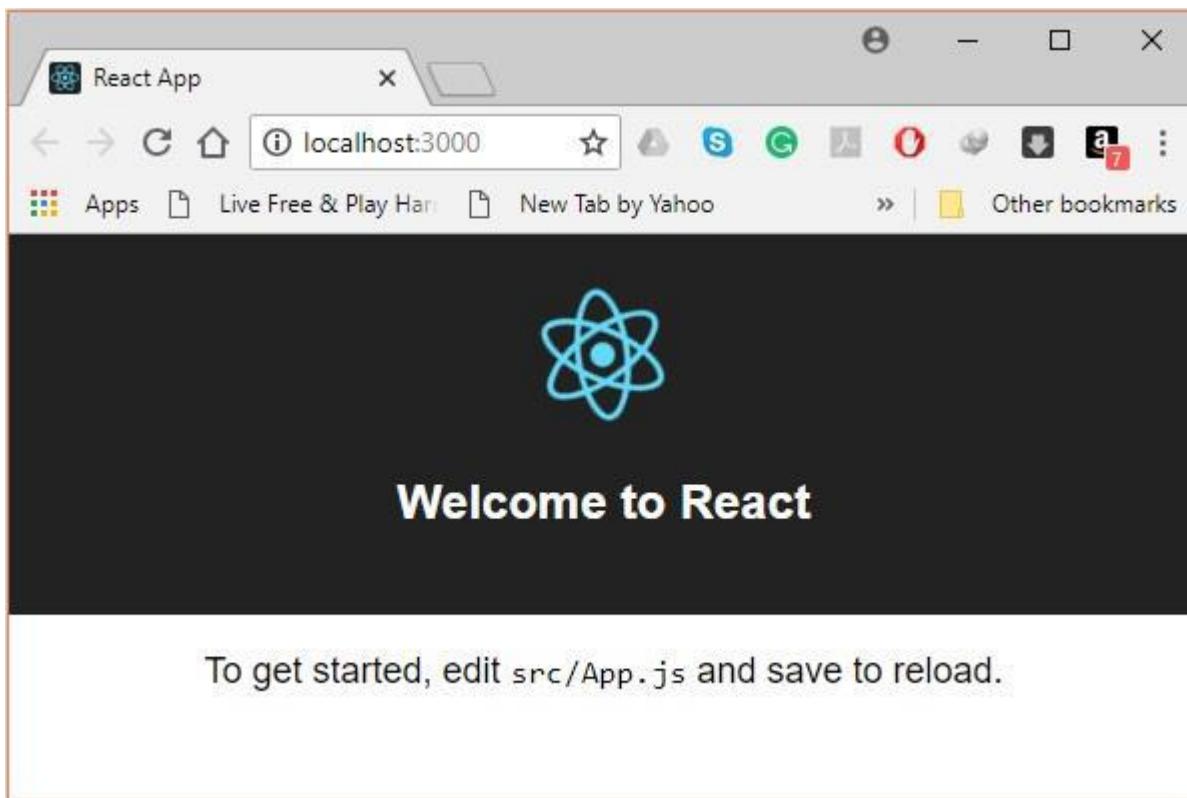
In the **index.js** file add the following code

```
import React from 'react';  
import ReactDOM from 'react-dom';  
import './index.css';
```

Step 4 - Run the project

Finally, run the project using the **start** command.

```
npm start
```



2.1.1.2 GraphQL

GraphQL is an open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data. GraphQL was developed internally by Facebook in 2012 before being publicly released in 2015. On 7 November 2018, the GraphQL project was moved from Facebook to the newly-established GraphQL Foundation, hosted by the non-profit Linux Foundation. Since 2012, GraphQL's rise has followed the adoption timeline as set out by Lee Byron, GraphQL's creator, with accuracy. Byron's goal is to make GraphQL omnipresent across web platforms.

It provides an approach to developing web APIs and has been compared and contrasted with REST and other web service architectures. It allows clients to define the structure of the data required, and the same structure of the data is returned from the server, therefore preventing excessively large amounts of data from being returned, but this has implications for how effective web caching of query results can be. The flexibility and richness of the query language also adds complexity that may not be worthwhile for simple APIs. Despite the name, GraphQL does not provide the richness of graph operations that one might find in a full-fledged graph database such as Neo4j, or even in dialects of SQL that support transitive closure. For example, a GraphQL interface that reports the parents of an individual cannot return, in a single query, the set of all their ancestors.

GraphQL consists of a type system, query language and execution semantics, static validation, and type introspection. It supports reading, writing (mutating), and subscribing to changes to data (realtime updates – most commonly implemented using Websockets).^[10] GraphQL servers are available for multiple languages, including Haskell, JavaScript, Perl, Python, Ruby, Java, C++, C#, Scala, Go, Rust, Elixir, Erlang, PHP, R, D and Clojure.

POST request:

```
{  
  orders {  
    id  
    productsList {  
      product {  
        name  
        price  
      }  
      quantity  
    }  
    totalAmount  
  }  
}
```

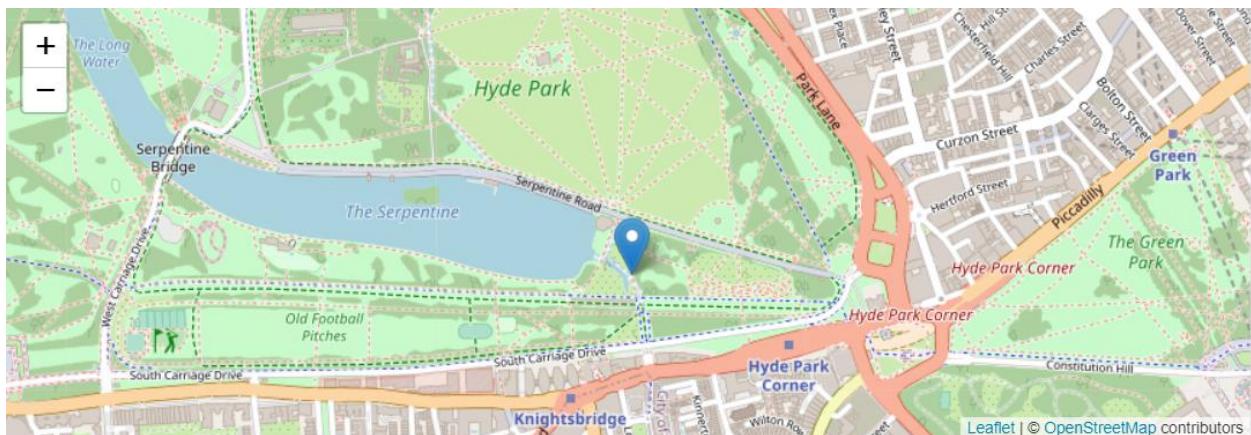
Response:

```
{  
  "data": {  
    "orders": [  
      {  
        "id": 1,  
        "productsList": [  
          {  
            "product": {  
              "name": "orange",  
              "price": 1.5  
            },  
            "quantity": 100  
          }  
        ],  
        "totalAmount": 150  
      }  
    ]  
  }  
}
```

2.1.1.3 Leaflet

Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps. Weighing just about 39 KB of JS, it has all the mapping features most developers ever need.

Leaflet is designed with *simplicity, performance and usability* in mind. It works efficiently across all major desktop and mobile platforms, can be extended with lots of plugins, has a beautiful, easy to use and well-documented API and a simple, readable source code that is a joy to contribute to.



Here we create a map in the 'map' div, add tiles of our choice, and then add a marker with some text in a popup:

```
var map = L.map('map').setView([51.505, -0.09], 13);

L.tileLayer('https://s.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
}).addTo(map);

L.marker([51.5, -0.09]).addTo(map)
    .bindPopup('A pretty CSS3 popup.<br>Easily customizable.')
    .openPopup();
```

2.1.1.4 Json Web Token (JWT)

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on *signed* tokens. Signed tokens can verify the *integrity* of the claims contained within it, while encrypted tokens *hide* those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

When should you use JSON Web Tokens?

Here are some scenarios where JSON Web Tokens are useful:

- **Authorization:** This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.
- **Information Exchange:** JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

2.2 Tools Used

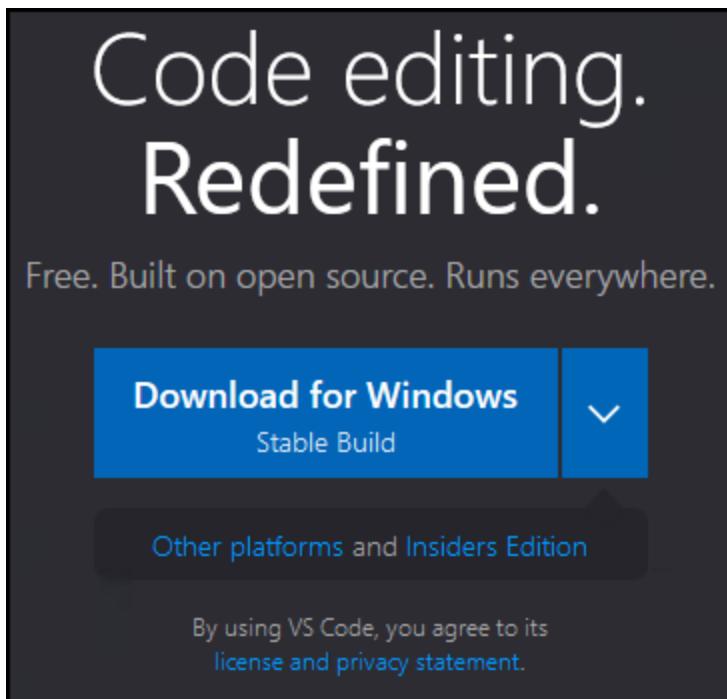
2.2.1 Visual Studio Code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity). Begin your journey with VS Code with these introductory.

2.2.2 Installing Visual Studio Code

1) Downloading VSCode Executable

- Go to [Visual Studio Code](#) website.
- Click on [Download for Windows](#)

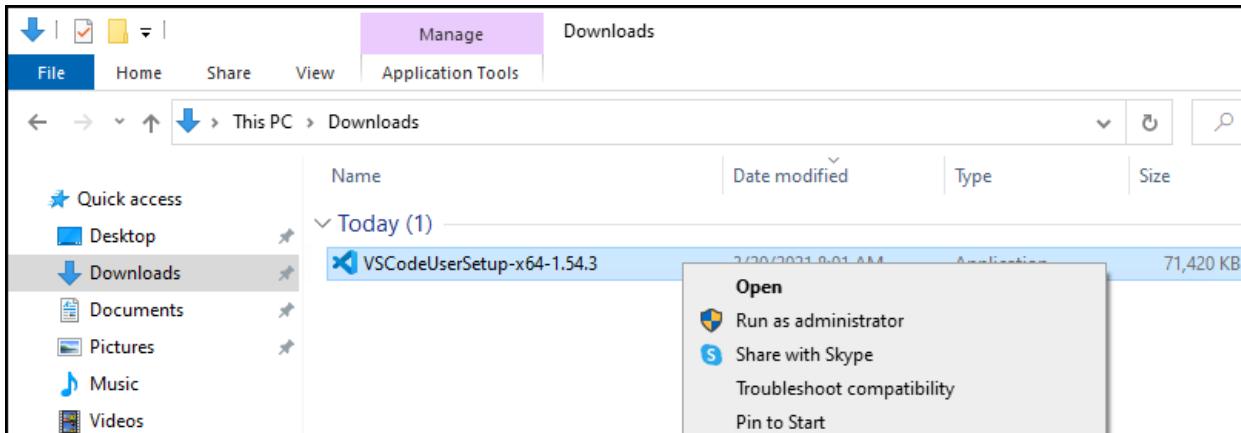


- Click on [Save File](#).



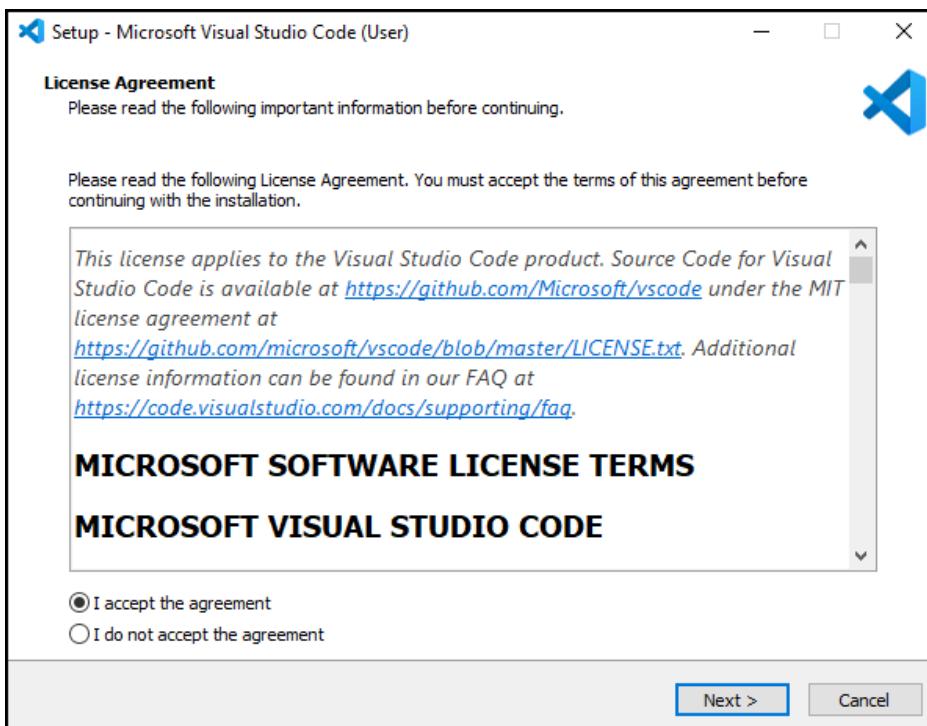
2) Running VSCode Executable

- In your Downloads directory, right click on the VSCode executable and select Open.
-

