МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.

«___» _____2021 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: «Веб-додаток CRUD з використанням технологій Spring і Hibernate»

Виконавець: Новохатко Олексій Миколайович

Керівник: Коба Олена Вікторівна

Нормоконтролер: Тупота Євгеній Вікторович

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Освітнього ступеня бакалавр

Напрям (спеціальність) 123 "Комп'ютерна інженерія"

(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Литвиненко О.Є.
«_____»_____ 2021 р.

ЗАВДАННЯ
на виконання дипломної роботи (проєкту)

Новохатка Олексія Миколайовича

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

1. Тема дипломної роботи (проєкту):« Веб-додаток *CRUD* з використанням технологій *Spring* і *Hibernate*»
затверджена наказом ректора від «04» лютого 2021 р. № 135/ст.

2. Термін виконання роботи (проєкту): з 24.05.2021 по 20.06.2021

3. Вихідні дані до роботи (проєкту):системи взаємодії з клієнтом, база даних *MySql*, мова програмування *Java*

4. Зміст пояснювальної записки:
1) Огляд існуючих аналогів автоматизації взаємодії з клієнтом через користувацький інтерфейс.
2) Проєктування веб-додатку для автоматизації взаємодії з клієнтом.
3) Розробка веб-додатку для автоматизації взаємодії з клієнтом.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:
1) Схема руху запиту користувача через додаток
2) Схеми класів.
3) Зображення користувацького інтерфейсу.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Провести аналіз предметної області та програмних продуктів за темою дипломного проекту	24.05.2021 – 26.05.2021	
2	Підготувати перший розділ пояснювальної записки	27.05.2021 – 29.05.2021	
3	Спроэктувати додаток	29.05.2021 – 31.05.2021	
4	Підготувати другий розділ пояснювальної записки	01.06.2021 – 03.06.2021	
5	Розробити додаток	04.06.2021 – 06.06.2021	
6	Підготувати третій розділ пояснювальної записки	07.06.2021 – 08.06.2021	
7	Завершити оформлення пояснювальної записки	09.06.2021 – 10.06.2021	
8	Підготувати презентацію та графічні матеріали	11.06.2021 – 12.06.2021	

Дата видачі завдання: 24.05.2021р.

Керівник дипломної роботи (проекту) _____ Коба О.В.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Новохатко О.М.

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Веб-додаток CRUD з використанням технологій Spring і Hibernate»: 40 с., 16 рис., 16 літературних джерел.

ВЕБ-ДОДАТОК CRUD З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ SPRING І HIBERNATE.

Об’єкт дослідження – методи створення CRUD веб-додатку.

Предмет дослідження – розробка веб-додатку за допомогою технологій Spring та Hibernate.

Мета дослідження – розробка веб-додатку за допомогою Spring та Hibernate для автоматизації та полегшення керування процесу взаємодії з клієнтами сервісу оренди автомобілів.

Проведено аналіз предметної області, досліджено існуючі засоби та їх аналоги для взаємодії з клієнтами через UI. Виділено їх найкращі рішення та ті, які можна було б модифікувати та покращити. Було спроектовано базу даних, систему класів та їх взаємозв’язки.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ 6
ВСТУП 7
РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ АНАЛОГІВ АВТОМАТИЗАЦІЇ ВЗАЄМОДІЇ З КЛІЄНТОМ ЧЕРЕЗ КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС 9
1.1. Опис проблем, які вирішує система автоматизації взаємодії з клієнтом 9
1.2. Види систем автоматизації взаємодії з клієнтом 10
1.3. Існуючі провайдери систем автоматизації взаємодії з клієнтом 11
1.4. Існуючі методи розробки, які використовуються для створення CRM систем 12
1.5 Висновки до розділу 15
РОЗДІЛ 2 ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ ДЛЯ АВТОМАТИЗАЦІЇ ВЗАЄМОДІЇ З КЛІЄНТОМ 16
2.1. Вибір засобів розробки 16
2.2. Проєктування класів серверної частини додатку 17
2.3. Проєктування бази даних 23
2.4. Висновки до розділу 24
РОЗДІЛ 3 РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ АВТОМАТИЗАЦІЇ ВЗАЄМОДІЇ З КЛІЄНТОМ 25
3.1. Розробка бізнес-логіки додатку 25
3.2. Розробка користувацького інтерфейсу 31
ВИСНОВКИ 38
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ 40
Додаток А ..... **Ошибка! Закладка не определена.**
Додаток Б ..... **Ошибка! Закладка не определена.**
Додаток В ..... **Ошибка! Закладка не определена.**
Додаток Г ..... **Ошибка! Закладка не определена.**
Додаток Ґ ..... **Ошибка! Закладка не определена.**
Додаток Д ..... **Ошибка! Закладка не определена.**
Додаток Е ..... **Ошибка! Закладка не определена.**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

CRM – *Customer Relationship Management*

DNS – *Domain Name Service* (сервер доменних імен)

FTP – *File Transfer Protocol* (протокол передачі файлів)

HTML – *Hypertext Markup Language* (гіпертекстова мова описання)

HTTP – *Hypertext Transport Protocol* (протокол передачі гіпертексту)

SQL – *Structured Query Language* (структурована мова запитів) *URL*

REST – *Representational state transfer*

SOAP – *Simple Object Access Protocol*

ВСТУП

В наш час, дуже стрімко розвиваються системи, які надають можливості бізнесу з легкістю взаємодіяти зі своїми клієнтами, відслідковувати їх замовлення та зберігати їх до бази даних. Особливо стрімкого розвитку, такі системи зазнали під час кризи, пов'язаної з COVID-19. Велика кількість підприємств мусила прийняти реалії сьогодення, та адаптувати свій бізнес до роботи онлайн.

Тому актуальність роботи полягає саме у створенні веб-додатку, який би допоміг підприємцям вести інформацію про своїх клієнтів та їх замовлення онлайн, автоматизовано та з мінімальними навичками володіння комп'ютером.

Під автоматизованою роботою будемо розуміти те, що користувачу необхідно буде докласти мінімум зусиль для того, щоб занести інформацію до системи. Логіка роботи веб-додатку буде обмежувати людину, яка намагатиметься внести некоректну інформацію, виводячи попереджувальні повідомлення. Дії користувача, також, будуть обмежені згідно його ролі у системі.

Ведення інформації про клієнтів та її аналіз - одна з основних задач бізнеса, а автоматизація цього процесу дає змогу зберегти час та зусилля, та вкласти їх у інші, не менш важливі, аспекти ведення бізнесу. Також, підприємець може бути впевненим, що зібрана інформація буде збережена та доступна за будь-яких умов, адже буде налагоджено процеси бекапів, транзакцій та резервного копіювання, які надаються існуючими провайдерами баз даних, такими як MySql, PostgreSql чи MSSQL.

Під час створення веб-додатку такого типу, можуть виникнути питання, щодо обраної архітектури додатку, бази даних, мови програмування та способи представлення інформації користувачу. Таким чином, методами дослідження даної роботи є методи написання веб-додатків та мова об'єктно-орієнтованого програмування Java.

Під час написання додатку мовою Java, необхідно впевнитися у вірності обраних методів розробки, спроектованої архітектури додатку та бази даних.

Також, не менш важливим аспектом написання веб-додатку є відсутність конфліктів версій фреймворків та бібліотек, адже під час оновлення якогось

елементу, можуть виникнути непередбачувані помилки та ситуації, не передбачені системою.

Об’єкт дослідження: методи створення CRUD веб-додатку.

Предмет дослідження: розробка веб-додатку за допомогою технологій Spring та Hibernate.

Метою роботи є розробка веб-додатку за допомогою Spring та Hibernate для автоматизації та полегшення керування процесу взаємодії з клієнтами сервісу оренди автомобілів.

Для того аби досягти поставленої мети необхідно виконати наступні задачі:

- Провести аналіз існуючих рішень, та виділити їх переваги;
- Спроєктувати власний веб-додаток на основі даних, отриманих в ході аналізу;
- Обрати необхідні інструменти та методи розробки, які будуть використані у ході написання веб-додатку;
- Написати веб-додаток, використовуючи обрані інструменти та методи розробки, враховуючи результати проведеного аналізу існуючих аналогів.

