МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій

(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерної алгебри та дискретної математики

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»

(освітньо-кваліфікаційний рівень)

на тему

«Система моніторингу працівників фармацевтичної компанії»

«Система мониторинга сотрудников фармацевтической компании»

«Monitoring system for employees of a pharmaceutical company»

Виконав: студентка денної форми навчання
напряму підготовки 123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)

Позднікова Анастасія Олексіївна
(прізвище, ім'я, по-батькові)

Керівник канд. фіз.-мат. наук, доцент Варбанець С.П.
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент канд. техн. наук, доцент Якімова Н.А.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___від «___»_____2021 р.

Завідувач кафедри

_____ Варбанець П.Д.
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол №___від «___»_____2021 р.

Оцінка / /
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК_____

_____ _Н.Ф.Казакова_
(підпис) (прізвище, ініціали)

Одеса – 2021

АНОТАЦІЯ

Мета даної дипломної роботи - розробка системи моніторингу співробітників фармацевтичної компанії. Особливістю даної системи є створення адміністративної частини, з урахуванням надможливостей для керівника, таких як додавання графіку роботи, співробітництва з новими мережами аптек, відстеження статистичних даних щодо виконання робочого плану, прибутків компанії і так далі. Також розробка окремого кабінету співробітника, який може переглянути свій графік і дізнатися заробітну плату, не звертаючись до інших програм, або джерел.

Додатково буде проведено аналіз різних завдань, що вирішуються у визначеній предметній області, наприклад, визначення середньомісячного доходу підприємства за певний місяць, підрахунок найбільшого доходу по аптекам, визначення співробітників, що мають вагомість, а також співробітників, що не виконують робочого плану.

Результатом дипломної роботи є інформаційна система зі зручним, логічним і дружелюбним інтерфейсом для користувача, що сприймається без додаткових зусиль.

АННОТАЦИЯ

Цель данной дипломная работа - разработка системы мониторинга сотрудников фармацевтической компании. Особенностью данной системы является создания административного части, с учетом сверхвозможностей для руководителя, таких как добавление графика работы, сотрудничества с новыми сетями аптек, отслеживание статистических данных касательно выполнения рабочего плана, прибыли компании и так далее. Также разработка отдельного кабинета сотрудника, который может просмотреть свой график и узнать заработную плату, не обращаясь к сторонним программам, или источникам.

Дополнительно будет проведен анализ разных задачи, что решаются в определенной предметной области, например, определение среднемесячного дохода предприятия за определенный месяц, подсчет наибольшего дохода по аптекам, определение сотрудников, которые имеют значимость, а также работников, которые не выполняют рабочий план.

Результатом дипломной работы является информационная система с удобным, логичным и доброжелательным интерфейсом для пользователя, которая воспринимается без дополнительных усилий.

ANNOTATION

The purpose of this thesis is to develop a monitoring system for employees of a pharmaceutical company. A feature of this system is the creation of the administrative part, taking into account the superpowers for the manager, such as adding a work schedule, cooperation with new pharmacy chains, tracking statistical data regarding the implementation of the work plan, company profits, and so on. Also, the development of a separate office for an employee who can view his schedule and find out wages without resorting to third-party programs or sources.

Additionally, an analysis will be carried out of various tasks that are solved in a certain subject area, for example, determining the average monthly income of an enterprise for a certain month, calculating the highest income for pharmacies, identifying employees who are of importance, as well as employees who do not fulfill the work plan.

The result of the thesis is an information system with a convenient, logical and friendly interface for the user, which is perceived without additional effort.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

MVC – Model-View-Controller

UI – User Interface

ПрО – предметна область

Валідація - перевірка значень, зазначених користувачем, і відображення знайдених помилок.

Роутінг - організація маршрутів, переходи по посиланнях і все що з ними пов'язано, маршрутизація всередині проекту.

ER-діаграма (Entity-Relationship) - це різновид блок-схеми, де показано, як різні «сутності» пов'язані між собою всередині предметної області.

Фреймворк - програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проекту.

Аудит аптеки - це комплекс заходів, спрямованих на організацію різного виду перевірок та інспекцій, метою яких є систематичне виявлення і усунення порушень, що допускаються в роботі.[9]

Аудитор індивідуальний - аудитор, який має право здійснювати аудиторську діяльність, а також надавати інші послуги, якщо інше не передбачено законодавством.

Електронний чек-лист - це зручний мобільний додаток, в якому ведеться робота щодо організованого в певному порядку документа, який містить перелік питань (пунктів, критеріїв), об'єднаних в категорії (групи) за єдиною смисловою ознакою.

Аптечний мерчандайзинг - комплекс заходів щодо найбільш ефективного розміщення товарів і інформаційно-рекламних матеріалів в торговому просторі аптечного закладу з метою надання інформаційно-консультаційних фармацевтичних послуг і збільшення обсягу продажів.[10]

Аптечна організація - організація, структурний підрозділ медичної організації, що здійснюють роздрібну торгівлю лікарськими препаратами, зберігання, виготовлення та відпуск лікарських препаратів для медичного застосування відповідно до вимог закону.

Ритейл - роздрібний продаж, роздріб, роздрібна торгівля, магазин - продаж товарів і послуг кінцевому споживачеві. У більш широкому сенсі, ритейл слід розглядати не тільки будь-яку торговельну точку, а принцип організації торгівлі.

Ритейл точка - точка роздрібного продажу, магазин, аптека.[12]

Mock - об'єкти, які замінюють реальний об'єкт в умовах тесту і дозволяють перевіряти виклики своїх методів. Містять заздалегідь підготовлені описи викликів, які вони очікують отримати. Застосовуються в основному для тестування поведінки користувача.

ВСТУП

Фармацевтичні компанії після випуску своєї продукції прагнуть поширити її через максимальну кількість аптечних мереж. Для цього вони наймають представників, завдання яких укладати договори з якомога більшою кількістю аптек на реалізацію їхнього товару, а також контролювати те, як реалізується цей самий товар.[13] Однак практика показала, що через відсутність контролю над представниками багато з них просто не виконують аудит аптек належним чином, через що статистика продажів часто складається некоректно.

Зрозуміло, що робота співробітників повинна бути злагодженою, структурованої, щоб не виникало конфліктів, і робітники виконували свої задачі вчасно та коректно. Основоположним є правильна організація роботи персоналу. Саме тому інформаційні системи такого типу будуть довгий час залишатися затребуваними на ринку інформаційних послуг.

Сайт та мобільний додаток для співробітників і адміністратору фармацевтичної компанії розроблено для систематизування та контролю операцій, наданих користувачам. Подібні сайти або програми значно покращують рівень сервісу за рахунок підвищення швидкості і автоматизації обслуговування, так як не потрібно вручну заповнювати всі дані, а також шукати вільний час в графіку, витрачаючи на це багато часу. Часто в подібних ситуаціях, робітник компанії може заплутатися в своєму графіку та не встигнути виконати робочий план, тому система, крім перерахованих можливостей, надає можливість делегування свого робочого часу іншим працівникам.

В цілому, аудит аптеки в рамках контролю стандартів аптечної мережі - це комплекс заходів, спрямованих на організацію різного виду перевірок, метою яких є систематичне виявлення і усунення порушень. Аудит аптеки потрібен не тільки для поліпшення якості роботи взагалі, але і для успішного проходження різних перевірок. За допомогою такого інструменту, як електронний чек-лист

аудитор, співробітник відділу внутрішнього контролю або керуючий мережі може організувати всеохоплюючий контроль якості роботи як окремо взятих аптек, так і аптечної мережі в цілому.

За електронним чек-листом можна максимально ефективно і швидко проводити будь-які внутрішні перевірки, виявляти допущені порушення і відхилення від прийнятих стандартів, починаючи від перевірки документації, викладки лікарських засобів на вітрині, закінчуючи оцінкою роботи персоналу. Крім фармацевтичних аудитів та різних видів внутрішньо-аптечного контролю, систематичні перевірки по чек-листам служать надійним захистом від цілої плеяди контролюючих органів. Залежність тут пряма - чим частіше проводяться самоперевірки в рамках аптечного аудиту і чим сучасніше і більш технологічні застосовувані для цього інструменти, тим менше ймовірність окремо взятої аптеки або аптечної мережі долучитися до штрафів.

Наявність системи, яка дозволить полегшити роботу співробітників, збільшить потенційний прибуток, кількість аптек для розширення мережі та перспектив.

Метою даного проекту є розробка системи моніторингу співробітників фармацевтичної компанії, де буде реалізована адміністративна частина, щоб адміністратор міг з легкістю складати аудити, економити час на облік та складання статистичних даних, також окремий кабінет для співробітників, які зможуть переглядати свої аудити, делегувати їх, та мати актуальну інформацію щодо заробітної плати.

Така інформаційна система:

1. для адміністратора відповідної фармацевтичної компанії, який потребує налагоджену систему для організації робочого часу для працівників;

2. для співробітників, які зможуть швидко і в будь-який момент дізнаватися свій робочий графік і зарплату за поточний місяць, зокрема історію щомісячних надходжень.

Для досягнення зазначеної мети в роботі необхідно вирішити наступні завдання:

1. провести аналіз предметної області;
2. сформувати вимоги до системи;
3. проектування функціональної системи;
4. розробка архітектури системи;
5. проектування БД;
6. вибір технологій і засобів реалізації системи;
7. реалізація системи, яка дасть можливість зручно використовувати дані ПрО для різних видів користувачів;
8. забезпечення розмежування доступу до системи з боку різних видів користувачів і захисту від несанкціонованого доступу.

ПОСТАНОВКА ЗАДАЧІ

Для того, щоб налагодити контроль за працівниками та прибутками компанії необхідно організувати чимало робочих процесів - скласти графік для робітників, визначити аптеки для співробітництва, зазначити правильний обсяг на поставки товарів до аптек. У свою чергу, працівнику потрібно бачити свій робочий графік та мати можливість керувати своїм часом, тобто делегувати свій аудит в разі нестачі ресурсів на його виконання. Коли є представлення всіх можливих дій, можна скласти призначену для користувача історію, на основі якої спроектувати сильну систему, що полегшить процеси на всіх етапах.

Виходячи з процесів, що мають місце в даній ПрО, було виділено категорії користувачів, що мають доступ до інформаційної системи:

1) адміністратор - має можливість переглядати всі записи аудитів, аптек та товарів в них і видаляти їх, додавати нові аптеки, аудити і робочий час, а також вести роботу з співробітниками - вести систему обліку бонусів, штрафів , наймати нових співробітників, звільняти співробітників;

2) співробітники - можуть переглядати свій робочий графік (аудит), а також володіти інформацією про поточну заробітну плату, з огляду на штрафи і бонуси, також мають право на додавання записів до аптек, перегляд свого поточного місцезнаходження.

Таким чином, кожен користувач даної інформаційної системи має свою роль, а ґрунтуючись на цій ролі, функціональність, яку йому надає продукт, в даному випадку – сайт та мобільний додаток. Інформаційна система також повинна забезпечити підтримку рішення наступних аналітичних задач:

1) відстежити середньомісячний прибуток компанії;

2) перегляд інформації про співробітників, що виконують свою роботу в повній мірі та найгірших працівників;

3) проаналізувати прибуток кожної аптеки за певними часовими періодами.

1 АНАЛОГИ НА РИНКУ ІНФОРМАЦІЙНИХ СИСТЕМ

Проведено первісний огляд продуктів, призначених для моніторингу ефективності співробітників і обліку робочого часу для аналізу ринку на схожі системи та переваги.