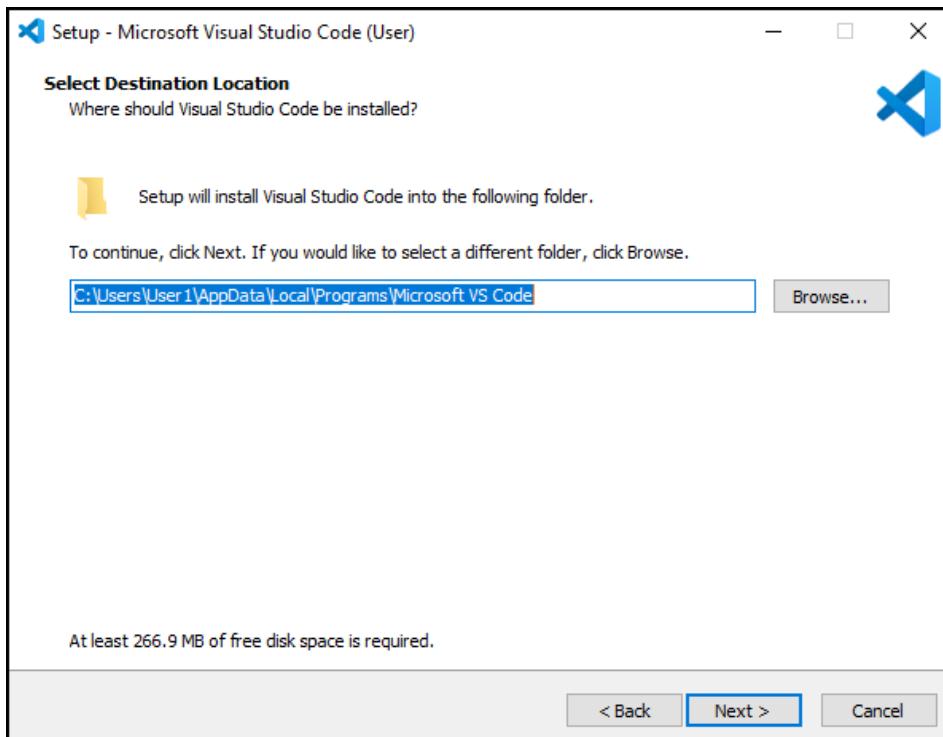


3) Completing the VSCode Setup

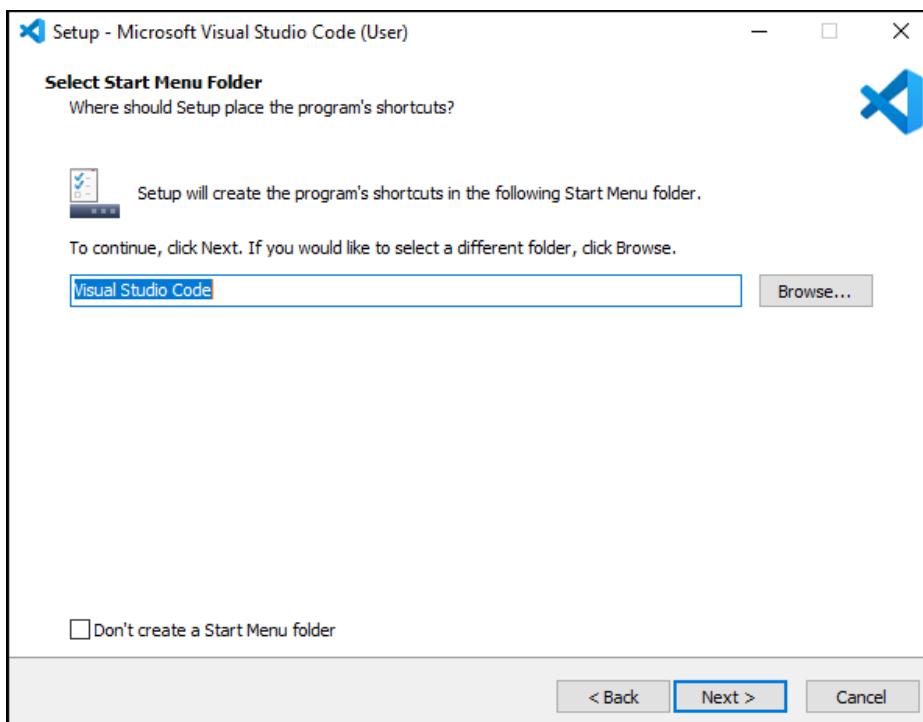
- Accept the agreement.



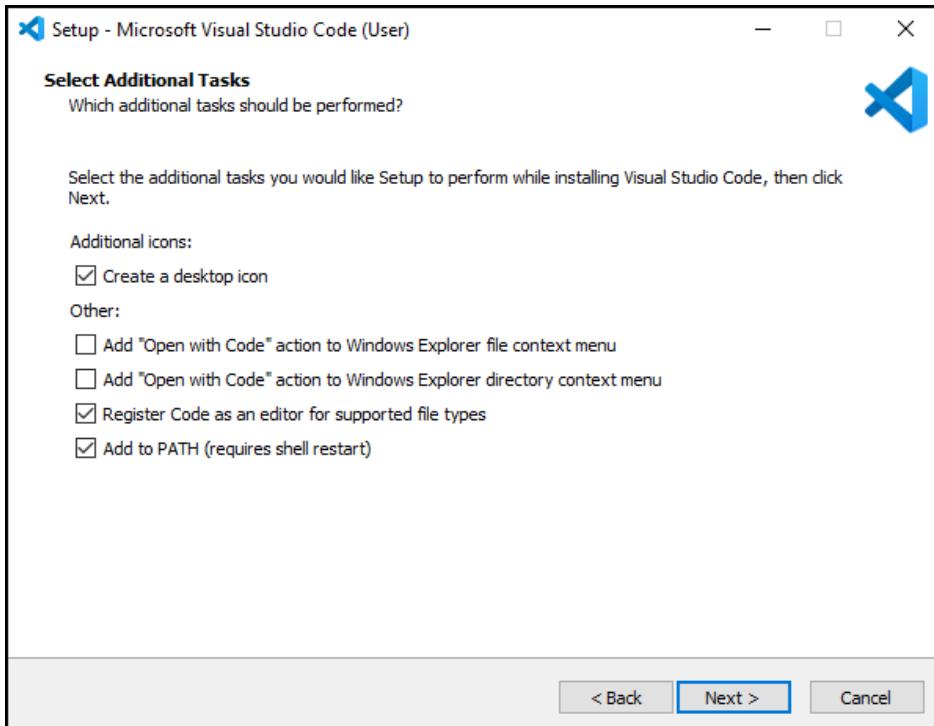
- Select Destination folder and click on Next.



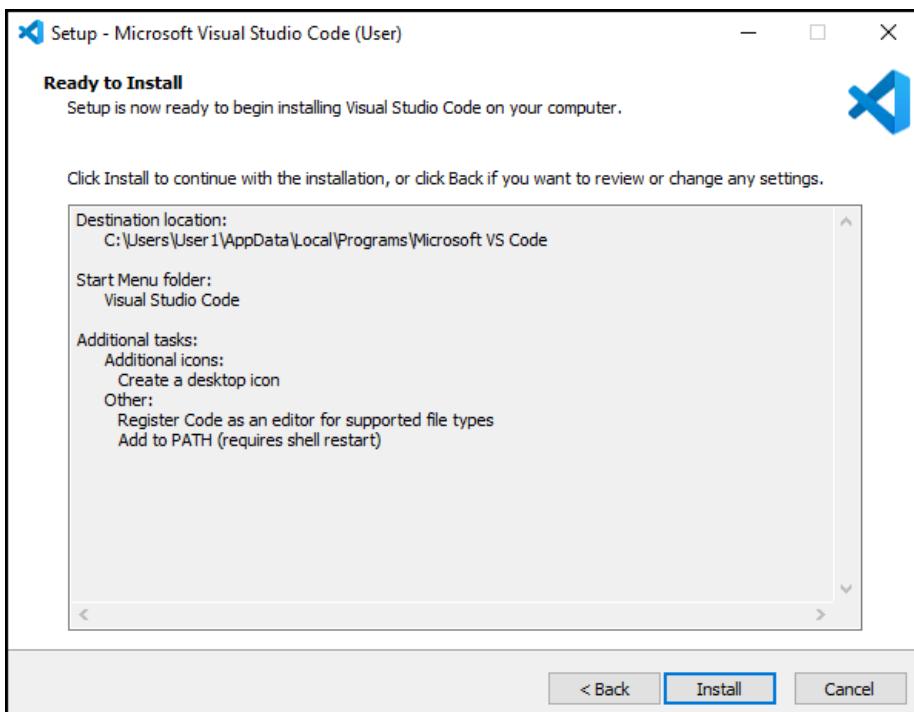
- Select Start Menu folder and click on Next.



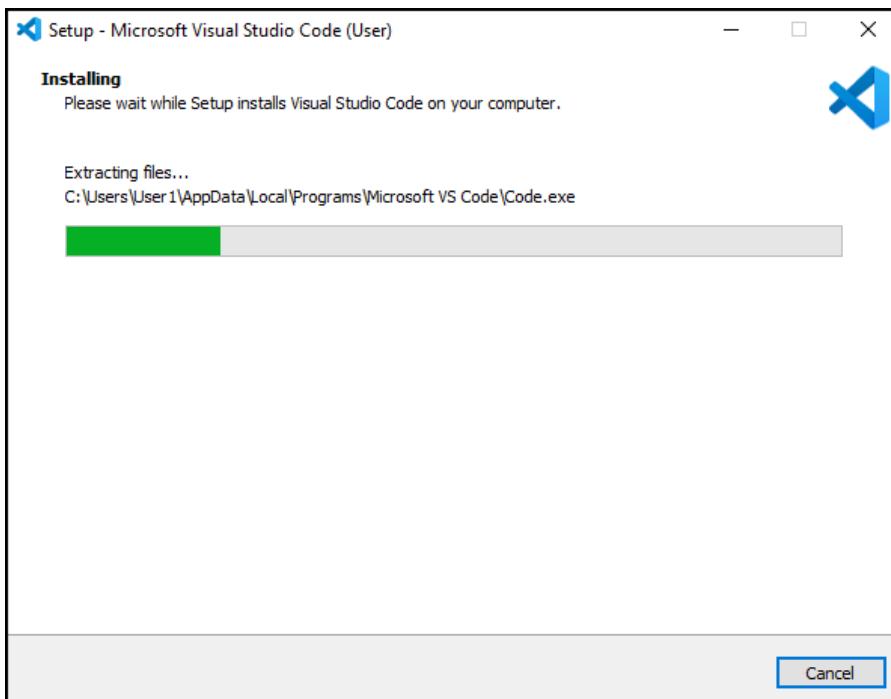
- Select Additional Tasks as desired and click on Next.



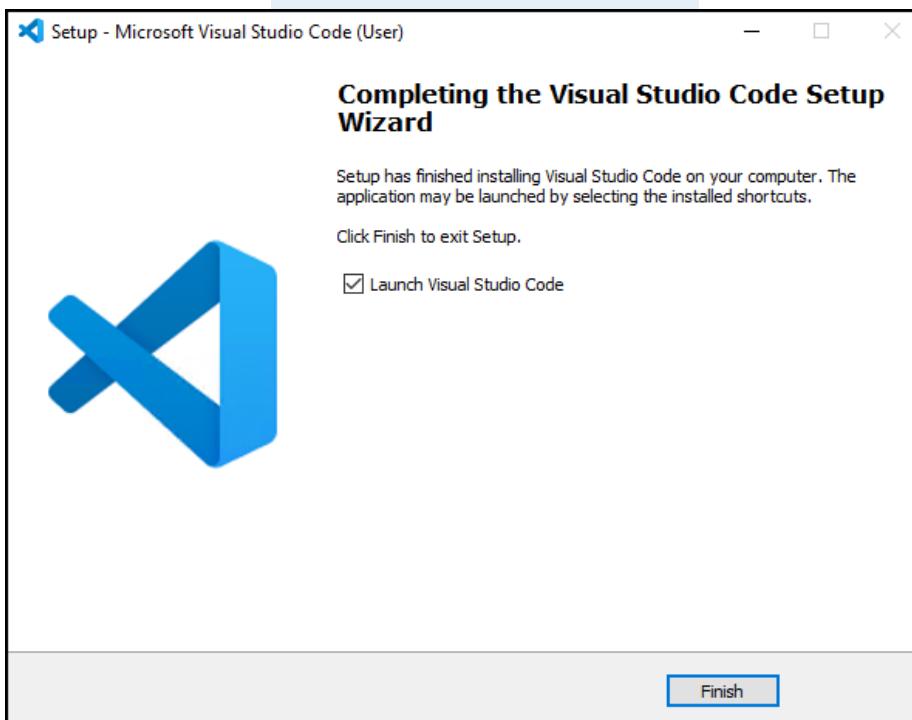
- Click on Install.



- Wait for the installation process to complete.



- Check the Launch Visual Studio Code box and click on Finish.



2.2.3 MongoDB Atlas

MongoDB Atlas is a cloud service by MongoDB. It is built for developers who'd rather spend time building apps than managing databases. This service is available on AWS, Azure, and GCP.

It is the worldwide cloud database service for modern applications that give best-in-class automation and proven practices guarantee availability, scalability, and compliance with the foremost demanding data security and privacy standards. We can use MongoDB's robust ecosystem of drivers, integrations, and tools to create faster and spend less time managing our database.

Advantages of MongoDB Atlas

- **Global clusters for world-class applications:** Using MongoDB Atlas, we are free to choose the cloud partner and ecosystem that fit our business strategy.
- **Secure for sensitive data:** It offers built-in security controls for all our data. It enables enterprise-grade features to integrate with our existing security protocols and compliance standard.
- **Designed for developer productivity:** MongoDB Atlas moves faster with general tools to work with our data and a platform of services that makes it easy to build, secure, and extend applications that run on MongoDB.
- **Reliable for mission-critical workload:** It is built with distributed fault tolerance and automated data recovery.
- **Built for optimal performance:** It makes it easy to scale our databases in any direction. We can get more out of our existing resources with performance optimization tools and real-time visibility into database metrics.
- **Managed for operational efficiency:** It comes with built-in operational best practices, so we can focus on delivering business value and accelerating application development instead of managing databases.

2.2.4 Create an Atlas Account and deploying a Free Tier Cluster

Step 1: Go to <https://www.mongodb.com/cloud/atlas> to register for an Atlas account to host your data.

The screenshot shows the MongoDB Atlas landing page. At the top, there's a navigation bar with links for MongoDB, Cloud, Software, Learn, Solutions, and Docs. On the right side of the header are Contact, Search, Sign In, and a Try Free button. The main content area has a green background with the title "MongoDB Atlas". Below the title, there's a brief description: "Move faster with a cloud MongoDB service. Built for agile teams who'd rather spend time building apps than managing databases. Available on AWS, Azure, and GCP." A prominent "Start free" button is located in the center. Below it, a link says "Already have an account? Log in here →". At the bottom of the main content area, there are links for Pricing, Getting started, Migrate to MongoDB Atlas, and Frequently Asked Questions. A circular icon with a play button is on the far right. A modal window titled "Cloud Provider & Region" is open, showing options for AWS, Google Cloud, and Azure. It also displays a map of the world with various regions highlighted in green, indicating their availability. The "AWS" option is selected, showing "N. Virginia (us-east-1)" as the recommended region.

Step 2: When you click on Start Free, you will be redirected to the Registration form for an account on the MongoDB Atlas.

The screenshot shows the MongoDB Atlas registration form. At the top, there's a header with the MongoDB logo and the text "Get started free" followed by "No credit card required". Below this, there are input fields for "Your Work Email", "First Name", "Last Name", and "Password". A note below the password field says "8 characters minimum". There's also a checkbox for agreeing to the "terms of service and privacy policy". At the bottom of the form is a large green "Get started free" button. The background features a faint globe graphic.

Step 3: Select Starter Clusters and click create a Cluster. The Starter cluster includes the M0, M2, and M5 cluster tiers. These low-cost clusters are suitable for users who are learning MongoDB or developing small proof -of- concept applications.

The screenshot shows the MongoDB Atlas cluster creation interface. At the top, it says "Choose a path. Adjust anytime." and "Available as a fully managed service across 60+ regions on AWS, Azure, and Google Cloud".

Starter Clusters
For teams learning MongoDB or developing small applications.
Starting at **FREE**
Create a cluster

Single-Region Clusters
For teams building applications that need advanced development and production-ready environments.
Starting at **\$0.08/hr***
*estimated cost \$56.8/month
Create a cluster

Multi-Region Clusters
For teams developing world-class applications that require multi-region resiliency or ultra-low latency.
Includes all features from Starter and Single Region Clusters.
Replicate data across multiple regions.
Global Clusters
Globally distributed read and write operations.
Control data residency at the document level.
Starting at **\$0.13/hr***
*estimated cost \$89.60/month
Create a cluster

Step 4: Select your preferred Cloud Provider & Region. It supports M0 Free Tier clusters on Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. The regions that support M0 Free tier clusters are marked with the "*Free Tier Available*" label.

Welcome to MongoDB Atlas! We've recommended some of our most popular options, but feel free to customize your cluster to your needs. For more information, check our [documentation](#).

Cloud Provider & Region

AWS, N. Virginia (us-east-1)

Create a free tier cluster by selecting a region with **FREE TIER AVAILABLE** and choosing the M0 cluster tier below.

