МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри

_____ Є. О. Давиденко
підпис
« »_____2022р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Хмарний вебсервіс міської лікарні

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408.21810814

Студент _____ М. Д. Забєлєнков
підпис
«__»_____2022р.

Керівник доцент кафедри ІПЗ, д-р. техн. наук _____ А. В. Швед
підпис
«__»_____2022р

Консультант канд. техн. наук, доцент _____ А. О. Алексеєва
підпис
«__»_____2022р.

Миколаїв – 2022

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ІС – інформаційна система

МІС – медична інформаційна система

ПК – персональний комп’ютер

Юзер – користувач

СКБД – система керування базами даних

SQL - structured query language

CSS – Cascading Style Sheets

HTML – HyperText Markup Language

ВСТУП

Інформаційні технології у медичній сфері є доволі важливим елементом розвитку країни. Наразі до медичним закладам надається велика кількість вимог в області діяльності, фінансування, а також з боку простих пацієнтів та власне працівників медичних закладів. Велика кількість проблем має доволі прості методи вирішування, а саме використання інформаційних технологій. Але на даний момент існує багато складнощів щодо впровадження потрібних технологій. Усі проблеми формують потребу у створенні відповідної системи інформаційного забезпечення лікарів і пацієнтів. Що стосується саме практичного використання системи, то це може вирішити питання, яке зумовлене великими обсягами даних, які зростають з кожним днем, а також з безпосередню більш активну участь пацієнта у лікуванні. Але може виникати питання доступності та зручності використання інформації. Будь які дані можуть зберігатися на різноманітних носіях, а саме: на переносних пристроях, локальних сховищах, жорстких дисків або навіть компакт, хмарних сховищах. Хочеться наголосити на останньому пункті, а саме на «хмарах», адже їх використання вирішує будь яке питання зручності та швидкодоступності, дані становляться у доступні з будь якого кутка світу, де є Інтернет. Уся інформація зберігається на віддаленому сервері, а користувач може працювати з ними ніби вони завантажені в нього на пристрої. Є різновиди хмарних сервісів, але усі вони мають один сенс – зберігати дані та надавати віддалений доступ до них. Існують публічні, приватні та гібридні хмарні сервіси. Публічні сервіси обмежують управління хмарою та мають невеликий безкоштовний обсяг. Приватні ж навпаки, надають повний контроль управління хмарою, та дозволяють регулювати обсяги і інші параметри. Гібридні ж зливають в одне ціле приватні та публічні хмари, це робиться з метою стабілізації навантажень на сховище, за рахунок публічного сховища.

Об’єкт роботи: процеси управління інформацією медичного закладу з використанням хмарного сховища даних.

Предмет роботи: інформаційні технології та інструментальні засоби розробки вебсервісів.

Мета і завдання роботи: дослідити можливості спрощення управління процесами збереження та доступу до даних медичного закладу з використанням хмарних сховищ.

Для досягнення мети було визначено перелік наступних завдань:

1. Огляд інформаційних систем у медицині;
2. Огляд вже існуючих аналогів систем, призначених для поліпшення роботи лікарень;
3. Дослідити специфікації вимог;
4. Вибір інструментів для реалізації Web-застосунку;
5. Спроектувати та розробити модель застосунку;
6. Програма реалізація комплексу Web для вебсервісу;
7. Тестування комплексу Web для вебсервісу.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ АВТОМАТИЗАЦІЇ ВЕДЕННЯ ОБЛІКУ МЕДИЧНИХ ПОСЛУГ ТА УПРАВЛІННЯ МЕДИЧНОЮ ІНФОРМАЦІЄЮ

1.1 Опис предметної сфери автоматизації та управлінського обліку медичних закладів

Інформаційні процеси (пошук, збирання, зберігання, передавання, опрацювання, використання, захист інформації) займають своє місце у всіх галузях медицина та охорони здоров’я. Однією з не менш важливих частин інформаційних процесів є інформаційні потоки. Від їх реалізації та структури залежить коректність функціонування певної галузі і ефективність керування нею. Усі інформаційні потоки починаються з тих місць, у яких виникає різноманітна інформація, й забезпечують переміщення інформації до головних місць прийняття рішень. Усі вони мають свою певну структуру, яка відображається у вигляді окремих повідомлень, відображених у сигналах та документах, вони переміщуються у просторі та часі від головного джерела та до самого отримувача. Але задля роботи з цими важливими елементами призначені інформаційні системи (ІС) [5].

Інформаційною системою (ІС) називається впорядкована та організована сукупність документів або навіть великих масивів, інформаційних автоматизованих технологій, що має можливість реалізовувати необхідні інформаційні процеси. Кожна інформаційна система має певну галузь застосування.

Опрацювання інформації у системі реалізується різноманітними способами, а саме: механічним, ручним, автоматизованим і автоматичними способами. Оскільки комп’ютери вже відіграють не мало важливу роль у нашому житті та впливають на велику кількість галузей та сфер. То можна зазначити, що приблизно 30 років назад розпочалася так звана революція у процесах опрацювання інформації, стали виникати нові інформаційні технології, у тому числі й у медицині та у системах охорони здоров’я. Застосування нових інформаційних технологій у сфері охорони здоров’я, а також у загальній медицині

називають інформатизацією усієї медичної системи [1].

Головне означення інформатизації – це створення необхідного комплекту заходів, задля використання вірних та перевірених знань у будь-якій сфері діяльності людини. Розглядаючи інформатизацію у обраній сфері, а саме у сфері медицини та охорони здоров'я можна зазначити, що головною метою є прогрес у самій системі та у напрямках розвитку служби контролю за станом здоров'я людини. Основою інформатизації з боку технологічного та технічного середовища є потужна мережа інформаційних структур, які у свою чергу орієнтовані як на працівників галузі медицини так і на усе населення.

Управлінський облік – це структурована система, яка виявляє, виміряє, збирає, реєструє, інтерпретує, узагальнює, готує і надає важливу інформацію для ухвалення рішень щодо роботи структурування інформації та усіх показників задля управлінської частини організації.

Однією з головних задач яку вирішує інформаційна система, це управлінський облік. Слід зазначати, що управлінський облік є важливою частиною будь-якої організації. Але важливо розрізняти та не плутати його с бухгалтерським обліком.

Управлінський облік відіграє дуже важливу роль в успіхах розвитку медичного закладу, але для цього дуже важливо його побудова й розвиток. У медичних організаціях управлінський облік виконує такі завдання:

- Робить облік усіх витрат за різноманітними послугами та може підраховувати собівартість, щоб мати інформацію задля обґрунтування цінової політики;
- Контролює грошові кошти, та надає змогу оптимізувати їх використання;
- Управління загальним фінансовим становищем лікарні.

Також будь-яка інформаційна система, яка має у собі управлінський облік зможе долучити увесь персонал медичних закладів до структурних змін, які насамперед будуть відбуватися за допомогою створення науково-практичних груп чи команд. У такій системі найчастіше передбачається створення освітніх комплексів, що може організовувати безперервне покращення навичків персоналу

та надавати додаткову мотивацію.

Аналізуючи сучасні українські тенденції, то налічується понад 15000 медичних закладів. Серед яких більшу частку займають державні установи. Проте процент закладів у які впроваджено інформаційну систему близько 7%. Впровадження інформаційних технологій є доволі актуальним, тому що без використання комп'ютерних технологій складно надавати медичну допомогу на належному рівні, оскільки уся робота лікарів, у сучасному світі, супроводжується великою кількістю інформації.

Перехід на нові інформаційні системи дозволить дуже легко та швидко вести облік усіх послуг які надає установа, аналізів, рецептів. При автоматизації медичної організації заповнюються картки пацієнтів та їх повні історії хвороб, за рахунок цієї інформації складаються необхідні звіти задля відслідковування динаміки пацієнта. Пришвидшення роботи не потребує спеціального обладнання, медичний персонал може використовувати смартфони, планшети та ПК задля надання необхідних послуг.

Автоматизація медичних закладів — це безпосереднє створення єдиного інформаційного простору ЛПУ, який надає можливість надавати спеціальні автоматизовані робочі місця медичних співробітникам, організовувати роботу усіх відділів, у тому числі відділу статистку, створювати БД, працювати з цифровими історіями хвороб та узагальнювати усі лікувальні, фінансові, діагностичні, господарські та адміністративні процеси.

1.2 Класифікація медичних інформаційних систем

Під час класифікації медичних інформаційних систем слід розрізняти різноманітні ознаки, а саме що:

1) Від рівня автоматизації процесів отримання й обробки інформації медичні інформаційні системи ділять на автоматизовані та автоматичні. У першому випадку частина виконуваних дій, а саме збір та обробка інформації проходить саме за допомогою людини. А у другому випадку, усі системи такого типу мають на увазі повне виключення людини з процесу збору та обробки

інформації.

2) Від типу інформаційного багажу медичної інформаційної системи, виділяють певні системи, що оперують інформацією, що оперують знаннями. Слід зазначити, що саме системи, що базуються на знаннях, ще мають назву як експертні. З назви можна зрозуміти, що їх функціонал спирається на досвід та звання експертів, а результати роботи будуть доволі схожі, на аналітичну діяльність експертна.

3) Також МІС можна поділити на певні групи, які залежать від типу завдань. Інформаційно – довідкові системи дозволяють окрім безпосередньо пошуку, має можливість робити перетворення інформації для формування необхідного документу. Інформаційно – логічна система виконує обов'язки щодо роботи з інформацією так, щоб можливо було отримувати одразу нову інформацію, яка у свою чергу відсутня у інформаційній базі. Системи управління відрізняються, тим що реалізують у собі абсолютно нові функції прийняття керуючих рішень. Вони втілюють збір інформації про будь – які об'єкти управління, обробку та передачу даних до верхівок управління, створюють керуючі рішення. Інформаційно – пошукові системи наразі одержують статус найпопулярніших, тому що у залежності від наданої інформації можуть розподілятися на фактографічні і документальні.

Фактографічні системи зберігають у собі великі масиви фактичних даних. Вони насамперед відіграють доволі важливу роль у швидкості роботи медичного закладу, оскільки є прямим аналогом паперових документів, тощо. На зміну застарілим технологіям приходять нові, у системах інформація міститься у базах даних, що прискорює пошук необхідних даних.

Документальні ж системи працюють з усією інформацією у вигляді документів. Якщо розглядати приклади, то можна визначати різноманітні види картотек. Під час пошуку система надає або усі номера необхідних даних, або повний список назв. Але ця система не є автоматичною, оскільки оцінку отриманих даних дає людина.

4) Також є можливість класифікації МІС і за ієрархічною структурою,

що є прямою відповідністю багаторівневій структури галузі медицини. Існують чотири головних рівні у межах яких класифікація реалізується за функціональним принципом [2].

Системи базового рівня представлені лише на клінічному рівні. Вони призначені задля забезпечення інформацією усіх медичних співробітників підприємства. Найбільш ефективні ці системи задля профілактичної і діагностичної роботи, в умовах дефіциту часу або великих обсягів роботи. У свою чергу спираючись на ієрархічний принцип, можна розподілити, ще на 4 групи. Інформаційно – довідкові системи, їх головне визначення це пошук та видача усієї необхідної інформації за запитом користувача. Консультативно – діагностичні системи надають можливості для діагностики станів пацієнта при захворюваннях різних ступенів важкості. Приладо – комп'ютерні системи підтримують та повністю автоматизують діагностичний та лікувальний процес, який робиться з контактом до пацієнта. А ось системи автоматизованого робочого місця (АРМ) лікаря значно допомагають для загальної автоматизації роботи лікаря. Насамперед це ведення медичної документації, розробка лікувальних планів, отримання довідкових зазначень. Також система забезпечує загальну інформаційну підтримку, щодо прийнятих рішень.

Наступним рівнем інформаційної системи є рівень лікувально – профілактичного закладу.

У свою чергу вони містять у собі медичній інформаційні системи, що спрямовані задля повного забезпечення роботи підрозділів й інформації для лікарів під час консультування, тощо.

Існують також банки інформації медичних закладів і дочірніх служб, які зберігають дані про кадровий склад підприємства, статистику й інші необхідні дані.

Персоніфіковані реєстри – це один з різновидів систем, що містять у собі великі обсяги інформації. Реєстри надають усім лікарям швидко отримувати необхідну інформацію про пацієнта, відслідковувати динаміку його стану, а також аналізувати якість наданих заходів.

Скринінгові системи забезпечують профілактичний долікарський огляд, який впливає на формування груп ризику, що потребують більш пильного догляду.

У підсумку МІС даного рівня засновані на структуруванні усіх інформаційних потоків у єдину велику систему, а також надають загальну автоматизацію різноманітних видів діяльності. Уся інформація яка виходе з таких систем, є вирішенням головних завдань, задля керування установи, так і для вирішення завдань для систем вищих рівнів.

Останніми рівнями МІС територіальні та державні програмні комплекси. Вони надають керування спеціальними та профільними службами.

Територіальні системи містять у собі адміністративні та управлінські системи, які допомагають вирішувати організаційні задачі керівникам служб, а також іншим медичним фахівцям. Статистичні системи, виконують набір функцій які допомагають отримувати скомпоновану інформацію за основними показниками. Існують також системи які, допомагають вирішувати медико-технічні задачі, які насамперед підтримують інформаційну діяльність цілого підприємства. Застосування телекомунікаційних медичних мереж, забезпечують створення загального простору з інформацією на рівні певного регіону.

ІС серед державного рівня, мають різноманітні типи та свої особливості. ІС які належать до державних органів здійснюють велику підтримку у організації управління міністерством. Також забезпечують роботу комплексу завдань задля керування галуззю, що насамперед покращує обробку та залучення ресурсів медичних служб. У свою ж чергу статистичні МІС, роблять пошук та обробку усієї інформації та узагальнює її за необхідними показниками. Спеціальні медико – технічні ІС надають можливості вирішення завдань інформаційної підтримки співробітників установи, а також спеціалізованих служб. До ряду ІС входять спеціальні інформаційні системи для особливих напрямків. Розрізняють також і галузеві МІС, які у свою чергу підтримують та оптимізують роботу галузевих служб. Телекомунікаційні медичні мережі, надають змогу розробки та впровадження загального інформаційного простору на рівні держави.

1.3 Аналіз та порівняльна характеристика сучасних медичних інформаційних систем

Оскільки головним пріоритетом у діяльності установ охорони здоров'я є покращення рівня доступності та якості медичної допомоги. Спираючись на досвід використання МІС, їх впровадження дозволяє розширити ефективність та якість послуг медичної системи країни.

Узагалі ідея впровадження МІС містить у собі демократизацію процесів створення та використання інформації, захист інформації, тощо. Головною метою державних діячів у сфері охорони здоров'я є розвиток інформатизації. Наразі було реалізовано комплекс задля розвитку сімейної медицини. Але існують й інші МІС.

Аналізуючи ринок України у сфері медичних інформаційних систем визначають дві головні системи документообігу та процесів. Головними відмінностями є, що якщо головним предметом користувача є хроніка захворювань або інші акти, то це є типовою системою документообігу. А під час використання ІС коли користувач керує лікувальним процесом, їх статусами, то система є медичною.

Типи ІС впливають на підсумки їх впровадження. Адже системи документообігу вирішують одну з найголовніших проблем, зменшення паперових документів, за допомогою копіювання. Вирішує проблему промахів під час оформлення, а також загально типізує їх. А ось процесні системи використовуються задля економії ресурсів та задля вдосконалення медичного обслуговування за рахунок збільшення КПД медичних співробітників.

Також існує й інша класифікація МІС та вона зв'язана з галуззю. Radiology Information System (RIS) надають можливість структурування медичних зображень, а також надає доступ до них. Enterprise Resource Planning (ERP) надають можливість організації керування економічними, трудовими, а також матеріальними ресурсами установи. Electronic Medical Records (EMR) надають можливість керування медичними записами.

Сучасні інформаційні системи мають вирішувати доволі важливі завдання. Одне із завдань є централізація даних. Це означає, що уся інформація яка була внесена до системи, доступна з будь – якого місця. У результаті вирішення завдання можна отримати оптимізацію роботи медичних закладів. Другим на мало важним завданням є типізація даних, як для інформації так і для усіх процесів. Третім завданням є надання легкого доступу до інформації та її обробки. Інформація, що є загально типізованою та централізованою доступна задля обробки та її аналізу. Це є одна з найголовніших переваг МІС.

Спираючись на усю наведену інформацію було проведено порівняльний аналіз медичних систем в Україні.

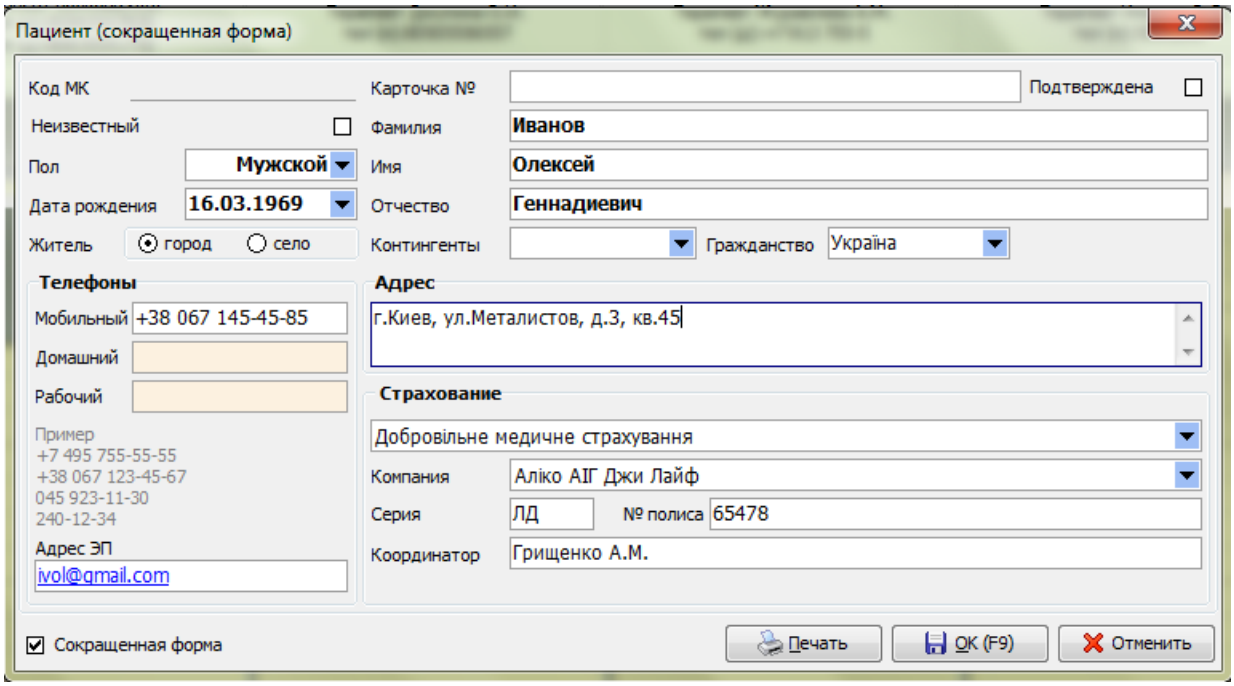

Рисунок 1.1 – Реєстрація пацієнта у системі ЕМСІМЕД

Медична система ЕМСІМЕД – направлена на автоматизацію процесів, що є відбуваються у сучасних лікувально-профілактичних закладах. Система надає можливість проводити запис на прийом до лікаря, вести медичні картки, робити різноманітні касові операції, а також використовувати графік роботи лікарів. У результаті використання системи зберігається електрона картка пацієнта. Однією з головних особливостей системи є доволі унікальна функція відслідковування посад медичних співробітників та їх кар'єрний зріст [3].

