Project Report : Hospital Managment System

Вовед:

Hospital Management System е веб базирана апликација дизајнирана за менаџирање и поедноставување на процесите во болниците, како што се закажување термин за преглед, менаџирање со корисници (пациент, доктор или админ), како и испраќање на потсетници за тековни прегледи преку маил. Овој систем беше направен како full-stack апликација со користење на Node.js и Express.js за бекендот, React.js за фронтенд-от и PostgreSQL како база на податоци.

Главните карактеристики на системот вклучуваат автентикација на корисникот, закажување состаноци, контрола на пристап заснована на улоги и автоматизирани потсетници за е-пошта за претстојните состаноци.

Барања и употребени технологии:

Функционални барања:

- Регистрација и автентикација на корисници:

Пациентите и лекарите можат да креираат сметки и да се логираат.

Администраторите имаат контрола врз управувањето со системот и менување на улогите на секој корисник.

- Закажување прегледи:

Пациентите можат да закажат прегледи со лекарите.

Лекарите можат да потврдат или негираат прегледи.

Администраторите можат да управуваат со сите состаноци и да испраќаат потсетници со клик на копче.

- Автоматски потсетници за е-пошта:

Системот испраќа потсетници 24 часа пред термините и на пациентите и на лекарите. Ова е овозможено преку cron job

- Контрола на пристап заснована на улоги:

Различни кориснички улоги (админ, лекар, пациент) со различни пристапи во системот.

Нефункционални барања:

Безбедност: шифрирање на лозинка со bcrypt и пристап заснован на улоги до одредени дејства.

Перформанси: Системот треба да реагира и ефикасно да се справува со барањата за базата на податоци.

Употребливост: Едноставен, лесен интерфејс.

Користени технологии:

Backend: Node.js, Express.js
Frontend: React.js
База на податоци: PostgreSQL со Sequelize ORM
Автентикација: JWT, bcrypt за шифрирање на лозинка
Известувања за е-пошта: Nodemailer, MailHog како Fake SMTP сервер за време на девелопирање
Контрола на верзијата: GitHub

Дизајн на системот

Архитектура:

Системот за управување со болници ја следи архитектурата клиент-сервер, каде backend-от е одговорен за справување со API calls, интеракциите со базата на податоци и бизнис логиката, додека frontend-от е Single Page Application (SPA) базирана на React која комуницира со задниот дел преку RESTful API.

Главните архитектонски обрасци што се користат во системот се:
MVC (Model-View-Controller): Логиката од страна на серверот ја следи оваа шема.
RESTful API: Сите функционални акции, како што се автентикација на корисникот, резервирање состаноци и управување со потсетници, се ракуваат преку API endpoints.

Дизајн на базата на податоци:

Базата на податоци се состои од четири главни ентитети: Users, Appointments, Departments и Prescriptions. Сите ентитети имаат врски меѓу себе за функционалности како што се управување со улоги на корисници, прегледи и рецепти.

