НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Програмне забезпечення інформаційно-комунікаційних систем»
спеціальності 121 «Інженерія програмного забезпечення»
на тему: «Клієнт-серверний застосунок обліку пацієнтів у лікарні»

Виконав (-ла):
студент (-ка) IV курсу, групи ІТ-61
Мусатов Дмитро Станіславович _____

Керівник:
Доц. каф. АУТС, к.т.н
Савчук Олена Володимирівна _____

Рецензент:
Доц. каф. ТК, к.т.н.
Корнага Ярослав Ігорович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Програмне забезпечення інформаційно-комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Мусатову Дмитру Станіславовичу

1. Тема проєкту «Клієнт-серверний застосунок обліку пацієнтів у лікарні», керівник проєкту доцент Савчук Олена Володимирівна, затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проєкту 09.06.2020

3. Вихідні дані до проєкту

Програмне забезпечення на клієнт-серверній архітектурі. Мова програмування JavaScript, середовище розробки Visual Studio Code, обрані технології – Node.js, Express.js, бібліотеки React.js та Redux, СКБД – PostgreSQL.

4. Зміст пояснювальної записки

1. Вступ 2.Опис предметної області 3. Огляд існуючих рішень 4. Варіанти використання системи 5. Аналіз вимог до програмного забезпечення 6. Вибір технологій розробки 7. Реалізація програмного забезпечення 8. Впровадження та використання розробленої системи

5. Перелік графічного матеріалу

Діаграма використання, схема серверної частини, ER-діаграма бази даних, діаграма послідовності, діаграма компонентів

6. Дата видачі завдання 6.03.2020

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вибір тематичного напрямку та узгодження теми дипломного проєкту	6.03.2020	
2	Аналіз теоретичних матеріалів та вивчення предметної області	10.04.2020	
3	Огляд існуючих рішень з тематики роботи	15.04.2020	
4	Розробка структури прототипу та проектування системи	21.04.2020	
5	Розробка технічного завдання, вибір методів та засобів реалізації задачі	24.04.2020	
6	Реалізація проєкту	15.05.2020	
7	Налагодження та перевірка програми	17.05.2020	
8	Оформлення пояснювальної записки	23.05.2020	
9	Передзахист дипломного проєкту	25.05.2020	
10	Доопрацювання пояснювальної записки та підготовка презентації	07.06.2020	
11	Подання ДП на основний захист	9.06.2020	

Студент Дмитро МУСАТОВ

Керівник проєкту Олена САВЧУК

АНОТАЦІЯ

Мусатов Д.С. Клієнт-серверний застосунок для обліку пацієнтів у лікарні. КПІ ім. Ігоря Сікорського, Київ, 2020.

Ключові слова: клієнт-серверний застосунок, медичне програмне забезпечення, діаграма використання, діаграма послідовності, JavaScript, Node.js, React.js, Redux.

Основна частина документу викладена у пояснювальній записці, що містить 67 сторінок, 29 таблиць та 4 рисунки. До змісту входить 5 креслеників.

У дипломному проєкті проведено огляд існуючих рішень, їх детальний аналіз та порівняння. Виконано огляд технологій, на базі яких можливе створення клієнт-серверного застосунку для обліку пацієнтів у лікарні. Було створено застосунок на мові програмування JavaScript з використанням технологій Node.js, Express.js, бібліотек React.js та Redux, СКБД — PostgreSQL.

Готовий застосунок відповідає поставленим до нього при розробці вимогам та містить увесь необхідний функціонал.

SUMMARY

Musatov D.S. “Hospital’s patient client-server management application”. Igor Sikorsky KPI, Kyiv, 2020.

Keywords: client-server application, medical software, use case diagram, sequence diagram, JavaScript, Node.js, React.js, Redux.

The bulk of the document is presented in an explanatory note, which contains 67 pages and 29 tables. In its content are included 5 drawings.

The diploma project reviews the existing solutions, their detailed analysis and comparison. An overview of technologies was done, based on which it is possible to create a hospital’s patient client-server management application. An application was created on JavaScript programming language using technologies Node.js, Express.js and libraries React.js, Redux, DMS — PostgreSQL.

Completed application meets the requirements, which were set for it during development, and contains all the necessary functions.

Номер рядка	Формат	Позначення	Найменування	Кіл. листів	№ екз.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІТ61.13.0БАК.004 ПЗ	Клієнт-серверний застосунок	67		
6			обліку пацієнтів у лікарні.			
7			Пояснювальна записка			
8	А3	ІТ61.13.0БАК.004 Д1	Клієнт-серверний застосунок	1		
9			обліку пацієнтів у лікарні.			
10			Діаграма використання			
11	А3	ІТ61.13.0БАК.004 Д2	Клієнт-серверний застосунок	1		
12			обліку пацієнтів у лікарні.			
13			Схема серверної частини			
14	А3	ІТ61.13.0БАК.004 Д3	Клієнт-серверний застосунок	1		
15			обліку пацієнтів у лікарні.			
16			ER-діаграма бази даних			
17	А3	ІТ61.13.0БАК.004 Д4	Клієнт-серверний застосунок	1		
18			обліку пацієнтів у лікарні.			
19			Діаграма компонентів			
20	А3	ІТ61.13.0БАК.004 Д5	Клієнт-серверний застосунок	1		
21			обліку пацієнтів у лікарні.			
22			Діаграма послідовності			
23						
24						

Змін.	Арк	№ докум.	Підпис	Дата	ІТ61.130БАК.004 ТП		
Розроб.		Мусатов Д. С.			Клієнт-серверний застосунок для обліку пацієнтів у лікарні. Відомість технічного проєкту	Літ.	Арк 1 / Аркушів 1
Перевір.		Савчук О. В.					
Реценз						КПІ ім. Ігоря Сікорського ФІОТ гр. ІТ-61	
Н. Контр.							
Затверд..							

Пояснювальна записка

до дипломного проєкту

на тему: «Клієнт-серверний застосунок для обліку пацієнтів у лікарні»

Київ – 2020 рік

ЗМІСТ

ВСТУП

Україна сьогодення стрімко рухається в напрямку цифрової трансформації. Все більше і більше державних та приватних установ нашої країни переходять на системи електронного документообігу (ЕДО). Але, на жаль, через відсутність достатніх технічних потужностей не всі сфери діяльності мають можливості впровадити такі важливі і необхідні технології. Однією з таких галузей є система охорони здоров'я.

Спираючись на аналіз забезпеченості галузі охорони здоров'я лікарями, який було проведено ДУ «Український інститут стратегічних досліджень МОЗ України», забезпеченість населення лікарями, які займаються безпосередньо лікувальною діяльністю складає 26,8 на 10 тис. населення [1]. Згідно з цими даними ми можемо дійти висновку, що кількість медичних кадрів є недостатньою для гідного забезпечення потреб населення України у медичному обслуговуванні. Звичайно, на такі результати впливає безліч різноманітних чинників, проте одним з них є недостатнє технічне забезпечення лікарень, що значно ускладнює роботу кожного окремого лікаря і збільшує його робоче навантаження.

Крім того, існує велика кількість проблем, з якими стикаються і пацієнти. Певним чином це пов'язане з тим, що для отримання власних медичних даних пацієнти змушені щоразу повторно звертатись до лікарень, що веде до виникнення нескінчених черг, які неабияк гальмують процес отримання медичної допомоги.

У наш час клієнт-серверні застосунки набувають масового поширення. Вони використовуються майже у всіх галузях з метою спрощення та оптимізації роботи. На жаль, їх розробка наразі є досить високовартісною і потребує немалої кількості розробників, проте готовий продукт повністю виправдовує витрачені на нього сили та кошти.

Саме тому завданням цього дипломного проєкту стало створення клієнт-серверного застосунку для обліку пацієнтів у лікарні, який допоможе вирішити хоча б частину з наведених вище проблем. Цей застосунок дозволяє лікарям

контролювати хід лікування пацієнта від першого звернення у лікарню до останнього повторного візиту, призначати необхідні діагностичні та лікувальні маніпуляції та складати простий і зрозумілий список призначень. В свою чергу пацієнт має змогу дистанційно записатись на прийом до потрібного йому лікаря, що дозволить оминути постійні черги, а також отримати доступ до всіх своїх медичних даних не виходячи з дому. Крім того, цей застосунок привносить в життя пацієнтів таку зручну річ, як існування наочного списку призначених лікарем фармацевтичних засобів, що є дуже важливим, особливо для людей похилого віку та тих, хто часто не розуміє написані лікарем від руки призначення.

Саме такий застосунок має шанси значно спростити комунікацію між лікарем та пацієнтом і стати ще одним кроком у переході галузі охорони здоров'я до електронного документообігу.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Обґрунтування актуальності розробки

Зазвичай клієнт-серверні застосунки реалізують за допомогою мов програмування Java, C# та PHP, але в останнє десятиліття великого розповсюдження для їх створення набула мова програмування JavaScript[2].

У використанні саме JavaScript дійсно можна виділити безліч переваг, першою з яких, звичайно, є швидкість. Така характеристика і на базовому рівні є визначальною для цієї мови. Крім того, створено велику кількість технологій та бібліотек, метою яких є не лише примножити швидкість роботи готового продукту, написаного мовою JavaScript, а й збільшити функціонал готового застосунку.

Також слід зазначити, що використання цієї мови та споріднених технологій і засобів усуває проблеми з безпекою, що іноді виникають при використанні інших базових мов. Крім того, JavaScript не є надто складним в освоєнні та відрізняється достатньо високою швидкістю розробки.

Окремо слід виділити те, що в останні роки спостерігалось значне зростання спільноти навколо цієї мови та стрімке збільшення кількості розробників, що нею володіють. Такий ріст спільноти значно полегшує виправлення помилок, що можуть виникати при розробці власних застосунків, а також збільшує ймовірність швидкого вирішення проблемних питань, що мають змогу виникати.

1.1.1 Опис та аналіз предметного середовища

Застосунок для обліку пацієнтів у лікарні – це програмне забезпечення, яке слугує для оптимізації та підвищення ефективності роботи медичних закладів.

Зазвичай, застосунки для обліку пацієнтів у лікарні створюються з використанням клієнт-серверної архітектури.

Клієнт-серверна архітектура – це мережева архітектура, у якій обчислювальне навантаження розподіляється між постачальниками, які

називаються серверами, та замовниками послуг, що називаються клієнтами. Фактично сервер та клієнт – це програмне забезпечення[3].

Одним з випадків клієнт-серверної архітектури є трирівнева архітектура, яка складається з трьох компонентів, або рівнів:

– рівень клієнта – інтерфейсний рівень, який надається кінцевому користувачу;

– рівень сервера – цей рівень координує програму, зв'язуючий рівень між клієнтом та даними;

– рівень бази даних – шар, де зберігається уся інформація, яка відправляється у серверний рівень для обробки та повертається користувачеві.

Основними користувачами клієнт-серверного застосунку є лікарі та пацієнти, тому цей застосунок варто розглянути зі сторони кожного з користувачів.

Однією з основних переваг для пацієнтів при користуванні таким застосунком є можливість дистанційно отримати доступ до свої медичних даних та рекомендацій лікаря. Також пацієнти мають змогу записатись на прийом, уникнувши черг.