Наприклад, компанія розробник програмного забезпечення «Біткоп» для контролю ефективності персоналу пропонує систему обліку робочого часу Bitcop Security. Система веде кількісний і якісний облік відпрацьованого часу співробітниками в розрізі ряду аналітик, формуючи розгорнуту картину реального стану справ в організації. Система дозволяє:

- вести облік часу початку і закінчення робочого дня, запізнень і переробок;
- контролювати інтенсивність зайнятості співробітника в робочий час, а також його активність протягом дня;
- вести статистику роботи з програмами, які запускав певний колега, а також бачити, як часто і на які сайти він заходив;
- вести оперативне зведення зайнятості всього персоналу і список існуючих порушень;
- оцінити продуктивність роботи на основі кількості програм і сайтів в робочий час.

Система видає всю перераховану вище інформацію в певній, зручною і зрозумілій формі. Аналіз даних в системі контролю персоналу Bitcop Security відбувається на підставі наочних детальних звітів. Різноманітні звітні форми відображають реальний графік роботи співробітників, допомагають встановити обсяг і причини втрат робочого часу, свідчать про продуктивність службовців стосовно займаним посадам. Bitcop Security у своєму розпорядженні великий потенціал щодо застосування фільтрів. Всі аналітичні звіти можна будувати за

обраним часового інтервалу (день, тиждень, місяць, рік) - що дозволяє швидко отримати вибірку тільки цікавлять результатів.[11]

Переваги та особливості системи:

— синхронізація системи Bitcop Security з Active Directory;

— сповіщення про події в режимі реального часу;

— високий ступінь захисту агенту.

Ще один приклад компанії, що пропонує продукт з подібним функціоналом - «Атом Безпека», просуває на ринку систему моніторингу StaffCop Enterprise, що дозволяє аналізувати дії персоналу за комп'ютерами для забезпечення інформаційної безпеки підприємства, виявлення витоків конфіденційних даних, розслідування інцидентів, обліку робочого часу і оцінки ефективності роботи персоналу.

Продукт покликаний вирішувати кілька завдань. Перша - допомогти оптимізувати бізнес-процеси: отримати картину робочого дня співробітників, виявити «вузькі» місця в системі, виділити неефективних співробітників. Друга - знизити ризики виникнення інцидентів, пов'язаних з витоком конфіденційної інформації, зловмисних або випадкових дій співробітників здатних привести до фінансових і репутаційних втрат компанії. Третя - розслідування інцидентів.

StaffCop Enterprise досить потужне endpoint-рішення для моніторингу та аналізу дій персоналу на робочих станціях Windows, GNU / Linux і термінальних серверах. Має клієнт-серверну архітектуру з підтримкою PostgreSQL і володіє сучасним стеком технологій: «тонкий» клієнт, OLAP-куби, власні алгоритми аналізу дій співробітників. Веб-інтерфейс дозволяє отримати швидкий доступ до аналітики та управління системою без установки спеціальних програм. Рішення має гнучку налаштуванням фільтрів і сповіщень, тому можливий витік або вторгнення вдається виявити на ранній стадії, ніж істотно скоротити наслідки. Глобальна система запису всіх можливих подій дозволяє в разі інциденту швидко дістатися до джерела витоку і точно назвати час, автора і обсяг втраченої

інформації. Застосовувана технологія «дрілл-даун», що дозволяє в кілька кліків встановити кореляцію між даними, зібраними агентами на кінцевих точках.

Переваги:

— оперативний контроль співробітників і визначення груп ризику;
— аналітичні зрізи по будь-яким видам подій і даними (багатовимірні звіти);
— оцінка і контроль ефективності роботи персоналу з категоризацією роботи співробітників по продуктивності і видам діяльності;
— підрахунок часу активності співробітника в додатках і на сайтах;
— блокування додатків, доступу до сайтів для окремих користувачів;
— облік робочого часу співробітників.

2 ОБҐРУНТУВАННЯ ПРОЕКТНОГО РІШЕННЯ

Було зазначено переваги розробки такої системи. Вагомими з яких виявляється зручний інтерфейс, який дозволяє швидко маніпулювати даними, автоматичний моніторинг виконання робочого плану, відстежування співробітників компанії в реальному часі, а також розширені можливості для співробітників, в рахунок яких входить делегування свого робочого часу, що додає оптимальності для цієї системи.[14]

Розмежування прав доступу до інформації та реалізація ролей, які дозволяють використовувати спрощений перехід від звичайного користувача, тобто працівника до адміністратору – також послугували значущим фактором для розробки такого проекту. Окрім цього, адміністратору надано можливості збирати статистики щодо виконання робочого плану та якості роботи співробітників.[15]

3 ПРОЕКТУВАННЯ СИСТЕМИ

3.1 Проектування інформаційної системи

Для реалізації інформаційної системи обрана трирівнева архітектура клієнт-сервер. У такій технології існує тенденція, заснована на великому використанні розподілених обчислень. Вона реалізується на основі моделі сервера додатків, де мережевий додаток розділене на дві і більше частин, в даному випадку - три частини, кожна з яких може виконуватися на окремому комп'ютері. Виділені частини додатка взаємодіють один з одним, обмінюючись запит - відповідями в заздалегідь відомому форматі.

У цій архітектурі між клієнтом і сервером баз даних є проміжний шар - сервер додатків, що є для користувача сервером, а для системи управління базами даних - клієнтом. На сервері додатків зберігається програмне забезпечення, що містить основну логіку інформаційної системи. Сервер додатків аналізує вимоги користувача і формує запити до сервера БД. Сервер БД крім зберігання даних, здійснює підтримку їх цілісності, виконує запити до даних, забезпечує безпеку і розмежування доступу до даних.

Дана система пропонує наступний алгоритм, який підлаштовується під цю архітектуру. Наприклад, в моєму додатку є призначений для користувача інтерфейс, тобто «клієнт». Коли користувач системи проробляє якісь маніпуляції на призначеному для користувача інтерфейсі, дані про ці маніпуляції або ж зміни надходять на сервер додатки у вигляді запиту. Після отримання даних, або ж запиту, сервер обробляє їх, також якщо необхідно перевіряє на коректність, і тільки тоді відправляє на сервер баз даних. В наслідок чого, сервер БД відправляє назад відповідь (успішно виконаний запит чи ні).

Для реалізації даної структури, був обраний шаблон проектування (патерн) MVC, який підходить до даної архітектурі. Схема поділу даних програми,

призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: модель, уявлення і контролер - таким чином, що модифікація кожного компонента може здійснюватися незалежно.

Уявити реалізацію взаємодії даного патерну можна у відповідному вигляді (рис. 3.1).

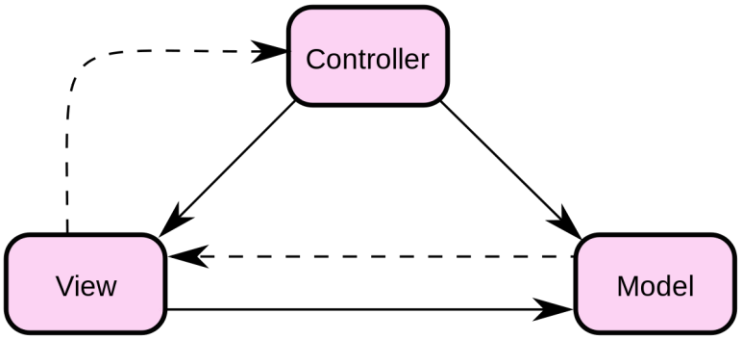

Рисунок 3.1 – Діаграма роботи шаблону проектування MVC

Контролер - це спеціальний клас, який отримує дані з запитів користувача і управляє ними. Зазвичай це HTTP-запити видів GET або POST. Коли виникає необхідність, контролер викликає відповідну модель для виконання поставленого завдання і відправки даних в БД.

Модель - це клас, що зв'язує з відповідною таблицею в БД. Модель передає контролеру дані, запитувані користувачем. Далі, отримані і оброблені дані відправляються у view (вид). Вид являє собою користувальницький інтерфейс, на якому відображаються всі запитувані дані, які отримані з моделі.

Веб-додаток складається з великої кількості контролерів, моделей і видів. Контролер в основному виконує одну логічну сформульовану задачу. Такий контролер має всього кілька функцій і повертає в цілому тільки один view файл, тільки одне подання.

3.2 Інформаційне моделювання

Перший етап проектування полягає в створенні схеми БД, яка включає визначення сутностей і існуючих між ними зв'язків і атрибутів. Результатом даного проектування є схема БД, представлена у вигляді ER-діаграми, на якій відображаються основні сутності ПрО, також відповідні атрибути і зв'язки. Схема БД створюється на основі користувальницької історії та вимог, які спочатку ставляться перед системою. Створення схеми не залежить від вибору реалізації інформаційної системи.

Послідовність етапів створення схеми бази даних:

1. визначення сутностей;
2. визначення зв'язків між сутностями;
3. визначення атрибутів сутностей;
4. завдання первинних і зовнішніх ключів.

У цій роботі поставлена задача побудови ER-діаграми для фармацевтичної компанії.

Далі для кожної сутності потрібно задати первинний ключ - унікальний ідентифікатор, який однозначно характеризує кожен екземпляр, а також унікальні ключі. На прикладі суті Товар: id - первинний ключ.

Наступним кроком формалізація зв'язків між сутностями. Зазначимо, якщо тип зв’язку між двома сутностями має вид «багато-до-багатьох», тоді використовується формалізація зв'язку - нова сутність.

В першу чергу, бачимо, що між сутностями «Товар» і «Аптека» можна створити окремі сутності - «Поставка» і «Продаж» - з такими атрибутами: id - аптеки, id - товару. На нову сутність «Поставка» і «Продаж» будуть посилатися за допомогою зовнішнього ключа, а зв'язок приймає форму «багато-до-багатьох», тому що у суті «Товар» може бути багато аптек, і «Аптека» має багато товарів.

Раніше згадувалися й інші сутності, які мають тип зв’язку «один-до багатьох». Такі відносини також були формалізовані на даній БД.

Після формалізації було отримано наступну діаграму, яка представлена на рис. 3.2.