Робота включає три розділи. При виконанні будуть використані різноманітні джерела інформації (веб-ресурси, монографії, методичні та теоретичні матеріали, тощо).

РОЗДІЛ 1

ОГЛЯД ІСНУЮЧИХ АНАЛОГІВ АВТОМАТИЗАЦІЇ ВЗАЄМОДІЇ З КЛІЄНТОМ ЧЕРЕЗ КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС

1.1. Опис проблем, які вирішує система автоматизації взаємодії з клієнтом

У сучасному бізнесі необхідність автоматизація різних процесів стала вже звичним явищем. Вже стає складно уявити собі складський або бухгалтерський облік без застосування спеціалізованого програмного забезпечення, торгові представники використовують спеціальні додатки для оформлення і відправки замовлення в офіс прямо з ноутбука або мобільного телефону, досить велика частина замовлень приходить з сайту вже у вигляді готових до обробки документів.

CRM-система — це прикладне програмне забезпечення для організацій, призначене для автоматизації стратегій взаємодії з замовниками (клієнтами), зокрема, для підвищення рівня продажів, оптимізації маркетингу і поліпшення обслуговування клієнтів шляхом збереження інформації про клієнтів і історію взаємин з ними, встановлення і поліпшення бізнес-процесів і подальшого аналізу результатів [1].

Що відбувається, якщо робота відділу продажів ведеться без системи обліку? Кожен менеджер з продажу працює так, як йому зручніше, веде фіксацію дзвінків, інших видів взаємодії з клієнтами на власний розсуд: хтось на папері, хтось в Excel таблицях, а хтось взагалі не вважає за потрібне фіксувати процес своєї роботи.

Вихід з цієї ситуації - автоматизація і стандартизація управління відносин з клієнтами, тобто впровадження CRM-системи, яка допоможе:

- Отримати загальну для компанії стандартизовану базу контактів (клієнтів, контрагентів);
- Ефективно здійснювати контроль якості роботи відділу продажів в будь-який момент часу;
- Планувати підвищення якості роботи та розробляти стратегію розвитку бізнесу

Таким чином, витративши час на розробку даної системи, можна позбутися проблем у майбутньому, економлячи величезні кошти та час.

1.2. Види систем автоматизації взаємодії з клієнтом

Існує два типи CRM-систем, створених на основі різних технологій:

- Saas (система як сервіс) – полягає у тому, що все програмне забезпечення та дані знаходяться на сервері на стороні постачальника послуг, а користувач отримує доступ до цієї системи через веб-додаток у браузері.

- Standalone – користувач отримує ліцензію на встановлення додатку на свій сервер та використання локально. Існує можливість модифікувати додаток під свої потреби самостійно.

Saas система, більш проста у використанні, адже не потребує значних навичок, а всі маніпуляції з системою проводить постачальник послуг, але така система потребує підключення до мережі.

Standalone, в свою чергу, надає більшу незалежність користувачеві та простір для самостійного внесення змін та налаштування системи.

Standalone система задовольнить потреби, наприклад, підприємств, які не мають постійного та безперебійного доступу до мережі, а Saas – позбавить користувача від зайвих маніпуляцій із системою та зробить її використання надзвичайно легким та доступним для розуміння.

1.3. Існуючі провайдери систем автоматизації взаємодії з клієнтом

Для аналізу та проектування своєї системи, необхідно розглянути вже існуючі та перевірені часом системи, перейняти від них переваги, та спробувати позбутися недоліків. Всі ці системи мають відмінності між собою, деякі із них безкоштовні та доступні у вільному доступі, деякі – потребують придбання ліцензії, але усіх їх об’єднує одне – мета, з якою вони створювалися.

1.3.1. Microsoft Dynamics CRM

Microsoft Dynamics 365 являє собою набір інтелектуальних додатків, розділених на кілька напрямків. Основними є: продажі, маркетинг та обслуговування клієнтів.

«Продажі» надають можливість відслідковувати та аналізувати інформації про клієнтів, виявляти вже готових до покупки, та тих, які потенційно можуть стати клієнтом. Дозволяє вести масив даних стосовно клієнта за допомогою користувацького інтерфейсу.

"Обслуговування клієнтів" надає багатоканальну взаємодію з клієнтами в міру необхідності. При цьому клієнти самі вибирають пристрої і канали зв'язку з компанією, в тому числі соцмережі.

Головною перевагою цієї системи є наявність значного арсеналу функцій, які доступні користувачеві. Також, це система, розроблена всесвітньо відомою компанією, а отже, рівень довіри до неї на високому рівні.

1.3.2. CRM FreshOffice

Як і попередній приклад, нараховує велику кількість функцій для роботи з даними про клієнта, але в додаток до цього, вбудований планувальник дозволяє призначати завдання членам робочої групи, стежити за ходом їх виконання і

здійснювати повний контроль над роботою. Можна контролювати весь цикл роботи, відслідковувати статуси та дати їх зміни.

1.4. Існуючі методи розробки, які використовуються для створення CRM систем

Для розробки веб-додатку необхідно врахувати велику кількість змінних, які можуть вплинути на його роботу. Найголовнішими з них є:

- архітектура додатку;
- мова програмування;
- архітектура бази даних

1.4.1. Архітектура додатку

Існує два підходи, при проектуванні веб-додатків:

- REST
- SOAP

Сервіси, які використовують REST архітектуру називаються RESTful сервісами.

REST-це абревіатура від Representational State Transfer («передача стану подання»). Це узгоджений набір архітектурних принципів для створення більш масштабованої та гнучкої мережі. Ці принципи відповідають на ряд питань. Які у системи компоненти? Як вони повинні взаємодіяти один з одним? Як бути впевненим, що можна замінювати різні частини системи в будь-який час? Як система може масштабуватися для обслуговування мільярдів користувачів? [2].

Такі сервіси мають відповідати таким принципам, а саме:

- Клієнт-сервер - перше обмеження вказує, що мережа повинна складатися з клієнтів і серверів. Сервер - це комп'ютер, який має необхідні ресурси, а клієнт - це комп'ютер, якому потрібно взаємодіяти з ресурсами, що зберігаються на сервері.

- Відсутність стану – клієнту, для взаємодії з сервером, не потрібно знати про стан системи. Йому не потрібна інформація про минулий чи майбутній стан,

клієнт має можливість надіслати запит на сервер в будь-який момент часу, та отримати відповідь.

- Однаковість інтерфейсу – говорить нам про те, що клієнт та сервер мають певну спільну мову, яка дозволяє їм проводити маніпуляції над ресурсами, змінювати, діставати та видаляти їх.

- Система слоїв – у системі має бути більше, ніж два шари. Окрім слоїв клієнт та сервер, можна додати шар «Контролер» для обробки HTTP запитів, або шар «Сервіс» для виконання бізнес-логіки додатку.

SOAP - це скорочення від Simple Object Access Protocol. Це протокол обміну повідомленнями на основі XML для обміну інформацією між комп'ютерами. SOAP, на відміну від RESTful, який використовується для розробки простих CRUD додатків, застосовують при проектуванні складних систем. Прикладом може бути система відслідковування курсу валют та вартості акцій на біржі.

Головною відмінністю SOAP є його стандартизованість та надійність. Використовуючи цей протокол обміну повідомленнями, ви можете бути впевнені, що все пройде так, як було заплановано та спроєктовано. Ціною такої надійності є складність таких систем. Використання цього протоколу потребує значного рівня кваліфікації розробника, а будь-яка помилка жорстоко карається.

RESTful сервіси, в свою чергу, дуже прості у використанні та створенні, але не надають таких широких можливостей налаштування та модифікування.

1.4.2. Мова програмування

Для розробки подібних додатків, найкраще використовувати об’єктно-орієнтовані, строго-типізовані мови програмування.

По-перше, використовуючи ООП та переваги цієї методології, значно легше спроєктувати систему, яка б повністю відображала реальну бізнес-модель.

По-друге, завдяки строгій типізації, буде досягнуто ефекту, що більшість можливих помилок, які можуть бути допущені в ході розробки додатку, будуть

виявлені на етапі компіляції. Дана особливість дасть змогу виправити помилки ще до моменту, коли буде проведено реліз продукту, а клієнт навіть не побачить їх.

Існує 2 монополісти у цій сфері, які задовольняють вищенаведені умови: Java та C#.

