A Mini Project Report

on

“Clinic Management System”

Submitted by

Group 3

33312 - Kshitij Deshpande
33313 - Vedant Deshpande
33323 - Aditya Kangune
33356 - Yash Patwardhan

Guide

Mrs. Sumitra Jakhete

Department Of Information Technology

Pune Institute of Computer Technology College of Engineering

Sr. No 27, Pune-Satara Road, Dhankawadi, Pune - 411 043.

A.Y. 2021-2022

Abstract

The management of a clinic with the data of doctors, patients and pharmacists is a tedious and complicated task. The problems faced by traditional clinics leading to improper storage of important and critical data of the patients are the reason why there is a need for a simple online application with proper roles of a doctor, a patient and a pharmacist.

This mini project shows how a complicated human task of storing critical data and timelines can be solved by using a mongoDB as the key tool in the development. The basic functionality of the system is to keep track of prescriptions of the patients where there are important details like date of prescription, medication name, dosage, symptoms, etc. Data of the patients should be visualized to the user in a simple and intuitive way. There should be a simple way of getting an overview of the prescriptions according to the user's role. The techstack includes HTML, CSS, Bootstrap and Javascript for the frontend, MongoDB as the database and NodeJS/Express as the backend.

Various different features for different roles of the doctors, pharmacists and the patients will make this system a very powerful clinic administrative system. The security model used based on different privileges for different users also will make the system a real-life application.

Keywords: Traditional clinics, prescriptions,Bootstrap, Javascript, MongoDB. NodeJS/Express, roles

Acknowledgement

“As our circle of knowledge expands, so does the circumference of darkness surrounding it.”

This mini project has helped us a lot to discover various new things. We extend our gratitude to Pune Institute of Computer Technology for giving us an opportunity to enhance our knowledge through this mini project.

We are extremely grateful to Dr. R. Shreemathy, Principal, PICT, and Dr. Anant Bagade, Head of the Department (Information Technology), for providing all the required resources for the successful completion of this mini project. Heartfelt gratitude to our mini project guide Mrs. Sumitra Jakhete, for her valuable suggestions and guidance in the preparation of the mini project report. We express our regards to all the staff members and friends for all the help and coordination. We acknowledge the authors of the references and other pieces of literature referred to in this report.

List Of Figures

CONTENT

CHAPTER 1

INTRODUCTION

1.1 Purpose

The manual method of small scale clinics is characterized with numerous problems. Some of those are :

1. Patients have to manually enter their data for registering in the records of the clinic.
2. Patient/Pharmacist may not clearly understand what medications the doctor has given to them.
3. Patients might forget the timeline of their medication.
4. Patients cannot find the specialist/expert doctor in a particular field.
5. The patients might lose the hardcopy version of their prescription.
6. The foul play that sometimes occurs when information about the patient that has already registered in the records of the clinic is not officially documented by the concerned people.
7. Doctors have to manually write their prescriptions according to the symptoms of the patient.
8. Doctors cannot get to know the previous history of the patient in one glance.
9. Doctors do not have a single place where they can see all the history of the patient along with previous reports and prescriptions.

The purpose of this project is to build a single platform for the use of a clinic where the users can be a patient, a doctor or a pharmacist.

The aim is to develop a data management system to consolidate,organize, document,store and distribute information related to the patient amongst the doctors.

1.2 Scope

The scope of the project is clear to give a simple and attractive application to simplify the work as well as to reduce the efforts while doing it offline or we can say by doing it with traditional methods.

Aim is to develop a centralised digital prescription application - instead of having to physically prescribe medicines which are then given to a pharmacist. A doctor should be able to prescribe medicines with dosages to a patient on his terminal who can then go to any pharmacy and get them. The system must be secure and external stakeholders shouldn't have access to records shared between 2 parties in any case.

1.3 Background and Motivation

The definition of our problem lies in a manual system and a fully automated system.

The manual traditional system of using registers to record data and hardcopy prescriptions is very time consuming and tedious. This system is more prone to errors and sometimes the approaches to various problems are unstructured**.** The main purpose of this work is therefore to develop a web application program that would help automate the traditional tasks for a doctor, pharmacists and the patients. Our project tries to circumvent all the problems that are mentioned above in the purpose.

Having experienced all those problems personally we thought of building an application that can be helpful for a clinic to exist in these modern times.

1.4 Methodology

To implement the project goals, the following methodology needs to be followed:

1. Proper commendations.
2. Proper validations.
3. Go through the created schedule.
4. Proper development of a project by following SDLC.
5. Proper division of tasks between project group members.
6. Proper integration of divided tasks.
7. Proper testing for all modules individually and combined testing after integration.

CHAPTER 2

GENERAL DESCRIPTION

2.1 Product Function Perspective:

Clinic Management System provides different features for doctors, pharmacists and patients. It includes several functionalities describes as below:

1. Users can register into the database by entering information viz. Name, email, age, gender, phone number and password.
2. The user should verify their email address via OTP for security purposes.
3. The user can choose their role from: a doctor, a pharmacist or a patient.
4. The doctor can view their information from their profile, get a list of all their patients, get a detailed timeline of prescriptions provided along with symptoms of a particular patient, in the prescription the doctor has fields like date, symptoms, medication name, dosage and description.
5. The patient can view their information from their profile, get a detailed timeline of their medication and prescriptions alongwith name of medication and dosage. The patient cannot change their medication details unlike the doctor.
6. The pharmacist can view their information from their profile. Main feature for them is that they can view the prescriptions of a patient wherein there are details like symptoms, medication name, date, dosage and description.

2.2) User Characteristics:

There are 3 main users who can use our product:

1. **Doctor:**

a. The doctor can view their information from their profile.
b. Get a list of all their patients, get a detailed timeline of prescriptions provided along with symptoms of a particular patient.
c. In the prescription the doctor has fields like date, symptoms, medication name, dosage and description.
d. The doctor has the authority to edit the patient's prescription.
e. Fast and secure sign up feature.
f. Login will be possible only through registered email ID & account password. After login, the doctor will be able to see their profile and patients data.
g. After clicking on My Patients, they can see the medical history as well as submit a new prescription based on the recent consultation.

2. **Patient:**

a. The patient can view their information from their profile.
b. Get a detailed timeline of their medication and prescriptions alongwith name of medication and dosage.
c. The patient cannot change/edit their medication details unlike the doctor.
d. Fast and secure sign up feature.
e. Login will be possible only through registered email ID & account password.

f. After login, the patient will be able to see ONLY THEIR OWN profile and prescription history.
g. If a new prescription has been submitted by their doctor, they will receive it on their dashboard.
h. There is one-to-one correspondence between the doctor and patient. One patient's data cannot be accessed or tampered by any other patient or user.

3. **Pharmacist:**

a. The pharmacist can view their information from their profile.
b. Main feature for them is that they can view the prescriptions of a patient wherein there are details like symptoms, medication name, date, dosage and description.
c. Similar to the patients, unlike the doctor they cannot change/edit their medication details unlike the doctor.
d. Fast and secure sign up feature.
e. Login will be possible only through registered email ID and account Password. The next step of verification requires the pharmacist to enter the email ID of the patient and OTP is sent to the registered email ID of the patient.
f. After login, the pharmacist will be able to see only that particular patient's most recent prescription.
g. The pharmacist does not have access to any other information about the patients and doctors registered on the portal.

