# CHAPTER 4 SYSTEM IMPLEMENTATION

This chapter discusses system implementation. Systems implementation is a set of procedures performed to complete the design contained in the systems design document.

# PATIENT MODULE

Patients can view their personal and medical records. Patients can view their medical records, vaccination details, prescription,etc.

### Main Code:

const getpatient=asynchandler(async(req,res)=>{ res.json(req.patient)

})

//update patients with id

//route put:patients/update/:id

const updatepatient=asynchandler(async(req,res)=>{ res.json({

message:'update is working'

})

})

//get patient vaccine

const patientgetvaccine=asynchandler(async(req,res)=>{ const {patientid}=req.body

const patient\_exists=await vaccines.findOne({patientid}) if(!patient\_exists){

throw new Error('vaccine data doesnt exists')

}

const getvaccinedata=await vaccines.findOne({patientid}) res.json(getvaccinedata)

})

//get prescriptions

const patientgetprescription=asynchandler(async(req,res)=>{ const {patientid,doctorlicense}=req.body

const prescriptiondata=await prescriptions.findOne({patientid,doctorlicense}) if(!prescriptiondata){

throw new Error('prescription not existe')

}

res.json(prescriptiondata)

})

const {patientid}=req.body

const patient\_exists=await vaccines.findOne({patientid}) if(!patient\_exists){

throw new Error('vaccine data doesnt exists')

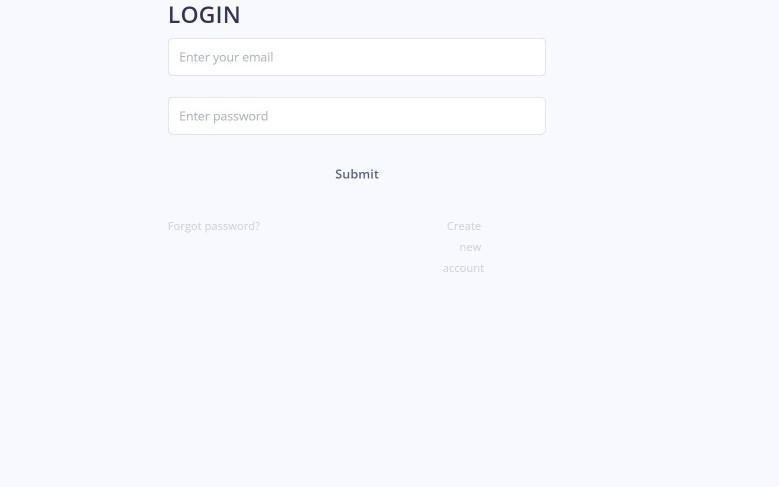
}

const updatepatient=asynchandler(async(req,res)=>{ res.json({

message:'update is working'