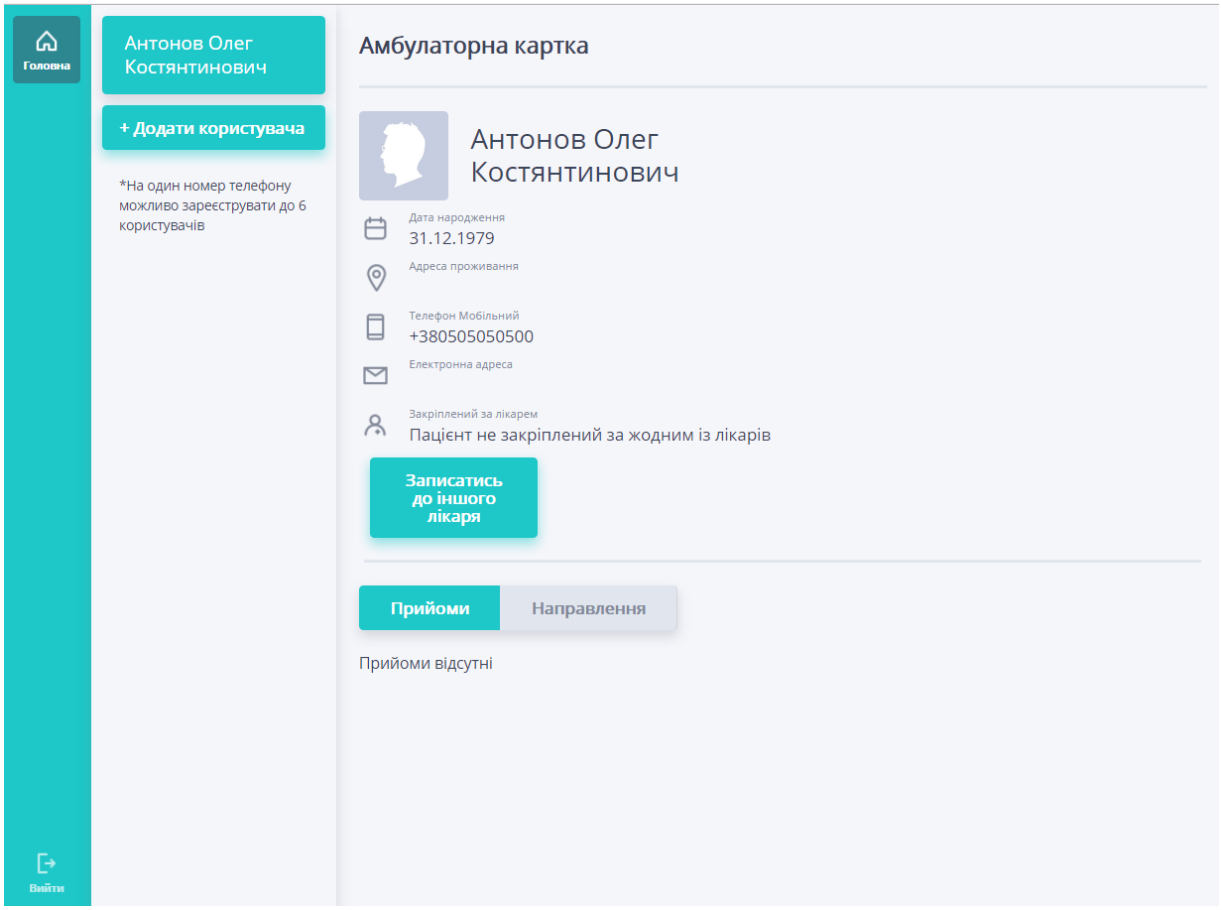

Рисунок 1.2 – Картка пацієнта у системі

Поліклініка без черг – це система керування пацієнтами, яка надає можливість орієнтуватися на графік роботи лікарів та дозволяє записатися на прийом онлайн. Запис можливо зробити не тільки до свого лікаря, а до усіх лікарів у закладі. Застосунок має простий та інтуїтивно зрозумілий дизайн. Щоб записатися на прийом лише потрібно натиснути кнопку запису до лікаря, а після цього вже підтвердити запис за мобільним телефоном. Але навіть розглядаючи систему слід зазначити, що вона має доволі малий функціонал для користувача. Наприклад, пацієнт не може переглянути власну електрону картку. Також система не дозволяє відслідковувати динаміку проходження хвороби, за рахунок того, що не має можливості внести симптоми чи загальний стан пацієнта. Оскільки ця система є доволі популярною на території України, то й розробка вебсервісу буде проходити з врахуванням недоліків системи.

Medakadem. Clinica – є продуктом задля автоматизації діяльності насамперед приватних закладів, який надає можливість прискорення і оптимізації

роботи. Має доволі сучасний дизайн, але поступається конкурентам у певних функціях. Є можливість запису на прийом та внесення переліку послуг, за які пацієнт повинен сплатити гроші. Містять аналітичні та статистичні засоби, що дають змогу використовувати інформацію про кількість пацієнтів, що були у закладі більше декількох разів. Створює рейтинг співробітників за часом, який потрібен на виконання певних завдань [6].

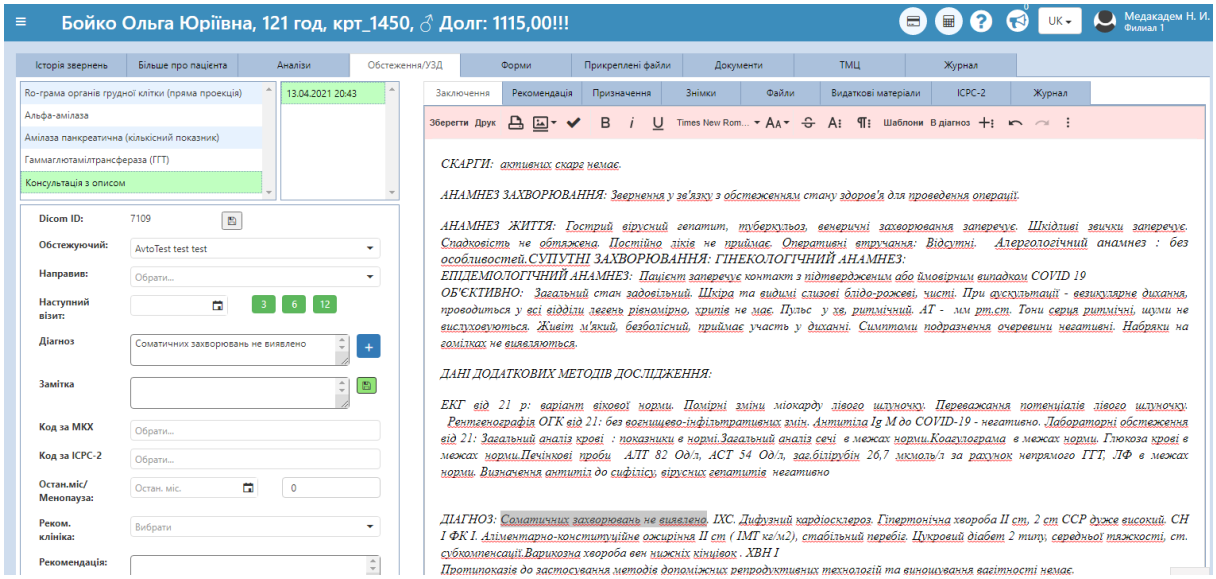

Рисунок 1.3 – Медичні записи у системі Medakadem. Clinica

Таблиця 1.1 – Порівняння аналогічних систем

Критерії	ЕМСІМЕД	Поліклініка без черг	Medakadem. Clinica
Запис пацієнтів	Присутній	Присутній	Присутній
Ведення медичної картки	Присутній	Не повний	Відсутній
Запис на прийом	Присутній	Присутній	Відсутній
Робота з розкладом	Присутній	Присутній	Відсутній
Управління послугами	Присутній	Відсутній	Присутній
СКБД	На базі керуючих систем MS SQL	На базі SQL	На базі Mongo

Усі наведені вище МІС мають досі великі недоліки, адже нестача самих цих функцій значно б поліпшила роботу системи. Динамічні дані та їх оновлення у

режимі реального часу значно покращило б роботи усіх МІС. Також обробка редагування у режимі реального часу вирішила б проблему втрати певної інформації. Також впровадження нелімітованих списків прискорило б пошук даних, завдяки структуруванні цих же даних.

1.4 Специфікація вимог до програмного забезпечення вебсервісу міської лікарні

Специфікація вимог являє собою певний документ, що містить у собі набір вимог до продукту. Задля опису вимог використовуються use cases. У сценаріях же користувача представляється варіант того, як він може працювати з програмним забезпеченням.

Для досягнення вдалої розробки будь-якої системи потрібно чітко визначати вимоги. Перш за все потрібно визначити головне призначення системи, задля розуміння для чого узагалі це треба. Головним призначенням розробки задуманої системи є виконання вебсервісу для міської лікарні [7].

Визначення ролей застосунку важливий процес розробки, це надає можливість логічного формування інших вимог. Розглядаючи розроблювану систему, можна відзначити чотири головні ролі, а саме: адміністратор, пацієнт, лікар, лаборант. Кожна роль виконує свої власні функції. Адміністратор відповідає за додавання лікарів до системи, а також різноманітних медичних рекомендацій. Лікар та пацієнт мають пряму взаємодію, оскільки лікар може додавати пацієнта до системи та безпосередньо працювати з його даними. Роль лаборант відповідає за додавання результатів аналізів пацієнта, він дає можливість зробити процес роботи лікарні автоматизованим .

Головною вимогою до роботи з вебсервісом є доступ до інтернету.

Функціональні вимоги системи управління лікарнями містять різноманітні процесу, а саме реєстрацію, виписку, генерацію звітів та базу даних. Розглянемо список вимог більш докладно:

– Додавання пацієнтів. Дозволяє додавати пацієнтів безпосередньо у реєстратурі, що зможе пришвидшити процес роботи;

– Зберігати інформацію, щодо медичних рекомендацій. Можливість перегляду та опрацювання інформації;

– Зберігати інформацію про ліки, їх дозування, тощо. Можливість переглядати інформацію про ліки та їх дозування у разі нагальної потреби;

– Створення та зберігання електронної медичної картки пацієнта. Надає можливість швидко оперувати даними пацієнта задля автоматизації процесу роботи;

– Оновлення інформації пацієнта. Мати можливість оновлювати інформацію про пацієнта у режимі реального часу;

– Зберігання даних медичних досліджень;

– Зберігати у власних кабінетах направлення на здачу аналізів, тощо.

Висновки до розділу 1

Проаналізовано предметну сферу у якій будується вебсервіс. Знання у цій сфері надають повне розуміння усієї картини. Чітке уявлення про системи дає можливості розробки власної інформаційної системи.

Крім того, досліджено різноманітні класифікації медичних інформаційних систем. Які дають змогу та можливість орієнтуватися у існуючих медичних інформаційних системах.

Розглянуті аналоги медичних інформаційних систем. І можна визначити те, що не існує однієї ідеальної системи, яка б змогла виконувати усі необхідні функції.

Досліджено головні специфікація щодо вимог у медичних інформаційних системах .Визначено функціональні вимоги застосунку.

2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ХМАРНОГО ВЕБСЕРВІСУ МІСЬКОЇ ЛІКАРНІ

2.1 Діаграма прецедентів

Діаграма прецедентів – це UML-діаграма, яка зображує усі можливі відносини між акторами та прецедентами, також є однією з головних частин моделі, задля опису системи на рівні концепції [8].

Актор – це користувач (людина, система, клас, тощо), що насамперед працює з сутністю.

Прецедент – це частина функціональності системи, за допомогою якої актор отримує результат.

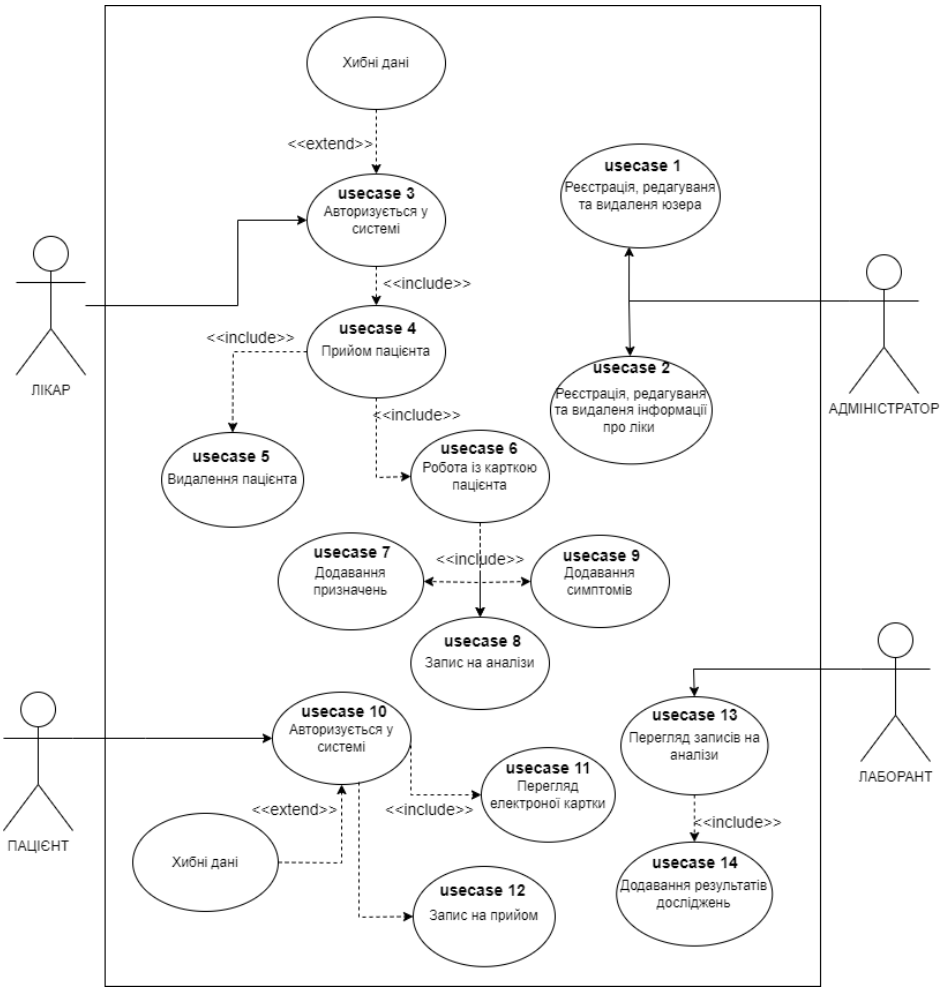

Рисунок 2.1 – Діаграма прецедентів системи [10]

На діаграмі прецедентів зображені можливі сценарії взаємодії з системою.

Далі проаналізуємо кожен сценарій, що був зазначений на діаграмі прецедентів. Наведені в таблицях 2.1 – 2.14.

Таблиця 1.1 - Сценарій використання usecase 01

Назва	Реєстрація, редагування та видалення користувача у системі
Опис	Адміністратор додає, редагує чи видаляє користувача у системі.
Учасники	Адміністратор
Передумови	Адміністратор авторизований у системі.
Постумови	У базі даних створюється, редагується чи видаляється інформація про користувача. Також інформація додається до відповідних таблиць користувачів.
Основний сценарій	Адміністратор за допомогою відповідного меню обирає яку операцію потрібно виконати та здаля якого типу користувача. При цьому при реєстрації заповнюючи усі інформаційні поля. Після чого йде перевірка чи усі поля були заповнені вірно.

Таблиця 2.2 - Сценарій використання usecase 02

Назва	Реєстрація, редагування та видалення інформації про ліки у системі.
Опис	Адміністратор додає, редагує чи видаляє інформації про ліки у системі.
Учасники	Адміністратор
Передумови	Адміністратор авторизований у системі.
Постумови	У базі даних створюється, редагується чи видаляється інформація про ліки.
Основний сценарій	Адміністратор за допомогою відповідної кнопки переходить до форми додавання інформації про ліки, де безпосередньо заповнює її та вносить необхідні дані, а саме: назву та загальну інформацію про препарат.

Наведені сценарії реєстрації користувачів та інформації про ліки.

Таблиця 3.3 - Сценарій використання usecase 03

Назва	Авторизація у системі
Опис	Лікар може увійти до системи за допомогою власних даних.
Учасники	Лікар
Передумови	Користувач існує у базі
Постумови	Перехід до власного кабінету.
Основний сценарій	Користувач натискає кнопку авторизації, де після зазначає власні дані, що потрібні для входу.

Таблиця 4.4 - Сценарій використання usecase 04

Назва	Прийом пацієнта
Опис	Лікар приймає пацієнта
Учасники	Лікар
Передумови	Лікар увійшов до системи
Постумови	Інформацію було додано до бази даних та відображено у списку пацієнтів
Основний сценарій	Лікар виконує усі необхідні процедури щодо пацієнта.

Таблиця 5.5 - Сценарій використання usecase 05

Назва	Видалення пацієнта.
Опис	Лікар може видалити пацієнта з системи.
Учасники	Лікар
Передумови	Лікар увійшов до системи
Постумови	Пацієнта було видалено з системи.
Основний сценарій	Лікар обирає пацієнта та має можливість видалити його з системи повністю.