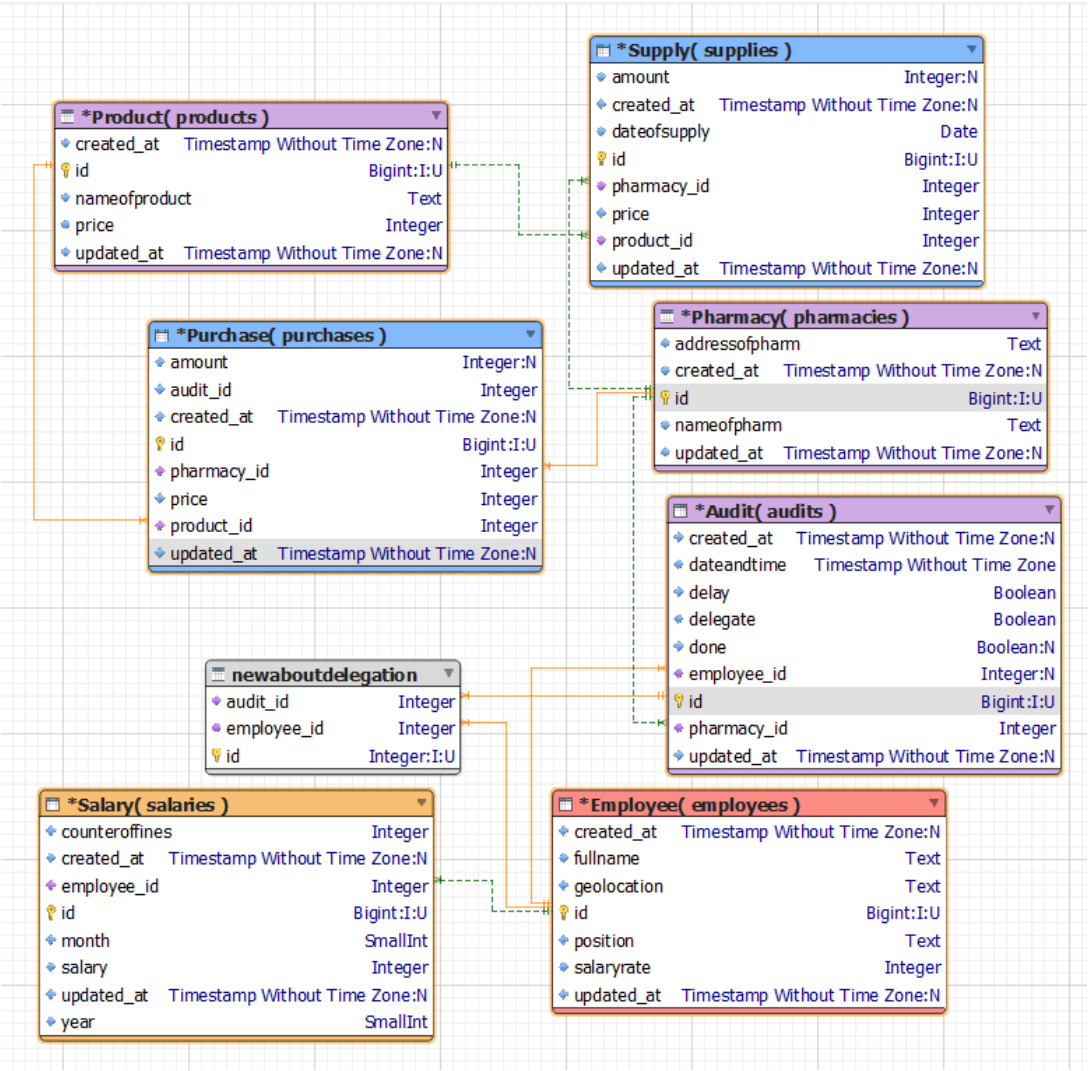

Рисунок 3.2 - ER-діаграма БД фармацевтичної компанії

Існують інші допоміжні таблиці такі як migrations, oauth_access_tokens, oauth_auth_codes, oauth_clients, oauth_personal_access_clients, oauth_refresh_tokens, password_resets, users.[2] Вони сформовані в цілях підтримки реєстрації та входу користувача до системи, а також підтримки API, необхідність якого виникла для подальшої роботи з мобільним додатком.

3.3 Вибір програмного забезпечення

В якості мови програмування був обраний PHP, в якості фреймворку обраний Laravel, який має ряд переваг:

- розвивається фреймворк має на увазі хорошу документацію, в порівнянні з іншими фреймворками або CMS, в більшості яких потрібно шукати додаткову інформацію;
- є можливість створювати масштабні проекти, незалежно від складності та напрямки, в тому числі і багаторівневі веб-сайти;
- надійна вбудований захист бази даних від SQL, CSRF, XSS;
- патерн MVC, який є зручним для розробки, а також при роботі навіть з великими базами даних.[3]

Щодо клієнтської частини мобільного додатку, було вирішено використовувати Flutter. Дана технологія містить комплект засобів розробки і являється фреймворком з відкритим вихідним кодом для створення мобільних додатків під Android та iOS, а також веб-додатків з використанням мови програмування Dart, розроблений і створюваний корпорацією Google. Він має такі особливості:

- висока швидкість: додатки на Flutter компілюються в машинний код, який використовує графіку і механізм візуалізації, вбудований в C/C++, тому додатки виходять швидкими і високопродуктивними;
- більш помітне зростання продуктивності в порівнянні з іншими багатоплатформовими технологіями;
- наявність докладної документації та великого числа прикладів;
- наявність великої кількості шаблонів проектування для інтерфейсу та високопродуктивний механізм візуалізації.

В якості СУБД був обраний PostgreSQL. Дана СУБД володіє такими перевагами:

— PostgreSQL - об'єктно-реляційна СУБД, що дає їй переваги перед іншими SQL базами даних з відкритим вихідним кодом;
— розширення - існує можливість розширення функціоналу за рахунок використання збережених процедур, тригерів, уявлень, складових типів даних;
— розроблена хороша і зрозуміла документація, так само існує безліч джерел, в тому числі англомовних, які структуровано пояснюють ці запитання;
— існує безліч API під різні платформи розробки.

В ході розробки виникла потреба у використанні таких інструментів як Figma, Postman, Google APIs & Services, Valentina Studio.

Перший це онлайн-сервіс для розробки інтерфейсів і реалізація прототипів з можливістю організації спільної роботи в режимі реального часу. Використовується як для створення спрощених прототипів інтерфейсів, так і для детального опрацювання дизайну інтерфейсів мобільних додатків, веб-сайтів, корпоративних порталів.

Postman – набір інструментів тестування API. Основне призначення програми - створення колекцій із запитами до розробленого API. Також Postman дозволяє проектувати дизайн API і створювати на його основі Mock-сервер.

Google APIs & Services - набір класів, процедур, функцій, структур і констант, що надаються сервісами Google для використання в зовнішніх середовищах. Зокрема, API Google Map - картографічний сервіс, який дозволяє додавати карти у додатки, а також маніпулювати даними, які надаються сервісом.

Останній - Valentina Studio, це інструмент для роботи з базами даних - один зі способів для перетворення даних в значущу інформацію, програма підтримує власну Valentina DB, а також MySQL, SQLite і Postgre.

4 ПРОГРАМНА РЕАЛІЗАЦІЇ СИСТЕМИ

4.1 Створення та наповнення бази даних

Детально розглянемо запит на створення однієї з таблиць БД - Products.

```
public function up()
{
Schema::create('products', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->integer('price');
        $table->text('nameofproduct');
        $table->timestamps();
    });
}
```

Створення таблиці відбувається за допомогою функції up(), яка користується об’єктом типу Schema для відтворення таблиці методом create. Який, у свою чергу, приймає два параметри: перший – назва для створення таблиці, а другий – функція, що створює атрибути таблиці за допомогою об’єкта Blueprint. Поле id є первинним ключем таблиці Products. Про те, що це первинний ключ каже запис bigIncrements. Запис буде являтися NOT NULL - означає, що при заповненні таблиці командою INSERT значення не може бути нульовим, або ж порожнім, відповідно, запис NULL означає, що поле може бути не заповнено і залишатися порожнім. Integer - рядок, який створює атрибут з числовим типом даних. Таке позначення як time without time zone є timestamp, являє собою час. Text – поле, що уособлює текстовий тип даних.

Всі інші таблиці створені з використанням відповідних запитів, детально наведено в Додатку А.

Для швидкого наповнення БД тестовими даними існують фабрики моделей (factory) і наповнювачі (seeders), що дозволяють створювати записи тестової бази даних з використанням моделей і відносин Eloquent застосування, яке було

прописано на стороні серверу. Для створення однієї з таких фабрик було запущено таку команду:

```
php artisan make:factory PharmacyFactory
```

У разі прикладу послугує наповнення таблиці pharmacies:

```
$factory->define(Pharmacy::class, function (Faker $faker) {
    return [
        'addressofpharm' => $faker->secondaryAddress,
        'nameofpharm' => $faker->company
    ];
});
```

З наведеного коду, видно, що ініціюється змінна, яка переймає у свої параметри назву моделі згідної сутності та функцію, що повертає сформовані тестові дані в відповідні атрибути сутності за допомогою класу Faker. Для того щоб згенерувати деяку кількість записів у БД було запущено таку команду:

```
Pharmacy::factory()->count(3)->make();
```

Згідно до цього запису, зрозуміло, що до таблиці БД додано 3 записи з фейковими даними, які містять в собі назву аптеки та її адресу. Надалі, щоб уникнути непорозумінь, дані будуть змінюватися та використовуватимуться аптеки, що дійсно існують.[1] Перевірятися це буде за допомогою Google APIs & Services.

4.2 Розподіл ролей в базі даних

У даній інформаційній системі фармацевтичної компанії існують дві ролі.[5] Розглянемо їх докладніше з точки зору прав на таблиці бази даних:

1. співробітник - має права на читання сутностей Аптека, Аудит, Продаж, Зарплатня, Співробітник, Новини делегованого, Товар. Також має права на редагування деяких з них: Аудит, Продаж, Новини делегованого;
2. адміністратор - всі CRUD операції над такими сутностями, як Аптека, Співробітник, Аудит, Зарплатня, Товар, Поставка. Право на читання

таблиць, які не вимагають змін, наприклад, сутності Новини делегованого, Продаж, Знижка, Зарплатня.

4.3 Програмна реалізація інформаційної системи

Для реалізації інформаційної системи був створений ряд класів, які вирішують різні завдання в залежності від рівня, в якому вони знаходяться. Так як основний компонент ООП - клас, реалізація програми представлена за допомогою класів. Основні з них на кожному з рівнів:

1) Model - це базовий клас, який представляє собою відповідні таблиці БД. Під кожну сутність в БД існує своя модель. Наприклад, модель Pharmacy - успадковується від основного класу Model і має в собі три функції, які описують відносини суті в БД. В даному прикладі, публічні функції audit(), purchase() і supply() описують ставлення один-до-багатьох до сутностей Audit, Purchase, Supply відповідно. Це показано методом hasMany, на прикладі з першим ставленням - $ this -> hasMany (Pharmacy:: class);

2) Controller – базовий клас, який взаємодіє з моделлю і представленням. На кожну основну задачу створюється свій контролер. На прикладі, розглянемо контролер, що оброблює модель класу Pharmacy - PharmacyController. У ньому є функція __invoke (), яка дозволяє при ініціалізації контролера вже виконувати якісь логічні дії, і в підсумку повертатися файл уявлення оператором return. Повний вид такої - return view('pharmacies',['pharmacies'=>\App\Pharmacy::all()]), де в квадратних дужках можуть передаватися параметри, отримані в результаті деяких дій логіки контролера, а так само дані, отримані при взаємодії з Моделлю.

3) Migration – клас, який дозволяє визначити і спроектувати поля таблиць БД, для того щоб при роботі з моделлю та даними з БД було зрозуміло як називаються поля, який у них тип і обмеження. В принципі, можна обійтися

без опису всіх полів таблиць в міграціях, але необхідно враховувати деякі незручності, які можуть виникнути в зв'язку з цим. Нижче приведена діаграма класів, на прикладі PharmacyController, сутності Pharmacy і виду pharmacy (рис.4.1).

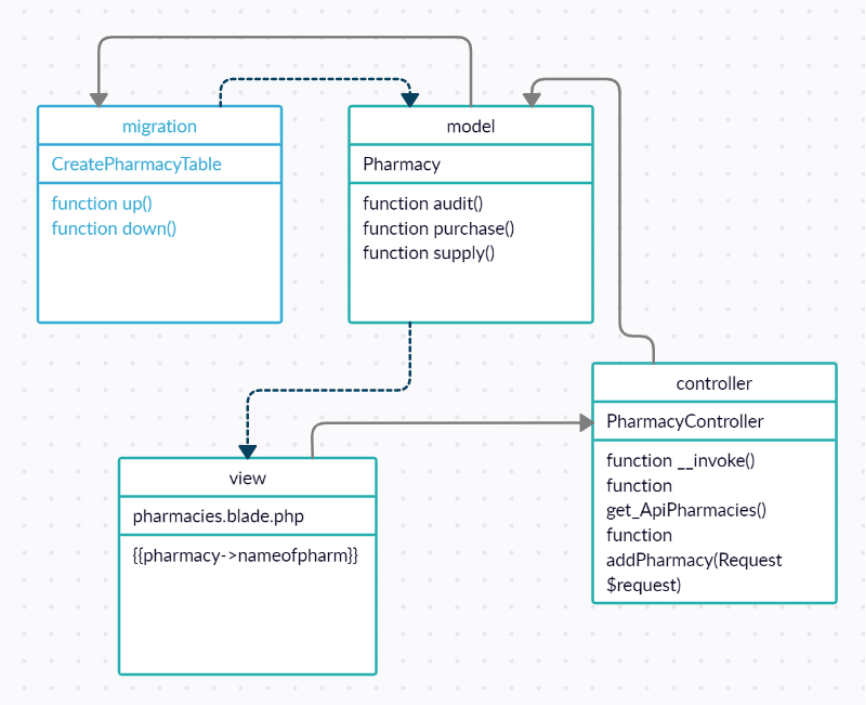

Рисунок 4.1 – Діаграма класів з розподілом за рівнями шаблону

Так як виникла необхідність використовувати чисті запити в БД, не використовуючи ORM, довелося реалізувати свій підхід. Використовуємо об'єкт, який дозволяє на чистому мовою Php створювати підключення до БД і за допомогою нього демонструвати чисті SQL запити.[4] Має такий вигляд –

```
$dbh = new PDO($dsn,$user,$password);
```

Розглянемо реалізацію розмежування ролей з боку архітектури програми. Перед тим як приступити до розгляду, варто сказати, що спочатку доступ до системи належить ролі Гість. У даної ролі є обмежені права, що було описано вище.

