МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.
«_____» _____2022 р.

ДИПЛОМНИЙ ПРОЄКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Програмний додаток моніторингу руху громадського транспорту

Виконавець: Бойко Б.М.

Керівник: Росінська Г.П.

Нормоконтролер: Тупота Є.В.

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо–професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри
Литвиненко О.Є.
«_____» _____ 2022 р.

ЗАВДАННЯ

на виконання дипломної роботи (проєкту)

Бойко Богдана Миколайовича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи (проєкту) Програмний додаток моніторингу руху громадського транспорту
затверджена наказом ректора від «15» лютого 2022 р. № 251/ст.
2. Термін виконання роботи (проєкту): з 16.05.2022 по 19.06.2022
3. Вихідні дані до роботи (проєкту): *Java*, середовище розробки *Intellij IDEA*
4. Зміст пояснювальної записки:
   1) аналіз систем управління перевезень та технологій розробки веб систем;
   2) алгоритмічна розробка програмного додатку;
   3) практична реалізація програмного модулю.
5. Перелік обов'язкового графічного (ілюстративного) матеріалу:
   1) Архітектура *Spring Framework*
   2) Архітектура бази даних системи
   3) Діаграма класів програмного компоненту
   4) Вікно авторизації користувача
   5) Сторінка функціоналу додавання нового маршруту

6. Календарний план–графік

№ пор	Завдання	Термін виконання	Примітка
1	Провести аналіз літератури за темою дипломної роботи та аналіз аналогів на ринку	16.05.2022 – 20.05.2022	
2	Написати розділ 1	21.05.2022 – 22.05.2022	
3	Провести аналіз та розробку оптимальних алгоритмів роботи програмного модуля	23.05.2022 – 25.05.2022	
4	Написати розділ 2	25.05.2022 – 26.05.2022	
5	Розробити програмний модуль	26.05.2022 – 01.06.2022	
6	Провести тестування отриманого модуля	01.06.2022	
7	Написати розділ 3	01.06.2022 – 02.06.2022	
8	Оформити пояснювальну записку	02.06.2022 – 03.06.2022	
9	Підготувати графічний матеріал	04.06.2022	
10	Отримання допуску до захисту та подача роботи в ДЕК	09.06.2022	

7. Дата видачі завдання: « 16 » травня 2022 р.

Керівник дипломної роботи (проєкту) _____ Росінська Г.П.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Бойко Б.М.
(підпис випускника) (П.І.Б)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту «Програмний додаток моніторингу руху громадського транспорту»: 57 с., 12 рис., 20 літературних джерел, 1 додаток.

Ключові слова: ВЕБ-ЗАСТОСУНОК, *Java*, *Thymeleaf* .

Об’єкт дослідження – система для моніторингу руху громадського транспорту.

Предмет дослідження – програмний додаток моніторингу руху громадського транспорту.

Мета дипломного проєкту – створити веб-застосунок, який надасть користувачеві можливість знайти громадський транспорт по заданому маршруту між будь-якими містами в Україні.

Методи дослідження – теоретичне ознайомлення з існуючими технологіями, які використовують при розробці веб-застосунків з використанням баз даних, мова програмування *Java*.

Результати дипломного проєктування полягають у створенні сервісу який дасть можливість швидкого пошуку міжміського транспорту для людей незнайомих з певним регіоном.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

Java – інтерпретована об'єктно–орієнтована мова програмування високого рівня зі строгою динамічною типізацією та кросплатформенністю

IDE – комплекс програмних засобів, який використовується для розробки програмного забезпечення

API – це набір способів і правил взаємодії та обміну даними між різними програмами

URL – стандартна адреса певного ресурсу в інтернеті

CSRF – тип веб атаки, що призводить до виконання певних дій від імені користувача на веб-сторінці де останній аутентифікований

DTO – шаблон проєктування (data transfer object)

БД – база даних

СУБД – система управління базами даних

ВСТУП

Територіальний розвиток міст в усі часи обумовлювався швидкісними характеристиками масових міських пересувань. Так, для середньовічного міста, в якому був відсутній масовий транспорт і пересування здійснювалися пішки зі швидкістю сполучення 4 км/год, максимальні розміри міста не перевищували 12 – 15 км2. За умови використання кінської тяги на залізниці (V= 8 км/год) розміри міста збільшуються до 50 км2, при використанні звичайного вуличного транспорту зі швидкістю сполучення V= 16 км/год розміри міста поширюються до 200 км2. Підвищення швидкості сполучення до 25 км/год шляхом використання експресних ліній вуличного транспорту дозволило збільшити територію міста до 500 км2, а з використанням метрополітену і міських залізниць (V= 35 – 40 км/год) міста досягають розмірів 1000 – 1250 км2.

Якщо прослідкувати еволюцію плану будь-якого великого міста протягом останніх 200 – 300 років, можна наочно побачити зв’язок між прогресом транспортних засобів і територіальним розвитком міста. Існує цілком чіткий взаємозв’язок розвитку міста і транспорту: місто росте до певних розмірів і використовує певний вид транспорту, потім у місті починають виникати труднощі в транспортному обслуговуванні, що вимагає поліпшення транспортної системи. Розвиток транспорту поширює можливості розселення та збільшує дальність поїздок, сприяє розвитку міста [2].

Із зростанням чисельності населення міст та їхньої території обсяг роботи транспорту зростає швидше за його територію, бо росте так звана «транспортна рухомість» населення (середнє число поїздок, що припадає на одного жителя за рік), а також підвищується дальність поїздки.

Стабільна робота усіх видів транспорту є невід'ємною умовою нормального функціонування економіки України. Тим часом за останні роки становище в транспортно-дорожньому комплексі країни почало трохи покращуватись. Внаслідок зростання обсягів перевезень збільшуються прибутки транспортних

підприємств, що дає можливість для розширення функцій, які зможуть задовільнити більше потреб користувачів. Також з розвитком доріг на міжміському рівні збільшується швидкість та якість перевезень. Тому, я думаю, що в майбутньому збільшиться попит на громадський міжміський транспорт, це ж в свою чергу призведе до необхідності розкладу руху маршрутів та автоматизація роботи громадського транспорту.

Людина на відміну від системи не може порахувати велику кількість варіантів і, як правило, зупиняється на більш легких для планування. Результати планування системи завжди будуть кращими за результати людини, адже сучасні технології дозволяють виробляти мільйони розрахунків в хвилину. Головне, щоб на вході були якісні дані і правильні налаштування. Системи автоматизації транспортної логістики, які призначена для управління та моніторингу щоденних перевезеннь людей досить непогано покращують продуктивність бізнесу та подальший його розвиток. Весь набір маршрутів оптимізується в розрізі вартості, тому забезпечує мінімальні витрати з дотриманням всіх обмежень і заданих умов.

Об’єкт дослідження – система для моніторингу руху громадського транспорту.

Предмет дослідження – програмний додаток моніторингу руху громадського транспорту.

Мета дипломного проєкту – створити веб застосунок, який дасть користувачеві можливість знайти громадський транспорт по заданому маршруту між будь-якими містами в Україні.

У результаті дипломного проектування буде розроблено програмний засіб, що дає можливість диспетчерам в системі швидко та легко додавати нових водіїв та маршрути за вказаними на карті координатами або містами, також є можливість отримати звіт про витрати та доходи за певним маршрутом.

Таким чином, диспетчер має можливість:

- додавати нові маршрути;
- реєструвати в системі нових водіїв;
- переглядати журнал медоглядів транспорту;

- додавати новий розклад за певним маршрутом;
- створювати та редагувати поїздки;
- вести облік витрат пального;
- вести облік відпрацьованих водієм кількості годин.

Відповідно до поставленої мети необхідно вирішити наступні задачі:

- проаналізувати існуючі системи управління перевезень;
- розглянути існуючі технології та мови програмування за допомогою яких можливо розробити програмний додаток;
- на базі обраних технологій розробити додаток з графічним інтерфейсом;
- протестувати розроблений програмний додаток.

РОЗДІЛ 1

АНАЛІЗ СИСТЕМ УПРАВЛІННЯ ПЕРЕВЕЗЕНЬ ТА ТЕХНОЛОГІЙ РОЗРОБКИ ВЕБ СИСТЕМ

1.1. Веб сайти та їх застосування в мережі інтернет

Веб-сайт — це сукупність логічно зв'язаної гіпертекстової інформації, оформленої у вигляді окремих сторінок і доступної в мережі Інтернет [3].

Подібне визначення веб-сайту було правильним на початку існування Інтернету, коли мережа і веб-сайти використовувалися в основному як розважальна система. Для створення веб-сайту було потрібне лише знання мови гіпертекстової розмітки — *HTML*. Якщо ж сторінка надавала якісь програмні засоби — це були виключно засоби, що міг надати сервер, на якому розташований веб-сайт.

Веб-сайт виконує такі основні завдання:

- реклама продукції, послуг, ідей. Правильно зроблений веб-сайт із легкістю приведе клієнта до висновку про необхідність покупки товару, або послуг, або ідей, що пропагуються на ньому;
- продаж товарів, послуг, інформації, ідей. У сучасної людини немає багато часу для ходіння по магазинах. Тому можливість замовлення товарів і послуг, не відходячи від комп'ютера, значно розширює можливості і клієнта, і продавця;
- безкоштовне надання інформації або послуг. Насправді надання інформації або послуг — це засіб залучення відвідувачів до даного ресурсу для здобуття, наприклад, статистичної інформації або ж для показу реклами;

1.2. Аналіз алгоритму реалізації програмних додатків моніторингу руху транспорту

Загальний алгоритм розробки веб систем зазвичай стандартний і містить в собі 3 основних етапи:

1. проєктування;
2. розробка;
3. видання та підтримка.

На етапі проєктування визначається мета застосунку та його функціонал, стек необхідних технологій та архітектура бази даних.

Засобами розробки будь-якої веб-системи є мова програмування на якій буде писатись програмний код, адже в залежності від розміру системи та її продуктивності деякі мови матимуть значні переваги над іншими. Також не менш важливим фактором є вибір сховища для зберігання даних, адже деякі системи потребують швидкого доступу до даних, в такому випадку краще обрати документноорієнтовану БД, а коли необхідні строго типізовані данні то тут згодиться реляційна модель БД.

До реалізації проєкту переходять тільки після того, як буде спроєктована архітектура бази даних та буде створений макет проєкту для подальшого написання коду.

Розробка самий складний етап реалізації проєкту, до якого входить велика кількість кроків, кожен з яких впливає на працездатність проєкту, кількість помилок та час розробки.

Розглянемо сучасні методи розробки веб-застосунків з застосуванням клієнт-серверної архітектури, а саме розробка з використанням мови *Java* з фреймворком *Spring Framework* та *ORM Hibernate* в інтегрованому середовищі розробки *Intellij IDEA.* Також для динамічного відображення вмісту на веб сторінці розглянемо один із видів шаблонізаторів, а саме *Thymeleaf* в застосуванні з додаванням технологій *HTML CSS* та *JavaScript*. Для зберігання даних проєкту розглянемо як варіант реляційну СУБД Postgree *SQL*.

Розроблений проєкт повинен мати наступні функції:

– авторизація та аутентифікація користувача та визначення його ролі;

– для неавторизованих користувачів повинна бути можливість швидкого пошуку маршрутів за початковою та кінцевою точкою на певну дату;

– для авторизованого користувача з роллю «Диспетчер» можливість додавання нових водіїв маршрутів та транспортних засобів, перегляд та завантаження звітів за поточний місяць про прибуток за певним маршрутом, або звіт про кількість відпрацьованих водієм годин. Також було додано журнал ведення стану здоров'я водіїв;

– Для авторизованого користувача з роллю «Водій» є можливість перегляду призначених поїздок а також сторінка із запропонованими поїздками для подальшого підтвердження або відмови;

– Графічне відображення маршрутів із використанням технології *Google Maps*;

Перше, що потрібно вирішити на початку розробки, це те як користувачу буде зручно та зрозуміло користуватись сайтом, тобто *UI* та *UX*. Після цього продумати основні етапи розробки, а саме: перенесення моделі бази даних в код, написання бізнес логіки для покриття всього функціоналу який буде необхідний, обмеження користувача в залежності з його роллю та забезпечення безпеки і звичайно з'єднання фронт-енду з бек-ендом.