2.3 Assumption and Dependencies:-

Proper working of this app is dependent on the internet connectivity of the users' computer. Assumptions and dependencies are:

- It is assumed that the user has a basic knowledge of how the software works.
- It is assumed that the data entered by the user while registering is true.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Inputs and Outputs:

Input

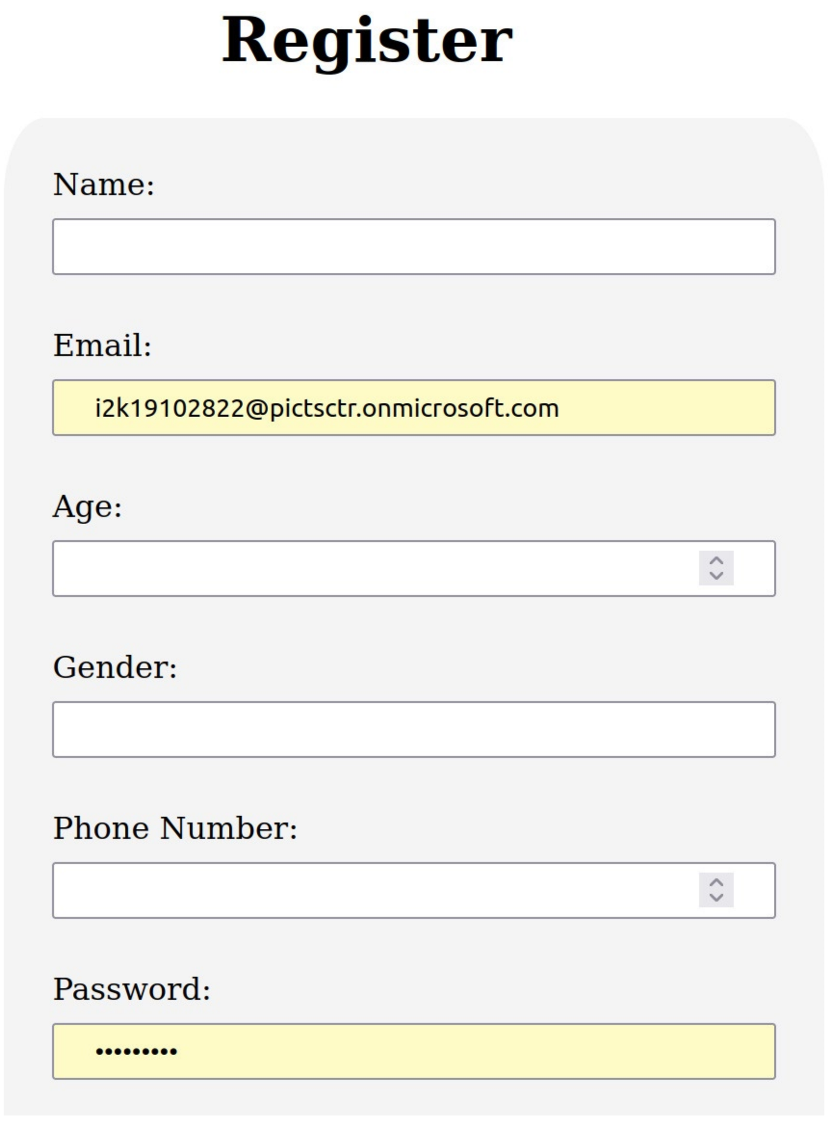
1. Every user (doctor, patient or pharmacist) has to register all the personal details before using the application for the first time. Email and password is used for login.
2. Doctors can get a list of all their patients, get a detailed timeline of prescriptions provided along with symptoms of a particular patient.
3. Patients can view their information from their profile.
4. Pharmacists can view the prescriptions of a patient wherein there are details like symptoms, medication name, date, dosage and description.

Output:

1. The doctor has the authority to edit the patient's prescription.
2. If a new prescription has been submitted by their doctor, patients will receive it on their dashboard.
3. The pharmacist will be able to see only a particular patient's most recent prescription.

3.2 Functionality Requirements:

1. Registration


Register

Name:

Email:
i2k19102822@pictsctr.onmicrosoft.com

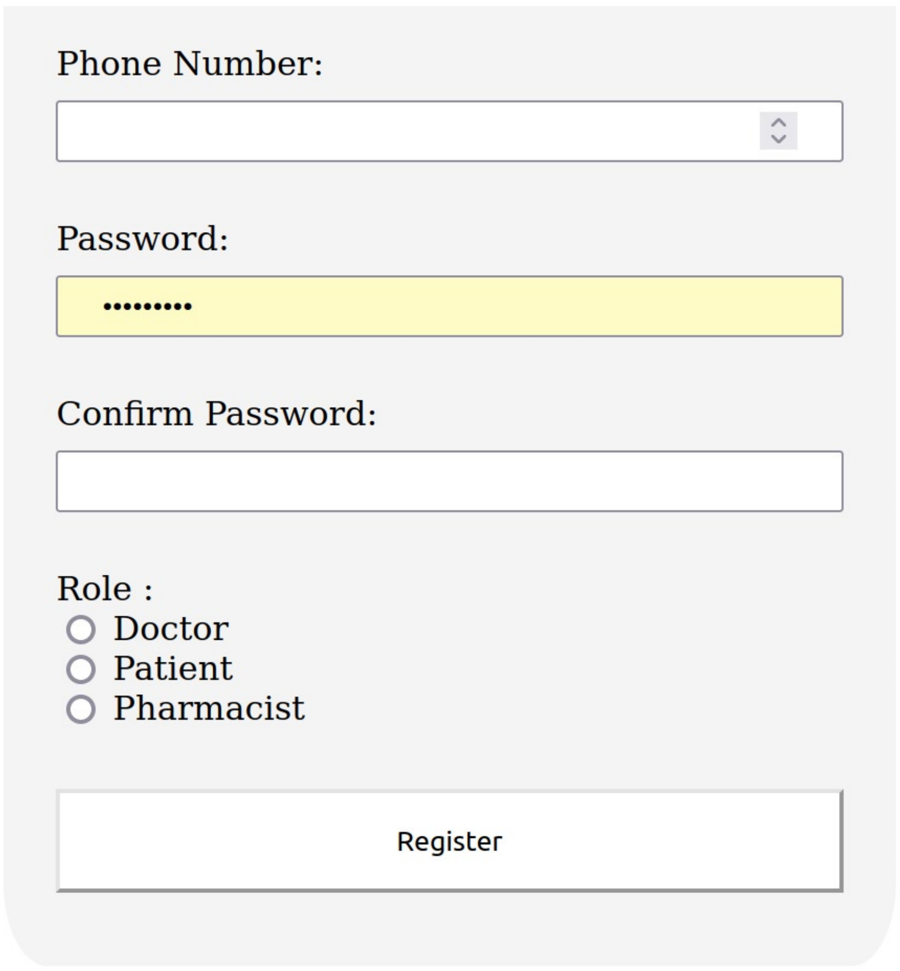
Age:

Gender:

Phone Number:

Password:
•••••••••

Fig. 1


Phone Number:

Password:

•••••••••

Confirm Password:

Role :

