МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система обліку пацієнтів ГепатаЦентру медичного інституту СумДУ»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Берест О.Б.**

Студента групи ІК.м – 91 — **Морозов М.Ю.**

Нормоконтроль — **Проценко О.Б.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра Комп'ютерних наук
Спеціальність Інформаційно-комунікаційні технології

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Морозову Михайлу Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна система обліку пацієнтів ГепатаЦентру медичного інституту СумДУ*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Інформаційний огляд. 2) Вибір методу рішення 3) Практична реалізація

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 59 стор., 22 рис., 0 табл., 4 додатків, 18 джерел.

Об’єкт дослідження — інформаційна система лікувальної установи.

Мета роботи — розробка автоматизованої інформаційної системи для обліку пацієнтів ГепатаЦентру медичного інституту СумДУ.

Методи дослідження — є теоретичний та порівняльний аналіз сучасних технологій розробки користувацьких інтерфейсів. в процесі програмної реалізації проекту було застосовано стек технологій для побудови веб-додатків MERN (MongoDB, Express, Node.js, React).

Результати — розроблена інформаційна система, з можливістю обліку пацієнтів ГепатаЦентру медичного інституту СумДУ та перегляду статистичних даних, які пов'язані з обліком пацієнтів.

СИСТЕМА МОНІТОРИНГУ ЗДОРОВ’Я, МОБІЛЬНИЙ ДОДАТОК, REACT, NEXT.JS, DOCKER, БАЗА ДАНИХ, MONGODB, КРОСПЛАТФОРМА, EXPRESS.JS, NODE.JS, JAVASCRIPT, REACT NAVIGATION, АВТОМАТИЗОВАНА ІНФОРМАЦІЙНА СИСТЕМА

ЗМІСТ

ВСТУП

Медицина є однією з найбільших галузей наукової та практичної діяльності людини. Сфера охорони здоров'я дуже важлива для життєдіяльності суспільства. У людей часто виникають проблеми зі здоров’ям і в зв’язку з цим виникає потреба в отриманні допомоги від працівників медичної сфери. Тому автоматизації процесів в цій сфері будуть завжди актуальною задачею. Також можна сказати, що це є галузь, яка потребує інновацій, адже якість лікування пацієнтів є дуже важливою.

Автоматизація може бути проявлена в різних областях медицини та взагалі людського побуту. На сьогоднішній день впровадження інформаційних технологій у сфери життєдіяльності особи, суспільства є основною рушійною силою сучасних соціальних трансформацій.

Велике значення в медицині набувають новітні технології, пов'язані з розвитком науково-технічного прогресу, зокрема, автоматизація робочого місця, впровадження комп'ютерних програм, спеціально розроблених або адаптованих для окремих сторін діяльності медичних закладів.

Використання нових технологій в медичних центрах дозволить швидше вести повний облік всіх пацієнтів, наданих послуг та дозволить легко збирати та обробляти дані. Зазвичай при автоматизації медичної установи використовуються електронні медичні карти для перегляду історії хвороби пацієнта та для перегляду медичної статистики.

Невідкладного рішення потребують питання своєчасної профілактики захворювань, ранньої діагностики патологічних процесів і надання кваліфікованої медичної допомоги. Інформаційні системи використовують для зберігання, збору, поширення, та обробки інформації.

Автоматизація має цілий ряд переваг над виконанням робіт “вручну”, оскільки вона підвищує ефективність роботи, виключає елементарні помилки, що виникають через недбалість, неосвіченість персоналу, суб'єктивність у виконанні завдань

Високоякісне та швидке медичне обслуговування та ефективна організація робочих процесів медичного закладу неможливі без використання автоматизованої медичної.

Отже, якісно спланована та побудована медична інформаційна система дозволить підвищити якість та доступність надання медичних послуг населенню.

РОЗДІЛ 1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Поняття інформаційної системи

Створення і вдосконалення комп'ютерів призвело до створення нових технологій в різних сферах життєдіяльності. Одне з найважливіших досягнень людства є Інтернет, здатний зв'язувати мільйони комп'ютерів по всьому світу.

На сьогоднішній день завдяки швидкому розвитку інформаційних технологій в програмуванні виділяють окремий напрямок – веб-програмування. Спочатку веб-сайти являли собою сукупності статичних документів. В даний час для сайтів властива інтерактивність і динамічність. Адаптивність та інтерактивність сайту можна реалізувати за рахунок широко використовуваних технологій веб-програмування, таких як CSS, Javascript і HTML. Сучасні технології веб-програмування для розробки додатків включають набір різних мов програмування та мови розмітки тексту. Ці програмні технології можливо розділити на два основні види: клієнтські і серверні. Клієнтські скрипти виконуються на стороні клієнта вже після завантаження скрипту на компютер із сервера. Серверні скрипти виконуються на стороні веб-сервера (на комп'ютері, на якому розміщена база даних та вихідні файли сайту). До цих скриптів належать nodeJS, ruby, php тощо. Ці скрипти виконуються на стороні веб-сервера ще до завантаження сторінок сайту на клієнтській стороні.

Розвиток інформаційних систем і інформаційних технологій розглядається в сучасному світі як один з ключових елементів технологічної модернізації.

Інформаційні системи пов’язані, як з системами для видачі та збереження інформації, так і з системами, котрі забезпечують обмін великої кількості інформацією в процесі управління. Ці системи застосовують методи та засоби, що дозволяють збирати, обробляти, зберігати та передавати потрібну інформацію. Особливістю таких системи є обробка інформації, яка потрібна для обробки та управління ресурсами, створення інформаційно-технічного середовища для управління діяльністю підприємства[1].

Інформаційні системи класифікують по цілому ряду різних ознак. Для побудовы ефективних інформаційних систем важливо знати наступні принципи.

Принцип інтеграції. Даний принцип полягає в тому, що дані, котрі були вже введені в систему, багаторазово використовуються для вирішення сукупності завдань[2].

Принцип комплексності, який полягає в автоматизації усіх процедур трансформування даних на всіх етапах функціонування інформаційної системи.

Принцип системності, який полягає в обробці різних даних, для отримання інформації, яка необхідна для прийняття рішень.

Інформаційні системи розглядають, як звичайні програмні продукти, але такі системи відрізняються від звичайних прикладних систем. Інформаційні системи можуть суттєво відрізнятися за своїми функціями у залежності від сфери використання. Проте можна виділити декілька властивостей, які є спільними.

1. Інформаційні системи вирішують питання зберігання, обробки і збору інформації. Базою для таких систем є середовище зберігання даних.

2. Інформаційні системи мають бути орієнтовані на користувача, який не є експертом в області обчислювальної техніки. Клієнтські програми інформаційної системи мають бути простим та зрозумілими для кінцевого користувача. Мати зручний, легко освоюваний інтерфейс, який надає всі потрібні функції для роботи кінцевого користувача і в той же час не дає йому можливість виконувати будь-які зайві дії.

1.1.2 Вимоги, що пред'являються до інформаційних систем

1. Гнучкість

Гнучкість системи є однією із головних критерій її вибору. Якщо компанія планує розвиватися та удосконалювати процеси діяльності або просто функціонує в умовах постійних змін внутрішніх та зовнішніх умов, то це є

важливим критерієм. Здатність до адаптації передбачає можливість адаптування інформаційної системи до нових умов та потреб підприємства.

2. Надійність

Вимога надійності гарантуються створенням резервних копій інформації, що зберігається, а також використанням новітніх програмних і апаратних засобів, виконання операцій протоколювання, забезпеченням якості каналів зв'язку та фізичних носіїв інформації.

3. Ефективність

Система є ефективною, якщо вона дозволяє вирішувати потрібні завдання у мінімальні терміни. Ефективна інформаційна система є результатом оптимізації даних і методів їх обробки та застосування нетрадиційних методів розробок та проектування.

4. Безпека

Під поняттям безпеки ІС, передбачається властивість системи, в силу якої сторонні особи не повині мати доступу до інформаційних системи організації. Вимога безпеки забезпечується сучасними засобами для розробки інформаційних систем, методами захисту інформації, безперервним моніторингом стану безпеки систем і засобів їх захисту.

1.1.3 Класифікація ІС

1. за ступенем автоматизації (в залежності від рівня автоматизації виділяють автоматичні, ручні й автоматизовані інформаційні системи);

2. за сферою призначення (оскільки інформаційні системи утворюються для поліпшення інформаційних потреб у рамках конкретної предметної галузі, то кожна предметна галузь відповідає свій тип ІС);

3. за місцем діяльності ІС (наукові, інформаційні системи автоматизованого керування, розробку нових виробів і технологій їхнього виробництва, різноманітні інженерні розрахунки, створення графічної документації , моделювання проектованих об'єктів, створення керуючих

програм для верстатів із числовим програмним керуванням, ІС організаційного керування, ІС керування технологічними процесами);

4. за функціональним призначенням (проектувальні, керувальні, наукового пошукуові, моделювальні, діагностичні, систем підготовки прийняття рішення).

ІС, як система управління, дуже схожа, як з системами видачі та збереження інформації, так і з системами, що забезпечують обмін інформацією в процесі управління. Ці системи охоплють сукупність методів та засобів, що дозволяють користувачу зберігати, збирати, передавати і обробляти інформацію.

Інформаційні системи пов’язанні з процесами життєдіяльності людини та мають велике значення в повсякденному житі людини. Такі системи доцільно використовувати через те, що для нормального функціонування суспільних процесів необхідний обмін інформації та передача знань між окремими членами суспільства та цілими поколіннями.

1.2 Медичні інформаційні системи

Новітні медичні організації зберігають та обробляють величезні обсяги даних. Якість та швидкість медичної допомоги залежить від ефективної обробки інформації, яка використовується лікарями, керівниками, які керують підрозділами. Від цих факторів залежить загальний рівень життя населення та рівень розвитку країни. Створення інформаційних систем для медичних установ є необхідним фактором при використанні великої кількості обновлюваної інформації, яка використовується при вирішенні діагностичних, адміністративних та інших завдань.

До недавнього часу в нашій країні абсолютно повністю були відсутні які-небудь ознаки автоматизації в сфері охорони здоров'я. Медичні картки, процедурні та аналізі звіти, облік лікарів, лікарських препаратів та пацієнтів - весь документообіг проводився на папері. Такий підхід значно повільніший, а отже, може впливати на якість обслуговування пацієнтів та затрудити роботу

медичного персоналу, а також призвести до лікарських помилок, великих витрат часу на заповнення звітів та медичних карток. Це перевантажувало керівництво і роботу органів контролю.

Більш того, стає очевидним той факт, що від ефективності впровадження інформаційних технологій в медицині вже в недалекому майбутньому буде залежати здоров'я, а значить, і процвітання всієї нації.

Медична інформаційна система (МІС) — це система, призначена для обробки даних, збереженими та зібраними в будь-якому медичному закладі. Інформаційні системи медичних закладів можуть використовуватися кожним у и сфері, від пацієнтів до лікарів. Такі системи збирають дані та обробляють їх таким чином, щоб їх можна було використовувати для швидкого прийняття потрібних рішень у сфері охорони здоров'я[3].

Однією в головних особливостей МІС є перехід від локальної роботи з медичною інформацією до автоматизованої системи, де всі дані, котрі використовуються в медичному закладі, доступні з єдиного інформаційного середовища. В такій системі реалізується безпаперова технологія, але користувачі завжди мають можливість отримання "твердої копії" будь-якого документа. Сучасні медичні технології дозволяють оптимізувати управління будь-якими структурними медичними підрозділами, підвищити якість та швидкість надання медичних послуг та дозволяє створити конкуренцію на світовому рівені медичного обслуговування.

Мета використання медичної інформаційної системи полягає в оптимізації лікування пацієнтів за допомогою швидкого надання актуальних даних про пацієнтів кожному медичному працівнику, який надає допомогу пацієнту.

Медичні інформаційні системи включають:

1. Електронний медичний запис.

Електронний медичний запис призначений замінити паперову версію записів про пацієнта. Даний запис може містити інформацію про стан здоров'я пацієнта в будь-який момент, інформацію щодо алергій, історії хвороби,

інформацію про попередні діагнози та результати аналізів. Медичні працівники мають можливість використовувати ці дані для отримання інформації про огляд хвороби пацієнта, щоб процес діагностики був швидше.

2. Практичне програмне забезпечення для управління

Такі ІС допомагають медичним працівникам швидше керувати процесам закладів, а саме планування пацієнтів та перегляд медичних послуг. Також медичні працівники мають можливість використовувати системи управління практикою. Це передбачає автоматизацію адміністративних завдань, що виконуються в рамках ведення бізнес процесів на об'єкті.

3. Віддалений моніторинг пацієнтів

Медичні працівники можуть працювати з даними пацієнта з дому. RPM знижує витрати, пов'язані з хронічним доглядом та реадмісією в лікарнях, а також надає можливість надавати якіснішу медичну допомогу.

RPM може контролювати рівень глюкози в крові і артеріальний тиск, що дуже корисно для пацієнтів з хронічними захворюваннями. Дані, які були зібрані та передані через RPM, можуть використовуватися медичним працівником для виявлення медичних подій, таких як інсульт або серцевий напад, які вимагають негайного медичного втручання. Зібрані та оброблені дані можуть бути використані для дослідження здоров'я пацієнта або дослідницького проекту.