Після написання коду потрібно буде зробити скрипти для початкового наповнення бази даних та скрипти для бекапу вже існуючої інформації, які в разі збою в системі дадуть можливість повернутися до попередньої працюючої версії системи.

Далі розпочинається етап написання юніт тестів, які покриють більшу частину коду та в майбутньому передбачать зміну в бізнес логіці при зміні коду. Це забезпечить систему від появи нових багів або неточностей в логіці. Якщо всі тести успішно проходять можна переходити до етапу повного ручного тестування проєкту: перевірка тестовим користувачем всіх процесів в системі та очікуваного результату певного функціоналу.

Фінальним етапом розробки є контейнеризація веб застосунку, задля спрощеного розгортання окремих частин на веб-сервері та майбутня його підтримка.

Коли сайт офіційно випущений та розгорнутий на існуючому сервері, його слід підтримувати та своєчасно виправляти помилки, про які будуть сповіщувати користувачі. Також слід не обмежуватись вже існуючим функціоналом а додавати щось нове для залучення нових користувачів та клієнтів які будуть користуватись цим веб застосунком.

Таким чином, стало зрозуміло, що стадії розробки веб систем потребують глибоких знань в розумінні всього процесу, обов'язкове знання мов програмування які застосовуються, розуміння моделі *MVC*(модель-відображення-контролер).

1.3. Аналіз аналогічних проєктів

Для аналізу було знайдено сайти, які реалізують моніторинг руху транспорту:

- європейського виробництва *12Go* [4];
- *busfor.ua,* який розроблений в Україні [3].

12Go – веб застосунок, який дає можливість знайти квитки на будь-який транспорт в Європі, також є підтримка багатьох мов в тому числі і української. Але більше ніяку функціональність він не представляє, тобто він підійде для людей, які хочуть порівняти ціни на проїзд між містами на різних видах транспорту. Інтерфейс форми пошуку за заданим напрямком рис.1.1.

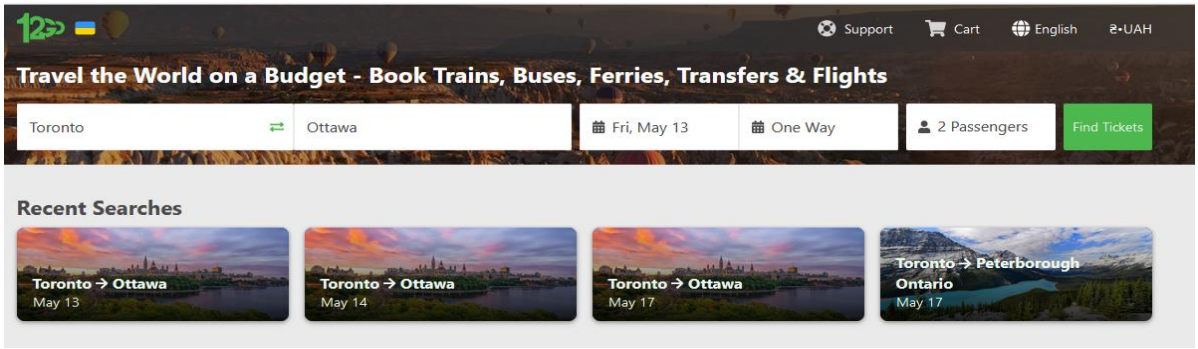

Рис. 1.1. Інтерфейс сайту для пошуку квитків *12Go*

Застосунок *12Go* має недоліки, не здійснюється авторизація та реєстрація користувача, що ставить під питання безпеку всієї системи. Незареєстрований користувач може забронювати всі квитки, що призведе до проблеми із купівлею квитків іншими користувачами. Також багато маршрутів недоступні, що додає проблем із швидким пошуком квитків.

Український веб-додаток *busfor.ua*, зображений на рис.1.2. Цей сайт має особистий кабінет користувача в якому зберігається історія усіх куплених квитків, це додає впевненість та безпеку користувача. Також цей сайт інтегрований з *BlaBlaCar* сервісом, що дає можливість шукати поїздки не тільки із щоденних, а ще й пропозиції попутного руху.

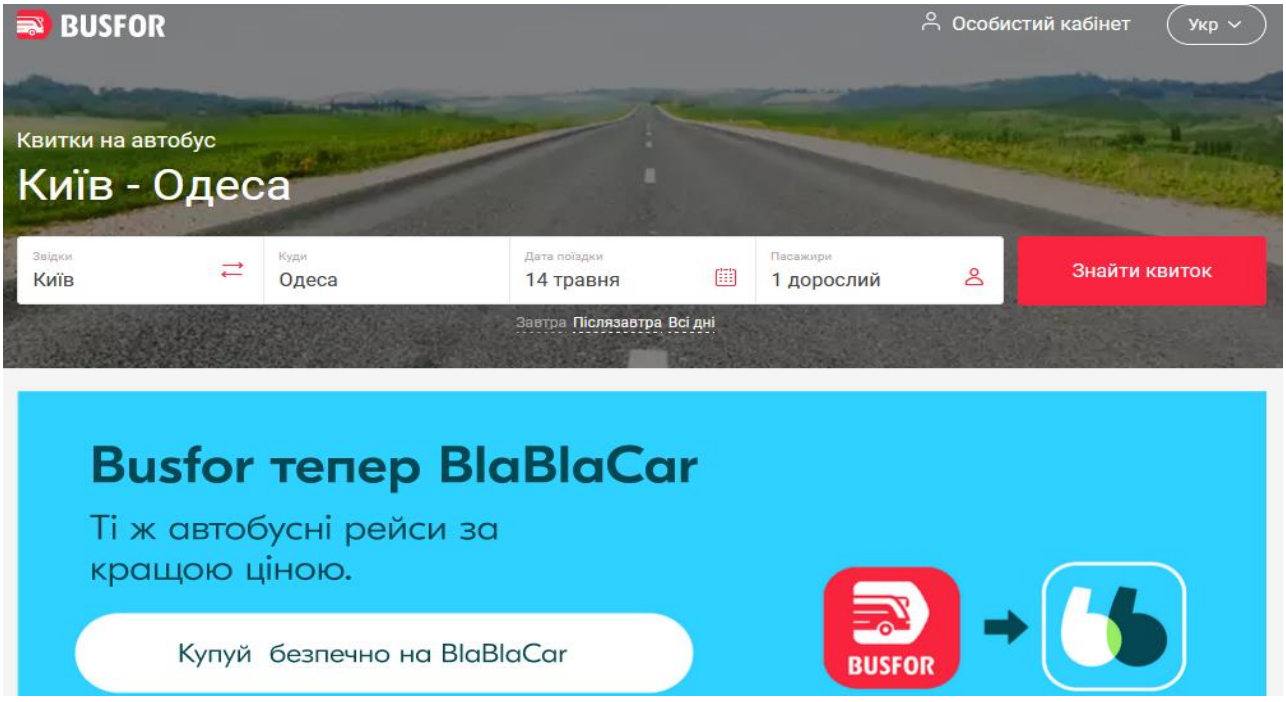

Рис. 1.2. Інтерфейс сайту для пошуку квитків *Busfor.ua*

Сайт також має свої недоліки: немає додаткових сторінок для адміністратора чи менеджера, через які можна буде додавати нові маршрути чи водіїв, генерувати звіти по прибутках. Все це власники цього сервісу роблять власноруч, виконуючи певні запити до бази даних і тому не мають змоги моніторити свою систему.

Розглянуті системи мають і деякі переваги: купівля квитків, велика кількість маршрутів.

Розроблюваний застосунок представляє адміністраторам та диспетчерам додаткові можливості в перегляді звітів по певному маршруту або виду транспорту, також для водія є розрахунок пального на ввесь маршрут та його протяжність, що дасть змогу водіям легше розраховувати свої потреби.

1.4. Інструменти та технології реалізації

У наступних розділах будуть розглянуті наступні технології та інструменти:

– мова програмування *Java*;

– фреймворк *Spring 5* (*Spring Data*, *Spring Security*);

– інтегроване середовище розробки *IntellijIDEA*.

1.4.1. *Java*

Java - високорівнева мова програмування загального призначення з динамічною строгою типізацією і автоматичним управлінням пам'яттю, орієнтована на підвищення продуктивності розробника, читання коду і його якості, а також на забезпечення кросплатформенності та захисту.

Мова програмування *Java* зародилася в 1991р. в лабораторіях компанії *Sun Microsystems inc*. Як не дивно, поштовхом для створення *Java* став зовсім не *Internet*. Головним мотивом була потреба в мові програмування, яка не залежала б від платформи (тобто від архітектури) і яку можна було б використовувати для створення програмного забезпечення, яке вбудовується в різноманітні побутові електронні прилади, такі як мобільні засоби зв’язку, пристрої дистанційного управління тощо. Розробка першої робочої версії зайняла 18 місяців і вона мала назву «*Oak*», але 1995 р. проект був перейменований на «*Java*» [10].

Один із ключових принципів розробки мови Java полягав у забезпеченні захисту від несанкціонованого доступу. Програми на *Java* не можуть викликати глобальні функції й одержувати доступ до довільних системних ресурсів, що забезпечує в *Java* рівень безпеки, недоступний для інших мов. Даний рівень безпеки виконання *Java*-програм забезпечує віртуальна машина *Java*, котра

вбудована в операційну систему. Об'єктна модель у *Java* проста і легко розширюється, у той же час, заради підвищення продуктивності прості типи даних *Java* не є об'єктами.

В сучасному світі *Java* знайшла своє застосування на великих та середніх комерційних проєктах. Через те що код який якісно написаний цією мовою досить легко розширювати та масштабувати ця мова програмування дуже популярна у всьому світі, але потребує багато знань навіть для написання невеликої та простої програму. Тому для початківця в цьому напрямку потрібні хороші знання паттернів проєктування та вміння використовувати мікросервісну архітектуру, не менш важливим є досвід роботи з базами даних та частково знання вебу для розуміння того що відбувається після виконання *Java* коду.

1.4.2. *Spring Framework*

Spring Framework або просто *Spring* - один із самих популярних фреймворків для створення веб-застосунків мовою *Java*. Фреймворк - це щось схоже на бібліотеку, але є один момент. Використовуючи бібліотеку, ви просто створюєте об'єкти класів, які в ній є, викликаєте потрібні вам методи, і таким чином виходить потрібний вам результат. Тобто, тут більш імперативний підхід: ви чітко вказуєте у своїй програмі в який конкретний момент, потрібно створити якийсь об’єкт, у який момент викликати конкретний метод тощо. Фреймворк це набагато потушніший інструмент. Ви просто пишете свої класи, прописуєте там частину логіки, а створюєте об'єкти ваших класів і викликаєте методи для вас уже сам фреймворк. Частіше всього, ваші класи імплементують інтерфейси з фреймворка або наслідують класи з нього, таким чином отримана частина вже написана для вашої функціональності. Не обов'язково саме так.

Розробники *Spring* (рис.1.3) намагаються по максимуму відійти від такої жорсткої зв’язку (коли ваші класи напряму залежать від класів/інтерфейсів із цього фреймворка), і використовують для цієї цілі анотації. Якісний дизайн має слабку зв'язаність (*low coupling*) і сильну зв'язність (*high cohesion*). Це означає, що

програмний компонент має невелику кількість зовнішніх зв'язків та відповідає за вирішення близьких за змістом завдань [12].