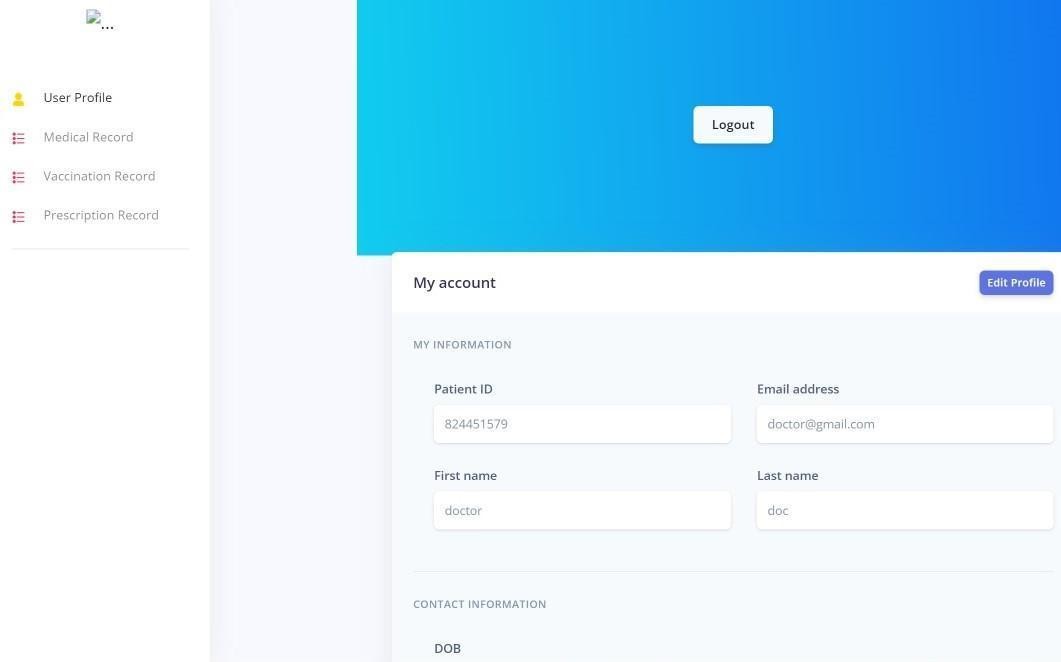
})

})



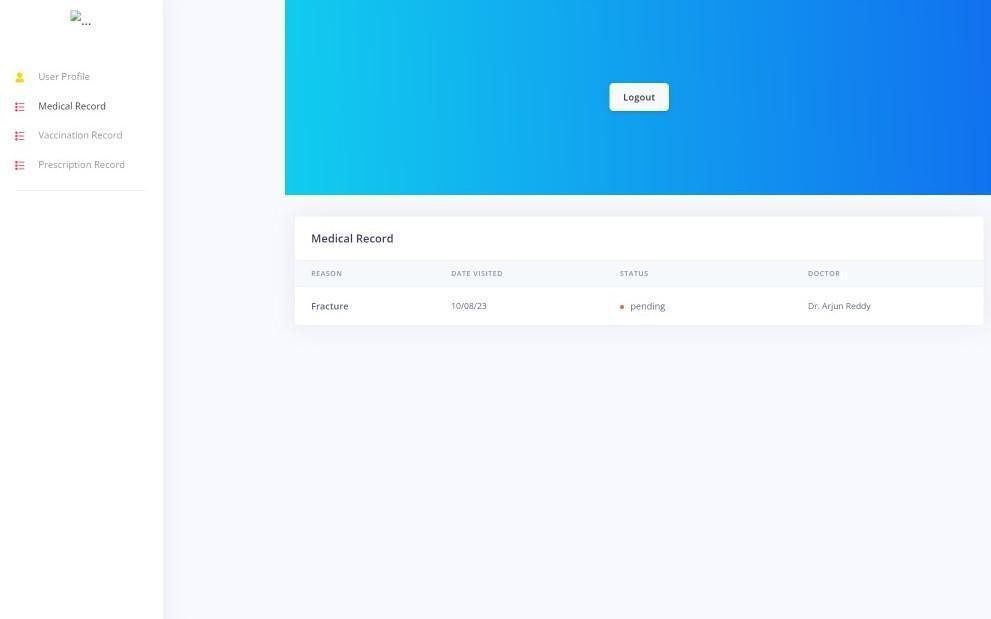
### Fig 4.1 Patient Login Page

Fig 4.1 represents the patient login page, the user enters Email ID and password to login in .