4. Портали пацієнтів

Кожен пацієнт має можливість переглядати особисті дані щодо здоров'я, таких як інформація про призначені ліки та результати аналізів через портали пацієнтів. Також портали пацієнтів надають можливість в будь-який час спілкуватися зі своїми лікарями, дізнатися про поповнення рецепту і можливість зустрічі.

Однією з важливих складових роботи будь-якого медичного закладу є автоматизовані робочі місця для спеціалістів. Це автоматизовані інформаційні системи, що потрібні для автоматизація всього робочого процесу лікаря окремої спеціалізації: ведення документації, лікувально-профілактичної, планування

роботи та отримання різноманітної допоміжної інформації. Ці системи забезпечують інформаційну підтримку у прийнятті лікувальних, діагностичних, та організаційних рішень.

1.3 Актуальність медичних інформаційних систем

Медична інформаційна система призначена для інформаційного забезпечення як основних, так і допоміжних бізнес-процесів медичного закладу.

Використовування автоматизованої системи підвищує ефективність і якість надання медичної допомоги за рахунок тих можливостей, які забезпечує комп'ютер в здійсненні збору, обробки, зберігання, подання і використання медичної інформації, необхідної для адекватного вирішення лікувально-діагностичних завдань

Головною метою, що підштовхує медичні організації впроваджувати інформаційні системи, є те, що галузь охорони здоров'я намагається підвищити ефективність та продуктивність закладу, а також скоротити витрати. Автоматизація більшості ручних процесів може бути важливою частиною для реалізації продуктивної роботи медичного закладу. Використання автоматизації для заміни ручних завдань, котрі швидше можуть виконатися машиною, може значно заощадити час.

МІС можуть виконувати наступний функціонал:

- реєстрація, обробка, структуризація та створення інформаційного простору.
- Аналіз і контроль роботи установ, управління ресурсами установи.
- Обміну інформацією.
- Статистичний аналіз даних.
- Контроль якості та ефективності надання медичної допомоги.
- Підтримка прийняття рішень.
- Зберігання та пошук інформації.
- Підтримка економічної складової лікувального процесу.

Дані системи відносяться до медико-технологічних інформаційних систем. Вони використовуються для довідково-інформаційного забезпечення прийняття рішень у професійній діяльності лікарів різної спеціалізації. Дані системи дозволяють підвищити якість роботи в умовах масового обслуговування. Інформаційні системи є цінним інструментом, який надає допомогу допомагає керівництву та лікарям у забезпеченні бездоганного догляду пацієнта.

Потрібно сказати, що автоматизація не може замінити лікарів і медсестр. Однак, автоматизація може бути використана разом з робочими процесами, щоб зробити цей процес ефективнішим і підвищити продуктивність.

Завдяки впровадженню автоматизованих інформаційних технологій можна збільшити пропускну здатність медичного закладу. Медичний працівник, який використовує новітні засоби автоматизації, може приймати та обробляти більшу кількість пацієнтів за один раз. Замість того, щоб зменшувати чи збільшувати кількість працівників, коли обсяг пацієнтів зростає чи зменшується, ця платформа може гнучко масштабувати роботу з групами різних розмірів[4].

До переваг, котрі можуть бути реалізовані шляхом застосування автоматизації медичної сфери можна віднести:

- аналіз даних: медична галузь постійно створює дані. Медичні інформаційні системи допомагають збирати та обробляти дані про здоров'я пацієнта, щоб допомогти медичним працівникам покращити надання медичних послуг та зменшити витрати на охорону здоров'я. Аналіз даних медичних закладів може поліпшити догляд за пацієнтами;
- управління здоров'ям населення – дані, котрі були зібранні під час лікування пацієнтів можуть допомогти визначити закономірності, передбачити або запобігти спалаху, виявити групи ризику та багато іншого;
- обмін медичними даними між медичними працівниками та науковими дослідниками з метою розробки новітніх медичних препаратів та терапій;

- зменшення відходів за рахунок відмови від використання паперових документів.

1.4 Огляд існуючих рішень

Ринок медичних інформаційних систем переживає період бурхливого розвитку. Зараз існує досить велика кількість медичних інформаційних систем, які існують для вирішення завдання автоматизації діяльності медичних закладів.

Розглянемо більш детально декілька таких інформаційних систем.

Medods - платформа для організації роботи медичний закладів. Дану платформу можно використовувати в локальних и хмарних версіях. Також додаток дозволяє записувати пацієнтів на прийом та вести медичні картки в електронному вигляді, автоматично формувати різні види документів. Наявність онлайн-запису, адміністративна панель керівника, вбудована взаємодія з телефонією UIS та з іншими телефонами по API є важливими перевагами Medods. Але дана система не дуже гнучка.

Рисунок 1.1 — платформа Medods

MedElement - медична інформаційна система, яка надає можливість використання новітніх технологій, таких як хмарних сервісів і довідкових системи для лікарів, науківців. Дану МІС можна використовувати для автоматизація роботи клінік, стоматологій, аптек. Головною особливістю MedElement є те, що вона має потужну довідкову систему. В даній системі містяться велика кількість довідників різноманітних захворювань, лікарських засобів, розміщуються огляди світових практик та ін. Також , вона має усі переваги хмарних систем: формування звітів, що підтримує автоматизацію всієї медичної документації, збір актуальної маркетингової інформації, облік фінансів та послуг.

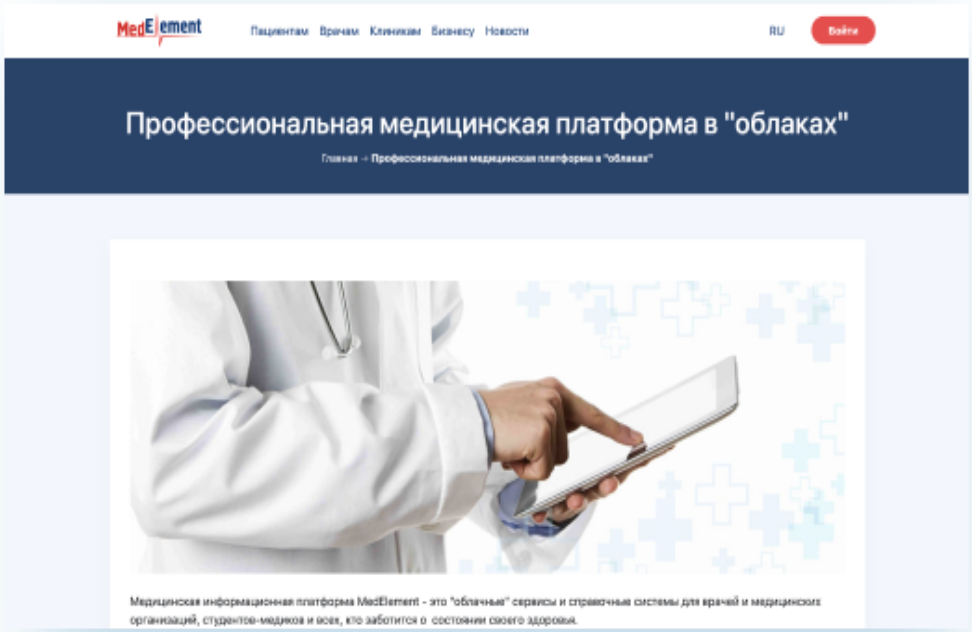

Рисунок 1.2 — платформа MedElement

Clinic365 - більш спеціалізоване CRM-рішення для медичних закладів, ніж комплексна МІС. Дану CRM можна використовувати, як на сервері, так и у хмарі. Clinic365 надає можливість обліку пацієнтів та фінансових взаємовідносин з пацієнтами. Однією з головних особливостей МІС Clinic365 є можливість побудувати більш швидкий та гнучкий алгоритм роботи з пацієнтом. Також CRM підтримує інтеграція телефонії.

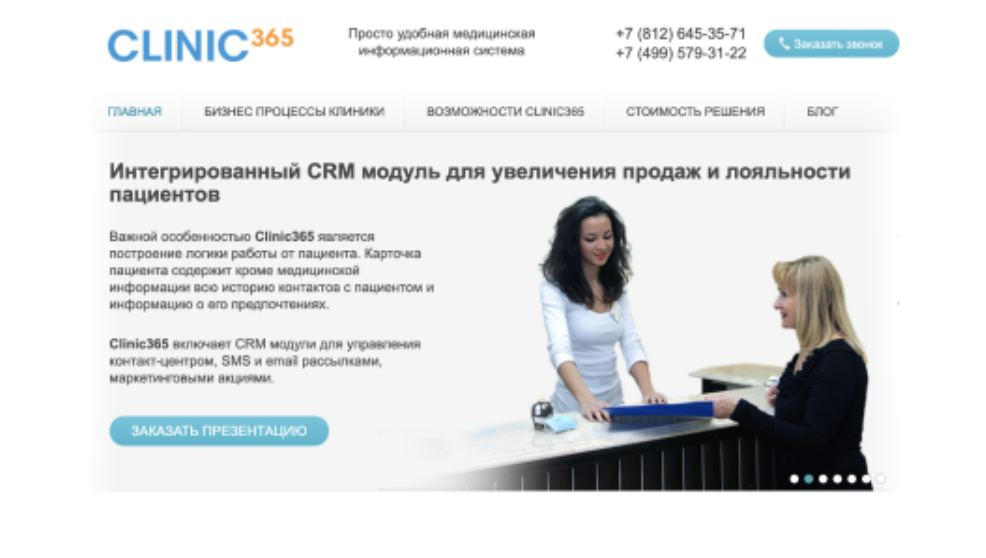

Рисунок 1.3 — платформа Clinic365

1.5 Постановка задачі

Впровадження новітні технологій в медичній сфері дозволить вести повний облік всіх пацієнтів та швидше надавати медичні послуг. Високоякісне та швидке медичне обслуговування та ефективна організація робочих процесів медичного закладу збільшить якість та доступність надання медичних послуг населенню.

Розробка інформаційної системи обліку пацієнтів ГепатоЦентру є доцільною та відповідно постає завдання створення такої системи.

РОЗДІЛ 2. ВИБІР МЕТОДІВ ВИРІШЕННЯ

Зараз існує багато різноманітних мов програмування, але варто сказати, що не існує універсальної мови програмування, що краще усіх інших. Для вирішення певних задач можна використовувати переваги будь-якої мови програмування, які виявляються тільки в контексті певної задачі. На даний момент можна ефективно вирішити майже будь-яку задачу за допомогою будь-якої з сучасних популярних мов програмування. Під час дослідження було обрано наступний стек технології: React.js, Redux, Node js, MongoDB, JavaScript, Docker.

2.1 Технології для реалізації front-end частини

Щоб реалізувати клієнтську частину інформаційної веб-системи було розглянуто багато технологій та було обрано такі технології: мова програмування JavaScript, Next.js, JavaScript-бібліотека React.js, Redux, скриптова метамова SCSS. Розглянемо кожну із зазначених технологій детальніше.

2.1.1 Мова програмування JavaScript

На сьогоднішній день світ веб-додатків дуже важко уявити без мови JavaScript. JavaScript - це мова, що робить веб-сторінки, які ми кожен день переглядаємо в своєму браузері інтерактивними.[7]

Мова JavaScript була створена в 1995 році в компанії Netscape в якості мови сценаріїв в браузері Netscape Navigator 2. Спочатку мову називали LiveScript, але на той час була дуже популярна мови програмування Java та LiveScript перейменовали в JavaScript. Однак даний момент до цих пір іноді призводить до певної плутанини: деякі початківці розробники вважають, що Java і JavaScript мало не один і той же мова. Ні, це абсолютно дві різні мови, і вони пов'язані тільки за назвою.

Спочатку JavaScript мав досить невеликими можливостями. Дану мову

програмування використовували в тому, щоб додати динамічної поведінки на веб-сторінку. За допомогою JS можно будо обробляти натискання кнопок на веб-сторінці ти провести будь-які інші дії, пов'язані перш за все з елементами управління.

Однак розвиток веб-програмування та поява HTML5[6] і технології Node.js відкрило багато нових можливостей в JS. Зараз JavaScript також використовується для створення веб-додатків, але він став більше, надає набагато більше можливостей.

Також він застосовується як мову серверної сторони. Тобто якщо раніше JavaScript застосовувався тільки на веб-сторінці, а на стороні сервера нам треба було використовувати такі технології, як PHP, ASP.NET, Ruby, Java, то зараз завдяки Node.js ми можемо обробляти всі запити до сервера також за допомогою JavaScript.

Останнім часом переживає бум сфера мобільного розробки. І JavaScript знову ж таки не залишається осторонь: збільшення потужності пристроїв і повсюдне поширення стандарту HTML5 призвело до того, що для створення додатків для смартфонів і планшетів ми також можемо використовувати JavaScript.

При початковому розвитку веб-технологій існувало кілька веб-браузерів (Netscape, Internet Explorer), які надавали різні можливості. Для того щоб уніфікувати різні реалізації до загального виду і стандартизувати мову під керівництвом організації ECMA був розроблений стандарт ECMAScript. В принципі самі терміни JavaScript і ECMAScript є багато в чому взаємозамінними і відносяться до одного і того ж мови.