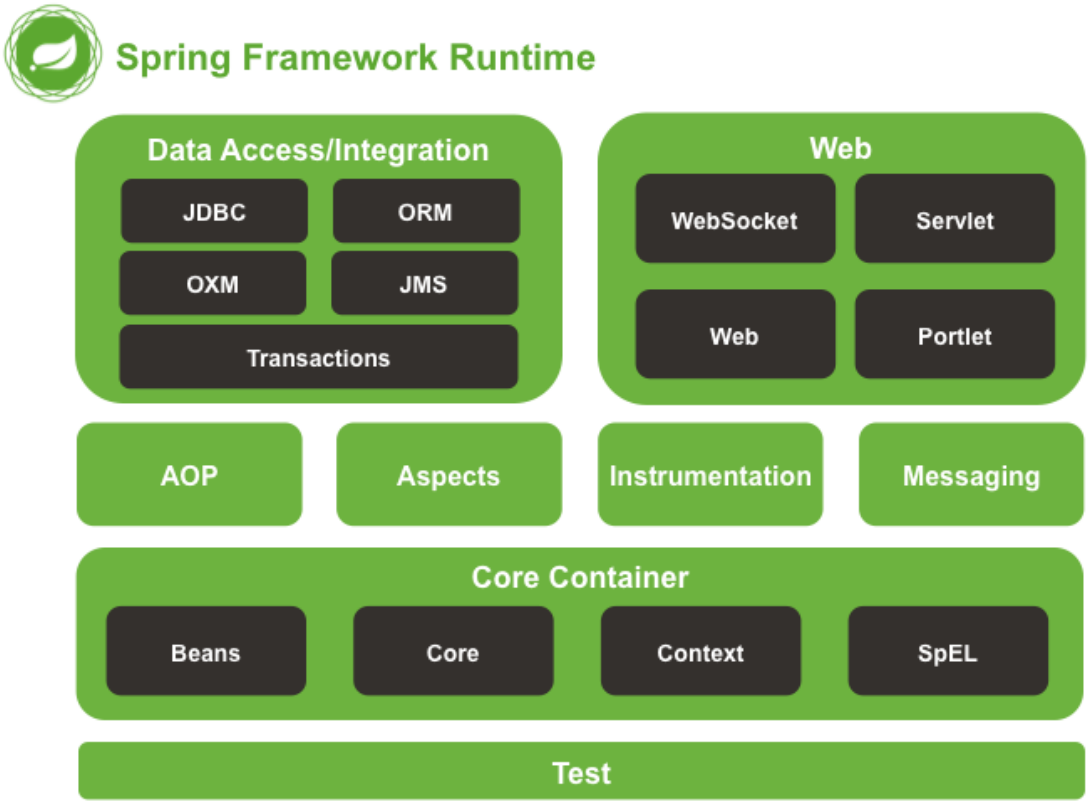

Рис. 1.3. Архітектура *Spring Framework*

Spring має модульну структуру. Це дозволяє підключати тільки ті модулі, які нам потрібні для нашої програми і не підключати ті, якими ми не будемо користуватися. Такий саме підхід і допоміг спрінгу обійти свого конкурента в той час (*EJB*) і захопити лідерство. Тому що програми, що використовують *EJB*, тягли дуже багато залежностей за собою, та й взагалі виходили повільні та неповороткі.

Два основні поняття якими оперує спрінг це *DI* та *IOC*.

IoC (Inversion of Control) – інверсія управління. При використанні бібліотеки ви самі прописуєте у своєму коді якийсь метод якого об'єкта викликати, а у випадку з фреймворками — найчастіше вже фреймворк викликатиме в потрібний момент той код, який ви написали. Тобто тут вже не ви керуєте процесом виконання коду/програми, а фреймворк це робить за вас. Ви передали йому керування (інверсія керування).

Під *DI* розуміють то *Dependency Inversion* (інверсію залежностей, тобто спроби не робити жорстких зв'язків між вашими модулями/класами, де один клас безпосередньо зав'язаний на інший), то *Dependency Injection* (використання залежностей, це коли об'єкти створюєте не ви і потім передаєте їх у свої методи, а за вас їх створює спрінг, а ви йому просто вказуєте куди їх необхідно передати).

Одне з ключових понять у спрінгу – це бін. По суті це просто об'єкт якогось класу. Кожен клас який оголошений з анотаціями *@Service, @Component, @Bean* та інші анотації які використовуються на різних рівнях абстракції є бінам. Всі біни зберігаються в контейнері залежностей, в якому в свою чергу буде побудовано дерево залежностей при першому запуску програми. Далі розглянемо *Spring Data* для роботи з базами даних на рівні абстракції який вище за *JPA*.

Завдання *Spring Data* полягає в тому, щоб забезпечити знайому та послідовну модель програмування на основі *Spring* для доступу до даних, зберігаючи при цьому особливі риси основного сховища даних. Це спрощує використання технологій доступу до даних, реляційних і нереляційних баз даних, фреймворків зі скороченням карт і хмарних сервісів даних. Це загальний проект, який містить багато підпроектів, специфічних для даної бази даних. Проекти розробляються разом із багатьма компаніями та розробниками, які стоять за цими захоплюючими технологіями.

Щоб почати використовувати модель програмування *Spring Data*, необхідно створити власний клас який імплементує спеціальний інтерфейс *JPARepository*. Це дозволить Spring Data знайти цей інтерфейс і автоматично створити для нього реалізацію. Розширюючи інтерфейс, ми отримуємо найбільш релевантні методи *CRUD* для стандартного доступу до даних, доступні в стандартному *DAO*. Щоб визначити більш конкретні методи доступу, Spring *JPA* підтримує досить багато параметрів:

- просто визначити новий метод в інтерфейсі;
- надати фактичний запит *JPQL* за допомогою анотації *@Query*;
- визначати користувацькі запити через іменовані запити *JPA*.

Розглянемо що таке *Spring Security* та можливості цього модулю.

Spring Security — це потужна і добре настроювана система аутентифікації та контролю доступу. Це де-факто стандарт захисту додатків на основі Spring. Spring Security — це фреймворк, який зосереджується на забезпеченні як аутентифікації, так і авторизації для програм *Java*. Як і всі проекти *Spring*, справжня сила Spring Security полягає в тому, як легко його можна розширити, щоб відповідати користувацьким вимогам. Особливості цього модулю полягають у налаштуванні доступів для різних типів користувачів, також є представлення функціоналу для *OAuth2*, який дуже часто використовують великі компанії такі як *Google* та *Amazon*. Також правильне налаштування захистить сайт від кібератак таких як фіксація сесії, підробка міжсайтових запитів.

Щоб налаштувати свій сервіс з допомогою Spring Security необхідно оголосити бін який наслідує клас *WebSecurityConfigurerAdapter* в якому можна перевизначити метод configure та налаштувати відповідно до маршрутизації яка є в проєкті, цей метод надає можливості надавати відповідні доступи авторизованим користувачам, та користувачам які вперше на сайті, також ще є дуже багато функціоналу який можна знайти в документації. Головною особливістю цього модуля є своя унікальність, так як ніяких модуль не має схожого функціоналу з тим що є в *SpringSecurity*.

1.4.3. Інтегроване середовище розробки *IntellijIDEA*.

IntellijIDEA – інтегроване середовище розробки для мови програмування *Java*. Надає засоби для аналізу коду, графічний відладчик, інструмент для запуску юніт-тестів і підтримує веб-розробку з *Spring Framework*. *IntellijIDEA* розроблена компанією *JetBrains* використовуючи мову програмування Java. Це крос-платформенне середовище розробки сумісне з *Windows, macOS, Linux*.

Можливості *IntellijIDEA*:

– допомога при написанні коду:

IntellijIDEA робить розробку максимально продуктивною завдяки функціям автодоповнення і аналізу коду, миттєвої підсвічуванні помилок і швидких виправлень. Автоматичні рефакторингом допомагають ефективно

редагувати код, а зручна навігація дозволяє миттєво переміщатися по проекту;

– зручна навігація:

Розумний пошук дозволяє швидко перейти до будь-якого класу, файлу або символу, а також на відповідне вікно або дії *IDE*. Перехід до вищестоящого методу, тесту, оголошенню, входженню або реалізації здійснюється в одне натискання;

– вбудовані інструменти для розробників:

IntellijIDEA пропонує великий набір інструментів з коробки: вбудований відладчик і інструмент запуску тестів, вбудований термінал, інструменти для роботи з базами даних: *IntellijIDEA* допомагає редагувати *SQL*-код, виконувати запити, переглядати дані і змінювати схеми. *IDE* інтегрована з популярними системами контролю версій, містить вбудований *SSH*-термінал, підтримує можливості віддаленої розробки та віддалені інтерпретатори, а також інтеграцію з *Docker* і *Vagrant*;

– інтерфейс, який можна налаштувати під свої потреби:

Кожен розробник любить налаштовувати інструменти під свої переваги. Тому інтерфейс *IDE* максимально адаптований під потреби користувача. Налаштуйте робоче середовище так, як вам подобається: виберіть відповідну колірну схему і зручні поєднання клавіш, також можна кастомізувати панель загрузки та індексації проєкту;

– модулі:

За роки розробки платформи *IntellijIDEA* було створено більше 50 плагінів, які забезпечують підтримку додаткових систем контролю версій, інтеграцію з інструментами і фреймворками, а також розширюють можливості редактора, наприклад за рахунок емуляції *Vim*. Також ця IDE підтримує більшість форматів файлів які використовуються для різних мов програмування або ж файлів структури передачі даних.

1.5. Висновки до розділу

У даному розділі було розглянуто та проаналізовано технології та інструменти, які досить часто використовують при розробці веб додатків та дозволяють підтримувати архітектуру з розширенням функціональності, а саме:

– мова програмування – *Java*;

– *Spring Framework*;

– інтегроване середовище розробки *IntellijIDEA*.

Також в цьому розділі було порівняно існуючі системи, які виконують схожу функціональність із веб-додатком, який буде розроблятися. Розглягнуто переваги та недоліки, які потім будуть враховані в написанні розроблювальної системи.

Розібрано особливості роботи з фреймворком *Spring* мовою *Java*, а саме робота з рівнем зв’язаним з базою даних, з модулем який відповідає за безпеку системи. Розглянуті функціональні можливості середовища розробки *IntellijIDEA*, яке прискорить розробку даного проєкту та без додаткових програм чи утиліт дасть можливість протестувати всю систему, переглянути базу даних та створити проєкт у віддаленому репозиторії.

РОЗДІЛ 2

АЛГОРИТМІЧНА РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

2.1. Функціональні вимоги

Для того щоб створити мінімально життєздатний продукт було обрано такий функціонал від якого можна буде в майбутньому розширювати бізнес логіку та додавати новий функціонал. Тому після завершення розробки першої версії, система виконує наступні функціональні вимоги:

- завантаження, відображення та створення файлів-звітів в форматі *PDF*;
- можливість створення маршруту який відображається графічно за допомогою використання технологій *Google API*.
- розподіл користувачів за трьома ролями – водій, диспетчер та звичайний користувач та відповідні сторінки відповідно до ролі яку надав адміністратор системи при реєстрації.
- перегляд існуючих поїздок та операції підтвердження або відмови певної поїзди для користувачів які авторизовані як водій;
- для єдиного користувача диспетчера є можливість створення нових маршрутів, додавання водіїв та поїздок, створення записів про стан водія та транспортного засобу та занесення їх в журнал, перегляд журналу;
- швидкий пошук маршрутів за двома містами на обрану дату;
- валідація вхідних даних для всіх форм в системі.

2.2. Структура програмного веб-застосунку

Програмний застосунок розробляємо з використанням сучасних підходів в створенні кросплатформенної системи, з масштабованою архітектурою яку легко підтримувати та розширяти. Тому було обрано багатошарову архітектуру для розробки бек-енду рис.2.1.

