Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)
Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))
Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: «Розроблення системи зберігання медичних препаратів засобами Angular»

Виконав студент 4 курсу, групи ІСТС-21
спеціальності:126 „Інформаційні системи
та технології"
(шифр і назва напрямку підготовки спеціальності)
Губицький М.М.
(прізвище, ініціали)
Керівник: Сало М.Ф., Борецька І.Б.
(прізвище, ініціали)
Рецензент: Мокрицька О.В.
(прізвище, ініціали)

Львів-2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший(бакалаврський)
Спеціальність 126 " Інформаційні системи та технології "

ЗАТВЕРДЖУЮ:
Завідувач кафедри ІСКМ
_____ Сторожук О.Л.
„ 06 " 02 2024року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Губицький Микола Миронович
(прізвище, ім'я, по батькові)

1.Тема бакалаврської роботи: «Розроблення системи зберігання медичних препаратів засобами Angular»

керівник роботи Сало М.Ф., Борецька І.Б. кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "6" лютого 2024 року, №С-87
2.Термін подання студентом проекту(роботи) 10 червня 2024 р.
3. Вихідні дані до проекту (роботи) Розробити програмне забезпечення для контролю терміну придатності ліків. Реалізувати зручний та простий інтерфейс для використання розроблених функцій проекту. Для розробки використати фреймворк «Angular JS» та «MongoDB».

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Стан проблемної області
Інформаційне та математичне забезпечення
Програмне та технічне забезпечення
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до доповіді .
6. Дата видачі завдання 7 лютого 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	07.02-20.02	виконано
2.	Постановка задачі і її формалізація	20.02-05.03	виконано
3.	Виконання вхідного етапу технології	05.03-25.03	виконано
4.	Реалізація головних класів проекту	25.03-14.04	виконано
5.	Виконання етапу відлагодження проекту	14.04.-10.05	виконано
6.	Виконання етапу впровадження та випуску бета-версії.	10.05.-02.06.	виконано
7.	Оформлення записки до дипломного проекту.	02.06.-10.06.	виконано

Студент _____ Губицький М.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Сало М.Ф.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Борецька І.Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 51 сторінок пояснювальної записки, 36 рисунків, 1 додаток, 16 джерел.

Дипломна робота зосереджена на створенні програмного забезпечення для організації та моніторингу прийому лікарських засобів, а також відстеження термінів їх придатності. Проект реалізовано за допомогою фреймворку «Angular JS». Для побудови бази даних застосовано документо-орієнтовану платформу «MongoDB». Додаток включає функцію реєстрації користувачів. Інтерфейс програми відображає список ліків з їх зображеннями та інформацією про терміни придатності.

Крім того, додаток забезпечує контроль за прийомом та дозуванням лікарських засобів через систему сповіщень. Актуальність проекту аргументована. Створений додаток відрізняється простим та зрозумілим інтерфейсом, що підходить навіть для людей похилого віку.

Ключові слова: Angular JS, MongoDB, програмне забезпечення, фреймворк.

ABSTRACT

The diploma project contains 51 pages of explanatory report, 36 images, 1 appendix, 16 sources.

The diploma project is focused on the creation of software for organizing and monitoring the administration of medicines, as well as tracking their expiration dates. The project was implemented using the Angular JS framework. The document-oriented platform "MongoDB" was used to build the database. The application includes a user registration function. The program interface displays a list of medications with their images and information about expiration dates.

In addition, the application provides control over the intake and dosage of medicines through the notification system. The relevance of the project is well-argued. The created application has a simple and clear interface that is suitable even for the elderly.

Keywords: Angular JS, MongoDB, software, framework.

ТЕХНІЧНЕ ЗАВДАННЯ

Створити програмне забезпечення для відстеження терміну придатності лікарських засобів. Для цього використовувати фреймворк «Angular JS». Для побудови бази даних застосувати «MongoDB». У функціонал включити можливість реєстрації та аутентифікації користувачів. Також розробити інтерфейс, який буде попереджати про наближення терміну придатності медикаментів і надсилати відповідні повідомлення на електронну пошту користувача.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

AWS — Amazon Web Services (хмарна платформа компанії Amazon).

BSON — Binary JSON (двійкове представлення JSON-документів).

CDC — Centers for Disease Control and Prevention (Центри контролю та профілактики захворювань США).

CLI — Command Line Interface (інтерфейс командного рядка).

DOM — Document Object Model (об’єктна модель документа).

FDA — Food and Drug Administration (Управління з контролю за продуктами і ліками США).

JWT — JSON Web Token (токен для аутентифікації та авторизації користувачів).

MVC — Model-View-Controller (архітектурний шаблон побудови програм).

POJO — Plain Old JavaScript Object (простий об’єкт JavaScript).

RDBMS — Relational Database Management System (реляційна система керування базами даних).

SPA — Single Page Application (односторінковий веб-додаток).

ВСТУП

Згідно з даними Управління з контролю за продуктами і ліками, 50% призначених лікарських засобів не приймаються відповідно до вказівок медичних працівників. Основною причиною цього є забудькуватість. Дослідження свідчать, що учасники відкладають прийом ліків у 80–85% випадків, а забувають прийняти їх у 44–46% випадків. Невиконання призначень лікарів щодо прийому ліків призводить до близько 125 000 смертей щороку.

Під час двотижневого експерименту було встановлено, що після нагадувань кількість людей, які забули прийняти ліки, зменшилася з 46% до 5%. Частка тих, хто затримував прийом ліків, знизилася з 85% до 18%.

Об’єктом цього дослідження є застосування фреймворку «Angular JS» для створення програмного забезпечення, спрямованого на контроль своєчасного прийому ліків та моніторинг терміну придатності медичних препаратів.

Метою роботи є розробка додатку, який надаватиме користувачам інформацію про терміни придатності ліків та допомагатиме контролювати своєчасний прийом, що підвищить ефективність та безпечність лікування.

Предметом дослідження є розробка веб-орієнтованого додатку із використанням фреймворку «Angular JS».

Практичне значення роботи полягає в застосуванні теоретичних знань та практичних навичок для створення програмного забезпечення, яке полегшить дотримання режиму лікування, автоматизує та відстежуватиме дозування ліків, а також контролюватиме термін їх придатності. Створений додаток відрізняється простим та інтуїтивно зрозумілим інтерфейсом, що робить його доступним навіть для людей похилого віку. Важливо зазначити, що додаток не містить реклами і є безкоштовним для користувачів.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. **Огляд проблемної області.**

Вчені вказують на зростаючу кількість доказів, що багато лікарських засобів, зокрема ті, що призначені для серця, проявляють кращу ефективність, якщо їх приймати в певний час доби. Стаття, що була опублікована у журналі European Heart Journal, є найбільшим дослідженням на сьогоднішній день і включає дані про 19 тисяч учасників. Учасників експерименту було поділено на дві групи: одні приймали таблетки вранці, а інші - перед сном. Науковці відстежували стан пацієнтів протягом наступних п'яти або більше років. В результаті дослідження вони прийшли до висновку, що у учасників, які приймали ліки ввечері, ризик смерті від серцевого нападу, інсульту або серцевої недостатності був вдвічі менший. Для вірусних захворювань важливо приймати ліки регулярно, щоб переконатися, що в організмі завжди є достатня концентрація препарату. Недостатнє приймання ліків може призвести до занадто низького рівня препарату в крові, що ускладнює ефективність його дії проти вірусу. Це може сприяти мутаціям вірусу і розвитку стійкості до лікарського засобу.

Більшість медикаментів рекомендується приймати на голодний шлунок (за годину до або через 2 години після їжі), оскільки вони можуть взаємодіяти з деякими продуктами харчування. Однак є препарати, які краще застосовувати безпосередньо після їжі, наприклад, протизапальні засоби та препарати заліза, що можуть подразнювати ніжну слизову оболонку шлунка і потребують захисту від шлункової кислоти.

Препарати для лікування захворювань органів ЖКТ слід вживати відповідно до графіку. Наприклад, засіб від печії або підвищеної кислотності рекомендується приймати за 40-50 хвилин до або після перекусу. Ферменти для травлення (наприклад, Фестал, Мезим) і пробіотики варто вживати під час приготування їжі.

Зазвичай лікарські засоби приймають від 1 до 3 разів на добу (кожні 4-8 години). Деякі нові препарати мають пролонгований ефект і потребують лише одного прийому на добу. Важливий часовий інтервал між дозами. Наприклад, парацетамол слід приймати не рідше одного разу за чотири години, оскільки інакше може виникнути ризик отримання токсичної дози.

Для інших ліків більш доцільним є прийом вранці. Це стосується діуретиків, наприклад, фуросеміду, який допомагає вивести зайву рідину, препаратів гормональної терапії і засобів з тонізуючим ефектом, таких як кофеїн, елеутерокок, женьшень і інші. З іншого боку, у вечірній час рекомендується приймати седативні та снодійні препарати, а також антигістамінні, ліки від астми, язв і гастриту.

А що нарахування лікарських препаратів, які знижують артеріальний тиск?

Система контролю кров'яного тиску в організмі залежить від таких гормонів, як ренін, ангіотензин і альдостерон. Вони реагують на різні сигнали, такі як низький кров'яний тиск або стресові ситуації, щоб регулювати об'єм крові і стан кровоносних судин. Цей механізм дозволяє підтримувати стабільний рівень кров'яного тиску, що є важливим для здоров'я серцево-судинної системи. Ці препарати проявляють більшу активність під час сну. Серед них є інгібітори ангіотензинперетворюючого ферменту (ІАПФ), наприклад, периндоприл, а також блокатори рецепторів ангіотензину (БРА), такі як ірбесартан. Ці засоби розширюють кровоносні судини, що сприяє зниженню кров'яного тиску. До цього часу медики і фармацевти часто радили пацієнтам приймати їх вранці, вважаючи, що це оптимальний час для лікування, особливо в умовах активного руху. Важливо також враховувати індивідуальні потреби та реакцію організму на препарати, оскільки правильний час прийому може значно впливати на їх ефективність та комфорт для пацієнта. Дослідження свідчать, що прийом препаратів від артеріального тиску вночі сприяє значному зниженню наслідків серцевих захворювань, включаючи менше випадків інсультів, серцевих нападів і

серцевої недостатності, порівняно з прийомом їх вранці. Це відкриття підкреслює важливість правильно підібраного часу для прийому ліків у практиці лікування артеріальної гіпертензії.

Іноді дуже важливо визначити оптимальний час для прийому ліків. Наприклад, метотрексат, який використовується для лікування ревматоїдного артриту та важкого псоріазу, потребує точного дотримання графіку прийому. Він має прийматись один раз на тиждень у той самий день, щоб забезпечити безпечне й ефективне лікування. Однак, якщо лікарський засіб приймати помилково щодня, це може викликати серйозні ускладнення, включаючи можливість смертельних наслідків.