★ Recommended region ⓘ

NORTH AMERICA	EUROPE	ASIA
N. Virginia (us-east-1) ★ FREE TIER AVAILABLE	Ireland (eu-west-1) ★ FREE TIER AVAILABLE	Singapore (ap-southeast-1) ★ FREE TIER AVAILABLE
Oregon (us-west-2) ★ FREE TIER AVAILABLE	Frankfurt (eu-central-1) ★ FREE TIER AVAILABLE	Mumbai (ap-south-1) FREE TIER AVAILABLE
AUSTRALIA		
Sydney (ap-southeast-2) ★ FREE TIER AVAILABLE		

FREE Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Back Create Cluster

Step 5: Select **M0 Sandbox** for cluster tier: Selecting M0 automatically locks the remaining configuration options. If you can't select the M0 cluster tier, return to the previous step and choose a Cloud Provider & Region that supports M0 Free Tier clusters.

Sydney (ap-southeast-2) ★
FREE TIER AVAILABLE

Cluster Tier M0 Sandbox (Shared RAM, 512 MB Storage) Encrypted

Base hourly rate is for a MongoDB replica set with 3 data bearing servers.

Shared Clusters for development environments and low-traffic applications

Tier	RAM	Storage	vCPU	Base Price
M0 Sandbox	Shared	512 MB	Shared	Free forever
M0 clusters are best for getting started, and are not suitable for production environments.				
500 max connections		Low network performance	100 max databases	500 max collections
M2	Shared	2 GB	Shared	\$8 / MONTH
M3	Shared	5 GB	Shared	\$20 / MONTH

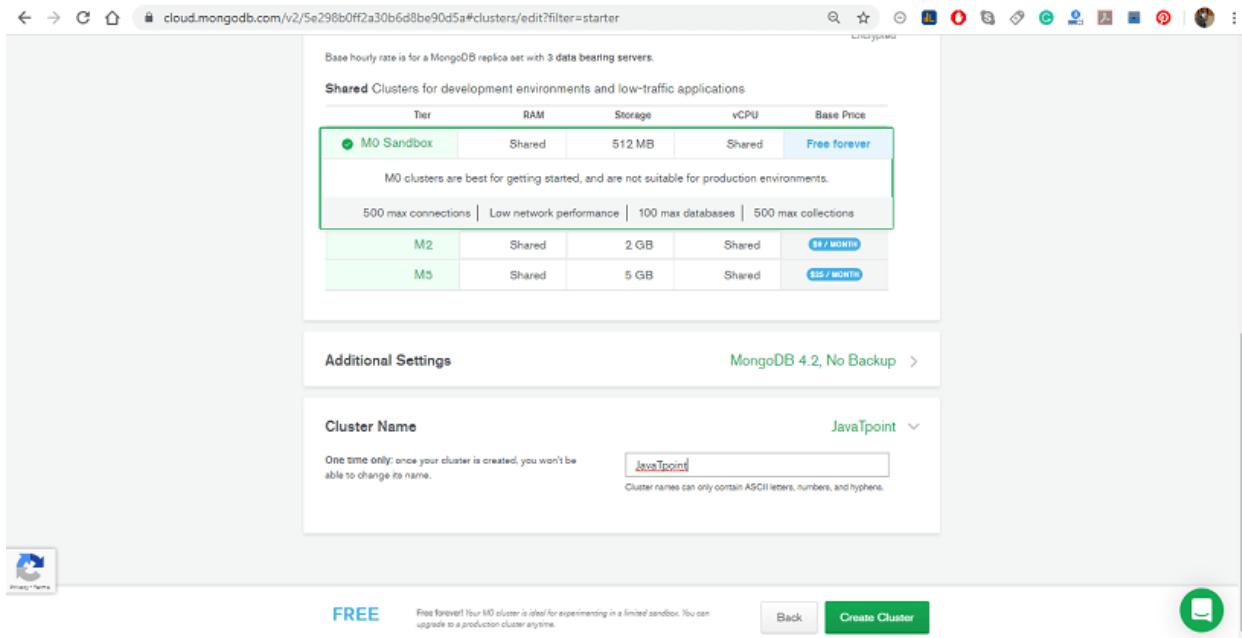
Additional Settings MongoDB 4.2, No Backup >

Cluster Name Cluster0 >

FREE Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

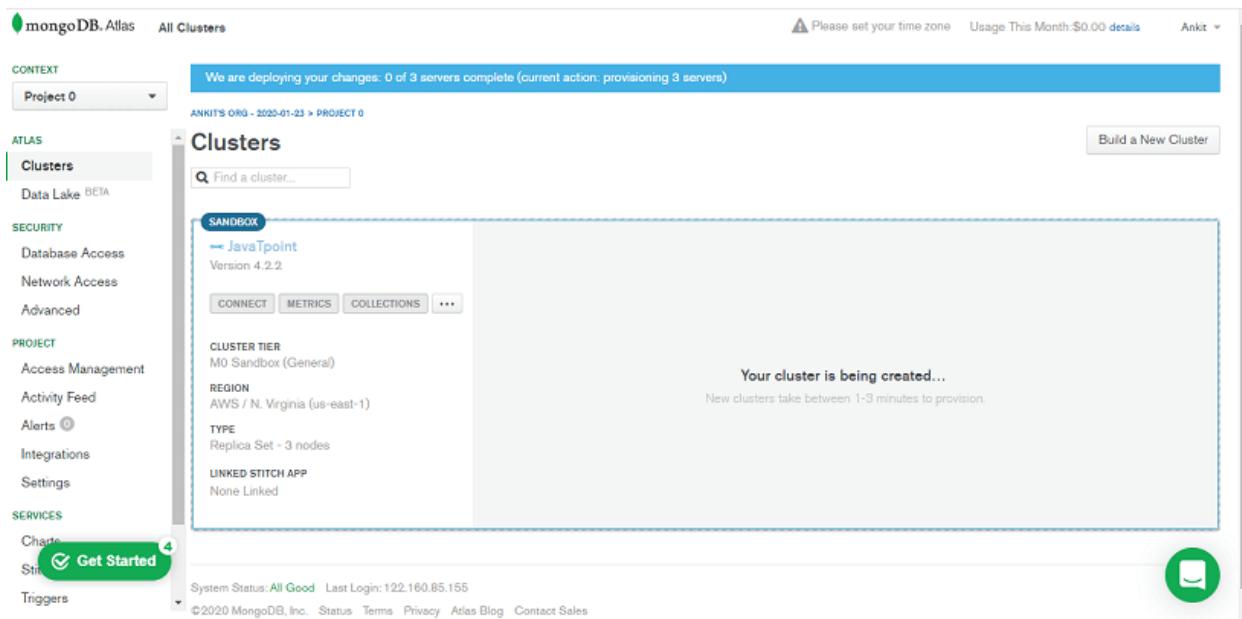
Back Create Cluster

Step 6: Enter a name for your cluster in the Cluster Name field; you can enter any name for your cluster. The cluster name contains ASCII letters, numbers, and hyphens.



The screenshot shows the MongoDB Atlas 'Create Cluster' interface. At the top, it says 'Shared Clusters for development environments and low-traffic applications'. Below this is a table with columns: Tier, RAM, Storage, vCPU, and Base Price. The 'M0 Sandbox' row is highlighted with a green border. The 'Base Price' column for M0 Sandbox is 'Free forever'. Below the table, it says 'M0 clusters are best for getting started, and are not suitable for production environments.' and lists '500 max connections | Low network performance | 100 max databases | 500 max collections'. Under 'Additional Settings', there's a dropdown for 'MongoDB 4.2, No Backup'. The 'Cluster Name' field is filled with 'JavaPoint'. A note below says 'One time only: once your cluster is created, you won't be able to change its name.' A 'FREE' badge indicates the cluster is free forever. At the bottom right are 'Back' and 'Create Cluster' buttons.

Step 7: Click on Create Cluster to deploy the cluster. Once you deploy your cluster, it can take up to 5-10 min for your cluster to provision and become ready to use.



The screenshot shows the MongoDB Atlas 'Clusters' page. On the left, a sidebar has 'Clusters' selected under 'ATLAS'. The main area shows a cluster named 'JavaPoint' in the 'Sandbox' tier. It lists 'Version 4.2.2', 'CONNECT', 'METRICS', 'COLLECTIONS', and an ellipsis button. To the right, a message says 'Your cluster is being created...' and 'New clusters take between 1-3 minutes to provision.' At the bottom, it shows 'System Status: All Good' and 'Last Login: 122.160.85.155'. The footer includes links for 'Status', 'Terms', 'Privacy', 'Atlas Blog', and 'Contact Sales'.

Step 8: Once we register, Atlas automatically creates a default organization and project where we can deploy our first cluster. We can add additional organizations and projects later.

The screenshot shows a step-by-step connection wizard for MongoDB Atlas. The title is "Connect to JavaTpoint". The steps are: "Setup connection security" (highlighted in blue), "Choose a connection method", and "Connect". A note says: "You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)". A callout box says: "You can't connect yet. Set up your firewall access and user security permission below." Step 1: "Whitelist your connection IP address" has two buttons: "Add Your Current IP Address" (green) and "Add a Different IP Address".

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Hardware Requirements

- Window 7 or above
- Processor Base Frequency 1.8 GHz or higher
- 2 GB Ram or higher
- 500 GB available disk space or more
- Minimum Screen Resolution: 1366x768

3.2 Software Requirements

- Nodejs 5.2 or higher version
- Visual Studio Code or other IDE for development
- Web Browser - Google Chrome/Mozilla Firefox/Internet Explorer

3.3 Introduction of Front-End

Front End and Back End: Frontend and Backend are the two most popular terms used in web development. These terms are very crucial for web development but are quite different from each other. Each side needs to communicate and operate effectively with the other as a single unit to improve the website's functionality.

Front End Development: The part of a website that the user interacts with directly is termed the front end. It is also referred to as the 'client side' of the application. It includes everything that users experience directly: text colors and styles, images, graphs and tables, buttons, colors, and navigation menu. HTML, CSS, and JavaScript are the languages used for Front End development. The structure, design, behavior, and content of everything seen on browser screens when websites, web applications, or mobile apps are opened up, is implemented by front End developers.

Responsiveness and performance are two main objectives of the Front End. The developer must ensure that the site is responsive i.e. it appears correctly on devices of all sizes no part of the website should behave abnormally irrespective of the size of the screen.

Front end Languages: The front-end portion is built by using some languages which are discussed below:

- **HTML:** HTML stands for Hypertext Markup Language. It is used to design the front-end portion of web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. The markup language is used to define the text documentation within the tag which defines the structure of web pages.
- **CSS:** Cascading Style Sheets fondly referred to as CSS is a simply designed language intended to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages. More importantly, CSS enables you to do this independent of the HTML that makes up each web page.
- **JavaScript:** JavaScript is a famous scripting language used to create magic on the sites to make the site interactive for the user. It is used to enhancing the functionality of a website to running cool games and web-based software.

There are many other languages through which one can do front-end development depending upon the framework for example *Flutter* uses *Dart*, *React* uses *JavaScript* and *Django* uses *Python*, and much more.

Front End Frameworks and Libraries:

- **AngularJS:** AngularJS is a JavaScript open-source front-end framework that is mainly used to develop single-page web applications(SPAs). It is a continuously growing and expanding framework which provides better ways for developing web applications. It changes the static HTML to dynamic HTML. It is an open-source project which can be free. It extends HTML attributes with Directives, and data is bound with HTML.
- **React.js:** React is a declarative, efficient, and flexible JavaScript library for building user interfaces. ReactJS is an open-source, component-based front-end library responsible only for the view layer of the application. It is maintained by Facebook.
Bootstrap: Bootstrap is a free and open-source tool collection for creating responsive websites and web applications. It is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.
- **jQuery:** jQuery is an open-source JavaScript library that simplifies the interactions between an HTML/CSS document, or more precisely the Document Object Model (DOM), and JavaScript. Elaborating the terms, jQuery simplifies HTML document traversing and manipulation, browser event handling, DOM animations, Ajax interactions, and cross-browser JavaScript development.
- **SASS:** It is the most reliable, mature, and robust CSS extension language. It is used to extend the functionality of an existing CSS of a site including everything from variables, inheritance, and nesting with ease.
- **Flutter:** Flutter is an open-source UI development SDK managed by google. It is powered by Dart programming language. It builds performant and good-looking natively compiled applications for mobile (Ios, Android), web, and desktop from a single code base. The key selling point of flutter is flat development is made easier, expressive, and flexible UI and native performance. In march 2021

- flutter announce Flutter 2 which upgrades flutter to build release applications for the web, and the desktop is in beta state.
- Some other libraries and frameworks are Semantic-UI, Foundation, Materialize, Backbone.js, Ember.js, etc.

3.4 Introduction of Back-End

Backend Development: Backend is the server-side of the website. It stores and arranges data, and also makes sure everything on the client-side of the website works fine. It is the part of the website that you cannot see and interact with. It is the portion of software that does not come in direct contact with the users. The parts and characteristics developed by backend designers are indirectly accessed by users through a front-end application. Activities, like writing APIs, creating libraries, and working with system components without user interfaces or even systems of scientific programming, are also included in the backend.

Back end Languages: The back end portion is built by using some languages which are discussed below:

- **PHP:** PHP is a server-side scripting language designed specifically for web development. Since PHP code executed on the server-side, so it is called a server-side scripting language.
- **C++:** It is a general-purpose programming language and widely used nowadays for competitive programming. It is also used as a backend language.
- **Java:** Java is one of the most popular and widely used programming languages and platforms. It is highly scalable. Java components are easily available.
- **Python:** Python is a programming language that lets you work quickly and integrate systems more efficiently.
- **JavaScript:** JavaScript can be used as both (front end and back end) programming languages.

- **Node.js:** Node.js is an open-source and cross-platform runtime environment for executing JavaScript code outside a browser. You need to remember that NodeJS is not a framework, and it's not a programming language. Most people are confused and understand it's a framework or a programming language. We often use Node.js for building back-end services like APIs like Web App or Mobile App. It's used in production by large companies such as Paypal, Uber, Netflix, Walmart, and so on.

Back End Frameworks:

- The list of back-end frameworks are: Express, Django, Rails, Laravel, Spring, etc.
- The other back-end program/scripting languages are C#, Ruby, REST, GO, etc.

CHAPTER 4

SYSTEM ANALYSIS

4.1 Introduction

The entire project has been developed keeping in view of the distributed client server computing technology, in mind. The specification have been normalized up to 3NF to eliminate all the anomalies that may arise due to the database transaction that are executed by the general users and the organizational administration. The user interfaces are browser specific to give distributed accessibility for the overall system. The internal database has been selected as MongoDB. The basic constructs of table spaces, clusters and indexes have been exploited to provide higher consistency and reliability for the data storage. The MongoDB was a choice as it provides the constructs of high-level reliability and security. The total front end was dominated using the c#.net technologies. At all proper levels high care was taken to check that the system manages the data consistency with proper business rules or validations. The authentication and authorization was crosschecked at all the relevant stages.

4.2 Requirement analysis and specification

Requirements analysis involves frequent communication with the system users to determine specific feature exceptions, resolution of conflict or ambiguity in requirements as demand by the various users or groups of users, avoidance of feature creep and documentation of all aspects of the project development process from start to finish. Energy should be directed towards ensuring tat the final system or product conforms to client needs rather than attempting to meld user expectations to fit the requirements. Requirements analysis is a team effort that demands a combination of hardware, software and human engineering expertise as well as skills in dealing with people. Requirements of this project were to secure the user data from unauthorized users with respect to read and writes. Beside the core process, it also should have a very user-friendly interface to handle all the operations of the project very easily.

4.2.1 Functional Requirement

Functional requirements define what a system is supposed to do. It defines the function of a system and its components. A function is described as a set of inputs, the behavior and outputs.