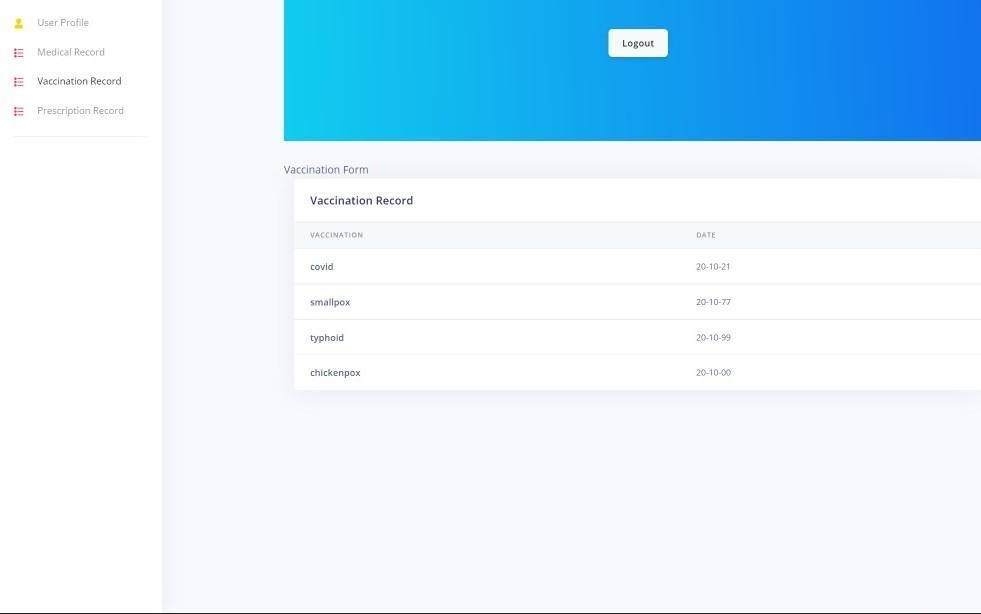
### Fig 4.2 Patient User-Profile

Patient can view their profile details and make changes as shown in Fig 4.2.

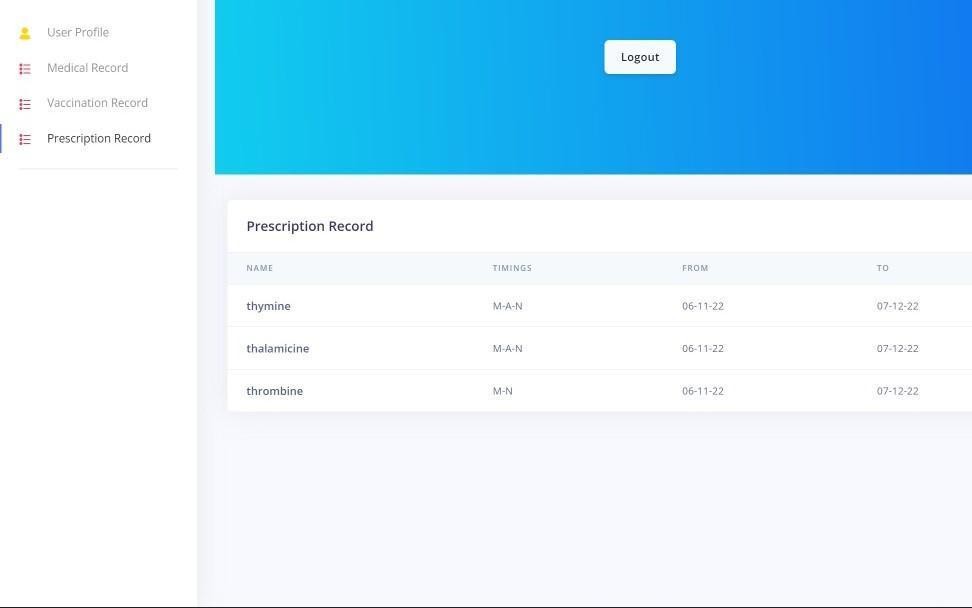


### Fig 4.3 Patient Medical Record

Patient can view their medical record and download it in PDF format as shown in Fig 4.3.



### Fig 4.4 Patient Vaccination Record



**Fig 4.5 Patient Prescription**

Vaccination and prescriptions of the patient are shown in Fig 4.4 and Fig 4.5.