Лікареві такі системи також надають багато переваг. Наприклад, можливість дистанційно заповнювати медичну картку пацієнта або надавати рекомендації щодо лікування.

1.1.2 Аналіз функціональних можливостей системи

Під функціональними можливостями системи слід розуміти наступний функціонал:

– механізм автентифікації та реєстрації користувачів;
– реалізація різних ролей для надання різного рівня доступу до сайту;
– навігаційне меню;
– головна сторінка;
– різні сторінки профілю для кожної з ролей;
– відповідні можливості взаємодії з даними системи для різних ролей.

1.2 Призначення розробки

Розроблений продукт є специфічним по відношенню до галузі, але це не робить його менш універсальним. У медичній сфері він може використовуватись для великої кількості різноманітних задач: від організації роботи невеликої приватної клініки до збільшення ефективності роботи крупних медичних закладів. Крім того, слід мати на увазі, що даний продукт може бути надзвичайно корисним для корекції роботи не лише міських, а й сільських медичних установ, що наразі є досить актуальною проблемою.

1.3 Задачі розробки

Головною задачею розробки можна вважати оптимізацію системи обліку пацієнтів у лікарні та надання можливості переведення усієї паперової документації в єдину електронну систему.

Серед додаткових задач можна виділити покращення роботи вже існуючих механізмів оптимізації роботи лікарні та надання альтернативних можливостей для вирішення питань обліку пацієнтів.

1.4 Цілі розробки

Цілі розробки даного застосунку можна умовно розділити на три категорії: для медичних працівників, для пацієнтів та для власне розробників.

Головними цілями створення цього застосунку для медичних працівників можна вважати полегшення їх роботи, поліпшення її якості та значне прискорення ефективного обслуговування пацієнтів. Крім того, додатковою метою є загальна оптимізація функціонування медичної сфери з боку її працівників та підвищення ступеню безпеки медичних даних пацієнтів.

Головними цілями створення цього застосунку для пацієнтів безумовно є зростання ефективності медичного обслуговування та усунення недоліків існуючої медичної системи. Також слід зазначити, що збільшення швидкості та ефективності медичного обслуговування стане для пацієнтів поштовхом для зростання їх поваги до медичної системи, що можна вважати додатковою соціальною метою.

Наостанок, головною ціллю створення цього застосунку для власне розробників є збільшення інтересу до розробок у цій галузі. Додатковою метою можна вважати можливість створювати власні клієнт-серверні застосунки, спираючись на застосовувані технології.

Крім того, можна окремо визначити загальну ціль, що об'єднує у собі бажання лікарів, пацієнтів та розробників, а саме – якісні зміни в роботі системи медичного обслуговування. Адже лише за умови злагодженої роботи усіх діючих осіб медичної системи можна досягти максимального її розквіту та покращення.

1.5 Висновки до розділу

Клієнт-серверний застосунок для обліку пацієнтів у лікарні має високу актуальність розробки. Перш за все, це пов'язано з використанням мови програмування JavaScript, яка володіє не лише високою швидкістю, а й забезпечує надійність безпеки даних користувачів. Модернізація роботи готового застосунку відбувається за допомогою використання новітніх технологій та бібліотек, які забезпечують збільшення функціоналу, гарантують якість роботи системи безпеки та полегшують власне процес розробки.

Зазвичай, такі застосунки створюються на базі клієнт-серверної архітектури, з визначення якої випливає, що клієнт і сервер фактично виступають у ролях програмного забезпечення. При створенні застосунку буде застосовуватись клієнт-серверна архітектура з трирівневою архітектурою, що пояснюється досягненням найбільшої функціональності для користувачів.

Незважаючи на наявність досить вузької сфери застосування – медичної галузі – застосунок має широкі можливості використання та виступає корисним інструментом у руках кожної медичної установи.

Основна задача розробки – це можливість модернізації сучасної системи обліку пацієнтів у лікарнях та переведення усього документообігу в електронний варіант.

Основні цілі застосунка умовно поділяються на три категорії: для лікарів, для пацієнтів та для власне розробників. Це питання детально висвітлене у розділі. Але загальною метою всього створення цього застосунку є покращення системи медичного обслуговування.

2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

2.1 Існуючі клієнт-серверні застосунки обліку пацієнтів

Далі будуть розглянуті декілька прикладів з існуючих у наш час систем обліку пацієнтів у лікарнях.

2.1.1 Helsi

Helsi — це сучасна електронна медична система, створена для пацієнтів, лікарів, приватних та державних медичних закладів на базі мови програмування JavaScript та клієнтської бібліотеки React [4].

До головних переваг даної платформи можна віднести простоту та зрозумілість інтерфейсу, досить високий функціонал — наявність різноманітних опцій як для лікарів та керівників медичних закладів, так і для пацієнтів, а також наявність надійної системи захисту даних пацієнтів.

Серед недоліків можна виділити відсутність функції створення електронної медичної картки до запису на прийом, що унеможливлює для пацієнта доповнення картки власними медичними даними, отриманими в інших медичних закладах, та відсутність фільтру лікарів за фінансовими можливостями пацієнта. Також недоліком є те, що жителі не всіх міст України можуть бути користувачами даної платформи, адже лікарі не з усієї України є користувачами платформи.

Основні переваги:

- простота та зрозумілість інтерфейсу;
- велика кількість функцій як для пацієнтів, так і для лікарів;
- відповідність вимогам сучасної медичної реформи;
- надійні системи захисту даних пацієнтів;
- можливість налаштування платформи згідно вимог окремих державних та приватних установ для власного користування.

Основні недоліки:

– неможливість створення електронної медичної картки пацієнта до першого запису на прийом;
– відсутність сортування лікарів за фінансовими можливостями пацієнта;
– поширеність платформи та її функцій не на всю територію України.

2.1.2 Поліклініка без черг

Поліклініка без черг — це система для зручного онлайн запису до лікарів різних фахів, створена на базі фреймворку Microsoft ASP.NET у якості серверної частини та JavaScript фреймворку AngularJS у якості клієнтської частини [5].

Головною перевагою даної системи є повна відповідність вимогам сучасної медичної реформи та можливість не лише записатись на прийом, а й викликати свого сімейного лікаря додому.

Проте, ця платформа має більше недоліків, ніж переваг. Перш за все, недоліком є відсутність зручного інтерфейсу — для реєстрації на сайті та запису на прийом необхідно спочатку користуватись окремою незрозумілою інструкцією, наданою платформою. Крім того, вона має суто регіональне значення і діє лише на території міста Києва. Також є один недолік ще на етапі розробки, а саме — використання повільного фреймворку AngularJS [6], що є недоречним і зайвим для такої системи.

Основні переваги:

– можливість виклику лікаря додому;
– відповідність медичній реформі.

Основні недоліки:

– незручність та незрозумілість інтерфейсу;
– надто локальна поширеність платформи;
– використання повільних технологій для реалізації клієнтської частини.

2.1.3 «Доктор Элекс»

«Доктор Элекс» — це медична інформаційна система для українських державних та приватних лікарень, створена на базі платформи .Net Framework 3.5, сервер імплементований на основі технології Windows Communication Foundation (WCF) [7].

Перевагами даної системи є можливість лікаря швидко проводити медичний огляд, користуючись підказками системи, та можливість за наявності відповідності системним вимогам організувати за допомогою застосунку роботу клініки з більш як 201 працівником.

Основним величезним недоліком даної системи є відсутність пацієнта у користувачах. Для нього немає передбачених функцій та можливості доступу до власних медичних даних. Цю систему можна розглядати лише як засіб автоматизації та діджиталізації роботи медичного закладу. Іншим недоліком є висока вартість інтеграції та обслуговування даної програми і ненадійність захисту даних пацієнтів. Окремою величезною проблемою є використання застарілих технологій при створенні застосунку.

Основні переваги:

- зручність у роботі для лікарів;
- можливість організації роботи лікарні з великою кількістю працівників.

Основні недоліки:

- відсутність доступу для пацієнта;
- застарілі технології;
- висока вартість інтеграції та обслуговування системи;
- локальність системи, тобто можливість інтеграції її лише в межах однієї клініки;
- ненадійний захист даних пацієнтів.

2.1.4 ASKEP

ASKEP — це медична інформаційна система, реалізована на базі мови програмування PHP з використанням фреймворку Laravel [8].

Основними перевагами даної системи є широкий набір функцій як для лікарів і керівників медичних закладів, так і для пацієнтів, а також надійність систем захисту інформації і зручність інтерфейсу. Крім того, система відповідає вимогам сучасної медичної реформи та є офіційно акредитованою Міністерством охорони здоров'я.

Серед недоліків можна виділити те, що дана система є платною і те, що в презентації системи та на її веб-сайті вказана велика кількість функцій, яких насправді немає у системі.

Основні переваги:

- широкий набір функцій;
- зручність інтерфейсу;
- надійність захисту даних;
- можливість переходу з інших платформ без втрати даних;
- відповідність сучасній медичній реформі.

Основні недоліки:

- відсутність можливості безкоштовного користування;
- запевнення користувачів у наявності функцій, які насправді ще не інтегровані у платформу.

2.1.5 Медікіт

Медікіт — це сучасна телемедична система, реалізована на базі Microsoft ASP.NET [9].

Головною перевагою даної платформи є унікальність застосовуваної технології через те, що на даний момент в Україні немає єдиного лідера в галузі

телемедицини. Іншою перевагою є те, що для користувачів створено зручний і зрозумілий інтерфейс.

Серед недоліків можна виділити, перш за все, те, що сервіс є платним. Також компанія-розробник особисто визнає, що платформа є незахищеною від помилок, вірусів, зовнішніх атак та їх наслідків.

Основні переваги:

- унікальність застосовуваної технології;
- зручність і зрозумілість інтерфейсу.

Основні недоліки:

- відсутність можливості безкоштовного користування;
- вразливість платформи та незахищеність даних пацієнта;
- відсутність єдиної медичної картки пацієнта;
- досить невисокий функціонал системи.

2.2 Висновки до розділу

Розглянуті вище системи та застосунки наочно відображають те, що клієнт-серверні застосунки обліку пацієнтів у лікарні можуть бути реалізовані за допомогою найрізноманітніших технологій та можуть мати будь-який функціональний обсяг — від мінімальної кількості простих можливостей до максимально різносторонніх функцій. Детально проаналізувавши усі розглянуті системи можна виділити основні моменти, що мають бути присутніми у розробленому застосунку, а саме:

- використання швидких та сучасних технологій;
- простий та зрозумілий інтерфейс;
- простота користування та налаштування;
- широкий набір функцій;
- наявність окремого доступу для лікарів, керуючого персоналу та пацієнтів;

– комплексне та структуроване зберігання усіх даних із швидким доступом до них;

– високий ступінь безпеки.

Для створення власного застосунку слід, перш за все, розглянути усі переваги та недоліки, властиві вже існуючим застосункам та системам. Це необхідно для уникнення помилок, допущених розробниками цих засобів. Особливу увагу слід приділяти технічній базі розробки, адже вона має відповідати швидкості та обсягам процесів, для яких призначена. Крім того, особливого значення треба надати різноманітності функцій, що надає застосунок. Має бути чітко витримана межа між необхідними функціями та такими, що не є необхідними для повноцінної роботи даного продукту.