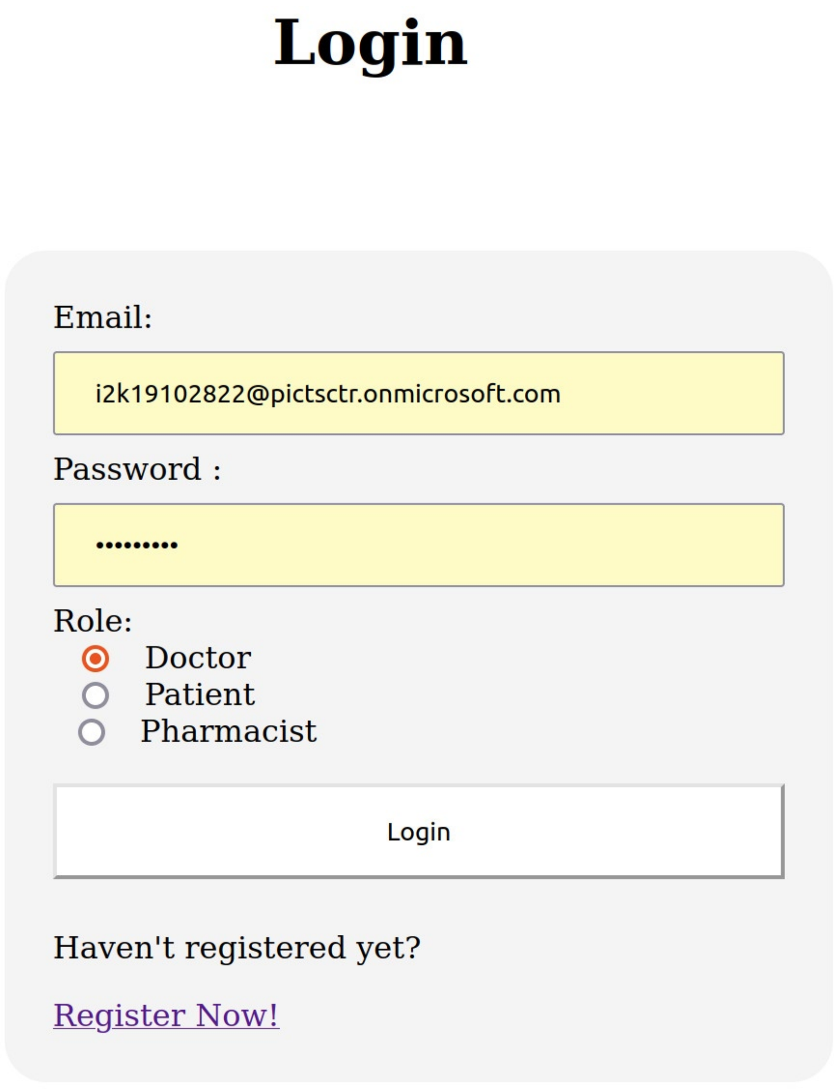
Doctor

Patient

Pharmacist

Fig. 2

2. Login


Login

Email:

i2k19102822@pictsctr.onmicrosoft.com

Password :

•••••••••

Role:

[✓ Doctor]
Patient
Pharmacist

Haven't registered yet?

Register Now!

Fig. 3

3. Verification is done using email ID through OTP

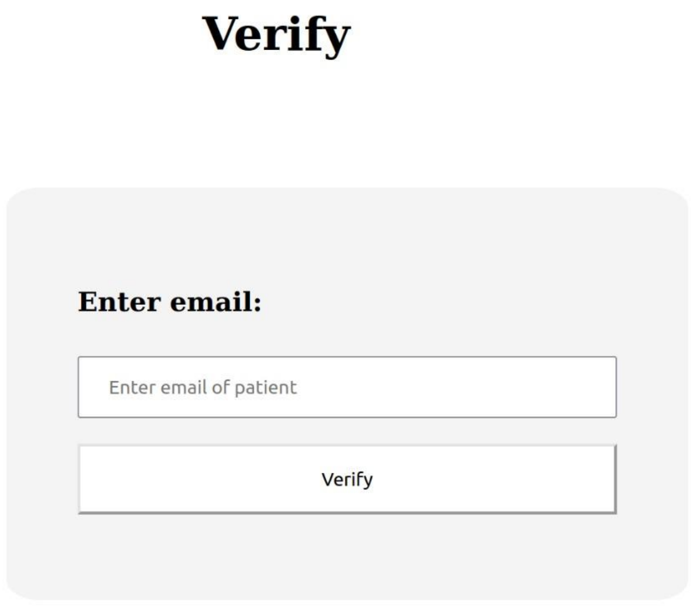


Fig. 4

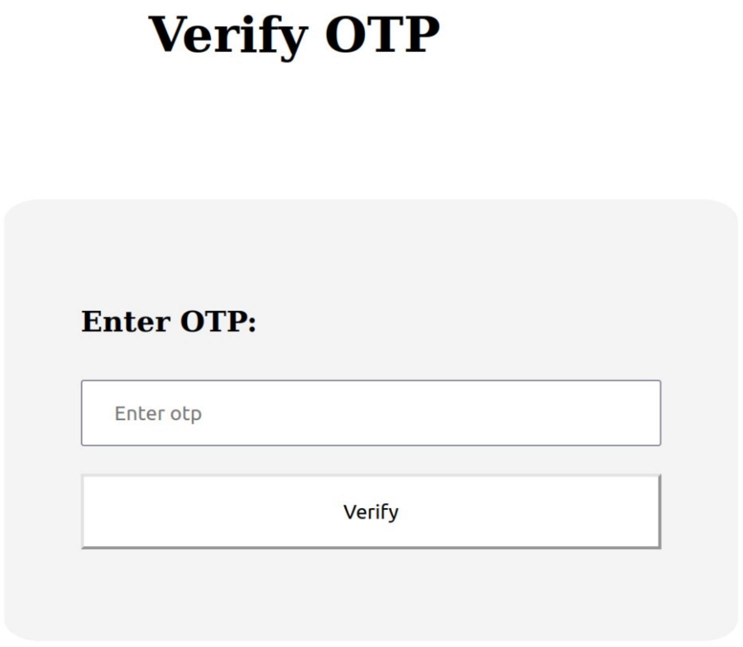
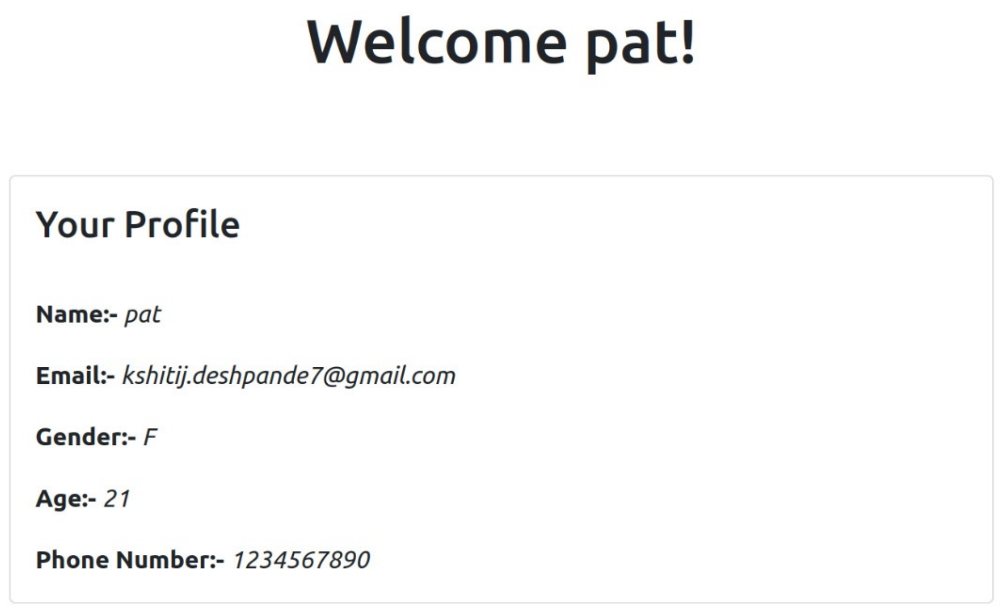


Fig. 5

4. Patient Profile


Welcome pat!