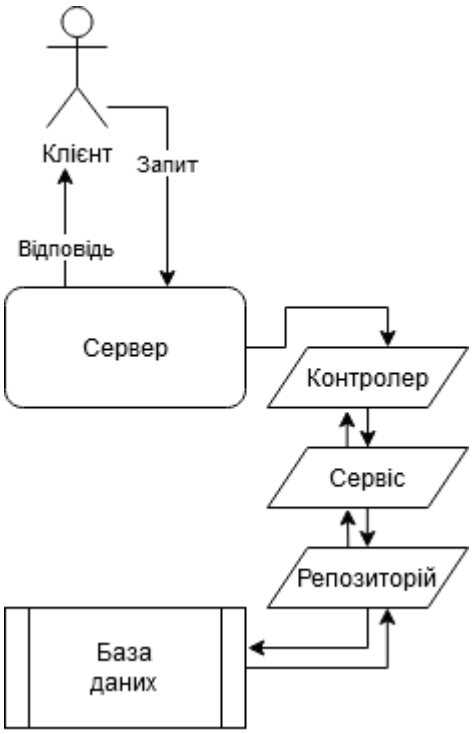

Рис 2.1. Відображення багатошарової клієнт-серверної архітектури

Всі дані які вводить користувач будуть оброблювати контролери які налаштовані для різних шляхів в пошуковій строці та можуть приймати різні параметри для подальшої обробки. Після цього йде шар сервісів. На цьому рівні виконується маніпуляція даними та основна бізнес логіка і результат відправляється на контролер. Але для того щоб отримати певну інформацію з бази даних створюється ще один шар – репозитарій.

На цьому етапі виконуються всі запити в базу даних для подальшої вставки, отримання або оновлення інформації в одній із таблиць. Для передавання необхідної інформації між шарами створюються спеціальні об’єкти, так звані *DTO* або об’єкти передачі даних — це об’єкти, які переносять дані між процесами, щоб

зменшити кількість викликів методів. Головна мета такого об’єкту — скоротити перехід до сервера за рахунок групування кількох параметрів за один виклик. Це зменшує витрати мережі під час таких віддалених операцій.

Іншою перевагою є інкапсуляція логіки серіалізації (механізм, який перекладає структуру об’єкта та дані в певний формат, який можна зберігати та передавати). Він забезпечує єдину точку зміни нюансів серіалізації. Він також відокремлює моделі домену від рівня презентації, дозволяючи обом змінюватися незалежно.

Після розробки серверної сторони, переходимо до етапу прототипування в результаті якого отримуємо готовий дизайн. Чітко підібрана кольорова гама в одному стилі без різких кольорів, робить сайт більш приємним для користування. Не менш важливим є зрозуміле розташування всіх кнопок для зручності користувача. Також в сучасних системах повинна бути присутня зміна мови, що робить веб-застосунок більш гнучким.

І фінальним етапом розробки є верстка сайту відповідно вже узгодженого макету та дизайну. Для цього було використано шаблонізатор з динамічним відображенням контенту який можна рендерити після кожної компіляції програми. Для швидкого написання коду користувацької частини було використано бібліотеку яка надає функціонал з коробки, який можна застосувати просто використавши вже існуючі класи або перевизначивши певні властивості або директиви.

Проєкт побудовано за допомогою структури, яка розбита по рівнях абстракції, тобто в корені будуть знаходитись файли конфігурацій, такі як *pom.xml* який використовує для завантаження сторонніх бібліотек та залежностей системою *Maven*. Також на цьому рівні знаходяться скомпільовані файли програми та деякі допоміжні файли конфігурацій середовища розробки.

Далі розташовується папка *src* яка в свою чергу містить директорію *main*, в якій знаходяться всі класи з вихідним кодом, та директорію *resources* в якій знаходяться конфігураційні властивості бази даних, ресурси для локалізації системи та *html* файли із стилями та скриптами.

2.3. Архітектура програми

У наступних розділах розглянуто шлях побудови архітектури бази даних та макету програми з покроковим описом проєктування. Для кращого розуміння буде додано схеми які відображають все описане графічно.

2.3.1. Архітектура бази даних

Перше про що необхідно подумати при проєктуванні архітектури БД це визначення головних сутностей та встановлення зв'язків між ними. Було обрано три основні сутності це *custom_users, buses, routes*. Всі інші були додані для розширення головних таблиць. Після чого було побудовано основні зв'язки між таблицями, що покращить розуміння залежностей та дасть додаткові можливості при опису реляційної моделі в коді. На фінальній стадії розробки архітектури БД було використано другу форму нормалізації, в результаті чого отримав результат який відображено на рис.2.2.

Після того як база даних архітектурно підготовлена треба перенести її в СУБД, виконавши запити які створять всі таблиці та побудують зв'язки між ними, після чого можна переглянути діаграму таблиць за допомогою спеціальних утиліт. Для полегшеного розуміння системи краще написати всі запити які буде виконувати програма для отримання або вставки даних, тому на цьому кроці необхідно розробити спеціальні *SQL* інструкції. Також для того щоб не створювати нові дані після перезапуску бази необхідно написати міграційні скрипти для вставки даних в таблиці, для прикладу візьмемо вставку в таблицю *Fuel* для того щоб додати два типи пального (бензин та дизельне пальне).

```
INSERT INTO public.fuel (id, fuel_type, price) VALUES (1, 'BENZINE', 31.98);
INSERT INTO public.fuel (id, fuel_type, price) VALUES (2, 'DIESEL', 27.98);
```

Рис. 2.2. Архітектура бази даних системи

2.3.2. Архітектура програмного компоненту

Тепер на прикладі *UML*-діаграми класів яка зображена на рис.2.3, розглянемо архітектуру маленького компонента в системі який відповідає за обробку інформації, зв'язаної із сутністю автобуса. Для свого проєкту було обрано структурний шаблон *MVC*. Він дозволяє побудувати дизайн який легко розширювати та створювати абстракції на різних рівнях.

Так на першому рівні, на рівні моделей, для цього компоненту було вибрано основну сутність *Bus*, яка містить у собі додаткові залежності на *Fuel* (який в свою

чергу містить поле *FuelType*) та залежність на *ComfortClass* який також містить прив’язку до *ComfortClassType*.

Також додана допоміжна таблиця для контролю за основною сутністю, та в випадку будь яких помилок, зв’язаних з цією сутністю, вся інформація про них буде занесена в таблицю, що спрощує процес відловлювання помилок на низькому рівні.

Наступний шар репозиторіїв, який в свою чергу є структурним шаблоном, він має дві цілі: по-перше, це абстракція рівня даних, а по-друге, це спосіб централізації обробки об'єктів домену. На цьому рівні створюються запити з прив’язкою до певної сутності. Іноді для деякого функціоналу не потрібна вся інформація, в такому випадку створюють трансферні об’єкти які місять тільки необхідні поля і з допомогою спеціальних класів з однотипною функціональністю, заносять всю інформацію з бази в такі об’єкти для подальшого використання.

Наступний шар – сервіси , на цьому етапі використовуються дані, отримані в репозиторіях для виконання бізнес логіки та оброблення даних, якими можна змінювати або розширювати вміст бази даних. Можна використовувати безліч репозиторіїв в одному сервісі, але краще розподіляти так щоб в одному класі було як можна менше залежностей для простоти та розуміння коду. Для сервісів було використано структурний шаблон проєктування – міст. Він дозволяє розділити клас на дві окремі ієрархії: абстракцію та реалізацію. Так в моєму проєкті є абстракція *BusService* та його реалізація, де описаний ввесь бізнес функціонал - *DefaultBusService*.

Контролери це шар який відповідає за обробку запитів від фронтального контролера за допомогою механізм диспетчеризації, також він гарантує що для кожного запиту буде свій унікальний контролер для виконання певної логіки. Контролер може приймати різні типи запитів, для прикладу:

- *GET* – запит що використовується тільки для отримання певного ресурсу;
- *POST* – запит що використовується для оновлення даних або вставки нової інформації в деякі таблиці;
- *DELETE* – запит що використовується для видалення існуючих ресурсів.

В залежності від того як пройшов бізнес процес контролер повертає окрім значень ще й статус код, по якому можна визначити що відбулося з ресурсом який передавали в параметри одного з методів обробки запиту.

На діаграмі відображений *BusInfoPageController* який містить у собі залежність тільки на *BusService*, що є прикладом досить непоганої архітектури з слабкою зв'язністю, також контролер не знає про *DefaultBusService* всю відповідальність делегує *BusService*, тому тут використовується ще й принципи інкапсуляції.

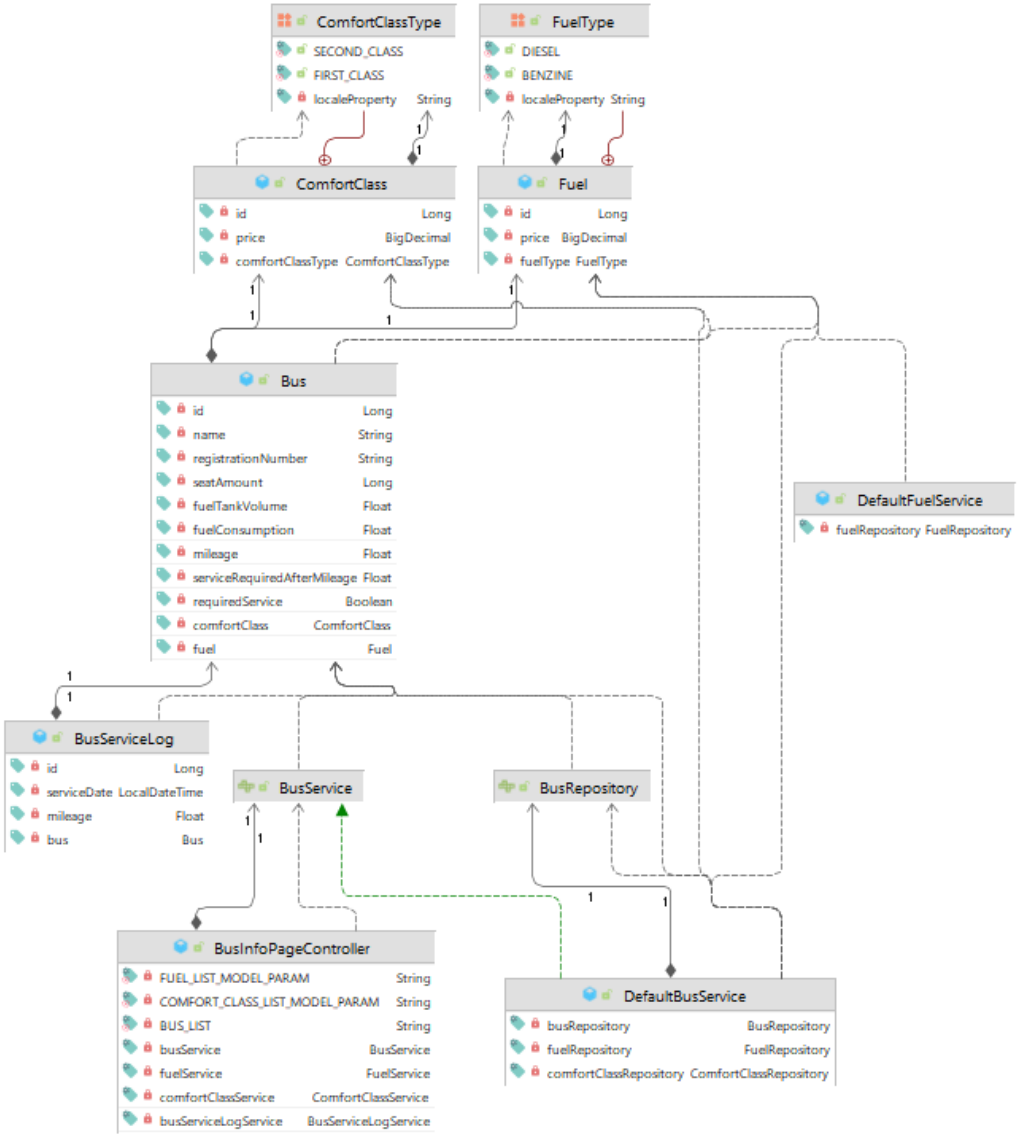

Рис. 2.3. Діаграма класів програмного компоненту

2.4. Висновки до розділу

Даний розділ було посвячено опису функціональних вимог та архітектури програмного додатку. Була описана вся функціональність яку виконує даний веб додаток. Також була описана структура бек-енд частини та архітектура системи починаючи від шару контролерів та закінчуючи моделями для відображення таблиць в базі даних. Частково були описані шаблони проєктування, які застосовувалися при розробці, та пояснення чому саме ці шаблони було обрано для написання певного класу. Описано процес проєктування архітектури бази даних від створення основних сутностей до її нормалізації.