3 ВАРІАНТИ ВИКОРИСТАННЯ СИСТЕМИ

3.1 Завдання застосування варіантів використання

При створенні власного застосунку розробник має, перш за все, чітко розуміти, яким чином повинна працювати його система. На сьогоднішній день популярним засобом для опису функціоналу системи є варіанти використання.

Варіанти використання являють собою опис дій та їх послідовностей, що можуть здійснюватися у відповідь на вимоги користувачів або інших систем. Вони демонструють функціональність та гнучкість системи з точки зору користувача та результату, який він бажає отримати.

Для розробника варіанти використання є наочним відображенням функціональних вимог до системи. Вони виконують роль фундаменту для процесу розробки.

За допомогою варіантів використання можуть бути описані вимоги користувача (актора) до взаємодії з системою для досягнення визначеної мети [10]. Найкращим способом для опису варіантів використання є діаграма прецедентів, а також текст. Діаграму прецедентів наведено у кресленику ІТ61.130БАК.004 Д1.

3.2 Текстовий опис варіантів використання

Текстовий опис усіх сценаріїв використання застосунку наведено у таблиці 3.1.

Таблиця 3.1 – Сценарії використання системи

Use Case Id	Назва сценарію використання
UC-1	Перегляд сторінки авторизації користувача
UC-2	Авторизація пацієнта (Log in as Patient)

Use Case Id	Назва сценарію використання
UC-2.1	Перегляд профілю пацієнта
UC-2.1.1	Перегляд даних електронної медичної картки
UC-2.1.2	Записатися на прийом (Create Appointment)
UC-2.1.3	Відмінити прийом (Cancel Appointment)
UC-3	Вихід з системи (Log out)
UC-4	Реєстрація пацієнта (Register)
UC-5	Авторизація лікаря (Log in as Doctor)
UC-5.1.	Перегляд профілю лікаря
UC-5.1.1	Перегляд прийомів
UC-5.1.2	Перегляд профілю пацієнта, записаного на прийом (Patient`s Profile)
UC-5.1.3	Додавання інформації до електронної медичної картки пацієнта
UC-6	Авторизація адміністратора (Log in as Admin)
UC-6.1	Перегляд усіх лікарів (All Doctors)
UC-6.1.1	Додавання лікаря (Create Doctor)
UC-6.1.2	Видалення лікаря (Delete Doctor)
UC-7	Перегляд головної сторінки (Home)

У таблицях 3.2-3.19 наведено детальний опис кожного з сценаріїв використання системи.

Опис сценарію з ідентифікатором UC-1 представлено у таблиці 3.2.

Таблиця 3.2 – Сценарій використання UC-1

Назва	Перегляд сторінки авторизації користувача
Опис	Користувач може переглянути сторінку авторизації

Продовження таблиці 3.2

Актори	Неавторизований Користувач
Передумови	
Постумови	Відображено сторінку авторизації
Успішний сценарій	1. Система відображує кнопку «Log in». 2. Користувач натискає кнопку «Log in». 3. Система відображує сторінку авторизації.
Розширення	

Опис сценарію з ідентифікатором UC-2 представлено у таблиці 3.3.

Таблиця 3.3 – Сценарій використання UC-2

Назва	Авторизація пацієнта
Опис	Користувач може авторизуватися у системі як пацієнт на сторінці авторизації
Актори	Користувач
Передумови	Користувач перейшов на сторінку авторизації
Постумови	Користувача авторизовано системою як пацієнта
Успішний сценарій	1. Система відображує сторінку авторизації. 2. Користувач заповнює форму вводу, де обирає роль пацієнта. 3. Користувач натискає кнопку «Log in». 4. Система авторизує користувача.
Розширення	4.1 Системою виявлено, що дані користувача введено невірно. 4.2 Система відображає спливаюче повідомлення про помилку авторизації.

Опис сценарію з ідентифікатором UC-2.1 представлено у таблиці 3.4.

Таблиця 3.4 – Сценарій використання UC-2.1

Назва	Перегляд профілю пацієнта
Опис	Після авторизації користувача як пацієнта, система направляє на сторінку профілю, де пацієнт може продивитися свій профіль
Актори	Пацієнт
Передумови	Користувач має бути авторизований як пацієнт
Постумови	Система відображує сторінку профілю пацієнту
Успішний сценарій	1. Користувача авторизовано як пацієнта. 2. Система перенаправляє пацієнта на його профіль.
Розширення	1.1 Система визначила, що користувач не авторизувався у системі як пацієнт. 1.2 На сторінці відображується пропозиція авторизуватися.

Опис сценарію з ідентифікатором UC-2.1.1 представлено у таблиці 3.5.

Таблиця 3.5 – Сценарій використання UC-2.1.1

Назва	Перегляд даних електронної медичної картки
Опис	Пацієнт може переглянути медичні дані, що занесені до його електронної медичної картки
Актори	Пацієнт
Передумови	Пацієнт авторизувався у системі; Пацієнт знаходиться на сторінці свого профілю
Постумови	Система відображує медичні дані з електронної медичної картки

Продовження таблиці 3.5

Успішний сценарій	1. Пацієнт знаходиться на сторінці свого профілю. 2. Система відображує блок з медичними даними. 3. Пацієнт обирає тип необхідних даних. 4. Система відображує дані, запитані користувачем.
Розширення	4.1 Система визначила, що медичні дані у електронній медичній картці відсутні. 4.2 Система відображує повідомлення, яке інформує про те, що дані відсутні.

Опис сценарію з ідентифікатором UC-2.1.2 представлено у таблиці 3.6.

Таблиця 3.6 – Сценарій використання UC-2.1.2

Назва	Записатися на прийом (Create Appointment)
Опис	Пацієнт має змогу записатися на прийом
Актори	Пацієнт
Передумови	Пацієнта авторизовано у системі; Пацієнт знаходиться на сторінці свого профілю
Постумови	У системі створюється новий прийом для пацієнта і лікаря
Успішний сценарій	1. Пацієнт знаходиться на сторінці свого профілю. 2. Система демонструє блок з інформацією про прийоми. 3. Пацієнт натискає кнопку створення прийому. 4. Система відображує модальне вікно, в якому знаходиться форма запису на прийом. 5. Пацієнт заповнює форму запису на прийом. 6. Пацієнт натискає кнопку «Create Appointment». 7. Система створює та відображує новий прийом.

Продовження таблиці 3.6

Розширення	7.1 Система визначила, що введені дані невірні. 7.2 Система відображує повідомлення, про некоректні дані.

Опис сценарію з ідентифікатором UC-2.1.3 представлено у таблиці 3.7.

Таблиця 3.7 – Сценарій використання UC-2.1.3

Назва	Відмінити прийом (Cancel Appointment)
Опис	Пацієнт має змогу відмінити будь-який прийом, на який він записався
Актори	Пацієнт
Передумови	Пацієнт знаходиться на сторінці свого профілю
Постумови	Система видаляє прийом для лікаря та пацієнта
Успішний сценарій	1. Пацієнт знаходиться на сторінці свого профілю. 2. Система відображує блок усіх прийомів, на які записано пацієнта. 3. Пацієнт натискає кнопку «Cancel Appointment» навпроти запису на прийом, який він хоче відмінити. 4. Система видаляє прийом для лікаря та пацієнта.
Розширення	

Опис сценарію з ідентифікатором UC-3 представлено у таблиці 3.8.

Таблиця 3.8 – Сценарій використання UC-3

Назва	Вихід з системи (Log out)
Опис	Надає змогу користувачеві вийти зі свого аккаунту в системі.

Продовження таблиці 3.8

Актори	Пацієнт, Адміністратор, Лікар
Передумови	Користувач авторизувався у системі під будь-якою роллю
Постумови	Користувач виходить із системи та більше не має доступу до контенту, який потребує авторизації
Успішний сценарій	1. Система демонструє кнопку «Log out». 2. Користувач натискає кнопку «Log out». 3. Система перенаправляє користувача на головну сторінку.
Розширення	

Опис сценарію з ідентифікатором UC-4 представлено у таблиці 3.9.

Таблиця 3.9 – Сценарій використання UC-4

Назва	Реєстрація пацієнта (Register)
Опис	Користувач має змогу зареєструвати акаунт у системі
Актори	Неавторизований Користувач
Постумови	Система створює аккаунт нового пацієнта
Успішний сценарій	1. Система відображує кнопку «Register». 2. Користувач натискає кнопку «Register». 3. Система перенаправляє користувача на сторінку реєстрації. 4. Система відображує сторінку реєстрації, на якій знаходиться форма вводу. 5. Користувач заповнює форму. 6. Користувач натискає кнопку «Register account». 7. Система створює аккаунт нового пацієнта.

Продовження таблиці 3.9

	8. Система перенаправляє користувача на сторінку авторизації.
Розширення	7.1 Система фіксує некоректні дані, які відправив користувач. 7.2 Система відображує повідомлення з текстом відповідної помилки.

Опис сценарію з ідентифікатором UC-5 представлено у таблиці 3.10.

Таблиця 3.10 – Сценарій використання UC-5

Назва	Авторизація лікаря
Опис	Користувач може авторизуватися у системі як лікар на сторінці авторизації
Актори	Користувач
Передумови	
Постумови	Користувача авторизовано системою як лікаря
Успішний сценарій	1. Система відображує сторінку авторизації. 2. Користувач заповнює форму вводу, де обирає роль лікаря. 3. Користувач натискає кнопку «Log in». 4. Система авторизує користувача як лікаря.
Розширення	4.1 Системою виявлено, що дані користувача введено невірно. 4.2 Система відображає спливаюче повідомлення про помилку авторизації.

Опис сценарію з ідентифікатором UC-5.1. представлено у таблиці 3.11.

Таблиця 3.11 – Сценарій використання UC-5.1

Назва	Перегляд профілю лікаря
Опис	Після авторизації лікаря система перенаправляє авторизованого лікаря на сторінку його профілю
Актори	Лікар
Передумови	Лікар авторизувався у системі
Постумови	Система відображує профіль лікаря
Успішний сценарій	1. Користувача авторизовано, як лікаря. 2. Користувача перенаправляє на його профіль.
Розширення	1.1 Система визначила, що користувач не авторизувався у системі. 1.2 На сторінці відображується пропозиція авторизуватися.

Опис сценарію з ідентифікатором UC-5.1.1 представлено у таблиці 3.12.

Таблиця 3.12 – Сценарій використання UC-5.1.1

Назва	Перегляд прийомів
Опис	Лікар може переглянути всі прийоми
Актори	Лікар
Передумови	Лікар знаходиться на сторінці свого профілю
Постумови	Система відображує блок з усіма прийомами
Успішний сценарій	1. Лікар знаходиться на сторінці свого профілю. 2. Система відображує блок з усіма прийомами.
Розширення	2.1 Система визначила, що немає жодного запису на прийом. 2.2 Система інформує лікаря відповідним повідомленням.

Опис сценарію з ідентифікатором UC-5.1.2 представлено у таблиці 3.13.