За даними Центрів контролю та профілактики захворювань (CDC), необхідність точного дотримання режиму лікування є важливою для успішного контролю хронічних захворювань, зокрема серед тих, хто приймає статини (препарати для зниження холестерину). Від 25 до 50 відсотків пацієнтів припиняють лікування протягом першого року, що підвищує їхній ризик смерті до 25 відсотків.

Чимало пацієнтів з різних причин не дотримуються рекомендацій лікарів щодо прийому ліків. Наприклад, це може бути пов'язано з недорозумінням інструкцій, забудькуватістю, складністю схем прийому декількох лікарських засобів, неприємними побічними ефектами або відчуттям, що ліки не дають очікуваного результату. Окрім цього, вартість ліків також може бути фактором, що впливає на дотримання рекомендацій з лікування.

Прийом ліків за призначенням або дотримання лікувальних режимів є критично важливим для ефективного контролю хронічних захворювань, лікування тимчасових станів та підтримання загального довгострокового здоров'я та благополуччя. Особиста відповідальність за власне життя та здоров'я є ключовим аспектом в цьому процесі, оскільки кожен пацієнт має право та обов'язок брати активну участь у своєму лікуванні.

Отже, як пацієнтам не заблукати в системах прийому препаратів та забезпечити вчасне та правильне застосування лікарських засобів? Важливою частиною успішного лікування є розуміння інструкцій і рекомендацій лікаря, що сприяє досягненню максимальної ефективності та безпеки лікування.

Отож, ось декілька порад, які можуть сприяти вчасному і правильному прийому лікарських засобів:

- Регулярно приймайте ліки в один і той же час кожного дня.
- Пов’яжіть прийом ліків із щоденною рутиною, такою як чищення зубів або підготовка до сну. Важливо перевіряти, чи потрібно приймати ліки на повний або порожній шлунок перед вибором часу прийому.
- Ведіть «календар ліків» з пляшечками таблеток і записуйте кожен прийом дози.
- Використовуйте контейнер для таблеток з розділенням на різні часи доби, наприклад, на ранок, обід, вечір і ніч.
- Наповнюйте контейнер для таблеток в один і той же час кожного тижня, наприклад, щонеділі після сніданку.
- Придбайте ковпачки з таймером для пляшечок з таблетками і налаштуйте їх так, щоб вони спрацьовували під час наступної дози. Деякі коробки для таблеток також оснащені функціями таймера.

Проте, всі вищезазначені рекомендації можна замінити однією, що є особливо ефективною:

- Встановіть спеціальний додаток на свій смартфон, який буде нагадувати вам про час прийому ліків, контролювати необхідність дозування та загалом керувати станом вашої аптечки, включаючи наявність необхідних ліків та їхній термін придатності.

1.2. **Система контролю прийому медичних препаратів**

Для багатьох людей прийом ліків став необхідною частиною їхнього щоденного ритму, де лікарські засоби є ключовим елементом для лікування та покращення здоров’я. Використання ліків може значно поліпшити якість життя

та сприяти одужанню, проте важливо розуміти, що всі препарати, будь то за рецептом чи без нього, мають свої певні ризики та переваги.

Користь від прийому ліків виявляється у позитивних впливах, які пацієнт спостерігає, включаючи зниження рівня артеріального тиску, зцілення інфекцій або зменшення болю. Небезпека ліків - це шанс на те, що при їх застосуванні можуть виникнути непередбачувані або небажані наслідки для пацієнта. Ці наслідки можуть охоплювати менш важкі проблеми, наприклад розлади шлунка, або серйозніші ураження, такі як вплив на роботу печінки. Рекомендації Управління з контролю за продуктами і ліками (FDA) спрямовані на те, щоб допомогти пацієнтам об'єктивно оцінити можливі ризики та переваги при виборі конкретних препаратів для лікування.

Управління ризиками

Коли користь від ліків переважає відомі ризики, FDA розглядає препарат як безпечний для схвалення. Проте перед початком застосування будь-яких лікарських засобів важливо уважно взважити всі їхні переваги та потенційні ризики.

Існують різні види ризиків, пов'язаних з прийомом ліків:

- Шанс небажаної взаємодії лікарського засобу з їжею, напоями, дієтичними добавками (включаючи вітаміни та трави) або іншими ліками.
  Використання разом будь-яких з цих продуктів може збільшити
  ймовірність такої взаємодії.
- Можливість, що ліки можуть не виявитися ефективними.
- Потенційна можливість, що препарат може призвести до додаткових проблем чи ускладнень.

Щоб знизити ризики та максимально скористатися ліками, важливо інформувати свого лікаря про наступне:

- Повний перелік усіх лікарських засобів та дієтичних добавок, включаючи вітаміни та трави, які ви приймаєте, навіть ті, що застосовуєте рідко.
- Будь-які алергічні реакції або чутливість, які вас можуть стосувати.

- Фактори, які можуть впливати на вашу здатність приймати ліки, наприклад, проблеми з ковтанням.
- Ваші плани щодо вагітності або фактична вагітність, якщо такі є.

Важливо знати свої ліки:

- бренд і загальні назви
- як вони виглядають
- як їх правильно зберігати
- коли, як і як довго їх використовувати
- як і за яких умов ви повинні припинити їх використання
- що робити, якщо ви пропустили дозу
- що вони повинні робити і коли очікувати результатів
- побічні ефекти та взаємодії
- чи потрібні вам якісь тести чи моніторинг.

  Це допоможе забезпечити ефективне та безпечне лікування, підходяще саме для вас.

Уникати взаємодій

Перед початком застосування будь-яких нових лікарських засобів або дієтичних добавок, включаючи вітаміни або трав'яні добавки, важливо з'ясувати, як вони можуть взаємодіяти з тими, які ви вже приймаєте. Це включає оцінку можливих взаємодій з іншими ліками, дієтичними добавками, напоями та їжею, які ви вживаєте наразі. Такий підхід дозволяє максимально уникнути негативних наслідків та забезпечити безпечне та ефективне лікування.

Під час прийому будь-яких медикаментів важливо бути уважним до їхнього впливу на ваше тіло. Якщо ви помітили будь-які побічні ефекти, негайно повідомте про це свого лікаря. Також важливо відслідковувати покращення після прийому препаратів. Наприклад, ви можете зменшити можливість побічних ефектів, прийнявши ліки після їжі, що дозволяє мінімізувати вплив на шлунок. Такий підхід допомагає забезпечити ефективне та безпечне лікування.

Кожен пацієнт має самостійно оцінити, які потенційні ризики він готовий прийняти для досягнення своїх цілей здоров'я.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. MongoDB

У сучасному світі баз даних, найпоширеніші та найпопулярніші системи це RDBMS (Relational Database Management Systems). Вони забезпечують надійне зберігання і ефективний доступ до великих обсягів даних. Вибір правильної системи баз даних залежить від потреб проекту та специфіки даних, які обробляються. RDBMS, такі як SQL Server або Oracle, забезпечують структуровані дані з високою швидкістю і точністю запитів. З іншого боку, NoSQL бази даних, наприклад MongoDB чи CosmosDB, створені для роботи з невструктурованими або напівструктурованими даними, надаючи гнучкість у моделюванні даних і швидкості в обробці. Вибір між цими категоріями баз даних залежить від конкретних потреб проекту і вимог до продуктивності та надійності системи.

Сучасні бази даних NoSQL є не лише альтернативою традиційним SQL базам даних, але і пропонують широкий спектр функцій, які зазвичай доступні в реляційних системах управління базами даних. Вони здатні забезпечувати як горизонтальне, так і вертикальне масштабування, а також простий контроль над даними завдяки своєму простому дизайну. Бази даних NoSQL суттєво відрізняються від традиційних реляційних баз даних у підході до структури зберігання даних, що дозволяє розробникам оптимізувати зберігання даних згідно з потребами їхніх додатків. Цей тип баз даних відкриває нові можливості для зберігання і обробки даних, які часто не досяжні у традиційних СУБД.

Типи NoSQL

Сучасні бази даних NoSQL поділяються на чотири основні типи, кожен з яких має свої унікальні особливості і використовується залежно від конкретних

потреб проекту.

MongoDB — це інноваційна база даних документів, яка змінює підхід до зберігання і обробки даних в інтернет-додатках. Вона славиться своєю гнучкістю схеми, що особливо цінується серед розробників, які працюють за гнучкими методологіями розробки. MongoDB дозволяє швидко розпочати проект завдяки готовим драйверам для усіх важливих мов програмування, що уникне необхідності витрачати час на налаштування бази даних. Вона побудована на основі архітектури горизонтального масштабування, що робить її ідеальним вибором для розширених інтернет-проектів.

Записи в MongoDB зберігаються у вигляді документів у форматі BSON, що є двійковим представленням даних, і їх легко можна отримати у форматі JSON для використання в програмах. MongoDB, створена у 2007 році, швидко здобула популярність серед розробників усього світу, завдяки своїй простоті використання і масштабованості.

Ось простий JSON-документ, що містить інформацію про історичну фігуру:

```
{
  "_id": 1,
  "name": {
    "first": "Ada",
    "last": "Lovelace"
  },
  "title": "The First Programmer",
  "interests": ["mathematics", "programming"]
}
```

Документні бази даних відрізняються високою гнучкістю, що дозволяє змінювати структуру документів і зберігати частково завершені записи. Вони підтримують вбудовані документи, що спрощує зберігання пов'язаної інформації разом. Поля в документі виконують ту ж функцію, що й стовпці в реляційних базах даних SQL, і можуть індексуватися для покращення швидкодії пошуку.

З моменту свого започаткування MongoDB активно розвивається на основі гнучкої архітектури, яка дозволяє об'єднувати невеликі машини в одну систему, забезпечуючи високу продуктивність та ефективну обробку великих обсягів

даних. Основний акцент у розробці MongoDB завжди був зроблений на забезпечення високоякісного користувацького досвіду, що разом з іншими унікальними особливостями робить її першим вибором для розробників у всьому світі для різноманітних застосувань.

MongoDB використовує масштабовану архітектуру, що стала улюбленою серед розробників для створення високомасштабованих додатків зі змінюваними схемами даних. Як база даних документів, MongoDB спрощує зберігання як структурованих, так і неструктурованих даних. Її формат, аналогічний JSON, нативно взаємодіє з об'єктами в сучасних мовах програмування, що зроблює її природним вибором для розробників, оскільки вони можуть уникати нормалізації даних. MongoDB здатна обробляти великі обсяги даних і масштабуватися як по вертикалі, так і по горизонталі, щоб забезпечити оптимальну продуктивність при високих навантаженнях.

MongoDB приваблює компанії та команди розробників з різних сфер завдяки своїй спрощеній схемі даних і здатності швидко масштабуватися. Однією з ключових переваг, яка привертає користувачів, є можливість ефективно зберігати як структуровані, так і неструктуровані дані в форматі, що легко взаємодіє з сучасними мовами програмування. Крім того, MongoDB підтримує горизонтальне масштабування, що дозволяє обробляти великі обсяги даних і забезпечує високу доступність додатків, що розвиваються.