Your Profile

Name:- *pat*

Email:- *kshitij.deshpande7@gmail.com*

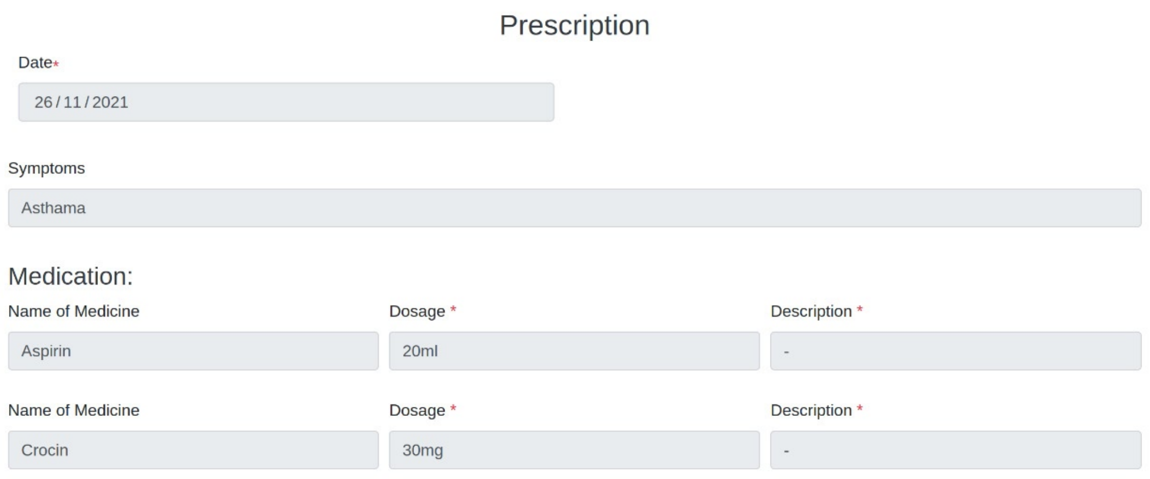
Gender:- *F*

Age:- *21*

Phone Number:- *1234567890*

Fig. 6

5. Patient Prescription - Prescription cannot be edited by the patient.


Prescription

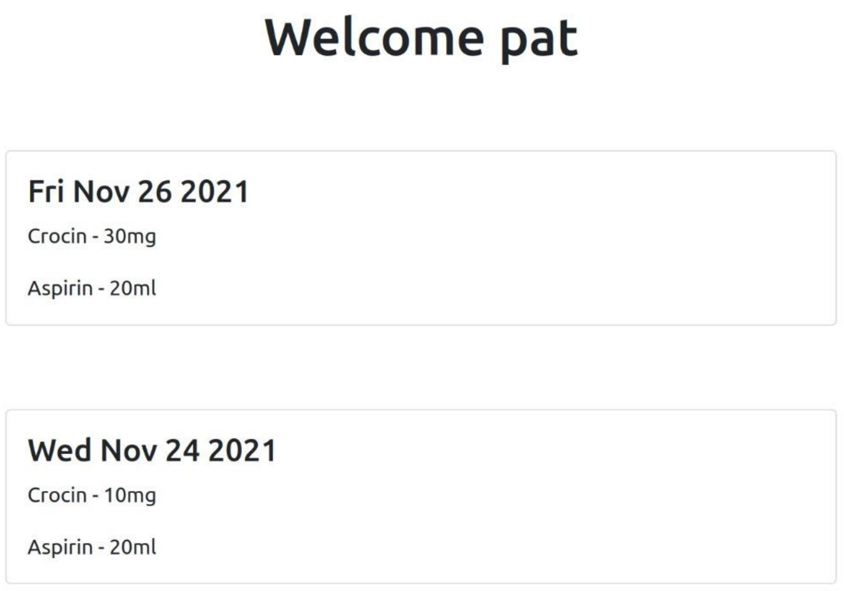
Date* 26 / 11 / 2021		
Symptoms Asthama		

Medication:

Name of Medicine	Dosage *	Description *
Aspirin	20ml	-
Crocin	30mg	-

Fig. 7

6. List of prescriptions for patient:


Welcome pat

Fri Nov 26 2021

Crocin - 30mg

Aspirin - 20ml

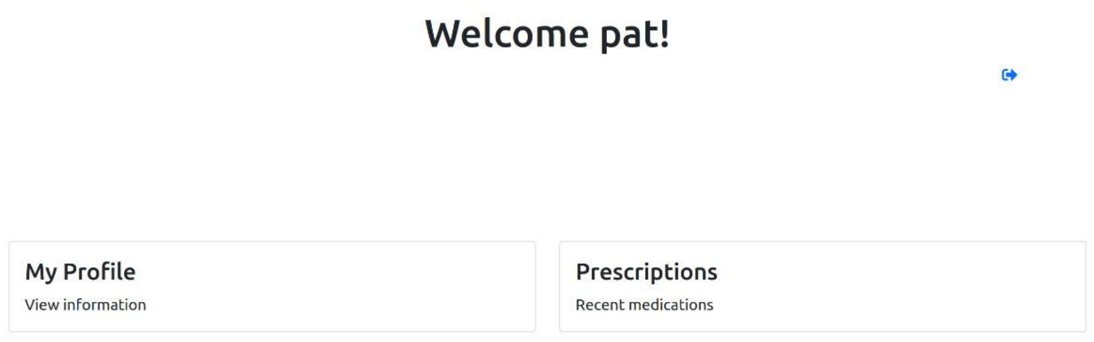
Wed Nov 24 2021

Crocin - 10mg

Aspirin - 20ml

Fig. 8

7. Patient Menu

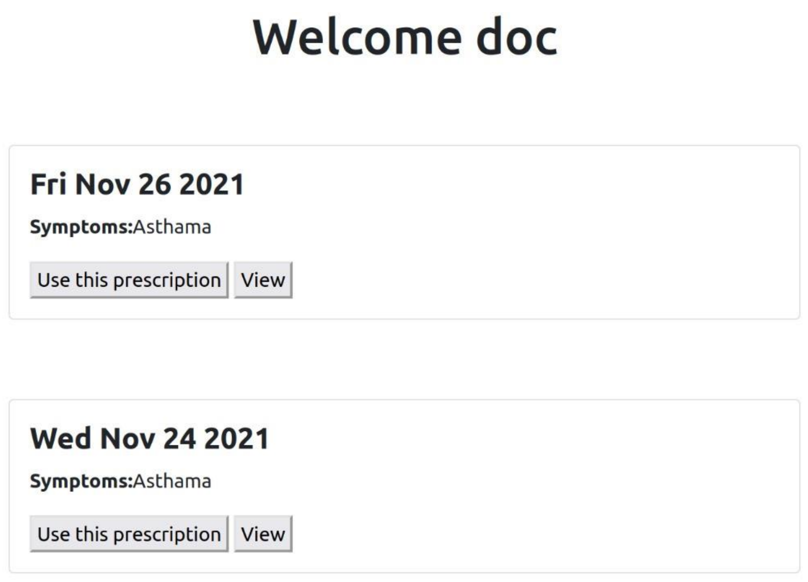

Welcome pat!