В ході вибору архітектури серверної частини було з’ясовано що багатошарова архітектура підібрана досить вдало для написання проєкту такого розміру, адже в майбутньому додавати нових функціонал буде зрозуміло навіть для програміста, незнайомого з системою. Головною перевагою такої архітектури є те, що при розширенні кожного шару це не впливає на інші рівні, а також дає можливість паралельної розробки для кількох програмістів.

Було побудовано дві діаграми. Перша відображає архітектуру бази даних на її фінальному етапі, друга – це діаграма одного з програмних компонентів для відображення структури класів серверної архітектури.

РОЗДІЛ 3

ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСТОСУНКУ

3.1. Створення макету програми

Для початку необхідно створити *Spring Boot Application*, використовуючи ініціалізатор який створений компанію розробників *Spring* для швидкого старту. Вибрати систему зборки проєкту, в нашому випадку це Maven, який містить файл конфігурації *pom.xml,* де потрібно визначити всі залежності, які необхідні для програми. Початкова точка проєкту виглядає так:

```
@SpringBootApplication
public class TransportManagementSystem {

   public static void main(String[] args) {
      SpringApplication.run(TransportManagementSystem.class, args);
   }
}
```

Також переглянемо один із прикладів *maven* залежностей, які завантажуються при побудові артефакту та зборки проєкту:

```
<dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
 <dependencies>
```

Кожна залежність містить в собі унікальні ідентифікатори групи - *groupId* та спеціального артефакту – *artifactId*, за допомогою яких *maven* визначає звідки завантажити певну залежність.

Одним із важливих кроків є підключення файлів конфігурацій для локалізації певних даних, які будуть відображатися користувачеві. В дипломному проєкті буде використано дві мови: англійська та українська, тому потрібно створити два файли, в яких будуть визначені певні змінні, у форматі властивість=значення :

header.navigation.find-trips=Find trips

Також для макету необхідно підключити тестову html сторінку, та спробувати запустити додаток і переглянути цю сторінку:

```
<!doctype html>
<html>
<head>
        <title>This is the title of the webpage!</title>
</head>
<body>
<p>First Page.</p>
</body>
</html>
```

3.2. Підключення бази даних

На цьому кроці потрібно встановити з’єднання з базою даних. Для цього необхідно прописати властивості в файлі конфігурації, вказавши користувача, пароль та *URL* бази даних:

```
spring.datasource.username=postgres
spring.datasource.password=*******
spring.datasource.url=jdbc:postgresql://localhost:5432/spring-data
```

Після цього створимо першу модель та пов’яжемо її з таблицею в базі даних. Для цього необхідно написати клас, вказавши для нього анотацію *@Table* зазначивши в параметрі name її ім’я. Також необхідно створити id таблиці,

вказавши для нього анотації *@Id* та *@GeneratedValue* для постійного інкременту, що дасть унікальність запису, після цього можемо додати всі інші поля які необхідні, в результаті чого отримаємо клас:

```
@Table(name = "FUEL")
public class Fuel {
    @Id
    @GeneratedValue
    private Long id;
    private BigDecimal price;
    @Enumerated(EnumType.STRING)
    private FuelType fuelType;
}
```

Для перевірки працездатності напишемо інтерфейс який наслідує *JpaRepository* та додамо для нього анотацію *@Repository*, після цього створимо метод для отримання всіх записів в таблиці:

```
@Repository
public interface FuelRepository extends JpaRepository<Fuel, Long> {
    @Query(value = "SELECT * from FUEL", nativeQuery = true)
    List<Fuel> getAll();
}
```

Щоб перевірити що все коректно працює потрібно запустити програму та спробувати викликати метод *FuelRepository.getAll*(), або написати тестовий сценарій.

3.3. Написання бізнес логіки

Після того як ми маємо готовий каркас додатку, можна приступити до написання основного функціоналу, для прикладу оберемо процес авторизації користувача. Першим ділом створимо клас та додамо для нього анотацію *@Service*, щоб явно вказати те що це компонент *Spring*, в якому буде написаний функціонал проєкту. Він має такий вигляд:

```
@Service
@RequiredArgsConstructor
public class DefaultCustomUserDetailsService implements
CustomUserDetailsService, UserDetailsService {
    private final CustomUserRepository customUserRepository;
    private final BCryptPasswordEncoder passwordEncoder;

    @Override
    public UserDetails loadUserByUsername(String email) {
        CustomUser customUser = customUserRepository.findByEmail(email)
                .orElseThrow(() -> new UsernameNotFoundException
(String.format("User with email %s not found", email)));
        checkPasswordOrElseThrow();
        Set<GrantedAuthority> grantedAuthorities = new HashSet<>();
        grantedAuthorities.add(new SimpleGrantedAuthority
       (customUser.getRole().name()));

        return neworg.springframework.security.core.userdetails.User
(customUser.getEmail(), customUser.getPassword(), grantedAuthorities);
    }}
```

В цьому методі іде покроковий опис наступної логіки:

1. Пошук користувача за поштою. Якщо користувач не знайдений, то буде створено виключення, яке потім обробляється спеціальним контролером;
2. Перевірка паролю користувача за вказаною існуючою поштою;
3. Присвоєння користувачеві ролі, яка знаходиться в базі даних.

Після цих кроків знайдений користувач з визначеною роллю обробляється контролером, який переадресовує на наступну сторінку – сторінку профілю.

3.4. Відображення бізнес логіки на веб-сторінці

Тепер переглянемо процес з’єднання серверної та користувацької частини. Для прикладу знову оберемо процес авторизації та аутентифікації (рис.3.1). Створимо *html* форму з полями username та password та кнопкою :

```
</div>
  <input name="username" id="username" type="email" class="form-control"
                    th:placeholder="#{page.login.form.input.email}">
</div>
</div>
   <input name="pass" id="password" type="password" class="form-control"
                    th:placeholder="#{page.login.form.input.password}">
 </div>
```

У випадку коли користувач не знайдений або невірно введено пароль, буде додана секція яка відображає про цю помилку

```
<div class="alert alert-danger" style="margin-top: 10px;" th:if="${errorMessage}">
    <div class="container">
        <div class="alert-icon">
            <i class="material-icons">error_outline</i>
        </div>
    <b th:text="#{page.wrong-login-or-password}">Wrong login or password</b>
    </div>
</div>
```

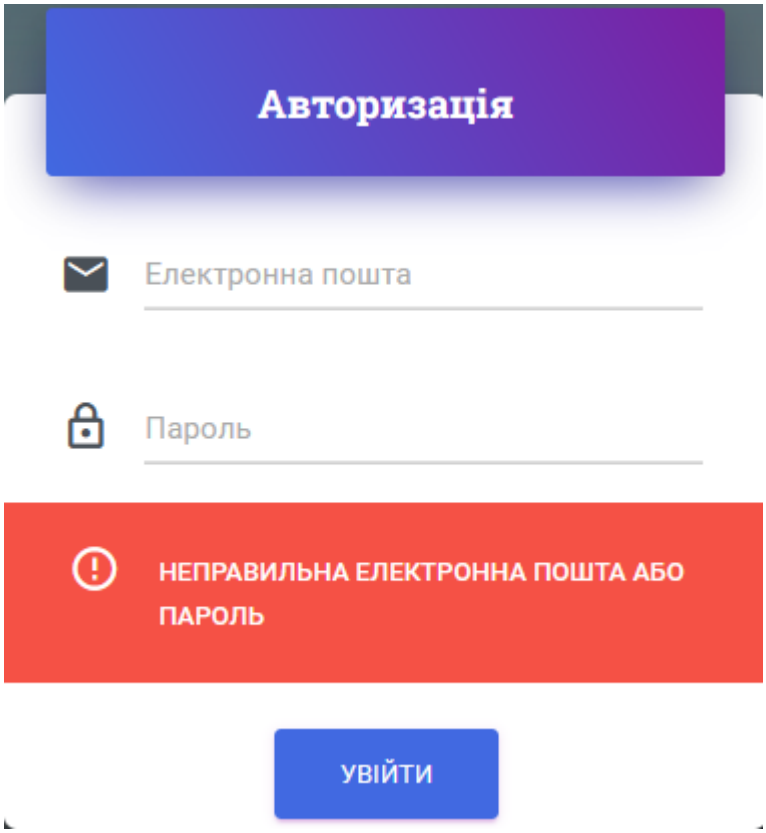

Рис.3.1.Помилка на сторінці авторизації

При натисканні кнопки “Увійти” відправляється *POST* запит, який буде опрацьовано контролером, який спочатку перевірить результат на наявність помилок, після цього використовуючи методи сервісу знайде користувача та додасть його в атрибути моделі. При успішному виконанні всіх кроків користувач зайде на свою сторінку профілю та зможе використовувати ввесь функціонал, який він має.

```
@PostMapping("/login")
   public String authenticateUser(@Valid @ModelAttribute("userDTO")
CustomUserDTO currentUser, BindingResult bindingResult, Model model) {
      if (bindingResult.hasErrors()) {
         return ControllerConstants.Pages.DRIVER_INFO_PERFORMING;
      }
      Optional<CustomUser> user = customUserDetailsService
 .findByEmail(currentUser.getEmail());
```

```
    model.addAttribute(user.get().toString(), "currentUser");
    return "main";
}
```

3.5. Перегляд функціоналу зі сторони користувача

Головною ідеєю цього веб-застосунку було створення механізму швидкого пошуку поїздок за певним маршрутом для всіх користувачів, які мають доступ до інтернету. Тому сервіс розроблювальної системи представляє можливість, функціонал якої зображено на рис.3.2.

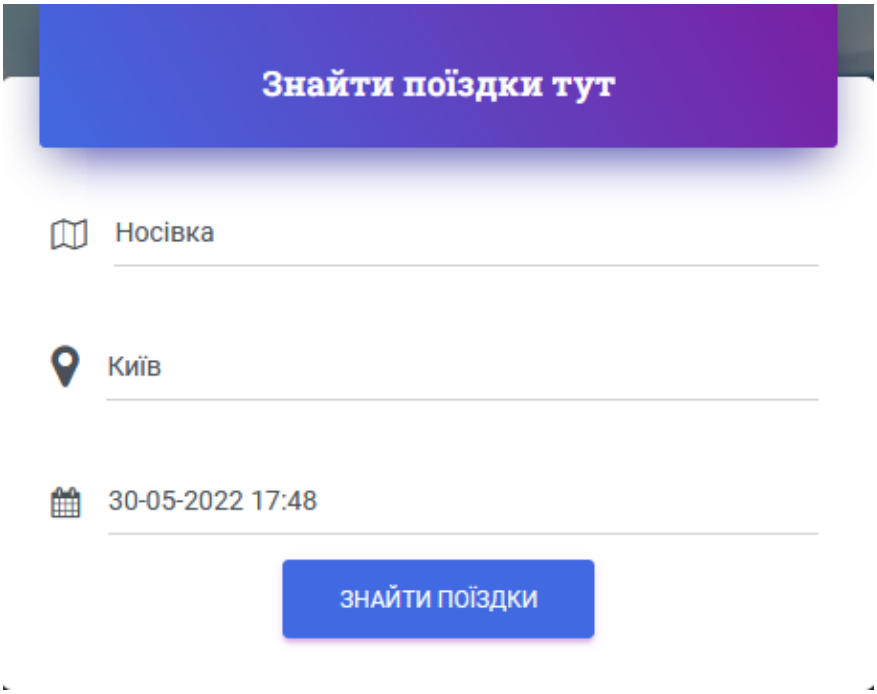

Рис.3.2.Пошук поїздок

Для того щоб знайти необхідну поїздку потрібно ввести місце відправлення та місце призначення і звичайно дату, на яку запланована поїздка, якщо така поїздка знайдена, то отримаємо список поїздок, для прикладу, зображені на рис.3.3.