Модель документа

Модель даних документа є інноваційним рішенням для зберігання та маніпулювання інформацією в сучасних програмах. Цей підхід не лише спрощує розробку, але й дозволяє ефективно використовувати дані в різних контекстах, використовуючи мови програмування з широким спектром можливостей.

Параметри розгортання

MongoDB можна легко розгорнути в будь-якому великому публічному хмаровому сервісі, такому як AWS, Azure або Google Cloud, завдяки MongoDB Atlas. Користувачі мають можливість вибрати між різними версіями, такими як

Enterprise Advanced для великих центрів обробки даних або безкоштовна версія Community з відкритим кодом, яка ідеально підходить для розробників і стартапів.

Швидкість та лаконічність

MongoDB пропонує відмінний досвід для розробників, які швидко можуть розгорнути базу даних і розпочати програмування без зайвих труднощів.

Масштаб

Архітектура MongoDB, спрямована на горизонтальне масштабування, здатна ефективно обробляти великі обсяги як даних, так і трафіку.

Спільнота однодумців

MongoDB побудувала розширену та динамічну екосистему платформи. Завдяки глобальній спільноті розробників і консультантів, ви можете легко отримати підтримку та поради. Крім того, надається підтримка на корпоративному рівні, що дозволяє впевнено впроваджувати рішення на великій масштабі.

Які переваги MongoDB?

MongoDB завоювала популярність як одна з найбільш вимогливих баз даних у світі, оскільки вона спрощує процеси зберігання, управління та доступу до даних під час розробки додатків на різних мовах програмування. Вона надає розробникам інструменти для ефективного маніпулювання інформацією, що сприяє швидкому створенню високопродуктивних програмних рішень.

1. *Сила документно-орієнтованих баз даних*

MongoDB стала однією з інноваційних баз даних NoSQL, які з'явилися для відповіді на потреби в масштабуванні та швидкому розвитку сучасних додатків. Концепт NoSQL охоплює різноманітні підходи до зберігання даних, такі як документно-орієнтовані бази даних, стовпчасті системи та бази даних, що працюють у пам'яті, кожна з яких відповідає на унікальні вимоги до обробки та доступу до інформації в різноманітних програмних середовищах.

У MongoDB дані зберігаються у вигляді документів, які представлені у файлах BSON. Цей формат дозволяє отримати дані безпосередньо у форматі JSON, що має численні переваги:

- ✓ JSON є природним форматом для зберігання даних, що спрощує їх розуміння.
- ✓ Він підтримує структуровану та неструктуровану інформацію в одному документі.
- ✓ Можливість вкладати JSON дозволяє зберігати складні об’єкти даних у структурованому форматі.
- ✓ Гнучка та динамічна схема JSON спрощує додавання або зміну полів без проблем.
- ✓ JSON має гнучку та динамічну схему, тому додавання полів або залишення поля не є проблемою.
- ✓ Документи в MongoDB можуть безпосередньо відображатися на об’єкти у найбільш популярних мовах програмування.
- ✓ Для більшості розробників робота з JSON є простим та потужним способом опису та зберігання даних, що полегшує розробку програм.

Один з ключових аспектів MongoDB полягає в тому, що розробники мають повний контроль над схемою бази даних. Вони можуть вносити зміни та адаптувати структуру документів у процесі розвитку програми без необхідності звертатися до адміністратора бази даних. MongoDB надає можливість координувати і контролювати зміни в схемі документів за допомогою вбудованої перевірки схеми, що дозволяє забезпечити консистентність та відповідність даних під час їхнього зміщення або оновлення.

MongoDB використовує спеціальний формат даних BSON (Binary JSON), який був розроблений для підтримки більшого різноманіття типів даних, ніж звичайний JSON. Цей двійковий формат дозволяє ефективніше зберігати та

обробляти дані, що пришвидшує їх аналіз. Дані, збережені в форматі BSON, можуть бути легко індексовані та шукані, що значно підвищує продуктивність при доступі до великих обсягів інформації. MongoDB підтримує різноманітні методи індексації, включаючи текстові, числові, геопросторові та часткові індекси.

2. *Досвід користувача для розробників*

MongoDB підтримує широкий спектр мов програмування, включаючи C, C#, .NET, C++, Go, Java, JavaScript, PHP, Python, Ruby, Rust, Scala і Swift. Це робить її універсальним інструментом для розробників, які працюють у різних технологічних стеках. Зростаюча кількість бізнес-користувачів приєднується до спільноти MongoDB, що спонукає команду розробників постійно вдосконалювати і розширювати можливості бази даних. MongoDB активно впроваджує нові функції, спрямовані на підтримку потреб iT-відділів підприємств, надаючи першокласну підтримку та консультації своїм клієнтам.

За допомогою MongoDB Atlas, як бази даних у форматі сервісу в хмарі, MongoDB стає доступною для користувачів простіше, ніж будь-коли. За декілька клацань у веб-інтерфейсі можна налаштувати кластер і почати розробку програм майже миттєво.

MongoDB Atlas упрощує початок роботи для розробників у будь-якій великій загальнодоступній хмарі, а також спрощує процес перенесення локальних екземплярів MongoDB у хмару.

MongoDB Atlas також надає ряд потужних можливостей:

- ✓ MongoDB Atlas Search, що базується на пошуковій системі Lucene, для реалізації повнотекстового пошуку.
- ✓ MongoDB Realm, повністю керовані серверні служби для створення мобільних і веб-додатків.

- ✓ Озеро даних MongoDB Atlas, що дозволяє розробникам легко запитувати та об'єднувати дані, збережені в Atlas, з даними з Amazon S3 або іншими сховищами HTTPS.

3. Масштабованість і трансакційність

Архітектура MongoDB, яка спрямована на горизонтальне масштабування, дозволяє розподіляти завдання між низкою менших серверів. Це забезпечує здатність вашої програми ефективно обробляти значні коливання у трафіку, що можуть виникнути зі зростанням вашого бізнесу.

Інженерні інновації MongoDB підтримують високу продуктивність для обробки величезної кількості операцій читання і запису. Основою цих інновацій є підхід до шардування, що дозволяє групувати кластери даних разом, коли інформація розподіляється по комп'ютерному кластеру. На відміну від більшості баз даних SQL, які покладаються на збільшення потужності окремих серверів, підхід MongoDB забезпечує гнучкість і масштабованість, розподіляючи навантаження між меншими серверами в кластері.

Під час моделювання даних у MongoDB часто використовується вбудовування об'єктів один в одного. Це дозволяє знижувати кількість транзакцій, необхідних для оновлення даних, що раніше вимагалося в традиційних реляційних базах даних. MongoDB спрощує процес взаємодії з даними, дозволяючи вбудовувати складні структури та здійснювати зміни за меншу кількість операцій, що полегшує і прискорює розробку та обслуговування додатків.

4. Зрілість платформи та екосистеми

MongoDB вже існує з 2007 року і здобула популярність у тисячах компаній завдяки своїй універсальності і гнучкості використання. Цей високий рівень прийняття спонукає платформу постійно розвиватися, щоб задовольнити ростучі вимоги користувачів. У світі великих організацій ключовим фактором є

можливість отримати швидку та ефективну підтримку, що стає критичним для будь-якої технології, що використовується в їхньому бізнесі.

MongoDB активно співпрацює з великою і динамічною спільнотою розробників у відкритому коді, академічному середовищі, а також серед системних інтеграторів та консультаційних компаній по всьому світу.

Отже, MongoDB — це універсальна база даних, яка пропонує рішення з численними перевагами для вашого процесу розробки програм. Її можливості масштабування та гнучка схема дозволяють створювати програми, що мають значні переваги. MongoDB забезпечує зручний інтерфейс для розробників з драйверами для більшості основних мов програмування та активною спільнотою користувачів.

2.2. Angular.js

Фреймворк Angular.js з відкритим вихідним кодом широко використовується для створення інтерактивних компонентів на веб-сайтах. Він отримав визнання завдяки своїй ефективності, простоті та великій гнучкості, що робить його популярним вибором для створення динамічних веб-додатків. Angular.js дозволяє розробникам легко структурувати та впроваджувати складні функціональність на веб-сайтах, забезпечуючи широкі можливості для розширення і підтримки.

Останні зміни в бізнес-середовищі змусили інвесторів і менеджерів шукати динамічні підходи до відповіді на попит на ринку. Серед актуальних тенденцій інтерфейсні додатки та односторінкові додатки виявилися найбільш ефективними інструментами для випередження конкурентів і задоволення потреб споживачів. У цьому контексті фреймворк Angular.js займає особливе місце, оскільки він підтримується Google і пропонує інтеграційне та модульне тестування, спрощуючи процес тестування веб-додатків. Серед компаній, які

успішно використовують Angular.js, можна відзначити YouTube, PayPal, Netflix, LinkedIn Corporation, Forbes і численних інших лідерів ринку.

Однією з ключових причин популярності та актуальності AngularJS є його інтуїтивно зрозумілий і декларативний інтерфейс, який спрощує процес розробки та забезпечує швидку і ефективну роботу програмістів.

Angular використовує HTML для визначення інтерфейсу веб-програм, що є набагато більш інтуїтивно зрозумілим і декларативним у порівнянні з програмуванням інтерфейсу через JavaScript. Такий підхід спрощує процес розробки і полегшує розуміння структури веб-додатків для розробників.

- Ефективне використання MVC

Численні фреймворки зазвичай вимагають від розробника розділення програми на компоненти MVC, що може бути часом зайвим та призводити до додаткової складності коду. У випадку Angular підхід до реалізації MVC є простим і ефективним: розробник просто розділяє програму на компоненти MVC, а Angular бере на себе обов'язок керувати зв'язками між ними. Це дозволяє зосередитися на логіці програми, не втрачаючи часу на деталі інтеграції компонентів.

- Менший код

Поскільки ми уникаємо написання коду для організації конвеєра MVC, наш код стає коротшим і менш складним. Крім того, ми використовуємо HTML для визначення представлень, що спрощує їх структуру. Механізм прив’язки даних дозволяє автоматизувати вставку даних у представлення без ручного введення. Фільтри в AngularJS дозволяють користувачам маніпулювати даними на рівні представлення без необхідності змінювати контролери.

- Моделі даних — це POJO

Моделі даних в Angular визначаються як прості об’єкти JavaScript (POJO), що спрощує їхнє використання. Вони не потребують визначення геттерів або

сеттерів, що дозволяє легко додавати або змінювати властивості без додаткових процедур обробки об’єктів чи масивів.

- Директиви

AngularJS має важливу функцію, що відома як директиви, що дозволяють розробникам створювати власні HTML-теги, які можуть бути як новими віджетами, так і спеціальними елементами. Крім цього, вони здатні ефективно маніпулювати атрибутами DOM, розширюючи можливості веб-інтерфейсів.