Таблиця 3.13 – Сценарій використання UC-5.1.2

Назва	Перегляд профілю пацієнта, записаного на прийом (Patient`s Profile)
Опис	Лікар має змогу передивитися профіль пацієнта, який записався до нього на прийом
Актори	Лікар
Передумови	Лікар знаходиться на сторінці свого профілю
Постумови	Система відображує блок, у якому міститься інформація про пацієнта
Успішний сценарій	1. Система відображує блок з усіма прийомами. 2. Система відображує кнопку «Patient`s profile» навпроти кожного запису. 3. Лікар натискає кнопку «Patient`s profile». 4. Система відображує профіль пацієнта.
Розширення	

Опис сценарію з ідентифікатором UC-5.1.3 представлено у таблиці 3.14.

Таблиця 3.14 – Сценарій використання UC-5.1.3

Назва	Додавання інформації до електронної медичної картки пацієнта
Опис	Лікар має можливість застосуни до електронної медичної картки пацієнта медичні дані пацієнта, а також інформацію щодо лікування
Актори	Лікар
Передумови	Система відображує інформацію про пацієнта, який записався на прийом

Продовження таблиці 3.14

Постумови	Система додає в електронну медичну картку пацієнта інформацію, яку заповнив лікар
Успішний сценарій	1. Система відображує форму. 2. Лікар заповнює форму необхідними медичними даними та рекомендаціями щодо лікування. 3. Лікар натискає кнопку «Add information». 4. Система додає в електронну медичну картку пацієнта інформацію, яку заповнив лікар.
Розширення	

Опис сценарію з ідентифікатором UC-6 представлено у таблиці 3.15.

Таблиця 3.15 – Сценарій використання UC-6

Назва	Авторизація адміністратора (Login as Admin)
Опис	Неавторизований Користувач має змогу авторизуватися у системі, як адміністратор
Актори	Неавторизований Користувач
Передумови	
Постумови	Користувача авторизовано, як адміністратора
Успішний сценарій	1. Система відображує сторінку авторизації. 2. Користувач заповнює форму вводу, де обирає роль адміністратора. 3. Користувач натискає кнопку «Log in». 4. Система авторизує користувача як лікаря.
Розширення	4.1 Системою виявлено, що дані користувача введено невірно. 4.2 Система відображає спливаюче повідомлення про помилку авторизації.

Опис сценарію з ідентифікатором UC-6.1 представлено у таблиці 3.16.

Таблиця 3.16 – Сценарій використання UC-6.1

Назва	Перегляд усіх лікарів (All Doctors)
Опис	Адміністратор має можливість переглядати усіх зареєстрованих лікарів у системі
Актори	Адміністратор
Передумови	Користувач авторизувався як адміністратор
Постумови	Система відображує список усіх зареєстрованих у системі лікарів
Успішний сценарій	1. Система відображує кнопку «All Doctors». 2. Адміністратор натискає кнопку «All Doctors». 3. Система відображує список усіх лікарів.

Опис сценарію з ідентифікатором UC-6.1.1 представлено у таблиці 3.17.

Таблиця 3.17 – Сценарій використання UC-6.1.1

Назва	Додавання лікаря (Create Doctor)
Опис	Адміністратор може створити аккаунт лікаря
Актори	Адміністратор
Передумови	Адміністратор знаходиться на сторінці, яка відображує список усіх лікарів
Постумови	У системі створено аккаунт для нового лікаря
Успішний сценарій	1. Система відображує кнопку «Create Doctor». 2. Адміністратор натискає кнопку «Create Doctor». 3. Система відображує форму вводу. 4. Адміністратор заповнює форму даними лікаря. 5. Адміністратор натискає кнопку «Submit».

Продовження таблиці 3.17

Розширення	5.1 Система визначила, що такий лікар існує. 5.2 Система виводить повідомлення про помилку з відповідним текстом.

У таблиці 3.18 представлено опис сценарію з ідентифікатором UC-6.1.2.

Таблиця 3.18 – Сценарій використання UC-6.1.2

Назва	Видалення лікаря (Delete Doctor)
Опис	Адміністратор має можливість видалити аккаунт лікаря з системи
Актори	Адміністратор
Передумови	Адміністратор знаходиться на сторінці, яка відображує список усіх зареєстрованих лікарів
Постумови	Лікаря видалено з системи
Успішний сценарій	1. Система відображує кнопку «Delete Doctor» навпроти кожного запису у списку лікарів. 2. Адміністратор натискає кнопку «Delete Doctor». 3. Система видаляє лікаря з системи.
Розширення	

У таблиці 3.19 представлено опис сценарію з ідентифікатором UC-.

Таблиця 3.19 – Сценарій використання UC-7

Назва	Перегляд головної сторінки (Home)
Опис	Будь-який користувач системи може переглянути головну сторінку
Актори	Будь-який користувач

Продовження таблиці 3.19

Передумови	
Постумови	Відображається головна сторінка
Успішний сценарій	1. Користувач потрапляє на сайт. 2. Система відображує головну сторінку.
Розширення	

3.3 Висновки до розділу

У цьому розділі було визначено поняття варіантів використання та окреслено основні їх характеристики і функціональне призначення.

Для розробки функціональних вимог до системи, а також для систематизації самого процесу розробки було створено перелік усіх можливих сценаріїв використання системи, а також наведено структурований текстовий опис кожного окремого сценарію використання клієнт-серверного застосунку для обліку пацієнтів у лікарні.

Крім того, було розроблено діаграму прецедентів, що являє собою зручний підсумований опис варіантів використання системи.

4 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Підхід до аналізу вимог

На етапі аналізу вимог до програмного забезпечення необхідно сформувати список унікальних вимог до системи, які мають бути виокремлені з надлишкових та тих, що дублюються, сценаріїв. Чітко сформовані вимоги допоможуть обійтися мінімальним функціоналом задля задоволення максимально великої кількості потреб, що в свою чергу дозволяє заощадити витрати та не дозволить вийти проєкту за рамки поставленої задачі.

4.2 Розробка функціональних вимог до системи

Функціональні вимоги – це перелік вимог, які система повинна виконувати, як вона має поводити себе у визначених ситуаціях[11].

На основні опису варіантів використання можна виокремити наступні функціональні вимоги:

- авторизація користувачів на основі ролей;
- можливість перегляду пацієнтом свого профілю;
- можливість перегляду пацієнтом своєї електронної медичної картки;
- можливість пацієнта записатися на прийом;
- можливість пацієнта відмінити прийом;
- можливість перегляду лікарем свого профілю;
- можливість лікаря переглядати усі його прийоми;
- можливість переглянути профіль пацієнта, який записався на прийом;
- можливість лікаря вносити записи до електронної медичної картки пацієнта;
- можливість адміністратора додавати лікарів;
- можливість адміністратора видаляти лікарів;
- можливість користувача вийти з системи.

4.3 Розробка нефункціональних вимог до системи

Нефункціональні вимоги до системи – це такі вимоги, що описують характеристики системи, а не її поведінку [12].

На основі проведеного аналізу предметної області дипломного проєкту та на основі огляду та аналізу існуючих рішень було визначено наступні нефункціональні вимоги до системи:

– масштабованість – вимога до системи, яка означає, що система має бути достатньо гнучкою для подальшого нарощення функціоналу;

– зручність використання (Usability) – вимога до системи з точки зору користувача. Визначає вимоги до зручності використання користувацького інтерфейсу;

– перенесення – система повинна бути адаптована та готова до розгортання на будь-якій платформі;

– підтримка – серед вимог до підтримки, основними є швидкість розробки, прозорість поведінки застосунку, простота аналізу помилок та проблем у роботі системи;

– безпека – система повинна бути стійкою до CSRF (Cross Site Request Forgery) та XSS (Cross-Site Scripting) атак, також паролі повинні зберігатися у зашифрованому вигляді.

4.4 Висновки до розділу

Головною задачею аналізу вимог є задоволення максимальної кількості потреб користувача за допомогою мінімального набору функцій. Необхідність складання чітко сформованих вимог полягає в тому, що це дозволить попередити вихід проєкту за рамки поставленої задачі.

У розділі було висунуто функціональні та нефункціональні вимоги до програмного забезпечення. Функціональними називають такі вимоги, що

характеризують безпосередньо поведінку програми, а нефункціональними – вимоги, що описують характеристики самої системи. Функціональні вимоги складаються, головним чином, з можливостей користувачів у системі та створюються на підґрунті описів варіантів використання. Нефункціональні вимоги являють собою перелік «якостей», якими має володіти розроблена система. Вони, в свою чергу, створюються на основі аналізу предметної області та аналізу існуючих рішень.

Даний етап – складання функціональних і нефункціональних вимог – є обов’язковим при розробці будь-якого застосунку, адже значним чином допомагає структурувати та оптимізувати процес розробки програмного забезпечення.

5 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

5.1 Огляд технологій для розв’язання поставленої задачі

На сьогоднішній день існує незліченна кількість технологій, які дозволяють зручно та якісно розробляти застосунки з клієнт-серверною архітектурою.

Для розробки серверної частини можна виділити найбільш популярні мови програмування, такі як C#, Java, Python, PHP, JavaScript, та відповідні їм фреймворки – ASP.NET, Spring/Spark, Django, Symfony/Laravel/Zend, Node.js (Express.js) [13].

Найбільш широко використовувана мова програмування для клієнтської частини – JavaScript. На основі JavaScript існує багато фреймворків та бібліотек, які дозволяють зробити розробку більш простою та гнучкою. Серед таких засобів можна відмітити Angular, React.js, JQuery, Vue.js [14]. Також варто зазначити, що невід’ємною частиною побудови web-інтерфейсів є мова гіпертекстової розмітки HTML та каскадні таблиці стилів CSS, за допомогою яких виконується організація контенту застосунку, а також його стилізація.

Вибір стеку технологій повинен ґрунтуватись на вимогах до майбутнього проєкту, а також на професійних навичках команди розробки.

Для розробки клієнт-серверного застосунку для обліку пацієнтів у лікарні було обрано наступні засоби та технології:

- мова програмування JavaScript, зокрема середа виконання Node.js;
- фреймворк Express.js;
- бібліотека Objection.js;
- бібліотека passport.js та jsonwebtoken для автентифікації та авторизації користувачів;
- бібліотека React.js для клієнтського застосунку;
- бібліотека Redux для керування станом клієнтського застосунку на React.js;
- бібліотека для стилізації web-інтерфейсів Bootstrap;
- бібліотека стилізованих React-компонентів Material-UI;

– бібліотека axios у якості HTTP-клієнта;
– система контролю версій git ;
– веб-сервіс для колективної розробки GitHub;
– система керування базами даних PostgreSQL.

5.1.1 JavaScript та Node.js

JavaScript – мультипарадігменна мова програмування. Підтримує об’єктно-орієнтований, імперативний, а також функціональний стилі. Node.js – це серверна платформа для роботи с JavaScript [15].

Основні переваги:

– можливість використання однієї мови програмування на клієнті та сервері;
– висока швидкість розробки;
– стрімкий розвиток технології;
– просто масштабується;
– існує велика кількість бібліотек, розроблених спільнотою, які можна встановити за допомогою npm (менеджер пакетів Node);
– операції блокуючого вводу/виводу.