Існує базовий клас Auth, який представляє собою аутентифікацію користувача. За допомогою нього, можна отримати методом check () інформацію про те, вийшов користувач в систему, чи ні. програмна реалізація –

```
if(\Auth::check()){
    $username = session('username');
    $ds;
    $u;
    $pass;
    $rol = \Auth::user()->role_;
```

З уявлення коду зрозуміло, що якщо користувач увійшов в систему, то з сесії дістається його для користувача ім'я, далі не започатковано змінні для нового підключення до БД. Потім, дістається значення із сутності БД Користувач, поле role_, яке повідомляє яку назву ролі у поточного користувача.[6]

Далі має бути умовна конструкція, яка вибирає ім'я і пароль для нового підключення до баз даних, під новою роллю. Після такої маніпуляції, в сесію під ключем 'role_panel' додається назва ролі.

```
session(['role_panel' => $user]);
```

Після чого відбувається перепідключення до БД з новою роллю і правами відповідно.

```
$dbh = new PDO($dsn,$user,$password);
```

За допомогою утиліти phpinfo () можна подивитися поточного користувача БД під відповідними глобальними змінними –

```
$_SERVER['DB_USERNAME']
$_SERVER['DB_PASSWORD']
```

4.4 Запити до бази даних для вирішення результатів

У попередньому розділі було згадано про виникнення необхідності реалізації чистих SQL запитів, і було запропоновано нове рішення. Ґрунтуючись на реалізованому рішенні, була побудована структура, яка є файлом, де запити описані в поданні публічних функцій. Для прикладу, візьмемо функцію, яка

здійснює запит до БД для отримання статистичного запиту щодо прибутків визначеної аптеки за визначений місяць.[7]

```
function statfarmformonth($dbh, $pharm_id, $month){
    $p = (int)$pharm_id;
    $m = (int)$month;
    $result=$dbh->query("select*from statfarmformonth($p,$m");
    $array = array();
    while($data = $result->fetch(PDO::FETCH_ASSOC)){
      array_push($array, $data);
    }
    return $array;
   }
```

З прикладу видно, що в параметр функції потрапляє підключення до БД. Радиться ще раз звернути увагу, що підключення відбувається вже під існуючими ролями. Далі, в деяку змінну $result потрапляє запит, який здійснюється за допомогою підключення до БД і методу query(), в параметр якого приходить чистий SQL запит.

Після таких викликів, створюється масив, в який будуть додаватися дані, отримані з описаної змінної $result.

Далі, такі функції будуть викликатися на рівні контролера, де вже буде відбуватися маніпуляція даними. Розглянемо один з таких прикладів - AuditController, призначений для перегляду та додавання даних в панелі для аудитора, а також більш розширених можливостей в адміністративній панелі, власне яка доступна тільки для користувачів з роллю - адміністратор.

Функціонал контролера досить великий, в ньому описана вагома частина логіка процесів, одна з них це маніпуляція з додаванням нового аудиту в інформаційну систему. Функція, яка реалізує даний функціонал - додавання користувача, має наступну реалізацію:

```
 public function createAuditAdmin(Request $request)
{
    $employeeName = $request->only('nameEmployee');
    $emp_id = \App\Employee::all()
    ->where('fullname', '=', $employeeName['nameEmployee'])
```

```
        ->pluck('id');
        $pharmacyName = $request->only('nameofpharm');
        $pharm_id = \App\Pharmacy::all()
        ->where('nameofpharm', '=', $pharmacyName['nameofpharm'])
        ->pluck('id');
        $date = $request->only('date');
        $date = $date['date'];
        $time = $request->only('time');
        $time = $time['time'];
        $dateandtime = $date. ' ' . $time;
        $audit = new Audit();
        $audit->pharmacy_id = $pharm_id[0];
        $audit->dateandtime = $dateandtime;
        $audit->employee_id = $emp_id[0];
        $audit->delegate = false;
        $audit->delay = false;
        $audit->done = false;
        $audit->save();
        return response()->json([
         'doneAudit' => 'changed to done'
        ], 200);
    }
```

Як видно, функція отримує в параметрах об’єкт типу Request, що дозволяє отримувати дані відправлені API-запитом. Розберемо детальніше обробку даних, наприклад, змінна $employeeName отримує ім’я працівника таким чином: $request->only('nameEmployee'), що означає із екземпляру об’єкта Request, тобто з запиту виокремлюється 'nameEmployee' та записується у відповідну зміну. Надалі отримані дані оброблюються наступним чином: викликається модель об’єкту працівника - \App\Employee, та виконується запит сформований ORM-конструкцією: where ('fullname', '=', $employeeName['nameEmployee']) -> pluck('id'). Це означає, що виокремлюємо id працівника з ім’ям, яке було отримано раніше.

Аналогічно відбувається обробка інших даних, які нам необхідні для створення нового аудиту. Після отримання та обробки таких даних, з’являється можливість створення об’єкту нового аудиту: new Audit(), після чого заповнення відповідних полів отриманими даними. У разі виконаної роботи, можна

здійснити збереження у БД створеного аудиту та відправити успішну відповідь (response) на запит:

```
$audit->save();
return response()->json([
  'doneAudit' => 'changed to done'
], 200);
```

Як видно відповідь відправляється у вигляді json-об’єкту з двома параметрами – тіло відповіді та код відповіді. У даному випадку код відповідає 200, тому що все виконано успішно. У ході таких маніпуляцій аудит додано до БД.

Тепер можна розглянути функцію, яке поверне вид сторінки, яка буде використовуватися для демонстрації можливостей адміністратору, в тому числі додавання нового аудиту:

```
public function __invoke()
{
    return view('audits',['audits'=>\App\Audit::all()]);
}
```

Можна розглянути декілька прикладів збережених процедур. Для вирішення задачі, такої як перегляд статистики по представнику (штрафи) за місяць, потрібен був такий запит:

```
CREATE OR REPLACE FUNCTION statempfinesmonth(idofthisemp int,
thismonth int)
RETURNS INTEGER AS $$
BEGIN
RETURN (SELECT counteroffines FROM salaries WHERE
employee_id=$1 AND salaries.month=$2);
END;
$$ LANGUAGE plpgsql;
```

Для перегляду статистики продажу аптеки за місяць треба було виконати такий запит:

```
CREATE OR REPLACE FUNCTION statfarmformonth(idofthisfarm int,
thismonth INT)
RETURNS INTEGER AS $$
BEGIN
RETURN (SELECT sum(price) FROM purchases JOIN audits ON
purchases.audit_id = audits.id
```

```
WHERE                purchases.pharmacy_id=$1                AND
date_part('month',audits.dateandtime)=$2);
END;
$$ LANGUAGE plpgsql;
```

4.5 Реалізація інтерфейсу мобільного додатку

Розглянемо декілька прикладів з клієнтської частини проекту. Проектування складається з віджетів, кожен з яких відповідає за свій набір функціоналу та має свої властивості. Віджет Expanded відповідає за блок, якому можна не задавати розмір, розширюється завдяки змісту внутрішніх спадків. ListView уособлює у собі конструктор для створення списку, що дублює внутрішні віджети згідно до кількості елементів вхідного масиву. За таку властивість відповідає itemCount. Описана поведінка представлена таким кодом:

```
Expanded(
 child: ListView.builder(
     itemCount:  _userAudits.length,  //визначається кількість
елементів списку
     itemBuilder: (context, i) {
       return Column(
         children: [
         Card(
         elevation: 2.0,
         margin:EdgeInsets.symmetric(horizontal:8,vertical:4),
         child: Container(
           decoration:
           BoxDecoration(color:
           (_userAudits[i].delay && _userAudits[i].done)
               ? Colors.green[400]
               : (_userAudits[i].delay &&_userAudits[i].done==
false)
               ? Color.fromRGBO(252, 60, 64, 0.9)
               : Color.fromRGBO(230, 240, 250, 0.95)
//згідно умови тернарного оператору змінюється колір блоку
           ),
           child: ListTile(
             contentPadding:  EdgeInsets.symmetric(horizontal:
10),
             leading: Container(
```

```
                    padding: EdgeInsets.only(right: 12),
                    child: Icon(  _userAudits[i].delegate
                        ? Icons.person_outline_sharp
                        :  null,//якщо  змінна  відповідає  true  –
обирається іконка для віджету, якщо ні – іконка відсутня
                        color:
Theme.of(context).textTheme.headline6.color),
                    decoration: BoxDecoration(
                        border: Border(
                            right:
                            BorderSide(width:       1,       color:
Colors.white24))),
                ),
                title: Text(_pharmNameAudits[i].pharmacy,
                    style: TextStyle(
                        color:
Theme.of(context).textTheme.headline6.color,
                        fontWeight: FontWeight.bold)),
                trailing: GestureDetector(
                  child: (iconMenu == true)
     ? Icon(Icons.menu_open,color:Colors.black87)
                :Icon(Icons.menu_open,color:Colors.black87),
                  onTap: () {
                    setState(() {
                      print('OPEN MENU');
                      iconMenu=(iconMenu==true?false:true);
                      menuHeight=(menuHeight==0.0?100.0:0.0;
                    menuTextTime=_userAudits[i].dateandtime;
                      auditId = _userAudits[i].id;
                      menuTextName                                  =
_pharmNameAudits[i].pharmacy;
                    });
//обробка кнопкою, що відповідає за зміну іконки, існування
випадаючого меню та демонстрацію інших налаштувань
                  },
                ),
                subtitle: Text(_userAudits[i].dateandtime,
                    style: TextStyle(
                        color:
Theme.of(context).textTheme.headline6.color,
                        fontWeight: FontWeight.normal)),
              ),), //вивід дати та часу заданого елементу
            ),
```

```
        ],);}),);
```

Також продемонструємо проектування веб-сайту. Надалі наведено реалізацію кнопки, що дозволяє адміністратору переглянути статистику щодо продаж обраної аптеки за весь час.

```
<button type="button" disabled  class="btn btn-primary btn-
lg">Продажи аптеки за все время</button>
              <div class="col">
                   <form       action=""       method="get"
enctype="multipart/form-data">
                        {{ csrf_field() }}
                        <br>
                        <select              name="pharm_id"
style="width:auto; display:inline-block;">
                        @foreach($pharmacies as $pharm)
                             <option>{{$pharm['id']}}
{{$pharm['nameofpharm']}} </option>
                        @endforeach
                        </select>
                        <input type="submit" value="Показать"
style="padding:5px;" ><br>
                   </form>
                        <input   type="text"   placeholder=""
name="" value="{{$sales_pharm}}"><br>
          </div>
```

Як видно, за кликанням на кнопку, запит оброблюється формую, що відправляє дані на сервер та поверне відповідь у зазначеному полі.[8] Перед тим як надати запит на обробку даних, користувачу надається можливість обрати аптеку згідно якої є намір виявити статистичні дані. Така можливість існує завдяки оператору foreach, який перебирає pharmacies та виводить на інтерфейс назви існуючих аптек таким чином: {{$pharm['nameofpharm']}}.