My Profile
View information

Prescriptions
Recent medications

Fig. 9

8. List of prescriptions for Doctor


Welcome doc

Fri Nov 26 2021
Symptoms: Asthama

Wed Nov 24 2021
Symptoms: Asthama

Fig. 10

9. Doctor can write a prescription

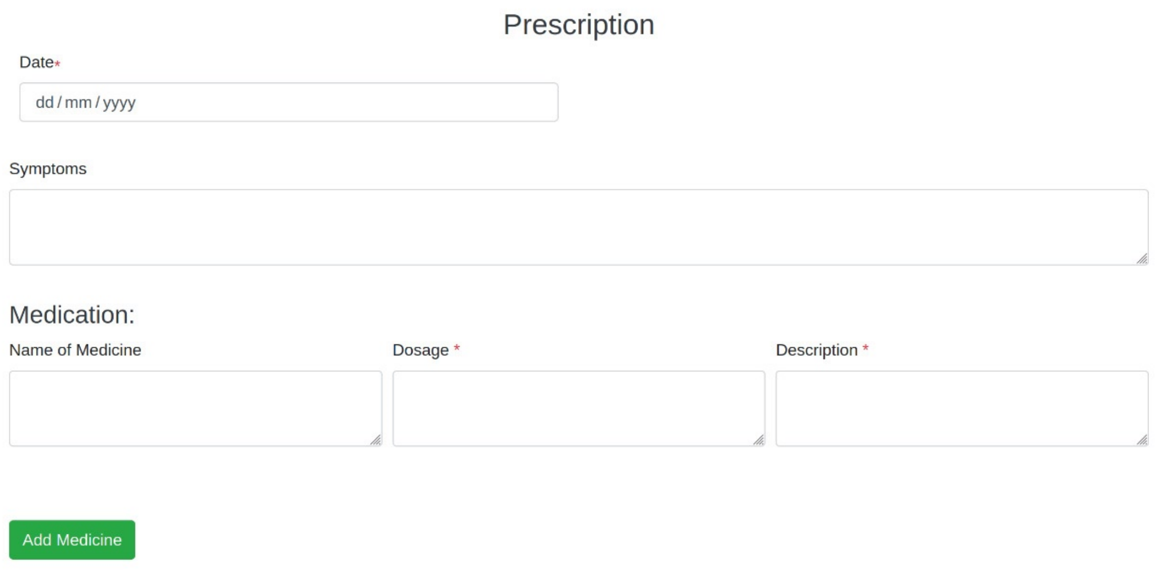
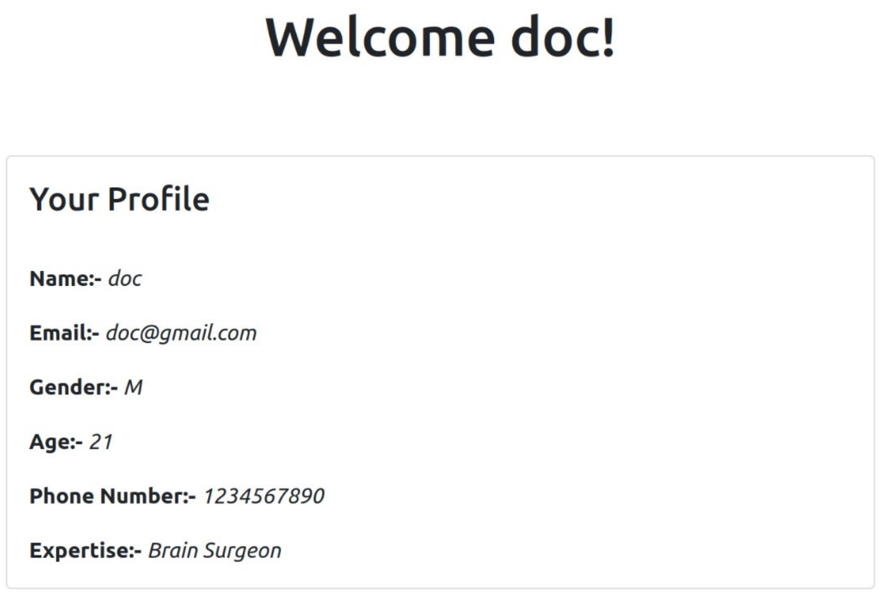


Fig. 11

10. Doctor Profile


Welcome doc!

Your Profile

Name:- *doc*

Email:- *doc@gmail.com*

Gender:- *M*

Age:- *21*

Phone Number:- *1234567890*

Expertise:- *Brain Surgeon*

Fig. 12

3.3 Functional Interface Requirements:
The website interface is very user friendly. All three types of users can easily add and delete information. First data is validated and only then inserted into the database.

3.4 Design Constraints:
We use mongodb as the database.
MongoDB has:

- Flexible document schemas.
- Code-native data access.
- Change-friendly design.
- Powerful querying and analytics.
- Easy horizontal scale-out.

Due to these immense advantages of Mongodb, the performance will be on top.

3.5 Acceptance Constraints:
The user cannot access any feature without login. The user can only access the privileges present for that particular role i.e. the patient cannot access the doctor privileges etc. The pharmacist cannot view a patient’s prescriptions without his consent, i.e without verifying the otp.

CHAPTER 4

SYSTEM DESIGN

4.1 NoSQL

NoSQL database is a type of database that provides a mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases. NoSQL data stores are commonly schema-less, providing no means for globally defining or managing the schema. NoSQL data stores vary hugely in terms of data model, query model, scalability, architecture, and persistence design. We thus resort to a (very common) classification into (1) key-value stores, (2) document stores, and (3) extensible record stores.
In this system, we are using MongoDB as NoSQL database. MongoDB is an example of Document stores that uses key-value pairs. However, MongoDB store “documents” in the value part. The term “document” connotes loosely structured sets of name-value pairs, typically in JSON (JavaScript Object Notation) or JSON.
Models are where we define our initial schema of the MongoDB’s collection. Although MongoDB is schema-less database, it does not mean we can easily doing action into the database without initial schema. Here is a sample of lecturer schema:
And for the real experience, these are the real documents that implements some concept of NoSQL Database schema design:

- Embedded documents:

```
students.profile : {
  "img_path" : "",
  "img_url" : "http://localhost:3500/static/images/profiles/qu18o8p3jd.png",
  "birthday" : "02/28/2017",
  "address" : "Pelesiran",
  "gender" : null,
  "nickname" : "Wildan",
  "fullname" : "Wildan Syahrun Nahar"
}
```

- Embedded arrays:

```
messages.members:
[
  "hendro",
  "10213075"
]
```

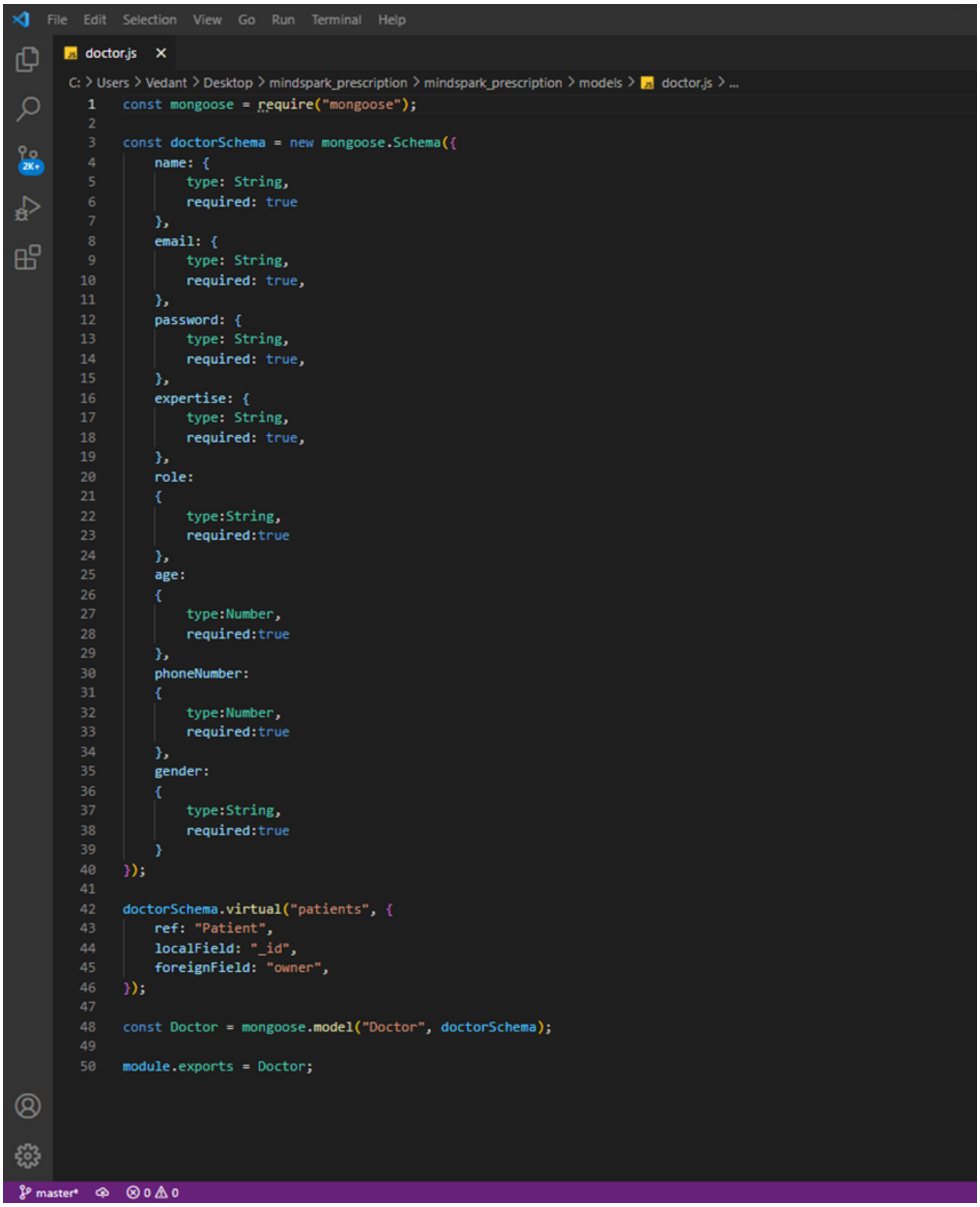
- Embedded arrays of documents:

```
students.milestones:[
  {
   "id" : 1,
    "date" : ISODate("2017-03-11T09:11:30.102Z"),
    "category" : "registered"
   },
   {
    "id" : 2,
    "date" : ISODate("2017-03-11T09:26:12.650Z"),
    "category" : "accepted"
   }
 ]
```