Основні недоліки:

– через швидкий розвиток, іноді нові можливості платформи можуть бути не до кінця налагодженими та стабільними;
– при вирішенні задач, які потребують багато ресурсів процесору, головний потік може блокуватись, але цю проблему можна вирішити написавши цей модуль на мові C++, адже платформа Node.js дозволяє це зробити.

5.1.2 Express.js

Express.js – фреймворк для розробки web-застосунків створених за допомогою платформи Node.js. [16]

Основні переваги:

– легкий та швидкий;

– гнучкість – можливість розширити функціонал за допомогою зовнішніх модулів;

– непогана документація.

Недоліки:

– треба встановлювати багато додаткових модулів.

5.1.3 Objection.js

Objection.js – ORM (object-relation mapping) – бібліотека, яка дозволяє описати сутності та зв’язки між ними і визначити те, як будуть відображатися ці сутності на базу даних. [17]

Переваги:

– швидкість;

– вбудована можливість робити нативні SQL-запроси;

– прозоре та зрозуміле створення моделей та відношень між ними;

– дуже гарна документація.

Недоліки:

– не має можливості автоматично створювати схему бази даних;

– неможливо створити міграцію на основі існуючої моделі, тому її треба створювати вручну.

5.1.4 passport.js та jsonwebtoken

passport.js – це проміжне програмне забезпечення (middleware) для аутентифікації користувачів.

jsonwebtoken – це відкритий стандарт для створення токенів доступу у форматі JSON. [18]

Основні переваги:

– максимально гнучке рішення;

– підтримує як базову аутентифікацію та авторизацію, так і авторизацію через численні соціальні мережі та сервіси;

– токени підписуються секретним ключем.

Недоліки:

– може бути вразливим до деяких видів атак, таких як CSRF (Cross Site Request Forgery) або XSS (Cross-Site Scripting).

5.1.5 React.js

React.js – це JavaScript бібліотека, яка була створена, щоб розробка користувацьких інтерфейсів стала більш швидкою та зручною, використовується при побудові SPA (Single-Page Application) або мобільних застосунків. [19]

Основні переваги:

– дуже наглядна та зрозуміла документація;

– зручне розділення на компоненти, це позитивно впливає на гнучкість та перевикористання коду;

– код, написаний за допомогою React, легко читати та підтримувати;

– великі можливості для масштабування;

– використовується Virtual DOM (document object model), що дозволяє оновити тільки потрібний елемент DOM дерева, що дозволяє мати більшу швидкодію.

Недоліки:

– React потребує додаткових бібліотек, для вирішення деяких задач;

– не досконалий менеджмент станів.

5.1.6 Redux

Redux – JavaScript бібліотека, яка часто використовується у зв’язці з React, та призначена для управління станами застосунку. Redux реалізує Flux-архітектуру –

шаблон проєктування користувацьких інтерфейсів, який поєднується з реактивним програмуванням та побудований на однонаправлених потоках даних. [20]

Переваги технології:

– простий доступ до спільного стану застосунку з різних компонентів;
– зручність налагодження та тестування;
– ізоляція станів від побічних ефектів;
– існує розширення для браузера, який дозволяє відстежувати усі зміни даних застосунку.

Недоліки:

– треба добре слідкувати за іммутабельністю даних;
– необхідність писати багато однотипного коду;
– для асинхронних операцій потрібно використовувати сторонні middleware бібліотеки.

5.1.7 Bootstrap

Bootstrap – це набір інструментів для створення графічних елементів web-застосунків. Включає в себе шаблони для типографіки, форм, кнопок, блоків навігації та інших компонентів web-інтерфейсів. [21]

Основні переваги:

– прискорює розробку та економить час на стилізацію компонентів;
– широкий вибір компонентів та їх варіацій;
– кросс-браузерність – можливість застосування в усіх браузерах;
– гарні приклади у документації.

Основні недоліки:

– елементи є шаблонними та не оригінальними;
– може мати чималий розмір;
– треба час на освоєння існуючих класів.

5.1.8 Material UI

Material UI – бібліотека готових типових React-компонентів, яка допомагає сильно прискорити розробку користувацького інтерфейсу. [22]

Переваги:

– багато готових складних компонентів, що дозволяє не витрачати час на їх розробку;
– гнучка стилізація існуючих компонентів.

Недоліки:

– має великий розмір;
– іноді може потребувати встановлення додаткових залежностей.

5.1.9 Axios

Axios – це JavaScript бібліотека, яка являє собою HTTP-клієнт, який базується на promise (зручний спосіб представлення асинхронного коду) та призначений для браузерів та Node.js. [23]

Основні переваги:

– невеликий об’єм шаблонного коду;
– інформативні повідомлення про помилки;
– дозволяє встановити значення за замовчуванням для запитів;
– автоматична конвертація відповіді у JSON формат.

Недоліки:

– оновлення бібліотеки з’являються нечасто;
– потрібно встановлювати як додаткову залежність.

5.1.10 git

git — це розподілена система керування версіями. Вона являє собою набір програм, розроблених спеціально з урахуванням їх застосування у сценаріях

(програмах, що працюють з готовими програмними компонентами). Це дозволяє зручно створювати спеціалізовані системи контролю версій на базі git або інтерфейси користувача. [24]

Переваги:

– підтримка швидкого розділення та злиття версій;

– наявність інструментів для візуалізації та навігації нелінійною історією розробки;

– можливість доступу кожного окремого розробника до всієї історії розробки;

– цілісність даних.

Недоліки:

– криптографічна ненадійність створюваних хеш-функцій;

– важкість використання на Microsoft Windows;

– важкість редагування великої кількості не пов’язаних між собою файлів через комплексний підхід системи до проєкту.

5.1.11 GitHub

GitHub – це веб-сервіс для хостингу IT-проєктів та їх колективної розробки. Веб-сервіс заснований на системі контролю версій git і є безкоштовним для проєктів з відкритим вихідним кодом та невеликих приватних проєктів. [25]

Переваги:

– можливість безкоштовного розміщення коду на загальний огляд;

– велика спільнота користувачів-розробників, які можуть коментувати дані один одного;

– наявність власної бази типових помилок;

– велика кількість корисних функцій для роботи з кодом;

– швидкий та зрозумілий інтерфейс;

– надійність системи захисту даних.

Недоліки:

– недостатня підтримка надто великих об’ємів даних;
– відсутність можливості співпраці у реальному часі;
– можлива незрозумілість інтерфейсу для нових користувачів.

5.1.12 PostgreSQL

PostgreSQL – це об’єктно-реляційна система керування базами даних (СКБД). Завдяки потужним технологіям PostgreSQL дуже продуктивна. [26]

Переваги:

– ця система керування базами даних є безкоштовною;
– відповідає стандартам SQL;
– існує можливість розширення функціоналу за рахунок створення власних процедур;
– PostgreSQL не тільки реляційна система керування базами даних, а й об’єктно-орієнтована з підтримкою наслідування тощо;
– забезпечує надійність у питаннях цілісності даних.

Недоліки:

– в деяких ситуаціях у простих операціях читання або запису може поступатися іншим СКБД.

5.2 Висновки до розділу

Проаналізувавши найбільш популярні засоби та технології, що використовуються для розробки клієнт-серверних застосунків, було обрано платформу Node.js у поєднанні з web-фреймворком Express.js у якості серверної частини та бібліотеку React.js у зв’язці з менеджером станів застосунку Redux у якості клієнтської частини. В якості системи керування базами даних було обрано

СКБД PostgreSQL. Крім того, обрані бібліотеки, завданням яких є полегшення розробки користувацьких інтерфейсів та деяких інших опцій.

Використання цих засобів і технологій дозволяє швидко та зручно розробити усі необхідні компоненти для створення клієнт-серверного застосунку для обліку пацієнтів у лікарні. Обрані технології мають велику кількість переваг, що у значній мірі нівелюють наявні у них недоліки.

6 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В даному розділі описана реалізація клієнт-серверного застосунку для обліку пацієнтів у лікарні.

6.1 Структура проєкту

Застосунок складається з двох окремих проєктів для серверної та клієнтської частин відповідно.

6.1.1 Серверна частина

Структура серверної частина застосунку наведена на рисунку 6.1:

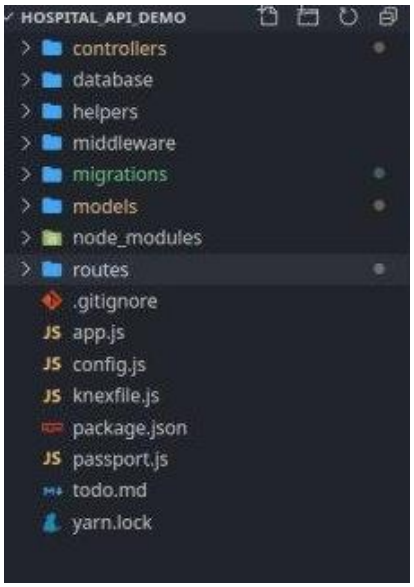

Рисунок 6.1 – Структура серверної частини проєкту

Структура проєкту серверної частина складається з наступних елементів:

– controllers – місце зберігання модулів, які обробляють вхідні HTTP-запити, використовуючи моделі, та відправляє клієнтові результат обробки у форматі JSON (JavaScript Object Notation);

– database – тека, у якій зберігаються конфігураційні файли та модулі для з’єднання за базою даних;

– helpers – місце зберігання допоміжних модулів, таких як обробку дати у потрібний формат;

– middleware – місце зберігання проміжного програмного забезпечення. Middleware-функції використовуються для проміжної обробки даних на основі даних запиту чи відповіді та передає обробку наступній функції;

– migrations – місце зберігання міграцій бази даних. Міграція – це перехід від однієї структури бази даних до іншої, зазвичай більш нової; [27]

– models – тека, у якій знаходяться модулі, які описують структуру та зв’язки між даними, які використовуються у застосунку;

– node_module – тут знаходяться усі необхідні залежності;

– routes – місце зберігання системи маршрутів, які зіставляють запити з маршрутами та обирають для обробки запиту необхідний контролер;

– .gitignore – конфігураційний файл, у якому декларуються директорії та файли, які мають ігноруватися системою контролю версій git; [28]

– app.js – головний файл, у якому ми створюємо та налаштовуємо сервер застосунку;

– package.json – файл, який полегшує налаштування та керування додатковими пакетами застосунку;

– passport.js – конфігурація для бібліотеки автентифікації та авторизації passport.js;

– yarn.lock – файл, який містить інформацію про версії додатково встановлених пакетів, що допомагає уникнути конфліктів версій на різних пристроях.

В загальному випадку серверна частина на платформі Node.js працює таким чином: коли сервер отримує HTTP-запит (request), система маршрутизації обирає потрібний контролер для обробки запиту. У процесі обробки запиту контролером, він може звертатись до даних через моделі. Результат (response) обробки контролером відправляється клієнту у вигляді JSON, але це може бути і HTML-

сторінка. Якщо при запиті виникає помилка, сервер у відповідь віддає відповідний код стану HTTP [29], основні коди помилок наведені у таблиці 6.1.