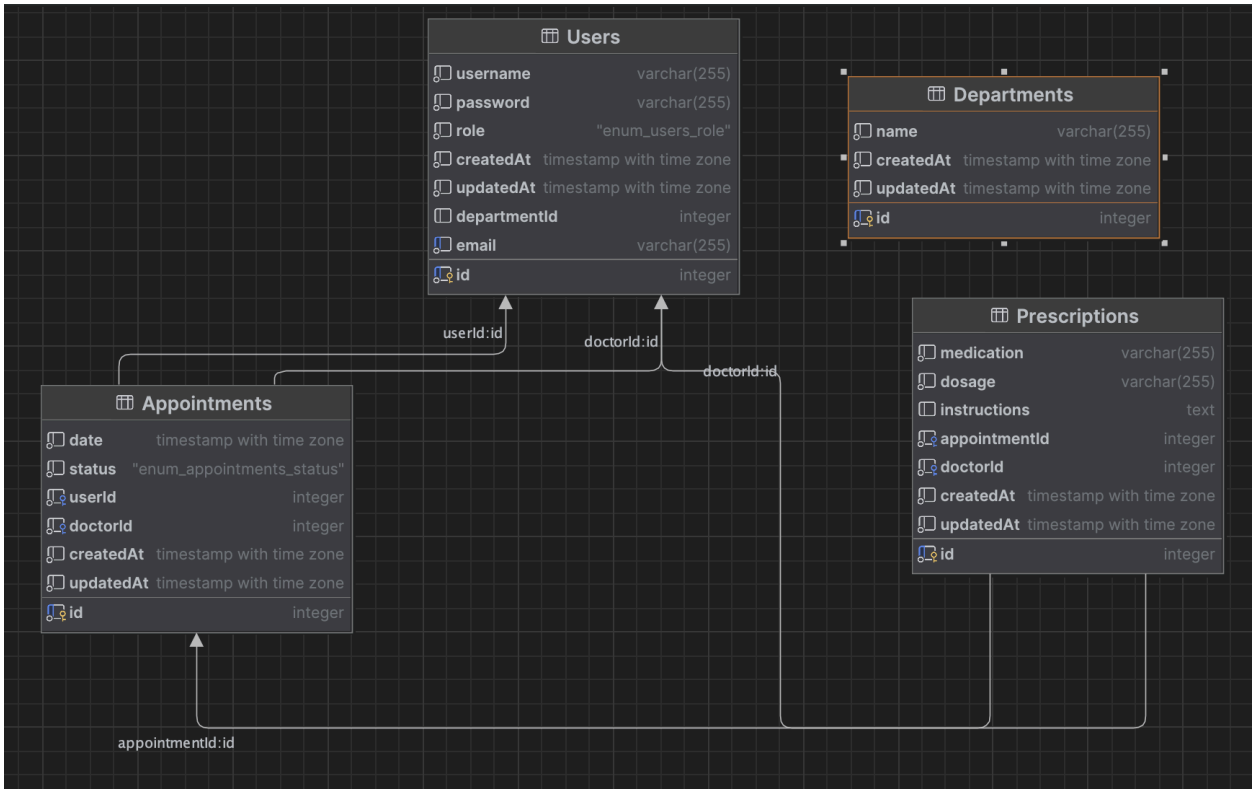
Корисници: Ги чува корисничките информации како што се корисничко име, е-пошта, лозинка, улога и ИД на одделот.
Улогите се однапред дефинирани (enum_users_role), а корисниците се класифицирани на пациенти, лекари или администратори.
Корисниците се поврзани со прегледите преку userId (за пациенти) и doctorId (за лекари).

Прегледи: Содржи детали за состаноци, вклучувајќи датум, статус, кориснички ID (пациент) и лекарски ID.

Полето за статус е enum (enum_appointments_status) што ја претставува моменталната состојба на состанокот (на пр., закажан, завршен или откажан).
Состаноците се поврзани и со пациентот и со лекарот.

Одделенија: Ги дефинира различните одделенија во болницата (на пр., кардиологија, неврологија).
Секој лекар припаѓа на оддел преку ИД на одделот.

Рецепти: Складира податоци за рецепти издадени од лекари, вклучувајќи лекови, дози и упатства.
Рецептите се поврзани со одредени термини (appointmentId) и лекари (doctorId).



Имплементација

Имплементацијата на системот за управување со болници беше развиена со користење на комбинација на современи веб технологии и за frontend-от и за backend-от, со силен фокус на употребливост, приспособливост и одржливост. Овој дел ќе ги наведе клучните компоненти и алатки што се користат за системот, фокусирајќи се на имплементираните основни функционалности.

Имплементација на Backend:

Backend-от на системот беше изграден со помош на Node.js и Express.js, кои служеа како примарна рамка за ракување со API и бизнис логика. Sequelize ORM се користеше за управување со базата на податоци, овозможувајќи непречена интеракција помеѓу Backend-от и базата на податоци.

1. Node.js обезбеди опкружување со високи перформанси, скалабилно за Backend-от , додека Express.js се користеше за управување со routes, middleweare endpoints на API. RESTful API на системот ги следи стандардните HTTP методи (GET, POST, PUT, DELETE) за операции како што се креирање термини за прегледи, управување со корисници и издавање рецепти.