# DOCTOR MODULE

Doctor can add and view the patients medical records of their respective patients. Adding of prescription, enabling of prescription and disabling of prescription of their respective patients. Doctor can view their profile, view the patient list.

## Main Code:

const createmedicalrecord=asynchandler(async(req,res)=>{

const {patientid,date,status,reason,doctorlicense,doctorname}=req.body

const createnewmedicalrecord=await patientmedicalrecordmodel.create({patientid,date,status,reason,doctorlicense,doctorname})

if(!createnewmedicalrecord){

throw new Error('medical record not created')

}

res.json(createnewmedicalrecord)

})

//get patientmedical record

const getpatientmedicalrecord=asynchandler(async(req,res)=>{ const {patientid}=req.body

const patientmedicalrecord=await patientmedicalrecordmodel.findOne({patientid})

if(!patientmedicalrecord){

throw new Error('medical record not available')

}

res.json(patientmedicalrecord)

})

//generate token

const getpatientmedicalrecord=asynchandler(async(req,res)=>{ const {patientid}=req.body

const patientmedicalrecord=await patientmedicalrecordmodel.findOne({patientid}) if(!patientmedicalrecord){

throw new Error('medical record not available')

}

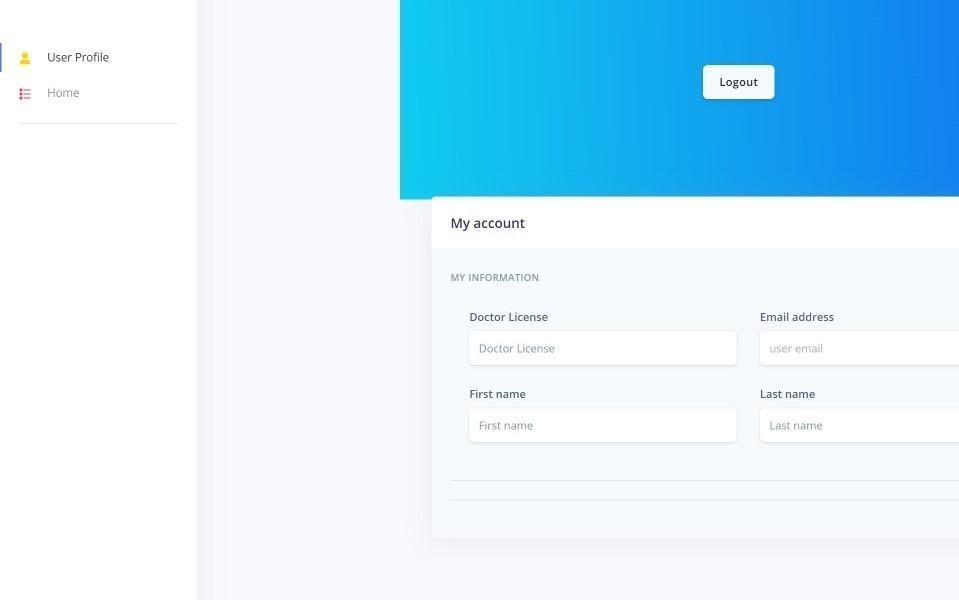
res.json(patientmedicalrecord)

})