Розглянуті сценарії авторизація лікаря, а також сценарії прийому та видалення пацієнта з системи. Надані сценарії відіграють важливу роль у повній функціональності системи.

Таблиця 6.6 - Сценарій використання usecase 06

Назва	Робота із карткою пацієнта.
Опис	Лікар має можливість додавати симптоми, призначення та направлення на аналізи.
Учасники	Лікар
Передумови	Лікар увійшов до системи
Постумови	У залежності від обраної операції здійснюється перехід до відповідної форми.
Основний сценарій	Лікар обирає необхідну операцію, що стосується пацієнта та згодом додає інформацію до системи.

Таблиця 7.7 - Сценарій використання usecase 07

Назва	Додавання призначень.
Опис	Лікар додає призначення до картки пацієнта.
Учасники	Лікар
Передумови	Лікар увійшов до системи
Постумови	Інформацію було додано да бази даних та відображено у медичній картці пацієнта.
Основний сценарій	Лікар натискає кнопку ДОДАТИ ПРИЗНЧЕННЯ, де заповнює відповідні поля.

Таблиця 8.8 - Сценарій використання usecase 08

Назва	Запис на аналізи.
Опис	Лікар виписує направлення на аналізи.
Учасники	Лікар
Передумови	Лікар увійшов до системи
Постумови	Інформацію було додано да бази даних та відображено у медичній картці пацієнта.
Основний сценарій	Лікар вносить направлення на аналізи, де заповнює відповідні поля.

Розглянуті сценарії роботи з карткою, запис на аналізи, виписування ліків.

Таблиця 9.9 - Сценарій використання usecase 09

Назва	Додавання симптомів.
Опис	Лікар додає симптоми до картки пацієнта.
Учасники	Лікар
Передумови	Лікар увійшов до системи
Постумови	Інформацію було додано да бази даних та відображено у медичній картці пацієнта.
Основний сценарій	Лікар натискає кнопку ДОДАТИ СИМПТОМИ, де заповнює відповідні поля.

Таблиця 10.10 - Сценарій використання usecase 10

Назва	Авторизація у системі
Опис	Пацієнт може увійти до системи за допомогою власних даних.
Учасники	Пацієнт
Передумови	Користувач існує у базі
Постумови	Перехід до власного кабінету.
Основний сценарій	Користувач натискає кнопку авторизації, де після зазначає власні дані, що потрібні для входу.

Таблиця 11.11 - Сценарій використання usecase 11

Назва	Перегляд електронної картки.
Опис	Пацієнт переглядає власну картку.
Учасники	Пацієнт
Передумови	Пацієнт увійшов до системи
Постумови	Пацієнт переглядає власну інформацію.
Основний сценарій	Пацієнт натискає на кнопку МОЯ КАРТКА та переходить до сторінки електронної картки, де може переглядати усі наявні пункти.

Визначені сценарії додавання симптомів до картки, сценарій авторизації до системи пацієнта, а також перегляд електронної картки

Таблиця 12.12 - Сценарій використання usecase 12

Назва	Запис на прийом.
Опис	Пацієнт записується на прийом до лікаря.
Учасники	Пацієнт
Передумови	Пацієнт увійшов до системи
Постумови	Запис додається до бази даних та відображається на сторінці лікаря.
Основний сценарій	Пацієнт натискає на кнопку ЗАПИС НА ПРИЙОМ, де заповнює усі необхідні поля.

Таблиця 13.13 - Сценарій використання usecase 13

Назва	Перегляд записів на аналізи.
Опис	Лаборант переглядає список записів на аналізи.
Учасники	Лаборант
Передумови	Лаборант увійшов до системи
Постумови	Відображення списку записів на аналізи.
Основний сценарій	Лаборант натискає на кнопку ЗАПИС НА АНАЛІЗИ, де може переглянути усі доступні записи та додати їх результати.

Таблиця 14.14 - Сценарій використання usecase 14

Назва	Додавання результатів досліджень.
Опис	Лаборант додає результати досліджень до системи.
Учасники	Лаборант
Передумови	Лаборант увійшов до системи
Постумови	Додавання інформації про аналізи до бази даних та до електронної картки пацієнта.
Основний сценарій	Лаборант натискає на кнопку ДОДАТИ РЕЗУЛЬТАТИ, де вносить до системи усі необхідні дані.

Було досліджено усі необхідні сценарії використання системи з боку користувачів.

2.2 Проєктування інтерфейсу хмарного вебсервісу міської лікарні

Проєктування інтерфейсу є доволі важливою частиною будь-якої розробки. Оскільки зовнішній вигляд повинен бути привабливим та змушувати користувача використовувати застосунок знов і знов. Але так чи ні будь-які системи мають власні цілі, деякі хочуть затягувати користувача яскравими кольорами, деякі хочуть бути інтуїтивно адаптованими, а інші усе одразу.

Існують різноманітні концепції проєктування, але одна з найпопулярніших це design thinking.

Процес проєктування за допомогою цієї концепції має п’ять етапів, а саме: визначення продукту, дослідження, аналіз, дизайн, тестування. Одним з головних етапів є визначення продукту, тому що треба розуміти розроблюваний продукт та його цілі. А потім вже спираючись на проведене дослідження відтворюються й інші етапи [13].

Після усіх етапів дослідження та аналізу застосунку робиться прототипування. Прототипування – це процес під час якого створюються прототип, які містять у собі певні скетчі задля загального розуміння зовнішнього виду застосунку. Точність прототипу, залежить від етапу розробки та переслідуваємих цілей. Якщо головною метою є надати користувачу початкове уявлення, то буде розроблено прототип з низькою точність.

Прототипування з низькою точність – це доволі простий та ефективний спосіб відтворити необхідне відображення продукту. Метою такого прототипування є перевірка зручності та функціональності інтерфейсу [14].

Під час проєктування інтерфейсу вебсервісу міської лікарні було розроблено головні вайфрейми задля розуміння функціональності та зручності застосунку. Головною ціллю вайфреймів сумістити архітектуру продукту та його зовнішній вигляд. Завдяки цьому можна побачити, як розподіляється місце на

сторінці, розділюються зображення, наявні функції, тощо.

Розглядаючи медичні інформаційні системи слід зазначити, що застосунки такого типу перш за все повинні бути інтуїтивно зрозумілими та функціонально повними. Адже доволі важливо швидко зорієнтуватися у інтерфейсі для роботи. Також слід зазначити, що такі системи не повинні мати у собі контент який міг якось навантажувати ПК, адже це може призвести до певних збоїв у системи та навість втраті інформації за найгіршим розвитком подій. Далі буде наведено основні екрани вебсервісу задля розуміння функціоналу. Але наведені вайфрейми можуть відрізнятися від остаточної версії продукту. Для розробки приблизного інтерфейсу було використано Balsamiq, доволі зручний застосунок який швидко використовується у світі.

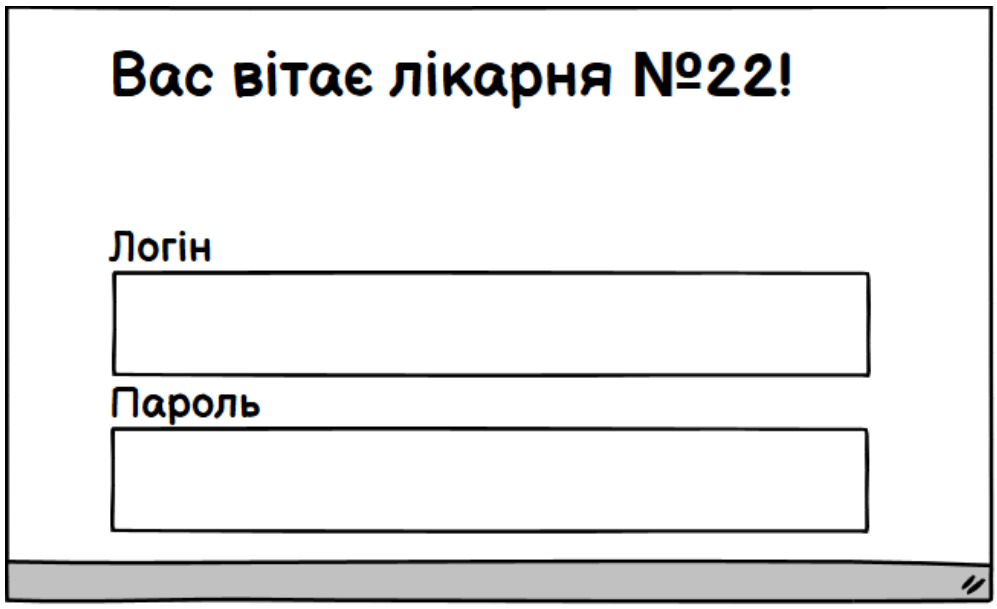

Рисунок 2.2 – Вхід до системи

Екран авторизації містить у собі головні поля, а саме поле логіну та паролю. Така система дозволяє розподіляти права між користувачами.

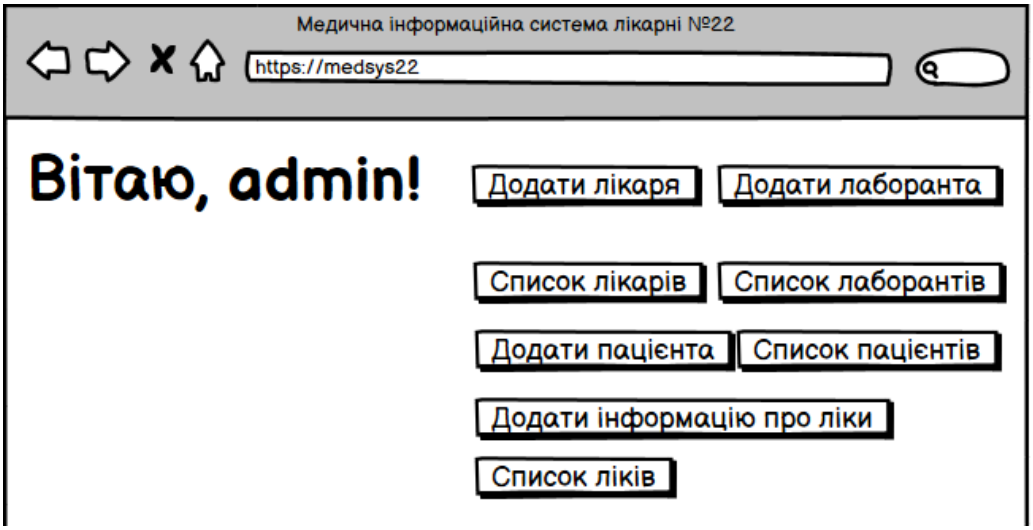

Рисунок 2.3 – Екран кабінету адміністратора

При авторизації адміністратора, він має найбільше повноважень у системі, а саме додавати лікарів та переглядати їх список , додавати лаборантів та медичні рекомендації.

Рисунок 2.4 – Додавання лікаря до системи

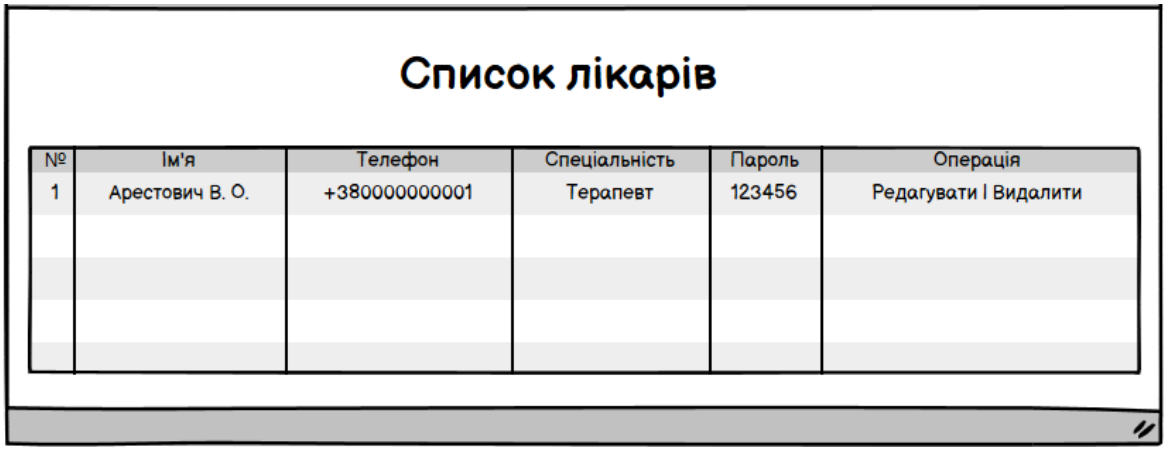

Рисунок 2.5 – Список лікарів

Задля успішної реєстрації лікаря у системі, адміністратор повинен внести певну інформацію, а саме: ID лікаря, його телефон, ПІБ, спеціалізацію та пароль.

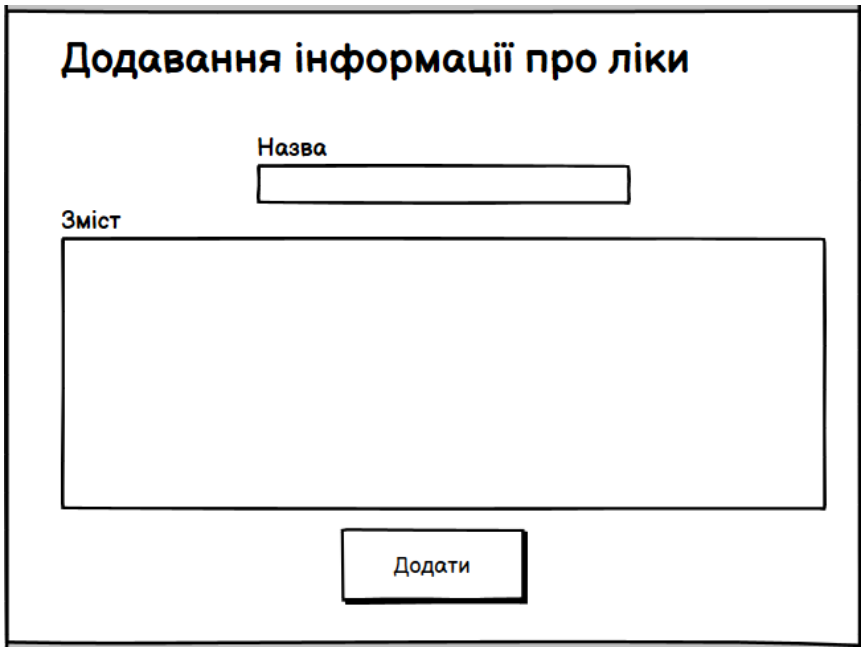

Рисунок 2.6 – Додавання медичних рекомендацій

Додавання рекомендацій є однією з функцій адміністратора, для того щоб це зробити треба внести певний заголовок, а також зміст рекомендації.

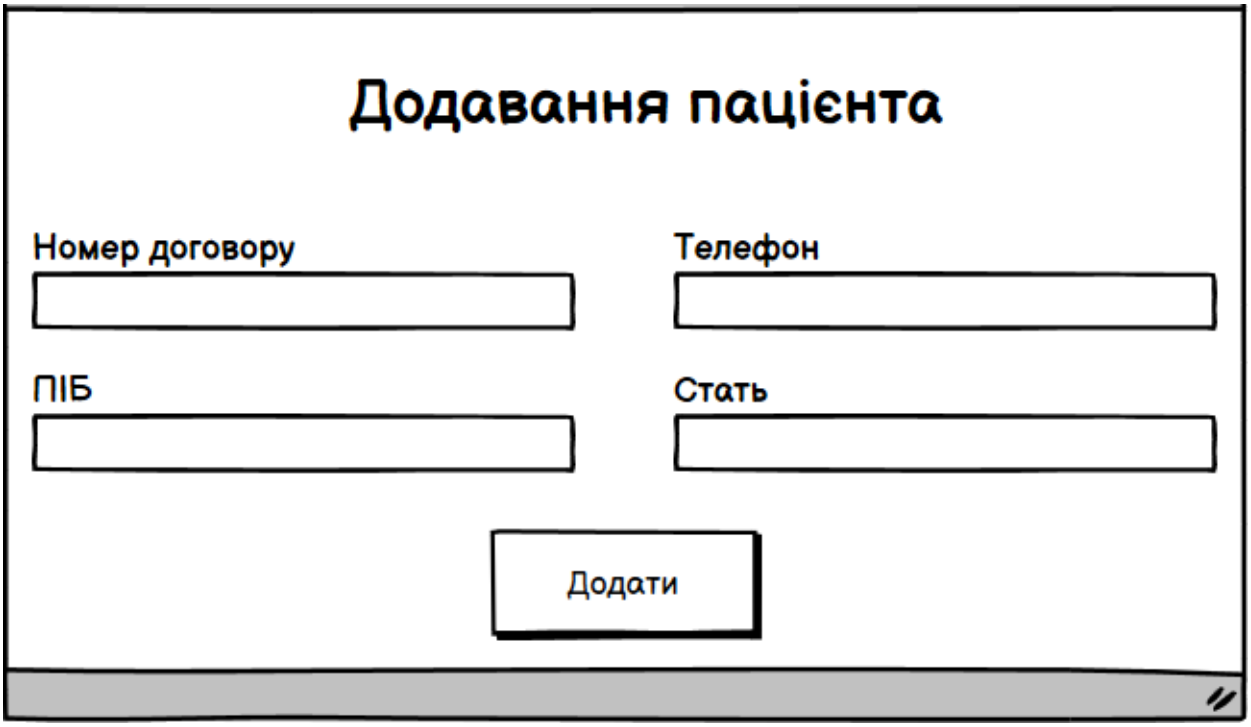

Рисунок 2.7 – Додавання пацієнта

Для додавання пацієнта потрібно внести номер договору, телефон, ПІБ, а також стать.