We can also separate the data into 2 collections. In order to connect those two, we could use method called linking : one-to-one relationship. For example, in students we have nim key where the value is int. Instead store all of the student’s report as array of documents, we can store it in different collection such as reports collection with a unique field nim.

4.2 Schema Design

Doctor's Schema:



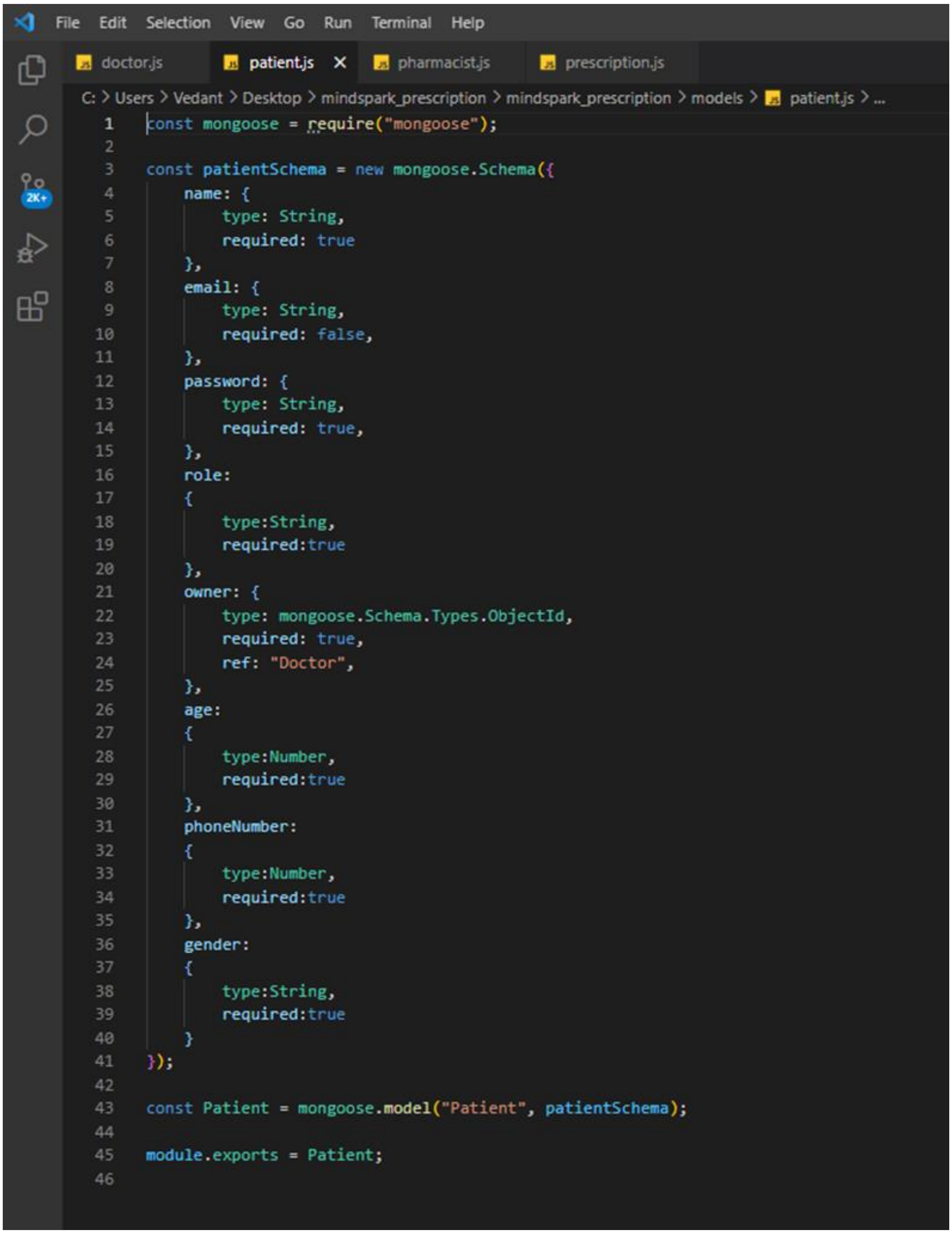
File Edit Selection View Go Run Terminal Help
doctor.js
C: > Users > Vedant > Desktop > mindspark_prescription > mindspark_prescription > models > doctor.js > ...
master*
0 0

```
const mongoose = require("mongoose");

const doctorSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true,
    },
    password: {
        type: String,
        required: true,
    },
    expertise: {
        type: String,
        required: true,
    },
    role:
    {
        type:String,
        required:true
    },
    age:
    {
        type:Number,
        required:true
    },
    phoneNumber:
    {
        type:Number,
        required:true
    },
    gender:
    {
        type:String,
        required:true
    }
});

doctorSchema.virtual("patients", {
    ref: "Patient",
    localField: "_id",
    foreignField: "owner",
});

const Doctor = mongoose.model("Doctor", doctorSchema);

module.exports = Doctor;
```

Fig. 13

The doctor schema contains name,email, password, expertise, role, age, phoneNumber,gender fields all of which are required. Age and phoneNumber are of type Number while the rest are of type string.