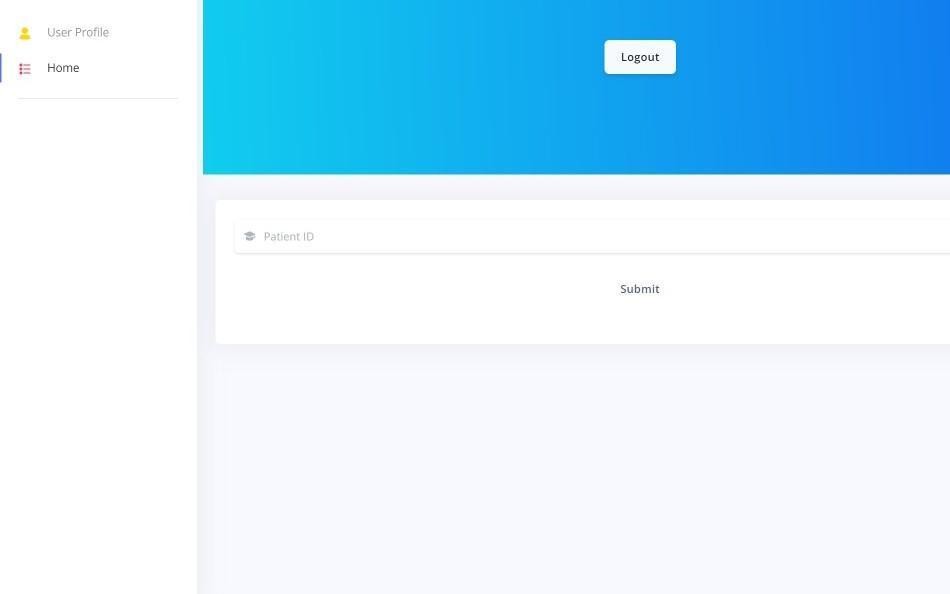
const generatetoken=(id)=>{

return jwt.sign({id},process.env.JWT\_SECRET,{expiresIn:'10d'},)

}

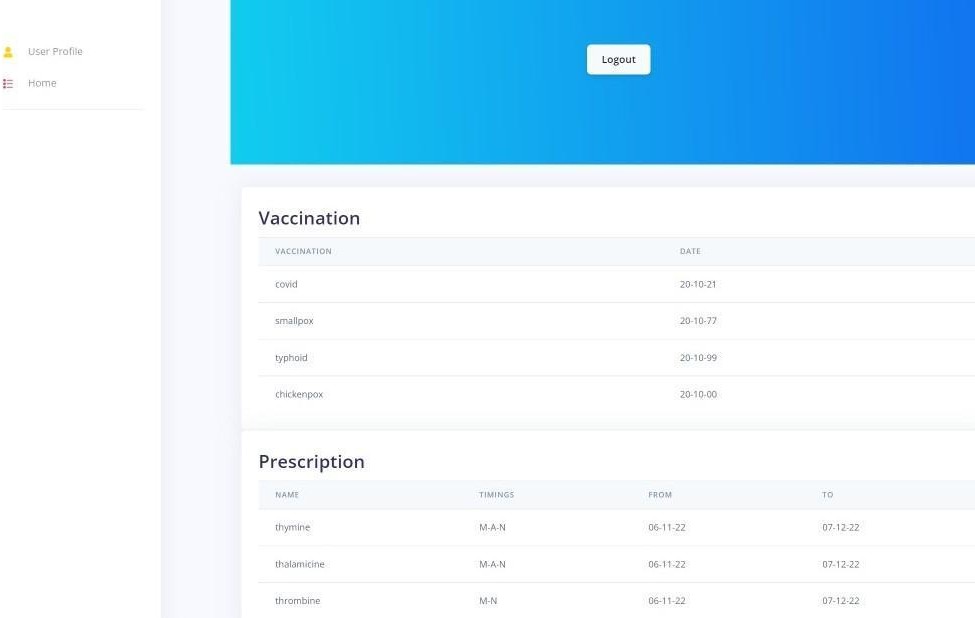


### Fig 4.6 Doctor User Profile



**Fig 4.7 Doctor Home Page**

Fig 4.6 and Fig 4.7 represents doctor can view their user-profile and enters the patient ID in the field added to their patient list.



### Fig 4.8 Doctor Viewing Patient Record

Fig 4.8 represents doctors who can view the past medical history of their respective patients. The doctor can add to the medical, vaccination record and prescription.

# PHARMACY MODULE

Pharmacy can view the active prescription of the Patients. Also, They can view the User-Profile.