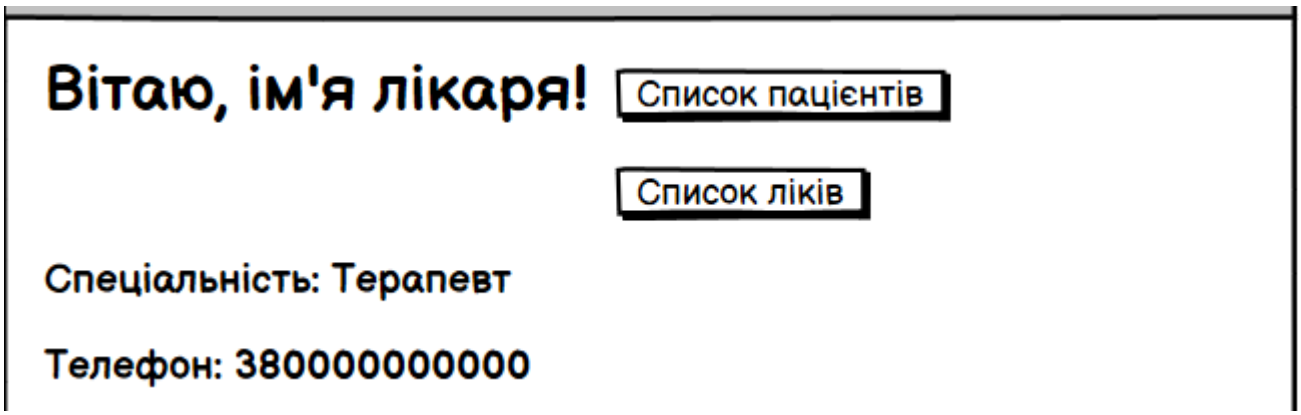

Рисунок 2.8 – Кабінет лікаря

Переходячи до кабінету лікаря, можливо одразу побачити відмінності у правах, лікар працює лише з пацієнтами, тому саме він може додавати їх до системи та дивитися список.

Рисунок 2.9 – Список пацієнтів

Після вдалого додавання є можливість проглянути список пацієнтів та за потреби відредагувати чи видалити його з системи. Також можливість роботи з карткою пацієнта.

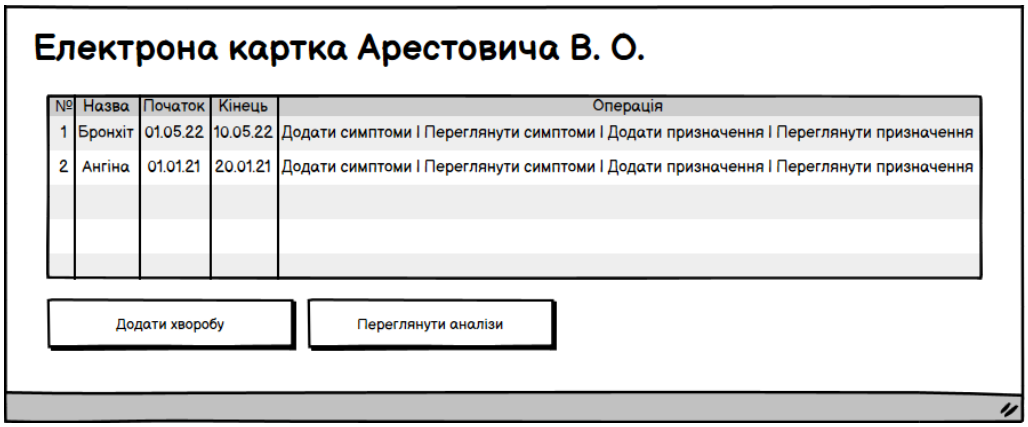

Рисунок 2.10 – Електрона картка пацієнта

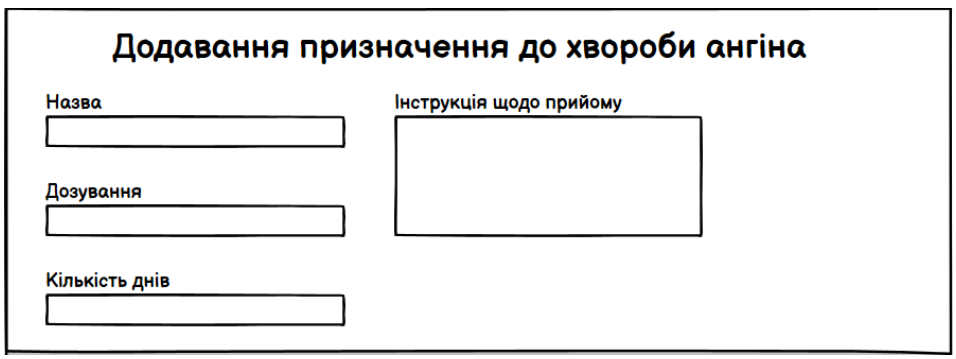

Рисунок 2.11 – Додавання призначення

Наведені вайфрейми є інтуїтивно зрозумілими, що робить систему зручною.

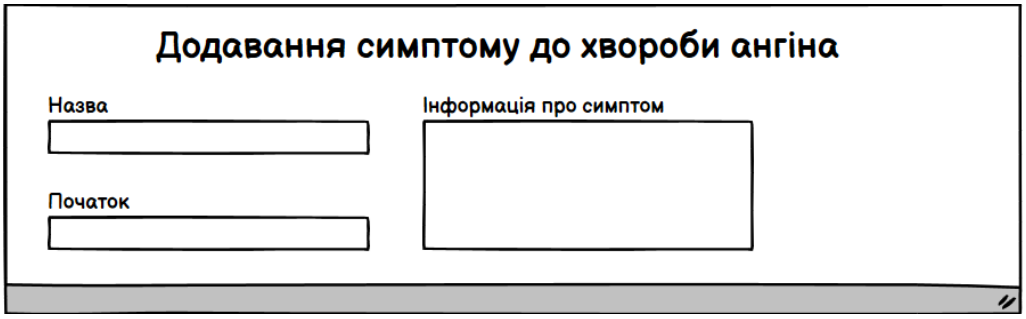

Рисунок 2.12– Додавання симптому

Екран картки пацієнта є важливою частиною системи, адже містить у собі усі необхідні дані, щодо стану пацієнта. Надає можливість додавання симптомів хвороби та медичних призначень.

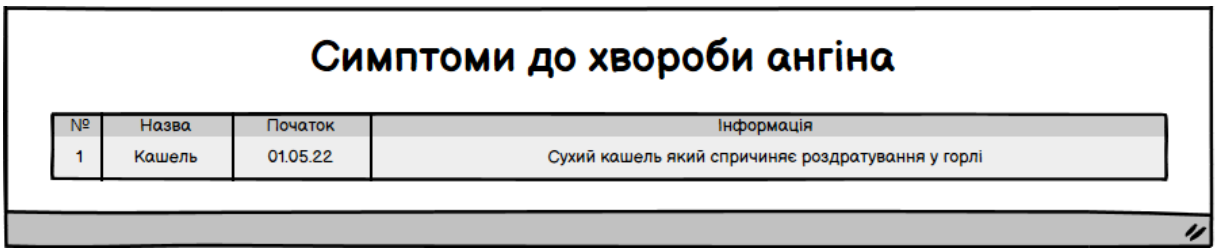

Рисунок 2.13– Перегляд симптомів

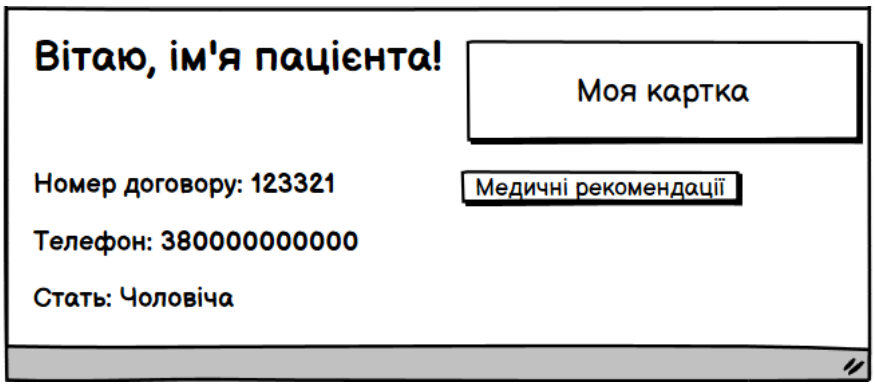

Рисунок 2.14 – Електрона картка пацієнта

Кабінет пацієнта містить обмежений функціонал, він може лише передивлятися інформацію про свої захворювання та медичні рекомендації.

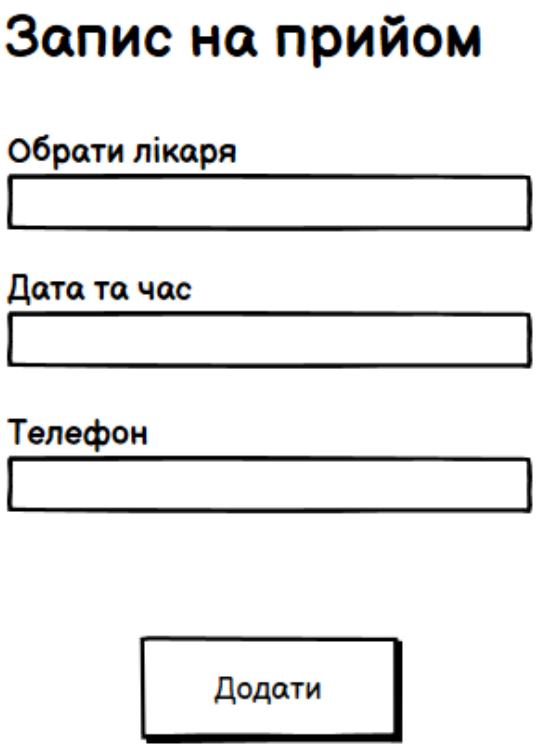

Рисунок 2.15 – Запис на прийом до лікаря

Форма запису на прийом до лікаря містить у собі такі поля, як поле для обрання лікаря, дату та час та мобільний телефон.

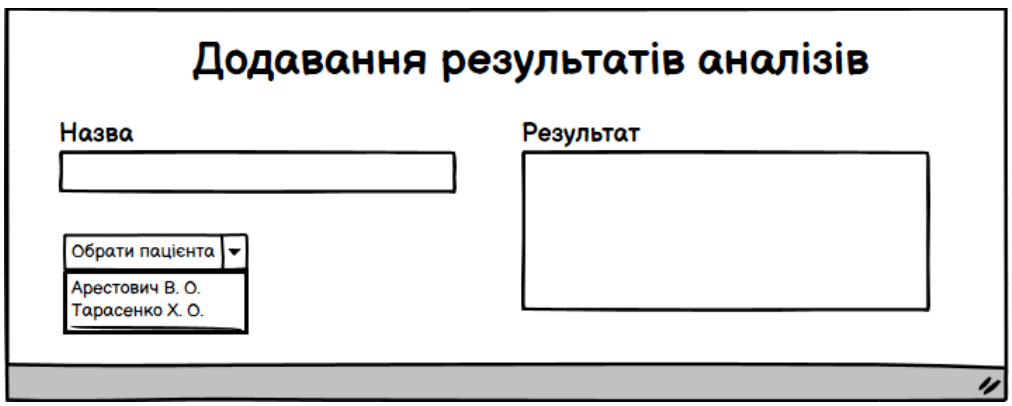

Рисунок 2.16 – Екран додавання результатів аналізів

Екран додавання аналізі має відповідні поля, а саме назву аналізу, список пацієнтів лікарні, а також поле для додавання результату.

2.3 Морфологічна структура хмарного вебсервісу міської лікарні

Морфологічна структура будь-якого застосунку це опис наявних сторінок чи переходів між ними. Розробка структури є важливою частиною будь-якого проекту, оскільки повне розуміння взаємодії сторінок між собою є ключем до успіху розробці. Спираючись на попередній пункт роботи, можна відмітити базовий інтерфейс, а також сторінки які підлягають розробці. Кожна сторінка має своє призначення та має відповідати усім необхідним нормам інтерфейсів та функціональних можливостей медичних інформаційних систем [15].

Перше шо бачить користувач під час роботи з застосунком, так це сторінку авторизації, де задаються необхідні дані задля переходу до особистого кабінету. Система авторизації дозволяє розподіляти юзерів на користувачів з різними правами.

Далі у залежності від типу користувача можна виконувати різноманітні функції. Адміністратор має можливості додавання лікарів, перегляду списку, додавати лаборантів та різноманітні медичні рекомендації. Усі ті функції допомагають функціонуванню системи, а також впливають на досягнення встановлених завдань. Після додавання лікаря його можна побачити у загальному списку, який у свою чергу має можливості редагування інформації про лікарів. Те ж саме й з лаборантами.

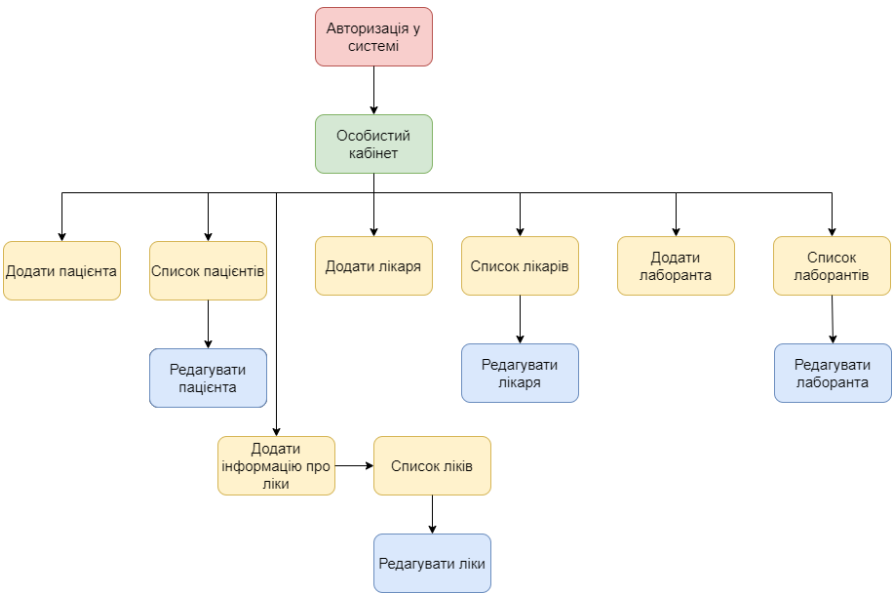

Рисунок 2.17 – Структура застосунку з боку користувача АДМІН

Розглядаючи користувача лікаря можна зазначити, що він має усі необхідні функції для автоматизації роботи. Має можливість перегляду пацієнтів, що записані до нього на прийом, додавання усіх необхідних даних щодо лікування від того чи іншого захворювання.

Кожна з цих функцій відповідає завданню та є важливою частиною системи. Автоматизація внесення симптомів та призначень значно оптимізує роботу лікаря. Адже заповнюючи усі необхідні поля складається загальна картина хвороби пацієнта.

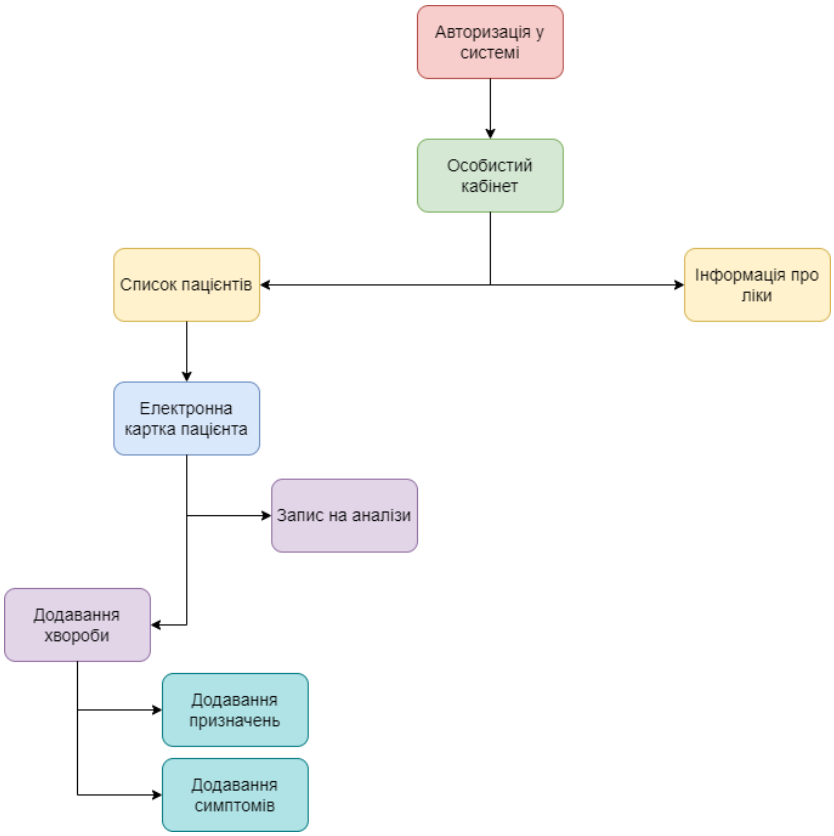

Рисунок 2.18 – Структура застосунку з боку користувача ЛІКАРЯ

Дивлячись на сторінку користувача типу ПАЦІЄНТ, можна побачити не такий великий функціонал. Він лише дозволяє переглядати інформацію про свій стан здоров’я, відслідковувати динаміку та продивлятися власні симптоми, призначення лікаря, а також записуватися на прийом до лікаря.

Але навіть незважаючи на це, на невеликий функціонал можна зазначити, що система автоматизує процес лікування та дає багато переваг з нинішньою.

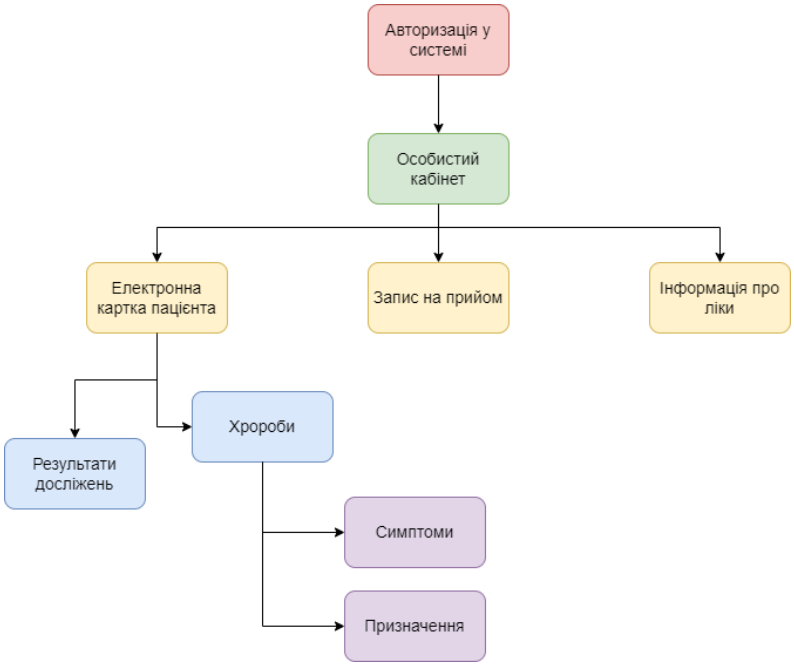

Рисунок 2.19 – Структура застосунку з боку користувача ПАЦІЄНТ

Розглянувши користувача лаборант, можна зазначити невеликий функціонал, але це дуже добре відображає автоматизацію процесу, адже лікарю не буде ніякої потреби самостійно вносити дані до системи, що значно прискорить лікування пацієнта. Кабінет лаборанта складається з ім’я та кнопки додавання результатів досліджень.