4.6 Тестування реалізованої серверної частини

Для тестування реалізованого REST API було обрано програму Postman. В ході роботи була створена структура папок (рис. 4.2), яка організовує та полегшує тестування коректності роботи.

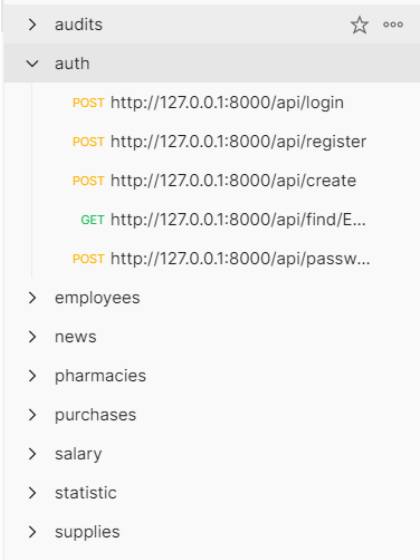

Рисунок 4.2 – Організація робочого простору для тестування REST API

У ролі прикладу на даному етапі роботи можна навести запит api/supply/all_suppliesInPharmaId, що відповідає за товари в заданій аптеці. Всі роути серверної частини реалізовані в файлі api.php. Роут, відповідний до наведеного запиту, виглядає таким чином:

```
Route::group(['prefix' => 'supply'], function () {
    Route::post('all_suppliesInPharmaId',
'SuppliesController@get_ApiSupplies');
    Route::post('set_supply',
'SuppliesController@setAdminSupply');});
```

Створюється група для запитів, яка об’єднується префіксом 'supply' і може розширюватися зі зростанням функціональності відповідної до логічної структури. Видно, що запит типу Post, тобто параметри передаються в тілі запиту (request body), а також звернення відбувається за таким ім’ям - all_suppliesInPharmaId. Крім цього, в роутах видно яким контролером оброблюється інформація і яка функція відповідає саме за конкретну задачу.

Як видно на рисунку 4.3, в параметрах body до запиту вказується ключ «pharmacy_id», що відповідає за id аптеки, щодо якої необхідно дізнатися про наявні товари.

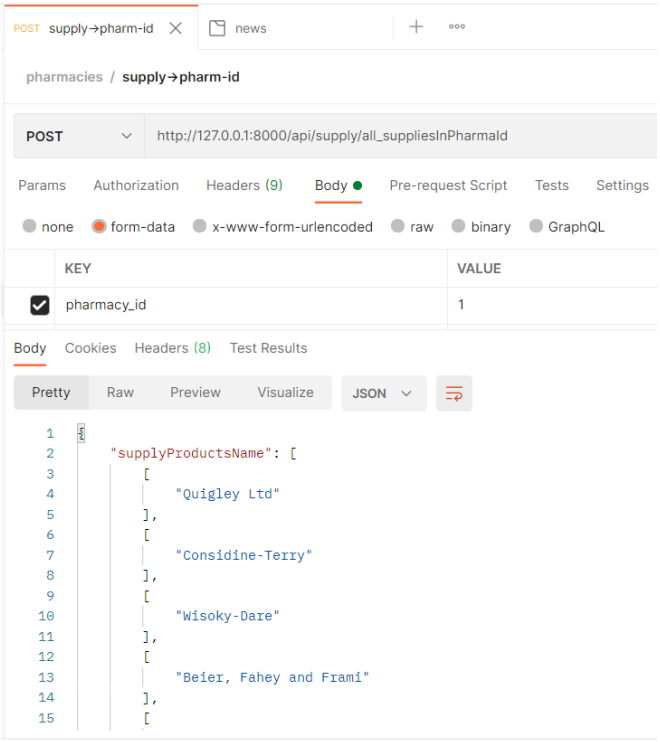

Рисунок 4.3 – Запит «Товари в заданій аптеці», request-response

В body response видно, що повернувся об’єкт з ключем supplyProductsName, який вказує на те, які товари було поставлено в аптеку за вказаним id. Згідно до цього, можна запевнити, що тестування запиту успішне, роут реалізований правильно і можна продовжити роботу далі.

4.7 Взаємодія клієнтської та серверної частини у мобільному додатку

Більшість дій між користувачем та програмою відбувається за допомогою кнопок, які дозволяють маніпулювати даними, які доступні користувачу. Однією з таких маніпуляцій є виведення інформації щодо аудитів, які виконані по робочому плану. За таку дію відповідає кнопка, яка задає розмір блоку, в якому буде доступна така інформація :

```
onPressed: () {
        setState(() {
        print('_doneAudits.length ${_doneAudits.length}');
```

```
                heightDoneAudit = (heightDoneAudit == 0.0 ?
_doneAudits.length*85.0 : 0.0);
        });
      },
```

Перед тим, як описати логіку блоку, яка вже демонструє вихідні дані, необхідно надати запит до серверу з відповідними вхідними даними. Даний етап здійснений за допомогою функції:

```
List<AuditDone> _doneAudits = List<AuditDone>();
Future <List<AuditDone>> fetchDoneAudit() async{

var response = await
http.get(Uri.parse("http://10.0.2.2:8000/api/audit/doneAudit")
);
var doneAudits = List<AuditDone>();
if(response.statusCode == 200) {
      print(response.body);
      Map<String, dynamic> map = json.decode(response.body);
      Map<String, dynamic> donesJson = map["doneAudits"];
     for(var doneJson in donesJson.values){
        doneAudits.add(AuditDone.fromJson(doneJson));
      }
     }
   return doneAudits;
}
```

Згідно реалізованого функціоналу, видно, що також існує клас AuditDone, який уособлює у собі сутність виконаних аудитів і виглядає таким чином:

```
class AuditDone {
  String dateandtime;
  String done;
  String delay;
  AuditDone({this.dateandtime,  this.done, this.delay});
  AuditDone.fromJson(Map<String, dynamic> json_){
    dateandtime = json_["dateandtime"].toString();
    done = json_['done'].toString();
    delay = json_['delay'].toString();
    print(dateandtime);
  }
}
```

З наведеного класу видно, що сутність переймає описані атрибути сутності Аудит з бази даних, що й повертає API запит.

Таку функцію, що реалізує взаємодію між клієнтом та сервером необхідно викликати при ініціалізації поточної сторінки таким чином:

```
fetchDoneAudit().then((value) {
      setState((){
        _doneAudits.addAll(value);
      });
    });
```

Згідно до логіки, дані щодо виконаних аудитів вже отримані, отож необхідно показати їх користувачу:

```
 ListView.builder(
     itemCount: _doneAudits.length,
     itemBuilder: (context, i) {
      ......
                 title: Text(_doneAudits[i].dateandtime),
                           trailing:
     GestureDetector(child:Icon(Icons.menu_open,
     color:Colors.black87)
                   onTap: () {
                     setState(() {
                       print('OPEN MENU');
                ......
                       });
                     },
                   ),
                   subtitle: Text(_doneAudits[i].dateandtime,
                      ......
          height: heightDoneAudit
     )]));
```

Змінна _doneAudits переймає всі характеристики з класу AuditDone та містить у собі масив отриманих даних, якими можна вільно користуватися. Після всіх описаних дій користувач отримує відповідь від серверу, яка після обробки даних продемонстрована на рис. 4.4.

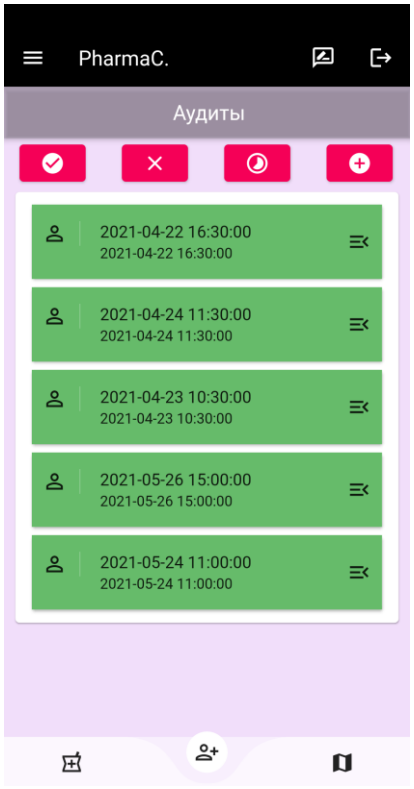

Рисунок 4.4 – Список виконаних аудитів

Також існує інший підхід для вирішення таких задач. Наприклад, використання функцій, які не потребують представлення даних користувачу для читання, тобто перегляду інформації. Частіше це збереження даних до БД для подальшого користування. Демонстрація кнопки, яка відповідає за виклик однієї з таких функцій:

```
child: RaisedButton(
                  onPressed: () {
                    print('SAVE');
                      addAuditAdmin(pharmaciesList,
      employeesList, datesList, timesList);
                  },
                  child:
                  Text(
                      "Сохранить", style:
                ......
                ),
```

Видно, що викликається функція addAuditAdmin з вхідними параметрами, які будуть у подальшому провалідовані та відправлені з відповідним запитом до серверу. Реалізація такої функції:

```
addAuditAdmin(String namePharm, nameEmployee, date, time) async
{
    Map body = {
      'nameofpharm': namePharm,
      'nameEmployee': nameEmployee,
      'date': date,
      'time': time,
    };
    var response = await http.post(
      Uri.parse("http://10.0.2.2:8000/api/audit/createAudit"),
body: body);
    var jsonResponse = null;
    if (response.statusCode == 200) {
      jsonResponse = json.decode(response.body);
      Fluttertoast.showToast(
          msg: "Аудит добавлен!",
          toastLength: Toast.LENGTH_SHORT,
          gravity: ToastGravity.CENTER,
          timeInSecForIosWeb: 1,
          backgroundColor: Colors.lightGreen,
          textColor: Colors.white,
          fontSize: 16.0);
      if (jsonResponse != null) {
        setState(() {});
      }
    }else {
      setState(() {});
      Fluttertoast.showToast(
          msg: "Ошибка добавления!",
          ......
);
      print('response error: ${response.body}');
    }}
```

На інтерфейсі користувача такі дії продемонстровані на рис. 4.5-4.6.