### Main Code:

const asynchandler=require('express-async-handler')

const pharmacyregistration=require('../models/pharmacyregmodel') const prescriptions=require('../models/prescriptions')

const jwt=require('jsonwebtoken') const bcrypt=require('bcryptjs')

//create pharmacy

//route post:pharmacy/register

const createpharmacy=asynchandler(async (req,res)=>{ var{pharmacylicense,pharmacyname,pharmacyphone,email,password}=req.body

//finding duplicates

const pharmacyexists=await pharmacyregistration.findOne({email})

if(pharmacyexists){

throw new Error('pharmacy already exists')

}

//hash password

const salt=await bcrypt.genSalt(10)

const hashedpassword=await bcrypt.hash(password,salt)

//create pharmacy

const newpharmacy=await pharmacyregistration.create({ pharmacylicense,pharmacyname,pharmacyphone, email,password:hashedpassword

})

//display created daya res.json({

pharmacylicense,pharmacyname,pharmacyphone,email,Token:generatetoken(newpharmacy.\_id)

})

})

//login

const loginpharmacy=asynchandler(async(req,res)=>{ const {email,password}=req.body

const pharmacy=await pharmacyregistration.findOne({email}) if(!pharmacy){

throw new Error('pharmist dont exists !!!')

}

const pharmacycheck=await bcrypt.compare(password,pharmacy.password) if(!pharmacycheck){

throw new Error('invalid credentials')

}

res.json({ pharmacylicense:pharmacy.pharmacylicense,

pharmacyname:pharmacy.pharmacyname, pharmacyphone:pharmacy.pharmacyphone, email:pharmacy.email, Token:generatetoken(pharmacy.\_id)

})

})

//getpharmacy

const pharmacyget=asynchandler(async(req,res)=>{

res.json(req.pharmacy)

})

const pharmacydelete=()=>{ res.json({

message:'delete working'

})

}

//get patient prescriptions

const pharmacygetprescriptions=asynchandler(async(req,res)=>{ const {patientid,doctorlicense}=req.body

const prescriptiondata=await prescriptions.findOne({patientid,doctorlicense}) if(!prescriptiondata){

throw new Error('prescription not existe')

}

res.json(prescriptiondata)

})

res.json({ pharmacylicense:pharmacy.pharmacylicense,

pharmacyname:pharmacy.pharmacyname, pharmacyphone:pharmacy.pharmacyphone, email:pharmacy.email, Token:generatetoken(pharmacy.\_id)

})

})

const pharmacydelete=()=>{ res.json({

message:'delete working'

})