1.4.3. Архітектура бази даних

При проєктуванні таких великих систем, невід’ємною частиною успіху є правильне проєктування та обрання бази даних.

По-перше, необхідно завчасно подумати про те, що в майбутньому, базу необхідно буде масштабувати, а отже вона не має бути зв’язаною. Для цього, база має відповідати, як мінімум, 3-ій нормальній формі.

По-друге, потрібно обрати тип бази даних: реляційна або нереляційна. Кожна з них має свої особливості, а заміна типу у майбутньому буде занадто складною, тому необхідно завчасно продумати цей момент.

Реляційна база даних - це сукупність пов'язаних між собою двовимірних таблиць, в яких зберігається інформація про об'єкти. Запис (рядок) включає в себе дані про один об'єкт, а в полях (стовпцях) містяться різні його характеристики.

Нереляційна база даних, найчастіше використовує модель ключ-значення, яка являє собою найпростіший вид бази даних, будучи, по суті, асоціативним масивом - кожному значенню зіставляється свій унікальний ключ. Простота сховищ цього типу відкриває простори неймовірної масштабованості. Не потрібно ніяких схем побудови бази даних, немає ніякого зв'язку між значеннями, по суті кількість елементів асоціативного масиву обмежена лише обчислювальними потужностями. Саме тому даний вид сховищ цікавий в першу чергу компаніям, що надають послуги хмарного хостингу.

1.5 Висновки до розділу

У першому розділі було виявлено, що таке CRM та які проблеми вона вирішує. Розглянуто існуючі аналоги систем керування та автоматизації взаємодії з клієнтом, проаналізовано їх особливості, переваги та недоліки.

Також, значну увагу було приділено підходам розробки подібних систем, пройдено по кожному пункту та виявлено особливості та застереження при проєктуванні. Адже проектування подібних систем, немов конструктор, дозволяє в залежності від бізнес-вимог та середи використання, налаштовувати та конфігурувати додаток так, щоб він найбільше задовольняв потреби замовника. Кількість користувачів системи, складність виконаних операцій над даними, їх вид – усе це буде впливати на обрані інструменти, методи та підходи до розробк

РОЗДІЛ 2

ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ ДЛЯ АВТОМАТИЗАЦІЇ ВЗАЄМОДІЇ З КЛІЄНТОМ

2.1. Вибір засобів розробки

Для реалізації серверної логіки додатку було обрано об’єктно-орієнтовану мову програмування Java, використана IDE – IntelliJ IDEA. Для реалізації бази даних обрано реляційну базу – MySQL. Для опису зв’язків моделі класів та бази – фреймворк Hibernate. Для керування життєвим циклом та використання об’єктів Java – фреймворк Spring. Для втілення користувацького інтерфейсу та доведення інформації до користувача використано шаблонізатор Thymleaf, мова розмітки сторінок HTML та CSS для конфігурації стилів. Класи додатку та файли html представлені у ДОДАТКУ.

Java – це перевірена часом мова програмування, яка швидко розвивається. Ця мова має багато очевидних переваг перед іншими популярними мовами програмування високого рівня а саме:

- Мова має синтаксис, що легко читається, що дозволяє як полегшити розуміння логіки роботи програми, так і прискорити процес написання коду;
- Незалежність від платформ. Завдяки JVM, неважливо, на якому пристрої чи під якою ОС буде запускатися додаток;
- Строга типізація даних означає, що тип даних змінних буде відомий в момент компіляції, що значно підвищує надійність коду та зменшує час, який міг би бути витрачений на виявлення помилок, які проявляються на етапі компіляції. Автоматичне управління пам’яттю дозволяє не турбуватися про виділенні пам’яті та її звільненні;

- Велика кількість фреймворків, деякі з них мають дуже гарну документацію, наприклад, Spring та Hibernate.

2.2. Проєктування класів серверної частини додатку

У процесі роботи була створена ієрархія класів (див. Додаток А). Під час розробки використовувалась MVC (Model-View-Controller) архітектура, саме тому усі класи поділені на пакети:

- Контролер – сюди поміщено класи, які відповідають за обробку HTTP запитів, та їх подальше перенаправлення слоями додатку.

- Сервіси – класи, в яких прописана вся бізнес-логіка додатку. Саме сюди перенаправляють класи контролери.

- Репозиторії – класи, функція яких полягає в зв'язку з базою даних, використовуючи певні інтерфейси, надані Hibernate

- Моделі – класи сущності, описують реальні об'єкти бізнес моделі.

Класи моделі містять в собі інформацію, яка описує їх модель з реального світу, а поля - параметри. Наприклад, класс Client містить в собі такі поля, як: ім’я, адреса, місто, номер телефону, електронна пошта. А за допомогою Hibernate та використання анотацій, які він надає, проставлено та налаштовано створення відповідних таблиць та полів у базі даних, конфігурація їх зв’язків. В інших класах моделі змінюються тільки поля та параметри налаштування об’єднання їх з базою даних.

Класи контролери також не сильно відрізняються один від одного. Змінюється лише точка входу, яку відловлює контролер, та дії, які він проводить після його перехоплення. Наприклад, клас ClientController відповідає за перехоплення всіх точок входу, пов’язаних з клієнтами. Містить в собі сервіси, які він викликає всередині своїх методів. Ці сервіси впроваджуються в контролер за допомогою DI у Spring, використовуючи анотацію @Autowired. Обробивши запит, контролер перенаправляє його на відповідний сервіс, який, в свою чергу, проведе певну маніпуляцію з даними, та поверне результат.

Класи сервіси, напевне, найголовніші класи, серце програми. Вони відповідають за всю бізнес-логіку додатку. У них ми програмуємо ту логіку, яку очікуємо побачити в результаті роботи додатку. Наприклад, алгоритм обробки логіна та пароля користувача, при спробі входу, чи дії, які виконуються з моделями додатку – все це, відбувається саме у класах-сервісах.

В якості архітектурного патерну для додатку було обрано MVC, адже він найбільше підходить та ідеально задовольнить потреби додатку. Суть цього патерну полягає в тому, що додаток ділиться на прошарки, які мають виконувати лише одну функцію. Наприклад, взаємодія за базою даних, обробка HTTP запитів, делегування цих запитів всередині додатку. Це дозволяє мати гнучку архітектуру, яка дасть можливість з легкістю змінювати та редагувати логіку додатку, додавати до нього нові модулі та редагувати старі, без значного ефекту на логіку роботи всього додатку.

Схема шляху запиту користувача крізь додаток наведено на рис.2.1.