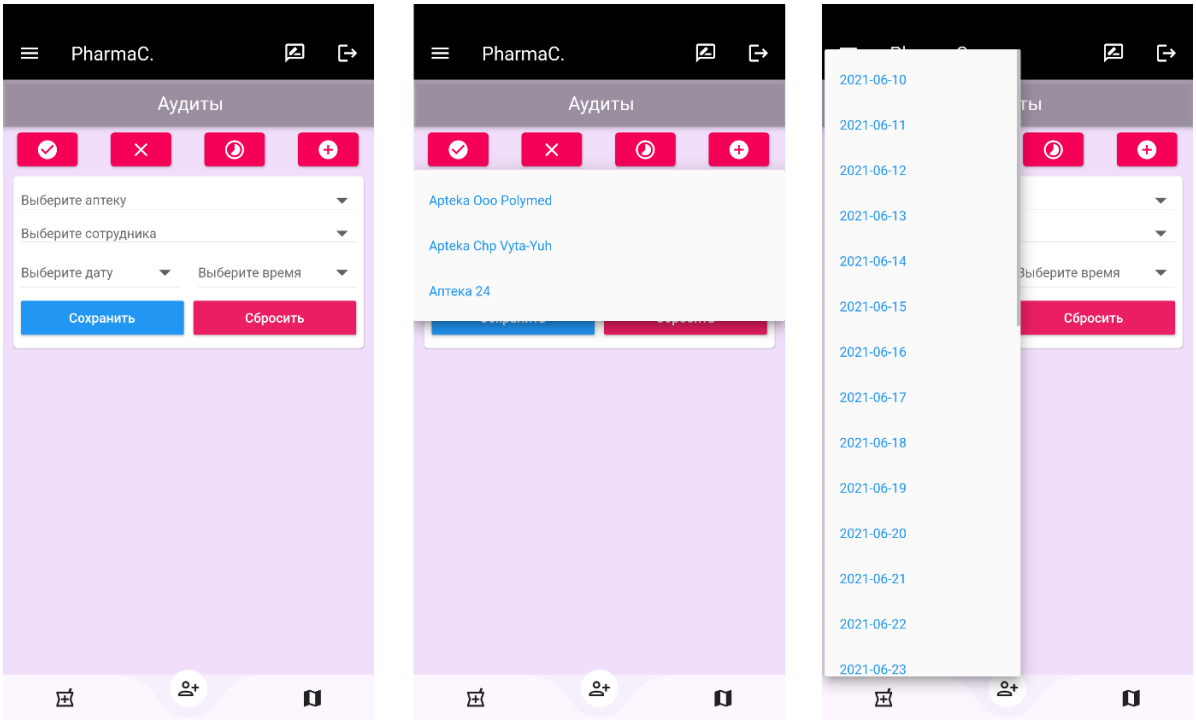

Рисунок 4.5 – Вибір даних для створення нового аудиту

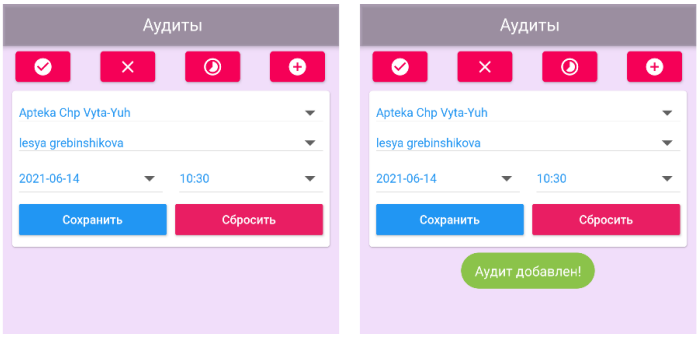

Рисунок 4.6 – Успішне створення та збереження аудиту до БД

Більша частина маніпуляцій даних відбувається подібним чином та має подібну структуру.

5 КЕРІВНИЦТВО КОРИСТУВАЧА

При запуску програми, в даному випадку мобільного додатку, користувач бачить головну сторінку на якій відображено вікно входу в систему з можливістю переключенням на реєстраційну форму (рис. 5.1). Також користувач має можливість побачити текст помилки при авторизації в разі вводу неправильних даних.

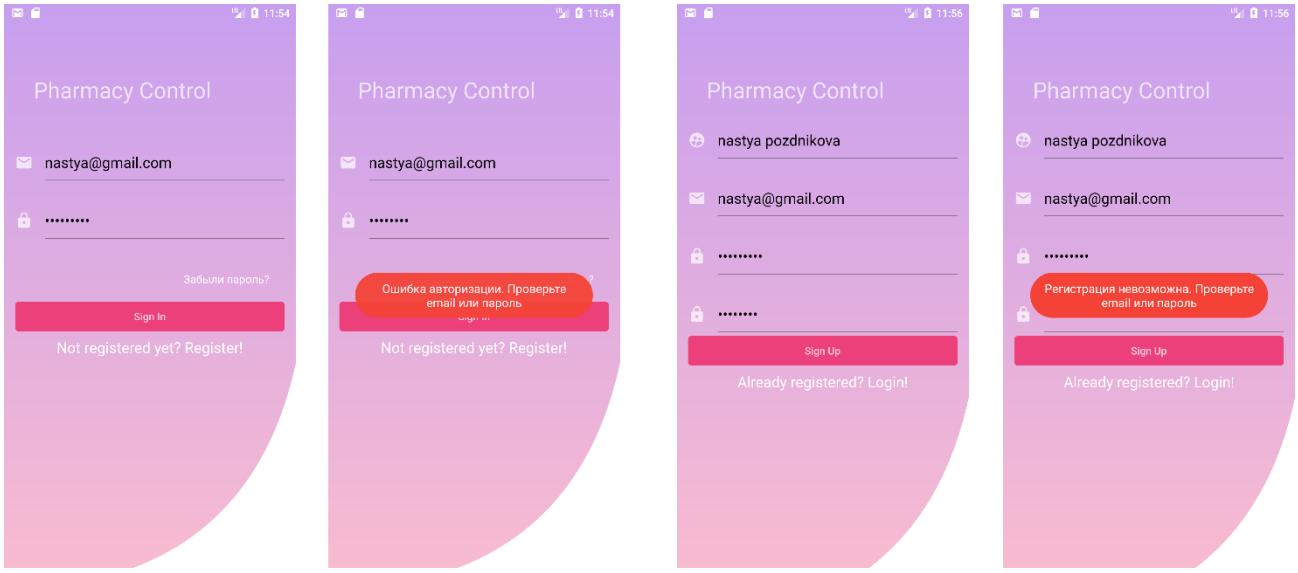

Рисунок 5.1 – Головна сторінка, авторизація користувача

Як вже було зазначено, системою запропоновано 2 типу користувачів: адміністратор та звичайний працівник. Якщо авторизований користувач був у ролі працівника, інтерфейс пропонує наступний вигляд, що представлений на рисунку 5.2. Зображено список аптек, по відкриттю яких відтворюється список товарів, що існує у відповідних аптеках. А також існує можливість заповнення аудиту по товарам (ціна та кількість проданого).

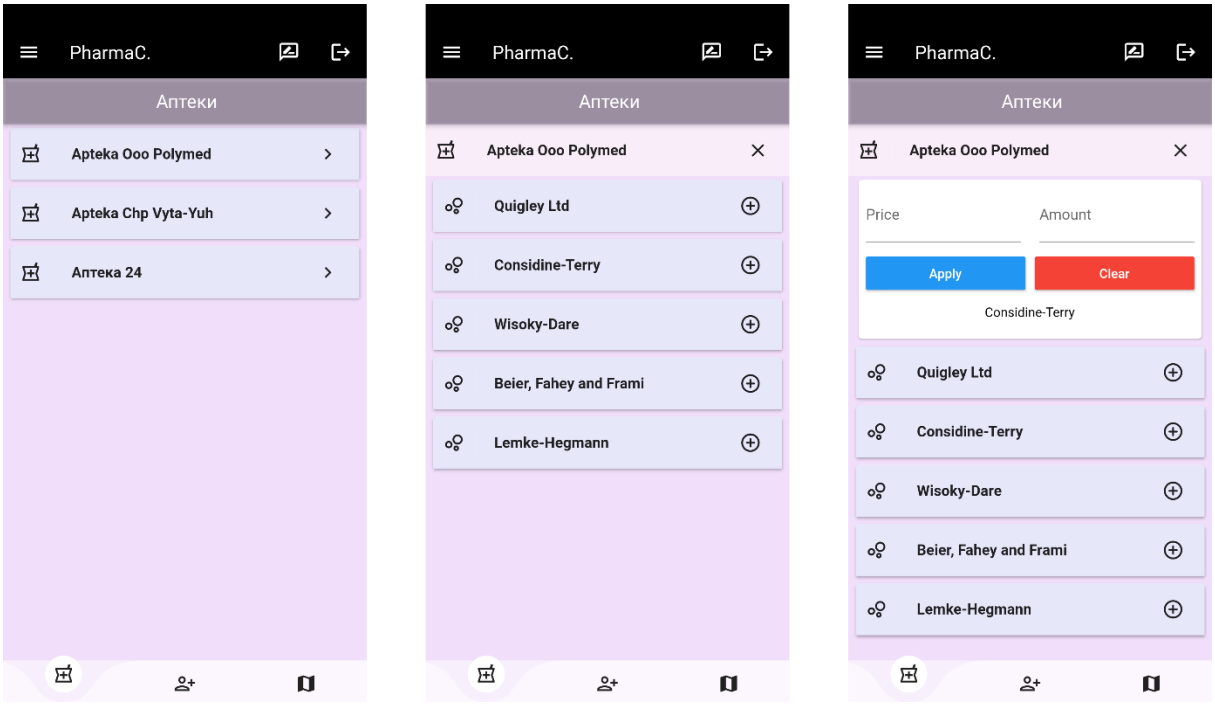

Рисунок 5.2 – Список аптек, товарів та заповнення аудиту по критеріям ціни та кількості

Згідно до правил користування додатком та політикою компанії, робітник може проводити аудит лише у відзначений до того час. У разі успішного заповнення, чи виключення щодо заповнення аудиту, користувач побачить відповідне повідомлення (рис. 5.3).

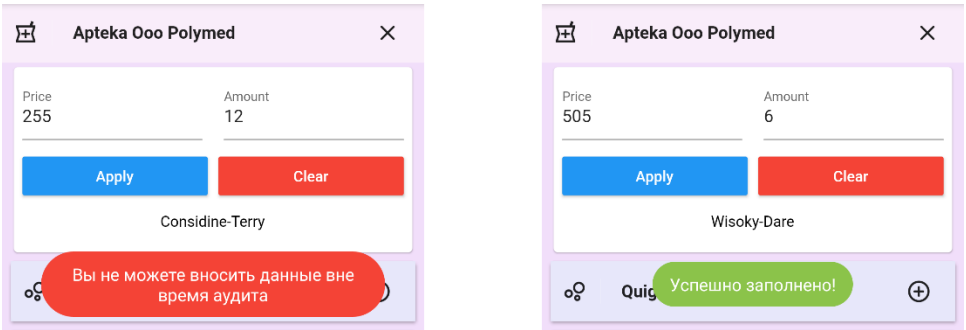

Рисунок 5.3 – Повідомлення: помилка, успішне заповнення аудиту

Також користувачу доступна праця з картами, при вході в систему відображається його місцезнаходження на карті. Є можливість знаходити найближчу аптеку, а також банк, якщо є необхідність (рис.5.4).

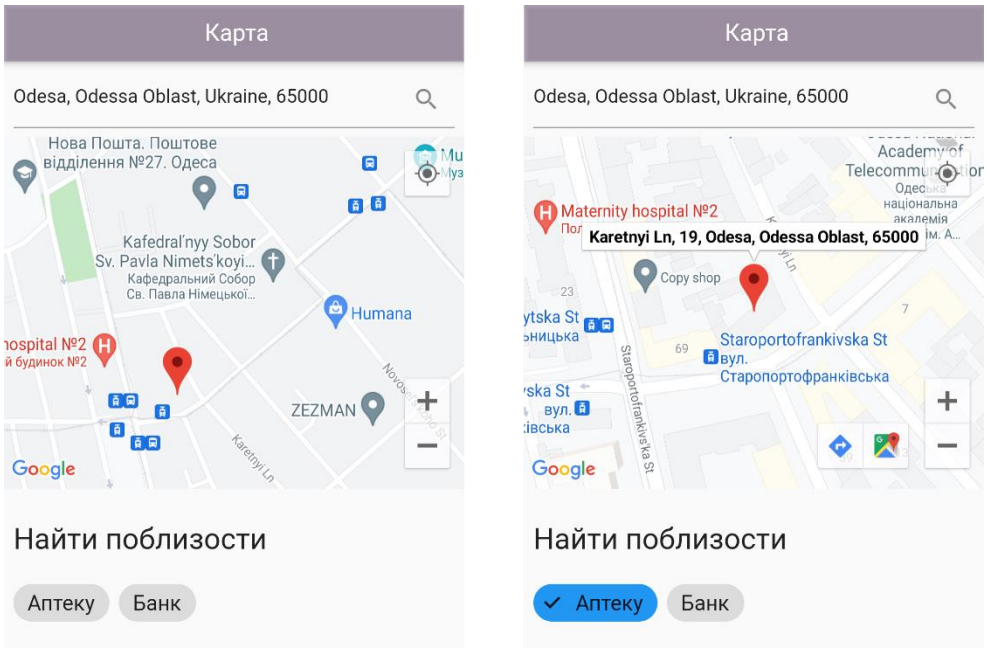

Рисунок 5.4 – Відображення карти: поточне місцезнаходження та пошук найближчої аптеки

Далі можна продемонструємо роботу з аудитами. По-першу, робітник бачить список всіх аудитів, які він зобов’язується виконати. Зеленим виділені ті аудити, що вже виконані, червоним – ті, що не виконані через затримку по графіку, голубим – заплановані аудити (рис. 5.5).