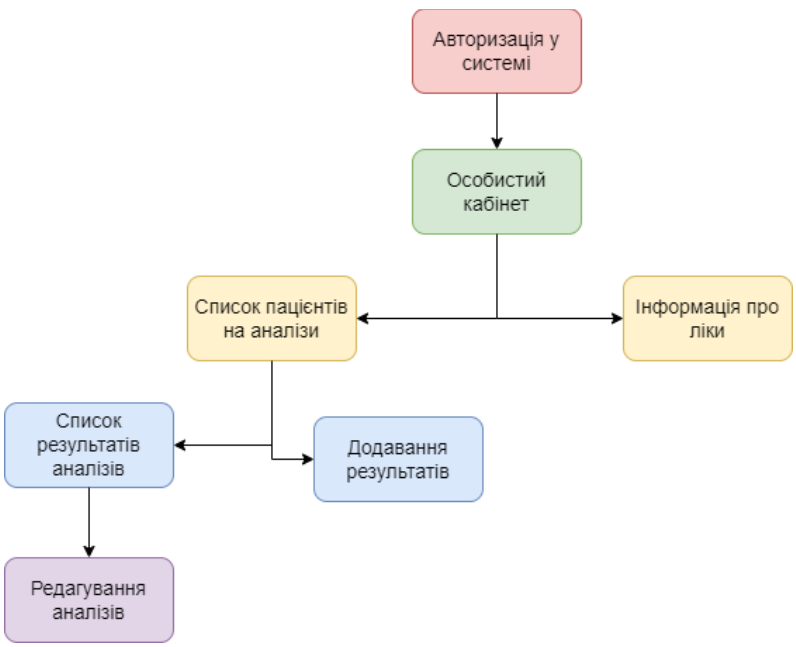

Рисунок 2.20 – Структура застосунку з боку користувача ЛАБОРАНТ

Визначені структури надають можливість розуміння системи.

Завершуючи морфологічний аналіз застосунку слід зазначити, що кінцева версія програмного забезпечення може відрізнятися за рахунок додавання нового функціоналу, тощо.

2.4 Концептуальна модель бази даних хмарного вебсервісу міської лікарні

Проєктування бази даних є процесом який містить у собі багато етапів, кожен з яких містить у собі певні важливі елементи задля повної реалізації бази даних. Основні етапи проєктування це: опис інформаційних об’єктів, проєктування концептуальної моделі, логічне моделювання, а також фізичне. Наразі буде розглянуто проєктування концептуальної бази даних, задля розуміння загальної структури застосунку. Узагалі існує два підходи концептуального проєктування, а саме метод семантичних та об’єктних моделей. Різновиди зв’язків між сутностями чітко відображають концепцію застосунку. Розглянемо їх детальніше [16].

Зв’язок один-до-одного позначає зв’язок між інформацією двох таблиць, тоді як кожен запис використовується у кожній таблиці один раз.

Зв’язок один-до-багатьох містить у собі те, що декілька рядків дочірньої таблиці залежать від одного рядка головної.

Зв’язок багато-до-багатьох відображає те, що кожному запису таблиці відповідає певна кількість записів іншої таблиці.

Дослідивши усі необхідні компоненти задля реалізації концептуальної моделі бази даних, відтворимо її.

На рисунку 2.21 наведено концептуальна модель бази даних застосунку.

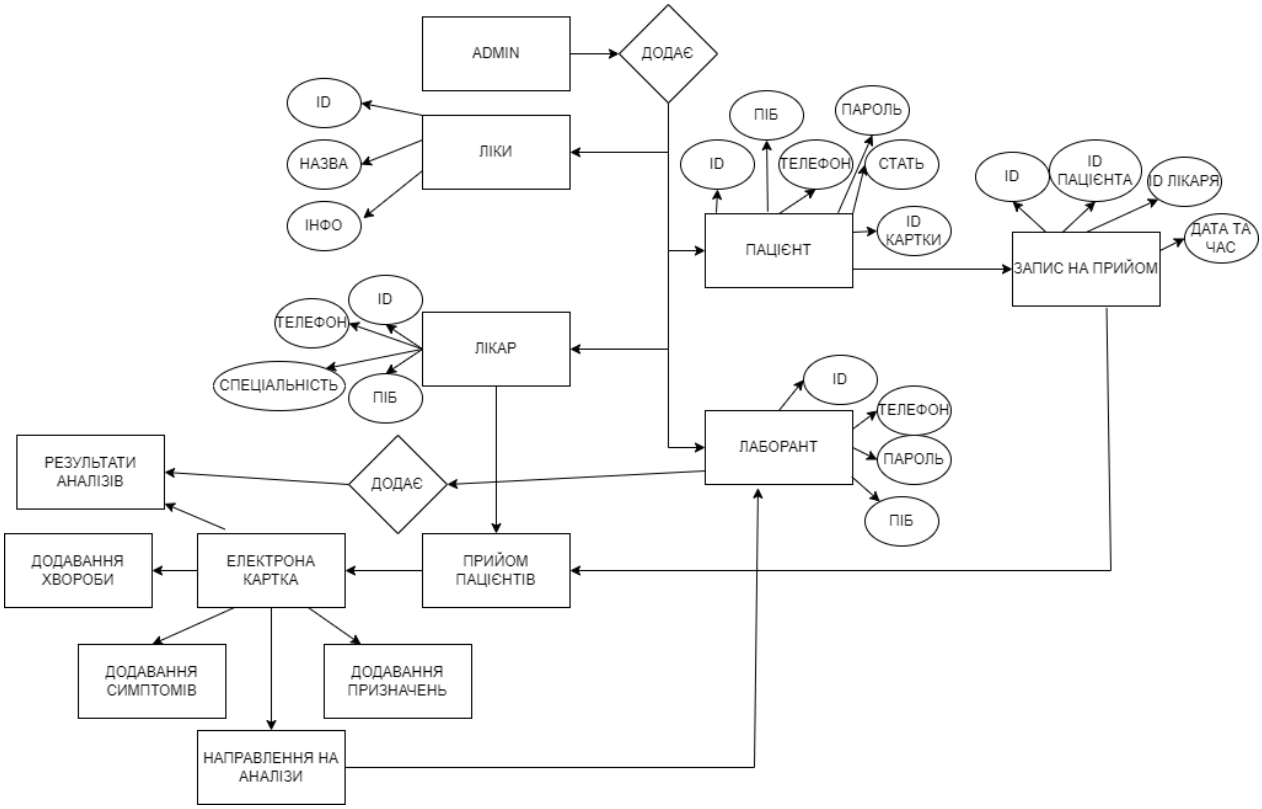

Рисунок 2.21 – Концептуальна модель бази даних

Аналізуючи розроблену концептуальну модель бази даних, можна зрозуміти усю сутність роботи системи та побачити яку інформацію містить та надає система для користувачів.

Висновки до розділу 2

Розроблено діаграму прецедентів та наведено відповідні сценарії роботи системи. Різні користувачі мають різні права на виконання тих чи інших функцій. Також спроектовано первинний інтерфейс хмарного вебсервісу міської лікарні. Вайфрейми відображають базовий функціонал та надають можливість зрозуміти можливості застосунку.

Проведено детальний морфологічний аналіз, де зазначено усі сторінки відповідно до поетапної роботи з сервісом. Дослідження корисність тих чи інших функцій з ретельною аргументацією.

Складено концептуальну модель бази даних, яка надає повну інформацію про структуру системи, а також дозволяє зробити аналіз усіх наявних полей та інформації яку надає сервіс.

3 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ХМАРНОГО ВЕБСЕРВІСУ МІСЬКОЇ ЛІКАРНІ

3.1 Вибір технологій розробки клієнтської частини вебсервісу міської лікарні

У процесі розробки будь-якого застосунку головним етапом проєктування є вибір мови програмування, які засоби та технології будуть використовуватися для тих чи інших нужд. Насамперед слід чітко розуміти складові застосунку та чітко обирати стек технологій. Він повинен включати в себе як технології й для клієнтської частини, серверної, а також бази даних.

Клієнтська структура застосунку відповідає за його зовнішній вигляд. Задля цього існує велика кількість необхідних технологій. Розглядаючи базові з них, то можна виділити такі, як: HTML, CSS, Javascript. Усі ці компоненти відіграють важливу роль у проектуванні клієнтської частини вебсервісу.

Рисунок 3.1 – Логотип HTML5

HTML - це стандартизована мова розмітки, яка потрібна для побудови структури вебзастосунку. Головною задачею є розміщення блоків, таблиць, зображень тощо. Основними частинами мови є теги, які відображають семантичну модель [17].

3.3.1 Технології стилізації вебзастосунку

CSS – це набір певних правил форматування, який застосовується до елементів документа, задля зміни представлення документу, що написаний за допомогою мови HTML. Правила надають можливість змінювати розміри блоків,

колір, позиціонування на сторінці та загальну поведінку на сторінці при тих чи інших умовах [18].

У сучасних реаліях фронтенду при написані стилів з’явилися різноманітні CSS препроцесори, що дозволяє з їх допомогою поліпшити читабельність коду та спрощують стилізацію у цілому. Розглядаючи їх головні переваги над базовим CSS, то можна визначити наступні:

– є можливість імпортувати файли;
– можливість проводити математичні розрахунки;
– розробляти ієрархічну структуру селекторів;
– створювати міксини, для прискорення та автоматизації процесів;
– створювати змінні та використовувати їх у будь-якій точці проекту.

Найпопулярнішими препроцесорами є: SASS, Stylus, Haml, Less. Спираючись на сучасні тенденції розробки, то при виконані проекту буде використано препроцесор SASS, оскільки він є одним з найпопулярніших та доступніших технологій такого типу [19].

Також у сучасній веброзробці для розширення можливостей написання коду використовують певні методології. Самими популярними з них є це: BEM, SMACSS, ECSS. Хоча навіть з трьох методологія BEM виходить на перше місце, тому що містить у собі велику кількість важливих деталей задля розробки застосунку.

Рисунок 3.2 – Логотип CSS

Рисунок 3.3 – Логотип SASS

BEM – у розгортці означає “Block, Element, Modifier”. Головними принципом роботи є розподілення застосунку на блоки. Тим самим, це запобігає

дублюванню коду за допомогою перевизначення блоків. Слід зазначити, що методологія містить у собі, ще більшу кількість важливих складових, а саме:

- відмова від ідентифікаторної стилізації;
- припинення використання селекторів тегів, що впливає на застосунок під час змінення розмітки;
- відмова від універсальних селекторів, що передбачає майбутні зміни;
- використовувати селектори класів, оскільки вони містять у собі більше інформації;

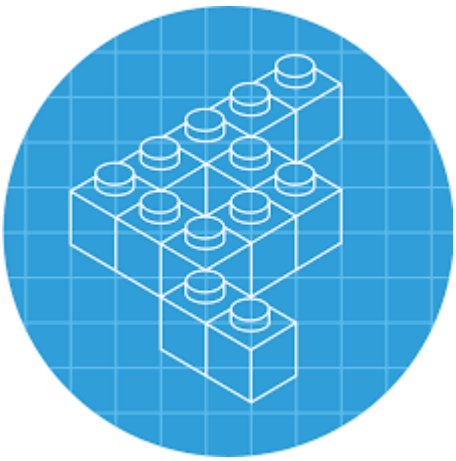

Рисунок 3.4 – Логотип методології БЕМ [20]

Отже розглядаючі методологію BEM, можна визначати, що наведені приципи є доволі суттєвою перевагою, у порівнянні з безметодологічнлю розробкою системи.

3.3.2 Вибір фронтенд Javascript фреймворку

Кожен клієнтський застосунок має свою власну архітектуру. Узагалі розрізняють наступні види, а саме: SPA, MPA, PWA.

SPA – це односторінкова програма, що містить HTML-сторінку, яка динамічно (без повного перезавантаження) оновлюється у відповідь на дії користувача. Архітектура програми влаштована так, що при початковому запуску відвідувач бачить основний контент сайту в браузері, а нові дані завантажуються на ходу при необхідності, наприклад, при прокручуванні або кліку на іконку. Також мають у собі перелік певних переваг, а саме те.

– легкість створення. Для розробки SPA програми вже готові бібліотеки та фреймворки, робота над frontend та backend може вестись паралельно. Крім того, на основі готового коду надалі можна побудувати мобільний додаток;

MPA чи Multi Page Applications за принципом роботи повністю протилежні SPA. MPA — це багатосторінкові програми, які працюють як звичні нам веб-сайти. Вони надсилають запит на сервер і повністю оновлюють сторінку, коли з нею відбувається будь-яка дія (перехід на іншу сторінку, внесення та зміна даних). Подібна архітектура програми значно впливає на швидкість і продуктивність, оскільки більшість даних підвантажується повторно при кожному переході. Також мають у собі перелік певних переваг, а саме те.

– проста SEO-оптимізація. MPA часто використовують сайти, для яких важливо потрапляти в топ пошукових систем. Кожна сторінка має унікальну URL-адресу і стабільна, що дозволяє пошуковим роботам адекватно її просканувати;

– масштабованість. У MPA програму можна вкласти стільки інформації, скільки потрібно, без обмежень за сторінками та функціями;

– перевірена класика. MPA працюють за тими ж принципами, що й знайомі користувачеві веб-сайти із класичною навігацією.

Основою будь-якого вебзастосунку є також язик інтерактивності та взаємодії, а саме Javascript.

Javascript – це мова яка робить взаємодію користувача та системи інтерактивною. Він містить у собі послідовність інструкцій, яка виконується інтерпретатором. Головними рисами є динамічна типізація, програмування за допомгою прототипів та керування пам’яттю з боку системи [21].

У сучасній фронтенд розробці існує багато різноманітних Javascript фреймворків та бібліотек, які призначені для вище сказаних архітектур. Використання фреймворків має переваги, а ніж написання інтерактивності на чистому Javascript. Їх використання допомагає не витрачати велику кількість часу, що пришвидшує розробку, а також зменшує об’єм пам’яті якій потрібен для реалізації застосунку. Чистий Javascript не є гарантом безперебійної роботи, тому

наявні фреймворки допомагають поліпшити його роботу, за рахунок динамічного оновлення інтерфейсу.

Найпопулярнішими Javascript фреймворками є React, а також Angular.js, Ember.js, Vue.js.

У розробці даної системи було обрано React, за невеликою кількістю переваг, а саме за те, що він підходить за своєю концептуальністю та архітектурою.

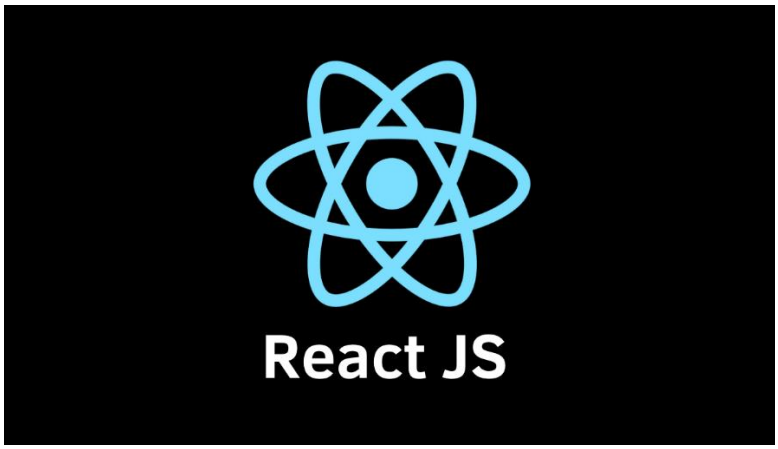

Рисунок 3.5 – Логотип фреймворку React [22]

React – це Javascript бібліотека, яка використовується для розробки користувацьках інтерфейсів, або для окремих частин застосунку. Фреймворк не несе у собі якийсь певний архітектурний шаблон для управління даними, він призначений для розробки інтерфейсів користувача за допомогою компонентів.

Компоненти складаються з методів візуалізації, які повертають опис того, що потрібно відобразити та представити у вигляді мови розмітки. Головне розширення яке використовується під час синтаксису це JSX, яке вміщує в собі HTML та Javascript. Головними складовими компонентів є state та props.

State дозволяє зберігати різноманітні значення та може адаптуватися чи змінюватися на основі дій користувача з системою.

Props – це дані, які надсилаються до компонента під час створення сторінки. З їх допомогою усі компоненти набувають більшої гнучкості та придатності до перевикористання.

Існують два види React компонентів, а саме: класові иа функціональні.

Функціональні компоненти – це функція яка у свою чергу приймає props як вхідні дані та повертає готовий елемент у вигляді JSX.

Класові ж компоненти генеруються шляхом розширення класу React.Component. Стан такого компоненту оновлюється за допомогою спеціального методу setState(). Використання методу дозволяє оновлювати та відображати компонет на сторінці користувача.

Підсумовуючи усе про React можна зазначити на його головних перевагах, а саме:

– потужна спільнота;
– має велику кількість додаткових бібліотек;
– підтримка браузерами, створені спеціальн розширення для налагоджування застосунку;
– DOM структура;
– покращує роботу з UI.

3.2 Вибір технологій розробки серверної частини вебсервісу міської лікарні

У комп’ютерному світі «бекенд» відноситься до будь-якої частини веб-сайту або програмного забезпечення, яку користувачі не бачать. Він контрастує з інтерфейсом, який відноситься до інтерфейсу користувача програми або веб-сайту. У термінології програмування бекенд — це «рівень доступу до даних», а інтерфейс — це «рівень презентації». Більшість сучасних веб-сайтів є динамічними, тобто вміст веб-сторінки створюється на льоту. Динамічна сторінка містить один або кілька сценаріїв, які запускаються на веб-сервері при кожному зверненні до сторінки. Ці скрипти генерують вміст сторінки, який надсилається у веб-браузер користувача. Усе, що відбувається до того, як сторінка буде відображена у веб-браузері, є частиною бекенда.