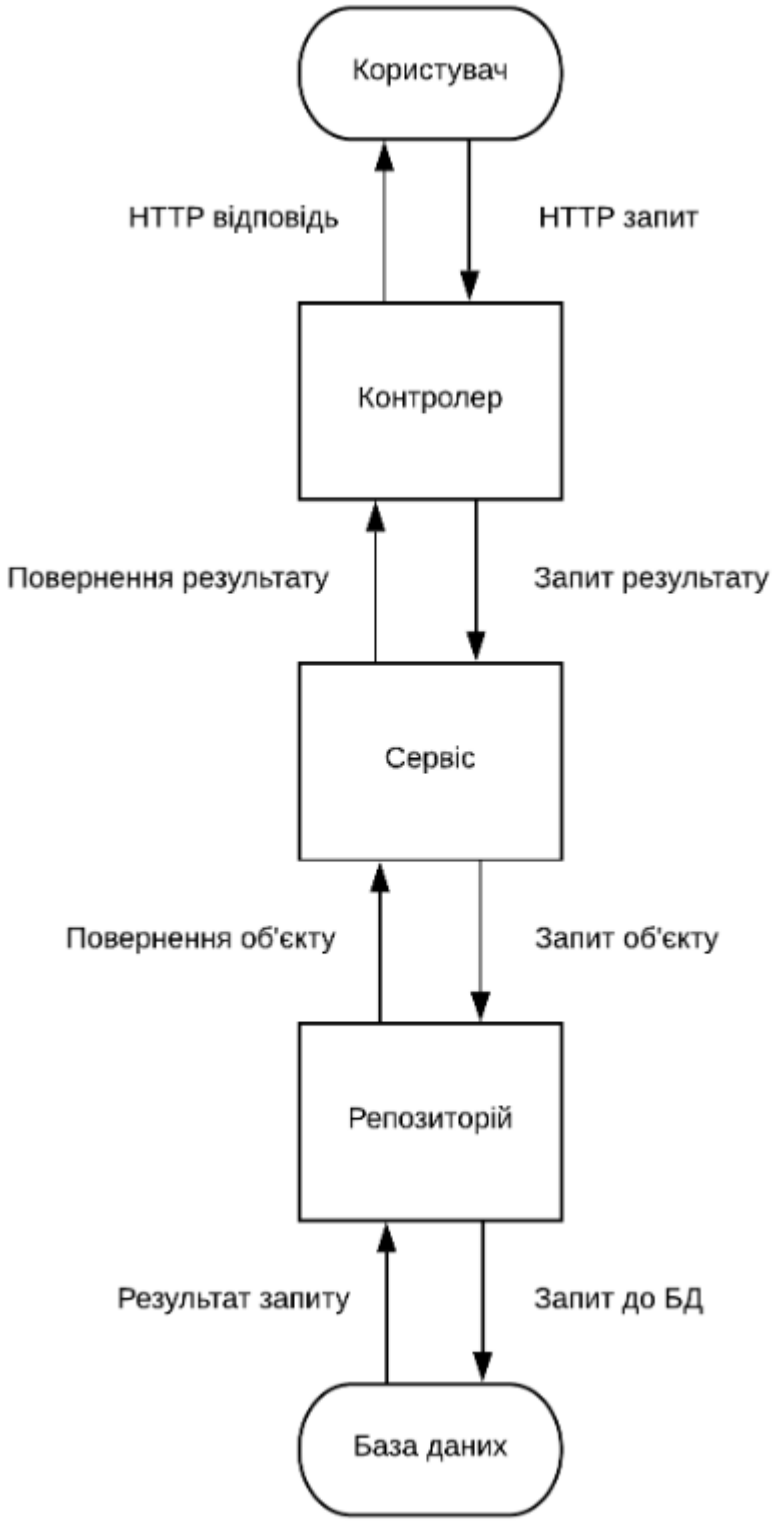

Рис.2.1 Схема шляху запиту користувача крізь додаток

Почнемо проєктування класів-моделей, які гратимуть роль шаблонів для зберігання даних. Виділимо найголовніші класи-моделі та розглянемо їх.

Клас Invoice міститиме в собі всю інформацію про замовлення, яка буде зберігатися в його полях. Наприклад, це така інформація як: ідентифікатор, дата замовлення, статус замовлення, ідентифікатор клієнта, коментарі до замовлення.

Клас Supplier міститиме в собі всю інформацію про постачальника, яка буде зберігатися в його полях. Наприклад, це така інформація як: ідентифікатор, назва постачальника, країна, місто, номер телефону, коментарі про замовника.

Клас Vehicle міститиме в собі всю інформацію про автомобілі, яка буде зберігатися в його полях. Наприклад, це така інформація як: ідентифікатор, тип автомобіля, дата реєстрації, закріплена локація автомобіля, ідентифікатор користувача автомобілем.

Клас Employee міститиме в собі всю інформацію про працівника фірми, яка буде зберігатися в його полях. Наприклад, це така інформація як: ідентифікатор, ім’я, тип працівника, дата найму.

Клас Country міститиме в собі всю інформацію про країни, яка буде зберігатися в його полях. Наприклад, це така інформація як: ідентифікатор, назва країни, столиця, континент, код країни.

Клас User міститиме в собі всю інформацію про замовлення, яка буде зберігатися в його полях. Наприклад, це така інформація як: ідентифікатор, логін, пароль.

Схеми класів представлено нижче на рис. 2.2 – 2.7

Рис.2.2 Схема класу Country

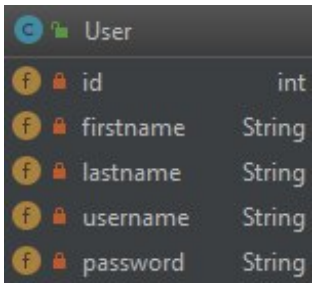

Рис.2.3 Схема класу User

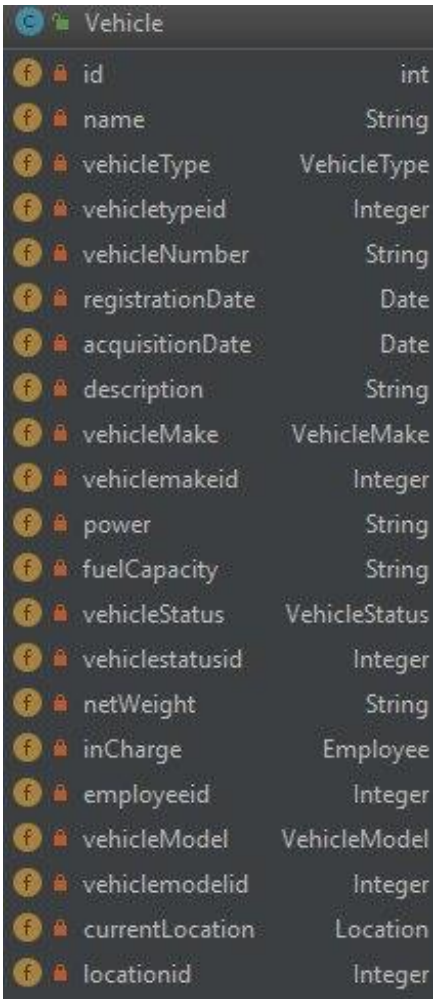

Рис.2.4 Схема класу Vehicle

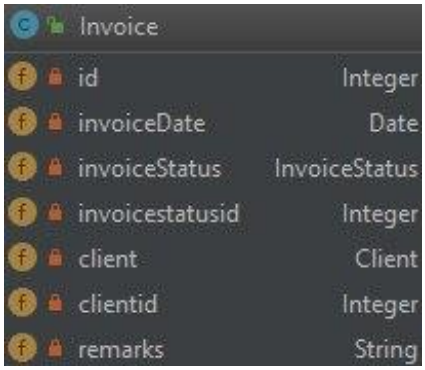

Рис.2.5 Схема класу Invoice

Рис.2.6 Схема класу Supplier

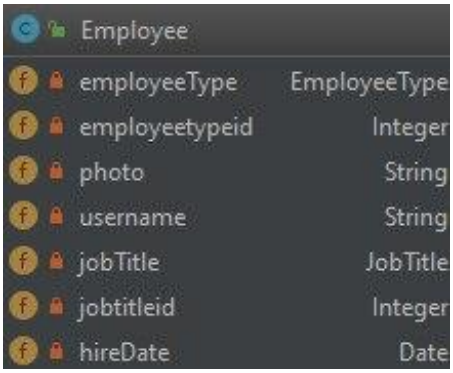

Рис.2.7 Схема класу Supplier

2.3. Проєктування бази даних

Також, для зберігання даних додатку, необхідно спроєктувати та створити структуру таблиць у базі даних. Для цього використаємо такий фреймворк як Hibernate.

Hibernate – це бібліотека, яка створена для мови програмування, яку ми використовуємо для написання додатку – Java. Дає можливість вирішувати проблеми об'єктно-реляційного відображення.

Дана бібліотека значно полегшує життя розробника під час створення бази даних та дає змогу сфокусуватися на бізнес-логіці.

Hibernate надає нам можливість використовувати весь арсенал його можливостей для створення баз даних. Завдяки написаним інтерфейсам для більшості провайдерів баз даних, ми можемо використовувати Hibernate, не прив’язуючись до конкретної бази даних.

Сам процес налаштування бази завдяки коду виглядає дуже просто. Користуючись анотацією @Entity над класом-моделлю, ми вказуємо Hibernate, що необхідно створити таблицю для цього класу. За замовчанням, таблиця матиме назву, ідентичну з назвою класу, але використовуючи анотацію @Table, можна змінити цю назву.

В якості полів для таблиці, буде використано поля класу. Логіка видачі назв аналогічна з логікою у таблиць. За замовчуванням береться назва поля, але завдяки анотації @Coloumn, ми маємо змогу корегувати цей процес.