- Ін'єкція залежності

AngularJS включає потужну систему ін'єкції залежностей. Ін'єкція залежностей може бути розглянута як ключовий аспект програмної архітектури, який визначає, як компоненти отримують необхідні для своєї роботи ресурси або сервіси.

- Підсистема інжектора

AngularJS сприяє створенню компонентів і вирішенню їх взаємозалежностей, забезпечуючи гнучкість інтеграції з іншими частинами програми згідно з потребами проекту.

- Висока продуктивність

AngularJS вражає свою потужність завдяки надійним функціям, таким як фільтри, анімація, валідація форм, взаємодія з клієнтським API та маршрутизація. Ці інструменти не лише спрощують процес веб-розробки, але й підвищують її ефективність, забезпечуючи високу якість взаємодії з користувачем.

- Фільтри

AngularJS пропонує розробникам фільтри, які дозволяють ефективно форматувати дані, зберігаючи при цьому їх вихідний формат незмінним.

Використання різних типів фільтрів в Angular спрощує і стандартизує процес обробки даних, що робить код чистішим та більш легким для розуміння.

AngularJS відзначається своїм динамічним підходом до веб-розробки, що дозволяє створювати веб-додатки з високою продуктивністю і гнучкістю. Його структура не лише спрощує процес розробки, але й дозволяє швидко адаптуватися до змінних вимог ринку та користувацьких потреб.

AngularJS, незважаючи на його типове використання для SPA, здатний створювати різноманітні програми завдяки багатьом корисним функціям: двостороннє прив’язування, можливості шаблонування, обробка RESTful API, модуляризація, AJAX, ін’єкція залежностей та інше. Враховуючи зростання популярності JavaScript-орієнтованих рішень серед веб-дизайнерів і розробників, очікується, що використання Angular буде ще більшим і зручнішим у майбутньому, що стане чудовою новиною для всіх, хто працює у сфері розробки веб-продуктів.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Послідовний опис розробки веб-орієнтованої програми

Для розробки веб-орієнтованої програми з використанням Node.js і Angular JS потрібно налаштувати середовище для компіляції JavaScript коду за допомогою Node.js і Angular CLI (рис. 3.1). Це включає установку та конфігурацію середовища розробки, налаштування залежностей і встановлення необхідних пакетів для ефективної роботи з обома технологіями.

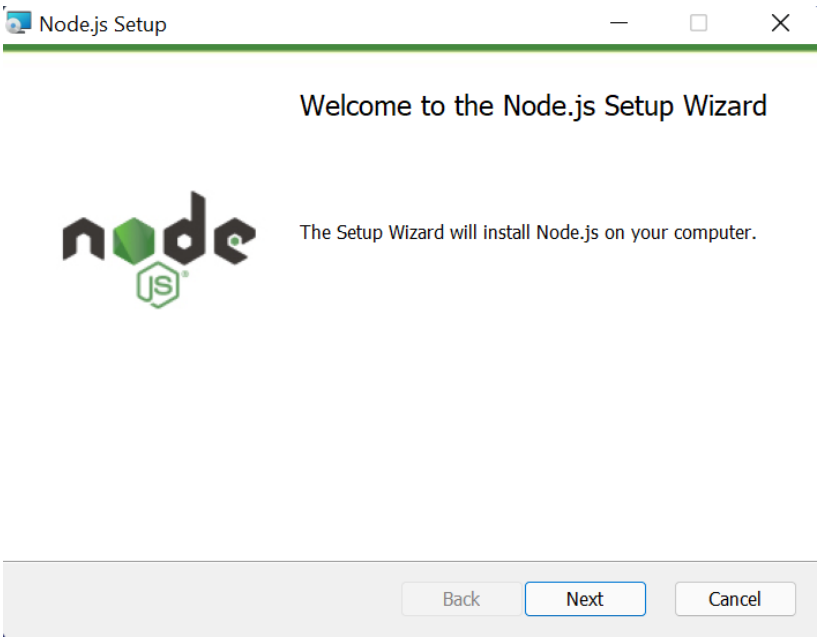

Рис. 3.1 Процес встановлення Node.js

Також важливо створити базу даних для цього за допомогою хмарного сервісу баз даних MongoDB (рис. 3.2). Цей крок забезпечить надійне зберігання даних і можливість швидкого доступу до них з будь-якого місця, що сприяє ефективній розробці та розгортанню програми.

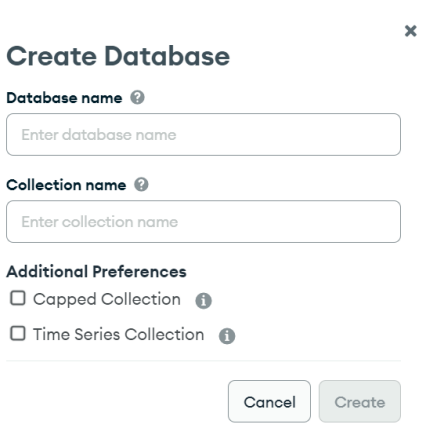

Рис. 3.2 Створення бази даних у MongoDB

Після цього етапу переходимо до розробки серверної частини застосунку з використанням Node.js (рис. 3.3). На початковому етапі створюємо скрипт для ініціалізації сервера та підключення до бази даних, що дозволяє забезпечити надійну роботу програми і ефективне управління даними.

```
const express = require('express')
const passport = require('passport')
const mongoose = require('mongoose')
const authRouter = require('./routes/auth-route')
const drugsRouter = require('./routes/drugs-route')
const PORT = process.env.PORT || 5000

// Mongo db link       You, 3 months ago • load server …
const db = 'mongodb+srv://StarScrap:Test101@cluster0.tyqpj.mongodb.net/drugs-shelf-life?retryWrites=true&w=majority';

const app = express()

app.use(passport.initialize())
require('./middleware/auth-middleware')(passport)
app.use('/uploads', express.static('uploads'))
// Routes
app.use(express.json())
app.use('/api/auth', authRouter)
app.use('/api/drugs', drugsRouter)

const start = async () => {
    try {
        await mongoose.connect(db)
        app.listen(PORT, () => console.log(`server started on port ${PORT}`))
    } catch (e) {
        console.log(e)
    }
}

start()
```

Рис. 3.3 Налаштування сервера

Після успішного запуску сервера наступним кроком є розробка моделі користувача та функції для валідації електронної адреси. Особлива увага при цьому приділяється перевірці наявності необхідних символів у введеній адресі, що забезпечує точність обробки даних та запобігає помилкам при роботі з інформацією користувачів (рис. 3.4).

Рис. 3.4 Модель користувача

Після успішного запуску сервера було створено модель лікарського засобу для подальшого використання в системі (рис. 3.5).

Рис. 3.5 Модель препарату

Після завершення створення моделей був розроблений клас «authController», який включає методи «registration» для процесу реєстрації користувачів і «login» для авторизації з використанням JWT токенів для забезпечення доступу (рис. 3.6).

```
class authController {

    async registration(req, res) {
        const candidate = await User.findOne({ email: req.body.email })
        if (candidate) {
            res.status(409).json({
                message: 'Такий email вже існує'
            })
        } else {
            const salt = bcrypt.genSaltSync(7)
            const password = req.body.password
            const user = new User({
                email: req.body.email,
                password: bcrypt.hashSync(password, salt)
            })
            try {
                await user.save()
                res.status(201).json(user)
            } catch (e) {
                console.log(e)
                res.status(400)
                    .json({ message: 'Register error' })
            }
        }
    }


    async login(req, res) {
        try {
            const { email, password } = req.body    // Get input User data
            const user = await User.findOne({ email })     // User name validation
            if (!user) {
                return res.status(400)
                    .json({ message: `Користувач ${email} не знайдений` })
            }
            if (email !== user.email) {     // Email validation
                return res.status(400)
                    .json({ message: `Введено невірний email` })
            }
            const validPassword = bcrypt.compareSync(password, user.password)     // Pasword validation
            if (!validPassword) {
                return res.status(400)
                    .json({ message: `Введено невірний пароль` })
            }
            const token = generateAccessToken(user._id)   // Generate JWT for user
            return res.json({ token: `Bearer ${token}` })    // Send JWT to the client
        } catch (e) {
            console.log(e)
            res.status(400)
                .json({ message: 'Login error' })
        }
    }


}
```

Рис. 3.6 Контролер авторизації

Далі був розроблений клас «drugsController» для управління лікарськими засобами, в якому реалізовано метод «addDrug», що відповідає за збереження нового лікарського засобу у базі даних (рис. 3.7).

```
async addDrug(req, res) {

    try {
        const { drugName, dateExpiring, reminderTime } = req.body;    // Get input data from for
        const userId = req.user.id
        const drugImage = req.file ? req.file.path : ''
        const drug = new Drug({ userId, drugName, drugImage, dateExpiring, reminderTime })    //
        await drug.save()       // Save new Drug in DB
        return res.json({ message: "Лік успішно додано" })
    } catch (e) {
        console.log(e)
        res.status(400)
            .json({ message: 'Add drug error' })
    }

}
```

Рис. 3.7 Метод «додати лікарський засіб»

Був створений метод «getDrugs», який дозволяє отримувати список усіх лікарських засобів користувача з бази даних (рис. 3.8).

```
async getDrugs(req, res) {
    try {
        const drugs = await Drug.find({ userId: req.user.id })
        return res.json(drugs)
    } catch (e) {
        console.log(e)
        res.status(400)
            .json({ message: 'Get drugs error' })
    }

}
```

Рис. 3.8 Метод «отримати лікарські засоби»

Був реалізований метод «getDrug», який забезпечує можливість отримати детальні дані про конкретний лікарський засіб із бази даних (рис. 3.9).

```
async getDrug(req, res) {
    try {
        const drug = await Drug.findById({ _id: req.params.id })
        if (drug) {
            return res.json(drug)
        } else {
            return res.status(400)
                .json({ message: 'Ліу не знайдено' })
        }
    } catch (e) {
        console.log(e)
        res.status(400)
            .json({ message: 'Get drug error' })
    }
}
```

Рис. 3.9 Метод «отримати лікарський засіб»

Був доданий метод «patchDrug», який дозволяє редагувати інформацію про конкретний лікарський засіб в системі (рис. 3.10).

```
async patchDrug(req, res) {
    try {
        const { drugName, dateExpiring, reminderTime } = req.body;    //  Get
        const _id = req.params.id
        const drugImage = req.file ? req.file.path : ''
        const drug = await Drug.findByIdAndUpdate(
            _id,
            { drugName, drugImage, dateExpiring, reminderTime },    // Find a
            { new: true }
        )
        if (drug) {
            return res.json({ drugImage,  message: "Дані успішно змінено" })
        } else if (!drug) {
            return res.status(400)
                .json({ message: "Лік не знайдений" })
        }
        if (!drug) { }
    } catch (e) {
        console.log(e)
        res.status(400)
            .json({ message: 'Patch drug error' })
    }
}
```