Існує велика кількість різноманітних бекенд технологій, які й наразі використовуються у сучасному світі розробки застосунків. До їх числа можна приписати такі: PHP, Python, Javascript.

3.3.3 PHP бекенд фреймворк

PHP — це мова сценаріїв на стороні сервера, яка використовується для створення високоінтерактивних і надійних веб-рішень, які забезпечують чудовий досвід роботи з користувачем.

Рисунок 3.6 – Логотип PHP

– рішення PHP може працювати на різних операційних системах – Windows, Linux, Mac, Unix;

– ця мова програмування забезпечує сумісність з більшістю серверів;

– PHP вважається легким для вивчення;

– у більшості випадків PHP працює швидко та ефективно на стороні сервера

– він підтримує різні типи баз даних;

– PHP є фреймворком з відкритим вихідним кодом, який можна завантажити безкоштовно.

Ряд визнаних компаній і технологічних гігантів використовують PHP для запуску своїх серверів і створення багатьох неймовірних речей.

– Facebook: Facebook використовує PHP для підтримки свого сайту. У свою чергу, компанія внесла свій внесок у спільноту, створивши реалізацію, відому як Hip Hop для PHP;

– Вікіпедія: одне з найбільших у світі джерел інформації на будь-яку тему, Вікіпедія побудована на PHP;

– системи керування вмістом (CMS): найпопулярніша у світі система керування вмістом, WordPress, побудована на PHP. Інші системи керування вмістом, такі як Drupal, Joomla та Magento, також побудовані на PHP. Shopify також працює на PHP;

– платформи веб-хостингу: багато веб-хостингових платформ, таких як BlueHost, Site ground та Whogohost, запускають свої сервери хостингу за допомогою PHP.

3.3.4 Python бекенд

Python — це високорівнева, інтерпретована й об’єктно-орієнтована мова програмування, яка підтримує велику кількість бібліотек. В основному він використовується при розробці веб-програм і системних сценаріїв для різних доменів. Крім того, Python може працювати на різних платформах, простий у використанні, оскільки має синтаксис, схожий на англійську. Python в основному використовується для розробки вебзастосунків, підключень до систем баз даних, оскільки він обробляє великі дані, змінює файли та виконує складні обчислення. Python висуває високі вимоги до введення коду, що підвищує його читабельність. Тому Python в основному використовується для менш досвідчених розробників.

Рисунок 3.7 – Логотип Python

Можна виділити основні функції розробки програмного забезпечення Python, щоб було зрозуміло, чи варто вибирати цю мову програмування чи ні.

– синтаксис чітко зрозумілий і легко вносити зміни та налагоджувати його;

– код простий в обслуговуванні;

– він підходить для різних платформ, оскільки є гнучким;

– Python полегшує розробку, оскільки надає готові бібліотеки;

– Python дає можливість протестувати рішення задовго до запуску;

– він підтримує бази даних і надає інтерфейси баз даних до більшості систем СКБД;

– Python підтримує програми з графічним інтерфейсом користувача та має структуру для Інтернету. Приклад: tkmter, WXPython, Django.

3.3.5 NodeJS

Node.js (або просто Node) — це серверна платформа для роботи з JavaScript через ядро V8. JavaScript виконує дію на стороні клієнта, а Node — на сервері. За допомогою Node можна написати повноцінні додатки. Вузол працює із зовнішніми бібліотеками, викликаючи команди з коду на JavaScript та виконуючи роль вебсервера [23].

Відомий тим, що пропонує високопродуктивні програми в режимі реального часу

– економічний з повним стеком JavaScript;

– пропонує високу розширюваність для задоволення індивідуальних вимог;

– пропонує легку масштабованість для сучасних додатків;

– велика та активна підтримка громади;

– допомагає створювати міжфункціональні команди;

– легко навчатися;

– покращує час відповіді та підвищує продуктивність програми;

– скорочує час завантаження за рахунок швидкого кешування;

– підтримка часто використовуваних інструментів.

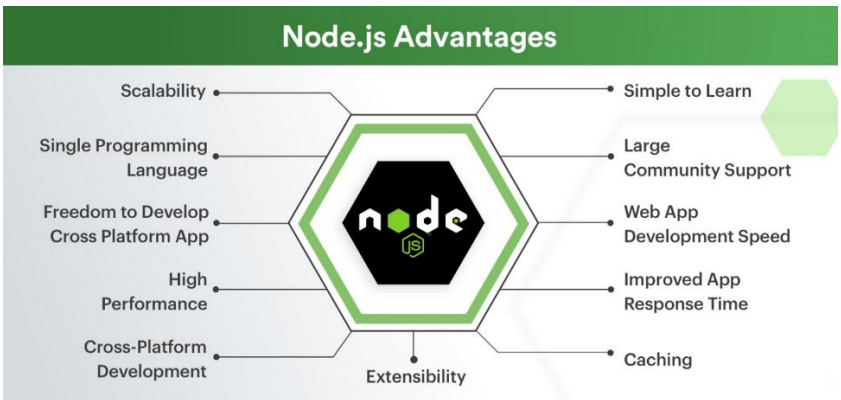

Рисунок 3.8 – Головні переваги NodeJs

Node.js є універсальним і вигідним інструментом для розробки динамічних

вебзастосунків, який має ряд переваг. Він розширив межі додатків JS і може використовуватися як для зовнішніх, так і для бекенд-серверів. Таким чином, він виявляється відмінним вибором на сучасному ринку веброзробки. Використання node.js може надати вам безліч ексклюзивних переваг для створення складних веб-додатків.

3.3 Вибір засобів реалізації бази даних хмарного вебсервісу міської лікарні

Розглядаючі усі наявні засоби задля реалізації бази даних для хмарного вебсервісу міської лікарні, було обрано одного з світових лідерів послуг даного типу, а саме Firebase.

Firebase - це платформа для розробки мобільних додатків та вебзастосунків від компанії Google, в якій є найсучасніші функції для розробки, перекомпонування та покращення додатків.

Firebase - це, по суті, набір інструментів, які розробники можуть використовувати, створюючи та змінюючи програми залежно від своєї потреби.

Використовуючы платформу надається доступ до спеціалізованих сервісів, за допомогою яких поліпшується якість розробки продукту [24].

Найпопулярніші функції платформи Firebase містять у собі бази даних, аутентифікацію, аналітику, зберігання файлів та інше. Наразі налічуєтсья майже 20 різноманітних служб.

Однією з головних особливостей є те, що значна частина сервісів знаходиться у хмарі, тим самим надаючі можливість маштабування застосунку. Також до головних переваг відносяться такі, а саме:

- безкоштовний початковий план;
- швидкість розробки;
- наскрізна платформа для розробки програм;
- працює на платформі Google;
- розробники можуть зосередитись на фронтенді;
- не потрібно використовувати сервер;

– закладено можливості машинного навчання;
– ненерація трафіку для вашої програми;
– моніторинг помилок;
– безпека.

Рисунок 3.9 – Логотип Firebase

База даних Firebase Realtime — це база даних NoSQL, з якої ми можемо зберігати та синхронізувати дані між нашими користувачами в режимі реального часу. Це великий об’єкт JSON, яким розробники можуть керувати в режимі реального часу. Використовуючи єдиний API, база даних Firebase надає програмі поточне значення даних і оновлює ці дані. Синхронізація в режимі реального часу полегшує нашим користувачам доступ до своїх даних з будь-якого пристрою, будь то Інтернет чи мобільний.

Рисунок 3.10 – Firebase Realtime Database

Firebase Authentication надає серверні послуги, прості у використанні пакети SDK і готові бібліотеки інтерфейсу користувача для автентифікації користувачів

у вашому додатку. Він підтримує автентифікацію за допомогою паролів, номерів телефонів, популярних постачальників федеративної ідентифікації, таких як Google, Facebook і Twitter, тощо.

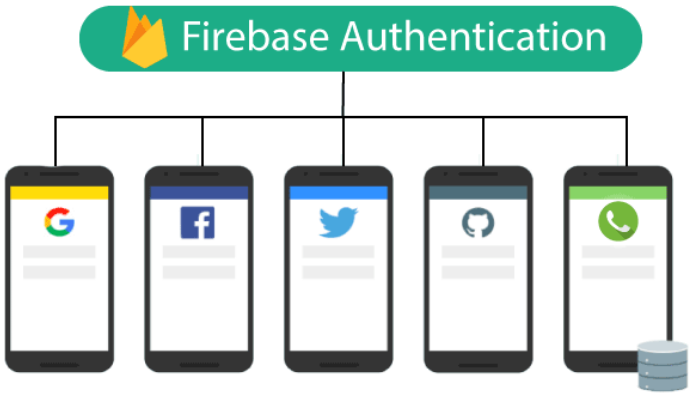

Рисунок 3.11 – Firebase Authentication

Firebase Analytics - це інструмент, який допоможе вам дізнатися більше про продуктивність вашої програми. Він надасть повну інформацію про те, як користувачі операційних систем iOS та Android взаємодіють із цією програмою.

Основою аналітики Firebase є Google Analytics, яка є необмеженим та безкоштовним аналітичним рішенням. У всіх сервісах Firebase буде інтегрована аналітика, яка здатна надавати своїм користувачам необмежену кількість звітів з 500 подій, які визначають користувачі Firebase SDK.

Рисунок 3.12 – Логотип Firebase Analytics

Firebase Analytics – це фантастична платформа для аналізу продуктивності мобільного додатка. Його переваги включають готову інтеграцію з SDKFirebase, готові до використання звіти та аналіз у реальному часі. Ціни, можна сказати, не можуть бути кращими, воно абсолютно безкоштовно і доступне в рамках планів Spark і Blaze.

Висновки до розділу 3

Проаналізовано сучасні тенденції розробки вебзастосунків. Розглянуто різноманітні технології, а також обрано бажаний стек задля реалізації сервісу. У результаті було обрано наступні технології для кожної складової розробки.

Для Frontend частини було обрано базові технології, а саме HTML, CSS та JavaScript, який у свою чергу включає в себе фреймворк React.

Для бекенд частини було проаналізовано декілька технологій та обрано бажану, а саме NodeJs, адже за сукупність переваг саме ця платформа має певну кількість переваг зі своїми конкурентами, саме для реалізації вебсервісу.

Вибір бази даних припав на Firebase, адже це система є справжнім лідером у своїй сфері, та включає в себе доволі багато різноманітних сервісів, які покращують розробку та дозволяють розподіляти увагу на інші важливі аспекти застосунку.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ХМАРНОГО ВЕБСЕРВІСУ МІСЬКОЇ ЛІКАРНІ

4.1 Фізична модель бази даних хмарного вебсервісу міської лікарні

Фізична модель бази даних – це спеціальна модель, що описує які дані перебувають у базі даних. Така модель включає в себе загальну специфікацію всіх таблиць та стовпців. Сама ж специфікація включає в себе деталізовану інформацію, а саме назву таблиці, кількість стовпців, а також ім’я стовпців та їх типи даних. Також модель містить у собі первинні ключі задля кожної таблиці та демонструє загальний взаємозв’язок між таблицями.

Під час розробки застосунку було використано Firebsae Realtime Database, що використовує NoSQL базу даних.

NoSQL – це спеціалізовний підхід до реалізації масштабуємого сховища інформації з доволі гнучкою модуллю даних, який має суттєви відмінності від реляційних СКБД.

Firebase використовує документо-орієнтований підхід зберігання інформації. Це надає можливість зберігати дані парами, а саме ключ-значення, що у свою чергу формують структуровані документи у форматі JSON.

Розглянемо декільки таблиць, що зберігаються у застосунку та наведемо їх детальний опис.

Таблиця 4.1 – Структура документа лікаря

Ключ	Тип	Опис
id	String	Унікальний ідентифікатор користувача
name	String	Ім’я лікаря
phone	String	Телефон лікаря
speciality	String	Спеціальність лікаря
password	String	Пароль лікаря від його особисого кабінету.

Документ ЛІКАРЯ надає розуміння про загальну форму та використані поля у застосунку. З допомогою цього можливо зрозуміти сутність реалізації системи.

Таблиця 4.2 – Структура документа пацієнта

Ключ	Тип	Опис
id	String	Унікальний ідентифікатор користувача
name	String	Ім’я пацієнта
phone	String	Телефон пацієнта(використовується як логін)
contract	int	Номер договору пацієнта(використовується як пароль)
sex	String	Стать пацієнта
birthday	date	Дата народження пацієнта

Таблиця 4.3 – Структура документа лаборанта

Ключ	Тип	Опис
id	String	Унікальний ідентифікатор користувача
name	String	Ім’я лаборанта
phone	String	Телефон лаборанта
password	int	Пароль лаборанта для особистого кабінету
laboratoryNumber	String	Номер лабораторії лаборанта

Таблиця 4.4 – Структура документа ліків

Ключ	Тип	Опис
id	String	Унікальний ідентифікатор запису
name	String	Назва ліків
text	String	Загальна інформація про ліки

Таблиця 4.5 – Структура документа симптомів

Ключ	Тип	Опис
id	String	Унікальний ідентифікатор запису
name	String	Назва симптому
infoAbout	String	Загальна інформація про симптом
start	date	Дата початку симптома

Визначено одни з базових докумнентів для успішної реалізації вебсервісу.

Таблиця 4.6 – Структура документа призначення

Ключ	Тип	Опис
id	String	Унікальний ідентифікатор запису
name	String	Назва призначення
dose	String	Необхідне дозування
amountDays	int	Кількість днів
instruction	String	Інформація про прийом ліків

У ході розробки вебсервісу було визначено та спроектовано необхідні документи, що містять усю необхідну інформація задля успішного функціонування системи. Вище було наведено одни з основних документів NoSQL бази даних. Кожна таблиця допомогає зроузміти усю сутність проекту. Слід зазначити, що використаний документо-орієнтований підхід надає певні переваги та є більш зручним для використання.

Головні переваги використання не реляційної бази даних є те, що це найкращий спосіб для досягнення високої швидкотсі читання у розподіленому середовищі. Також слід зазначити, можливість зберігати фізчині об’єкти у тому вигляді, в якому легче працювати.

4.2 Діаграма класів та загальна реалізація роботи

Діаграма класів – це одна з головних UML компонентів, що допомогають у розробці будь-якого застосунку. Розробки діаграми дозволяє структурувати логічну базу даних, а також сформувати загальний концепет застосунку. Надання статичного представелння застосунку є важливою частиною розробки. Загальний зовнішний вигляд діаграми дозволяє більш точно формувати уявлення про систему, адже усі елементи відображаються безпосередньо перед очима. Також прискорюється розробка та написання коду, що сприятливо впливає на загальну продуктивність роботи. Під час розробки проекту було розроблено діаграму класів, що зображена на рисунку 4.1.

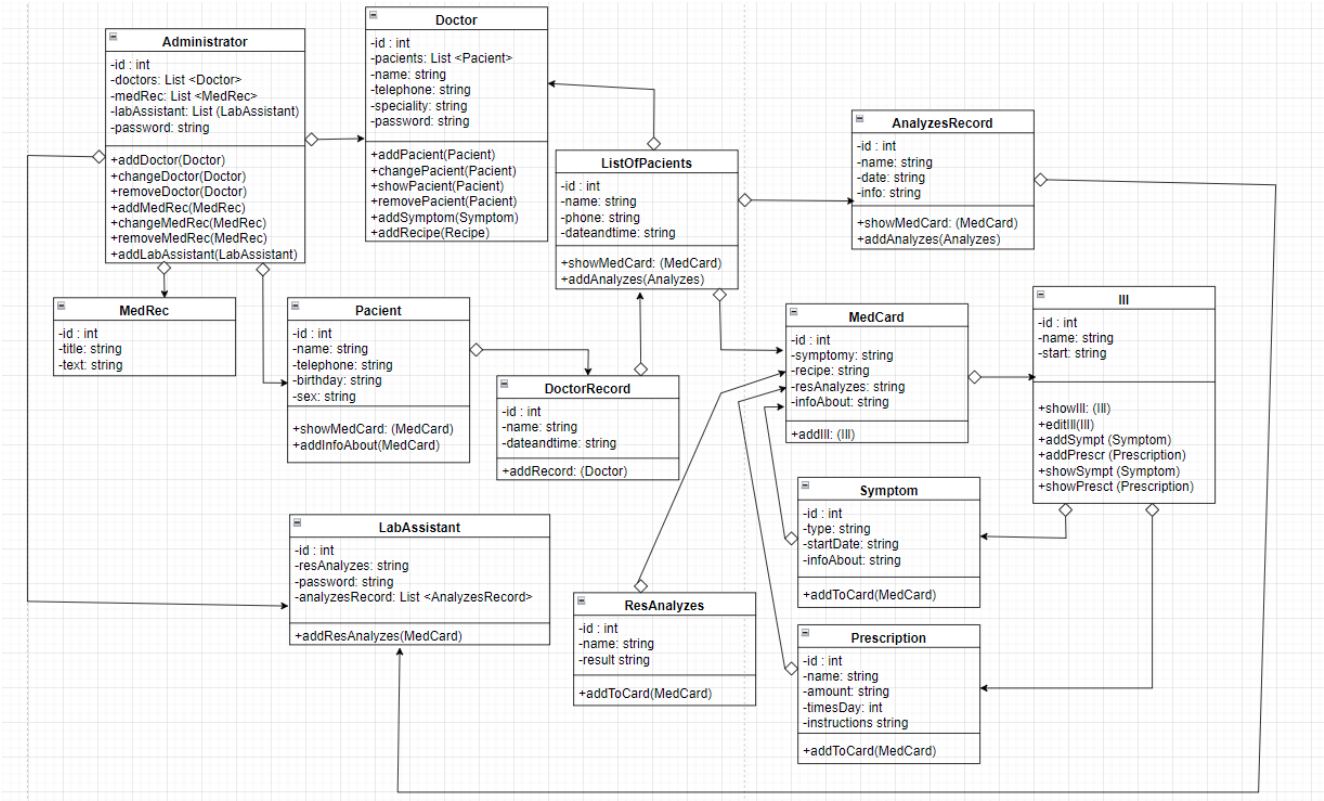

Рисунок 4.1 – Діаграма класів

Використання різноманітних класів та методів розширює загальний функціонал застосунком, що є доволі непоганим результатом. Отже після розробки діаграми можливо сформувати загальну структура застосунку, що зображена на рисунку 4.2.