2. ORM:
Sequelize беше искористена како алатка ORM (Објектно-релационо мапирање), дозволувајќи му на системот да комуницира со базата на податоци PostgreSQL користејќи JavaScript објекти наместо SQL queris.
Ентитетите опишани во делот Дизајн на база на податоци (корисници, прегледи, одделенија, рецепти) беа моделирани како модели на Sequelize со дефинирани односи.
Пример за модел на Sequelize за прегледи:

```
const Appointment : ModelCtor<Model>  = sequelize.define( modelName: 'Appointment',  attributes: {
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true,
  },
  date: {
    type: DataTypes.DATE,
    allowNull: false,
  },
  status: {
    type: DataTypes.ENUM( values: 'REQUESTED', 'CONFIRMED', 'DENIED', 'ARCHIVED'),
    defaultValue: 'REQUESTED',
    allowNull: false,
  },
  userId: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: User,
      key: 'id',
    },
  },
  doctorId: {
    type: DataTypes.INTEGER,
    allowNull: false,
    references: {
      model: User,
      key: 'id',
    },
  },
});
```

3. Известувања по е-пошта:
Една од основните карактеристики што беа имплементирани беа автоматски и рачни известувања преку е-пошта. Оваа функција е дизајнирана да испраќа потсетници до пациентите и лекарите за претстојните состаноци.
Библиотеката Nodemailer се користеше за испраќање е-пошта. Системот може да испраќа потсетници рачно (преку администраторскиот панел) или автоматски 24 часа пред закажаните состаноци.

4. Автентикација и авторизација базирана на JWT:

Со автентикацијата на корисникот се ракуваше со помош на JWT (JSON Web Tokens). Ова овозможи безбедно најавување за корисниците, а системот ги разликуваше различните кориснички улоги (администратор, лекар, пациент) користејќи контрола на пристап заснована на улоги.
Middleware делот беше имплементиран за заштита на одредени рути, осигурувајќи дека само овластени корисници можат да пристапат или менуваат одредени ресурси.

Имплементација на Frontend:

Frontend-от е изграден со помош на React.js, популарна библиотека за градење интерактивни кориснички интерфејси. Корисничкиот интерфејс на системот беше дизајниран да биде лесен и одговорен.
Главните карактеристики на Frontend:

1. React.js:
React.js обезбеди архитектура базирана на компоненти, овозможувајќи му на системот да биде модуларен, одржуван и скалабилен.
Основните функционалности на системот, како што се закажување термини за прегледи, управување со рецепти и испраќање потсетници, беа имплементирани како посебни компоненти на React.

Секоја рољa на корисниците им овозможува различен преглед на самата апликација. Има 3 компоненти: UserDashboard, DoctorDashboard и AdminDashboard.

Основен панел:
Ова е панелот кој го гледаат корисниците со User улога. Во него има можност за закажување на термини и преглед на своите рецепти.
Административен панел:
Администраторскиот панел е за да им овозможи на администраторите да управуваат со корисниците и состаноци и да испраќаат рачни потсетници за состаноци.

2. React Router:
React Router се користеше за управување со навигацијата помеѓу различни страници (на пр., најавување, контролна табла, административен панел, страница за состаноци).
Врз основа на улогата на корисникот (пациент, лекар или администратор), беа прикажани различни делови од апликацијата.

Заклучок

Системот за управување со болници успешно интегрира неколку основни функционалности потребни за ефикасно управување со болницата, вклучувајќи управување со корисници, закажување состаноци и потсетници за е-пошта. Системот обезбедува едноставен, но моќен интерфејс за пациенти, лекари и администратори.

Изградбата на овој систем ги подобри моите вештини за развој на full-stack апликација, и работа со Node.js, React и PostgreSQL.

HMS е скалабилен и може да се прошири со дополнителни функции, како евиденција на пациенти или наплата, во иднина. Севкупно, овој проект демонстрира успешна примена на принципите за развој на софтвер за решавање на реалните предизвици во здравствениот менаџмент.