Рис. 3.10 Метод «змінити лікарський засіб»

На завершення було реалізовано метод «deleteDrug», який призначений для видалення відомостей про обраний лікарський засіб (рис. 3.11).

```
async deleteDrug(req, res) {
    try {
        const _id = req.params.id       // Get product id from URL
        const drug = await Drug.findByIdAndRemove({ _id })      //
        if (drug) {
            return res.json({ message: "Лік успішно усунуто" })
        } else if (!drug) {
            return res.status(400)
                .json({ message: "Лік не знайдений" })
        }
    } catch (e) {
        console.log(e)
        res.status(400)
            .json({ message: 'Remove drug error' })
    }
}
```

Рис. 3.11 Метод «видалити лікарський засіб»

Після завершення створення контролера для лікарських засобів, ми перейшли до налаштування маршрутів для обробки запитів з клієнтської частини застосунку. Початково були створені маршрути для авторизації користувача (рис. 3.12).

```
const Router = require('express')       You, 3 months ago • load server …
const router = new Router()
const controller = require('../controllers/auth-controller')

// Middleware validation method
const { check } = require("express-validator")




// Register
router.post('/register', [
    // Candidat input data validation
    check('email', "Email користувача не може бути пустим").notEmpty(),
    check('password', "Пароль повинен бути більше 4 і менше 10 символів").isLength({ min: 4, max: 10 })
], controller.registration)

// Login
router.post('/login', controller.login)


module.exports = router
```

Рис. 3.12 Маршрути для авторизації

Додатково були створені маршрути для обробки лікарських засобів, які дозволяють клієнтській частині застосунку взаємодіяти з базою даних щодо інформації про медикаменти (рис. 3.13).

```
const Router = require('express')       You, 3 weeks ago • drugs-shelf-life …
const passport = require('passport')
const upload = require('../middleware/upload')
const router = new Router()
const controller = require('../controllers/drugs-controller')

// Add drug
router.post('/add-drug', passport.authenticate('jwt', {session: false}), upload.single('drugImage'), controller.addDrug)
// Get all drugs
router.get('/get-drugs', passport.authenticate('jwt', {session: false}), controller.getDrugs)
// Get single drug
router.get('/get-drug/:id', passport.authenticate('jwt', {session: false}), controller.getDrug)
// Cheng drug data
router.patch('/patch-drug/:id', passport.authenticate('jwt', {session: false}), upload.single('drugImage'), controller.patchDrug)
// Remove drug
router.delete('/delete-drug/:id', passport.authenticate('jwt', {session: false}), controller.deleteDrug)

module.exports = router
```

Рис. 3.13 Маршрути для лікарських засобів

Після цього кроку важливо створити middleware для забезпечення безпеки аутентифікації користувачів за допомогою JWT токенів та використання пакету "Passport". Це забезпечить надійний захист конфіденційності та інформації користувача під час їхньої взаємодії з системою (рис. 3.14).

Рис. 3.14 Проміжний шар для аутентифікації користувача

Крім того, був реалізований middleware для завантаження зображень лікарських засобів за допомогою бібліотеки "Multer". Цей middleware дозволяє користувачам завантажувати фотографії на сервер, після чого URL посилання на зображення зберігається в базі даних для подальшого використання (рис. 3.15).

```
const multer = require('multer')
const path = require('path')

const storage = multer.diskStorage({
    destination(req, file, cb) {
        cb(null, 'uploads/')
    },
    filename(req, file, cb) {
        cb(null, Date.now() + path.extname(file.originalname))
    }
})

const fileFilter = (req, file, cb) => {
    if (file.mimetype === 'image/png' || file.mimetype === 'image/jpeg' || file.mimetype === 'image/jpg') {
        cb(null, true)
    } else {
        cb(null, false)
    }
}

const limits = {       You, last week • drugs-shelf-life …
    fileSize: 1024 * 1024 *  5
}

module.exports = multer({
    storage: storage,
    fileFilter: fileFilter,
    limits: limits

})
```

Рис. 3.15 Проміжний шар для завантаження фотографій

На даному етапі ми успішно завершили розробку серверної частини додатку. Тепер ми переходимо до створення клієнтської частини застосунку з використанням фреймворку AngularJS. Першим кроком буде створення сервісів, які дозволять нам формувати запити до сервера для взаємодії з даними та ресурсами, необхідними для коректної роботи нашого додатку (рис. 3.16, рис. 3.17).

```
private token!: string

constructor(private http: HttpClient) { }

// Register request
register(user: User): Observable<User> {
  return this.http.post<User>('/api/auth/register', user)
}

// Login request
login(user: User): Observable<{ token: string }> {
  return this.http.post<{ token: string }>('/api/auth/login', user)
    .pipe(tap(
      ({ token }) => {
        localStorage.setItem('auth-token', token)
        this.setToken(token)
      }
    ))
}

// Save jwt metod
setToken(token: string) {
  this.token = token
}

// Get jwt metod
getToken(): string {
  return this.token
}

// Check if user is authoraized
isAuthenticated(): boolean {
  return !!this.token
}

// Destroy token metod
logout() {
  this.setToken(" ")
  localStorage.clear()
}
```

Рис. 3.16 Сервіси для реєстрації та управління авторизацією користувача

```
constructor(private http: HttpClient) { }

// Post drug
post(drug: Drug) {
  console.log(drug)
  return this.http.post('/api/drugs/add-drug', drug)
    .pipe(tap(console.log))
}
    You, 3 weeks ago • drugs-shelf-life …
// Get drugs
get(): Observable<Drug[]> {
  return this.http.get<Drug[]>('/api/drugs/get-drugs')
    .pipe(tap(console.log))
}

// Get drug by id
getById(id: string): Observable<Drug> {
  return this.http.get<Drug>(`/api/drugs/get-drug/${id}`)
    .pipe(tap(console.log))
}

// Puthc drug
pathch(id: string, drug: Drug): Observable<Drug> {
  return this.http.patch<Drug>(`/api/drugs/patch-drug/${id}`, drug)
    .pipe(tap(console.log))
}

// Puthc durgImage
pathchImage(id: string, file: any): Observable<Drug> {
  return this.http.patch<Drug>(`/api/drugs/patch-drug/${id}`, file)
    .pipe(tap(console.log))
}

// Delete drug
delete(id: string) {
  return this.http.delete(`/api/drugs/delete-drug/${id}`)
    .pipe(tap(console.log))
}
```

Рис. 3.17 Сервіси для виконання операцій CRUD з даними лікарських засобів

Після цього ми приступили до створення вкладки для реєстрації та авторизації користувачів. Нова вкладка дозволить користувачам не лише створювати облікові записи та входити в систему, але й управляти своїми персональними налаштуваннями та взаємодіяти з іншими функціями додатку зручним та ефективним способом (рис. 3.18, рис. 3.19, рис. 3.20).

```
form!: FormGroup
aSub!: Subscription

constructor(
  private auth: AuthService,
  private router: Router
) { }

// Register form validators
ngOnInit(): void {
  this.form = new FormGroup({
    email: new FormControl(null, [Validators.required, Validators.email]),
    password: new FormControl(null, [Validators.required, Validators.minLength(4)])
  })
}

// Memory clean
ngOnDestroy() {
  if (this.aSub) {
    this.aSub.unsubscribe()
  }
}

// Send user register form data
onSubmit() {
  this.form.disable()
  this.aSub = this.auth.register(this.form.value).subscribe(
    () => {
      this.router.navigate(['/login'], {
        queryParams: {
          registered: true
        }
      })
    },
    error => {
      console.warn(error)
      this.form.enable()
    }
  )
}
```

Рис. 3.18 Код компоненту вкладки реєстрації

```
form!: FormGroup
aSub!: Subscription


constructor(
  private auth: AuthService,
  private router: Router,
  private route: ActivatedRoute) {
}

// Login form validators and ecees check
ngOnInit(): void {
  this.form = new FormGroup({
    email: new FormControl(null, [Validators.required, Validators.email]),
    password: new FormControl(null, [Validators.required, Validators.minLength(4)])
  })

  this.route.queryParams.subscribe((params: Params) => {      You, 3 months ago * auth-sys
    if (params['registered']) {

    } else if (params['eccessDenied']) {

    }
  })
}

// Memory clean
ngOnDestroy() {
  if (this.aSub) {
    this.aSub.unsubscribe(),
      () => this.router.navigate(['/login'])
  }
}

// Send user login data
onSubmit() {
  this.form.disable()
  this.aSub = this.auth.login(this.form.value).subscribe(    // This.form.valut = user data
    () => this.router.navigate(['/drugs']),
    error => {
      console.warn(error)
      this.form.enable()
    }
  )
}
```

Рис. 3.19 Код компоненту вкладки авторизації

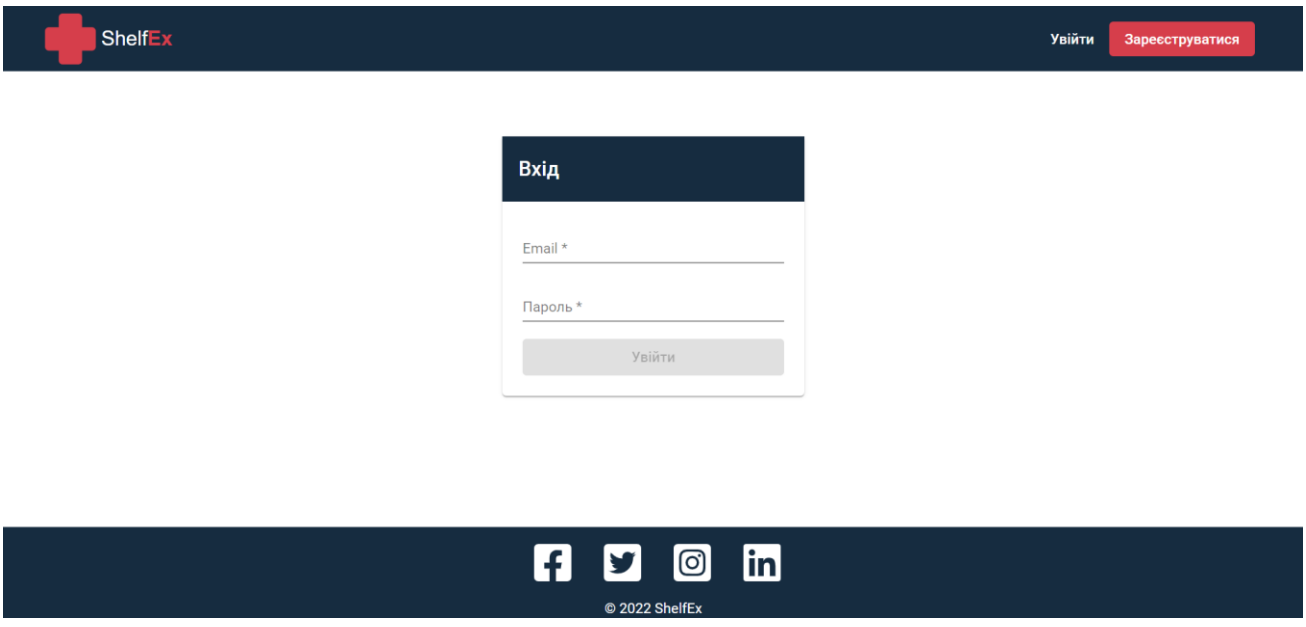

Рис. 3.20 Вкладка для реєстрації та авторизації

Також було розроблено основну сторінку, яка відкривається користувачеві після успішної реєстрації або входу в систему (рис. 3.21). Ця сторінка включає в себе не лише основний інтерфейс додатку, але й основні функції управління обліковим записом користувача та персоналізації досвіду використання програми (рис. 3.22 та рис. 3.23).