Таблиця 6.1 – Коди станів HTTP

200 OK	Запит пройшов успішно. Якщо клієнтом були запитані якісь дані, то вони будуть знаходитися у тілі повідомлення або у заголовку повідомлення.
201 Created	У результаті запиту був створений новий ресурс.
400 Bad Request	Сервер виявив помилку у запиті клієнта.
401 Unauthorized	Для доступу до ресурсу користувачеві треба пройти автентифікацію.
403 Forbidden	Сервер отримав запит, але він не може його виконати через обмеження у доступі для клієнта, тобто клієнт має недостатньо прав, щоб виконувати операції з даним ресурсом.
404 Not Found	Сервер зрозумів запит, але ресурс який відповідає вказаному URL не існує.
500 Internal Server Error	Будь-яка помилка з боку серверу.

Схему серверної частини наведено у кресленику ІТ61.130БАК.004 Д2.

У таблиці 6.2 наведена реалізація моделі, яка відповідає пацієнту.

Таблиця 6.2 – Реалізація моделі пацієнта

Назва моделі	Patients(patient.model)

Опис моделі	У моделі пацієнта описано типи зв’язків з іншими моделями, а саме: – з doctors.model – багато до багатьох, через зв’язуючу – модель appointment.model; – з analysis.model – багато до одного; – з diagnoses.model – багато до одного; – з prescription.model – багато до одного; – з role.model – багато до одного.

У таблиці 6.3 наведена реалізація моделі лікаря.

Таблиця 6.3 – реалізація моделі лікаря

Назва моделі	Doctor(doctor.model)
Опис моделі	У моделі лікаря описані зв’язки з іншими моделями, такими як: – з patients.model – багато до багатьох, через зв’язуючу модель appointment.model; – з specialty.model – багато до одного; – з role.model – багато до одного.

У таблиці 6.4 наведено опис методів контролеру лікарів, який знаходиться по маршруту «/doctors».

Таблиця 6.4 – Методи контролеру лікарів по маршруту «/doctors»

Назва методу	Опис методу
getAllDoctors(); GET-метод	Цей метод виконує вибірку всіх лікарів з бази даних. Після цього відправляє код стану HTTP 200 та результат у

Продовження таблиці 6.4

	форматі JSON. Якщо виникла помилка, то контролер відправляє код стану HTTP 500.
getDoctorById(); GET-метод	Метод шукає лікаря з ідентифікатором, який передається у маршруті, у вигляді «/{id}». Після цього відправляє код стану HTTP 200 та результат у форматі JSON. Якщо виникла помилка, то контролер відправляє код стану HTTP 500.
createDoctor(); POST-метод	Метод створює нового лікаря, усі дані передаються через тіло запиту, а також перевіряється чи існує такий лікар. Якщо такий лікар існує, то контролер повідомляє про це, відправивши HTTP код 400 та відповідне повідомлення. Якщо лікаря не існує, його данні відправляються до бази даних, при чому пароль перед цим хешується. Ідентифікатор створюється автоматично. Після цього контролер відправляє код стану HTTP 201, а також усі дані нового лікаря у форматі JSON, якщо при додаванні запису виникла помилка, контролер про це повідомляє кодом стану HTTP 500.
deleteDoctor(); DELETE-метод	Цей метод видаляє лікаря за ідентифікатором, який вказується у тілі запиту. Якщо запит пройшов успішно, то контролер повертає код 200, а також повідомлення про видалення. Якщо виникла помилка буде відправлено код 500.

У таблиці 6.5 знаходиться опис методів контролеру пацієнтів, який відповідає маршруту «/patients».

Таблиця 6.5 – Опис методів контролеру пацієнтів по маршруту «/patients»

Назва методу	Опис методу
getAllPatients();	Цей метод виконує вибірку усіх пацієнтів з бази даних. Після цього відправляє код стану HTTP 200
GET-метод	та результат у форматі JSON. Якщо виникла помилка, то контролер відправляє код стану HTTP 500.
getPatientById(); GET-метод	Метод шукає пацієнта з ідентифікатором, який передається у маршруті, у вигляді «/{id}». Після цього відправляє код стану HTTP 200 та результат у форматі JSON. Якщо виникла помилка, то контролер відправляє код стану HTTP 500.
createPatientsDiagnose(); POST-метод	Цей метод створює новий запис у базі даних про діагноз для пацієнта у відповідній таблиці, усі параметри передаються через тіло запита. Метод знаходиться по маршруту «/diagnoses». Якщо запит пройшов успішно, то контролер повертає код 200, якщо виникла помилка, то повертається код 500.
createPatientsAnalysis(); POST-метод	Цей метод створює новий запис у базі даних про аналізи для пацієнта у відповідній таблиці, усі параметри передаються через тіло запита. Метод знаходиться по маршруту «/analysis». Якщо запит. пройшов успішно, то контролер повертає код 200, якщо виникла помилка, то повертається код 500.

Продовження таблиці 6.5

createPatientsPrescripton(); POST-метод	Цей метод створює новий запис у базі даних про приписи для пацієнта у відповідній таблиці, усі параметри передаються через тіло запита. Метод знаходиться по маршруту «/prescriptions». Якщо запит пройшов успішно, то контролер повертає код 200, якщо виникла помилка, то повертається код 500.
createAppointment(); POST-метод	Цей метод створює новий запис у базі даних про прийом у відповідній таблиці, усі параметри передаються через тіло запита, ідентифікатор пацієнта контролер отримує з токену авторизації. Метод знаходиться по маршруту «/appointments». Якщо запит пройшов успішно, то контролер повертає код 200, якщо виникла помилка, то повертається код 500.

У таблиці 6.6 описано методи контролеру авторизації, який знаходиться за маршрутом «/auth».

Таблиця 6.6 – методи контролеру авторизації за маршрутом «/auth»

Назва методу	Опис методу
login(); POST-метод	Цей метод авторизує користувача у системі. Усі данні для авторизації передаються у тілі запиту. Метод перевіряє чи існує запис у базі з такими параметрами для

Продовження таблиці 6.6

	входу, якщо такого користувача не існує метод повідомляє про це відповідним повідомленням та кодом 404. Якщо ж користувач з таким даними існує, тоді порівнюється хеш введеного паролю та паролю, який знаходиться у базі даних, якщо порівняння пройшло успішно, система авторизує користувача у системі генеруючи jsonwebtoken, після цього метод повертає повідомлення про успішність входу, а також jwt-токен, який в подальшому використовується у клієнтській частині. При помилці повертається код 500.
Register(); POST-метод	Метод, який дозволяє пацієнтові зареєструватися у системі. Усі данні пацієнта, які він вказав при реєстрації передаються у тілі запиту. Система перевіряє чи існує користувач з такими даним, якщо користувача не існує, система додає нового пацієнта до бази даних, попередньо захешувавши пароль. Якщо ж такий пацієнт існує, метод повідомляє про це відповідним кодом HTTP, а також повідомленням.
getProfile(); GET-метод	Цей метод дозволяє отримати дані авторизованому користувачу. Відповідь надається у форматі JSON.
Logout(); GET-метод	Цей метод робить jwt-токен невалідним.

Варто відмітити, що усі відповіді повертаються у форматі JSON, тому що це найбільш популярний та універсальний формат для обміну даних при побудови

REST API[30]. Подальша обробка відповідей у JSON форматі відбувається на клієнтській частині застосунку.

Також слід пояснити принцип роботи автентифікації за допомогою jwt (jsonwebtoken). Сервер створює токен, який підписується секретним ключем, а потім передається користувачеві для підтвердження своєї особи. jwt токен складається з трьох частин: заголовку (header), корисного навантаження (payload) та підпису (signature). В заголовку вказуються алгоритм, який потрібно використовувати для підпису, а також тип токену. Корисне навантаження містить в собі користувацьку інформацію, таку як ім'я користувача, пароль та інше. Підпис визначається на основі заголовку та корисного навантаження. На рисунку 6.2 наведено схему автентифікації за допомогою jwt-токенів.

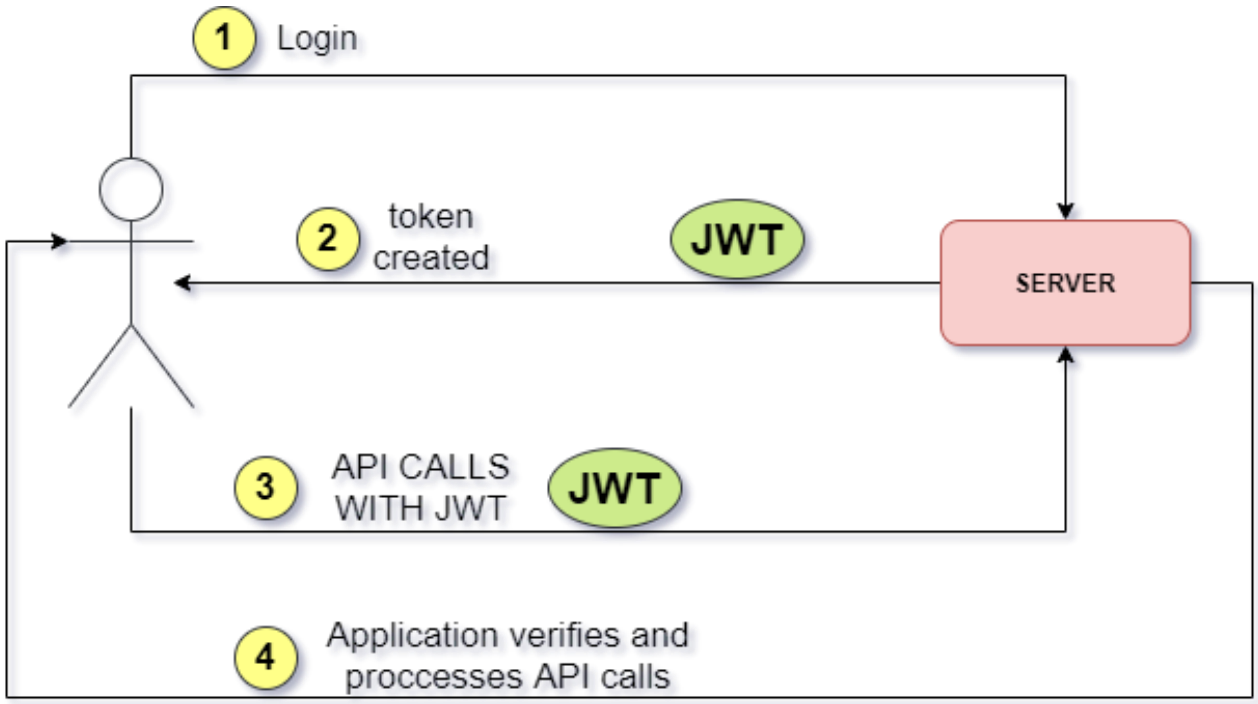

Рисунок 6.2 – Схема авторизації з використанням jsonwebtoken

6.1.2 Клієнтська частина

Структура клієнтської частини на рисунку 6.3:

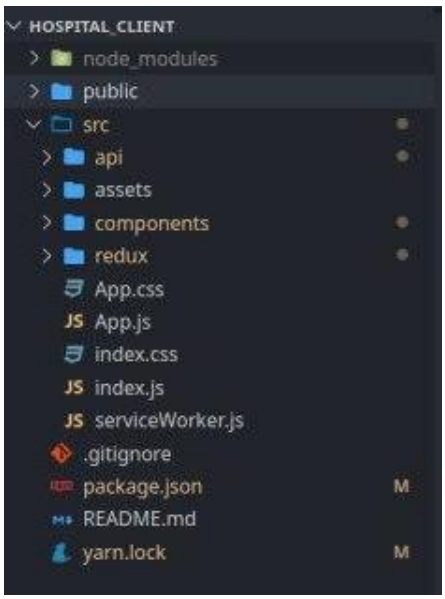

Рисунок 6.3 – Структура клієнтської частини

Структура клієнтської складається з наступних елементів:

– node_modules – тут знаходяться усі необхідні додаткові бібліотеки та залежності;

– public – папка у якій будуть знаходитися усі вихідні файли;

– api – тут зручно зберігаються усі запити на сервер;

– assets – місце зберігання допоміжних файлів, таких як зображення, шрифти, тощо;

– components – у цій теці знаходяться усі компоненти проєкту, які описують web-інтерфейс;

– redux – місце збереження файлів, які керують станом клієнтського застосунку;

– App.css – основний файл стилів;

– App.js – головний компонент застосунку;

– index.js – файл який зв’язує компонент App.js зі сховищем даних Redux та деякими допоміжними засобами;

– package.json – файл, який полегшує налаштування та керування додатковими пакетами застосунку;

– yarn.lock – файл, який містить інформацію про версії додатково встановлених пакетів, що допомагає уникнути конфліктів версій на різних пристроях.