Організацією ECMA було розроблено декілька стандартів мови, які відображають етапи його розвитку. Останнім прийнятим на сьогоднішній день стандартом є ECMAScript 2015 (ES 6). Але треба сказати, що реалізація цього стандарту в поширених веб-браузерах дуже далека до завершення, і, можливо, на його повне впровадження піде кілька років. Тому в цьому посібнику розглядаються переважно стандарт ES5 і ті функціональні можливості, які вже

доступні у всіх популярних браузерах.

JavaScript є мовою, що інтерпретується. Це означає, що скрипти на мові JS виконується за допомогою інтерпретатора. Інтерпретатор отримує інструкції мови JavaScript, які визначені на веб-сторінці, виконує їх.

JavaScript сам по собі досить гнучкий. Розробники написала велику кількість інструментів (бібліотек) над основою мови JavaScript, які надають величезну кількість додаткових функцій з дуже невеликим зусиллям. До них відносяться:

1. Програмні інтерфейси додатка (API), вбудовані в браузери. Дані інтерфейси забезпечують різні функціональні можливості, такі як установку стилів CSS, динамічне створення та видалення HTML, маніпуляція і захоплення відеопотоків, генерація 3Д графіки і аудіо семплів та робота з веб-камерою користувача.

2. Відкриті API дозволяють розробникам використовувати функціональність в свої сайти від інших розробників, таких як Google, Amazon Twitter або Facebook.

3. Зараз існує багато фреймворків і бібліотек, котрі можно застосувати до вашого HTML та CSS, що дозволить вам прискорити створення сайтів і додатків.

Мова JavaScript використовується для:

- написання сценаріїв веб-сторінок для надання їм інтерактивності;
- створення одно сторінкових веб-застосунків (React, Vue.js, AngularJS);
- програмування на стороні сервера (Node.js);
- стаціонарних застосунків (Electron, NW.js);
- мобільних застосунків (React Native, Cordova);
- сценаріїв в прикладному ПЗ.

Головною перевагою JS є його простота та популярність. Ця мова можна використовувати для розробки на усі мобільні платформи, в тому числі для розробки додатків на мобільні операційні системи Android і IOS. JavaScript

можна використовувати і в багатьох інших областях: обробка та аналіз даних, створення робототехніки, створення призначених для користувача інтерфейсів і т.д.. Довгий час JavaScript використовували для розробки front-end частини клієнт-серверних додатків, де в якості клієнта використовується браузер. В 2009 році JavaScript стає універсальною мовою програмування, завдяки появи платформи Node.js. Дана платформа дозволяє використовувати мову JS не тільки на клієнтській, але і на стороні сервера додатка.

JavaScript, наразі, є однією з найпопулярніших мов програмування в інтернеті.

2.1.2 Бібліотека React.js

React - це бібліотека JavaScript, яка використовується для створення призначеного для користувача інтерфейсу. Компанія Facebook створила React та перший реліз бібліотеки світ побачив у березні 2013 року.

Передусім React призначався для розробки веб-сайтів, проте пізніше з'явилася платформа React Native та Elector. React Native вже призначався для мобільних пристроїв.

React представляється ідеальний інструмент, що надає можливість розробляти масштабовані веб-додатки (в даному випадку мова йде про fron-end), особливо в тих ситуаціях, коли додаток являє SPA (одно сторінкове додаток).[9]

React має гарну документацію та є відносно простим в освоєнні, має зрозумілий та лаконічний синтаксис.

React дозволяє розробникам створювати великі веб-додатки, котрі можуть використовувати дані, які змінюються, без перезавантаження сторінки. Мета використання полягає в тому, щоб веб-застосунок був швидким, простим, масштабованим. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими бібліотеками, такими як mobX, redux.

React для забезпечення високої швидкості роботи використовує технологію Vitrual DOM. В пам’яті зберігається спрощена копія DOM, де за

вузлами закріплені конкретні екземпляри компонентів. Коли змінюється стан екземпляра, відбувається процес оновлення, що складається з таких етапів:

- Компоненти перевіряються на зміни;
- DOM в пам’яті перебудовується;
- Обраховується різниця з реальним деревом DOM та вносяться безпосередні зміни.

Концепція ReactJS надає можливість створювати незалежні компоненти і збирати їх разом у більш складні компоненти. Параметри передаються кожному із компонентів.

Рух даних по компонентам додатку відбувається лише в одному напрямку. Це дає вам спрошує контроль за рухом даних і дозволяє більш ефективно відстежувати зміни.

React демонструє доволі простий підхід до програмування, без дуже складних концепцій. Дана бібліотека легко інтегрується з усіма бібліотеками JavaScript. Також React має навколо себе дуже велику екосистему. Його активна спільнота постійно розробляє нові бібліотеки та багато інших корисних речей для інших розробників.

Зазвичай усі компоненти React написані за допомогою JSX - розширення синтаксису JavaScript, що нагадує XML, яке дозволяє використовувати синтаксис усі HTML теги для відображення компонентів. Код, який був написаний на JSX компілюється в виклики методів бібліотеки React[11].

На погляд багатьох фронтенд — розробників, React простіше для вивчення, ніж Angular або Ember — він набагато менше і добре працює з jQuery і іншими фреймворками.[8] Він, до того ж, надзвичайно швидкий, так як використовує віртуальний(virtual) DOM і оновлює тільки змінені частини сторінки (звернення до DOM досі є самою повільною частиною сучасних WEB-додатків, тому дана бібліотека і отримує перевагу в продуктивності, оптимізуючи його). 16 Клієнтський роутинг в цьому випадку можна реалізувати за допомогою бібліотеки react-router.

2.1.3 Redux

Redux — це JavaScript бібліотека з відкритим вихідним кодом, призначена для управління станом додатки[13]. В основному його використовують разом з React для створення користувацьких інтерфейсів.

Управління станом — це, по суті, спосіб полегшити спілкування та обмін даними між компонентами. Він створює відчутну структуру даних, яка представляє стан вашої програми, з якої ви можете читати та писати. Таким чином, ви можете бачити інакше невидимі стани, працюючи з ними.

Більшість бібліотек, таких як React, Angular тощо, побудовані таким чином, щоб компоненти могли внутрішньо керувати своїм станом без потреби у зовнішній бібліотеці чи інструменті. Це добре для додатків з невеликою кількістю компонентів, але в міру того як додаток стає більшим, управління станами, спільними між компонентами, стає звичною роботою.

У програмі, де дані обмінюються між компонентами, може бути заплутаним насправді знати, де має проживати держава. В ідеалі, дані в компоненті повинні містити лише один компонент, тому обмін даними між братами-компонентами стає важким.

Зрозуміло, що управління станом стає брудним, оскільки додаток ускладняється. Ось чому нам потрібен інструмент управління станом, такий як Redux, який полегшує підтримку цих станів.

Redux не є новим, але він залишається досить популярним.

Спосіб роботи Redux простий. Існує центральне сховище, яке зберігає весь стан програми. Кожен компонент може отримати доступ до збереженого стану без необхідності пересилати реквізити з одного компонента на інший.

Є три будівельні частини: дії, сховище та редьюсери. Давайте коротко обговоримо, що робить кожен із них. Це важливо, оскільки вони допомагають зрозуміти переваги Redux та спосіб його використання. [10]

Простіше кажучи, дії - це події. Це єдиний спосіб, яким ви можете надіслати дані із вашої програми до вашого магазину Redux. Дані можуть бути з взаємодії користувачів, викликів API або навіть подання форми.

Редьюсери - це чисті функції, які приймають поточний стан програми, виконують дію і повертають новий стан. Ці стани зберігаються як об'єкти, і вони вказують, як змінюється стан програми у відповідь на дію, надіслану до сховища.

Сховище зберігає стан програми. У будь-якій програмі Redux є лише один магазин. Ви можете отримати доступ до збереженого стану, оновити стан та зареєструвати або скасувати реєстрацію слухачів за допомогою допоміжних методів.

Redux суворо ставиться до того, як повинен бути організований код, що полегшує комусь, хто знає Redux, зрозуміти структуру будь-якої програми Redux. Як правило, це полегшує обслуговування.

Архітектура Redux дозволяє реалізувати принцип поділу відповідальностей в додатках, розбиваючи їх на чотири блоки[14] (див. рис. 1.1).

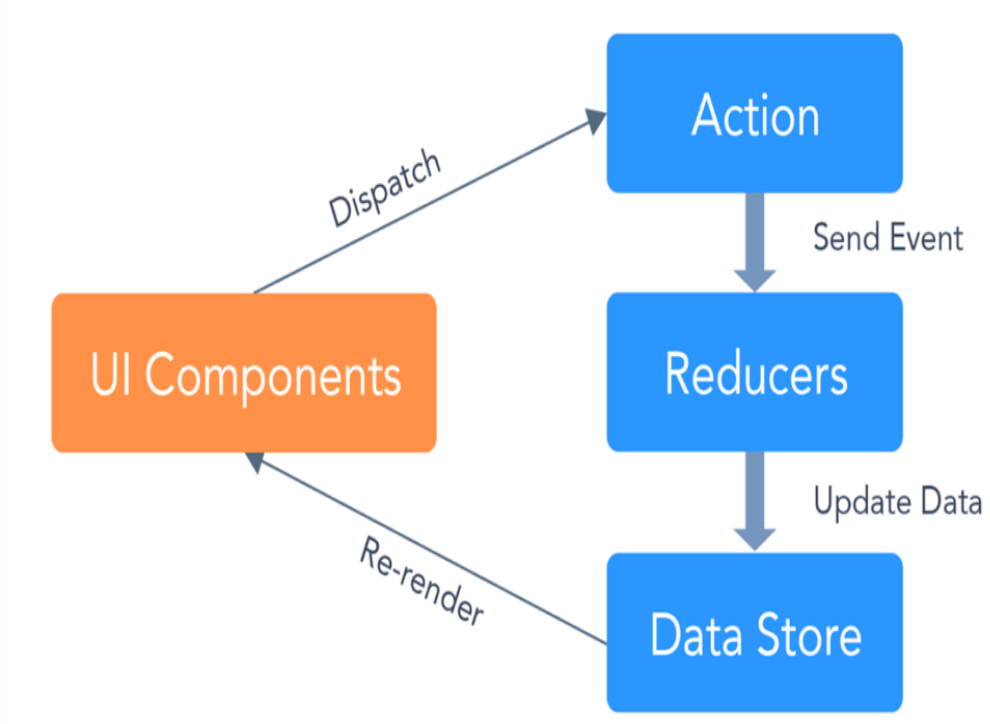

Рисунок 2.1 — Архітектура Redux

2.2 Технології для реалізації back-end частини

Для повної реалізації back-end частини інформаційного веб-додатку було використано такі інструменти: платформу Node.js, СКБД MongoDB та Docker. Розглянемо детальніше дані технології.

2.2.1 Node.js

Всі говорять про те, що Node.js переймає світ програмування та його великі переваги. Хоча всі фреймворки та мови мають деякі переваги, але саме Node.js бере на себе розробку на підприємствах.

Node.js - це серверна платформа, побудована на JavaScript Chrome від Google Chrome (V8 Engine)[5]. Node.js був розроблений Райаном Даль в 2009 році.

Node.js - це міжплатформене середовище виконання з відкритим кодом для розробки серверних та мережевих додатків. Додатки Node.js написані на JavaScript і можуть запускатися в середовищі виконання Node.js в OS X, Microsoft Windows та Linux[12].

Node.js також надає багату бібліотеку різноманітних модулів JavaScript, що значною мірою спрощує розробку веб-додатків за допомогою Node.js.

Найважливіші переваги Node включають:

це дозволяє дуже швидко створювати програми з великим трафіком у режимі реального часу (наприклад, чати чи ігри)

- дозволяє кодувати в JavaScript як для клієнта, так і для сервера;
- підвищує ефективність процесу розробки, оскільки заповнює прогалину між розробниками інтерфейсів та серверних систем (про це далі);
- постійно зростаючий NPM (Node Package Manager) надає розробникам безліч інструментів і модулів для використання, таким чином, додатково підвищуючи їх продуктивність;
- код виконується швидше, ніж будь-якою іншою мовою;

- Node ідеально підходить для мікросервісів, які є популярним рішенням серед корпоративних додатків.

Будь-яка мова програмування дасть вам кілька причин вибрати їх серед інших. Річ у Node.js полягає в тому, що він був розроблений для масштабних додатків. Сучасні інструменти та охоплення перспективним способом розробки складних додатків роблять Node.js відокремленим від інших технологій програмування.

2.2.2 MongoDB

Ґрунтуючись на наведених вище вимог, в якості СУБД вирішено вибрати MongoDB. MongoDb - це документа-орієнтована база даних. Тобто кожна запис - це документ без жорстко заданої схеми, який може містити вкладені документи. MongoDB має широкий функціонал і на даний момент є однією з найпопулярніших NoSQL систем.

Перша публічна версія MongoDB була випущена в 2009 році, а тепер це одна з найпопулярніших в світі NoSQL. MongoDB дозволяє оперувати JSON-документами, що зберігаються в колекціях, які є аналогом звичних SQL-таблиць. Для роботи з документами передбачені операції пошуку, вставки, видалення і оновлення. Для пошуку документів в колекції використовується метод запитів за зразком, підтримуються сортування, проекція, перегляд результатів запиту за допомогою курсору. Масштабованість в MongoDB досягається за рахунок поділу документів з колекції по вузлах на підставі обраного ключа (shard key). Підтримується асинхронна реплікація в режимі «головний-підлеглий»: операції записи обробляються тільки головним вузлом, а читання можуть здійснюватися як з головного вузла, так і з одного з підлеглих. Клієнт може працювати в різних режимах: асинхронному (не чекаючи відгуку) або блокуючому (чекаючи підтвердження від існуючих в розподіленої мережі вузлів). Таким чином, MongoDB підтримує різні моделі узгодженості в залежності від того, чи дозволені читання з вторинних вузлів і від скількох вузлів очікуються підтвердження під час запису. Ця система

використовується у великому числі великих компаній і проектів, серед яких SourceForge, Foursquare, The Guardian, Forbes, The New York Times та інші.