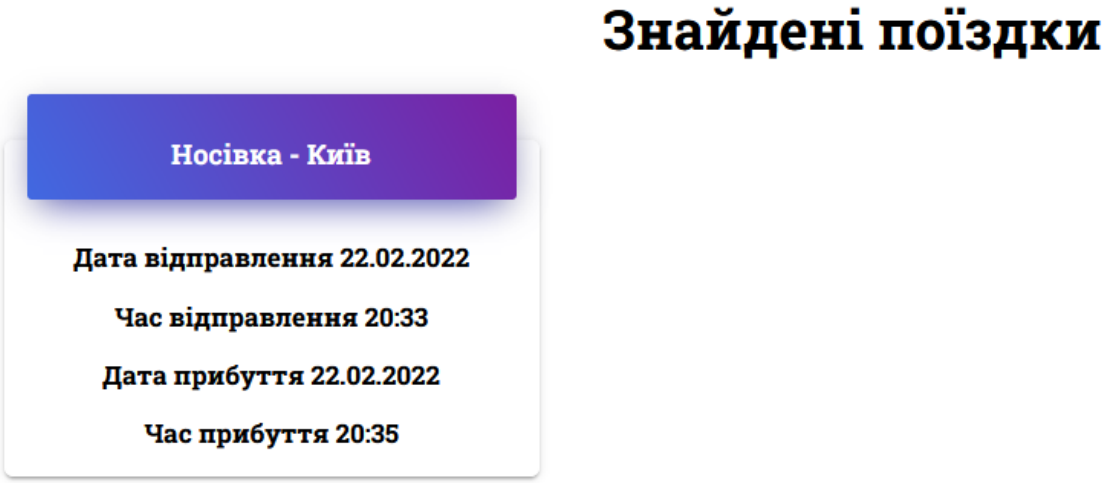

Рис.3.3.Результат пошуку поїздок

Також розглянемо функціональність яка доступна користувачам з роллю “Диспетчер”. Переглянемо деякі з функції, зображених на рис.3.4, окремо.

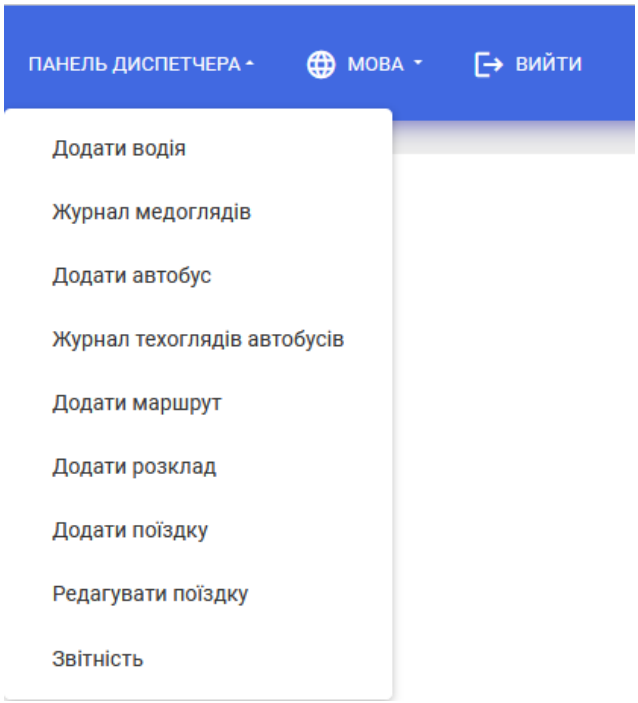

Рис.3.4.Панель диспетчера

Функція додавання нового маршруту, зображену на рис.3.5, містить карту, для кращого розуміння та наглядності. Щоб додати новий маршрут необхідно задати початкову та кінцеву точку, номер маршруту та його назву, після чого

натиснути кнопку “Побудувати маршрут”. В результаті отримаємо карту з побудованим на ній маршрутом та відстанню, яка також обраховується програмно.

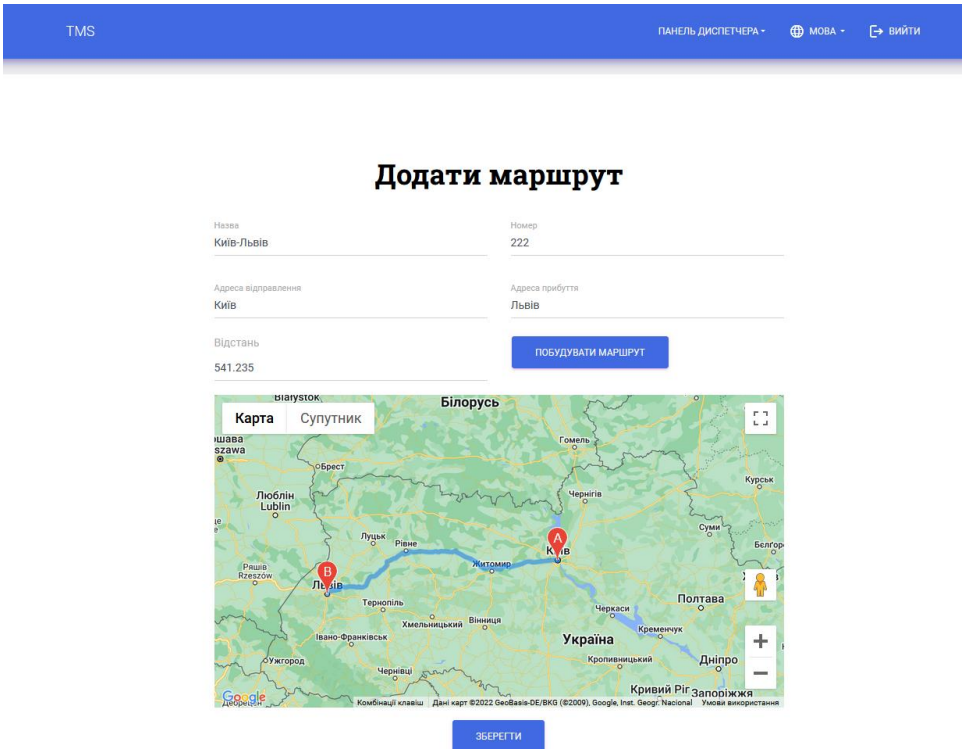

Рис.3.5.Функціонал додавання нового маршруту

Наступною розглянемо функцію перегляду звітності (рис.3.6).

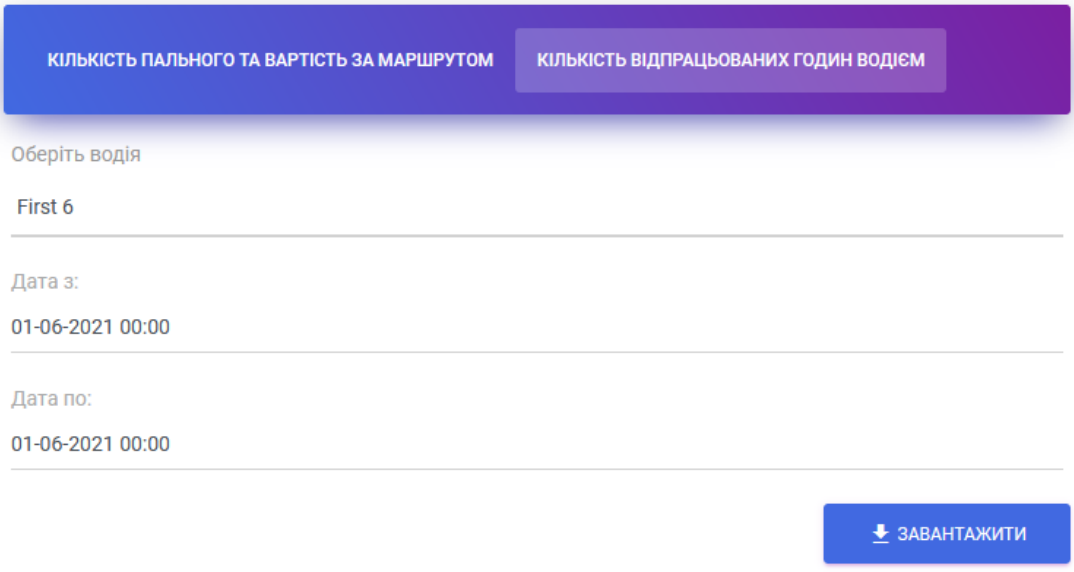

Рис.3.6.Функціонал перегляду звітів

Досить важливою є функція створення звітів за певними показниками. У моєму додатку є два типи звітів:

– за кількістю палива та вартість за маршрутом;
– за кількістю відпрацьованих годин водієм.

Для того щоб отримати звіт по кількості відпрацьованих водієм годин необхідно обрати водія для якого буде згенерований звіт та дату за який період необхідна інформація. Програма знаходить всі поїздки, які виконав даний водій за обраний період, потім рахує кількість годин по кожній поїздці та додає в результуючу суму. Такий звіт допоможе диспетчерам та власникам бізнесу для щомісячного моніторингу руху за певним маршрутом, і також дасть можливість для легшого обрахунку заробітної плати для водія. Приклад звіту, який можна завантажити зображено на рис.3.7.