Клієнтська частина функціонує наступним чином: клієнт змінює якісь дані, викликається функція action, яка в свою чергу викликає відповідну функцію reducer. Функція reducer модифікує дані, оновлює стан застосунку та відправляє ці дані на сервер, а також оновлює стан компоненту, який було отримано. Отримавши дані у відповідь, ці дані зберігаються у загальне сховище redux-store, а далі передаються у контейнерні та презентаційні компоненти react у вигляді стану компоненту [31]. Загальну схему роботи клієнтського застосунку наведено на рисунку 6.4:

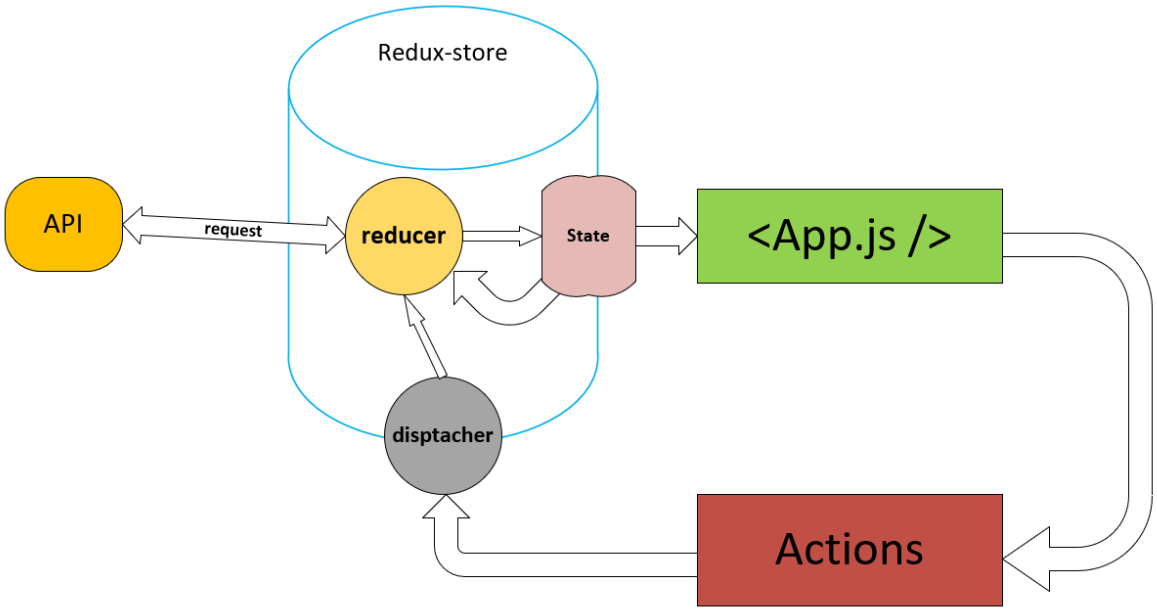

Рисунок 6.4 – Схема роботи клієнтської частини

6.2 Структура бази даних

Також варто окремо розглянути структуру бази даних та її схему.

Для взаємодії серверної частини з базою даних використовується ORM (object-relation mapping) бібліотека Objection.js. Слід зазначити, що окрім таблиць, які створює розробник, бібліотека Objection.js створює службові таблиці.

У кресленику ІТ61.130БАК.004 Д3 наведено ER-Діаграму бази даних, а у таблиці 6.7 наведено основні таблиці, їх опис та призначення.

Таблиця 6.7 – Таблиці бази даних.

Таблиця	Опис
doctors	Таблиця містить дані про лікарів, такі як ідентифікатор (id), прізвище (lastName), ім’я (firstName), по-батькові (fatherName), ідентифікатор спеціальності (specialtyId), ідентифікатор ролі (roleId), номер кабінету лікаря (room), контакти (contacts), електронну пошту (email) та пароль (password), який зберігається у захешованому вигляді.
patients	Таблиця містить інформацію про пацієнтів, таку як ідентифікатор (id), прізвище (lastName), ім’я (firstName), по-батькові (fatherName), дата народження (dateOfBirth), контакти (contacts), ідентифікатор ролі (roleId), електронну пошту (email) та пароль (password), який зберігається у захешованому вигляді.
diagnoses	У цій таблиці розташована інформація про діагнози, містить такі колонки: ідентифікатор (id), назва (title), опис (description), ідентифікатор пацієнта (patientId).
prescriptions	Ця таблиця зберігає дані про припис, такі як: ідентифікатор (id), назва препарату (pharmacyName), частоту прийому (intakeFrequency), тривалість прийому (intakeDuration), ідентифікатор пацієнта (patientId).

Продовження таблиці 6.7

analysis	У таблиці знаходиться інформація про аналізи пацієнтів, а саме: ідентифікатор (id), назва (title), опис (description), результат (result), ідентифікатор пацієнта (patientId).
appointments	Це таблиця, яка утворює зв'язок, а також містить у собі інформацію про прийоми. У таблиці існують такі колонки: ідентифікатор (id), дата (date), час (time), ідентифікатор пацієнта (patientId), ідентифікатор лікаря (doctorId).
roles	У таблиці колонками ідентифікатора (id) та імені (name) описані ролі користувачів.
specialties	Таблиця містить інформацію про спеціальності лікарів, а саме: ідентифікатор (id), назву (name), ідентифікатор лікаря (doctorId).

6.3 Опис роботи системи

У кресленику ІТ61.130БАК.004 Д4 зображено діаграму компонентів.

На діаграмі наочно відображено, що система розроблена з використанням клієнт-серверної архітектури. Клієнтом вважається web-браузер користувача. Серверна частина містить у собі містить API (Application Programming Interface), який було розроблено згідно з архітектурою REST (Representational State Transfer), а також систему керування базами даних PostgreSQL.

Опис роботи системи можна описати через діаграму послідовності, яка наведена у кресленику ІТ61.130БАК.004 Д5.

Діаграми послідовності використовують для того, щоб показати життєвий цикл системи та взаємодію акторів системи в рамках якого-небудь визначеного сценарію.

6.4 Висновки до розділу

У даному розділі було розглянуто структуру та наведено схему роботи клієнтської частини, розробленої за допомогою бібліотек React.js і Redux. Також було описано роботу серверної частини застосунку для обліку пацієнтів у лікарні, яка зроблена на платформі Node.js з використанням web-фреймворку Express.js, а також наведено вичерпний опис моделей та контролерів серверної частини, розроблено схему роботи серверу. Крім того, було описано структуру бази даних застосунку, розроблено ER-діаграму бази даних.

7 ВПРОВАДЖЕННЯ ТА ВИКОРИСТАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

7.1 Системні вимоги

Клієнт-серверний застосунок для обліку пацієнтів у лікарні було розроблено на платформі Node.js, а отже системні вимоги до серверу ідентичні вимогам Node.js. Також слід враховувати вимоги до системи керування базами даних PostgreSQL.

У таблиці 7.1 наведено мінімальні системні вимоги, а у таблиці 7.2 також описані оптимальні системні вимоги.

Таблиця 7.1 – Мінімальні системні вимоги

Процесор	2-ядерний процесор з тактовою частотою не менш ніж 1.5 ГГц.
Дисковий простір	Не менш ніж 2 гігабайти
Оперативна пам’ять	Не менше 2 гігабайт
Операційна система	64-розрядна, Windows 8.1 та вище, Linux-системи або MacOS

Таблиця 7.2 – оптимальні системні вимоги

Процесор	2-ядерний процесор з тактовою частотою не менш ніж 2.5 ГГц.
Дисковий простір	Не менш ніж 4 гігабайти
Оперативна пам’ять	Не менше 6 гігабайт

Продовження таблиці 7.2

Операційна система	64-розрядна, Windows 10, Debian або Red Hat Enterprise Linux.

Слід пояснити, що клієнт-серверний застосунок для обліку пацієнтів працює з великою кількістю даних, тому слід обирати якомога більший об’єм дискового простору.

Також, не можна не зазначити, що для сервера, у якості операційної системи, найкращим вибором буде Linux-дистрибутив, тому що Linux є більш ефективним, більш безпечним та надійнішим. Більш того, Linux є найпопулярнішою операційною системою на сервері, що спрощує пошук людини, яка зможе обслуговувати сервер. Найбільш використовувані дистрибутиви на сервері це Debian та Red Hat Enterprise Linux [32]. Виходячи з цих зауважень, рекомендується використовувати саме їх.

Застосунок для обліку пацієнтів використовує платформу Node.js версії 14.3.0, тому слід використовувати саме цю версію платформи, або ж новішу версію.

Також враховуючи, що при розробці було використано web-фреймворк Express.js версії 4.17.1, рекомендується використовувати цю версію, тому що у тих, які вийшли раніше відсутня підтримка деяких можливостей актуальної версії фреймворку.

Також для коректного встановлення залежностей має бути встановлено менеджер пакетів Yarn.

У якості клієнта в системі виступає браузер. Клієнтська частина розроблена за допомогою мови JavaScript та її бібліотек, а отже вимоги до клієнтського застосунку ідентичні з версіями браузера, у яких працюють технології, які були використані при розробці.

Було проведено аналіз сумісності бібліотек з версіями найпопулярніших браузерів, а саме Google Chrome, Safari, Internet Explorer, Microsoft Edge, Android Browser, Opera. Вимоги до браузерів наведено у таблиці 6.3.

Через те, що було використано останню специфікацію мови JavaScript, а саме EcmaScript 2019, деякі застарілі браузери не мають підтримки бібліотек React та Redux. Але цю проблему було вирішено, використавши JavaScript-транспілятор Babel, який перетворює код з найновішою специфікацією ES2019, у специфікацію ES5, яка сумісна з більшістю старих версій браузерів.