Вибір саме документно-орієнтованої СУБД заснований на тому, що на відміну від реляційних аналогів тут не накладаються обмеження на набір полів у документа. Так як документо-орієнтовані бази даних ще не так поширені як реляційні, але їхня популярність із кожним роком все збільшується, доцільно буде роз'яснити деякі нюанси термінології документо-орієнтованих баз даних. У короткому підручнику з MongoDB The Little MongoDB Book від автора KarlSeguin наведені шість основних концепцій MongoDB:

1. MongoDB - концептуально те ж саме, що звичайна, звична нам база даних (або в термінології Oracle - схема). Усередині MongoDB може бути нуль або більше баз даних, кожна з яких є контейнером для інших сутностей.

2. База даних може мати нуль або більше «колекцій». Колекція є аналогом таблиці з реляційних БД. Колекції складаються з нуля або більше «документів». Документ є аналогом записи в таблиці (рядки). Документ складається з одного або більше «полів», поля, в свою чергу, є аналогом "колонки".

3. «Індекси» в MongoDB майже ідентичні таким в реляційних базах даних.

4. Колекція з зафіксованим розміром. СУБД підтримує колекції з фіксованим розміром, які зберігають розмір вставки.

Вибір припав саме на MongoDB через її розвиненості, популярності і наявності хорошої і повної документації. Основними плюсами MongoDB для даного проекту, крім її документо-орієнтованості, стали:

1. Схожий з реляційними СУБД підхід до зберігання даних: база данних-колекція-документ-поле. Наприклад, у основного конкурента MongoDB, CouchDB, відсутнє поняття будь-якого поділу документів на групи і всі документи є рівнозначними, а для поділу документів, наприклад, за типом, необхідно додавати спеціальні поля.

2. Формат зберігання даних. Дані в MongoDB зберігаються в форматі BSON. BSON - це практично те ж саме, що і JSON, але в бінарному вигляді. Той факт, що дані зберігаються саме в бінарному вигляді, значно прискорює час їх обробки.

3. Структура документів в MongoDB дозволяє зберігати документи всередині документів, такий підхід в MongoDB називається ненормалізована модель даних.

2.2.3 Docker

У сучасному світі віртуалізації домінують технології гіпервізор віртуальних машин, наприклад, VMware, KVM і Xen, хоча є і альтернативні підходи. До числа останніх відноситься і набирає популярність технологія контейнерів компанії Docker на базі відкритого вихідного коду. Модель віртуалізації на базі гіпервізора, використовувана VMware, Xen та KVM, передбачає запуск віртуальної машини, що включає цілу ОС. Docker ж дотримується іншої підходу, в якому додаток поміщається в віртуальний контейнер, що функціонує поверх єдиною операційної системи базового рівня, що значно економить ресурси.

Віртуальні машини пропонують нам віртуалізацію на апаратному рівні в той час, як Docker Containers пропонують віртуалізацію на рівні операційної системи. Для уникнення неочікуваної поведінки застосунку використовувались віртуальні машини. Основна проблема у тому, що VM — це додаткова ОС, що працює поверх хостової, а це додаткові гігабайти для проекту. Docker розв’язує перераховані проблеми, розділяючи ядро ОС між усіма контейнерами, що працюють як окремі процеси хостової ОС.

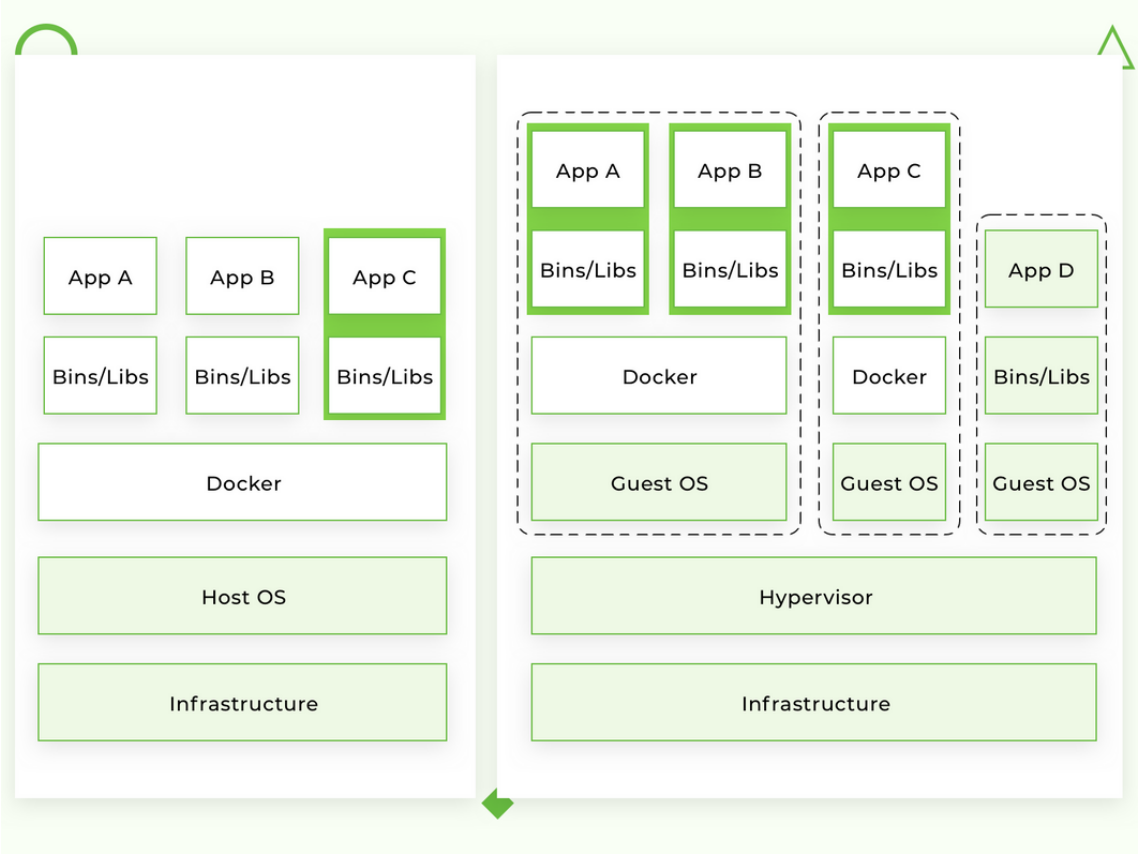

Рисунок 2.2 — Різниця між Docker та VM

У 2013 році Docker представив те, що стане галузевим стандартом для контейнерів. Контейнери - це стандартизована одиниця програмного забезпечення, що дозволяє розробникам ізолювати свою програму від оточення, вирішуючи головний біль "це працює на моїй машині". Для мільйонів розробників сьогодні Docker є фактичним стандартом для створення та обміну контейнерними програмами - від робочого столу до хмари. Ми базуємось на своєму унікальному підключеному досвіді від коду до хмари для розробників та команд розробників.

Підхід Docker до контейнеризації фокусується на можливості видалити частину програми для оновлення або ремонту, не видаляючи всю програму. На додаток до цього підходу, заснованого на мікросервісах, ви можете спільно використовувати процеси між різними програмами приблизно так само, як це робить сервісно-орієнтована архітектура (SOA).

Кожен файл зображення Docker складається з ряду шарів, які об'єднані в одне зображення. Шар створюється при зміні зображення. Щоразу, коли

користувач вказує команду, наприклад, запустити або скопіювати, створюється новий шар.

Docker повторно використовує ці шари для побудови нових контейнерів, що прискорює процес будівництва. Проміжні зміни розподіляються між зображеннями, додатково покращуючи швидкість, розмір та ефективність. Для шарування також властивий контроль версій: кожного разу, коли відбувається нова зміна, ви по суті маєте вбудований журнал змін, що забезпечує повний контроль над вашими зображеннями контейнерів[16].

Запуск нового обладнання, його експлуатація, підготовка та доступність займали дні, а рівень зусиль та накладних витрат був обтяжливим. Контейнери на основі Docker можуть скоротити розгортання до секунд. Створюючи контейнер для кожного процесу, ви можете швидко ділитися цими процесами з новими програмами. Оскільки операційна система не потребує завантаження для додавання або переміщення контейнера, час розгортання істотно коротший. У поєднанні з коротшим часом розгортання ви можете легко та економічно ефективно створювати та знищувати дані, створені вашими контейнерами, без занепокоєння.

2.3 Прототипи екранів інформаційної системи

Прототип - це базовий макет сайту, який віалізує розташування всіх елементів і функцій. Він дозволяє наочно проілюструвати всі задумки, а також внести правки ціною мінімальних зусиль і витрат.

Для створення UI/UX дизайну було застосовано Figma. Вона дозволяє повністю з створити будь-який повноцінний дизайн, логотип чи проекцію у повному обсязі.

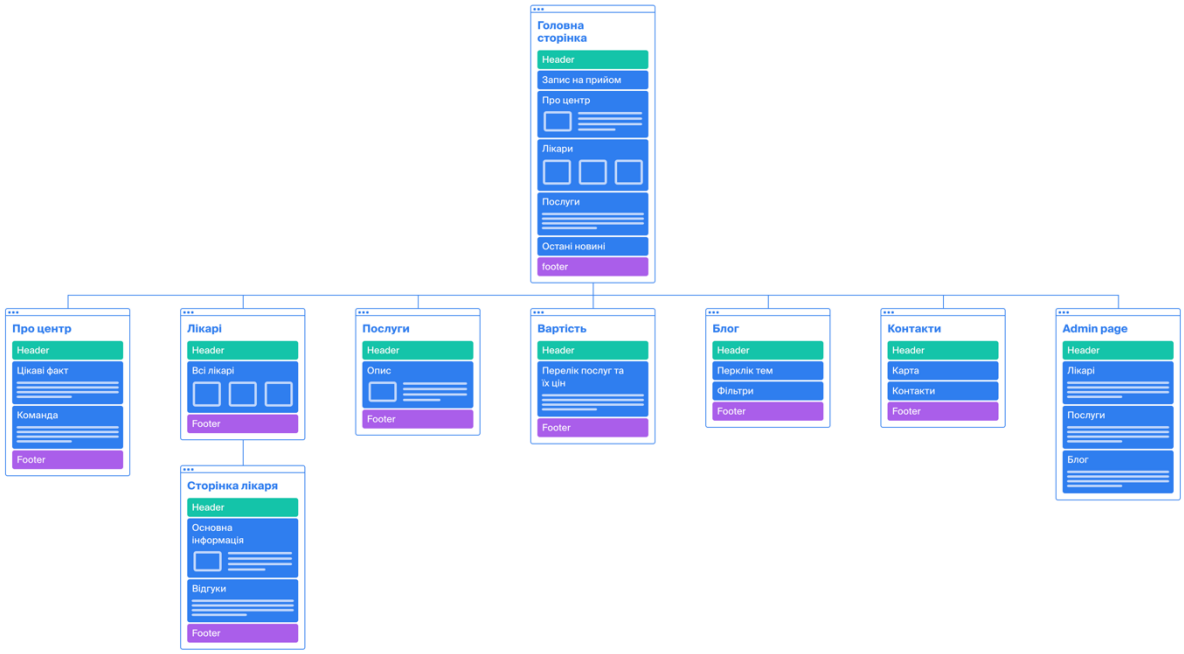

Рисунок 2.3 — Взаємозв’язок екранів інформаційної системи

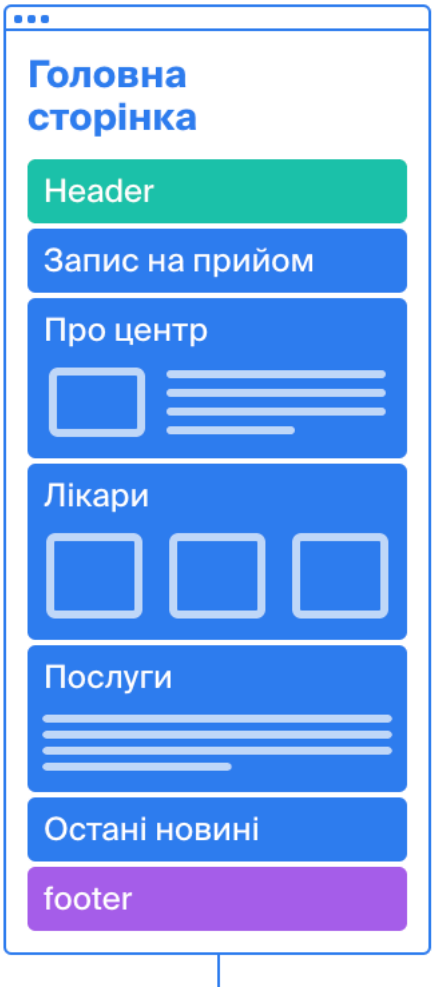

Рисунок 2.4 — Прототип головної сторінки

На прототипі (Рисунок 2.4) відображається головна сторінка системи. Також є можливістю переходити на інші сторінки системи за допомогою головного меню.