}

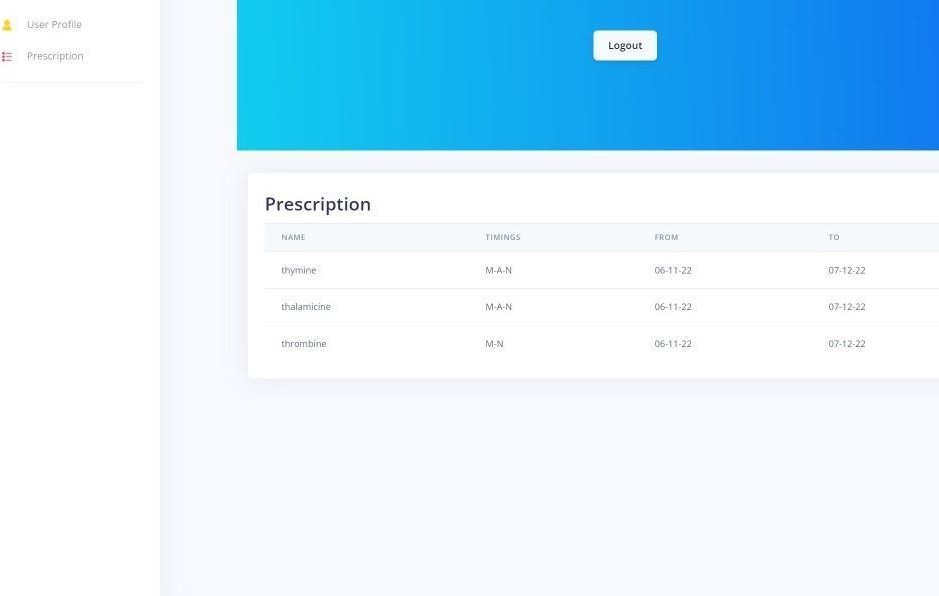
//generate token

const generatetoken=(id)=>{

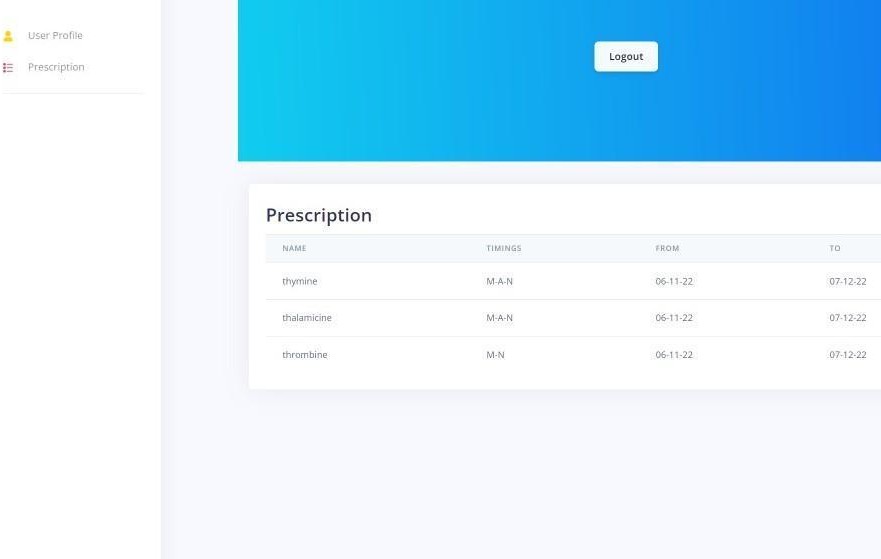
return jwt.sign({id},process.env.JWT\_SECRET,{expiresIn:'10d'},)

}

module.exports={createpharmacy,loginpharmacy,pharmacyget,pharmacygetprescriptions,pharma cydelete}



### Fig 4.9 Pharmacy User-Profile



**Fig 4.10 Pharmacy Prescription**

Fig 4.9 and Fig 4.10 represent pharmacies that can view their user-profile and prescription of the patient by entering the Patient ID.

This chapter discussed the system implementation of the project in detail.

# CHAPTER 5

**CONCLUSION AND FUTURE ENHANCEMENT**

Universal Authenticated Medical Records System manages Patient, Doctor and Pharmacy into a single integrated Application. UAHR stores the Patient medical records, prescription, vaccination records. To ensure validity of medical records, the records will be made immutable and permanent. Registered Doctors can view and add the medical records of respective patients. Pharmacy can issue medicines for only active prescription.This system is essential to maintain an authenticated electronic medical record and paperless prescriptions. Medical history can be accessed from anywhere and it is essential in emergencies.

The project can be enhanced in future by storing medical records in Blockchain to make it immutable and tamper proof.

# BIBLIOGRAPHY

**BOOKS**

1. Francesco Marchioni, Practical Enterprise Application Development, Independently published, 2019.
2. Kapila Bogahapitiya and Sandeep Nair, “Mastering Java EE 8 Application development”, Packt Publishing, 2018.

# WEBSITES

1. https://[www.w3schools.com/](http://www.w3schools.com/)
2. <https://stackoverflow.com/>
3. https://[www.mongodb.com/mern-stack](http://www.mongodb.com/mern-stack)