Рис. 3.21 Головна вкладка програми

```
@ViewChild('addImageInput') addInput!: ElementRef
@ViewChild('editImageInput') editInput!: ElementRef

opened!: boolean
drugs!: Drug[]
drug!: Drug
serchName!: string
order: any = 'none'
drugsAmount!: number
date = moment()
localTime: any
value = 0       You, last week • drugs-shelf-life …
imgsrc = 'http://localhost:5000/'
id!: any
image!: any



constructor(
  private http: drugsService,
  private dialog: MatDialog,
  private serchService: SerchService,
  private drugsAmountService: DrugsAmountService
) { }

// Get all Drugs
ngOnInit() {
  this.http.get().subscribe(···
    })
  this.serchService.sharedSerch.subscribe(serch => this.serchName = serch)
  this.drugsAmountService.amoutStatus.subscribe(drugsAmount => this.drugsAmount = drugsAmount)
  setInterval(() => {···
  }, 1000)
}

newStatus() {
  this.drugsAmountService.drugsAmountStatus(this.drugsAmount)
}

sortDrugs(value: number) {
  this.order = value;
}

openAddDrugDialog() {
  this.dialog.open(AddDrugDialogComponent)
}
```

Рис. 3.22 Код компоненту першої головної вкладки

```
openDeleteDrugDialog(id: any) {
  let dialogRef = this.dialog.open(DeleteDrugDialogComponent, {
    data: `<h3>Ви впевнені що хочете усунути лік?</h3>`
  })

  dialogRef.afterClosed().subscribe(res => {
    console.log(res.data)
    if (res.data) {
      this.deleteDrug(id)
    }
  })
}

openEditDrugDialog(drug: any) {
  this.dialog.open(EditDrugDialogComponent, {
    data: drug
  })
}

openReminderDialog(drug: any) {
  this.dialog.open(ReminderDialogComponent, {
    data: drug
  })
}

// Get Drug by id
getProducd(id: any) {
  this.http.getById(id).subscribe(
    (data) => {
      this.drug = data
    }
  )
}

// Delete Drug
deleteDrug(id: any) {
  this.http.delete(id)
    .subscribe()
  window.location.reload()
}

addClick(drug: Drug) {
  this.id = drug._id
  this.addInput.nativeElement.click()
}

editClick(drug: Drug) {
  this.id = drug._id
  this.editInput.nativeElement.click()
}

uploadFile(event: any) {
  this.image = event.target.files[0]
  let fd = new FormData()
  fd.append('drugImage', this.image, this.image.name)
  this.http.pathchImage(this.id, fd).subscribe()
  window.location.reload()
}
```

Рис. 3.23 Код компоненту другої головної вкладки

3.2. Послідовний опис використання веб-орієнтованою програмою

Початковим кроком для використання додатку є необхідність користувача пройти процедуру реєстрації або авторизації, введенням відповідних даних у спеціальну форму (рис. 3.24).

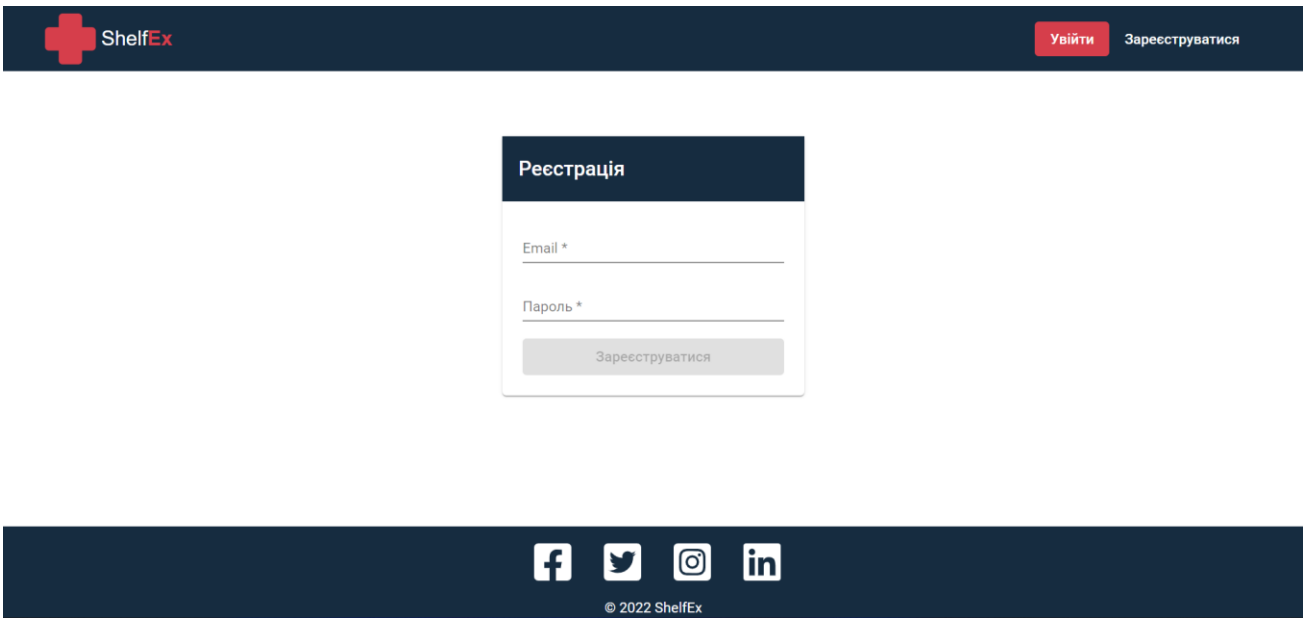

Рис. 3.24 Вкладка для реєстрації та авторизації

Після успішної авторизації користувач переходить на основний екран додатку, де може отримати доступ до усіх функцій та сервісів (рис. 3.25).

Рис. 3.25 Головна вкладка

Після цього користувач може додати новий лікарський засіб, натиснувши на кнопку з символом «Плюс». Це викличе появу діалогового вікна з формою, де потрібно вказати всі необхідні дані про лікарський засіб (рис. 3.26).

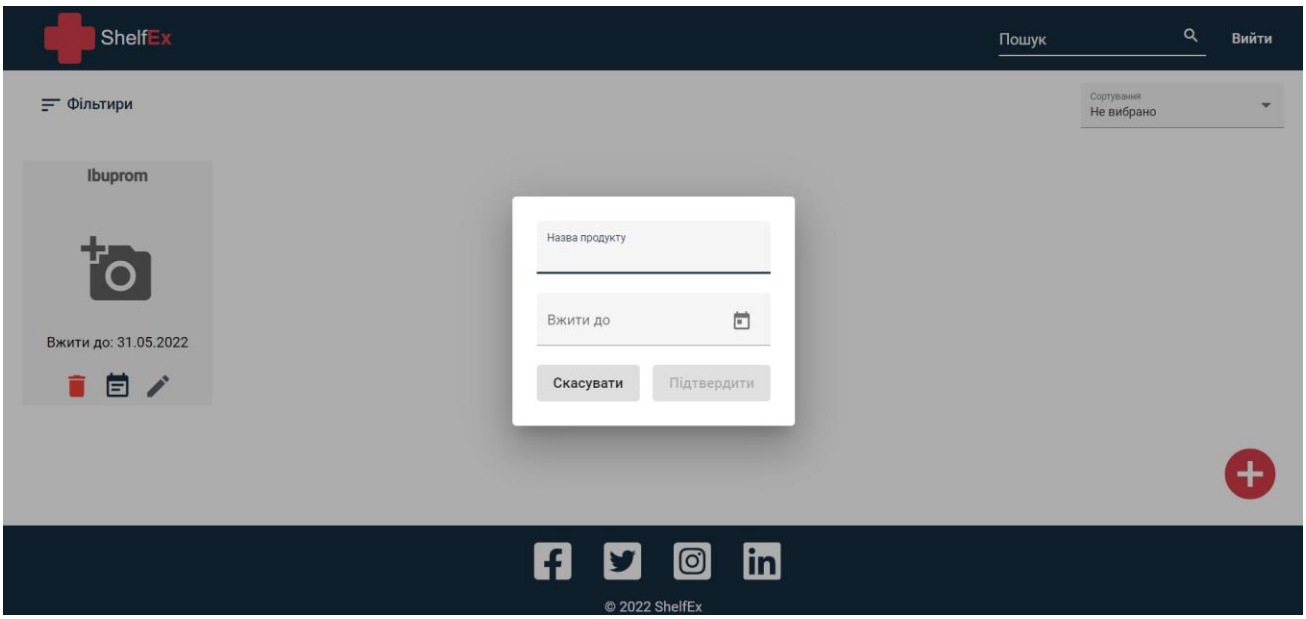

Рис. 3.26 Процес додавання нового лікарського засобу

Після додавання лікарського засобу користувач може змінити його дані, натиснувши на іконку з символом «Олівець». Це відкриє аналогічне вікно до того, що використовувалося при додаванні ліка, де можна буде внести необхідні зміни (рис. 3.27).