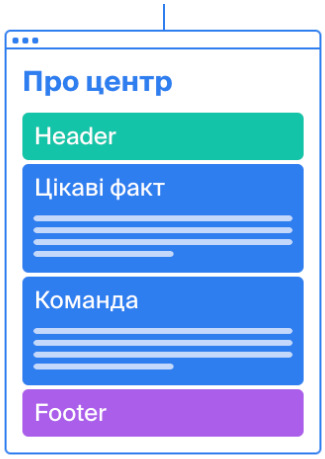

Рисунок 2.5— Прототип екрану детальної інформації про центр

Прототип (Рисунок 2.5) відображає головну інформацію про медичний центр. На ньому зображено основна інформація центра і можливість перейти на всі доступні сторінки інформаційної системи.

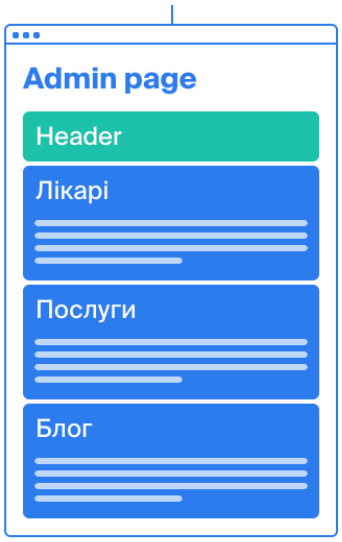

Рисунок 2.6— Прототип сторінки адміністратора

Прототип (Рисунок 2.6) відображає сторінки адміністратора. На цій сторінці адміністратор має можливість створювати, редагувати та видаляти послуги, новини та лікарів медичного центру.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Розробка моделей бізнес-процесів

На етапі створення концептуальних моделей для опису бізнес-діяльності використовуються моделі бізнес-прецедентів та діаграм видів діяльності, для опису бізнес-об'єктів - моделей бізнес-об'єктів та діаграм послідовності[18].

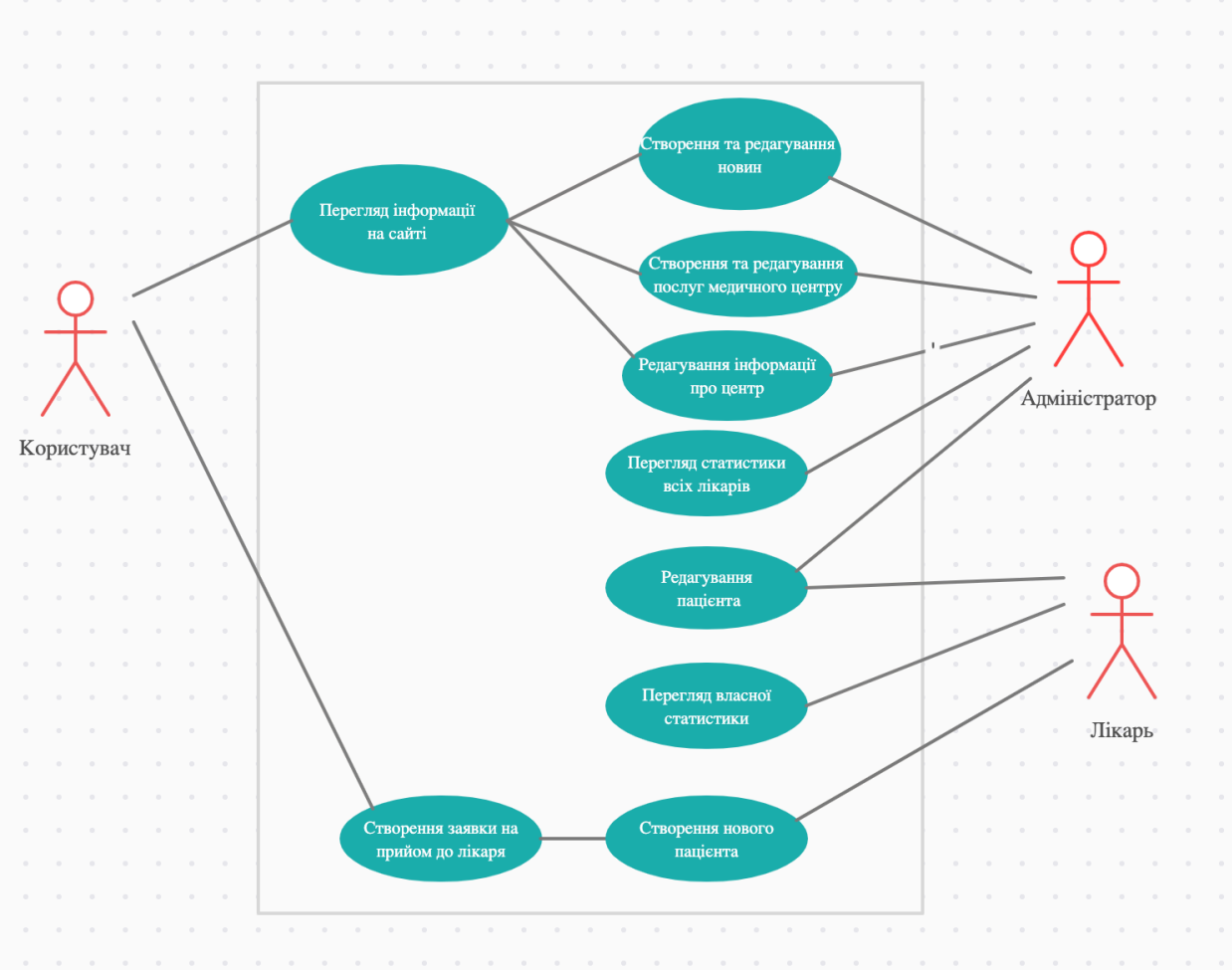

Рисунок 3.1 – Діаграма прецендентів

3.2 Розгортання додатку

Розгортання відбувається за допомогою Docker та Docker Compose. За ідеологією Docker для кожного мікро сервісу ми повинні робити свій контейнер. Це пов’язано з fault tolerance[17]. Тому що помилка в одному мікро сервісі потягне за собою інші. Тому ми будемо робити кожний мікросервіс в

окремому контейнері. Вивід консолі при розгортанні додатку за допомогою Docker Compose можна побачити на Рисунку 3.2.

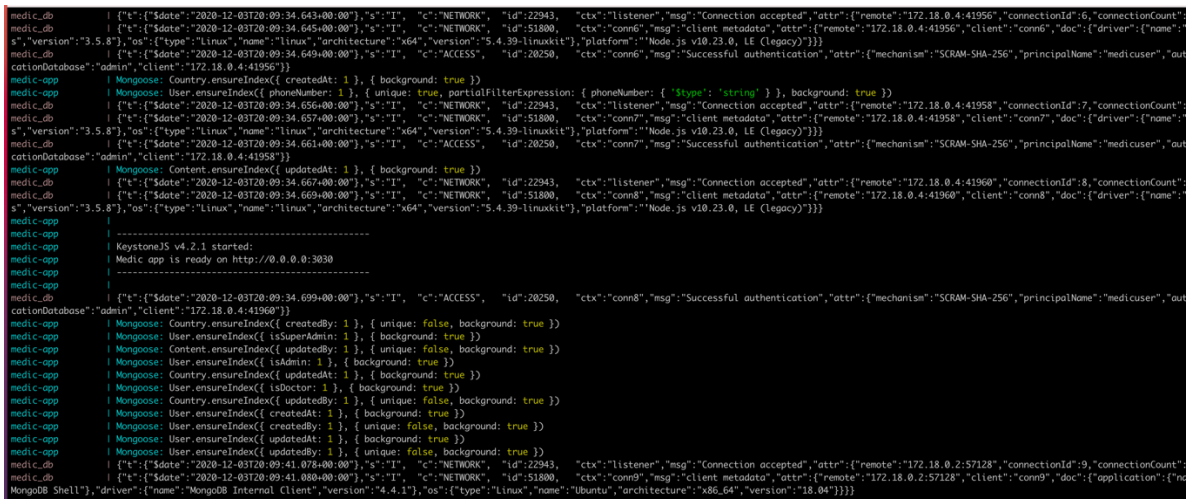

Рисунок 3.2 – Вивід консолі при розгортанні додатка

Для клієнтської та серверної частини за основу беремо образ з node:10-alpine.