Невід’ємною частиною таблиці у базі даних є її зовнішні та головні ключі, зв’язки з іншими таблицями. Все це також передбачено в даній бібліотеці. Вид зв’язку з іншою таблицею проставляється над полем, яке містить в собі параметр, який являється зовнішнім ключем. Робиться це анотаціями @OneToOne – для встановлення зв’язку один до одного, @OneToMany – для встановлення зв’язку один до багатьох, @ManyToOne – для встановлення зв’язку багато до одного та @Many-ToMany – для встановлення зв’язку багато до багатьох.

Поле, яке буде виконувати роль унікального ідентифікатора, проставляється за допомогою анотації @Id. Додатково, можна налаштувати стратегію інкрементації ідентифікатора, передавши параметр, що відповідає за це, як аргумент анотації.

2.4. Висновки до розділу

У другому розділі було спроєктовано систему класів та таблиць бази даних, враховуючи всі потреби нашого додатку та можливість шардінгу чи реплікації бази у майбутньому. Класи було спроєктовано таким чином, що всі прошарки майже не пов’язані між собою та виконують лише ті функції, які було покладено на них розробником. Це дасть змогу у майбутньому з легкістю, у разі потреби, змінювати логіку класів, додавати нові модулі та змінювати старі, не впливаючи при цьому на роботу вже написаних модулів. Модифікація системи за обраної архітектури також не викличе складнощів, адже запит користувача рухається чіткою та визначеною структурою у додатку, на кожному рівні задіючи лише ті класи, які необхідні йому для роботи. Саме для того, щоб мати цю гнучкість, було проведено аналіз існуючих архітектурних підходів написання додатків та обрано найбільш підходящий.

РОЗДІЛ 3

РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ АВТОМАТИЗАЦІЇ ВЗАЄМОДІЇ З КЛІЄНТОМ

3.1. Розробка бізнес-логіки додатку

В першу чергу, необхідно попіклуватися про те, аби додаток виконував свою основну функцію – обробка даних. Для цього нам треба написати систему класів, яка реалізувала би необхідну нам бізнес-логіку.

3.1.1. Розробка класів-моделей

Почнемо з написання класів-моделей, які являються шаблонними та вказують структуру, за якою будуть зберігатися дані.

Розробимо клас Invoice, який буде відповідати за замовлення, та міститиме в собі такі поля як: ідентифікатор, дата замовлення, статус замовлення, ідентифікатор клієнта, коментарі до замовлення.

Для роботи з цими полями, нам знадобляться конструктори, гетери та сетери. Для того, щоб не повторювати код та зберегти його читабельність використаємо плагін Lombok, який надає нам анотації для легкої інтеграції гетерів, сетерів та конструкторів у код. Наприклад, анотація @Getter згенерує гетери для всіх полів, а анотація @Setter – сетери. Конструктори з аргументами та без налаштовуються з такою ж легкістю. Анотація @NoArgsConstructor додає конструктор без параметрів, а анотація @AllArgsConstructor – конструктор за усіма параметрами. Всі ці дії можна замінити однією надзвичайно зручною анотацією @Data, вона додає все одразу.

До того ж, цей плагін зовсім не впливає на швидкість роботи додатку, адже всі маніпуляції з кодом виконуються перед етапом компіляції.

Розробимо клас Supplier, який буде відповідати за всю інформацію про постачальника, та міститиме в собі такі поля як: ідентифікатор, назва постачальника, країна, місто, номер телефону, коментарі про замовника.

Розробимо клас Vehicle, який буде відповідати за всю інформацію про автомобілі, та міститиме в собі такі поля як: ідентифікатор, тип автомобіля, дата реєстрації, закріплена локація автомобіля, ідентифікатор користувача автомобілем.

Розробимо клас Employee, який буде відповідати за всю інформацію про працівника фірми, та міститиме в собі такі поля як: ідентифікатор, ім’я, тип працівника, дата найму.

Розробимо клас Country, який буде відповідати за всю інформацію про країни, та міститиме в собі такі поля як: ідентифікатор, назва країни, столиця, континент, код країни.

Розробимо клас User, який буде відповідати за всю інформацію про користувача, та міститиме в собі такі поля як: ідентифікатор, логін, пароль.

Використовуючи анотації, які надає нам Spring, ми вказуємо, що об’єкти цих класів необхідно помістити в контейнер, а вже надалі, Spring, за потреби, буде видавати їх нашому додатку. Весь процес життєвого циклу Java об’єктів Spring бере на себе. Це значною мірою полегшує розробку та позбавляє нас від прив’язаності до ручного створення об’єктів.

Для того, щоб помітити клас як той, який треба помістити до контейнера ми використовуємо анотацію @Component.

3.1.2. Розробка класів-сервісів

Наступним, не менш важливим, прошарком є класи-сервіси. Вони містять в собі всю логіку роботи додатку, приймають запроси від класів-контролерів, та роблять свої запроси до класів-репозиторіїв.

Всі задачі додатку можна поділити за певною логікою. Наприклад, логіку обробки інформації про клієнта ми можемо винести в окремий сервіс, який буде в собі містити всі необхідні методи, які будуть використані під час роботи додатку.

Класи сервіси помічаються за аналогією з класами-моделями, але трохи іншою анотацією - @Service. Ця анотація має таку саму логіку всередині, але бажано розділяти свої класи за логікою, яку вони виконують. Це збільшує рівень читабельності коду та розуміння бізнес-процесів, які реалізує програма.

Також, методи цих класів будуть використовувати в собі класи-репозиторії, які дадуть змогу взаємодіяти з базою даних. Це знадобиться під час виконання операцій додавання, видалення, редагування чи отримання.

Після цього, класи-сервіси матимуть змогу обробити отримані з класів-репозиторіїв дані, та провести з ними додаткові маніпуляції, а вже надалі, повернути результат класу-контролеру.

Розробимо клас-сервіс ClientService. Він відповідає за обробку даних клієнта, їх збереження, видалення, редагування та додавання. Для цього створимо методи, які будуть виконувати цю логіку. Нам знадобляться методи для отримання клієнта за його ідентифікатором, отримання всіх клієнтів, видалення клієнта за його ідентифікатором та редагування клієнта за його ідентифікатором.

Цей клас буде використовувати в собі клас-репозиторій ClientRepository, а отже, необхідно запровадити його, використовуючи інструменти Spring, а саме: анотацію - @Autowired.

Розробимо клас-сервіс CountryService. Він відповідає за обробку даних країни, їх збереження, видалення, редагування та додавання. Для цього створимо методи, які будуть виконувати цю логіку. Нам знадобляться методи для отримання країни за її ідентифікатором, отримання всіх країн, видалення країни за її ідентифікатором та редагування країни за її ідентифікатором.

Цей клас буде використовувати в собі клас-репозиторій CountryRepository, а отже, необхідно запровадити його, використовуючи інструменти Spring, а саме: анотацію - @Autowired.

Розробимо клас-сервіс InvoiceService. Він відповідає за обробку даних замовлення, їх збереження, видалення, редагування та додавання. Для цього створимо методи, які будуть виконувати цю логіку. Нам знадобляться методи для отримання замовлення за його ідентифікатором, отримання всіх замовлень, видалення замовлення за його ідентифікатором та редагування замовлення за його ідентифікатором.

Цей клас буде використовувати в собі клас-репозиторій InvoiceRepository, а отже, необхідно запровадити його, використовуючи інструменти Spring, а саме: анотацію - @Autowired.

Розробимо клас-сервіс EmployeeService. Він відповідає за обробку даних працівника, їх збереження, видалення, редагування та додавання. Для цього створимо методи, які будуть виконувати цю логіку. Нам знадобляться методи для отримання працівника за його ідентифікатором, отримання всіх працівників, видалення працівника за його ідентифікатором та редагування працівника за його ідентифікатором.

Цей клас буде використовувати в собі клас-репозиторій EmployeeRepository, а отже, необхідно запровадити його, використовуючи інструменти Spring, а саме: анотацію - @Autowired.

Розробимо клас-сервіс VehicleService. Він відповідає за обробку даних автомобіля, їх збереження, видалення, редагування та додавання. Для цього створимо методи, які будуть виконувати цю логіку. Нам знадобляться методи для отримання автомобіля за його ідентифікатором, отримання всіх автомобілів, видалення автомобіля за його ідентифікатором та редагування автомобіля за його ідентифікатором.