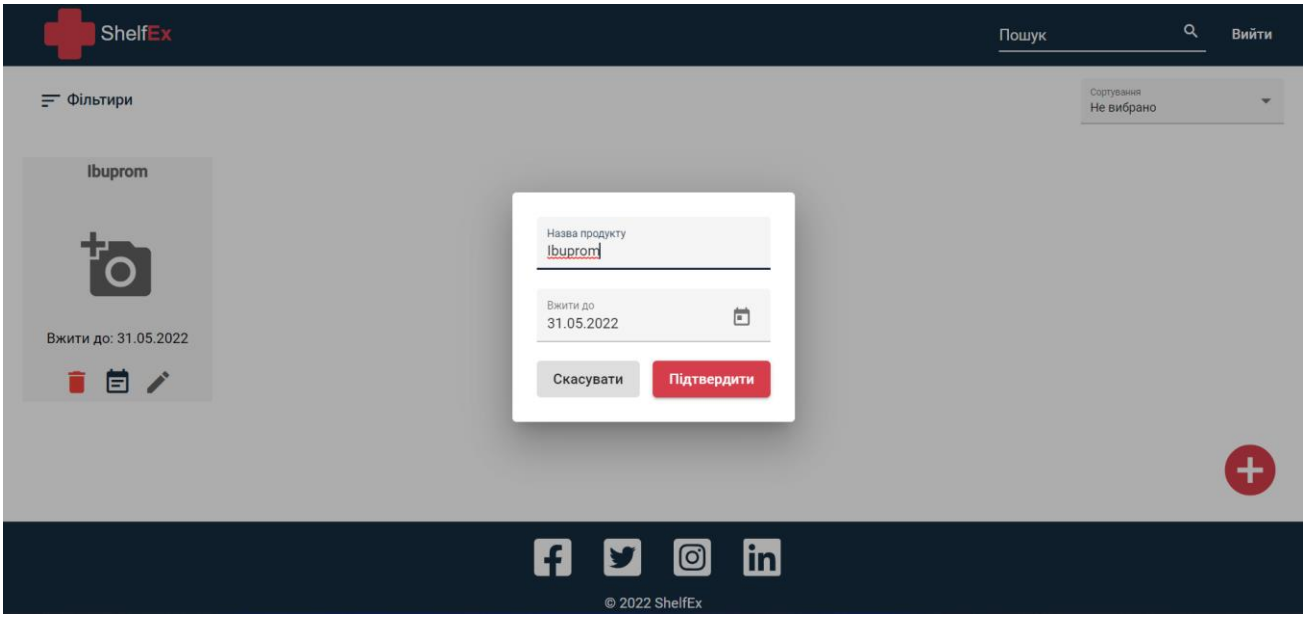

Рис. 3.27 Редагування даних лікарського засобу

Крім того, натиснувши на іконку, яка розташована ліворуч від піктограми "Олівець", користувач може скласти графік прийому лікарського засобу (рис. 3.28).

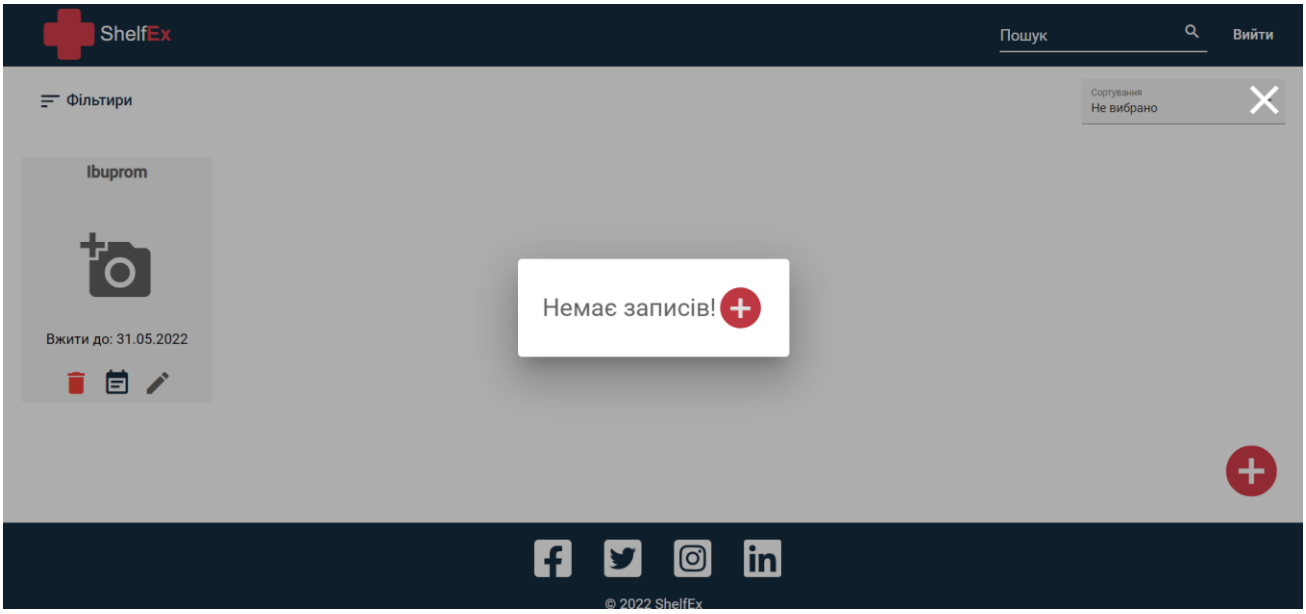

Рис. 3.28 Вікно графіку прийому препарату

Після цього ми натискаємо на кнопку з символом «Плюс», що розташована в цьому вікні, і обираємо час прийому для лікарського засобу (рис. 3.29).

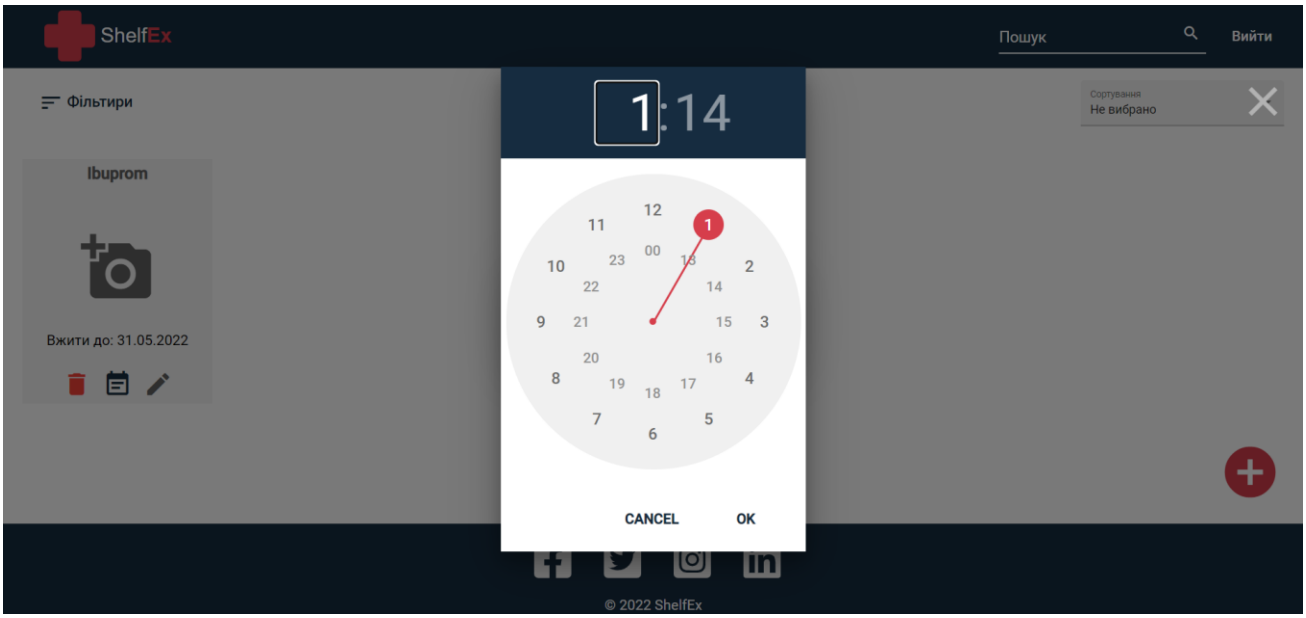

Рис. 3.29 Процес додавання часу прийому препарату

Після цього внизу з'являється новий запис щодо часу прийому препарату (рис. 3.30).

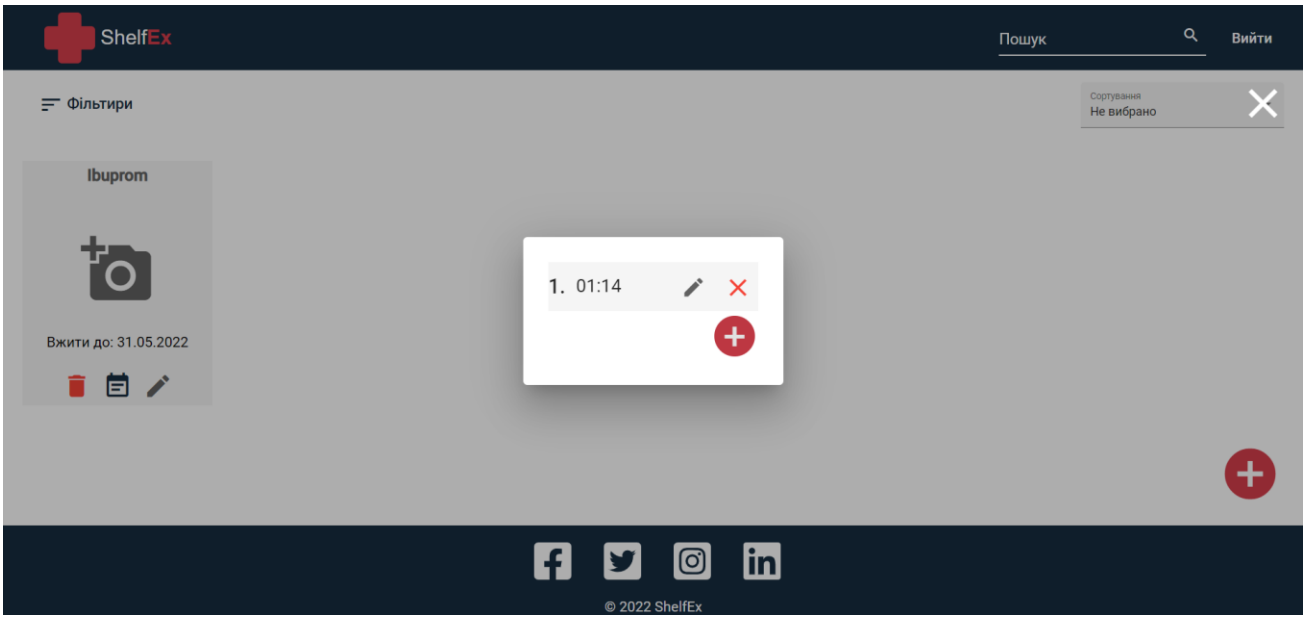

Рис. 3. 30 Налаштування часу прийому препарату

Якщо час прийому препарату збігається з поточним часом, то в правому верхньому куті іконки розкладу прийому препарату з'являється червоне коло зі знаком оклику, що нагадує користувачеві про необхідність взяти препарат (рис. 3.31).

Рис. 3. 31 Перше нагадування про прийом лікарського засобу

Такий самий символ з'являється поряд із часом прийому лікарського засобу в розкладі (рис. 3.32).