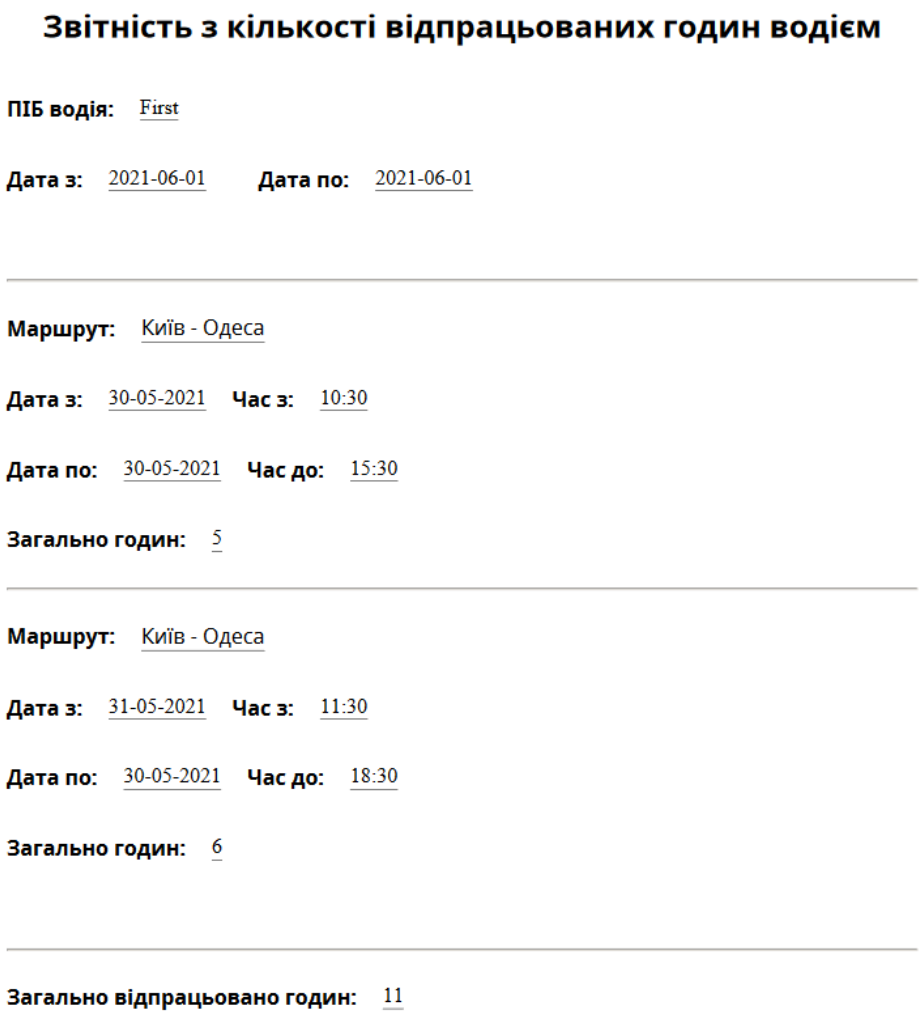

Рис.3.7. Результат генерованого звіту

3.6. Висновки до розділу

В розділі проведено розробку та зроблено опис програмного додатку, на прикладі одного компоненту. Було описано процес побудови макету програми, з’єднання його з базою даних. Також представлено бізнес логіку для авторизації та графічне відображення сторінки авторизації.

Для повного уявлення про роботу веб-додатку, наведено графічні приклади пошуку поїздок за певним маршрутом і також деякої функціональності яка доступна для диспетчера. Також представлено деякі тест кейси та проведено тестування всієї системи, після чого виявлено і виправлено декілька багів.

ВИСНОВКИ

Метою виконання дипломного проекту було створення програмного додатку моніторингу руху громадського транспорту та можливість швидкого пошуку транспорту за певним маршрутом. Під час процесу створення програмного додатку були вирішені наступні задачі:

1. проведено аналіз існуючих на українському на європейському ринках систем, які займаються пошуком поїздок за обраним маршрутом;
2. розглянуто сучасні технології, які застосовуються при розробці клієнт серверних додатків з врахуванням кросплатформеності;
3. покроково описано процес розробки програмного додатку;
4. створено програмний додаток для моніторингу руху;
5. протестована його працездатність та коректність роботи.

У дипломному проєкті було ознайомлено із принципами розробки веб-застосунків з використанням баз даних на основі кросплатформеної мови програмування *Java*, для безпосередньої прив'язки до якоїсь однієї платформи. Також всі вивчені теоретичні знання були застосовані на практиці під час проєктування архітектури та написання коду.

Розроблений проєкт має наступні функції:

– Авторизація зареєстрованих користувачів;
– Генерування *PDF*-звітів за обраним критерієм;
– Графічна побудова маршруту за двома координатами;
– Сторінка для пошуку поїздок на обрану дату.

У першому розділі було розглянуто та проаналізовано технології та інструменти, які досить часто використовують при розробці веб додатків та дозволяють підтримувати архітектуру з розширенням функціональності.

Також в цьому розділі були розглянуті особливості роботи з фреймворком *Spring* мовою *Java*, а саме робота з рівнем зв'язаним з базою даних та з модулем який відповідає за безпеку системи.

Другий розділ було присвячено опису функціональних вимог та архітектури програмного додатку. Була описана вся функціональність яку виконує даний

веб додаток. Також була описана структура бек-енд частини та архітектура системи починаючи від шару контролерів та закінчуючи моделями для відображення таблиць в базі даних. Частково були описані шаблони проєктування, які застосовувалися при розробці, та пояснення чому саме ці шаблони було обрано для написання певного класу. Описано процес проєктування архітектури бази даних від створення основних сутностей до її нормалізації.

В ході вибору архітектури серверної частини було з'ясовано що багатошарова архітектура підібрана досить вдало для написання проєкту такого розміру, адже в майбутньому додавати нових функціонал буде зрозуміло навіть для програміста, незнайомого з системою. Головною перевагою такої архітектури є те, що при розширенні кожного шару це не впливає на інші рівні, а також дає можливість паралельної розробки для кількох програмістів.

У третьому розділі було проведено розробку та зроблено опис програмного додатку, на прикладі одного компоненту. Було описано процес побудови макету програми, з'єднання його з базою даних. Також представлено бізнес логіку для авторизації та графічне відображення сторінки авторизації.

Для повного уявлення про роботу веб-додатку, наведено графічні приклади пошуку поїздок за певним маршрутом і також деякої функціональності яка доступна для диспетчера. Також представлено деякі тест кейси та проведено тестування всієї системи, після чого виявлено і виправлено декілька багів.

Аналізуючи результат виконання проєктної роботи, можна зробити висновки, що розроблений додаток має сучасний інтерфейс, який є масштабованим та виконаний згідно спроєктованого дизайну.

При розробці були виконані всі задачі, поставлені під час етапу аналізу та проєктування. В порівнянні з конкурентами розроблений додаток має право на життя та поступову експлуатацію, але потребує подальшої розробки та доповнення функціоналом.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Quick Documentation [Електронний ресурс]. – 2022. – Режим доступу: https://www.jetbrains.com/idea/ (дата звернення 11.05.2022).
2. Розвиток українських міст [Електронний ресурс]. – 2022. – Режим доступу: https://mistosite.org.ua/articles/rozvytok-ukrainskykh-mist-novi-merezhi-novi-mozhlyvosti.
3. Застосування веб сайтів [Електронний ресурс]. – 2022. – Режим доступу: http://www.webtec.com.ua/uk/articles/index/view/2011-05-05/web-site
4. Busfor UA Official site [Електронний ресурс]. – 2022. – Режим доступу: https://busfor.ua.
5. GO Official site [Електронний ресурс]. – 2022. – Режим доступу: https://12go.co/en.
6. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи(проєкти) випускників Національного авіаційного університету. – К.: НАУ,2017. – 63 с.
7. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення / Держстандарт України. – Вид. офіц. – [Чинний від 1995-02-23]. – Київ, 2007. – 86 с.
8. Роберт Мартін. Чиста архітектура: Мистецтво розроблення програмного забезпечення / пер. з англ. І. Бондар-Терещенко. – Харків : Вид-во “Ранок” : Фабула, 2020 – 368с.
9. Роберт Мартін. Чистий код: створення і рефакторинг за допомогою Agile пер. з англ. І. Бондар-Терещенко. – Харків : Вид-во “Ранок” : Фабула, 2020 – 448с.
10. ДСТУ ГОСТ 3008–95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.»
11. Джошуа Блох. Java: ефективне програмування, 3-е видання. – Харків : Вид-во “Ранок” : Фабула, 2020 – 464с.
12. Refactoring GURU [Електронний ресурс]. – 2022. – Режим доступу: https://refactoring.guru/uk (дата звернення 19.05.2022).

13. Spring Framework Documentation [Електронний ресурс]. – 2022. – Режим доступу: https://docs.spring.io/spring-framework/docs/current/reference/html/ (дата звернення 15.05.2022)
14. Spring Initializr [Електронний ресурс]. – 2022. – Режим доступу: https://start.spring.io/ (дата звернення 11.05.2022)
15. Bootstrap documentation [Електронний ресурс]. – 2022. – Режим доступу: https://getbootstrap.com/docs/5.0/about/overview/ (дата звернення 15.05.2022)
16. Google APIs for development [Електронний ресурс]. – 2022. – Режим доступу: https://developers.google.com/maps/ (дата звернення 16.05.2022)
17. Apache Maven Project [Електронний ресурс]. – 2022. – Режим доступу: https://maven.apache.org/plugins/maven-dependency-plugin/usage.html (дата звернення 11.05.2022)
18. Офіційний сайт PostgreeSQL [Електронний ресурс]. – 2022. – Режим доступу: https://www.postgresql.org (дата звернення 11.05.2022)
19. Microservices architecture [Електронний ресурс]. – 2022. – Режим доступу: https://microservices.io/patterns/microservices.html (дата звернення 11.05.2022)
20. JavaScript modern desicion [Електронний ресурс]. – 2022. – Режим доступу: https://uk.javascript.info/ (дата звернення 11.05.2022)

Додаток А

Лістинг файлу *MainPageController.java*

```
@Controller
@RequiredArgsConstructor
@RequestMapping(ControllerConstants.Mapping.BUS_PARK_EMPLOYEES_WORK
_CONTROL)
public class MainPageController {

   private static final DateTimeFormatter DATE_TIME_FORMATTER =
DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm");

   private final RouteCheckDocumentService routeCheckDocumentService;
   private final ScheduleService scheduleService;
   private final RouteService routeService;

   @GetMapping({ControllerConstants.Mapping.MAIN})
   public String showMainPage(@ModelAttribute FindTripForm findTripForm) {
      return ControllerConstants.Pages.MAIN;
   }

   @PostMapping("/findtrip")
   public String showFoundTripFragment(@Valid @ModelAttribute FindTripForm
findTripForm, BindingResult bindingResult,
                                       RedirectAttributes redirectAttributes) {

      if (bindingResult.hasErrors()) {
         return ControllerConstants.Pages.MAIN;
      }

      if (!DateTimeUtils.isDateTimeAfterNow(LocalDateTime.parse(
```

```
findTripForm.getDepartureDateTime(), DATE_TIME_FORMATTER))) {
            bindingResult.rejectValue(
"departureDateTime", "page.main.find-trip.error.past-date");
        }
        Optional<Route> routeOptional =
routeService.findByName(findTripForm.getDeparturePoint(),
findTripForm.getArrivalPoint());
        if (!routeOptional.isPresent()) {
            bindingResult.rejectValue(
"tripsNotFound", "page.main.find-trip.error.nothing-found");
            return ControllerConstants.Pages.MAIN;
        }
        List<Schedule> schedules = scheduleService.findAll();
        if (schedules.isEmpty()) {
            bindingResult.rejectValue("tripsNotFound", "page.main.find-
trip.error.nothing-found");
        }
        List<RouteCheckDocument> routeCheckDocuments =
routeCheckDocumentService.findAllByScheduleIn(schedules);
        if (routeCheckDocuments.isEmpty()) {
            bindingResult.rejectValue("tripsNotFound", "page.main.find-
trip.error.nothing-found");
        }
        if (bindingResult.hasErrors()) {
            return ControllerConstants.Pages.MAIN;
        }

        redirectAttributes.addFlashAttribute("foundTripList", routeCheckDocuments);

        return ControllerConstants.REDIRECT +
ControllerConstants.Mapping.BUS_PARK_EMPLOYEES_WORK_CONTROL +
```

```
            ControllerConstants.Mapping.MAIN;
    }
        }
```