Цей клас буде використовувати в собі клас-репозиторій VehicleRepository, а отже, необхідно запровадити його, використовуючи інструменти Spring, а саме: анотацію - @Autowired.

3.1.3. Розробка класів-репозиторіїв

Дані класи призначені для взаємодії з базою даних та надання можливості основній частині додатку проводити цю взаємодію на автоматичному рівні.

Взагалі, за допомогою Hibernate та мови запитів JPQL, яку він використовує для формування SQL запитів, ми можемо прописувати запити до бази даних напряму, але зважаючи на те, що в нас простий додаток без складних запитів, ми будемо використовувати можливості Spring Data JPA.

Для початку, помічаємо наш клас-репозиторій анотацією @Repository. Це повідомить Spring про те, що даний клас необхідно помістити до контейнера. Відмінністю від анотацій @Component та @Service є те, що класи, помічені даною анотацією, будуть відловлювати SQL помилки, оскільки вони напряму взаємодіють з базою даних.

Використовуючи Spring Data JPA, прописуємо методи для взаємодії з базою даних, спираючись на логіку звичайної мови. Те, що ми прописуємо словами, буде інтерпретовано в SQL запити. Наприклад, для того, аби отримати всіх клієнтів за певним ім’ям, назвемо метод findAllByName, а в якості параметра передамо ім’я, за яким будемо шукати.

Використовуючи цю властивість, ми збільшуємо читабельність коду, та позбавляємо себе необхідності напряму писати запити до бази даних.

А за замовченням, всі базові запити до бази, вже імплементовані у базовому класі-ропозиторії JpaRepository, від якого успадковуються всі наші класи-репозиторії.

Завдяки цьому, створення цих класів зводиться до оголошення класу та встановлення типу, який він може повертати. Наприклад Client чи Vehicle.

3.1.4. Розробка класів-контролерів

Ці класи, за аналогією з усіма іншими, діляться на прошарки, відповідно логіки та вхідних точок програми, за які вони відповідають. Також, необхідно помітити їх анотацією @Controller для того, щоб Spring помістив його до

контейнера. Головною відмінністю від інших класів є те, що даний прошарок перехоплює та оброблює HTTP запити. Програмно, це робиться за допомогою таких анотацій: @GetMapping, @PostMapping, @DeleteMapping, @PatchMapping – які проставляються, безпосередньо, над методами класу. А анотація @RestController дозволяє вказати який URL буде відловлювати даний клас.

Розробимо клас-контролер ClientController, якій буде перехоплювати та оброблювати всі URL, пов’язані з клієнтами.

Для перехоплення частини URL, створюємо методи, які будуть перенаправляти запити до відповідних сервісів. Такими методами будуть методи для отримання клієнта за його ідентифікатором, отримання всіх клієнтів, видалення клієнта за його ідентифікатором та редагування клієнта за його ідентифікатором.

Розробимо клас-контролер CountryController, якій буде перехоплювати та оброблювати всі URL, пов’язані з країнами.

Для перехоплення частини URL, створюємо методи, які будуть перенаправляти запити до відповідних сервісів. Такими методами будуть методи для отримання країни за її ідентифікатором, отримання всіх країн, видалення країни за її ідентифікатором та редагування країни за її ідентифікатором.

Розробимо клас-контролер EmployeeController, якій буде перехоплювати та оброблювати всі URL, пов’язані з працівниками.

Для перехоплення частини URL, створюємо методи, які будуть перенаправляти запити до відповідних сервісів. Такими методами будуть методи для отримання працівника за його ідентифікатором, отримання всіх працівників, видалення працівника за його ідентифікатором та редагування працівника за його ідентифікатором.

Розробимо клас-контролер InvoiceController, якій буде перехоплювати та оброблювати всі URL, пов’язані з замовленнями.

Для перехоплення частини URL, створюємо методи, які будуть перенаправляти запити до відповідних сервісів. Такими методами будуть методи для отримання замовлення за його ідентифікатором, отримання всіх замовлень,

видалення замовлення за його ідентифікатором та редагування замовлення за його ідентифікатором.

Розробимо клас-контролер VehicleController, якій буде перехоплювати та оброблювати всі URL, пов’язані з автомобілями.

Для перехоплення частини URL, створюємо методи, які будуть перенаправляти запити до відповідних сервісів. Такими методами будуть методи для отримання автомобіля за його ідентифікатором, отримання всіх автомобів, видалення автомобіля за його ідентифікатором та редагування автомобіля за його ідентифікатором.

3.2. Розробка користувацького інтерфейсу

Зазвичай, розробку користувацького інтерфейсу виконують за допомогою окремого додатку, який, використовуючи вхідні точки серверного додатку, відображає отримані дані на сайт, мобільний додаток чи ще кудись.

Проте, зважаючи на те, що наш додаток не має потреби в цьому, ми можемо налаштувати відображення всередині серверної частини додатку. Для цього скористаємося шаблонізатором HTML сторінок – Thymleaf, та, безпосередньо, самими HTML.

Коли користувач вперше потрапляє до додатку, він бачить сторінку для авторизації (див. рис. 3.1), де він може ввести свої дані для входу, чи зареєструватися, як новий користувач.

Також, він має можливість обрати пункт «запам’ятати мене», аби не вводити свої дані протягом деякого часу після виходу. Це реалізується за допомогою сесій.

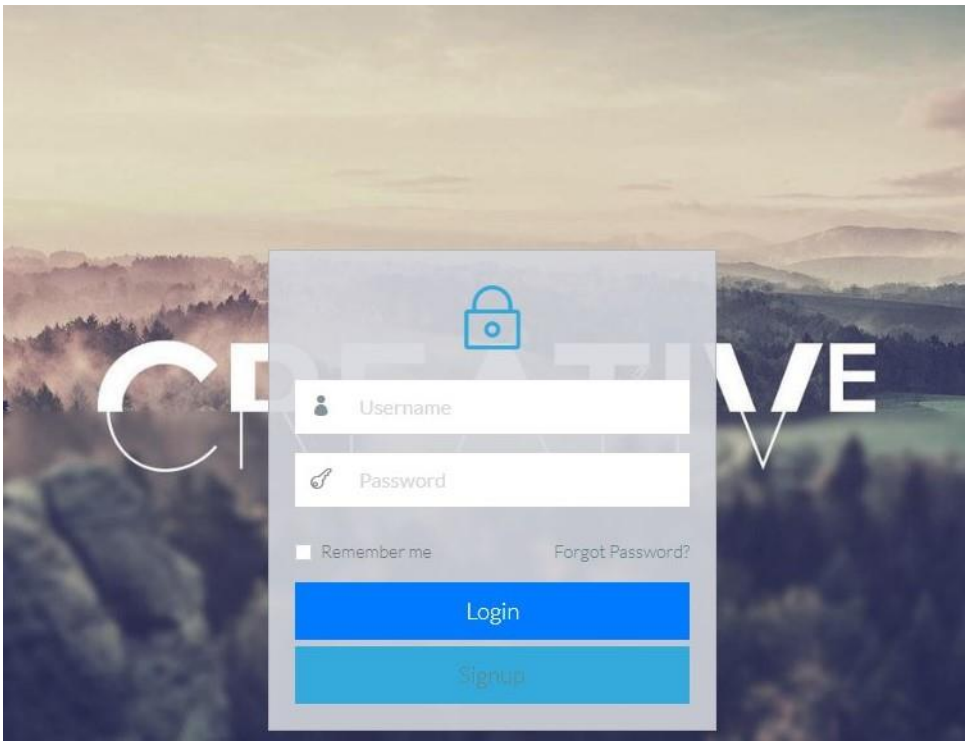

Рис.3.1 Сторінка входу

У випадку, якщо користувач захоче створити новий акаунт, після натискання на кнопку «зареєструватися» він потрапить на сторінку, де зможе ввести свої реєстраційні дані (див. рис 3.2).