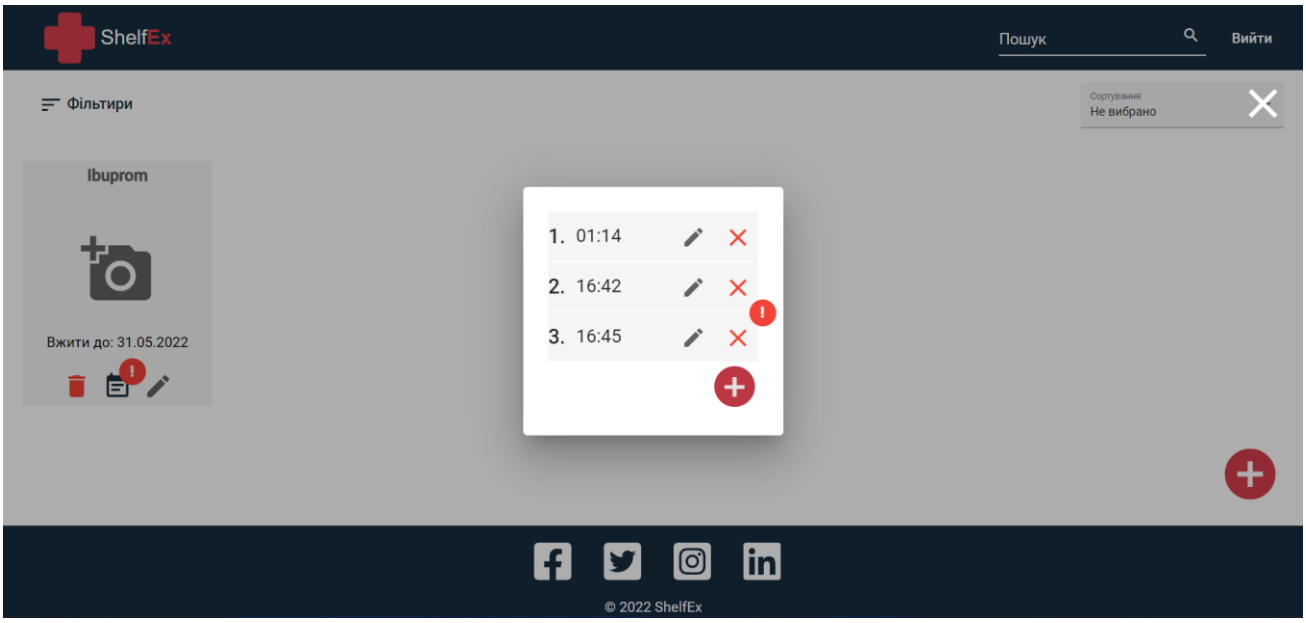

Рис. 3.32 Друге нагадування про прийом лікарського засобу

Користувач також може завантажити фотографію лікарського засобу, натиснувши на відповідну іконку "Додати фото", яка розміщена біля кожного лікарського засобу на головній сторінці (рис. 3.33).

Рис. 3.33 Процес додавання фотографії лікарського засобу

Для видалення лікарського засобу необхідно натиснути на червону іконку «Видалити» і після цього з’явиться вікно підтвердження, в якому користувачу потрібно підтвердити або скасувати видалення лікарського засобу.

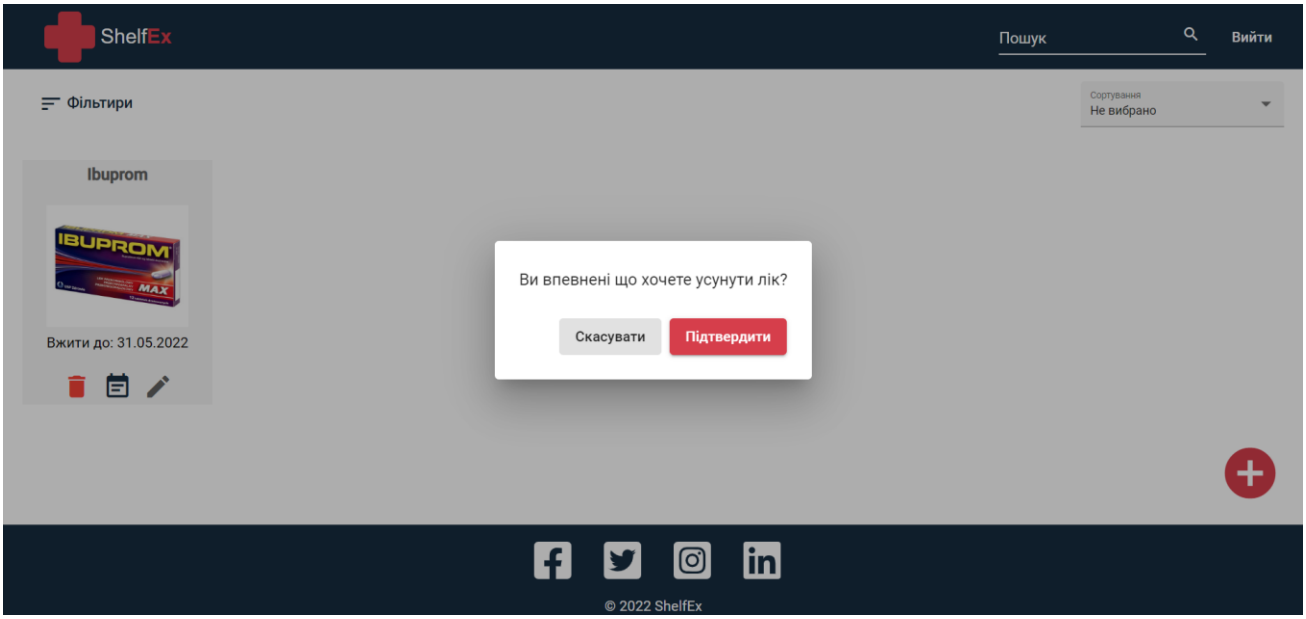

Рис. 3.34 Процес видалення лікарського засобу

У верхніх кутах екрану розташовані селектори, які дозволяють користувачу фільтрувати та сортувати продукти за терміном придатності.

Рис. 3. 35 Процес фільтрування лікарських засобів

У правому верхньому куті розташована спеціальна форма для швидкого пошуку лікарських засобів за їх назвою.

Рис. 3.36 Процес сортування лікарських засобів

ВИСНОВКИ

Ефективність лікарських засобів в значній мірі залежить від їх вчасного прийому, що дозволяє знизити ризик побічних реакцій. Наш додаток розроблено з метою забезпечення вчасного прийому медичних препаратів, використовуючи фреймворк Angular JS та базу даних MongoDB.

Однією з ключових функцій додатку є можливість користувачів складати індивідуальні графіки прийому лікарських засобів. Крім того, додаток надає можливість реєстрації користувачів та автентифікації, що підвищує зручність його використання. Інтерфейс додатку забезпечує вчасне нагадування про наближення до кінця терміну придатності лікарських засобів, що допомагає оптимально організувати домашню аптечку.

Крім того, наш застосунок може бути корисним для мереж аптек, які можуть використовувати його для контролю термінів придатності ліків, що пропонуються клієнтам. Ці функції разом з використанням передових технологій дозволяють нам забезпечити надійний та зручний сервіс для кожного користувача нашого додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Angular in Action" by Jeremy Wilken - Manning Publications, 2018, 400 сторінок.
2. "MongoDB: The Definitive Guide" by Kristina Chodorow - O'Reilly Media, 2013, 432 сторінок.
3. "Node.js Design Patterns" by Mario Casciaro - Packt Publishing, 2014, 454 сторінки.
4. "JavaScript: The Good Parts" by Douglas Crockford - O'Reilly Media, 2008, 176 сторінок.
5. "RESTful Web Services" by Leonard Richardson, Sam Ruby - O'Reilly Media, 2007, 448 сторінок.
6. "Pro AngularJS" by Adam Freeman - Apress, 2014, 688 сторінок.
7. "Learning Angular: A Hands-On Guide to Angular 2 and Angular 4" by Brad Dayley - Addison-Wesley Professional, 2017, 240 сторінок.
8. "Mastering MongoDB 4.x" by Alex Giamas - Packt Publishing, 2019, 420 сторінок.
9. "Node.js 8 the Right Way: Practical, Server-Side JavaScript That Scales" by Jim R. Wilson - Pragmatic Bookshelf, 2018, 344 сторінок.
10. "Eloquent JavaScript: A Modern Introduction to Programming" by Marijn Haverbeke - No Starch Press, 2018, 472 сторінки.
11. "Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems" by Martin Kleppmann - O'Reilly Media, 2017, 616 сторінок.
12. "Effective JavaScript: 68 Specific Ways to Harness the Power of JavaScript" by David Herman - Addison-Wesley Professional, 2012, 240 сторінок.
13. "The Complete Guide to RESTful APIs" by Mike Amundsen - Manning Publications, 2018, 300 сторінок.
14. "Building Single Page Applications with AngularJS" by Pawel Kozlowski - Smashing Magazine, 2014.

15."MongoDB: Scaling Out" by Kristina Chodorow - MongoDB, Inc., 2011.
16."Node.js: Using JavaScript on the Server" by Ryan Dahl - Google, 2009.

ДОДАТКИ

Index.js

```
const express = require('express');
const passport = require('passport');
const mongoose = require('mongoose');
const authRouter = require('./routes/auth-route');
const productsRouter = require('./routes/products-route');
const PORT = process.env.PORT || 5000;

// MongoDB link
const db = 'mongodb+srv://StarScrap:Test101@cluster0.tyqpj.mongodb.net/shelf-
life?retryWrites=true&w=majority';
const app = express();

// Middleware
app.use(express.json());
app.use(passport.initialize());
require('./middleware/auth-middleware')(passport);

// Routes
app.use('/api/auth', authRouter);
app.use('/api/products', productsRouter);

const start = async () => {
    try {
        await mongoose.connect(db, {
            useNewUrlParser: true,
            useUnifiedTopology: true,
            useFindAndModify: false,
            useCreateIndex: true
        });
        app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
    } catch (e) {
        console.error(e);
```

```
    }
};

start();
```

medecine-route.js

```
const Router = require('express');
const passport = require('passport');
const router = new Router();
const controller = require('../controllers/products-controller');

// Add product
router.post('/add-product', passport.authenticate('jwt', { session: false }), controller.addProduct);

// Get all products
router.get('/get-products', passport.authenticate('jwt', { session: false }), controller.getProducts);

// Get single product
router.get('/get-product/:id', passport.authenticate('jwt', { session: false }), controller.getProduct);

// Update product data
router.patch('/patch-product/:id', passport.authenticate('jwt', { session: false }),
controller.patchProduct);

// Remove product
router.delete('/delete-product/:id', passport.authenticate('jwt', { session: false }),
controller.deleteProduct);

module.exports = router;
```

Auth-route.js

```
const { Router } = require('express');
const router = Router();
const { check, validationResult } = require("express-validator");
const controller = require('../controllers/auth-controller');
```

```
// Registration
router.post('/register', [
    check('email', "Email користувача не може бути пустим").notEmpty(),
    check('password', "Пароль повинен бути більше 4 і менше 10 символів").isLength({ min: 4,
max: 10 }),
], controller.registration);

// Login
router.post('/login', controller.login);

module.exports = router;
```