Структура Dockerfile клієнтської частини системи:

```
FROM node:10-alpine
RUN apk add --update --no-cache --virtual .gyp g++ make python git

RUN mkdir -p /srv

WORKDIR /srv

COPY package*.json ./

RUN npm install --production
COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

3.3 Проектування додатку

Для налаштування початкового проекту було застосовано Next.js для клієнтської частини та KeystoneJS для серверної частини. Також було встановлено декілька допоміжних бібліотек. Весь файл залежностей та пакетів package.json наведений у Додаток А.

Для початку потрібно реалізувати серверну частину. Програмну реалізацію серверної частини сайту можна розділити на дві частини: розробка API для клієнта та створення структури бази даних сайту. Для розробки API було взято express.js. Express - це легкий фреймворк веб-додатків, який допоможе впорядкувати веб-додаток у архітектуру MVC на стороні сервера.

Зі сторони серверу було створено базу даних MongoDB, та моделі структур даних до неї. Потім використовуємо базу даних, як MongoDB з Mongoose, щоб забезпечити серверну програму для додатку Node.js. Express.js в основному допомагає керувати усім, починаючи від маршрутів, закінчуючи обробкою запитів та переглядів. При розробці структури бази даних було виділено шість сутностей, які потрібні для роботи даного додатку: користувачі, новини, послуги, пацієнти, заявки пацієнтів та налаштування додатку. При створенні самої бази даних, дані сутності було перенесено до наступних колекцій: user, news, appConfig, servise, pacientRequest, pacient.

Приклад створення колекції користувача наведений у Додаток Б

При розробці структури сервера, всі файли було розподілено до різних директорії, відповідно до їх функціоналу. Структура проекту зображена на рисунку 3.3.

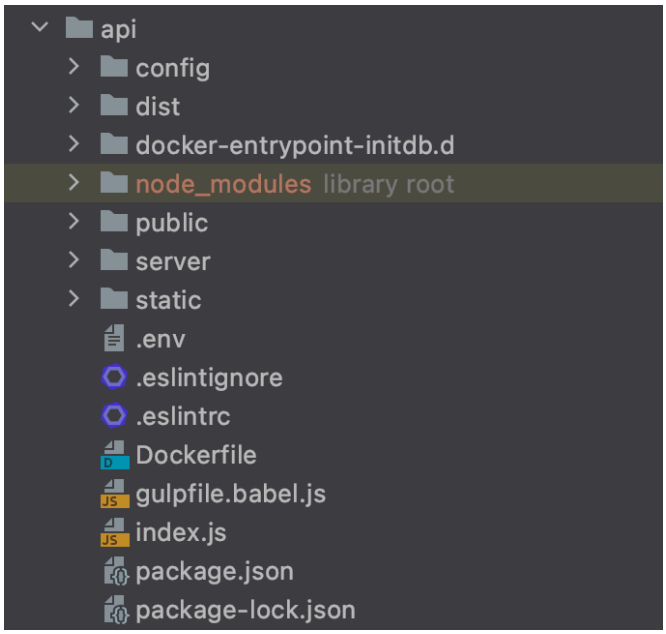

Рисунок 3.3 – Структура back-end

Як ми визначили в другому розділі, доцільно використати бібліотеку React та Redux для створення клієнтської частини. Для створення react додатку буде використано фреймворк Next.js Next.js це популярний й легкий фреймворк для статичних та серверних додатків, створений з допомогою React. Він включає готові рішення для стилізації та маршрутизації й передбачає, що ви використовуєте Node.js як серверний осередок. Next.js використовує компонент App для ініціалізації сторінок. Щоб замінити сторінку за замовчуванням, було створенно файл ./pages/_app.js Додаток В.

Структура клієнтської частини проекту зображено на рисунку 3.4.

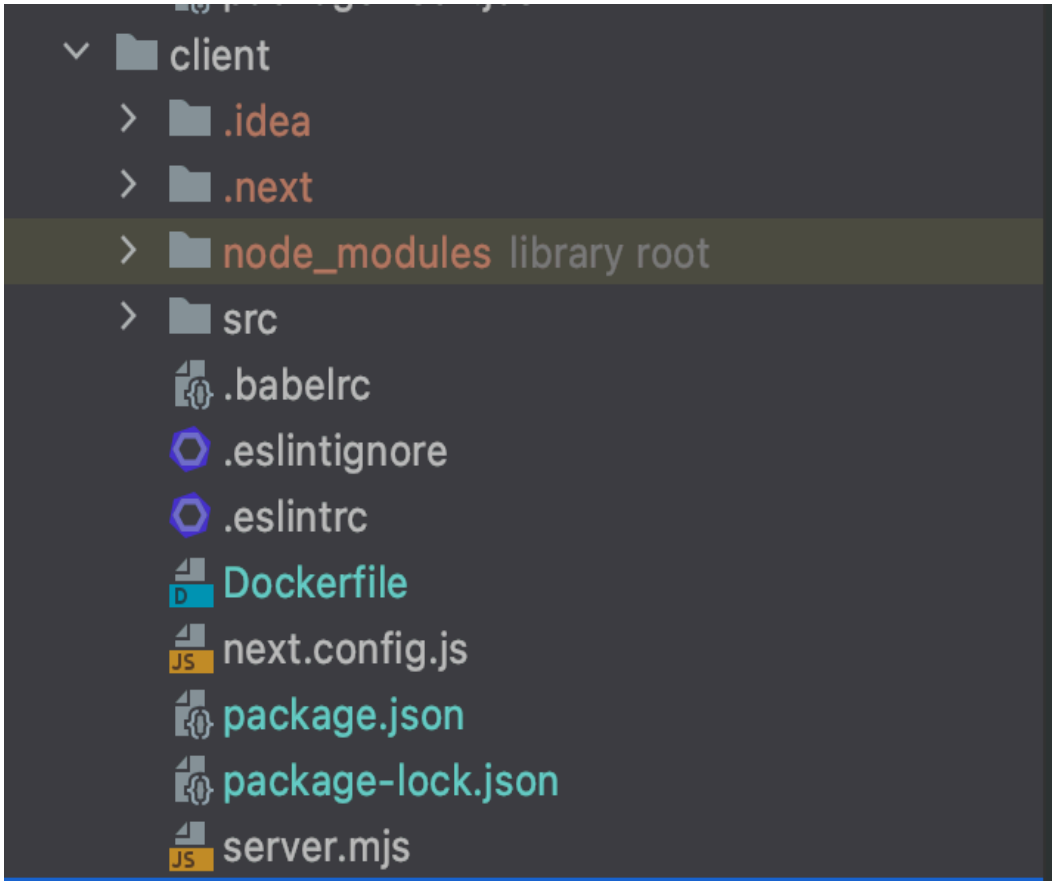

Рисунок 3.4 – Структура front-end

Також для стилізації UI елементів було використано бібліотеку Material-ui. React Material-UI - це набір компонентів React, які реалізують концепцію Material Design від Google[15]. Код частини навігації з використанням Material Design наведений у Додаток Г.

3.4 Інтерфейс веб-додатку

Важливих пунктом при розробці є дизайн веб-додатку. Додаток повинен мати сучасний, зручний інтерфейсу користувача. Також можемо додати такий пункт як адаптивність.

Для того щоб користувачу було більш комфортно переглядати вед-додаток на різних пристроях, треба користуватися про засобами адаптивної верстки. Розробити дизайн сайту - це складне завдання, адже веб-додаток повинен бути не тільки красивим, але і функціональним та залишати приємне враження для всіх відвідувачів. Цьому сприяє правильний вибір кольорової гами, шрифтів, декоративних засобів та грамотне розміщення всіх елементів сторінок.

Для завершення роботи над інформаційною системою були реалізовані всі необхідні сторінки. Нижче наведені зображення функціоналу інформаційної системи (рис. 3.5 – 3.13).

Рисунок 3.5 – Головні сторінка додатку

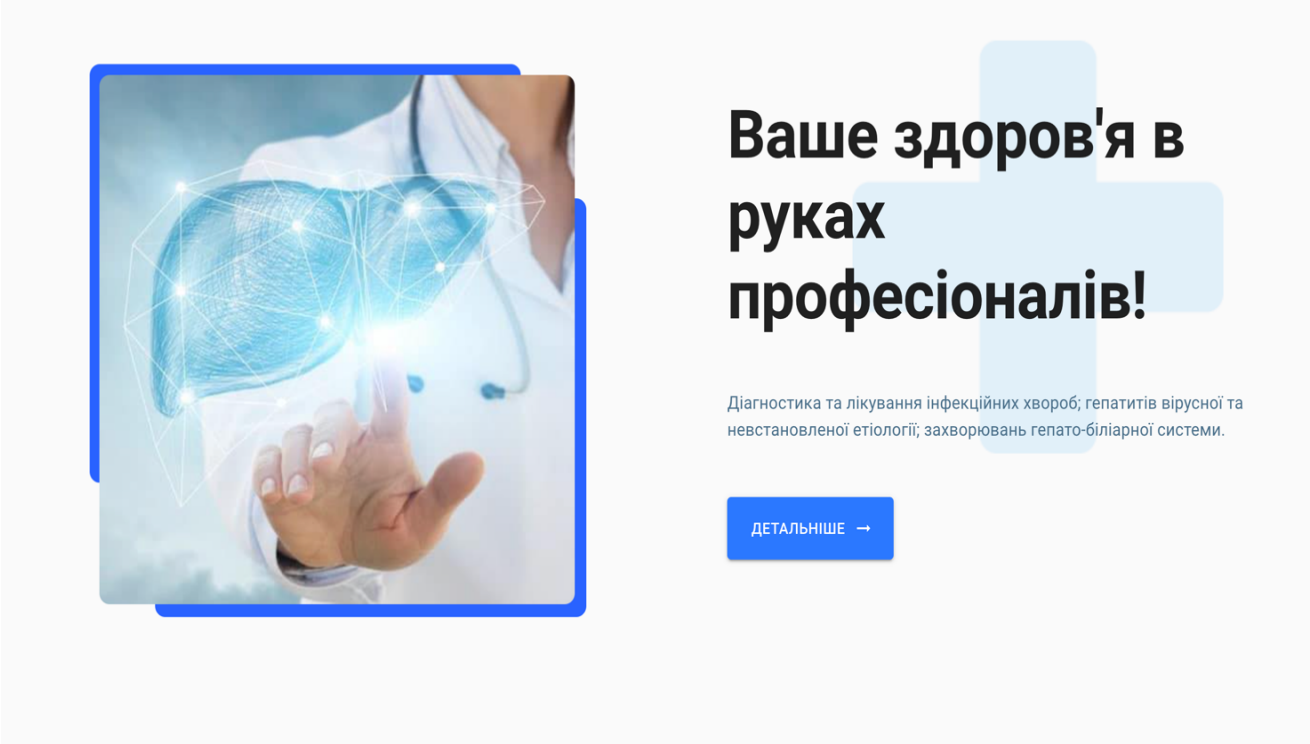

Рисунок 3.6 – Головні сторінка додатку

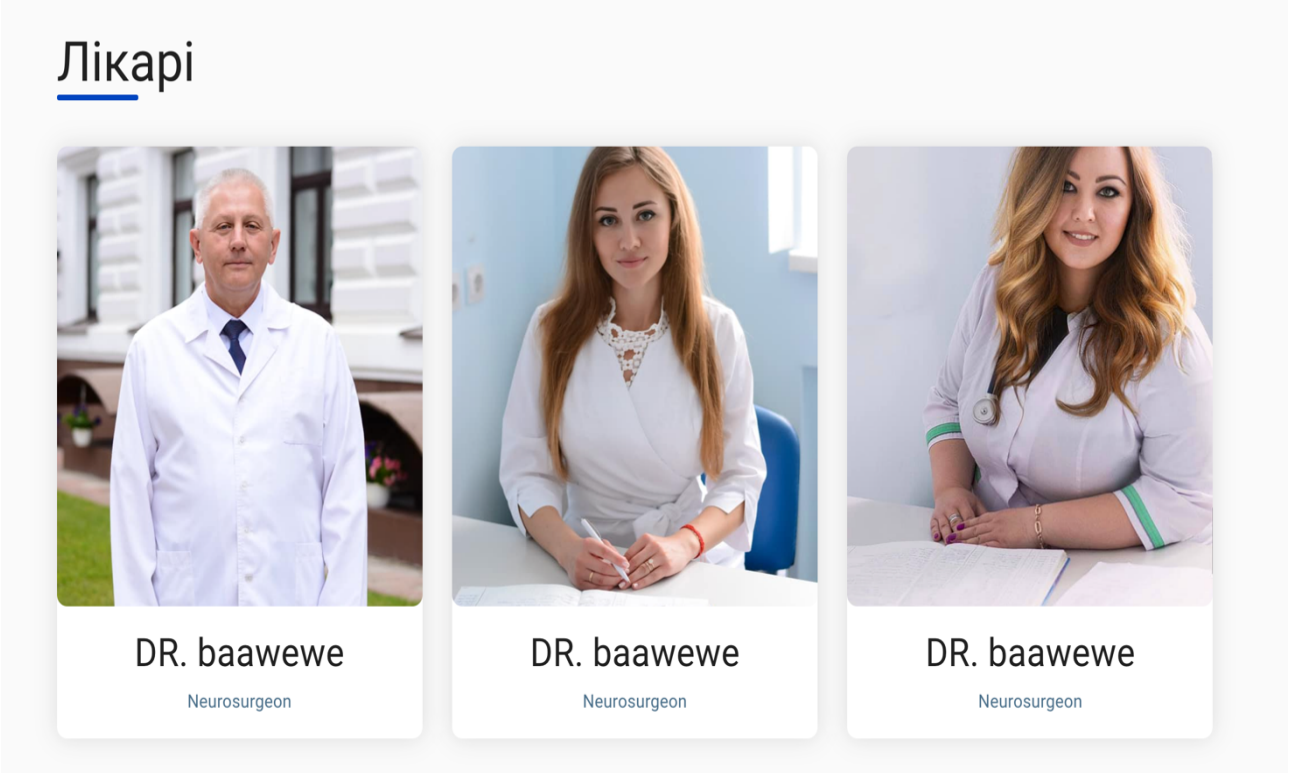

Рисунок 3.7 – Секція для перегляду лікарів на головній сторінці додатку

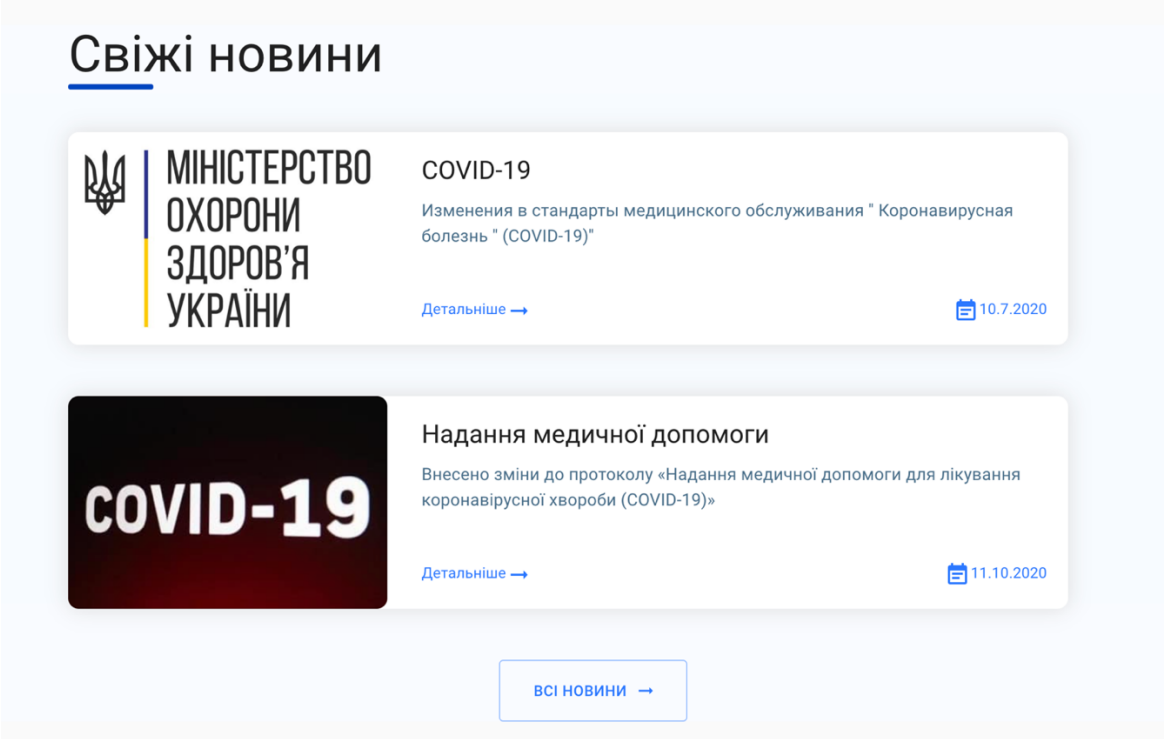

Рисунок 3.8 – Секція для перегляду лікарів на головній сторінці додатку

На рисунку 3.9 представлена панель меню. Дане меню містить посилання, що дозволяють перейти до інших розділів додатку, серед яких є «Про центр», «Лікарі», «Послуги», «Блог» і «Контакти».

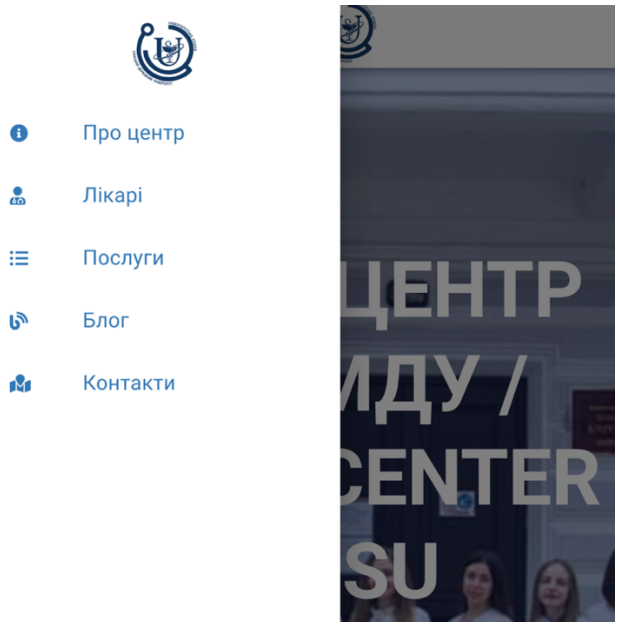

Рисунок 3.9 – Меню

На головній сторінці розміщені всі необхідні секції та посилання. Також на головній сторінці ми можемо записатися на прийом до лікаря. Форма запису на прийом до лікаря зображена на рисунку 3.10.

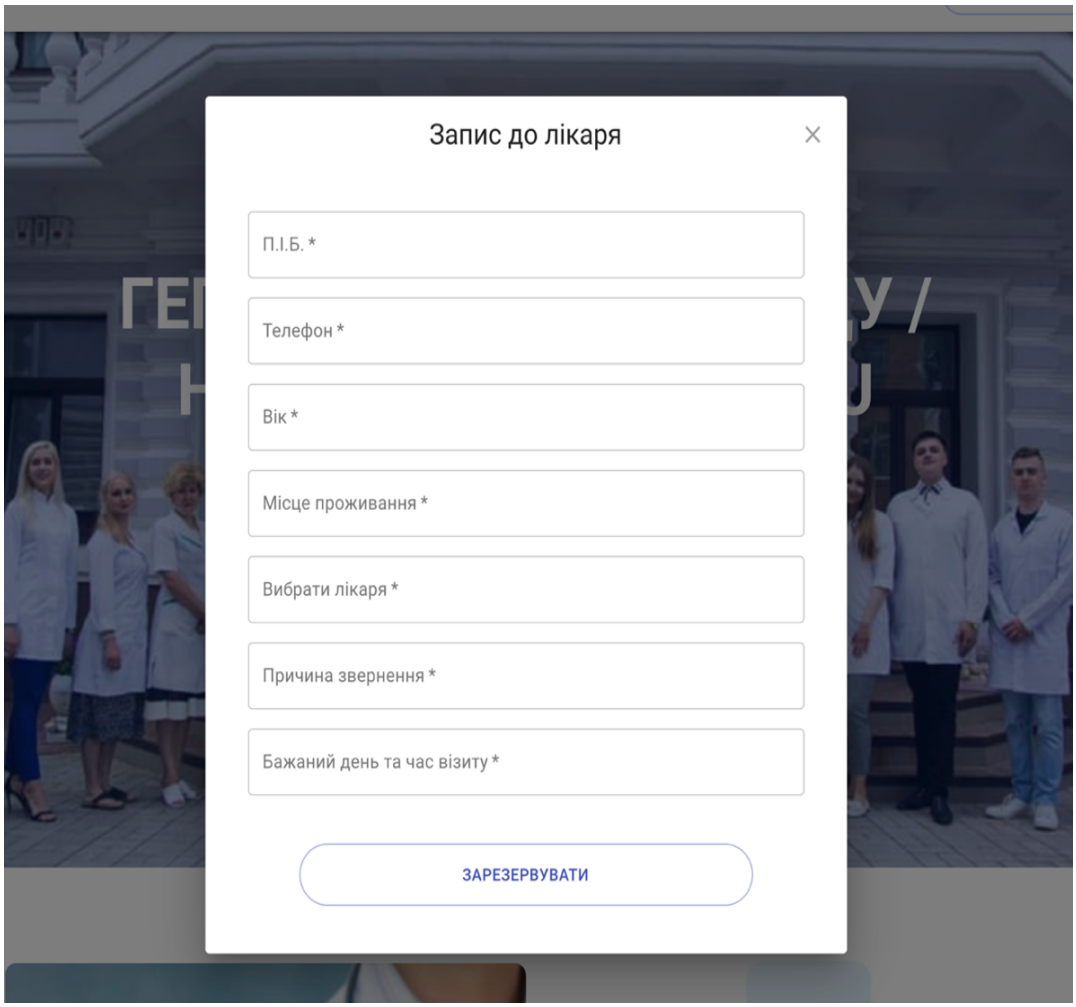

Рисунок 3.10 – Форма запису до лікаря

Будь-який користувач має змогу знавігуватися на розділ «Блог» та переглянути усі актуальні новини центру .

Рисунок 3.11 – Структура проекту

Тепер, переходимо до адмін-панелі. На рисунку 3.12 представлена форма входу в додаток. Дана форма містить два поля введення, які беруть відповідно адресу електронної пошти та пароль користувача. Кнопка «Вхід» запускає процес аутентифікації після введення валідних значень у відповідні поля.

Рисунок 3.12 – Форма авторизації додатку

Після натискання кнопки «Вхід» користувача буде переправлено на сторінку адміністратора, яка зображена на рисунку 3.13. На головній сторінці адміністративної панелі розмішена медична статистика та таблиця з заявками на прийом. Медична статистика має відображати підсумкову картину ефективності лікувально-діагностичного процесу в клініці, свідчити про те, наскільки добре лікують/діагностують у цьому конкретному закладі.

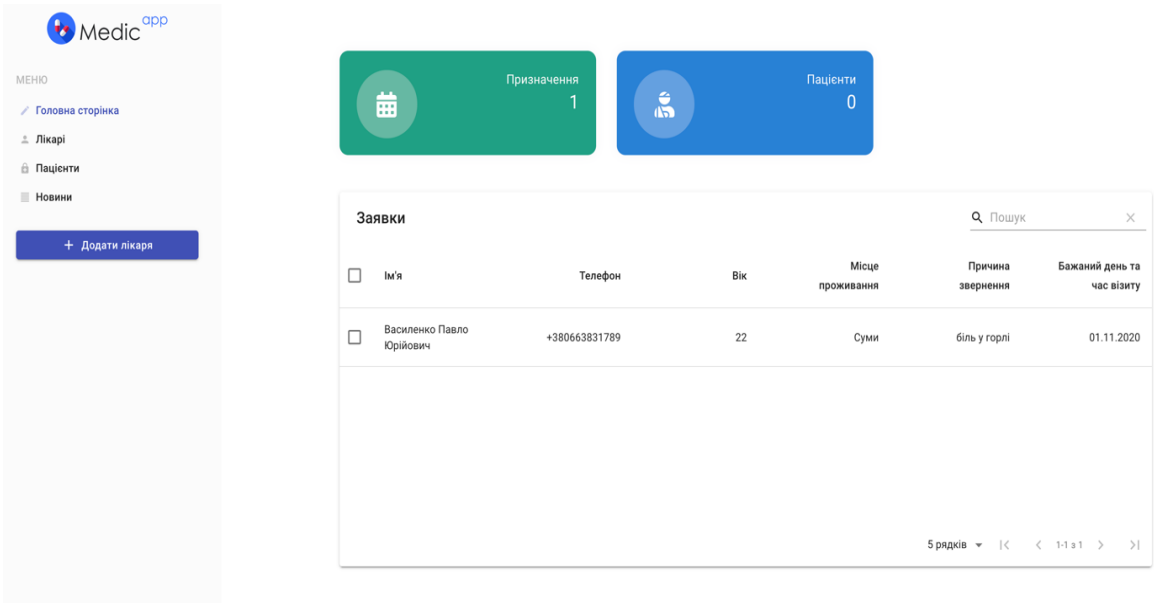

Рисунок 3.13 – Головні сторінка адміністратора

ВИСНОВКИ

В ході виконання роботи було виконано інформаційний огляд медичної галузі. Було проаналізовано існуючі медичні інформаційні системи. Розглянуто досвід розробки існуючих МІС, їх можливості та характеристики. Виявлені дані позитивні можливості МІС, співставленні їх функціональні можливості.

За результатом огляду було визначено, що створення інформаційної системи обліку пацієнтів ГепатоЦентру медичного інституту СумДУ є доцільним.

Було спроектовано модель бізнес-процесів, що дозволяє описатии взаємодії між користувачем та самою системою та відобразити визначені функціональні вимоги до системи.

На стадії розробки використовувався стек сучасних технології та методи для проектування веб-додатків. Завдяки сучасним технологіям, веб-додаток є дуже гнучким та його легко підтримувати. Сервер надає API для взаємодії з клієнтськими додатками.

Створений програмний продукт надає можливість підвищить рівень діяльності ГепатаЦентру медичного інституту СумДУ в зв’язку зі скороченням трудових і часових витрат на процеси, які пов'язані з обліком пацієнтів, за рахунок їх автоматизації. Отже, виявлено, що автоматизація процесів управління центру значно полегшує робочі процеси.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1 Ресурс pidru4niki.com [Електроний ресурс] Режим доступу: https://pidru4niki.com/1222090547713/informatika/informatsiyni_sistemi

2 Інформаційна система [Електронний ресурс] – 2020. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0

3 Типи інформаційних систем в галузі охорони здоров'я [Електронний ресурс]: методичні рекомендації / Львівський національний медичний університет імені Данила Галицького – Л, 2009. – 18 с. –Режим доступу: https://studfiles.net/preview/5280754/

4 Types of Hospital Information Systems. — Режим доступу: https://resources.infosecinstitute.com/category/healthcare-information-security/it-stack/types-of-hospital-information-systems/#gref

5 Офіційний сайт інформаційно-телекомунікаційною системи Helsi.— 2019.— Режим доступу: https://helsi.me/about

6 Fielding J. Beginning Responsive Web Design with HTML5 and CSS3 // Beginning Responsive Web Design with HTML5 and CSS3. 2014.

7 Freeman A. Pro JavaScript for Web Apps // Pro JavaScript for Web Apps. Apress, 2012.

8 Ambler T., Cloud N. JavaScript frameworks for modern web dev // JavaScript Frameworks for Modern Web Dev. Apress Media LLC, 2015. 1–485 p.

9 Gackenheimer C. Introduction to React // Introduction to React. 2015.

10 Pini C. React + Redux: Architecture Overview [Electronic resource] // Medium. 2016. P. 1. URL: https://medium.com/mofed/react-redux-architecture-overview-7b3e52004b6e.

11 Benefits of JSX in React [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/React_(JavaScript_library)#JSX.

12 Руководство для начинающих по Node.js [Електронний ресурс] . – Режим доступу: http://nodeguide.ru/doc/felix/beginner/.

13 Redux [Електронний ресурс].- Режим доступу: https://redux.js.org/, 2020.

14 5 React Architecture Best Practices [Електронний ресурс].- Режим доступу: https://www.sitepoint.com/react-architecture-best-practices/, 2018.

15 Material-UI [Електронний ресурс].- Режим доступу: https://materialui.com/, 2020.

16 Docker Architecture: Why is it important? [Електронний ресурс]. – 2018. – Режим доступу: https://www.edureka.co/blog/docker-architecture/.

17 Загороднюк А.О. “Архитектура Docker” - Молодой исследователь: вызовы и перспективы: сб. ст. по материалам LXIX междунар. науч.-практ. конф. — № 16(69). — М., Изд. «Интернаука», 2018. – 482 с.

18 Use cases diagram [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Use_case_diagram.

ДОДАТКИ

ДОДАТОК А

Лістинг файлу залежностей package.json:

```
{
  "name": "medic-app-front",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "concurrently --kill-others \"npm run start-
prod\"",
    "start-prod": "better-npm-run start-prod",
    "lint": "eslint --quiet --color --fix
'src/**/*.{js,ts,tsx}'",
    "dev": "better-npm-run start-dev",
    "build": "next build",
    "analyze": "ANALYZE=true next build"
  },
  "betterScripts": {
    "start-prod": {
      "command": "node --experimental-modules ./server.mjs",
      "env": {
        "NODE_PATH": "./src",
        "NODE_ENV": "production",
        "PORT": 8080,
        "APIPORT": 3030
      }
    },
    "start-dev": {
      "command": "node --experimental-modules ./server.mjs",
      "env": {
        "NODE_PATH": "./src",
        "NODE_ENV": "development",
        "PORT": 3000,
        "APIPORT": 3030
      }
    }
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@babel/plugin-proposal-decorators": "^7.8.3",
    "@fortawesome/fontawesome-svg-core": "^1.2.32",
    "@fortawesome/free-solid-svg-icons": "^5.15.1",
    "@fortawesome/react-fontawesome": "^0.1.13",
    "@material-ui/core": "^4.11.0",
    "@material-ui/icons": "^4.9.1",
    "@material-ui/lab": "^4.0.0-alpha.56",
    "@material-ui/styles": "^4.10.0",
```