The system must provide following functionalities:

- **Admin**
 - See all registered request from hospitals
 - Active their login functionality
 - Deactivate any hospital
- **Hospital**
 - Verify OTP generated by patient / user
 - Register / admit patient
 - Discharge patient
 - Enter hospital profile
 - Add / delete / edit hospital's wards
 - Add / delete / edit hospital's doctors
 - Add / delete / edit hospital's beds
- **User / patient**
 - Search nearest hospitals list
 - Show map
 - Show wards availability in that hospital
 - Show doctors availability in that hospital
 - Show beds availability in that hospital
 - Book bed (hold for 2 hours)

4.2.2 Non-Functional Requirement

Non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions.

4.2.2.1 Performance Requirement

Response Time:

System should provide quick response while generation of the report as well as while modifying the data.

Robustness:

System should be able to cope with errors during execution and cope with erroneous input.

Data Integrity:

System should provide consistent data insertion.

4.2.2.1 Software Requirement

Adaptability:

Software should be adaptable on different operating system.

Availability:

System should be available 24 x 7.

Backup/Recovery:

Backup/Recovery of software is provided.

Maintenance: It's easy to maintain and affordable.

4.3 Technical Feasibility

The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organisation and their applicability to the expected needs of the proposed system.

It is an evolution of the hardware and software and how it meets the need of the proposed system. So, after being ensured, we can say yes, the project is technically feasible. Because in development of the project, core concepts of TypeScript and JavaScript are used. Angular framework is licence free, platform independent and the project can be completed within limited resources.

Technical feasibility also performs the following tasks.

- Analyzes the technical skills and capabilities of the software development team members
- Determines whether the relevant technology is stable and established
- Ascertains that the technology chosen for software development has a large number of users so that they can be consulted when problems arise or improvements are required.

4.4 Operational Feasibility

Operational feasibility is a measure of how well a proposed system solves the problems and takes advantage of the opportunities identified during scope definition and how it specifies the requirements identified in the requirement analysis phase of system development.

The operations like registration of new user, management of categories, sub categories, users, management of bidding and advertising and receipt generation, all such modules work properly and produce the desired result accurately without any ambiguities.

Operational feasibility also performs the following tasks.

- Determines whether the problems anticipated in user requirements are of high priority
- Determines whether the solution suggested by the software development team is acceptable
- Analyzes whether users will adapt to a new software
- Determines whether the organization is satisfied by the alternative solutions proposed by the software development team

4.5 Economic Feasibility

This study is carried out to check the economic impact will have on the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products have to be purchased.

4.5 System Planning

Planning is the act of thinking about actions before they are carried out. Specific details are decided before carrying out the plan, as well as organizing the steps of the plan.

4.5.1 Software Project Planning

The software project management process begins with a set of activities that are collectively called project planning that involves estimation. Software cost and effort estimation will never become an exact but can be transformed from indistinguishable to a series of systematic steps. Following things have been estimated before the software development:

Project Complexity:

It has strong effort on uncertainty that is inherent in planning. Our project belongs to the Category of evolutionary project, as the requirements are very large. An expert team usually develops such system.

Project Size:

It is another factor that can affect the accuracy of estimates. As the project size increases; the Interdependency among various elements of the software grows rapidly. Therefore, it is essential to estimate the project size in lines of code.

Structural Uncertainty:

The structure refers to the degree to which requirements have been solidified, the ease with which functions can be compartmentalized, and the hierarchical nature of information that must be processed.

Risk:

Risk is measured by the degree by the degree of certainty in the quantitative estimates established for resources, cost and schedule.

CHAPTER 5

SYSTEM DESIGN

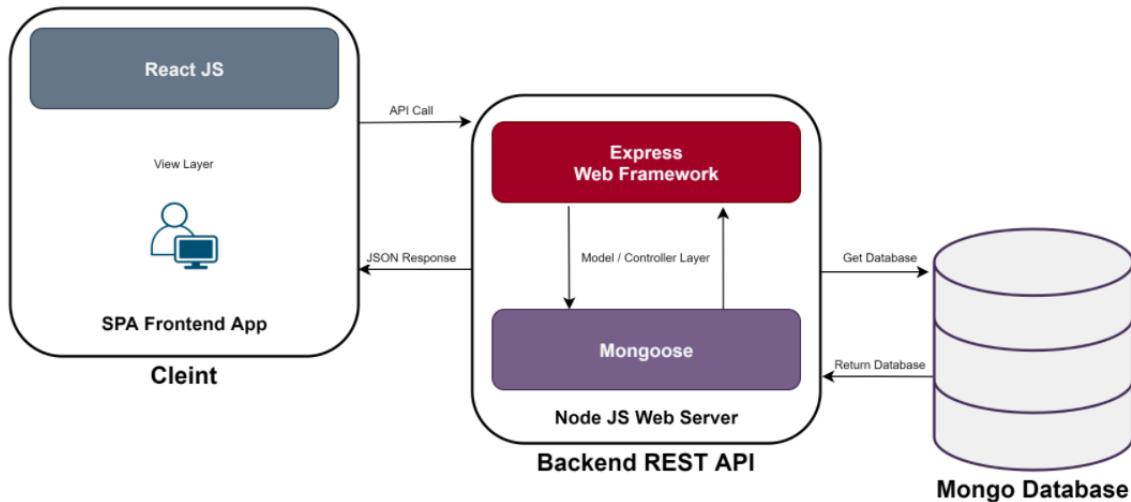
5.1 Introduction

Design is a meaningful engineering representation of something i.e. to be built. It can be traced to the customer requirement and at the same asset for quality against a set of predefines criteria. People tend to frequent seized property auctions because goods can be purchased at a low price relative to market value. This is good for the buyer, but not for the debtor whose budget and credit record are dependent on this money. A limited audience who will not pay the full value of the items up for bidding attends the auctions. The body appointed to seize and auction property operates it.

When debtors must liquidate their assets, they must first open an account with the Collection Authority. The Collection Authority is housed inside the sheriff's office or the organization responsible for public auctions of seized property and assets. It is a combination of customer service counters and an inventory/shipping warehouse. Seized goods and items brought in by a debtor are inventoried at the Collection Authority where daily shipments are made for the items sold. Once the item is sold, the buyer makes the payment to the Collection Authority, which then transfers the balance owed to the creditor, minus shipping costs and a percentage charged by the Collection Authority.

All items going through site , either seized or brought in by the debtor, use the Item Profiler to create a standard description page to post items on the on-line auction site. The page layout includes digital images representing the orthographic views of the item plus a text section detailing its make, model, year, history, etc. with an additional line worded by the debtor.

5.2 Architecture Design



5.2.1 What is MERN Stack?

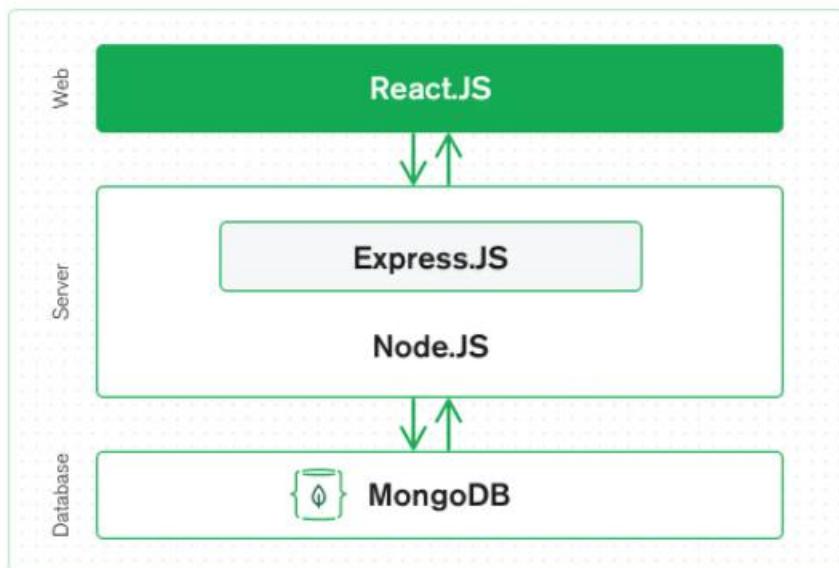
MERN stands for MongoDB, Express, React, Node, after the four key technologies that make up the stack.

- MongoDB - document database
- Express(.js) - Node.js web framework
- React(.js) - a client-side JavaScript framework
- Node(.js) - the premier JavaScript web server

MERN is one of several variations of the MEAN stack (MongoDB Express Angular Node), where the traditional Angular.js frontend framework is replaced with React.js. Other variants include MEVN (MongoDB, Express, Vue, Node), and really any frontend JavaScript framework can work.

Express and Node make up the middle (application) tier. Express.js is a server-side web framework, and Node.js the popular and powerful JavaScript server platform. Regardless of which variant you choose, ME(RVA)N is the ideal approach to working with JavaScript and JSON, all the way through.

5.2.2 How does MERN Stack work?



React.js Front End

The top tier of the MERN stack is React.js, the declarative JavaScript framework for creating dynamic client-side applications in HTML. React lets you build up complex interfaces through simple Components, connect them to data on your backend server, and render them as HTML.

React's strong suit is handling stateful, data-driven interfaces with minimal code and minimal pain, and it has all the bells and whistles you'd expect from a modern web framework: great support for forms, error handling, events, lists, and more.

Express.js and Node.js Server Tier

The next level down is the Express.js server-side framework, running inside a Node.js server. Express.js bills itself as a “fast, unopinionated, minimalist web framework for Node.js,” and that is indeed exactly what it is. Express.js has powerful models for URL routing (matching an incoming URL with a server function), and handling HTTP requests and responses.

By making XML HTTP Requests (XHRs) or GETs or POSTs from your React.js front-end, you can connect to Express.js functions that power your application. Those functions in turn use MongoDB's Node.js drivers, either via callbacks for using Promises, to access and update data in your MongoDB database.

MongoDB Database Tier

If your application stores any data (user profiles, content, comments, uploads, events, etc.), then you're going to want a database that's just as easy to work with as React, Express, and Node.

That's where MongoDB comes in: JSON documents created in your React.js front end can be sent to the Express.js server, where they can be processed and (assuming they're valid) stored directly in MongoDB for later retrieval. Again, if you're building in the cloud, you'll want to look at Atlas. If you're looking to set up your own MERN stack, read on!

Is MERN a full-stack solution?

Yes, MERN is a full-stack, following the traditional 3-tier architectural pattern, including the front-end display tier (React.js), application tier (Express.js and Node.js), and database tier (MongoDB).

Why choose the MERN stack?

Let's start with MongoDB, the document database at the root of the MERN stack. MongoDB was designed to store JSON data natively (it technically uses a binary version of JSON called BSON), and everything from its command line interface to its query language (MQL, or MongoDB Query Language) is built on JSON and JavaScript.

MongoDB works extremely well with Node.js, and makes storing, manipulating, and representing JSON data at every tier of your application incredibly easy. For cloud-native applications, MongoDB Atlas makes it even easier, by giving you an auto-scaling MongoDB cluster on the cloud provider of your choice, as easy as a few button clicks.

Express.js (running on Node.js) and React.js make the JavaScript/JSON application MERN full stack, well, full. Express.js is a server-side application framework that wraps HTTP requests and responses, and makes it easy to map URLs to server-side functions. React.js is a frontend JavaScript framework for building interactive user interfaces in HTML, and communicating with a remote server.

The combination means that JSON data flows naturally from front to back, making it fast to build on and reasonably simple to debug. Plus, you only have to know one programming language, and the JSON document structure, to understand the whole system!

MERN is the stack of choice for today's web developers looking to move quickly, particularly for those with React.js experience.

CHAPTER 6

METHODOLOGY

6.1 Methodology Flow

Process flows (or process diagrams) provide a visual overview or workflow diagram of all the tasks and relationships involved in a process.

The Purpose of Process Flows

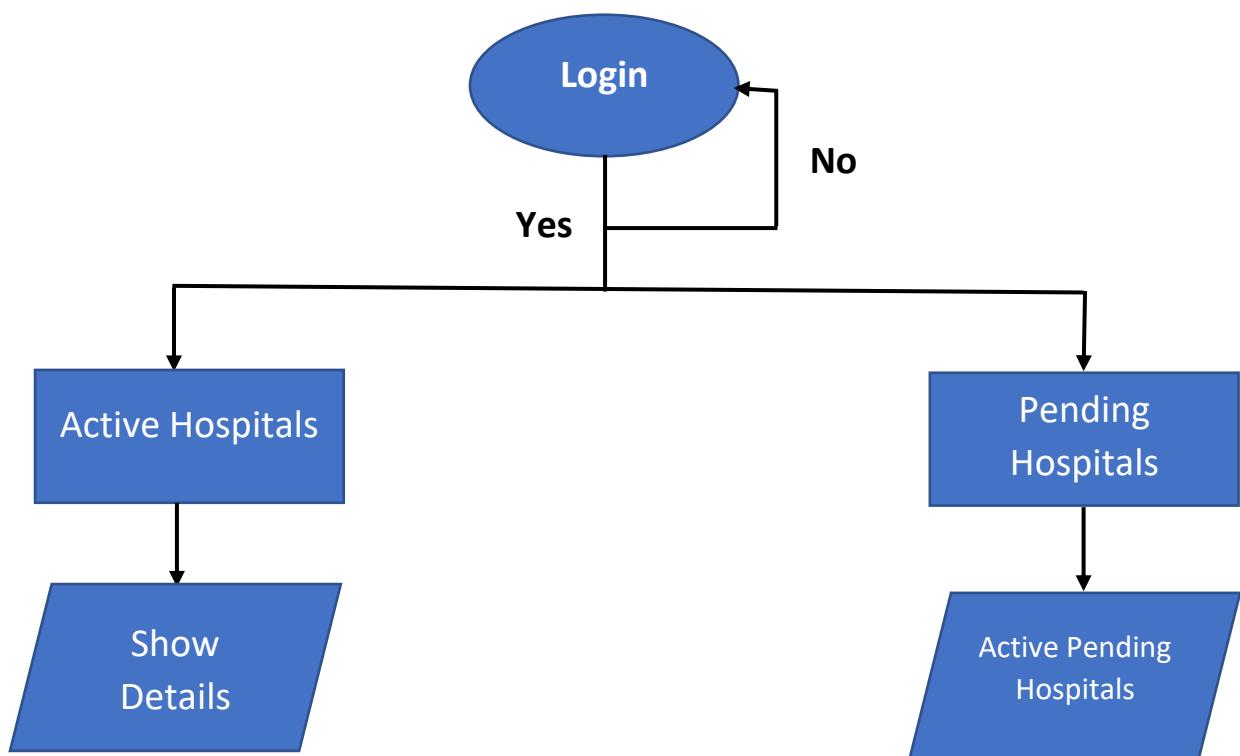
The goal of illustrating a process could be two-fold:

- To explain how a process works
- To improve a process

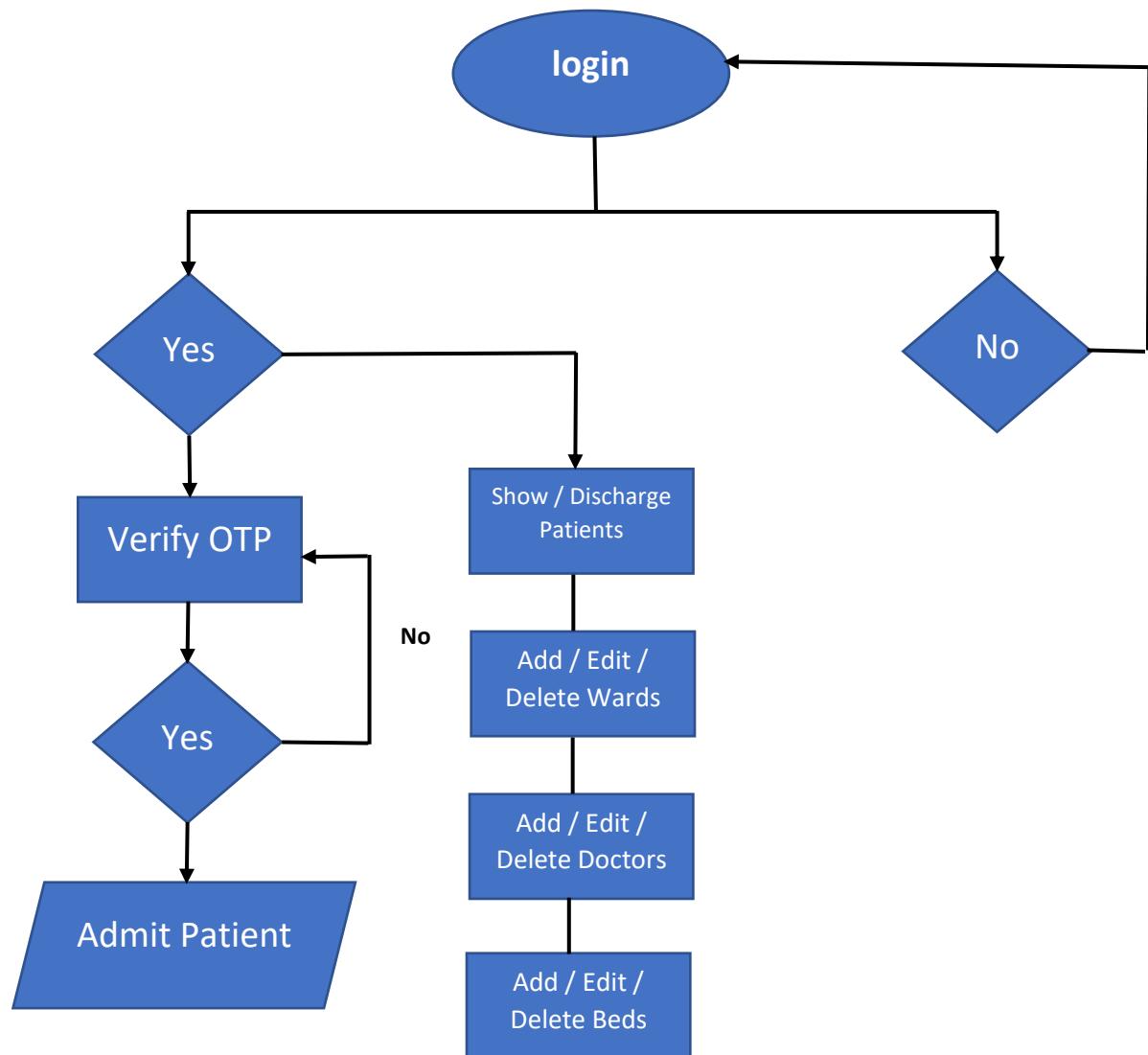
A flowchart is a picture of the separate steps of a process in sequential order. It is a generic tool that can be adapted for a wide variety of purposes, and can be used to describe various processes, such as a manufacturing process, an administrative or service process, or a project plan. It's a common process analysis tool and one of the seven basic quality tools.

Elements that may be included in a flowchart are a sequence of actions, materials or services entering or leaving the process (inputs and outputs), decisions that must be made, people who become involved, time involved at each step, and/or process measurements.

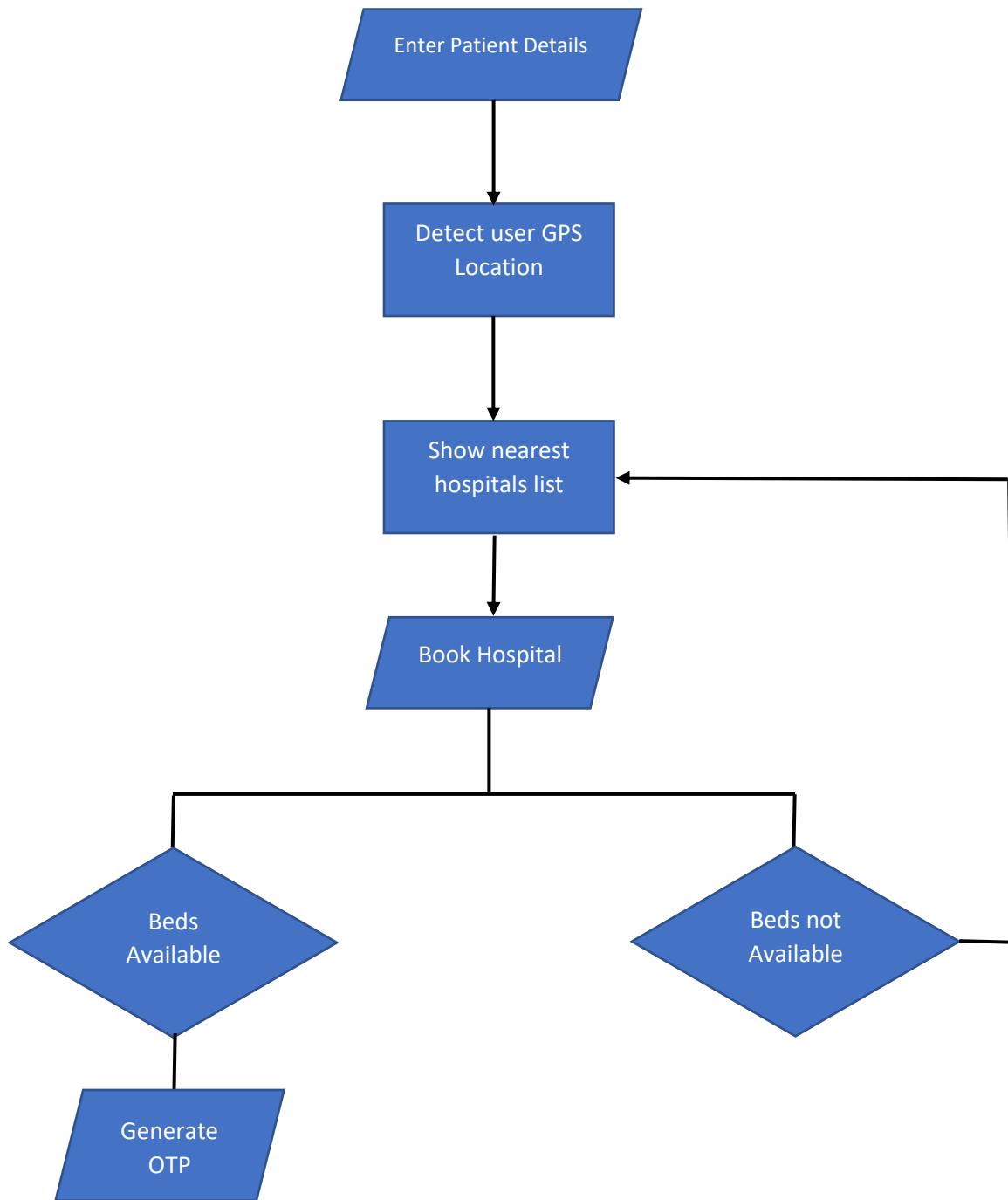
6.1 Admin Flow



6.2 Hospital Flow



6.3 Patient / User Flow



6.1 Front-End Development

React Js

React.js is an open-source JavaScript library that is used for building user interfaces specifically for single-page applications. It's used for handling the view layer for web and mobile apps. React also allows us to create reusable UI components. React was first created by Jordan Walke, a software engineer working for Facebook. React first deployed on Facebook's newsfeed in 2011 and on Instagram.com in 2012.

React allows developers to create large web applications that can change data, without reloading the page. The main purpose of React is to be fast, scalable, and simple. It works only on user interfaces in the application. This corresponds to the view in the MVC template. It can be used with a combination of other JavaScript libraries or frameworks, such as Angular JS in MVC.

React.js properties include the following

React.js is declarative

React.js is simple

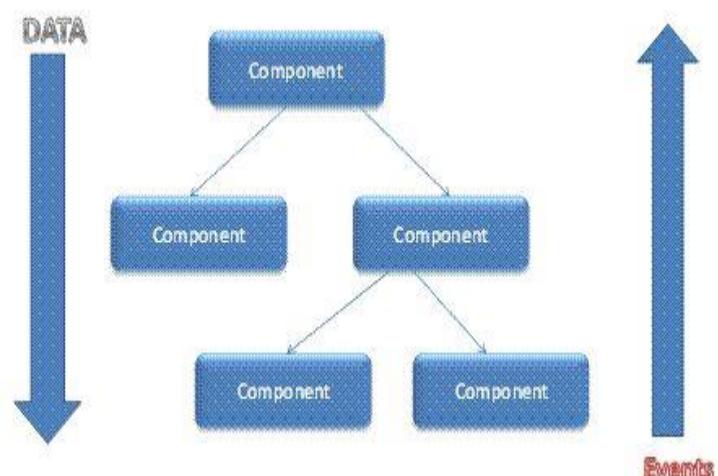
React.js is component based

React.js supports server side

React.js is extensive

React.js is fast

React.js is easy to learn



Components

```
└─ components
    └─ AdminComponents
        └─ AllHospitals.js
        └─ PendingHospitals.js
        └─ sidebar.css
        └─ Sidebar.js
    └─ HospitalComponents
        └─ AllAdmitPatients.js
        └─ Beds.js
        └─ DisplayWards.js
        └─ DisplayBeds.js
        └─ DisplayDoctors.js
        └─ Doctors.js
        └─ HospitalDataShow.js
        └─ HospitalDetails.js
        └─ HospitalNotConfirm.js
        └─ OtpVerify.js
        └─ RequestForms.js
        └─ sidebar.css
        └─ Sidebar.js
        └─ test.js
        └─ UserForm.js
        └─ Wards.js
```

Admin Components

All Hospitals:

```
import { Button, Table, Modal} from 'react-bootstrap';
import {useEffect, React, useState} from 'react';

function AllHospitals() {

    //Model open
    const [show, setShow] = useState(false);
    const handleClose = () => setShow(false);
    const handleShow = () => setShow(true);

    //Model close

    const [hospitalId, setHospitalId] = useState()
    const [stateHospital, setHospitals] = useState([])
    const [stateSingleHospital, setSingleHospitals] = useState({})

    useEffect(() => {
        fetchData()
    },[hospitalId,stateHospital])

    useEffect(() => {
        fetchSingleHospital()
    },[hospitalId])

    function fetchData(){
        const requestBody = {
            query: `query
{
    getActiveHospital
    {
        _id
    }
}
`
```

```

        hospitalName
        hospitalRegistrationNo
        hospitalType
        government
        address
        state
        district
        pincode
        website
        longitude
        latitude
        ownerName
        ownerContactNo
        ownerEmail
        password
        status
    }
}
` 
};

fetch('http://localhost:4000/graphql',{
  method: 'POST',
  body: JSON.stringify(requestBody),
  headers: {
    'Content-Type' : 'application/json'
  }
})
.then(res=> {
  if(res.status !==200 && res.status !== 201) {
    throw new Error ('Failed');
  }
  return res.json();
})
.then(resData => {
  //console.log(resData);
  const fetchHospitals = resData.data.getActiveHospital;
  setHospitals(fetchHospitals);
  console.log(fetchHospitals)
})
.catch(err => {
  console.log(err);
});

```

```
}

function fetchSingleHospital(){
  const requestBody = {
    query: `query
{
  getSingleHospital(id:"${hospitalId}") {
    hospitalName
    hospitalRegistrationNo
    hospitalType
    government
    address
    state
    district
    pincode
    website
    longitude
    latitude
    ownerName
    ownerContactNo
    ownerEmail
    status
  }
}
`,

};

fetch('http://localhost:4000/graphql',{
  method: 'POST',
  body: JSON.stringify(requestBody),
  headers: {
    'Content-Type' : 'application/json'
  }
})
.then(res=> {
  if(res.status !==200 && res.status !== 201) {
    throw new Error ('Failed');
  }
  return res.json();
})
.then(resData => {
```

```

        //console.log(resData);
        const fetchSingleHospitals = resData.data.getSingleHospital;
        setSingleHospitals(fetchSingleHospitals);
        console.log(fetchSingleHospitals)
    })
    .catch(err => {
        console.log(err);
    })
}

console.log("stateSingle", stateSingleHospital)

return (
    <div>
        <center><h2>All Active Hospitals</h2></center>
        <hr></hr>
        <Table bordered hover>
            <thead>
                <tr>
                    <th>Hospital Name</th>
                    <th>Registration No</th>
                    <th>Type</th>
                    <th>Owner Name</th>
                    <th>Show Details</th>
                </tr>
            </thead>
            <tbody>
                {
                    stateHospital.map(hos =>
                        <tr key={hos._id}>
                            <td>{hos.hospitalName}</td>
                            <td>{hos.hospitalRegistrationNo}</td>
                            <td>{hos.hospitalType}</td>
                            <td>{hos.ownerName}</td>
                            <td><Button variant="warning" size="sm" onClick={()=>{setHospitalId(hos._id); handleShow()}}>Show Details</Button></td>
                        </tr>
                    )
                }
            </tbody>
    </div>
)

```

```

        </Table>

<Modal
  show={show} onHide={handleClose}
  dialogClassName="my-modal"
  size='lg'
>
  <Modal.Header closeButton>
    <Modal.Title>Hospital Details</Modal.Title>
  </Modal.Header>
  <Modal.Body>

    <Table bordered hover>
      <tbody>
        <tr>
          <td><strong>Hospital Name:</strong></td>
          <td>{stateSingleHospital.hospitalName}</td>
        </tr>

        <tr>
          <td><strong>Hospital Registration No:</strong></td>
          <td>{stateSingleHospital.hospitalRegistrationNo}</td>
        </tr>

        <tr>
          <td><strong>Hospital Type:</strong></td>
          <td>{stateSingleHospital.hospitalType}</td>
        </tr>

        <tr>
          <td><strong>Hospital Government:</strong></td>
          <td>{stateSingleHospital.government}</td>
        </tr>

        <tr>
          <td><strong>Hospital Address:</strong></td>
          <td>{stateSingleHospital.address}</td>
        </tr>

        <tr>
          <td><strong>Hospital State:</strong></td>
          <td>{stateSingleHospital.state}</td>
        </tr>
      </tbody>
    </Table>
  </Modal.Body>
</Modal>

```

```

        <tr>
            <td><strong>Hospital District:</strong></td>
            <td>{stateSingleHospital.district}</td>
        </tr>

        <tr>
            <td><strong>Hospital Pincode:</strong></td>
            <td>{stateSingleHospital.pincode}</td>
        </tr>

        <tr>
            <td><strong>Hospital Website:</strong></td>
            <td><a href={stateSingleHospital.website} target="_
blank">{stateSingleHospital.website}</a></td>
        </tr>
        <tr>
            <td><strong>Owner Name:</strong></td>
            <td>{stateSingleHospital.ownerName}</td>
        </tr>
        <tr>
            <td><strong>Owner Contact:</strong></td>
            <td>{stateSingleHospital.ownerContactNo}</td>
        </tr>
        <tr>
            <td><strong>Owner Email:</strong></td>
            <td>{stateSingleHospital.ownerEmail}</td>
        </tr>
        <tr>
            <td><strong>Hospital Status:</strong></td>
            <td>{stateSingleHospital.status}</td>
        </tr>
    </tbody>
</Table>

</Modal.Body>
<Modal.Footer>
    <Button variant="secondary" onClick={handleClose}>
        Close
    </Button>
    {/* <Button variant="primary" onClick={handleClose}>
        Save Changes
    </Button> */}
</Modal.Footer>
</Modal>