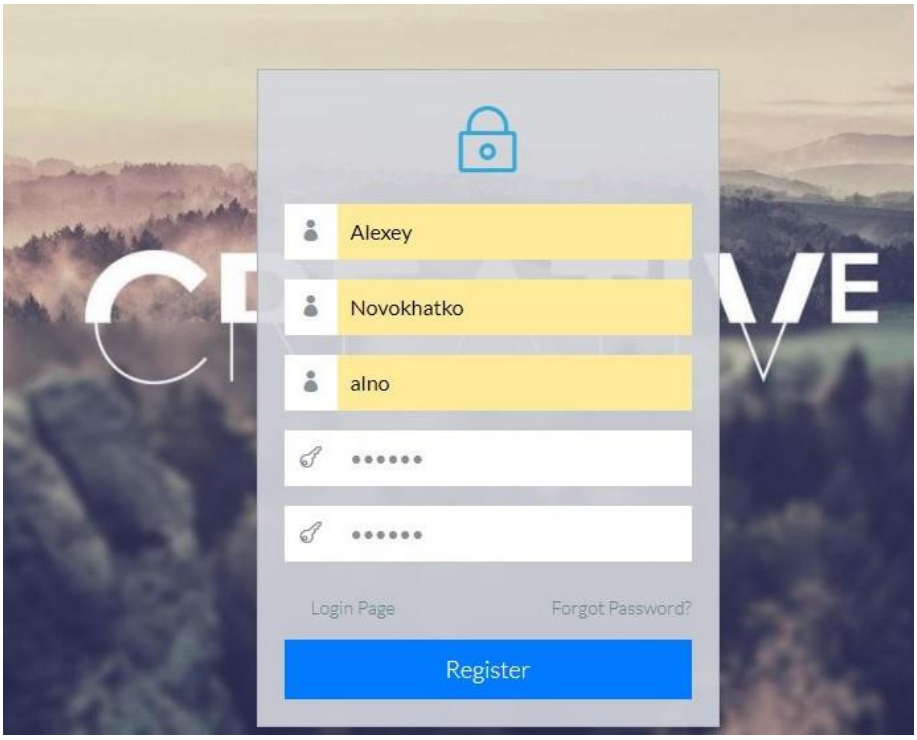

Рис.3.2 Сторінка реєстрації

Після реєстрації, користувач повернеться до сторінки входу, де зможе ввести свої дані, та успішно зайти до системи (див. рис 3.3 та рис 3.4).

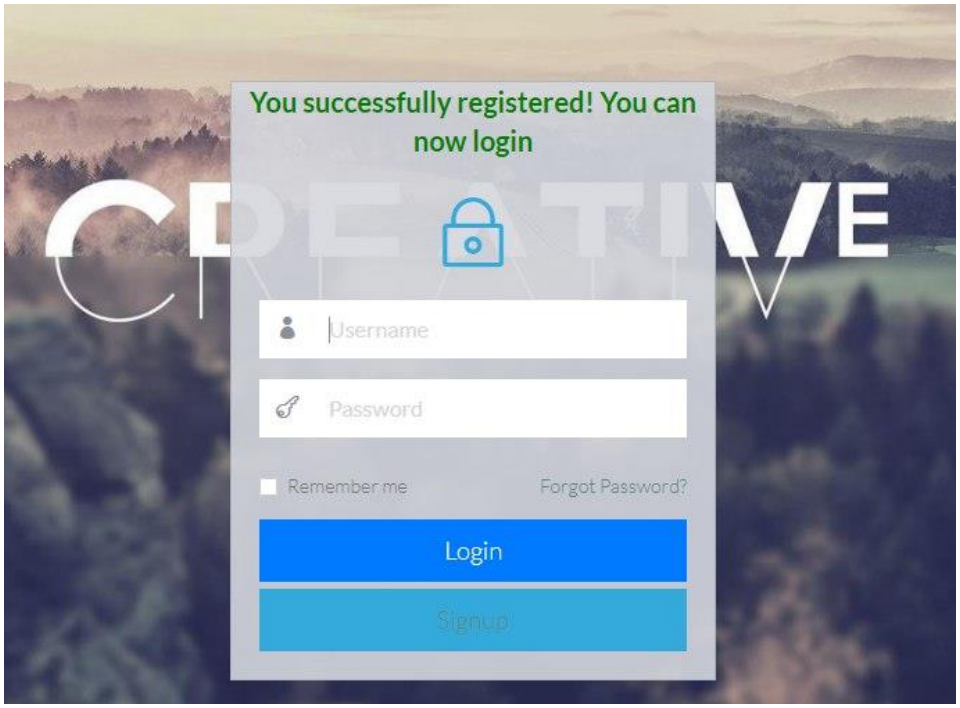

Рис.3.3 Сторінка входу після успішної реєстрації

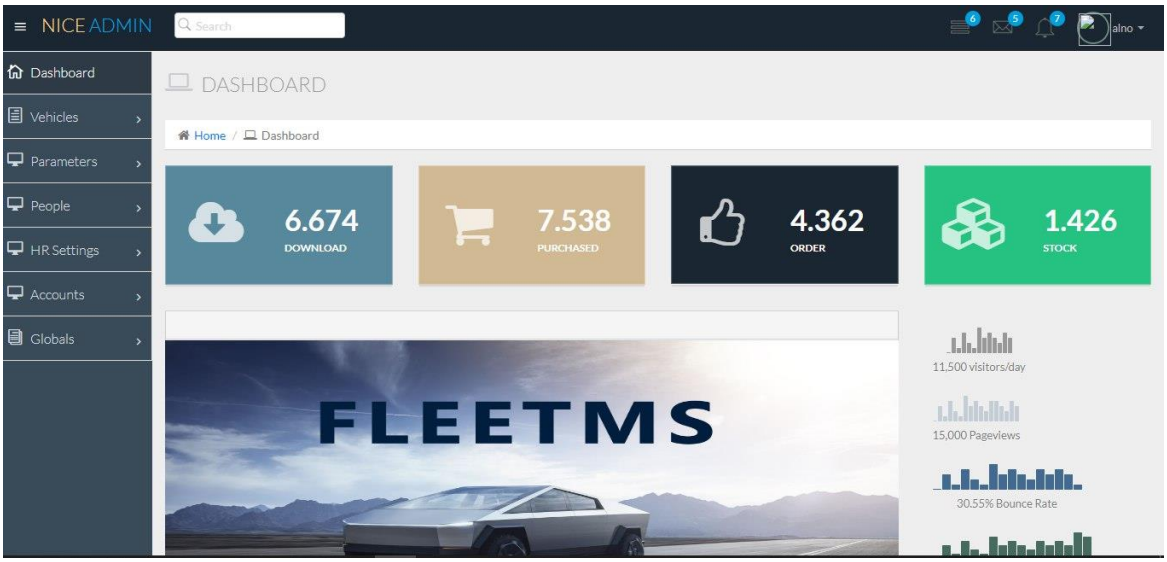

Рис.3.4 Головна сторінка додатку

На головній сторінці ми можемо бачити меню, яке розкривається та містить в собі додаткові підпункти меню. За допомогою них ми можемо потрапляти до різних частин додатку та виконувати різні маніпуляції з даними.

Наприклад, для створення нового типу автомобіля, перейдемо до вкладки «Автомобілі» та оберемо підпункт «Тип автомобіля». Ми побачимо список з усіма створеними нами типами автомобілів та елементи інтерфейсу для взаємодії з ними.

Виглядатиме це однаково на всіх сторінках (див. рис. 3.5), за винятком назви колонок.

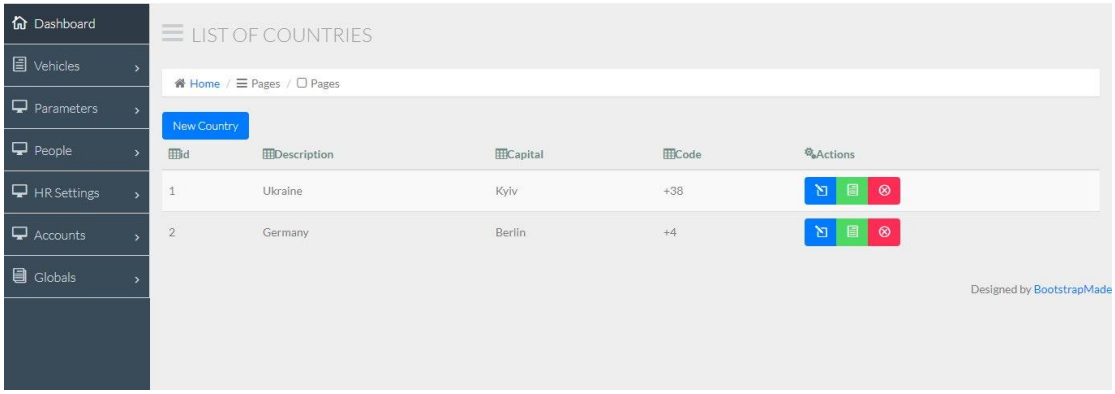

Рис.3.5 Сторінка зі списком всіх країн

Взаємодіячи з елементами інтерфейсу, ми можемо змінювати, видаляти чи додавати нові дані.