```
    "@next/bundle-analyzer": "^9.2.2",
    "@zeit/next-sass": "^1.0.1",
    "classnames": "^2.2.6",
    "compression": "^1.7.4",
    "cookie-parser": "^1.4.4",
    "dot-object": "^2.1.3",
    "dotenv": "^8.2.0",
    "express": "^4.17.1",
    "formik": "^2.2.5",
    "http-proxy": "^1.18.0",
    "js-cookie": "^2.2.1",
    "lodash": "^4.17.15",
    "material-table": "^1.69.2",
    "moment": "^2.24.0",
    "next": "^9.2.1",
    "next-cookie": "^2.1.3",
    "next-images": "^1.3.1",
    "node-sass": "^5.0.0",
    "pretty-error": "^2.1.1",
    "prop-types": "^15.7.2",
    "react": "^16.12.0",
    "react-dom": "^16.12.0",
    "react-parallax": "^3.1.2",
    "react-redux": "^7.1.3",
    "react-router-redux": "^4.0.8",
    "react-select": "^3.0.8",
    "redux": "^4.0.5",
    "redux-connect": "^10.0.0",
    "redux-form": "^8.3.0",
    "redux-thunk": "^2.3.0",
    "serve-favicon": "^2.5.0",
    "simplebar-react": "^2.1.0",
    "superagent": "^5.2.1",
    "svg-country-flags": "^1.2.7",
    "uniqid": "^5.2.0"
  },
  "devDependencies": {
    "babel-eslint": "^8.2.5",
    "eslint": "^6.8.0",
    "eslint-config-airbnb": "^16.0.0",
    "eslint-loader": "^2.0.0",
    "eslint-plugin-cypress": "^2.7.0",
    "eslint-plugin-import": "^2.13.0",
    "eslint-plugin-jsx-a11y": "^6.0.3",
    "eslint-plugin-react": "^7.10.0",
    "@babel/core": "^7.8.4",
    "babel-loader": "^8.0.6",
    "better-npm-run": "^0.1.1",
    "lodash-webpack-plugin": "^0.11.5",
    "webpack-bundle-analyzer": "^3.6.0"
  }
}
```

ДОДАТОК Б

Лістинг файлу навігації User.js:

```
import keystone from 'keystone';
import uniqueValidator from 'mongoose-unique-validator';

// validator
import { emailValidator, phoneNumberValidator } from
'../helpers/validators';

// helpers
import removeFile from '../helpers/removeFile';

const config = require('../../config/env');

const { Types } = keystone.Field;

/**
 * User Model
 * ==========
 */
const User = new keystone.List('User', {
  track: true,
  defaultSort: '-createdAt',
});

const imgStorage = new keystone.Storage({
  adapter: keystone.Storage.Adapters.FS,
  fs: {
    path: './public/uploads/users',
    publicPath: '/uploads/users/',
  },
  schema: {
    url: true
  },
});

User.add({
  name: { type: Types.Name, required: true, index: true },
  email: {
    type: Types.Email,
    lowercase: true,
    required: true,
    initial: true,
    trim: true,
    index: {
      unique: true,
      partialFilterExpression: { email: { $type: 'string' }
},
    },
    note: 'Email where would be send company admin emails',
  },
```

```
      phoneNumber: {
        type: String,
        initial: true,
        uniqueCaseInsensitive: true,
        trim: true,
        index: {
          unique: true,
          partialFilterExpression: { phoneNumber: { $type:
'string' } },
        },
        note: '1-15 digit numbers',
      },
      photo: {
        type: Types.File,
        storage: imgStorage
      },
      employmentYear: {
        type: String
      },
      doctorSpecialization: {
        type: String
      },
      description: { type: Types.Textarea },
      achievements: { type: Types.Textarea },
      active: { type: Boolean, default: false },
      password: {
        type: Types.Password,
        initial: true,
        workFactor: config.env === 'production' ? 10 : 4,
      },
      address: {
        street: { type: String },
        zipCode: { type: String },
        city: { type: String },
        country: { type: String },
      },
    }, 'Permissions', {
      isSuperAdmin: { type: Boolean, label: 'Can access Keystone
(Global Admin)', index: true },
      isAdmin: { type: Boolean, label: 'Can access to admin
panel', index: true },
      isDoctor: { type: Boolean, label: 'Can create patients',
index: true },
    });

    // Provide access to Keystone
    User.schema.virtual('canAccessKeystone').get(function () {
      return this.isSuperAdmin;
    });

    User.schema.post('init', function () {
      this._oldImg = this.toObject().photo;
    });
```

```
    User.schema.pre('save', function (next) {
      if    (this.isModified('photo')    &&    this._oldImg    &&
this._oldImg.url) {
        removeFile(this._oldImg.url, next);
      } else {
        next();
      }
    });
    User.schema.post('remove', (item) => {
      if          (item.photo          &&          item.photo.url)
removeFile(item.photo.url, console.error);
    });
    User.schema.pre('validate', function (next) {
      if (this.isModified('phoneNumber') && this.phoneNumber ===
'') {
        this.phoneNumber = null;
      }
      next();
    });
    User.schema.pre('validate', function (next) {
      try {
        if (this.isModified('phoneNumber') && this.phoneNumber) {
          this.phoneNumber                                        =
this.phoneNumber.trim().replace(/\D/g, '');
        }
        next();
      } catch (e) {
        return next(e);
      }
    });
    // Validate email and phone number
    User.schema.pre('save', async function (next) {
      try {
        await emailValidator(this, next);
        await phoneNumberValidator(this, next);
        next();
      } catch (e) {
        return next(e);
      }
    });

    User.schema.plugin(uniqueValidator, { message: 'This {PATH}
has already been taken' });

    /**
     * Registration
     */
    User.defaultColumns = 'name email isAdmin phoneNumber
isSuperAdmin isDoctor createdAt';
    User.register();

    export default User;
```

ДОДАТОК В

Лістинг файлу _app.js:

```
import React from 'react';
import App from 'next/app';
import Helmet from 'react-helmet';
import withRedux from '../redux/withRedux';
import { Provider } from 'react-redux';
import { MuiThemeProvider } from '@material-ui/core/styles';
import CssBaseline from '@material-ui/core/CssBaseline';
import { setContent } from '../redux/modules/appCurrentState';
import ReactContext from '../helpers/ReactContext';
import moment from 'moment';
import { withRouter } from 'next/router';
import { handleLocaleChange } from '../redux/modules/momentLocale';
import { setCurrentUser } from '../redux/modules/auth';
import Scroll from 'react-scroll';
import DefaultLayout from '../components/DefaultLayout';
import { theme } from '../theme/mu-theme';

import '../public/styles.scss';

const appConfig = {
  htmlAttributes: { lang: 'ua', amp: undefined },
  titleTemplate: '%s | Medic',
  defaultTitle: 'Medic'
};

class _App extends App {
  static async getInitialProps({ Component, ctx }) {
    const { reduxStore, client } = ctx;
    const store = reduxStore.getState();
    let content = {};
    try {
      // load content on server-side
      if (!store.content) {
        content = await client.get('/v1/contents', { params: { lang: 'en' } });
        reduxStore.dispatch(setContent(content.resource));
      }
    } catch (e) {
      console.log(e);
    }

    return {
      pageProps: {
        // Call page-level getInitialProps
        ...(Component.getInitialProps ? await Component.getInitialProps(ctx) : {})
      },
```

```
          content: store.appCurrentState.content || content,
          language: store.appCurrentState.language
        };
      }
      constructor(props) {
        super(props);
    props.reduxStore.dispatch(handleLocaleChange(props.content.na
meShort));
        this.handleMomentLocale(props.content.nameShort);
      }
      // Remove the server-side injected CSS.
      componentDidMount() {
        const { reduxStore } = this.props;
        const store = reduxStore.getState();
        // authorize user on client if not present
        if (!store.user) this.preloadUser(reduxStore);
        const  jssStyles  =  document.getElementById('jss-server-
side');
        if (jssStyles && jssStyles.parentNode) {
          jssStyles.parentNode.removeChild(jssStyles);
        }
      }

      componentDidUpdate(prevProps) {
        if              (prevProps.router.pathname              !==
this.props.router.pathname) {
          return    Scroll.animateScroll.scrollToTop({    smooth:
false, duration: 0 });
        }

        return false;
      }

      preloadUser = async (reduxStore) => {
        try {
          const              res              =              await
reduxStore.client.get('v1/authentication');
          // set current user if present
          reduxStore.dispatch(setCurrentUser(res));
        } catch (e) {
          console.error(e);
        }
      };

      handleMomentLocale = locale => moment.locale(locale);

      render() {
        const {
          Component,
          pageProps,
          reduxStore
        } = this.props;
```

```
        const ComponentLayout = Component.Layout ||
DefaultLayout;
        return (
          <>
            <Helmet {...appConfig} />
            <MuiThemeProvider theme={theme}>
              <CssBaseline/>
              <ReactContext.Provider
                value={{
                  moment,
                  client: reduxStore.client,
                  store: reduxStore,
                  handleMomentLocale: this.handleMomentLocale
                }}
              >
                <Provider store={reduxStore}>
                  <div id="app" className="app">
                    <ComponentLayout {...pageProps}>
                      <Component {...pageProps} />
                    </ComponentLayout>
                  </div>
                </Provider>
              </ReactContext.Provider>
            </MuiThemeProvider>
          </>
        );
      }
    }

    export default (withRedux(withRouter(_App)));
```

ДОДАТОК Г

Лістинг файлу Header.js:

```
// base
import React from 'react';
import PropTypes from 'prop-types';
// material-ui
import AppBar from '@material-ui/core/AppBar';
import Toolbar from '@material-ui/core/Toolbar';
import IconButton from '@material-ui/core/IconButton';
import Hidden from '@material-ui/core/Hidden';
import { withStyles } from '@material-ui/core/styles';
import MenuIcon from '@material-ui/icons/Menu';
import List from '@material-ui/core/List';
import ListItem from '@material-ui/core/ListItem';
import ListItemText from '@material-ui/core/ListItemText';
import NoSSR from '@material-ui/core/NoSsr';
import { styles } from './styles';

// components
import NavItem from '../NavItem';
import ContactUsForm from '../ContactUsForm';
import Drawer from '../Drawer';
import Link from '../../components/Link';
import Button from '@material-ui/core/Button';

import { FontAwesomeIcon } from '@fortawesome/react-
fontawesome'
import { faUserMd, faInfoCircle, faListUl, faBlog, faMapMarked
} from '@fortawesome/free-solid-svg-icons'
const logo = require('../../public/logo.svg');
@withStyles(styles)
export default class Header extends React.PureComponent {
    static propTypes = {
      classes: PropTypes.object.isRequired,
```

```
        }

        constructor(props) {
          super(props);
          this.routes = [
            {
              name: 'about',
              component: Link,
              href: '/about',
              label: 'Про центр',
              icon: <FontAwesomeIcon icon={faInfoCircle} />
            },
            {
              name: 'doctors',
              component: Link,
              href: '/doctors',
              label: 'Лікарі',
              icon: <FontAwesomeIcon icon={faUserMd} />
            },
          ];

          this.state = {
            contactUs: false,
            open: false,
                  isContactFormOpen: false
          };
        }
        onClickAway = (item) => this.setState({ [item]: false })
        windowWidth = () => {
          const w = window;
          const d = document;
          const documentElement = d.documentElement;
          const body = d.getElementsByTagName('body')[0];
          return w.innerWidth || documentElement.clientWidth ||
body.clientWidth;
```

```
}

handleLinkClick = (e, item) => {
  const width = this.windowWidth();
  if (width <= 960) this.setState({ open: false });
  // if click on child item, close parent collapse
  if (item) this.setState({ [item]: false });
}
handleToggleContactForm = () => {
     this.setState(state => ({
          isContactFormOpen: !state.isContactFormOpen
     }));
}
handleToggleDrawer = () => {
  this.setState(state => ({
    open: !state.open
  }));
}
render() {
  const { classes } = this.props;
  const { open, isContactFormOpen } = this.state;
    const mobile = (
      <>
        <IconButton
        color="inherit"
        aria-label="open drawer"
        onClick={this.handleToggleDrawer}
        className={classes.menuButton}
        >
          <MenuIcon/>
        </IconButton>
        <Drawer
        variant="temporary"
        anchor="left"
        open={open}
```

```
                onClose={this.handleToggleDrawer}
                classes={{
                                paper: classes.drawerPaper
                            }}
                mobileBackdropId="backdrop"
                ModalProps={{
                                keepMounted: true, // Better
open performance on mobile.
                            }}
                >
                  <List>
                    <ListItem>
                      <ListItemText
                      className={classes.menuLogoInverse}
                      primary={<img src={logo} alt=""
className={classes.logoInverse}/>}
                      />
                    </ListItem>
                    <Hidden mdUp className={classes.content}>
                      {this.routes.map((item, index) => {
                        const config = {
                          className: classes.navLink
                        };
                        if (item.component === Link) {
                          config.activeClassName =
classes.activeLink;
                          config.onlyActiveOnIndex = true;
                        }
                        return (
                          <NavItem key={index} {...item}
{...config}/>
                        );
                      })}
                    </Hidden>
                  </List>
```

```
                </Drawer>
              </>
            );

          return (
            <AppBar position="fixed" color="default">
              <Toolbar>
                 <Hidden mdUp className={classes.content}>
                  {mobile}
                 </Hidden>
                <NoSSR>
                  <div className={classes.logoInverseContainer}>
                    <Link href="/">
                      <img src={logo} alt="logo"
className={classes.logoInverse}/>
                    </Link>
                  </div>
                </NoSSR>
                <Hidden smDown className={classes.content}>
                  {this.routes.map((item, index) => <NavItem
key={index} variant="button" {...item}/>)}
                  <Button variant="outlined" color="secondary"
onClick={this.handleToggleContactForm}
className={classes.bannerBtn}>
                                          Запис на прийом
                  </Button>
                </Hidden>
              </Toolbar>
                      <ContactUsForm open={isContactFormOpen}
handleToggleContactForm={this.handleToggleContactForm}/>
            </AppBar>
          );
        }
    }
```