```

```
        </div>

    )

}

export default AllHospitals;
```

Pending Hospitals:

```
import { Button, Table, Modal} from 'react-bootstrap';
import {useEffect, React, useState, prevState} from 'react';

function PendingHospitals() {

    //Model open
    const [show, setShow] = useState(false);
    const handleClose = () => setShow(false);
    const handleShow = () => setShow(true);

    //Model close

    const [hospitalId, setHospitalId] = useState()

    const [hospitalIdConfirm, setHospitalIdConfirm] = useState(null)
    const [pendingHospital, setPendingHospitals] = useState(0)

    const [stateHospital, setHospitals] = useState([])

    const [stateSingleHospital, setSingleHospitals] = useState({})

    useEffect(() => {
        fetchPendingData()
    },[hospitalIdConfirm,pendingHospital])
```

```
useEffect(() => {
  fetchSingleHospital()
},[hospitalId])

function fetchPendingData(){
  const requestBody = {
    query: `query
{
  getPendingHospitals
  {
    _id
    hospitalName
    hospitalRegistrationNo
    hospitalType
    ownerName
    status
  }
}
`,

};

fetch('http://localhost:4000/graphql',{
  method: 'POST',
  body: JSON.stringify(requestBody),
  headers: {
    'Content-Type' : 'application/json'
  }
})
.then(res=> {
  if(res.status !==200 && res.status !== 201) {
    throw new Error ('Failed');
  }
  return res.json();
})
.then(resData => {
  //console.log(resData);
  const fetchPendingHospitals = resData.data.getPendingHospitals;
```

```

        setHospitals(fetchPendingHospitals);
        console.log("Fetch Pending",fetchPendingHospitals)
    })
    .catch(err => {
        console.log(err);
    });
}

function fetchSingleHospital(){
    const requestBody = {
        query: `query
        {
            getSingleHospital(id:"${hospitalId}"{
                hospitalName
                hospitalRegistrationNo
                hospitalType
                government
                address
                state
                district
                pincode
                website
                longitude
                latitude
                ownerName
                ownerContactNo
                ownerEmail
                status
            }
        }
    `;
};

fetch('http://localhost:4000/graphql',{
    method: 'POST',
    body: JSON.stringify(requestBody),
    headers: {
        'Content-Type' : 'application/json'
    }
})
.then(res=> {

```

```

        if(res.status !==200 && res.status !== 201) {
            throw new Error ('Failed');
        }
        return res.json();
    })
    .then(resData => {
        //console.log(resData);
        const fetchSingleHospitals = resData.data.getSingleHospital;
        setSingleHospitals(fetchSingleHospitals);
        console.log(fetchSingleHospitals)
    })
    .catch(err => {
        console.log(err);
    });
}

function confirmHospital(e){
    const requestBody = {
        query: `mutation
        {
            updateStatus(id:"${e}",status:"Active"){
                _id
                hospitalName
                status
            }
        }
        `;
    };

    fetch('http://localhost:4000/graphql',{
        method: 'POST',
        body: JSON.stringify(requestBody),
        headers: {
            'Content-Type' : 'application/json'
        }
    })
    .then(res=> {
        if(res.status !==200 && res.status !== 201) {
            throw new Error ('Failed');
        }
        return res.json();
    })
    .then(resData => {

```

```

        const updateValue = resData.data.getPendingHospitals;
        setPendingHospitals(pendingHospital+1);
        console.log("Pending",resData);
    })
    .catch(err => {
        console.log(err);
    });
    console.log(e)

}

return (
    <>
    <center><h2>All Active Hospitals</h2></center>
    <hr></hr>
    <Table bordered hover>
        <thead>
            <tr>
                <th>Hospital Name</th>
                <th>Registration No</th>
                <th>Type</th>
                <th>Owner Name</th>
                <th>Status</th>
                <th>Show Details</th>
                <th>Confirm</th>
            </tr>
        </thead>
        <tbody>
            {
                stateHospital.map(hos =>
                    <tr key={hos._id}>
                        <td>{hos.hospitalName}</td>
                        <td>{hos.hospitalRegistrationNo}</td>
                    >
                        <td>{hos.hospitalType}</td>
                        <td>{hos.ownerName}</td>
                        <td>{hos.status}</td>
                        <td><Button variant="warning" size="sm" onClick={()=>{setHospitalId(hos._id); handleShow()}}>Show Details</Button></td>
                )
            }
        </tbody>
    </Table>
)

```

```

                <td><Button variant="primary" size="sm" onClick={(e)=> confirmHospital(hos._id)}>Confirm</Button></td>
            </tr>
        )
    }
</tbody>
</Table>

<Modal
show={show} onHide={handleClose}
dialogClassName="my-modal"
size='lg'
>
<Modal.Header closeButton>
    <Modal.Title>Hospital Details</Modal.Title>
</Modal.Header>
<Modal.Body>

<Table bordered hover>
    <tbody>
        <tr>
            <td><strong>Hospital Name:</strong></td>
            <td>{stateSingleHospital.hospitalName}</td>
        </tr>

        <tr>
            <td><strong>Hospital Registration No:</strong></td>
            <td>{stateSingleHospital.hospitalRegistrationNo}</td>
        </tr>

        <tr>
            <td><strong>Hospital Type:</strong></td>
            <td>{stateSingleHospital.hospitalType}</td>
        </tr>

        <tr>
            <td><strong>Hospital Government:</strong></td>
            <td>{stateSingleHospital.government}</td>
        </tr>

        <tr>
            <td><strong>Hospital Address:</strong></td>

```

```

        <td>{stateSingleHospital.address}</td>
    </tr>

    <tr>
        <td><strong>Hospital State:</strong></td>
        <td>{stateSingleHospital.state}</td>
    </tr>

    <tr>
        <td><strong>Hospital District:</strong></td>
        <td>{stateSingleHospital.district}</td>
    </tr>

    <tr>
        <td><strong>Hospital Pincode:</strong></td>
        <td>{stateSingleHospital.pincode}</td>
    </tr>

    <tr>
        <td><strong>Hospital Website:</strong></td>
        <td><a href={stateSingleHospital.website}>{stateSingleHospital.website}</a></td>
    </tr>
    <tr>
        <td><strong>Owner Name:</strong></td>
        <td>{stateSingleHospital.ownerName}</td>
    </tr>
    <tr>
        <td><strong>Owner Contact:</strong></td>
        <td>{stateSingleHospital.ownerContactNo}</td>
    </tr>
    <tr>
        <td><strong>Owner Email:</strong></td>
        <td>{stateSingleHospital.ownerEmail}</td>
    </tr>
    <tr>
        <td><strong>Hospital Status:</strong></td>
        <td>{stateSingleHospital.status}</td>
    </tr>
</tbody>
</Table>

</Modal.Body>
<Modal.Footer>
    <Button variant="secondary" onClick={handleClose}>

```

```
        Close
      </Button>
      {/* <Button variant="primary" onClick={handleClose}>
        Save Changes
      </Button> */}
    </Modal.Footer>
  </Modal>
</>

)

}

export default PendingHospitals;
```

Pages:



Admin Dashboard

```
import {React} from 'react';
import {Container, Row, Col} from 'react-bootstrap'
import Sidebar from '../../components/AdminComponents/Sidebar'
import {Redirect, useHistory} from 'react-router-dom';
import NavbarMenu from '../../components/NavbarMenu'
import jwt from 'jsonwebtoken'

function AdminDashboard() {
    const history = useHistory()
    const AdminToken = localStorage.getItem('token')

    if(!AdminToken){
        return <Redirect to = "/admin_login" />
    }

    jwt.verify(AdminToken, 'superAdminSecretKey', function(err, decoded) {
        if (err) {
            history.push("/logout")
        }
    });

    var decoded = jwt.verify(AdminToken, 'superAdminSecretKey');
    return (
        <>
        <NavbarMenu />
        <Container>
            <Row>
                <Col>
                    <h1 style={{textAlign:'center',marginTop:'30px'}}>Admin Dashboard</h1>
                    <h3 style={{textAlign:'center',marginTop:'30px'}}>Welcome <strong>{decoded.loginId}</strong></h3>
                </Col>
            </Row>
        </Container>

        <Container fluid>
            <Sidebar />
        </Container>
    </>
)
```

```
        )
    }

export default AdminDashboard;
```

Hospital Dashboard

```
import { React, useState, useEffect } from 'react'
import { Container, Row, Col } from 'react-bootstrap'
import Sidebar from '../../../../../components/HospitalComponents/Sidebar'
import NavbarMenu from '../../../../../components/NavbarMenu';
import HospitalNotConfirm from '../../../../../components/HospitalComponents/HospitalNotConfirm';
import { Redirect, useHistory } from 'react-router-dom';
import jwt from 'jsonwebtoken'

function HospitalDashboard() {

    const history = useHistory()
    const HosToken = localStorage.getItem('hosToken')
    const [hospitalById, setHospitalById] = useState({});

    useEffect(() => {
        fetchSingleHospital()
    }, [])

    if (!HosToken) {
        return <Redirect to="/hospital_login" />
    }

    jwt.verify(HosToken, 'hospitalSecretKey', function (err, decoded) {
        if (err) {
            history.push("/hospitalLogout")
        }
    });
}
```

```

var decoded = jwt.verify(HosToken, 'hospitalSecretKey');
if (decoded) {
    localStorage.setItem('hospitalIdByToken', decoded.hosUserId)
}
const HosTokenById = localStorage.getItem('hospitalIdByToken')

function fetchSingleHospital() {
    const requestBody = {
        query: `
            query {
                getSingleHospital(id:"${HosTokenById}") {
                    _id
                    hospitalName
                    hospitalRegistrationNo
                    hospitalType
                    government
                    address
                    state
                    district
                    pincode
                    website
                    longitude
                    latitude
                    ownerName
                    ownerContactNo
                    ownerEmail
                    status
                }
            }
        `
    };
}

fetch('http://localhost:4000/graphql', {
    method: 'POST',
    body: JSON.stringify(requestBody),
    headers: {
        'Content-Type': 'application/json'
    }
})
    .then(res => {

```

```

        if (res.status !== 200 && res.status !== 201) {
            throw new Error('Failed');
        }
        return res.json();
    })
    .then(resData => {
        const fetchHospital = resData.data.getSingleHospital;
        setHospitalById(fetchHospital);
        console.log("Hospital", fetchHospital)
    })
    .catch(err => {
        console.log(err);
    })
}

console.log(hospitalById.status)
return (
    <>
        <NavbarMenu />
        <Container>
            <Row>
                <Col>
                    <h1 style={{ textAlign: 'center', marginTop: '30px' }}>Ho
spital Dashboard</h1>
                </Col>
            </Row>
        </Container>

        <Container fluid>
        {
            hospitalById.status === "Pending" ? <HospitalNotConfirm /> : <Sid
ebar />
        }
        </Container>
    </>
)
}

export default HospitalDashboard;

```

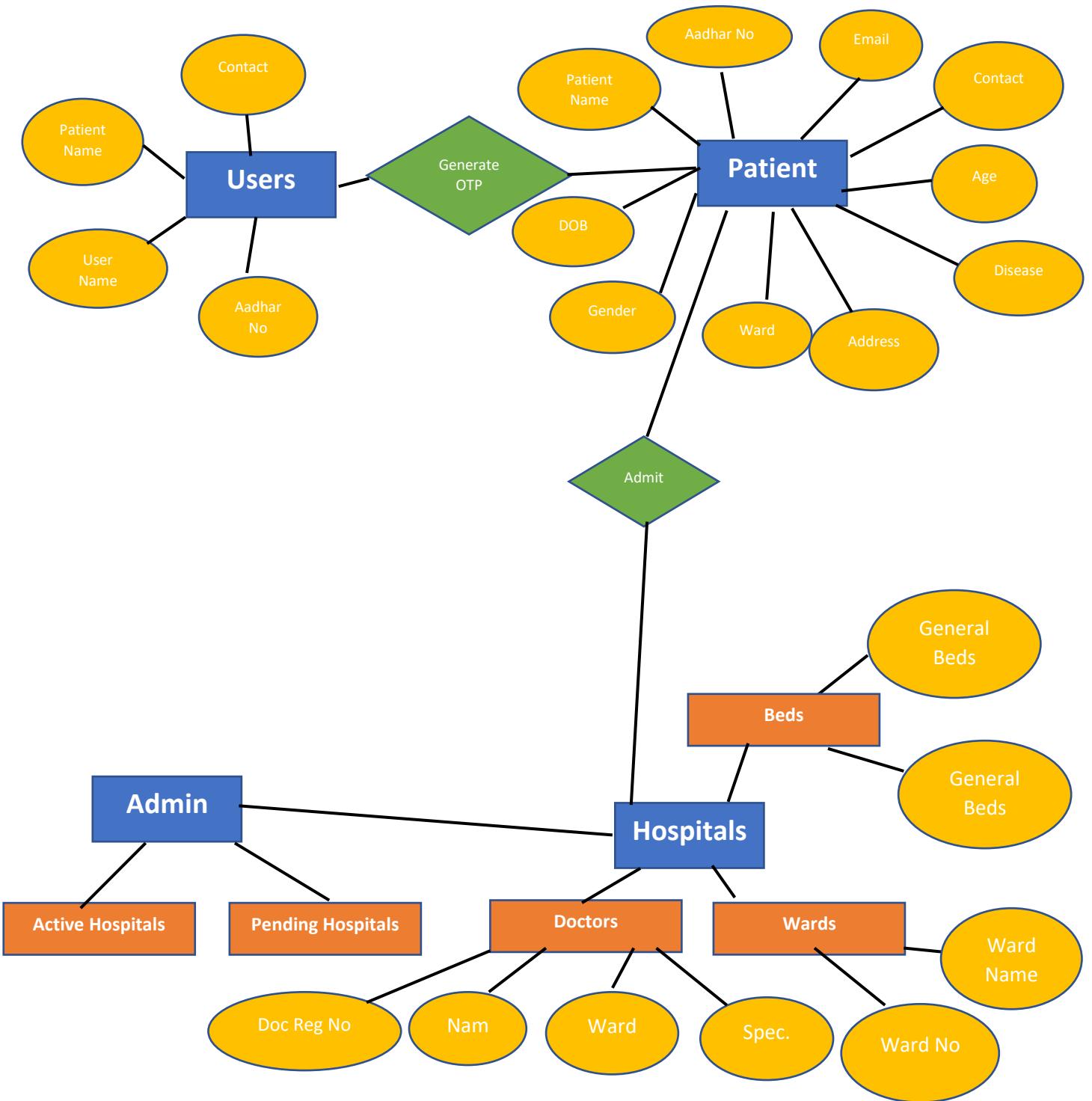
Database Development

MongoDB Atlas is a cloud-based, open-source, NoSQL database that uses JSON documents with dynamic schemas, serving as an alternative to table databases. Atlas provides all the features of MongoDB, while automating database administration tasks such as database configuration, infrastructure provisioning, patches, scaling events, backups, and more, freeing up developers to focus on what matters to them most.

MongoDB Atlas also provides the dual benefit of flexibility and scalability. Dynamic schemas allow users to change the schema of their data without modifying it, providing flexibility. While its “automatic sharding” feature allows users to scale up or out across a range of instances, with zero application downtime.