Рисунок 5.5 – Список аудитів робітника

Маніпуляції, які може виконувати робітник з аудитом: делегування іншим співробітникам - у випадку, якщо не вистачає часу на виконання аудиту по робочому плану; завершення аудиту – у разі заповнення звіту продаж аудиту у запланований час. При делегуванні обираються вільні співробітники у заданий час й інформуються можливістю виконати додаткову роботу системою повідомлень як відображено на рисунку 5.6.

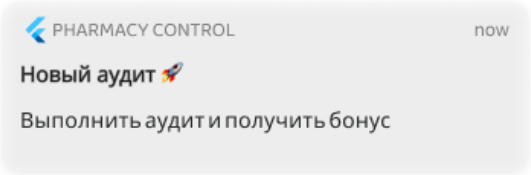

Рисунок 5.6 – Повідомлення при делегуванні

Представник має можливість перегляну історію своїх надходжень: бонусів і штрафів, а також зарплатню за поточний місяць як представлено на рисунку 5.7.

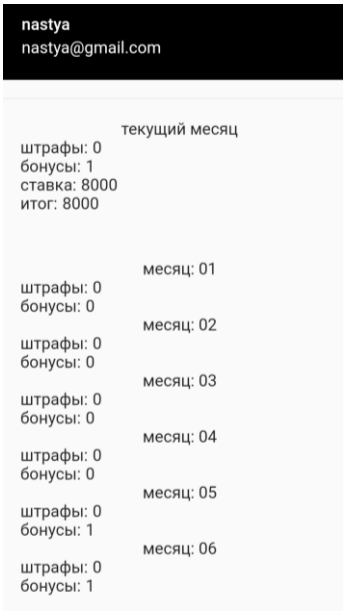

Рисунок 5.7 – Перегляд надходжень представника

Щодо можливостей адміністратору в такій системі, в нього з’являється потреба в перегляді статистик. Для цього на інтерфейс було виведено сторінку з функціоналом, що дозволяє обирати яку статистику необхідно дізнатися, а також обирати вхідні дані для обробки заданих статистик (рис.5.8). В якості тестування запитів статистики, пропоновано переглянути дані щодо представників: штрафи набрані за весь час. Крім цього, запропоновано переглянути дані щодо продажу аптеки за обраний місяць.

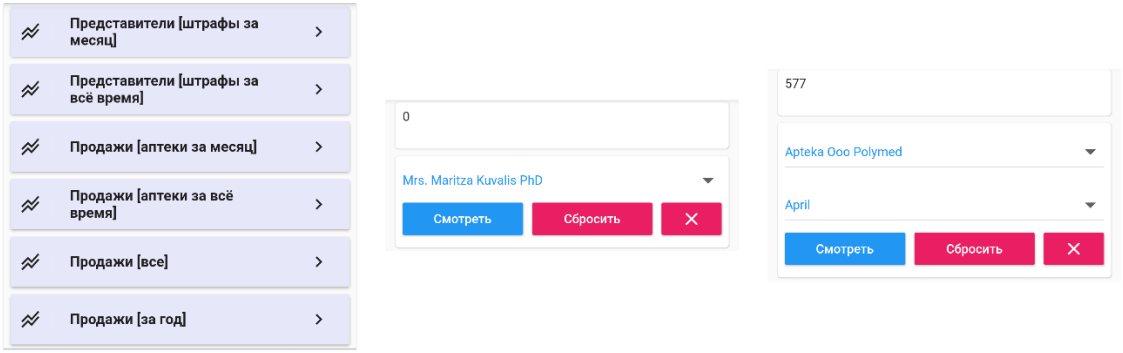

Рисунок 5.8 – Статистика: вибір та перегляд

Для контролю представників також використовується відстеження геолокації. При відкритті карт керівник може побачити поточне, чи останнє місце перебування працівника (рис.5.9).

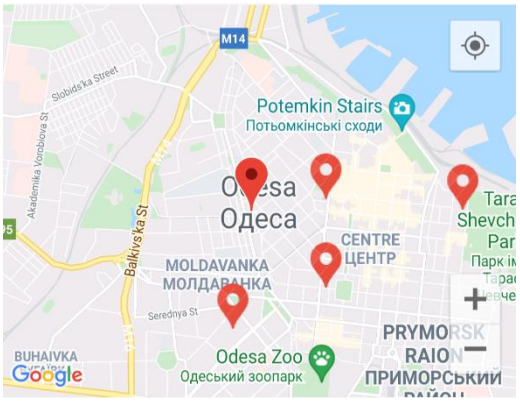

Рисунок 5.9 – Відстеження локації представників

Деякі можливості, що доступні адміністратору згадувалися у Розділі 4, а саме створення нових та перегляд виконаних аудитів на рисунках 4.4-4.6.

Звертаючись до веб-сайту такої системи, користувач при авторизації, будучи в ролі звичайного робітника, бачить наступний екран, що повідомляє за неможливість використовувати сайт, та пропонує завантажити мобільний додаток (рис5.10).

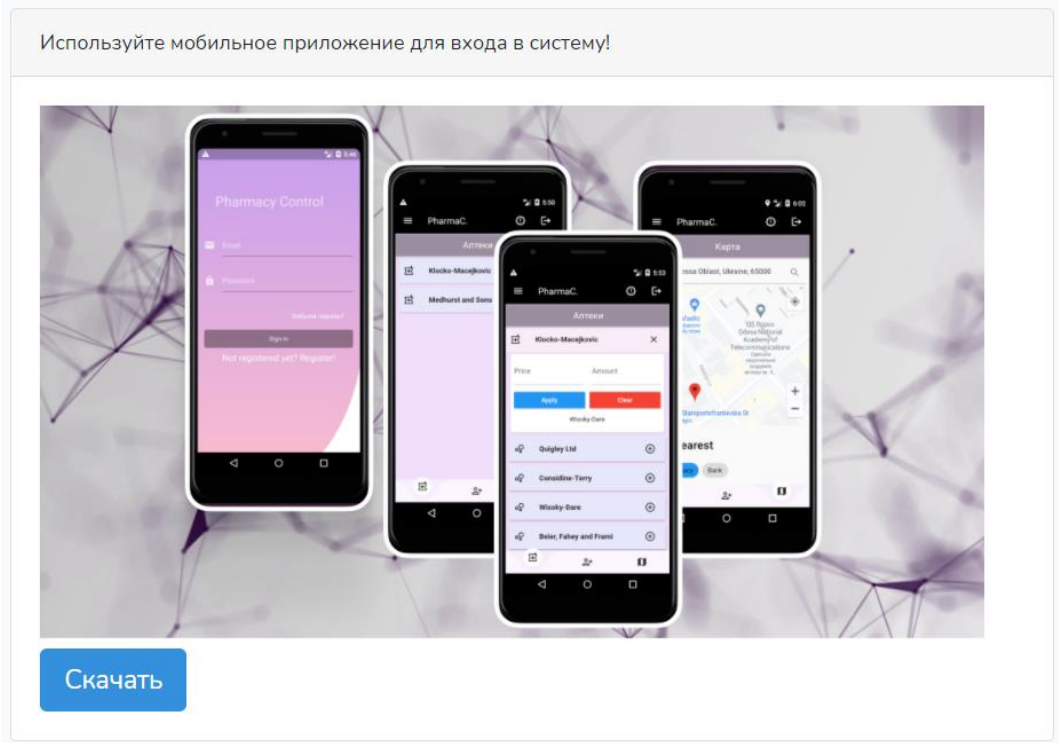

Рисунок 5.10 – Екран для співробітника

Адміністратор при авторизації успішно входить в систему. В його доступі аналогічний функціонал згідно мобільному додатку (рис. 5.11).

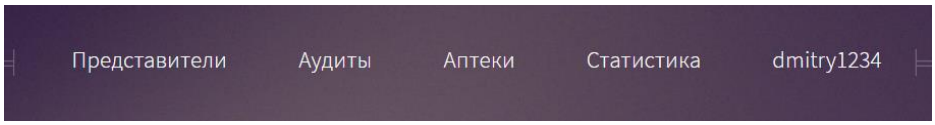

Рисунок 5.11 – Меню для адміністратора

При переході до пункту меню «статистика» користувачу доступні розділи для вибору наступних статистик, що представлені на рисунку 5.12

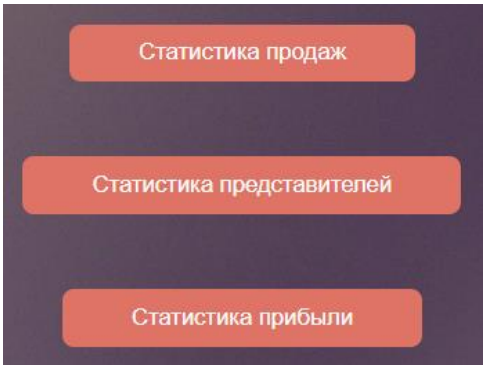

Рисунок 5.12 – Розділи для вибору представлення статистики

Для демонстрації функціоналу в якості прикладу послугує статистика: продажі обраної аптеки за обраний місяць (рис.5.13).

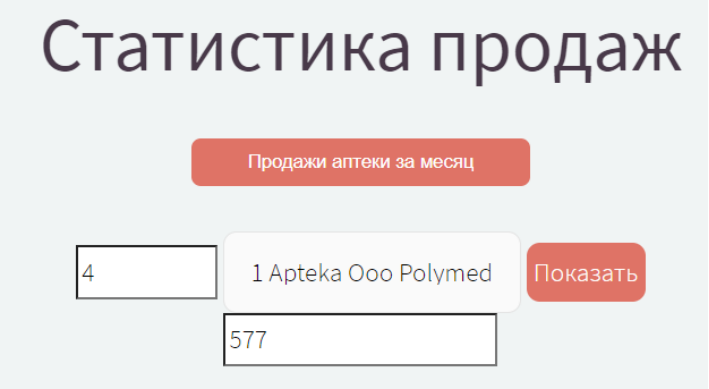

Рисунок 5.13 – Статистика продажу аптеки за місяць.

Подальші можливості для адміністратору на веб-сайті є аналогічними з можливостями у мобільному додатку.

ВИСНОВОК

В результаті реалізації інформаційної системи як внутрішньої, так і зовнішньої частин, сформульовані цілі створення проекту «Система моніторингу співробітників фармацевтичної компанії» були відтворені, визначено та виконано перелік завдань, які вирішуються в розглянутій ПрО. Визначено вимоги до збережених даних і спроектована база даних для використання в інформаційній системі.

Для створення проекту була використана трирівнева архітектура клієнт-сервер, що підходить під шаблон проектування MVC. Розробка виконувалася за допомогою таких інструментів: СУБД PostgreSQL, мова програмування PHP, Laravel Framework, Flutter.