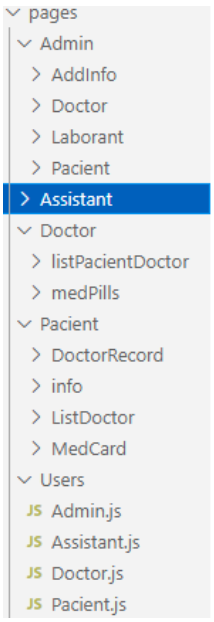

Рисунок 4.2 – Файлова структура застосунку

Розглядаючи файлову структуру можна визначити певні ознаки ієрархії. Також слід виділяти наявні класи та принципи розробки. Оскільки під час розробки використовується Firebase, то потрібно проветсти необхідні налаштування, що наводяться на рисунку 4.3.

```
import firebase from "firebase/compat/app";
import "firebase/compat/database";
import "firebase/compat/auth"


const firebaseConfig = {
  apiKey: "AIzaSyAiSsYS-nKFumUZl6-5t4iFRRJg1khExyI",
  authDomain: "hospital-ad05f.firebaseapp.com",
  databaseURL: "https://hospital-ad05f-default-rtdb.firebaseio.com",
  projectId: "hospital-ad05f",
  storageBucket: "hospital-ad05f.appspot.com",
  messagingSenderId: "164720660337",
  appId: "1:164720660337:web:b09077ededff10f2b6f0d0"
};

const fireDb = firebase.initializeApp(firebaseConfig);
export const auth = fireDb.auth();
export default fireDb.database().ref();
```

Рисунок 4.3 – Налаштування Firebase

Налаштований Firebase дозволяє викоистовувати відповідні влаштовані функції, а саме Firebase Realtime Database та Firebase Auth. Розглянемо загальне підключення до бази даних на рисунку 4.4.

```
import fireDb from "../../../../src/firebase";
```

Рисунок 4.4 – Підключення бази даних

Підкючення до бази даних відбувається за прямими посиланням на налаштування. Використавання бази даних виконується за рахунок імпортованого значення fireDb, що відображається на рисунку 4.5.

```
  if (!name || !text) {
    toast.error("Заповніть поля");
  } else {
    fireDb.child("pills").push(state, (err) => {
      if (err) {
        toast.error(err);
      } else {
        toast.success("Успіх!");
      }
    });
    setTimeout(() => navigate("/admin"), 500);
  }
};
return (
```

Рисунок 4.5 – Використання бази даних

Маршрутизація є важливою частиною розробки застосунків. Фреймворк React надає можливість зручної маршутрутизації за рахунок відповідних бібліотек.

```
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
```

Рисунок 4.6 – Використання бази даних

React Router – це спеціальна бібліотека для маршрутизації в React. Загальні можливості містять у собі збереження користувацького інтерфейсу та синхронизацію з URL. Бібліотек має більше використання з боку серверної частини, що дозволяє створювати динамічні посилання. Такий підхід допомагає зберегти читабельність програмного коду.

Загальний роутинг проекту є доволі великим за обсягом та у свою чергу може бути оптимізований за допомогою й інших бібліотек, що формують більш компактні списки роутингу. Головна маршрутизація проекту відображена на рисунку 4.7.

```
<Routes>
  <Route path="/signup" element={<Signup />}></Route>
  <Route path="/" element={<Login />}></Route>
  <Route path="/admin/*" element={<Admin />}></Route>
  <Route path="/editPacient/:id" element={<EditPacient />}></Route>
  <Route path="/editDoctor/:id" element={<EditDoctor />}></Route>
  <Route path="/editAssistant/:id" element={<EditAssistant />}></Route>
  <Route path="/editInfoPills/:id" element={<EditInfoPills />}></Route>
  <Route path="/pacient/*" element={<Pacient />}></Route>
  <Route path="/doctorRecord/:id" element={<DoctorRecord />}></Route>
  <Route path="/doctor/*" element={<Doctor />}></Route>
  <Route path="/addIll" element={<AddIll />}></Route>
  <Route path="/editIll/:id" element={<EditIll />}></Route>
  <Route path="/viewIll/:id" element={<View />}></Route>
  <Route path="/viewSympt" element={<Vieww />}></Route>
  <Route path="/addSympt" element={<AddSympt/>}></Route>
  <Route path="/addPrescr" element={<AddPrescr/>}></Route>
  <Route path="/viewPrescr" element={<ViewPrescr />}></Route>
  <Route path="/viewRes" element={<ViewRes />}></Route>
  <Route path="/addAnalyzes" element={<AddAnalyzes />}></Route>
  <Route path="/assistant/*" element={<Assistant />}></Route>
</Routes>
```

Рисунок 4.7 – Загальна маршрутизація проекту

Оскільки вебсервіс має певні рольові розподілення між користувачами, було реалізовано відповідні класи та методи.

```
const Admin = () => {
  const [activeTab, setActiveTab] = useState("Admin");
  return (
    <div>
      <div>
        <ul>
          <li>
            <Link className="link" to={"addPacient"}>
              <p onClick={() => setActiveTab("Add Pacient")}>Додати пацієнта</p>
            </Link>
          </li>
          <li>
            <Link className="link" to={"listPacient"}>
              <p onClick={() => setActiveTab("List Pacient")}>
                Список пацієнтів
              </p>
            </Link>
          </li>
          <li>
            <Link className="link" to={"addDoctor"}>
              <p onClick={() => setActiveTab("Add Doctor")}>Додати лікаря</p>
            </Link>
          </li>
```

Рисунок 4.8 – Фрагмент коду класа Admin

Узагалі загальна структура вебсервісу відповідає структурі CRUD, але має деякі зміни та більш складні зв’язки між класами, тощо.

4.3 Інструкція користувача хмарного вебсервісу міської лікарні

Під час завантаження вебсервісу надається можливість авторизації користувача у систему. Якщо користувача ж не знайдено у системі, то з’являється відповідне вікно про хибні дані. На рисунку 4.9 зображено форму авторизації до системи.

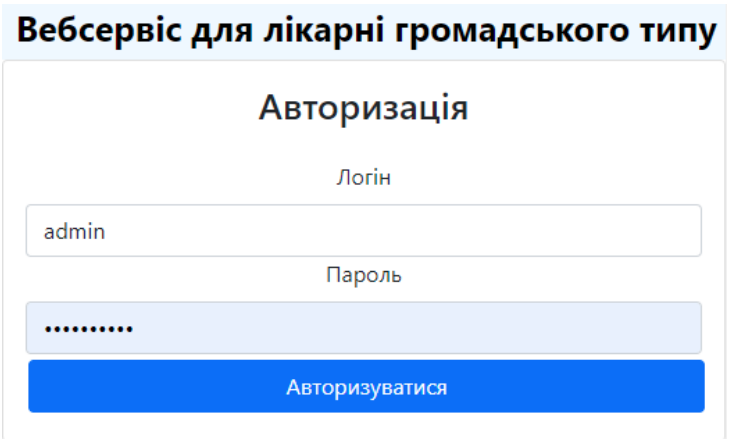

Рисунок 4.9 – Форма авторизації користувача

Також за специфікою застосунку лише адміністратор має право реєструвати користувачів у системі. На рисунку 4.10 зображено форму реєстрації користувачів у системі.

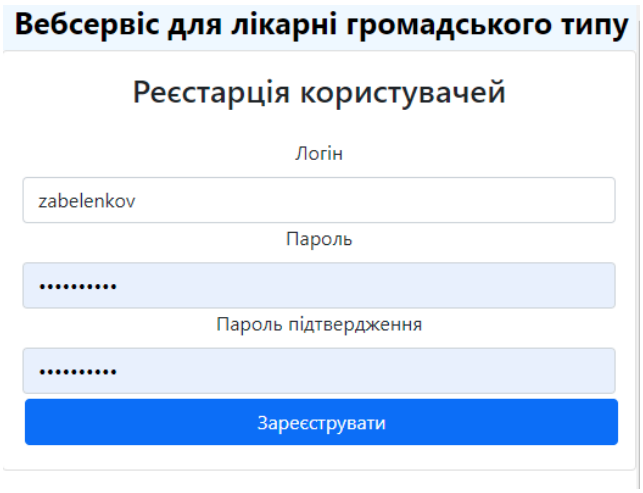

Рисунок 4.10 – Форма реєстрації користувачів

Після вдалої авторизації у системі користувач потрапляє до сторінки, де відображаються доступні функції. На рисунку 4.11 зображено функції адміністратора.

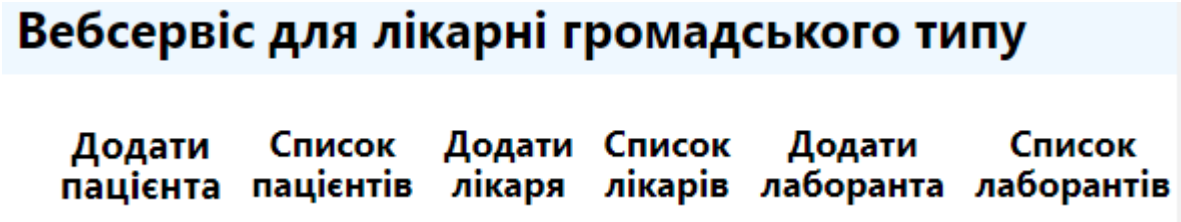

Рисунок 4.11 – Функції адміністратора

Головна функція адміністратора є додавання та генерування списків необхідних даних, а саме пацієнтів, лікарів, лаборантів. На рисунку 4.12 зображено форму додавання інформації про пацієнта до системи.

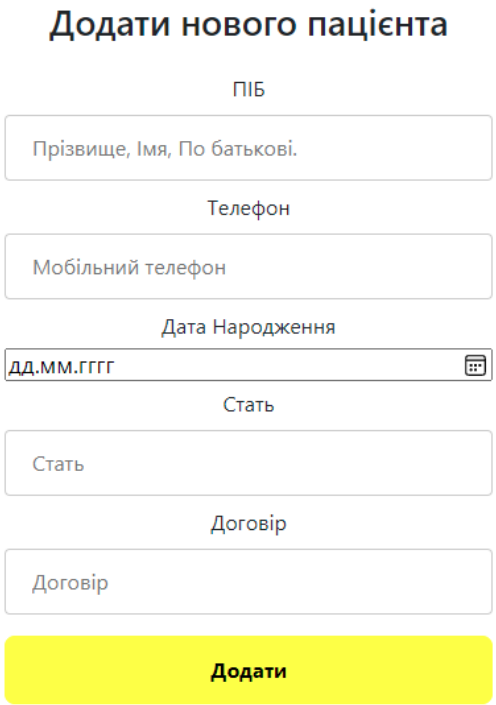

Рисунок 4.12 – Форма додавання інформації про пацієнта

Після вдалого додавання інформації формується спеціальний список за допомогою якого, можна відзначати пацієнтів у системі. На рисунку 4.13 зображено згенерований список.

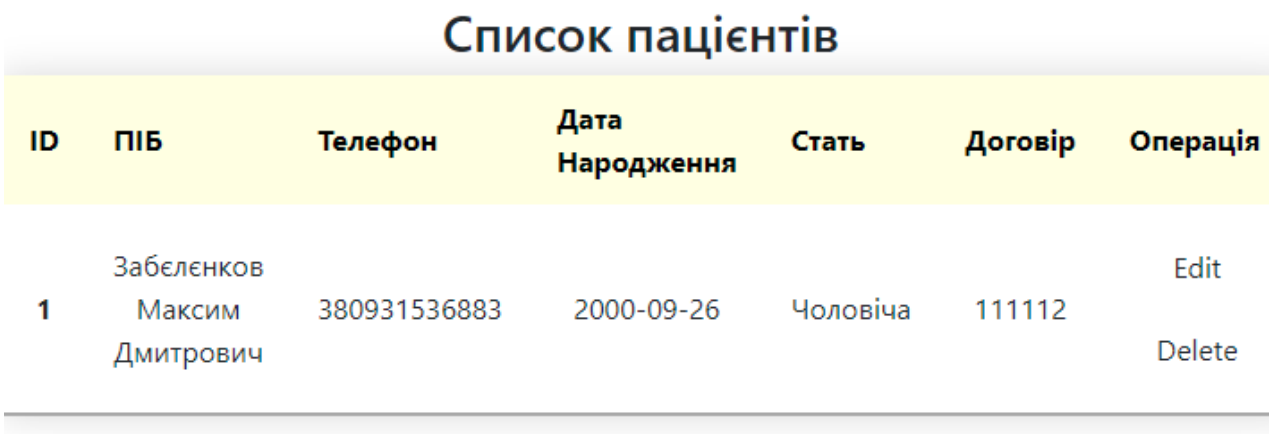

Рисунок 4.13 – Список пацієнтів у системі

Переходячи до пацієнта, слід зазначити спрощений функціонал, але не менше інформативний. На рисунку 4.14 зображено відповідні функції користувача.

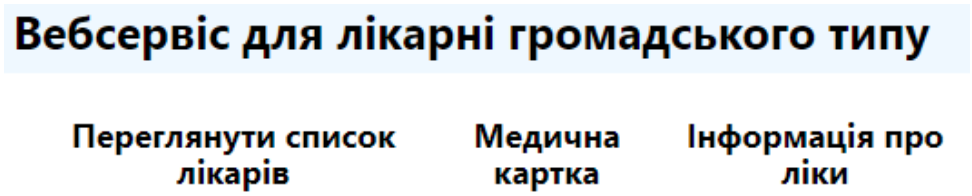

Рисунок 4.14 – Форма додавання інформації про пацієнта

Кожна з функцій є доволі важливою частиною роботи застосунку, переходячи до пункту ПЕРЕГЛЯНУТИ СПИСОК ЛІКАРІВ, надається можливість запису до бажаного лікаря. На рисунку 4.15 зображено список наявних лікарів у системі.

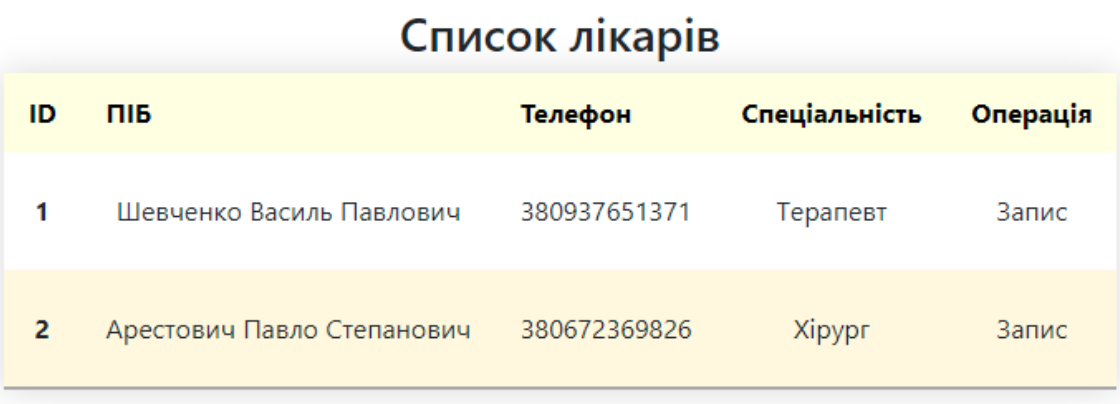

Рисунок 4.15 – Список наявних лікрів

Оскільки лікар має доволі важливу роль у системі, він має мати й відповідні функції. Наразі лікар маж функції перегляду списку пацієнтів, що зробили запис до нього на прийом, а також може дивитися медичні картки пацієнтів та додавати власні примітки, а саме хвороби, симптоми, призначення, а також виписувати направлення на аналізи.

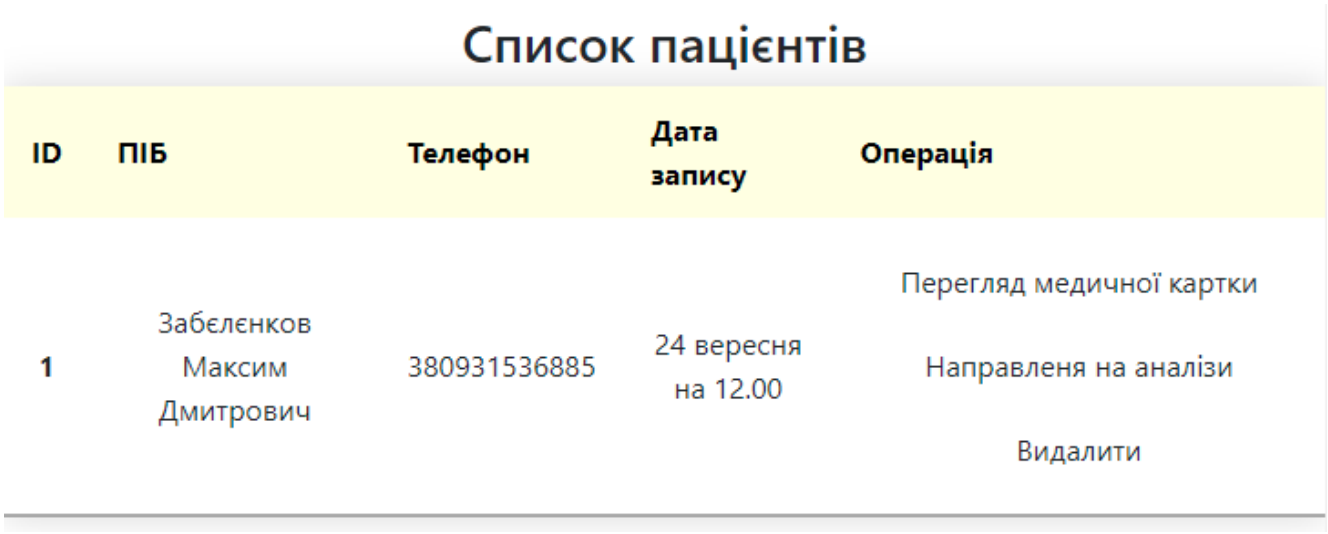

Рисунок 4.16 – Список пацієнтів, що записані на прийом

Натиснувши на перегляд медичної картки, лікар потравляє до картки пацієнта, де може робити різноманітні маніпуляції з усіма необхідними даними.

Лікар має можливіст додавання хвороби до картки пацієнта. На рисунку 4.17 зображено форму додавання хвороби.