Patient’s Schema:

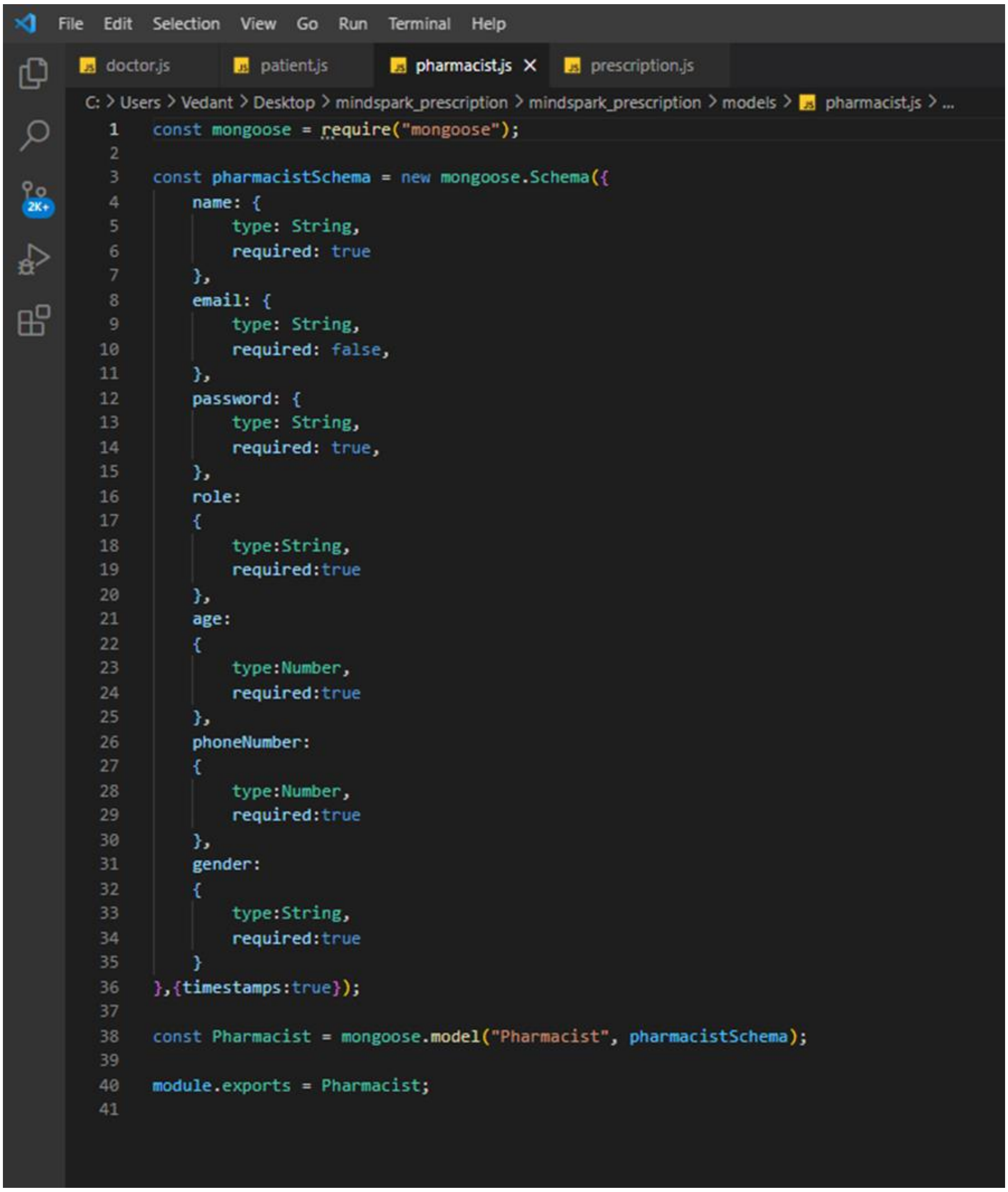

File Edit Selection View Go Run Terminal Help
doctor.js
patient.js
pharmacist.js
prescription.js
C: > Users > Vedant > Desktop > mindspark_prescription > mindspark_prescription > models > patient.js > ...

```javascript
const mongoose = require("mongoose");

const patientSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: false,
    },
    password: {
        type: String,
        required: true,
    },
    role:
    {
        type:String,
        required:true
    },
    owner: {
        type: mongoose.Schema.Types.ObjectId,
        required: true,
        ref: "Doctor",
    },
    age:
    {
        type:Number,
        required:true
    },
    phoneNumber:
    {
        type:Number,
        required:true
    },
    gender:
    {
        type:String,
        required:true
    }
});

const Patient = mongoose.model("Patient", patientSchema);

module.exports = Patient;

```

Fig. 14

The patient schema contains name,email, password, role, owner, age, phoneNumber,gender fields all of which are required. Age and phoneNumber are of type Number, owner is of type ObjectID while the rest are of type string.

Pharmacist Schema:

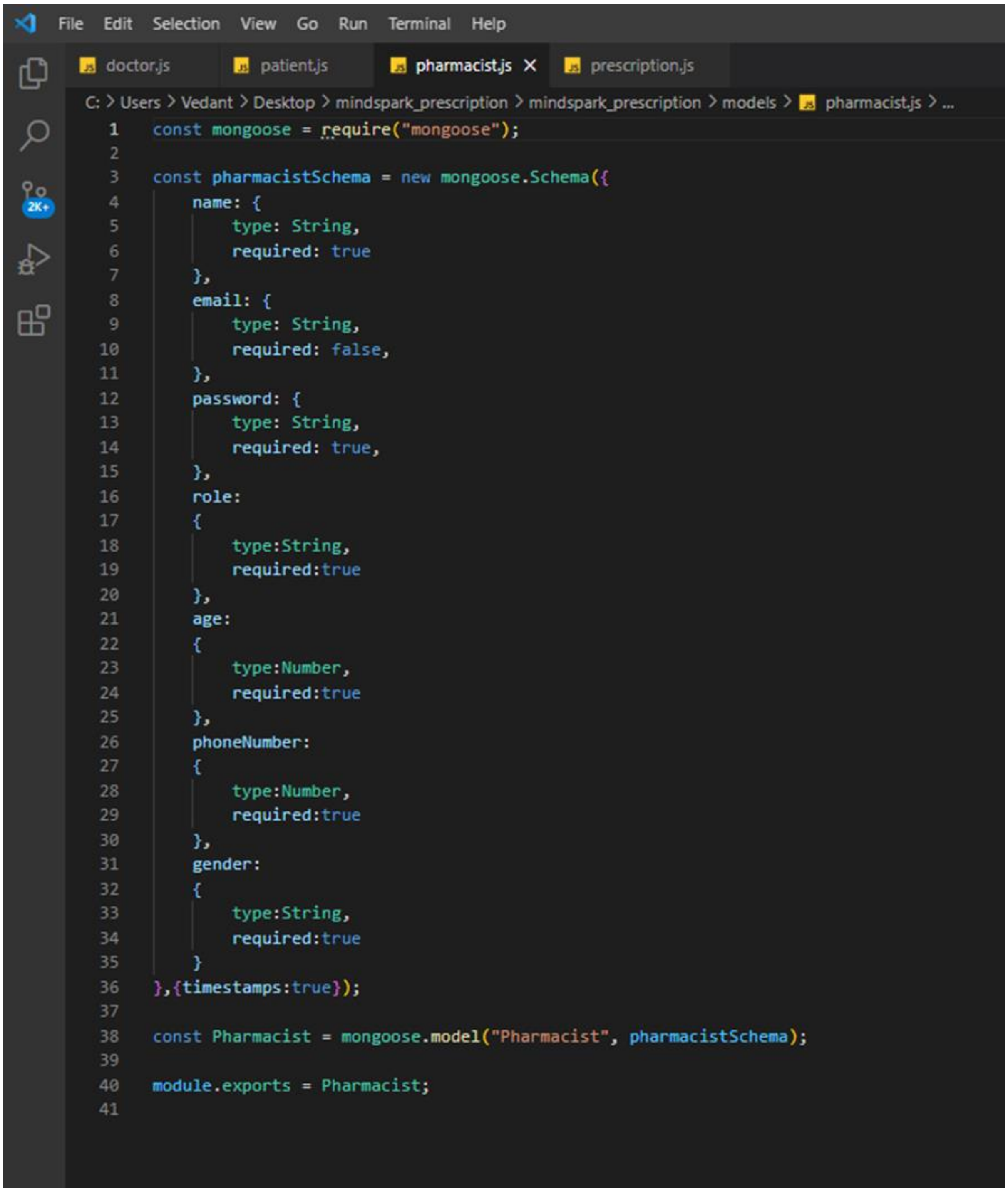

File Edit Selection View Go Run Terminal Help
doctor.js
patient.js
pharmacist.js
prescription.js
C: > Users > Vedant > Desktop > mindspark_prescription > mindspark_prescription > models > pharmacist.js > ...

```
const mongoose = require("mongoose");

const pharmacistSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: false,
    },
    password: {
        type: String,
        required: true,
    },
    role:
    {
        type:String,
        required:true
    },
    age:
    {
        type:Number,
        required:true
    },
    phoneNumber:
    {
        type:Number,
        required:true
    },
    gender:
    {
        type:String,
        required:true
    }
},{timestamps:true});

const Pharmacist = mongoose.model("Pharmacist", pharmacistSchema);

module.exports = Pharmacist;

```

Fig. 15

The pharmacist schema contains name,email, password, role, age, phoneNumber,gender fields all of which are required. Age and phoneNumber are of type Number while the rest are of type string.