Collection Name	Documents	Documents Size	Documents Avg	Indexes	Index Size	Index Avg
admins	1	144B	144B	1	32KB	32KB
beds	8	886B	111B	1	32KB	32KB
doctors	9	1.11KB	127B	1	32KB	32KB
hospitalmodels	6	3.2KB	547B	1	32KB	32KB
patientdischarges	0	0B	0B	1	12KB	12KB
patientmodels	3	1.14KB	389B	1	32KB	32KB
tempusers	0	0B	0B	2	24KB	12KB
users	0	0B	0B	2	8KB	4KB
wards	7	638B	92B	1	32KB	32KB

5.3.1 ER-Diagram



6.4 Back-End Development

Graphql

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

Schema

```
import { buildSchema } from 'graphql'

const schema = buildSchema(`

type Hospitals {
    _id: ID!
    hospitalName: String!
    hospitalRegistrationNo: String!
    hospitalType : String!
    government: String!
    address: String!
    state: String!
    district: String!
    pincode: String!
    website: String
    lognitude: String!
    latitude: String!
    ownerName: String!
    ownerContactNo: String!
    ownerEmail: String!
    password: String!
    status: String!
}
```

```
}

type wards {
    _id: ID
    wardsName: String!
    wardNo: String!
}

type doctors {
    _id: ID
    docName: String!
    docReg: String!
    docSp: String!
    docWard: String
}

type beds {
    _id: ID
    privateBeds: Int!
    generalBeds: Int!
    wardsName: String
}

type Admin {
    _id: ID
    adminId: String!
    password: String!
    role: String!
}

type AdminAuth {
    adminId: ID!
    token: String!
    tokenExpiration: String!
}

type HospitalLogin {
    _id: ID
    hosId: String!
    password: String!
}

type HospitalAuth {
    hosId: ID!
    hosToken: String!
}
```

```
        hosTokenExpiration: String!
    }

type User {
    name: String!
}

type TempUser {
    _id: ID
    UserName: String!
    PatientName: String!
    UserContext: String!
    UserAadhar: String!
    otp: String!
}

type PatientAdmit{
    _id:ID
    hospitalId:ID
    bedId: ID
    fname:String!
    lname:String!
    aadharNo: String!
    contactNo: String!
    email: String!
    dob: String!
    gender:String!
    marital: String!
    disease: String!
    age:Int!
    address: String!
    state:String!
    district:String!
    city:String!
    pincode:String!
    ward: String!
    bedtype: String!
}

input HospitalInput {
    hospitalName: String!
    hospitalRegistrationNo: String!
    hospitalType : String!
```

```
        government: String!
        address: String!
        state: String!
        district: String!
        pincode: String!
        website: String
        longitude: String!
        latitude: String!
        ownerName: String!
        ownerContactNo: String!
        ownerEmail: String!
        password: String!
        status: String!
    }

    input wardInput {
        wardsName: String
        wardNo: String
    }

    input doctorInput {
        docName: String!
        docReg: String!
        docSp: String!
        docWard: String
    }

    input bedInput {
        privateBeds: Int!
        generalBeds: Int!
        wardsName: String
    }

    input adminInput {
        adminId: String!
        password: String!
        role: String!
    }

    input HospitalLoginInput {
        hosId: String!
        password: String!
    }

    input userInput{
```

```
        name: String!
    }

input userTempInput {
    UserName: String!
    PatientName: String!
    UserContext: String!
    UserAadhar: String!
    otp: String!
}

input PatientAdmitInput{

    fname:String!
    lname:String!
    aadharNo: String!
    contactNo: String!
    email: String!
    dob: String!
    gender:String!
    marital: String!
    disease: String!
    age:Int!
    address: String!
    state:String!
    district:String!
    city:String!
    pincode:String!
    ward: String!
    bedtype: String!
}

type Query {
    getHospital: [Hospitals]!
    getActiveHospital: [Hospitals]
    getSingleHospital(id: ID): Hospitals!
    getPendingHospitals: [Hospitals]!

    getWards: [wards]
    getDoctors: [doctors]
    getBeds: [beds]
    getWardsById(id:ID): [wards]
```

```

getTempUser: [TempUser]

getUser: [User]

adminLogin(loginId: String! password: String!): AdminAuth

hospitalLogin(hosId: String! password: String!): HospitalAuth

getWardsByOwnId(id:ID): wards
getDoctorsByOwnId(id:ID): doctors
getBedsByOwnId(id:ID): beds

otpVerify(hospitalId: ID , otp: String): TempUser

getBedTypeByHosId(hospitalId: ID, wardName:String):beds

getAllAdmitPatient(hospitalId:ID): [PatientAdmit]
getAdmitPatientByOwnId(id:ID): PatientAdmit

getWardsByHosId(hospitalId:ID): [wards]
getBedsByHosId(hospitalId:ID): [beds]
getDoctorsByHosId(hospitalId:ID): [doctors]

}

type Mutation {
  createHospital(input: HospitalInput): Hospitals
  updateStatus(id: String!, status: String!): Hospitals

  createWards(id:ID!, WardInput: wardInput): wards
  createDoctors(id:ID!, DoctorInput: doctorInput): doctors
  createBeds(id:ID!, BedInput: bedInput): beds

  createAdmin(AdminInput:adminInput): Admin

  createUser(UserInput: userInput): User

  createTempUser(id:ID!, TempUserInput: userTempInput): TempUser

```

```

updateBeds(hospitalId: ID!, wardName: String!, BedInput: bedInput): beds
editDoctors(docId: ID!, DoctorInput: doctorInput): doctors
editBeds(bedId: ID!, BedInput: bedInput): beds
editWards(wardId:ID,WardInput: wardInput): wards
deleteWards(wardId:ID): wards
deleteDoctor(docId:ID): doctors
deleteBed(bedId:ID): beds

createPatientAdmit(hospitalId:ID!, bedId:ID, input: PatientAdmitInput): PatientAdmit
createPatientDischarge(hospitalId:ID!, bedId:ID, input: PatientAdmitInput): PatientAdmit
deleteAdmitPatient(patientId:ID): PatientAdmit

updateBedsByWardName(hospitalId: ID, wardName: String, BedInput: bedInput): beds
updateAdmitBedPlus(bedId:ID, BedInput:bedInput): beds

}

`)

export default schema;

```

Resolver

```

const bcrypt = require('bcryptjs')
const jwt = require('jsonwebtoken')
const HospitalModel = require('../models/hospital')
const Wards = require('../models/wards')
const Doctors = require('../models/doctors')
const Beds = require('../models/beds')
const Admins = require('../models/admins')
const User = require('../models/user')
const TempUser = require('../models/tempUser')
const PatientAdmit = require('../models/patientAdmit')
const PatientDischarge = require('../models/patientDischarge')

const resolvers = {

```

```
getHospital: () => {
    return HospitalModel.find()
},

getSingleHospital: args => {
    return HospitalModel.findById({ _id: args.id })
},

getPendingHospitals: args => {
    return HospitalModel.find({ status: "Pending" })
},

getActiveHospital: args => {
    return HospitalModel.find({ status: "Active" })
},

createHospital: args => {

    return bcrypt.hash(args.input.password, 12)
        .then(hashedpassword => {
            const hospital = new HospitalModel({
                hospitalName: args.input.hospitalName,
                hospitalRegistrationNo: args.input.hospitalRegistrationNo,
                hospitalType: args.input.hospitalType,
                government: args.input.government,
                address: args.input.address,
                state: args.input.state,
                district: args.input.district,
                pincode: args.input.pincode,
                website: args.input.website,
                longitude: args.input.longitude,
                latitude: args.input.latitude,
                ownerName: args.input.ownerName,
                ownerContactNo: args.input.ownerContactNo,
                ownerEmail: args.input.ownerEmail,
                password: hashedpassword,
                status: args.input.status
            })
            return hospital.save()
        })
        .catch(err => {
            throw err;
        });
}
```

```

    },

    updateStatus: args => {
        return HospitalModel.findOneAndUpdate({ _id: args.id }, { status: args.status }, { new: true })
    },

    createWards: args => {
        return HospitalModel.findOne({ _id: args.id })
            .then(hospital => {
                if (!hospital) {
                    throw new Error('Hospital Not Found.');
                }
                const WardsData = new Wards({
                    hospitalId: args.id,
                    wardsName: args.WardInput.wardsName,
                    wardNo: args.WardInput.wardNo
                })
                return WardsData.save()
            })
    },
    createDoctors: args => {
        return HospitalModel.findOne({ _id: args.id })
            .then(hospital => {
                if (!hospital) {
                    throw new Error('Hospital Not Found.');
                }
                const DoctorsData = new Doctors({
                    hospitalId: args.id,
                    docName: args.DoctorInput.docName,
                    docReg: args.DoctorInput.docReg,
                    docSp: args.DoctorInput.docSp,
                    docWard: args.DoctorInput.docWard
                })
                return DoctorsData.save()
            })
    },
    createBeds: args => {
        return HospitalModel.findOne({ _id: args.id })
            .then(hospital => {

```

```

        if (!hospital) {
            throw new Error('Hospital Not Found.');
        }
        const BedsData = new Beds({
            hospitalId: args.id,
            privateBeds: args.BedInput.privateBeds,
            generalBeds: args.BedInput.generalBeds,
            wardsName: args.BedInput.wardsName,
        })
        return BedsData.save()
    })
),

getWards: () => {
    return Wards.find()
},

getWardsById: args => {
    return Wards.find({ hospitalId: args.id })
},

getWardsByOwnId: args => {
    return Wards.findById({ _id: args.id })
},
getDoctorsByOwnId: args => {
    return Doctors.findById({ _id: args.id })
},

deleteWards: args => {
    return Wards.findByIdAndRemove(args.wardId)
},

getDoctors: () => {
    return Doctors.find()
},

deleteDoctor: args => {
    return Doctors.findByIdAndRemove(args.docId)
},

getBeds: () => {
    return Beds.find()
},
getBedsByOwnId: args => {

```

```

        return Beds.findById({ _id: args.id })
    },
    deleteBed: args => {
        return Beds.findByIdAndRemove(args.bedId)
    },
    getTempUser: () => {
        return TempUser.find()
    },
    otpVerify: (args) => {
        return TempUser.findOne({ hospitalId: args.hospitalId, otp: args.otp })
            .then(otp => {
                if (!otp) {
                    throw new Error('Otp Not Found.');
                }
                return otp
            })
    },
    getUser: args => {
        return User.find()
    },
    createAdmin: args => {
        return bcrypt.hash(args.AdminInput.password, 12)
            .then(hashedpassword => {
                const Admin = new Admins({
                    adminId: args.AdminInput.adminId,
                    password: hashedpassword,
                    role: args.AdminInput.role,
                });
                return Admin.save()
            })
            .catch(err => {
                throw err;
            });
    },
    adminLogin: async ({ loginId, password }) => {
        const admin = await Admins.findOne({ adminId: loginId });
        if (!admin) {
            throw new Error('Admin Not Exists');
        }
    }
}

```

```

    }

    const isEqual = await bcrypt.compare(password, admin.password);
    if (!isEqual) {
        throw new Error('Password Incorrect');
    }
    const token = jwt.sign({ adminUserId: admin.id, loginId: admin.adminId },
'superAdminSecretKey', {
    expiresIn: '1h'
})
return {
    adminId: admin.adminId,
    token: token,
    tokenExpiration: '1h'
}
}

hospitalLogin: async ({ hosId, password }) => {
    const hospital = await HospitalModel.findOne({ hospitalRegistrationNo: hosId });
    if (!hospital) {
        throw new Error('Hospital Not Exists');
    }

    const isEqual = await bcrypt.compare(password, hospital.password);
    if (!isEqual) {
        throw new Error('Password Incorrect');
    }
    const hosToken = jwt.sign({ hosUserId: hospital.id, hosId: hospital.hospitalRegistrationNo }, 'hospitalSecretKey', {
        expiresIn: '1h'
})
return {
    hosId: hospital.hospitalRegistrationNo,
    hosToken: hosToken,
    hosTokenExpiration: '1h'
}
}

createUser: args => {
    const Users = new User({

```

```

        name: args.UserInput.name
    })
    return Users.save()
}

,

createTempUser: args => {
    return HospitalModel.findOne({ _id: args.id })
        .then(hospital => {
            if (!hospital) {
                throw new Error('Hospital Not Found.');
            }
            const tempUserDetails = new TempUser({
                hospitalId: args.id,
                UserName: args.TempUserInput.UserName,
                PatientName: args.TempUserInput.PatientName,
                UserContect: args.TempUserInput.UserContect,
                UserAadhar: args.TempUserInput.UserAadhar,
                otp: args.TempUserInput.otp
            })
            return tempUserDetails.save()
        })
}

,

updateBeds: args => {
    return Beds.findOne({ hospitalId: args.hospitalId, wardsName: args.wardNa
me })
        .then(wards => {
            if (!wards) {
                const BedsData = new Beds({
                    hospitalId: args.hospitalId,
                    privateBeds: args.BedInput.privateBeds,
                    generalBeds: args.BedInput.generalBeds,
                    wardsName: args.BedInput.wardsName,
                })
                return BedsData.save()
            }
            const filter = { hospitalId: args.hospitalId, wardsName: args.war
dName };
            const update = { privateBeds: args.BedInput.privateBeds, generalB
eds: args.BedInput.generalBeds };

```

```

        return Beds.findOneAndUpdate(filter, update)

    })
},

updateDoctors: args => {
    return Doctors.findOne({ hospitalId: args.hospitalId, docId: args.docId })
)
    .then(docs => {
        if (!docs) {
            const DocData = new Docs({
                hospitalId: args.hospitalId,
                docName: args.DoctorInput.docName,
                docReg: args.DoctorInput.docReg,
                docSp: args.DoctorInput.docSp,
                docWard: args.DoctorInput.docWard
            })
            return DocData.save()
        }
        const filter = { hospitalId: args.hospitalId, wardsName: args.wardName };
        const update = { privateBeds: args.BedInput.privateBeds, generalBeds: args.BedInput.generalBeds };
        return Beds.findOneAndUpdate(filter, update)
    })
},
editWards: args => {
    return Wards.findOne({ _id: args.wardId })
    .then(wards => {
        if (!wards) {
            throw new Error('Ward Not Found.');
        }
        const filter = { _id: args.wardId };
        const update = { wardsName: args.WardInput.wardsName, wardNo: args.WardInput.wardNo };
        return Wards.findOneAndUpdate(filter, update)
    })
},
editDoctors: args => {
    return Doctors.findOneAndUpdate({ _id: args.docId }, {

```

```

        docName: args.DoctorInput.docName,
        docReg: args.DoctorInput.docReg,
        docSp: args.DoctorInput.docSp,
        docWard: args.DoctorInput.docWard
    })
},
editBeds: args => {
    return Beds.findOneAndUpdate({ _id: args.bedId }, {
        privateBeds: args.BedInput.privateBeds,
        generalBeds: args.BedInput.generalBeds,
        wardsName: args.BedInput.wardsName,
    })
},
getWardsByHosId: (args) => {
    return Wards.find({ hospitalId: args.hospitalId })
    .then(ward => {
        if (!ward) {
            throw new Error('Wards Not Found.');
        }
        return ward
    })
},
getBedsByHosId: (args) => {
    return Beds.find({ hospitalId: args.hospitalId })
    .then(bed => {
        if (!bed) {
            throw new Error('Beds Not Found.');
        }
        return bed
    })
},
getDoctorsByHosId: (args) => {
    return Doctors.find({ hospitalId: args.hospitalId })
    .then(doc => {
        if (!doc) {
            throw new Error('Doctors Not Found.');
        }
        return doc
    })
}