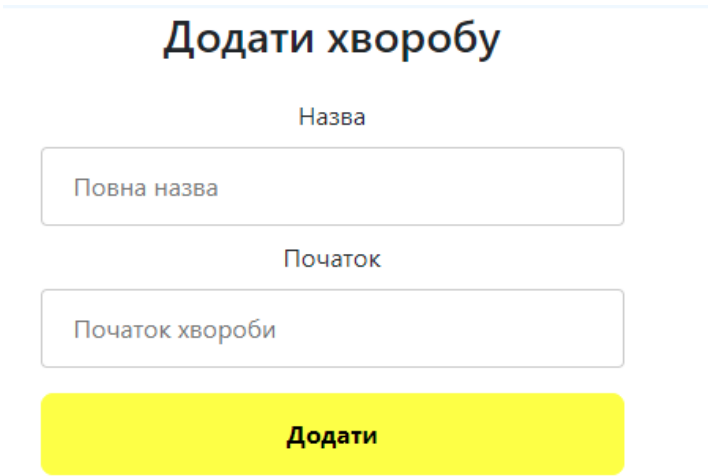

Рисунок 4.17 – Форма додавання хвороби

Після вдалого додавання хвороби, є можливість зазначити симптоми та призначення до тієї ж самої хвороби. На рисунку 4.18 зображено функції роботи з хворобою.

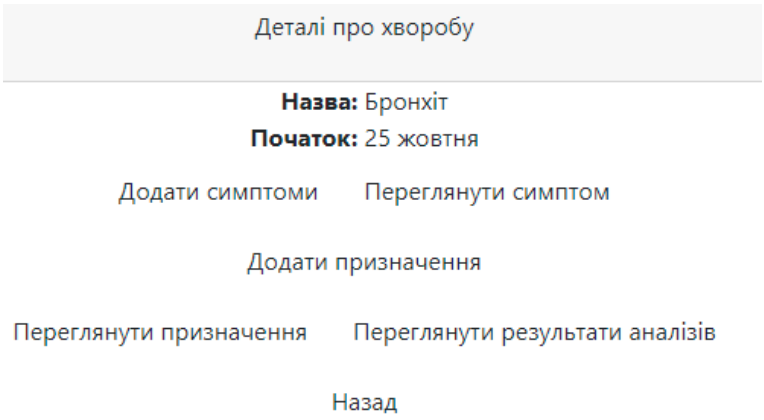

Рисунок 4.18 – Деталі про хворобу та функції додавання значень.

Як можна побачити на рисунку є функція переглянути результати аналізів, за цей функціонал відповідає Лаборант. Головним функціями є додавання результатів аналізів.

Загальний процес роботи лаборанта можна роздивитися наступним чином.На сторінці лаборанта формується список відповідних направлень на дослідження, які виписує лікар. Лабораторія проводить усі необхідні дослідження та згодом лаборант додає результати досліджень до системи. На рисунку 4.19 зображено форму додавання результатів до системи.

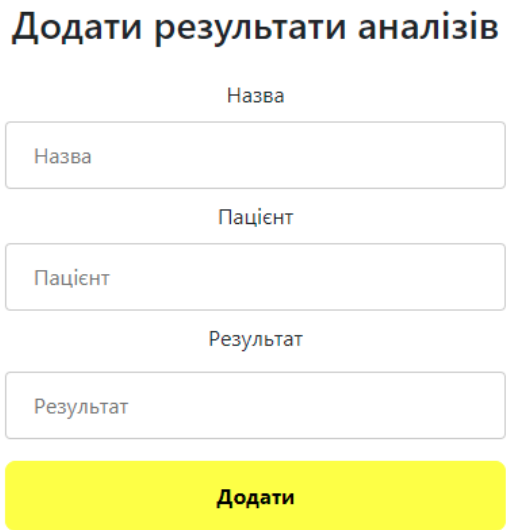

Рисунок 4.19 – Форма додавання результаів аналізів

Крім того є загальна функція виходу з системи, яка відображається поруч з головним меню кожного користувача. При натисненні користувач виходить з системи та повертається до вікна авторизації.

Висновки до розділу 4

Отже в ході розділу було розроблено фізичну модель бази даних, де було зазначено усі необхідні документи, було розглянуто головні переваги такого підходу до розробки.

Було розроблено діаграму класів, що у свою чергу надає можливості для реалізації, загальних методів та класів системи. Продемонстровано фрагменти програмного коду, що у свої сутності відображають основні функції системи.

Крім того, було створено керівництво користувача, яке описує загальні функціональні можливості системи. При виникненні будь-яких проблем з використанням застосунку, користувач може звернутися до керівництва та знайти там необхідну інформацію про ту чи іншу функцію.

ВИСНОВКИ

За результатом виконання кваліфікаційної роботи бакалавра було розроблено хмарний вебсервіс для міської лікарні. Задля цього та для розуміння вимог та загальних тенеденцій даної сфери, було проаналізовано та описано предметну сферу. Також було розглянуто аналоги систем, що виконують того ж самого роду функції.

Проектування проводилось на основі розглянутих аналогів та аналізу предметної сфери було створено загальне завдання, яке й використовувалося для проектування медичної інформаційної системи та загальної моделі застосунку.

Головний функціонал системи полягає створені єдиної інформаційної бази про пацієнтів, лікарів та лаборантів. Було розглянуто окремі пункти розроки, що сприятливо впливає на розробку.

Задля зберігання усієї інформації системи було використано документо-орієнтований підхід, який реалізується за допомогою Realtime Database. Клієнтскька частина була розроблена за допомогою React фреймворку та сучасних методологій БЕМ. Серверна частина була реалізована за допомгою NodeJs. Під час розробки було поліпшено навички та знання, щодо такого підходу як CRUD, а також знання щодо маршрутизації між елементами системи. Набутий досвід позивтивно сприятиме для подальших розробок та поліпшення вже наявних проектів включаючи й даний проект.

Також було написано спеціальний розділ з охорони праці, у якому було проаналізовано умови праці на робочому місці, а також розглянуто техніку безпеки для розробника програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Health Care Systems. URL: https://www.physio-pedia.com/Health_Care_Systems (дата звернення: 12.05.2022)
2. Дячук С. Ф., Фесина А. Ю., Ямщіков Д .Б. Медичні інформаційні системи. *Актуальні задачі сучасних технологій*: тези доп. VI Міжнародної науково-технічної конференції молодих учених та студентів. 2017. С. 62-63. URL: https://emci.ua (дата звернення: 13.05.2022)
3. Медичні інформаційні системи URL: http://ir.nmapo.edu.ua:8080/bitstream/lib/337/1/Mic.pdf (дата звернення: 13.05.2022)
4. Рябокінь Ю. М., Бех А. О., Руденко В. В. Автоматизація діяльності медичних закладів. *Інженерія програмного забезпечення*. 2015. № 4 (24). С. 44-52.
5. Пономаренко П. В. Аналіз сучасних медичних інформаційних систем України. Наука онлайн: Міжнародний електронний науковий журнал. 2018. №6 URL:https://nauka-online.com/ua/publications/tehnicheskie-nauki/2018/6/analiz-sovremennyh-meditsinskih-informatsionnyh-sistem-ukrainy/ (дата звернення: 18.05.2022)
6. Про технічні вимоги до електронної медичної інформаційної системи для її підключення до центральної бази даних електронної системи охорони здоров'я: Наказ Національної служби здоров'я України від 06.02.2019 № 28. URL: https://ehealth.gov.ua/wp-content/uploads/2021/08/Tehnichni-vymogy-v-redaktsiyi-nakazu-NSZU-346-vid-28.07.2021.pdf. (дата звернення: 15.05.2022)
7. Про діаграму прецедентів URL: https://www.quality-assurance-group.com/use-case-diagrams/ (дата звернення: 18.05.2022)
8. Моделювання бізнес-процесів та архітектура програмного забезпечення URL: https://uk.upwiki.one/wiki/business_process_modeling (дата звернення: 18.05.2022)
9. Про Use Case URL: https://training.qatestlab.com/blog/technical-articles/what-

is-a-use-case-and-what-are-they-for/ (дата звернення: 19.05.2022)

10. Федасюк Д. В. , Яковина В. С., Сердюк П. В., Нитребич О. О. Методи побудови сценаріїв тестування програмного забезпечення на основі аналізу його змінних. *Інформаційні технології та комп'ютерна інженерія*. 2014. № 2. С. 50-58.
11. Архітектура та проєктування програмного забезпечення URL: https://learn.ztu.edu.ua/mod/book/view.php?id=278&chapterid=80 (дата звернення: 20.05.2022)
12. Графічний інтерфейс користувача URL: https://www.wikiwand.com/uk/Графічний_інтерфейс_користувача (дата звернення: 22.05.2022)
13. Прототипування у вебдизайні URL: https://tilda.education/courses/web-design/prototypes/ (дата звернення: 22.05.2022)
14. Структура вебзастосунку та базові принципи URL: https://ru.weblium.com/blog/struktura-saita-chto-eto-kak-sozdat (дата звернення: 23.05.2022)
15. Про концептуальну модель бази даних URL: https://uk.theastrologypage.com/conceptual-data-model (дата звернення: 23.05.2022)
16. Про HTML та його базові поняття URL: https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/HTML_basics (дата звернення: 24.05.2022)
17. Про CSS та його базові поняття URL: https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics (дата звернення: 24.05.2022)
18. Препроцесори CSS та їх різновиди URL: https://mrmlnc.gitbooks.io/less-guidebook-for-beginners/content/chapter_1/css-reprocessors.html (дата звернення: 24.05.2022)
19. Методологія БЕМ URL: https://ru.bem.info/methodology/ (дата звернення: 24.05.2022)

20. Про JavaScript у веброзробці URL: https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/JavaScript_basics (дата звернення: 24.05.2022)
21. Про React та головні принципи роботи URL: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs (дата звернення: 24.05.2022)
22. Про NodeJs та головні принципи роботи URL: https://www.tutorialspoint.com/nodejs/nodejs_introduction.html (дата звернення: 25.05.2022)
23. Про Firebase та головні принципи роботи URL: https://www.techtarget.com/searchmobilecomputing/definition/Google-Firebase (дата звернення: 25.05.2022)

ДОДАТОК А

Програмний код класу Admin

```
const Admin = () => {
  const [activeTab, setActiveTab] = useState("Admin");
  const [error, setError] = useState("");
  const { currentUser, logout } = useAuth();
  const navigate = useNavigate();
  async function handleLogout() {
    setError("");
    try {
      await logout();
      navigate.push("/login");
    } catch {
      setError("Помилка");
    }
  }
  return (
    <div>
      <div>
        <ul>
          <li>
            <Link className="link" to={"addPacient"}>
              <p onClick={() => setActiveTab("Add Pacient")}>Додати пацієнта</p>
            </Link>
          </li>
          <li>
            <Link className="link" to={"listPacient"}>
              <p onClick={() => setActiveTab("List Pacient")}>
                Список пацієнтів
              </p>
            </Link>
          </li>
          <li>
            <Link className="link" to={"addDoctor"}>
              <p onClick={() => setActiveTab("Add Doctor")}>Додати лікаря</p>
            </Link>
          </li>
          <li>
            <Link className="link" to={"listDoctor"}>
              <p onClick={() => setActiveTab("List Doctor")}>Список лікарів</p>
            </Link>
          </li>
          <li>
            <Link className="link" to={"addAssistant"}>
              <p onClick={() => setActiveTab("Add Assistant")}>
                Додати лаборанта
              </p>
            </Link>
```

```
            </li>
            <li>
              <Link className="link" to={"listAssistant"}>
                <p onClick={() => setActiveTab("List Assistant")}>
                  Список лаборантів
                </p>
              </Link>
            </li>
            <li>
              <Link className="link" to={"addInfoPills"}>
                <p onClick={() => setActiveTab("Add Pills")}>
                  Додати інформацію про ліки
                </p>
              </Link>
            </li>
            <li>
              <Link className="link" to={"listInfoPills"}>
                <p onClick={() => setActiveTab("List Info Pills")}>
                  Список ліків
                </p>
              </Link>
            </li>
            <li>
              <Link className="link" to={"/"}>
                {" "}
                <p onClick={() => setActiveTab("List Info Pills")}>Вийти</p>
              </Link>
            </li>
          </ul>

          <Routes>
            <Route path="/addPacient" element={<AddPacient />}></Route>
            <Route path="/listPacient" element={<ListPacient />}></Route>
            <Route path="/addDoctor" element={<AddDoctor />}></Route>
            <Route path="/listDoctor" element={<ListDoctor />}></Route>
            <Route path="/addAssistant" element={<AddAssistant />}></Route>
            <Route path="/listAssistant" element={<ListAssistant />}></Route>
            <Route path="/addInfoPills" element={<AddInfoPills />}></Route>
            <Route path="/listInfoPills" element={<ListInfoPills />}></Route>
          </Routes>
        </div>
      </div>
    );
};

export default Admin;
```

Програмний код додавання інформації про пацієнта

```
const initialState = {
  name: "",
  phone: "",
  birthday: "",
  sex: "",
  contract: "",
};
const AddPacient = () => {
  const [state, setState] = useState(initialState);
  const { name, phone, birthday, sex, contract } = state;
  const navigate = useNavigate();
  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setState({ ...state, [name]: value });
  };
  const handleSubmit = (e) => {
    e.preventDefault();

    if (!name || !phone || !birthday || !sex || !contract) {
      toast.error("Заповніть поля");
    } else {
      fireDb.child("pacients").push(state, (err) => {
        if (err) {
          toast.error(err);
        } else {
          toast.success("Успіх!");
        }
      });
      setTimeout(() => navigate("/admin"), 500);
    }
  };
  return (
    <div>
      <h2>Додати нового пацієнта</h2>
      <form
        style={{
          margin: "auto",
          padding: "15px",
          maxWidth: "400px",
          alignContent: "center",
        }}
        onSubmit={handleSubmit}
      >
        <label htmlFor="name">ПІБ</label>
        <input
          type="text"
          id="name"
          name="name"
```

```
            placeholder="Прізвище, Імя, По батькові."
            value={name}
            onChange={handleInputChange}
          ></input>
          <label htmlFor="name">Телефон</label>
          <input
            type="phone"
            id="phone"
            name="phone"
            placeholder="Мобільний телефон"
            value={phone}
            onChange={handleInputChange}
          ></input>
          <label className="birth" htmlFor="birthday">
            Дата Народження
          </label>
          <input
            className="birthInput"
            type="date"
            id="birthday"
            name="birthday"
            placeholder="Дата народження"
            value={birthday}
            onChange={handleInputChange}
          ></input>
          <label className="sex" htmlFor="sex">
            Стать
          </label>
          <input
            type="text"
            id="sex"
            name="sex"
            placeholder="Стать"
            value={sex}
            onChange={handleInputChange}></input>
          <label htmlFor="contract">Договір</label>
          <input
            type="text"
            id="contract"
            name="contract"           placeholder="Договір"
            value={contract}
            onChange={handleInputChange}
          ></input>
          <input type="submit" value={"Додати"}></input>
        </form>
      </div>
    );
};
export default AddPacient;
```

Програмний код роботи з функціями користувача

```
import React, { useContext, useState, useEffect } from "react"
import { auth } from "../firebase"
const AuthContext = React.createContext()
export function useAuth() {
  return useContext(AuthContext)
}
export function AuthProvider({ children }) {
  const [currentUser, setCurrentUser] = useState()
const [loading, setLoading] =  useState(true)
  function signup(email, password) {
    return auth.createUserWithEmailAndPassword(email, password)
  }
  function login(email, password) {
    return auth.signInWithEmailAndPassword(email, password)
  }
  function logout() {
    return auth.signOut()
  }
  useEffect(() => {
    const unsubscribe = auth.onAuthStateChanged(user => {
      setCurrentUser(user)
      setLoading(false)
    })
    return unsubscribe
  }, [])
  const value = {
    currentUser,
    login,
    signup,
    logout
  }
  return (
    <AuthContext.Provider value={value}>
      {!loading &&  children}
    </AuthContext.Provider>
  )
}
```

Програмний код класу Doctor

```
import React, { useState } from "react";
import { BrowserRouter as Router, Route, Routes, Link } from "react-router-dom";
import "../../css/admin.css";
import ListDoctorPacient from "../Doctor/listPacientDoctor/pacientsListDoctor";
import ListInfo from "../Pacient/info/ListInfo";
const Doctor = () => {
  const [activeTab, setActiveTab] = useState("Doctor");
  return (
    <div>
      <div>
        <ul>
          <li>
            <Link className="link" to={"listPacientDoctor"}>
              <p
                className={`${activeTab === "List Pacient"} ? "active": ""}`}
                onClick={() => setActiveTab("List Pacient")}
              >
                Список пацієнтів
              </p>
            </Link>
          </li>
          <li>
            <Link className="link" to={"listInfo"}>
              <p onClick={() => setActiveTab("List Pacient")}>Ліки</p>
            </Link>
          </li>
        </ul>
        <Routes>
          <Route
            path="/listPacientDoctor"
            element={<ListDoctorPacient />}
          ></Route>
          <Route path="/listInfo" element={<ListInfo />}></Route>
        </Routes>
      </div>
    </div>
  );
};

export default Doctor;
```

Програмний код маршрутизації застосунку

```
function AppRoute() {
  return (
    <Router>
      <Header />
      <div className="App d-flex algn-items-center justify-content-center" style={{
minHeight: "100vh" }}>
        <div className="w-100">
          <Routes>
            <Route path="/signup" element={<Signup />}></Route>
            <Route path="/" element={<Login />}></Route>
            <Route path="/admin/*" element={<Admin />}></Route>
            <Route path="/editPacient/:id" element={<EditPacient />}></Route>
            <Route path="/editDoctor/:id" element={<EditDoctor />}></Route>
            <Route path="/editAssistant/:id" element={<EditAssistant />}></Route>
            <Route path="/editInfoPills/:id" element={<EditInfoPills />}></Route>
            <Route path="/pacient/*" element={<Pacient />}></Route>
            <Route path="/doctorRecord/:id" element={<DoctorRecord />}></Route>
            <Route path="/doctor/*" element={<Doctor />}></Route>
            <Route path="/addIll" element={<AddIll />}></Route>
            <Route path="/editIll/:id" element={<EditIll />}></Route>
            <Route path="/viewIll/:id" element={<View />}></Route>
            <Route path="/viewSympt" element={<Vieww />}></Route>
            <Route path="/addSympt" element={<AddSympt/>}></Route>
            <Route path="/addPrescr" element={<AddPrescr/>}></Route>
            <Route path="/viewPrescr" element={<ViewPrescr />}></Route>
            <Route path="/viewRes" element={<ViewRes />}></Route>
            <Route path="/addAnalyzes" element={<AddAnalyzes />}></Route>
            <Route path="/assistant/*" element={<Assistant />}></Route>
          </Routes>
        </div>
      </div>
      <ToastContainer position="top-center"></ToastContainer>
        </Router>
  );
}
export default AppRoute;
```