Лістинг файлу *DefaultRouteCheckDocumentService.java*

```
@Service
@RequiredArgsConstructor
public class DefaultRouteCheckDocumentService implements
RouteCheckDocumentService {

    private final RouteCheckDocumentRepository routeCheckDocumentRepository;
    private final CustomUserDetailsService customUserDetailsService;
    private final BusService busService;
    private final ScheduleService scheduleService;

    @Override
    public RouteCheckDocument save(RouteCheckDocumentDTO
routeCheckDocumentDTO) {

        CustomUser driver =
customUserDetailsService.findById(routeCheckDocumentDTO.getDriverId())
                .orElseThrow(() -> new EntityNotFoundException(
String.format("%s entity with id %s was not found",
                        CustomUser.class, routeCheckDocumentDTO.getDriverId())));

        Bus bus = busService.findById(routeCheckDocumentDTO.getBusId())
                .orElseThrow(() -> new EntityNotFoundException(
String.format("%s entity with id %s was not found",
                        Bus.class, routeCheckDocumentDTO.getBusId())));
```

```
        Schedule schedule =
scheduleService.findById(routeCheckDocumentDTO.getScheduleId())
                .orElseThrow(() -> new EntityNotFoundException(
String.format("%s entity with id %s was not found",
                        Schedule.class, routeCheckDocumentDTO.getScheduleId())));

        return routeCheckDocumentRepository.save(RouteCheckDocument.builder()
                .driver(driver)
                .bus(bus)
                .filledFuel(0F)
                .soldSeat(0L)
    .tripStatus(RouteCheckDocument.TripStatus.DRIVER_ACCEPTANCE_WAITING)
                .schedule(schedule)
                .build());
    }

    @Override
    public Optional<RouteCheckDocument> findById(Long id) {
        return routeCheckDocumentRepository.findById(id);
    }


    @Override
    public List<RouteCheckDocument> findAll() {
        return routeCheckDocumentRepository.findAll();
    }

        @Override
        public RouteCheckDocument update(RouteCheckDocumentUpdateDTO
      routeCheckDocumentUpdateDTO) {
```

```
    RouteCheckDocument routeCheckDocument =
routeCheckDocumentRepository.findById(routeCheckDocumentUpdateDTO.getTripId
Update()).
            orElseThrow(() -> new EntityNotFoundException(
String.format("%s entity with id %s was not found",
                    RouteCheckDocument.class,
routeCheckDocumentUpdateDTO.getTripIdUpdate())));

    CustomUser driver =
customUserDetailsService.findById(routeCheckDocumentUpdateDTO.getDriverIdUpd
ate())
            .orElseThrow(() -> new EntityNotFoundException(
String.format("%s entity with id %s was not found",
                    CustomUser.class,
routeCheckDocumentUpdateDTO.getDriverIdUpdate())));

    Bus bus =
busService.findById(routeCheckDocumentUpdateDTO.getBusIdUpdate())
            .orElseThrow(() -> new EntityNotFoundException(
String.format("%s entity with id %s was not found",
                    Bus.class, routeCheckDocumentUpdateDTO.getBusIdUpdate())));

    routeCheckDocument.setDriver(driver);
    routeCheckDocument.setBus(bus);
    routeCheckDocument.setTripStatus(RouteCheckDocument.TripStatus.forName(
routeCheckDocumentUpdateDTO.getTripStatusUpdate()));
    routeCheckDocument.setFilledFuel(
routeCheckDocumentUpdateDTO.getFilledFuelUpdate());
    return routeCheckDocumentRepository.save(routeCheckDocument);
  }
```

```
    @Override
    public List<RouteCheckDocument> findAllByDriverAndTripStatus(CustomUser
driver, RouteCheckDocument.TripStatus tripStatus) {
        return routeCheckDocumentRepository.findAllByDriverAndAndTripStatus(driver,
tripStatus);
    }
    @Override
    public List<RouteCheckDocument> findAllByDateAndRoute(LocalDateTime
departureDateTime, Route route) {
        return null;
    }
    @Override
    public List<RouteCheckDocument> findAllByScheduleIn(List<Schedule>
schedules) {
        List<RouteCheckDocument> routeCheckDocuments =
routeCheckDocumentRepository.findAllByScheduleIn(schedules);

        return routeCheckDocuments.stream()
                .filter(routeCheckDocument -> routeCheckDocument.getSoldSeat() <
routeCheckDocument.getBus().getSeatAmount())
                .collect(Collectors.toList());
    }
        }
```

Лістинг файлу *RouteCheckDocumentService.java*

```
public interface RouteCheckDocumentService extends
Service<RouteCheckDocumentDTO, RouteCheckDocument>{
    RouteCheckDocument update(RouteCheckDocumentUpdateDTO
routeCheckDocumentUpdateDTO);
```

```
    List<RouteCheckDocument> findAllByDriverAndTripStatus(
CustomUser driver, RouteCheckDocument.TripStatus tripStatus);
    List<RouteCheckDocument> findAllByDateAndRoute(
LocalDateTime departureDateTime, Route route);
    List<RouteCheckDocument> findAllByScheduleIn(List<Schedule> schedules);
}
```

Лістинг файлу *CustomUser.java*

```
@Data
@Table(name = "CUSTOM_USERS")
public class CustomUser {

    @Id
    @GeneratedValue
    private Long id;

    private String email;

    private String password;

    private String fullName;

    @Enumerated(EnumType.STRING)
    private CustomUserRole role;

    public enum CustomUserRole {
        DISPATCHER, DRIVER
    }
}
```

Лістинг файлу *CustomUserDTO.java*

```
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class CustomUserDTO {

    @NotEmpty(message = "{form.error.empty}")
    @Email(message = "{form.error.email.wrong-format}")
    private String email;

    @NotEmpty(message = "{form.error.empty}")
    @Size(min = 5, max = 30, message = "{form.error.password.wrong-size}")
    private String password;

    @NotEmpty(message = "{form.error.empty}")
    private String fullName;

    private CustomUser.CustomUserRole role;
}
```

Лістинг файлу *Trip-info-performing.html*

```
<body>
<div th:insert="reusable/header.html::header"></div>
<div class="main ">
    <div class="container">
        <div class="section section-contacts">
            <div class="row">
                <div class="col-md-8 ml-auto mr-auto">
                    <div class="col-md-12 ml-auto mr-auto">
                        <div style="margin: 0; padding: 0;" class="tab-content tab-space">
```

```
<div class="tab-pane active" id="dashboard-1">
  <div class="row">
    <div class="col-md-12 ml-auto mr-auto">
      <h2 class="text-center title" th:text="#{page.add-
trip.title}">Add new trip</h2>
      <form
action="/busparkemployeesworkcontrol/tripinfopreforming/save"
        method="post"
        th:object="${routeCheckDocumentDTO}">

      <div class="row">
        <div class="col-md-6">
          <div class="form-group">
            <label class="bmd-label-floating"
              th:text="#{page.add-trip.form.choose-
driver}">Choose driver</label>
        <select th:field="*{driverId}" class="form-control selectpicker"
            data-style="btn btn-link" id="driverSelect">
          <option th:each="driver : ${driverList}"
            th:value="${driver.id}"
            th:text="${driver.fullName}">
          </option>
        </select>
        <div class="alert alert-warning"
          style="margin-top: 5px; padding: 10px"
         th:each="error : ${#fields.errors('driverId')}">
        <div class="container">
          <div class="alert-icon">
            <i class="material-icons">error_outline</i>
          </div>
          <b th:text="${error}">Select bus</b>
        </div>
        </div>
      </div>
    </div>
    <div class="col-md-6">
      <div class="form-group">
        <label class="bmd-label-floating"
          th:text="#{page.add-trip.form.choose-
bus}">Choose bus</label>
        <select th:field="*{busId}"
          style="font-family: 'FontAwesome', 'Roboto';"
          class="form-control selectpicker"
          data-style="btn btn-link"
          id="busSelect">
```

```
                                        <option th:each="bus : ${busList}"
th:value="${bus.id}" th:text="${bus.requiredService} ?
                                    ${bus.name} + ' ' + ${bus.registrationNumber} +
'&#xf0ad;' :
                                    ${bus.name} + ' ' + ${bus.registrationNumber}">
                                            <i>&#xf0ad;</i>
                                        </option>
                                    </select>
                                    <div class="alert alert-warning"
                                        style="margin-top: 5px; padding: 10px"
                                        th:each="error : ${#fields.errors('busId')}">
                                        <div class="container">
                                            <div class="alert-icon">
                                                <i class="material-
icons">error_outline</i>
                                            </div>
                                            <b th:text="${error}">Select bus</b>
                                        </div>
                                    </div>
                                </div>
                            </div>

                            <div class="row">
                                <div class="col-md-12">
                                    <div class="form-group">
                                        <label class="bmd-label-floating"
                                            th:text="#{page.add-trip.form.choose-
schedule}">Choose schedule</label>
                                        <select th:field="*{scheduleId}"
                                            class="form-control selectpicker"
                                            data-style="btn btn-link"
                                            id="scheduleSelect">
                                        <option th:each="schedule : ${scheduleList}"
                                            th:value="${schedule.id}"
                                            th:text="${schedule.route.name} + ' ' +
                                    ${#temporals.format(schedule.departureDateTime, 'dd-
MM-yyyy HH:mm')} +' ' +${#temporals.format(
schedule.arrivalDateTime, 'dd-MM-yyyy HH:mm')}">
                                        </option>
                                    </select>
                                    <div class="alert alert-warning"
                                        style="margin-top: 5px; padding: 10px"
                                        th:each="error :
${#fields.errors('scheduleId')}">
                                        <div class="container">
```

```
                                        <div class="alert-icon">
                                            <i class="material-
icons">error_outline</i>
                                        </div>
                                        <b th:text="${error}">Select bus</b>
                                    </div>
                                </div>
                            </div>
                        </div>

                        <div class="row">
                            <div class="col-md-12 ml-auto mr-auto">
                                <div id="map-container-google-4" class="map-
container-4">
                                    <div id="googleMap"
style="width:100%;height:400px"></div>
                                    <script>
                                        function initMap() {
                                            const directionsService = new
google.maps.DirectionsService();
                                            const directionsRenderer = new
google.maps.DirectionsRenderer();
                                            const map = new
google.maps.Map(document.getElementById("googleMap"), {
                                                zoom: 5,
                                                center: {lat: 48.383022, lng: 31.1828699},
                                            });
                                            directionsRenderer.setMap(map);
                                            findRoute(directionsService,
directionsRenderer);

                                            const onChangeHandler = function () {
                                                findRoute(directionsService,
directionsRenderer);
                                            };

document.getElementById("scheduleSelect").addEventListener("change",
onChangeHandler);
                                        }

                                        function
calculateAndDisplayRoute(directionsService, directionsRenderer, data) {
                                            directionsService.route(
                                                {
                                                    origin: {
```

```
                                        query: data.departureAddress,
                                    },
                                    destination: {
                                        query: data.arrivalAddress,
                                    },
                                    travelMode:
google.maps.TravelMode.DRIVING,
                                },
                                (response, status) => {
                                    if (status === "OK") {

directionsRenderer.setDirections(response);
                                    } else {
                                        window.alert("Directions request failed
due to " + status);
                                    }
                                }
                            );
                        }

                function findRoute(directionsService, directionsRenderer) {
                                var scheduleId =
document.getElementById("scheduleSelect").value;

$.get("http://localhost:8080/busparkemployeesworkcontrol/routeinfoperforming/" +
scheduleId, function (data) {

calculateAndDisplayRoute(directionsService, directionsRenderer, data)
                                });
                            }
                        </script>
                        <script
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBg9Fd5RgUNrJ6f-
0ICV1L476BJiQCBW4w&callback=initMap&libraries=&v=weekly"
                                async></script>
                    </div>
                </div>
            </div>

            <div class="row">
                <div class="col-md-4 ml-auto mr-auto text-center">
                    <button class="btn btn-primary btn-raised"
                        th:text="#{page.add-driver.form.button.save}">
                        Save
                    </button>
                </div>
```

```
                        </div>

                        <div class="row" th:if="${infoSaved}">
                            <div class="col-md-6 ml-auto mr-auto text-center">
                                <div class="alert alert-success"
                                    style="margin-top: 5px; padding: 10px">
                                    <div class="container">
                                        <div class="alert-icon">
                                            <i class="material-icons">check</i>
                                        </div>
                                        <b th:text="#{page.add-driver.form.label.info-
saved}">Driver info was saved</b>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
</body>

<div th:insert="reusable/footer.html::footer"></div>
</html>
```