```

```

        })
    },

    getBedTypeByHosId: (args) => {
        return Beds.findOne({ hospitalId: args.hospitalId, wardsName: args.wardNa
me })
            .then(bedType => {
                if (!bedType) {
                    throw new Error('Bed Not Found.');
                }
                return bedType
            })
    },

    createPatientAdmit: args => {

        return HospitalModel.findOne({ _id: args.hospitalId })
            .then(hospital => {
                if (!hospital) {
                    throw new Error('Hospital Not Found.');
                }
                return Beds.findOne({ _id: args.bedId })
                    .then(bed => {
                        if (!bed) {
                            throw new Error('Bed Not Found.');
                        }
                        const patient = new PatientAdmit({
                            hospitalId: args.hospitalId,
                            bedId: args.bedId,
                            fname: args.input.fname,
                            lname: args.input.lname,
                            aadharNo: args.input.aadharNo,
                            contactNo: args.input.contactNo,
                            email: args.input.email,
                            dob: args.input.dob,
                            gender: args.input.gender,
                            marital: args.input.marital,
                            disease: args.input.disease,
                            age: args.input.age,
                            address: args.input.address,
                            state: args.input.state,
                            district: args.input.district,
                            city: args.input.city,
                        })
                    })
            })
    }
}

```

```

                pincode: args.input.pincode,
                ward: args.input.ward,
                bedtype: args.input.bedtype

            })
            return patient.save()
        })

    })

    updateBedsByWardName: args => {
        return Beds.findOne({ hospitalId: args.hospitalId, wardsName: args.wardNa
me })
            .then(bed => {
                if (!bed) {
                    throw new Error('Hospital Not Found.');
                }
                const filter = { hospitalId: args.hospitalId, wardsName: args.war
dName };
                const update = { privateBeds: args.BedInput.privateBeds, generalB
eds: args.BedInput.generalBeds };
                return Beds.findOneAndUpdate(filter, update)
            })
    },

    getAllAdmitPatient: (args) => {
        return PatientAdmit.find({ hospitalId: args.hospitalId })
            .then(hos => {
                if (!hos) {
                    throw new Error('Hospital Not Found.');
                }
                return hos
            })
    },

    getAdmitPatientByOwnId: args => {
        return PatientAdmit.findById({ _id: args.id })
    },

    updateAdmitBedPlus: args => {
        return Beds.findOne({ _id: args.bedId })
            .then(bed => {

```

```

        if (!bed) {
            throw new Error('Bed Not Found.');
        }
        const filter = { _id: args.bedId };
        const update = { privateBeds: args.BedInput.privateBeds, generalBeds: args.BedInput.generalBeds };
        return Beds.findOneAndUpdate(filter, update)
    )
),

createPatientDischarge: args => {

    return HospitalModel.findOne({ _id: args.hospitalId })
        .then(hospital => {
            if (!hospital) {
                throw new Error('Hospital Not Found.');
            }
            return Beds.findOne({ _id: args.bedId })
                .then(bed => {
                    if (!bed) {
                        throw new Error('Bed Not Found.');
                    }
                    const patientDis = new PatientDischarge({
                        hospitalId: args.hospitalId,
                        bedId: args.bedId,
                        fname: args.input.fname,
                        lname: args.input.lname,
                        aadharNo: args.input.aadharNo,
                        contactNo: args.input.contactNo,
                        email: args.input.email,
                        dob: args.input.dob,
                        gender: args.input.gender,
                        marital: args.input.marital,
                        disease: args.input.disease,
                        age: args.input.age,
                        address: args.input.address,
                        state: args.input.state,
                        district: args.input.district,
                        city: args.input.city,
                        pincode: args.input.pincode,
                        ward: args.input.ward,
                        bedtype: args.input.bedtype
                    })
                })
        })
}

```

```
        return patientDis.save()
    })

})

},

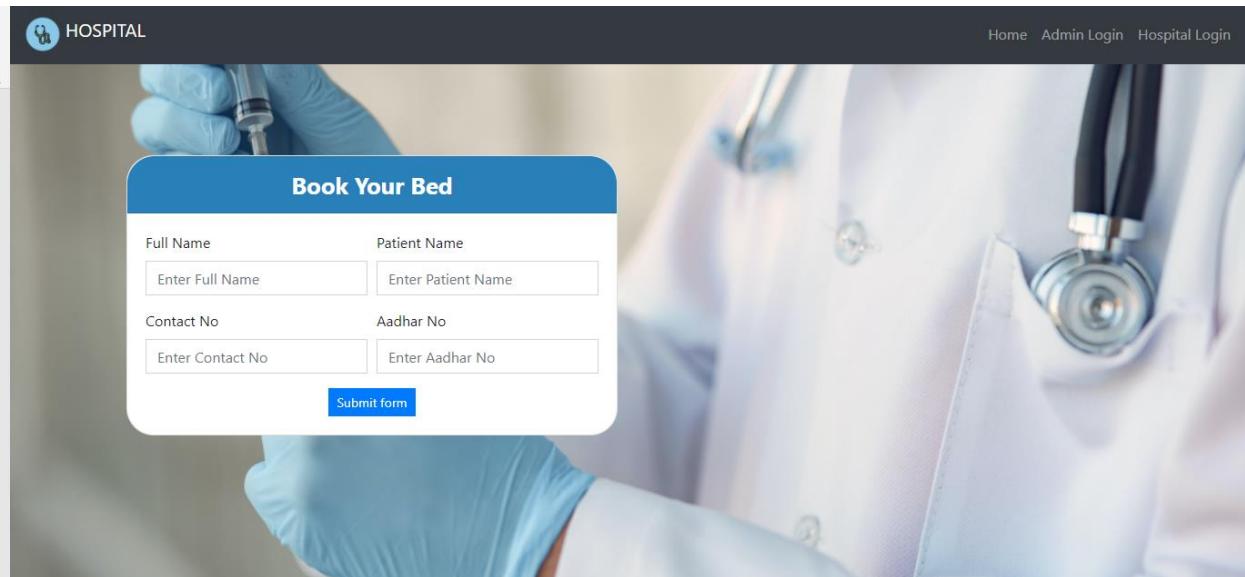
deleteAdmitPatient: args => {
    return PatientAdmit.findByIdAndRemove({ _id: args.patientId })
},
}

export default resolvers;
```

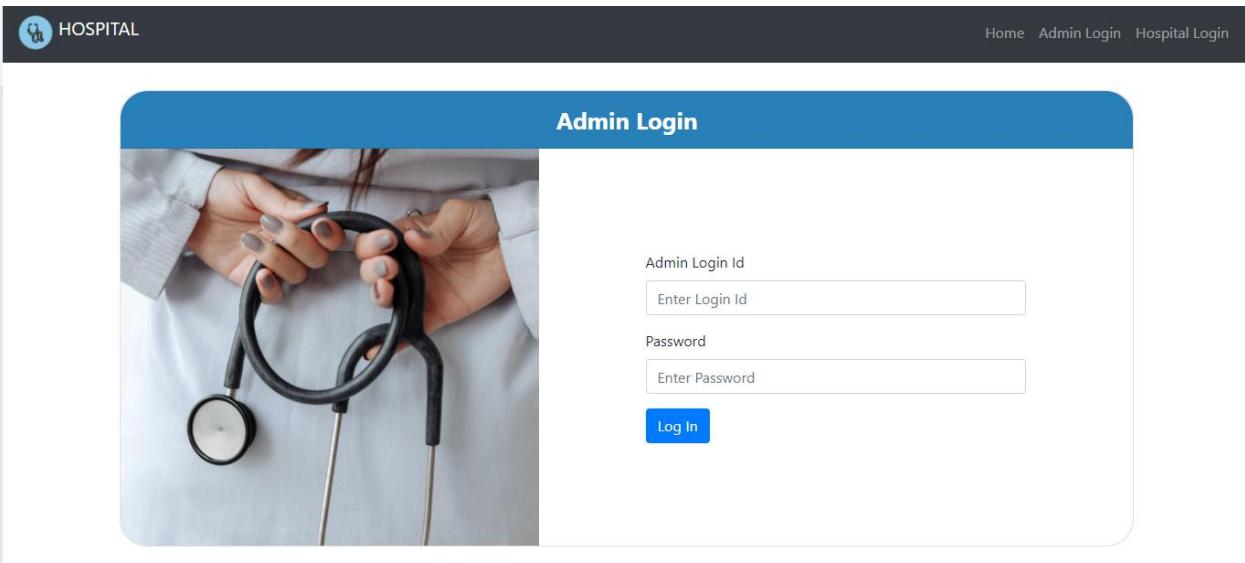
CHAPTER 7

SCREENSHOTS

7.1 Home Page



6.2 Admin Login



6.3 Hospital Login



The screenshot shows the Hospital Login page. At the top left is a logo with a stethoscope icon and the word "HOSPITAL". At the top right are links for "Home", "Admin Login", and "Hospital Login". The main title "Hospital Login" is centered at the top of the form area. To the left of the form is a photograph of a person in a white lab coat holding a black stethoscope. The form itself has two input fields: "Hospital Registration No." and "Enter Login Id" (placeholder text), and "Password" with "Enter Password" placeholder text. Below these is a large black "Log In" button. At the bottom of the form is a link "Don't have an account? [Register here](#)".

6.4 Hospital Registration



The screenshot shows the Register Your Hospital page. At the top left is a logo with a stethoscope icon and the word "HOSPITAL". At the top right are links for "Home", "Admin Login", and "Hospital Login". The main title "Register Your Hospital" is centered at the top of the form area. To the left of the form is a photograph of a person in a green medical scrub top holding a syringe and a stethoscope. The form contains several input fields: "Hospital Name" (placeholder "Enter Hospital Name"), "Owner Name" (placeholder "Enter Owner Name") and "Owner Contact No." (placeholder "Enter Owner Contact number"), "Owner Email Address" (placeholder "Enter Owner Email Id") and "Hospital Registration Number" (placeholder "Enter Hospital Registration Number"), "Hospital Type" (dropdown menu "Select Hospital Type" with "Government" option selected) and "Category of Hospital" (dropdown menu "Select Category of Hospital"), and "Hospital Address" (placeholder "Enter Your Hospital Address").

6.5 Nearest Hospitals List

The screenshot shows a user interface for finding nearby hospitals. At the top left is a logo with a stethoscope and the word "HOSPITAL". At the top right are links for "Home", "Admin Login", and "Hospital Login". Below this is a map of Sagar, Madhya Pradesh, showing major roads like NH146, NH934, SH20, and SH21. Blue markers indicate the locations of three hospitals: NEO CHILDREN HOSPITAL, Shri Chaitanya Hospital, and Shiv Sagar. To the right of the map, detailed information is provided for each hospital, including distance, address, and buttons to "Book Bed", "Show Wards", "Show Beds", and "Show Doctors".

Hospital Name	Registration No	Type	Owner Name	Show Details
Shiv Sagar	shiv12345	Super Speciality	Arvind Vishwakarma	Show Details
Chataniya Hospital	123	PHC	test	Show Details
NEO CHILDREN HOSPITAL	neo123	District Hospital	test	Show Details
Dufferin Hospital	duf123	District Hospital	test	Show Details
Shri Chaitanya Hospital	shri123	Specialised Hospital	test	Show Details

6.6 Admin Dashboard

The screenshot shows the Admin Dashboard. At the top left is a logo with a stethoscope and the word "HOSPITAL". At the top right are links for "Home", "Objectives", "Guide", "About Us", "Contact Us", "Dashboard", and "Logout". The main title "Admin Dashboard" is centered above a welcome message "Welcome admin123". On the left, there are two buttons: "Active Hospitals" (highlighted in blue) and "Pending Hospitals". The main content area is titled "All Active Hospitals" and displays a table of hospital data. The table has columns for "Hospital Name", "Registration No", "Type", "Owner Name", and "Show Details". Each row contains a "Show Details" button.

Hospital Name	Registration No	Type	Owner Name	Show Details
Shiv Sagar	shiv12345	Super Speciality	Arvind Vishwakarma	Show Details
Chataniya Hospital	123	PHC	test	Show Details
NEO CHILDREN HOSPITAL	neo123	District Hospital	test	Show Details
Dufferin Hospital	duf123	District Hospital	test	Show Details
Shri Chaitanya Hospital	shri123	Specialised Hospital	test	Show Details

6.7 Hospital Dashboard

The screenshot displays a hospital dashboard interface. On the left, a vertical sidebar contains links: 'Otp Verification' (highlighted in blue), 'Patients Details', 'Enter Hospital Details', 'Hospital Wards', 'Hospital Doctors', and 'Hospital Beds'. The main content area is titled 'Hospital Dashboard' and features a sub-section titled 'Verify Patient's OTP'. It includes a row of six input fields for OTP digits, a label 'OTP Entered -', and two buttons: 'Clear' and 'Verify OTP'.

HOSPITAL

Home Objectives Guide About Us Contact Us Dashboard Logout

Hospital Dashboard

Verify Patient's OTP

OTP Entered -

Clear Verify OTP

- Otp Verification
- Patients Details
- Enter Hospital Details
- Hospital Wards
- Hospital Doctors
- Hospital Beds

CHAPTER 8

CONCLUSION

8.1 Conclusion

The overall architecture and function flow of the system are analyzed and introduced in detail by using the overall system architecture diagram and the bed center function flow chart in this paper. We also present the three-tier structure of presentation layer, business logic layer and data layer respectively in details. According to the HIS function requirement as well as combining the intrinsic HIS preferably, our research has designed and realized a set of safe, stable and easy-to handle beds resource management information system aiming at the problems of “difficult to be hospitalized”, which provide the hospital beds centralized management with comprehensive information solutions .Although the hospital's bed management mode was optimized and the average length of hospital stay was reduced, the problem of hospitalization was relieved to a great extent. However, how to protect the quality of patients under the premise of the quality of care, making the average hospital stay in patients with a more reasonable arrangement, still need a better communication between hospital management and patient, to complete a more perfect bed management.

The research highlighted a very complex and difficult issue of hospital bed allocation experienced by most major hospitals. The current systems have vast amount of written information located in a variety of computer and hardcopy file areas which require specific targeted searching. E-Beds has integrated these existing information systems and formats and provides a level of visualization which allows health care professionals to make informed and more accurate decisions for patient administration. Efficiencies in bed allocation provide increased economical, appropriate and beneficial patient care and a vital service to the emergency services. Hospitals without effective bed management practices face increased staff time in planning and assigning patients to beds in addition to increased costs through alternative placement of patients when beds are not ready or available when needed. Accuracy and efficient use of information is highly dependent on the established patient discharge process and related bed management function. Applying e-Beds spatial technology is addressing these information needs. Hence, the e-Beds system is a practical, flexible and dynamic tool and has also demonstrated that spatial technology has a beneficial role within the hospital environment.

CHAPTER 9

BIBLOGRAPHY

- Australian Institute of Health and Welfare (2003). Australian Hospital Statistics 2003-04. Australian Institute of Health and Welfare (2003), Health Services Series No. 23.
- Richardson, Drew B. (2003) Reducing patient in the emergency department. *The Medical Journal of Australia*, 170 (10), p.516-517
- Walsh, M. (1998) What is a Bed? – Beds as a Measure of Resource Usage and Demand. *Better Outcomes Health Newsletter*, V4 N3
- Ward, Dr Michael (2004). Bed Management Collaborative. Queensland Health Quality and Safety Improvement Program, Queensland. Wikipedia 2006. The Free Dictionary. www.wikipedia.org.
- Pinggen,W., and Yunsuo,G. (2012)Preliminary Study on the Distribution Plan of Hospital Beds in Large General Hospital.Chinese Journal of Hospital Statistics,13,7-8.
- Microsoft Corporation, ASP.NET MVC: The Official Microsoft ASP.NET Site[EB/OL], <http://www.asp.net/mvc2012-2>.
- Ku,H.S.,Kim,S. and Kim,H.(2014)DialysisNet:Application for Integrating and Management Data Sources of Hemodialysis Information by Continuity of Care Record.*Healthc Inform Res*, 20, 145-151.
- Nguyen,J.M.,Six,P.and Antonioli,D.(2015)A simple method to optimize hospital beds capacity.*Journal of Health Geographies*,74,39-49.
- Jun,Y. and Cheng,C.(2014) Design of system architecture based on J2EE workflow platform.*Electronic technology and software engineering*, 11,59-60.