Створено мобільний додаток, який дає можливість зручно маніпулювати даними в залежності від доступу до системи, яка реалізована шляхом розмежування ролей і привілеїв. Окрім цього, сайт, до якого має доступ лише адміністратор системи, що пропонує такий самий функціонал як і мобільний додаток.

Для проекту було побудовано приємний, логічний, структурований інтерфейс, який зрозумілий, як працівникові, так і адміністратору.

За рахунок використання в створеній інформаційній системі трирівневої архітектури клієнт-сервер була досягнута гнучкість і слабка залежність між рівнями програмного забезпечення, через що додаток буде легко розширюватися при необхідності. Реалізація нових завдань, що відносяться до даної області, буде гнучкою і прийнятною.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Зміна таблиць БД. [Електронний ресурс] – Режим доступа: https://postgrespro.ru/docs/postgresql/9.6/ddl-alter
2. Валідація Laravel. [Електронний ресурс] – Режим доступа: https://laravel.com/docs/7.x/validation
3. Документація фреймворка Laravel [Електронний ресурс] – Режим доступа: https://laravel.com/docs/5.5
4. Керівництво по PHP. Розширення для роботи з базами даних. Рівні абстракції PDO. [Електронний ресурс] – Режим доступа: https://www.php.net/manual/ru/pdo.connections.php
5. Визначення прав доступу БД. [Електронний ресурс] – Режим доступа: https://postgrespro.ru/docs/postgresql/9.6/sql-grant
6. Сесії Laravel. [Електронний ресурс] – Режим доступа: https://laravel.su/docs/5.4/session
7. Створення функцій SQL. [Електронний ресурс] – Режим доступа: https://postgrespro.ru/docs/postgresql/9.5/sql-createfunction
8. CSRF-захист користувальницької сесії. [Електронний ресурс] – Режим доступа: https://laravel.ru/docs/v5/csrf
9. В. В. Малый, Л. П. Дорохова, С. В. Жадько, Учебное пособие для самостоятельной работы студентов по дисциплине менеджмент и маркетинг в фармации . - Х. : НФаУ, 2015. - 505 с.
10. Юрий Кочинев, Аудит: теория и практика. - Питер, 2010. - 421 с.
11. Сергей Вячеславович Пауков, Руководство для медицинского представителя фармацевтической компании. - Геотар-Медицина, 2007. - 458 с.
12. Павел Фельдман, Девять основных ошибок детейлинга на фармрынке. - Ridero, 2020. - 34 с.

13. Павел Фельдман, Как измерить эффективность работы с аптекой. Практические советы. - Ridero, 2019. - 80 с.
14. Павел Фельдман, KPIs для оценки работы медицинского представителя. - Ridero, 2019. - 70 с.
15. Олег Крышкин, Настольная книга по внутреннему аудиту: Риски и бизнес-процессы. - Альпина Паблишер. - Москва, 2013. - 80 с.

ДОДАТОК А

ЗАПИТИ НА СТВОРЕННЯ ТАБЛИЦЬ БАЗ ДАНИХ

```
class CreatePharmaciesTable extends Migration
{
    public function up()
    {
        Schema::create('pharmacies', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->text('nameofpharm');
            $table->text('addressofpharm');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('pharmacies');
    }
}
```

// створення таблиці 'pharmacies'

```
class CreateEmployeesTable extends Migration
{
    public function up()
    {
        Schema::create('employees', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->text('position');
            $table->text('fullname');
            $table->integer('salaryrate');
            $table->text('geolocation');
            // $table->text('news');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('employees');
    }
}
```

// створення таблиці 'employees'

```
class CreateProductsTable extends Migration
{
    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->integer('price');
            $table->text('nameofproduct');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('products');
    }
}
```

// створення таблиці 'products'

```
class CreateSalariesTable extends Migration
{
    public function up()
    {
        Schema::create('salaries', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->tinyInteger('month');
            $table->tinyInteger('year');
            $table->integer('salary');
            $table->integer('counteroffines');
            $table->unsignedInteger('employee_id');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('salaries');
    }
}
```

// створення таблиці 'salaries'

```
class CreateAuditsTable extends Migration
{
    public function up()
    {
        Schema::create('audits', function (Blueprint $table) {
```

```
            $table->bigIncrements('id');
            $table->unsignedInteger('pharmacy_id');
            $table->timestamp('dateandtime');
            $table->unsignedInteger('employee_id');
            $table->boolean('delegate');
            $table->boolean('delay');
            $table->boolean('done');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('audits');
    }
}
```

// створення таблиці 'audits'

```
class CreatePurchasesTable extends Migration
{
    public function up()
    {
        Schema::create('purchases', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->unsignedInteger('pharmacy_id');
            $table->unsignedInteger('audit_id');
            $table->unsignedInteger('product_id');
            $table->integer('price');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('purchases');
    }
}
```

// створення таблиці 'purchases'

```
class CreateSuppliesTable extends Migration
{
    public function up()
    {
        Schema::create('supplies', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->unsignedInteger('pharmacy_id');
            $table->date('dateofsupply');
```

```
            $table->unsignedInteger('product_id');
            $table->integer('price');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('supplies');
    }
}
```

// створення таблиці 'supplies'

```
class CreateNewsempsTable extends Migration
{
    public function up()
    {
        Schema::create('newsemps', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->unsignedInteger('employee_id');
            $table->unsignedInteger('audit_id');
            $table->text('trextofmessage');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('newsemps');
    }
}
```

// створення таблиці 'newsemps'

ДОДАТОК Б

ВИХІДНИЙ КОД ОСНОВНИХ КЛАСІВ

Клас, який відповідає за роботу сторінки з аудитами, а також запитів API щодо аудитів. При ініціалізації функції __invoke() отримує всі дані з БД про аудити та повертає вигляд сторінки аудиту для адміністратору.

```
class AuditsController extends Controller
{
    public function __invoke()
    {
        return view('audits',['audits'=>\App\Audit::all()]);
    }
    public function get_ApiAudits()
    {
        return response()->json([
            'audits' => \App\Audit::all(),
        ], 200);
    }
     // отримує список всіх існуючих аудитів
    public function get_ApiAuditUser(Request $request)
    {
        $user_id = $request->only('user_id');
        $all_audits = \App\Audit::all();
        $userN = $user_id['user_id'];
        $auditUsr   =   \App\Audit::all()->where('employee_id',
'=', $userN);
        $pharm = array();
        for($a=0; $a<count($auditUsr); $a++){
            $name_future  =  \App\Pharmacy::all()->where  ('id',
'=', $auditUsr->pluck('pharmacy_id')[$a]);
            array_push($pharm,                    $name_future-
>pluck('nameofpharm'));
        }
        return response()->json([
         'auditUser' => $auditUsr,
         'pharm' => $pharm
        ], 200);
    }
     // отримує аудити конкретного представника
```

```
    public function put_ApiAuditUserDelay(Request $request)
    {
        $delay = $request->only('delay');
        $auditId = $request->only('auditId');
        $audit    =    \App\Audit::find(1)->where('id',    '=',
$auditId['auditId'])->first();
        $audit->delay = true;
        $audit->save();
        return response()->json([
         'delay' => 'changed to true'
        ], 200);
    }
```

// отримує аудити з затримкою конкретного представника

```
    public function get_audit_dateTime(Request $request)
    {
        $currDate = Carbon::now()->setTimezone('Europe/Kiev')-
>addHours(3);
        $currDateSTR = str_replace("T", " ", $currDate);
        $datetime = $request->only('currentTime');
        $pharm_nameJson = $request->only('namePharm');
        $pharm_id = \App\Pharmacy::all()
        ->where('nameofpharm',                              '=',
$pharm_nameJson['namePharm'])
        ->pluck('id');
        $empl_idJson = $request->only('emp_id');
        $auditDateTime = \App\Audit::all()
        ->where('dateandtime', '<=', $datetime['currentTime'])
        ->where('dateandtime', '<', $currDateSTR)
        ->where('employee_id', '=', $empl_idJson['emp_id'])
        ->where('pharmacy_id', '=', $pharm_id[0])
        ->where('delay', '=', false);
        $pharm_name = \App\Pharmacy::all()
        ->where('id',           '=',           $auditDateTime-
>pluck('pharmacy_id')[0])
        ->pluck('nameofpharm');
        return response()->json([
         'auditDateTime' => $auditDateTime,
         'mytime' => $datetime['currentTime'],
         'carbon' => $currDateSTR,
         'pharm_id' => $pharm_id[0],
         'pharm_name' => $pharm_name[0],
        //  'user_id_audit' => $user_id_audit
```

```
        ], 200);
    }
```

// дозволяє дізнатися запланований аудит з поточного часу з запасом у 3 години

```
    public function set_doneAudit(Request $request)
    {
        $done = $request->only('done');
        $audit_id = $request->only('audit_id');
        $save = \App\Audit::where("id", $audit_id['audit_id'])-
>update(["done" => $done['done']]);

        return response()->json([
         'doneAudit' => 'changed to done'
        ], 200);
    }
```

// дозволяє завершити аудит

```
    public function createAuditAdmin(Request $request)
    {
        $employeeName = $request->only('nameEmployee');
        $emp_id = \App\Employee::all()
        ->where('fullname',                                '=',
$employeeName['nameEmployee'])
        ->pluck('id');
        $pharmacyName = $request->only('nameofpharm');
        $pharm_id = \App\Pharmacy::all()
        ->where('nameofpharm',                             '=',
$pharmacyName['nameofpharm'])
        ->pluck('id');
        $date = $request->only('date');
        $date = $date['date'];
        $time = $request->only('time');
        $time = $time['time'];
        $dateandtime = $date. ' ' . $time;
        $audit = new Audit();
        $audit->pharmacy_id = $pharm_id[0];
        $audit->dateandtime = $dateandtime;
        $audit->employee_id = $emp_id[0];
        $audit->delegate = false;
        $audit->delay = false;
        $audit->done = false;
        $audit->save();
```

```
        return response()->json([
         'doneAudit' => 'changed to done'
        ], 200);
    }
     // дозволяє адміністратору створити аудит
    public function findDoneAudit(Request $request)
    {
        $audits = \App\Audit::all()
        ->where('done', '=', true);
        return response()->json([
         'doneAudits' => $audits,
        ], 200);
    }
     // дозволяє адміністратору знайти виконані аудити
}
```

PurchasesController і його основний метод set_Purchases, який дозволяє представнику заповнити аудит продажу аптеки. Оператор new надає можливість створити новий об’єкт моделі Purchase, через що надалі з’являється можливість зберегти отримані дані до БД.

```
class PurchasesController extends Controller
{
    public function set_Purchases(Request $request)
    {
        $nameofpharm = $request->only('nameofpharm');
        $pharmacy          =          \App\Pharmacy::find(1)-
>where('nameofpharm',    '=',    $nameofpharm['nameofpharm'])-
>first();
        $pharmacy_id = $pharmacy->id;
        $nameofproduct = $request->only('nameofproduct');
        $product           =           \App\Product::find(1)-
>where('nameofproduct', '=', $nameofproduct['nameofproduct'])-
>first();
        $product_id = $product->id;
        $priceofproduct = $request->only('priceofproduct');
        $price = $priceofproduct['priceofproduct'];
        $amountofproduct = $request->only('amountofproduct');
        $amount = $amountofproduct['amountofproduct'];
        $audit = $request->only('audit_id');
        $audit_id = $audit['audit_id'];
```

```
            $purchase = new Purchase();
            $purchase->pharmacy_id = $pharmacy_id;
            $purchase->audit_id = $audit_id;
            $purchase->product_id = $product_id;
            $purchase->price = $price;
            $purchase->amount = $amount;
            $purchase->save();
            return response()->json([
                'set purchase' => 'SUCCESS'
            ], 200);
        }
    }
```