Prescription Schema:


File Edit Selection View Go Run Terminal Help
doctor.js
patient.js
pharmacist.js
prescription.js
C: > Users > Vedant > Desktop > mindspark_prescription > mindspark_prescription > models > pharmacist.js > ...

```
const mongoose = require("mongoose");

const pharmacistSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: false,
    },
    password: {
        type: String,
        required: true,
    },
    role:
    {
        type:String,
        required:true
    },
    age:
    {
        type:Number,
        required:true
    },
    phoneNumber:
    {
        type:Number,
        required:true
    },
    gender:
    {
        type:String,
        required:true
    }
},{timestamps:true});

const Pharmacist = mongoose.model("Pharmacist", pharmacistSchema);

module.exports = Pharmacist;
```

Fig. 16

The pharmacist schema contains date, symptoms, medicines, owner fields. Symptoms and medicines fields are of type string, date is of type Date while owner is of type ObjectID.

4.3 System Design Flow Chart:

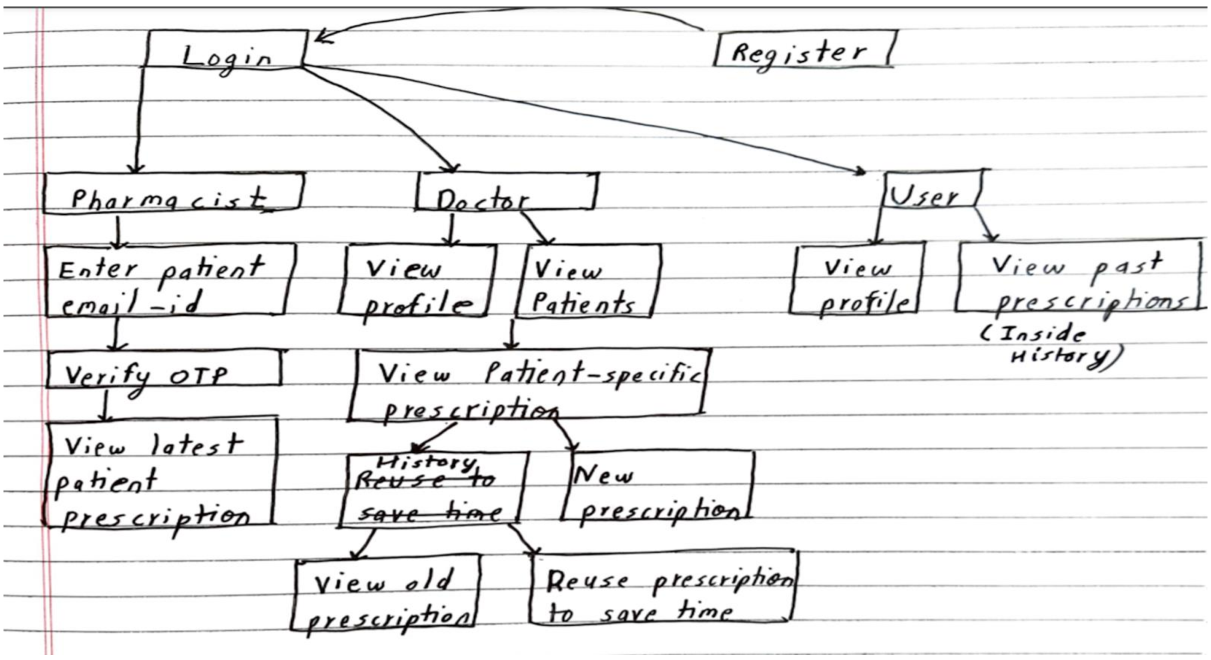


Fig. 17

CHAPTER 5

SYSTEM DESIGN

5.1 Hardware and Software Platform description

Software Platform Description:

Frontend

HTML: HTML, or HyperText Markup Language, is the standard markup language for texts intended to be displayed in a web browser. It can be aided by technology like Cascading Style Sheets (CSS) and programming languages like JavaScript.

CSS: CSS is intended to separate display from content, including layout, colours, and fonts. [3] This separation can improve content accessibility; provide more flexibility and control in the specification of presentation characteristics; allow multiple web pages to share formatting by specifying the relevant CSS in a separate.css file, reducing complexity and repetition in the structural content; and allow the.css file to be cached to improve page load speed between the pages that share the file and its formatting.

Bootstrap: Bootstrap is a free front-end framework that offers HTML and CSS-based design templates for typography, forms, buttons, tables, navigation, modals, picture carousels, and many other features, as well as optional JavaScript plugins. Bootstrap also allows you to simply develop responsive designs.

Backend:

NodeJS: Node.js is a back-end JavaScript runtime environment that is open-source, cross-platform, and runs on the V8 engine. It executes JavaScript code outside of a web browser. Node.js allows developers to utilise JavaScript to create command line tools and server-side scripting, which involves running scripts on the server to generate dynamic web page content before the page is transmitted to the user's web browser. As a result, Node.js symbolises a "JavaScript everywhere" paradigm bringing web-application development together around a single programming language rather than separate languages for server-side and client-side scripts.

Express: Express.js, or simply Express, is a Node.js backend web application framework that is free and open-source software licenced under the MIT License. It is intended for the development of web applications and APIs. It has been dubbed the de facto Node.js server framework.

Hardware Platform Description: Windows 10

5.2 Tools used

Hardware: Laptop

Software: MongoDB Compass, Microsoft VSCode

5.3 System Verification and Testing

In this project we have added various validations for name and email ID fields.User has three login options (doctor, patient, pharmacist) and a user can only login through their registered role. A pharmacist can sign up on the platform but they will only be able to login through an OTP sent to the registered email ID of the patient who has contacted them.

5.4 Future work/Extension

1. The project can be developed further by optimising the scalability of the database.
2. This application can be integrated with a mobile app.
3. Registering on the platform can be made more authentic by involving Govt agencies for verification.
4. Enabling blockchain security features for securely storing records in the database.
5. Users can be background checked to avoid fake profiles and scams.
6. Payment Gateway can be integrated in the application for a smooth end-to-end experience.
7. The software can be optimised further with respect to the frontend and backend to make it flexible, interactive and secure.
8. Machine Learning Algorithms can be applied to predict the type of disease the patient might have based on the symptoms provided by the patient.

5.5 Conclusion

In this project, a common platform for doctors, patients & pharmacists has been implemented. It solves the problem of handwritten prescriptions. A fast and secure system is designed and implemented where patient's data is protected and not accessible by any other patient or user. MongoDB is used as the database for storing doctors, patients and pharmacists data. The software can be optimised further by making it more flexible and secure.

5.6 References

[1] https://mongoosejs.com/docs/api/model.html
[2] Udemy - Complete Node.js Developer by Andrew Mead
[3] https://nodemailer.com/about/
[4] https://www.mongodb.com/products/compass
[5] https://www.npmjs.com/package/passport
[6] https://www.npmjs.com/package/bcryptjs
[7] https://www.mongodb.com/products/compass