Таблиця 7.3 – Вимоги до браузера для коректної роботи клієнтського застосунку

Назва продукту	Мінімальна версія
Google Chrome	Частково з версії 2, повна підтримка з версії 21 та вище
Safari	Частково з версії 3.1, повна підтримка з версії 5 та вище
Internet Explorer	Частково з версії 9, повна підтримка з версії 10 та вище
Microsoft Edge	12 або новіша
Android Browser	Частково з версії 2.1, повна підтримка з версії 4.4 та вище
Opera	Частково з версії 10, повна підтримка з версії 15 та вище

7.2 Інструкція по розгортанню системи

Для розгортання системи необхідно відкрити термінал або командний рядок, попередньо встановивши пакетний менеджер Yarn, а також платформу Node.js. У

терміналі треба перейти у директорію з проєктом та виконати команду «yarn install». Завдяки цьому пакетний менеджер Yarn встановить усі необхідні залежності та відповідні їм версії, які потрібні для роботи клієнт-серверного застосунку для обліку пацієнтів. Список залежностей можна передивитися у файлах package.yarn або yarn.lock.

Для запуску сервера потрібно перейти у директорію з серверною частиною та ввести команду «yarn server», яка запустить сервер Express.js.

Для запуску клієнтського застосунку потрібно перейти у відповідну директорію та ввести команду «yarn install».

Систему керування базами даних PostgreSQL має бути також обов’язково запущено.

7.3 Висновки до розділу

Враховуючи факт створення застосунку на базі платформи Node.js, вимоги до серверу є ідентичними до вимог самої платформи, а також мають враховувати вимоги системи керування базами даних PostgreSQL. У розділі було наведено мінімальні та оптимальні системні вимоги, а також рекомендований об’єм дискового простору.

Для сервера, у якості операційної системи, найбільш оптимальним вибором буде Linux-дистрибутив, зважаючи на його ефективність безпечність та надійність у використанні. Окремою перевагою при виборі Linux-дистрибутиву можна вважати полегшений пошук людини, що займається обслуговуванням сервера.

Крім того, у розділі наведено вимоги до клієнтської частини. Окремою роботою, виконаною при написанні розділу, можна вважати аналіз сумісності бібліотек з версіями найпопулярніших браузерів.

Також у розділі наведена детальна покрокова інструкція по розгортанню системи.

ВИСНОВКИ

Взявши до уваги прагнення України до переходу на систему електронної документації та вивчивши усі галузі, в яких планується застосовувати таку систему, метою дипломного проєкту стало створення клієнт-серверного застосунку для обліку пацієнтів у лікарні.

Розробка такого застосунку має високу актуальність у сучасному світі. Головною задачею розробки даного продукту можна вважати прагнення сучасної медичної системи до модернізації, а також переведення документообігу лікарень в електронний варіант. В загальному, головними цілями розробки є полегшення роботи лікарів, зростання ефективності та швидкості медичного обслуговування та привернення уваги розробників до цієї галузі.

Перш за все, був проведений огляд та аналіз існуючих рішень, а саме: сучасної електронної системи «Helsi», системи онлайн-запису до лікарів «Поліклініка без черг», медичної інформаційної системи «Доктор Элекс», медичної інформаційної системи «ASKEP» та сучасної телемедичної системи «Медікіт». В процесі аналізу існуючих рішень було виокремлено переваги та недоліки кожної з систем. За результатами роботи було визначено, якими саме властивостями має володіти готовий застосунок та які функції мають обов'язково бути створені у ньому.

Окремою зупинкою на шляху створення застосунку був аналіз вимог до програмного забезпечення. У процесі цього було створено Use Case діаграму, що чітко відображає варіанти використання готового застосунку та окремих його частин. Крім того, були створені таблиці, що вміщують у собі детальний опис кожного зі сценаріїв використання системи. Наприкінці роботи був проведений складний процес розробки функціональних та нефункціональних вимог до системи.

Наступним кроком був вибір технологій розробки. Для цього було проведено детальний огляд та аналіз найпопулярніших засобів та технологій, що

використовуються у сучасній розробці клієнт-серверних застосунків. Враховуючи завдання та призначення застосунку, для його розробки було обрано платформу Node.js у поєднанні з web-фреймворком Express.js у якості серверної частини та бібліотеку React.js у зв'язці з менеджером станів застосунку Redux. В якості системи керування базами даних використано СКБД PostgreSQL. Крім того, було застосовано деяку кількість бібліотек, завданням яких є полегшення розробки користувацьких інтерфейсів та деяких інших опцій.

Реалізація програмного забезпечення клієнт-серверного застосунку для обліку пацієнтів у лікарні включала в себе створення і розробку двох окремих проєктів для серверної та клієнтської частини відповідно. Крім того, було створено структуру бази даних для розроблюваного застосунку. Окремо було виконано детальний опис роботи системи.

Наостанок було наведено системні вимоги до серверної та клієнтської частин та надана чітка послідовна інструкція з розгортання системи.

Результатами роботи стало створення функціонуючого клієнт-серверного застосунку для обліку пацієнтів у лікарні. Готовий застосунок відповідає усім вимогам, що були поставлені у процесі розробки. Він оснащений необхідними функціями для лікарів, керівників медичних закладів та пацієнтів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шпак Г.В. Аналіз забезпечення галузі охорони здоров’я лікарями / Шпак Г.В., Волинкін І.І. // Науково-практичне видання. Україна. Здоров’я нації. – К.: «Експерт», 2011. – №2(18). – С.121-125.
2. Top Computer Languages 2019 - StatisticsTimes.com [Електронний ресурс] – Режим доступу (дата звернення: 8.04.2020): http://statisticstimes.com/tech/top-computer-languages.php
3. Клиент — сервер — Википедия [Електронний ресурс] – Режим доступу (дата звернення: 9.04.2020): https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_%E2%80%94_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80
4. HELSI - інформаційна система для пацієнтів [Електронний ресурс] – Режим доступу (дата звернення: 10.04.2020): https://helsi.me/
5. Поліклініка без черг [Електронний ресурс] – Режим доступу (дата звернення: 10.04.2020): https://global.newmedicine.com.ua/
6. Почему вам НЕ стоит использовать AngularJs / Хабр – Habr [Електронний ресурс] – Режим доступу (дата звернення: 11.04.2020):: https://habr.com/ru/post/246905/
7. Архітектура системи | Dr. ELEKS - Doctor Eleks [Електронний ресурс] – Режим доступу (дата звернення: 12.04.2020): https://doctor.eleks.com/product/system-architecture/
8. Медицинская Информационная Система - Askep.net [Електронний ресурс] – Режим доступу (дата звернення: 14.04.2020): https://askep.net/ru
9. Медікіт | Здоров'я у Ваших руках [Електронний ресурс] – Режим доступу (дата звернення: 15.04.2020): https://medikit.ua/
10. Как и зачем писать Use Cases | DOU [Електронний ресурс] – Режим доступу (дата звернення: 17.04.2020): https://dou.ua/lenta/articles/use-cases/

11. Пример написания функциональных требований к Enterprise-системе – Habr [Електронний ресурс] – Режим доступу (дата звернення: 18.04.2020): https://habr.com/ru/post/245625/

12. Нефункциональные требования к программному обеспечению. Часть 1 [Електронний ресурс] – Режим доступу (дата звернення: 20.04.2020): https://habr.com/ru/post/231961/

13. ТОП-10 фреймворков для веб-разработки в 2019 [Електронний ресурс] – Режим доступу (дата звернення: 21.04.2020): https://proglib.io/p/web-frameworks-2019/

14. Front-end разработка - wiki студи Клондайк [Електронний ресурс] – Режим доступу (дата звернення: 21.04.2020) https://klondike-studio.ru/wiki/front-end/https://proglib.io/p/web-frameworks-2019/

15. JavaScript | MDN – офіційна документація про відкриті web-технології [Електронний ресурс] – Режим доступу (дата звернення: 22.04.2020): https://developer.mozilla.org/ru/docs/Web/JavaScript

16. Веб-фреймворк Express (Node.js/JavaScript) | MDN – офіційна документація про відкриті web-технології [Електронний ресурс] – Режим доступу (дата звернення: 22.04.2020): https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs

17. Objection.js - ORM for Node.js – офіційна документація ORM Objection.js [Електронний ресурс] – Режим доступу (дата звернення: 22.04.2020): https://vincit.github.io/objection.js/

18. Implementing JSON Web Tokens & Passport.js in a JavaScript Application with React [Електронний ресурс] – Режим доступу (дата звернення: 22.04.2020): https://itnext.io/implementing-json-web-tokens-passport-js-in-a-javascript-application-with-react-b86b1f313436

19. React – A JavaScript library for building user interfaces – офіційна документація бібліотеки React.js [Електронний ресурс] – Режим доступу (дата звернення: 23.04.2020): https://reactjs.org/

20. Краткое руководство по Redux для начинающих React [Електронний ресурс] – Режим доступу (дата звернення: 23.04.2020): https://tproger.ru/translations /redux-for-beginners/

21. Bootstrap. Документация на русском языке. – офіційна російськомовна документація бібліотеки Bootstrap [Електронний ресурс] – Режим доступу (дата звернення: 23.04.2020): https://bootstrap-4.ru/

22. Material-UI: A popular React UI framework – офіційна документація фреймворку Material-UI [Електронний ресурс] – Режим доступу (дата звернення: 23.04.2020): https://material-ui.com/

23. Promise based HTTP client for the browser and node.js – офіційна документація бібліотеки axios [Електронний ресурс] – Режим доступу (дата звернення: 24.04.2020): https://github.com/axios/axios

24. Основы Git – Git – ресурс для ознайомлення з основами системи контролю версій git [Електронний ресурс] – Режим доступу (дата звернення: 24.04.2020): https://git- scm.com/book/ru/v2/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-Git

25. GitHub — отзывы реальных пользователей | Startpack [Електронний ресурс] – Режим доступу (дата звернення: 24.04.2020): https://startpack.ru/application/github/reviews

26. PostgreSQL - METANIT.COM – ресурс для вивчення різноманітних технологій програмування [Електронний ресурс] – Режим доступу (дата звернення: 24.04.2020): https://metanit.com/sql/postgresql/

27. Версионная миграция структуры базы данных: основные подходы [Електронний ресурс] – Режим доступу (дата звернення: 26.04.2020): https://habr.com/ru/post/121265/

28. Игнорирование файлов в Git - Основы Git – Hexlet – ресурс для вивчення різноманітних технологій програмування [Електронний ресурс]

– Режим доступу (дата звернення: 28.04.2020): https://ru.hexlet.io/courses/git_base/lessons/git_gitignore/theory_unit

29. Список кодов состояния HTTP — Википедия [Електронний ресурс] – Режим доступу (дата звернення: 28.04.2020): https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP

30. What is REST – Learn to create timeless REST APIs – [Електронний ресурс] – Режим доступу (дата звернення: 10.05.2020): https://restfulapi.net/

31. Редакс в реальной жизни — доклад Ивана Акулова [Електронний ресурс] – Режим доступу (дата звернення: 14.05.2020): https://iamakulov.com/talks/redux-in-real-life/

32. Best Linux server distro of 2020 | TechRadar [Електронний ресурс] – Режим доступу (дата звернення: 20.05.2020): https://www.techradar.com/best/best-linux-server-distro/