Після взаємодії, наприклад, з кнопкою редагування, нам відкриється нове вікно, в якому ми зможемо змінити вже існуючі дані (див. рис. 3.6). Або додати нові, також використовуючи вікно, яке відкриється (див. рис. 3.7).

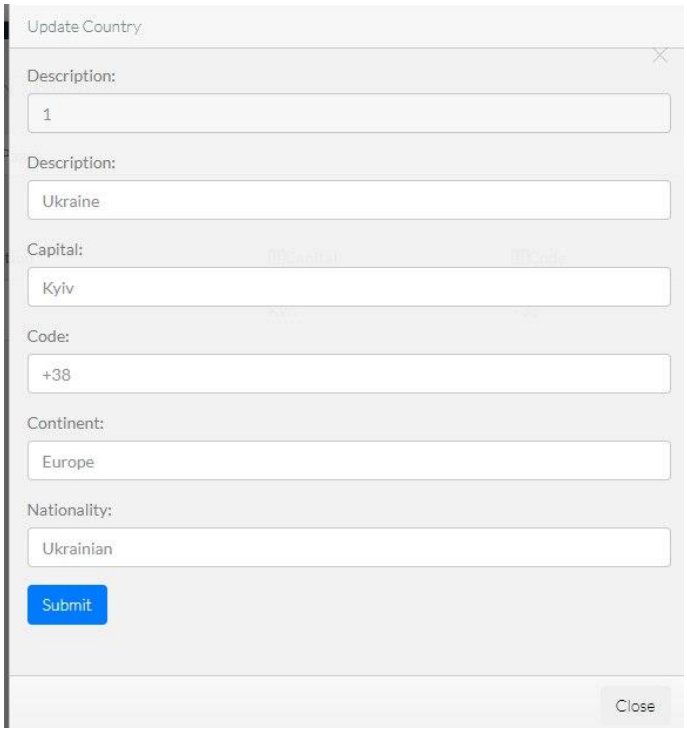

Рис.3.6 Вікно редагування існуючої країни

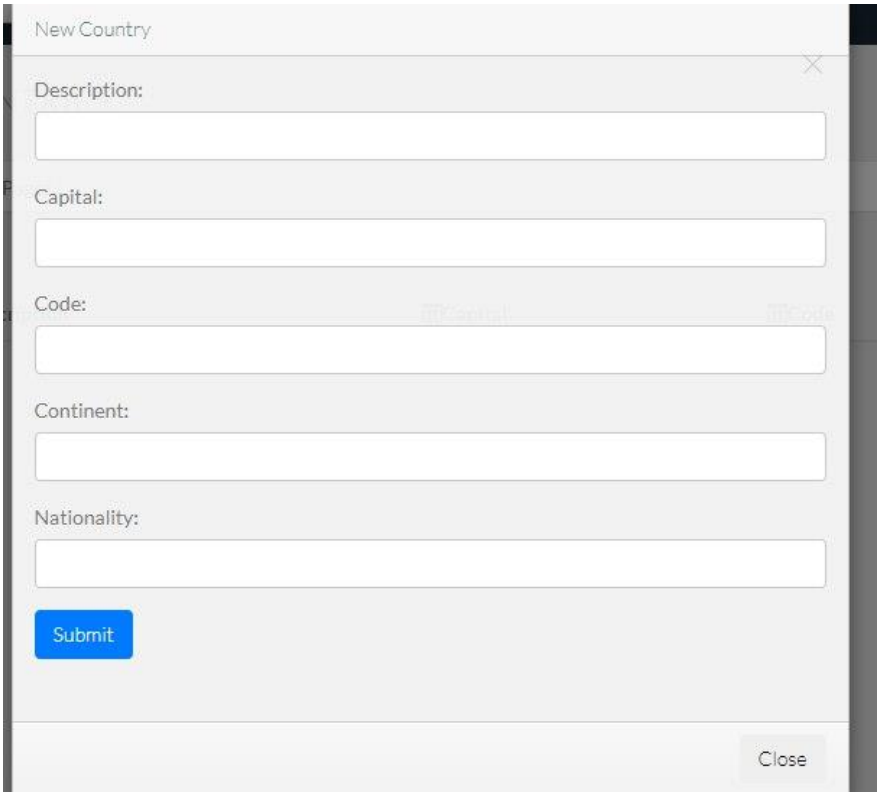

Рис.3.7 Вікно створення нової країни

Також, на деяких сторінках, при створенні нового об’єкту, ми можемо бачити більш складні форми заповнення даних, як, наприклад, під час створення нового замовлення (див. рис. 3.8). Там ми бачимо поля, які дають змогу обрати певні параметри, спираючись на вже створені нами об’єкти, такі як: тип автомобіля, виробник, країна і так далі. Також, додано зручне вікно для введення дати.

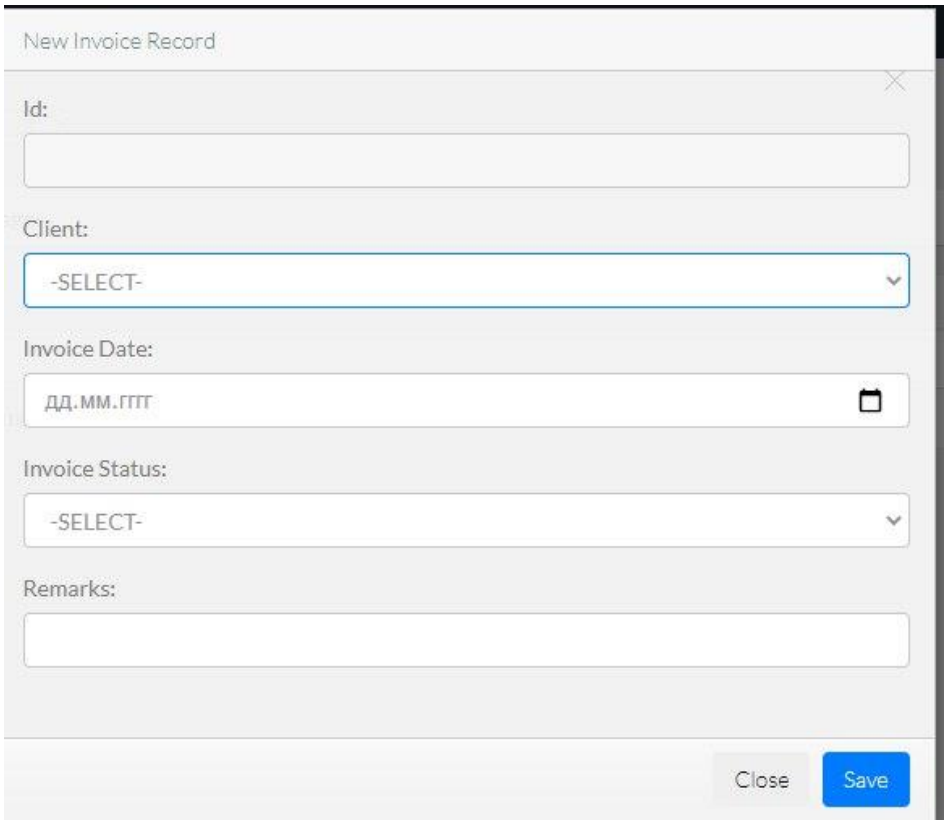

Рис.3.8 Вікно створення замовлення

Іншим прикладом складної форми створення об’єкта є сторінка створення нового працівника. Вона містить в собі складні, створені нами, конструкції, та дає змогу конфігурувати дані про працівника, спираючись на вже існуючі дані у системі (див. рис 3.9).

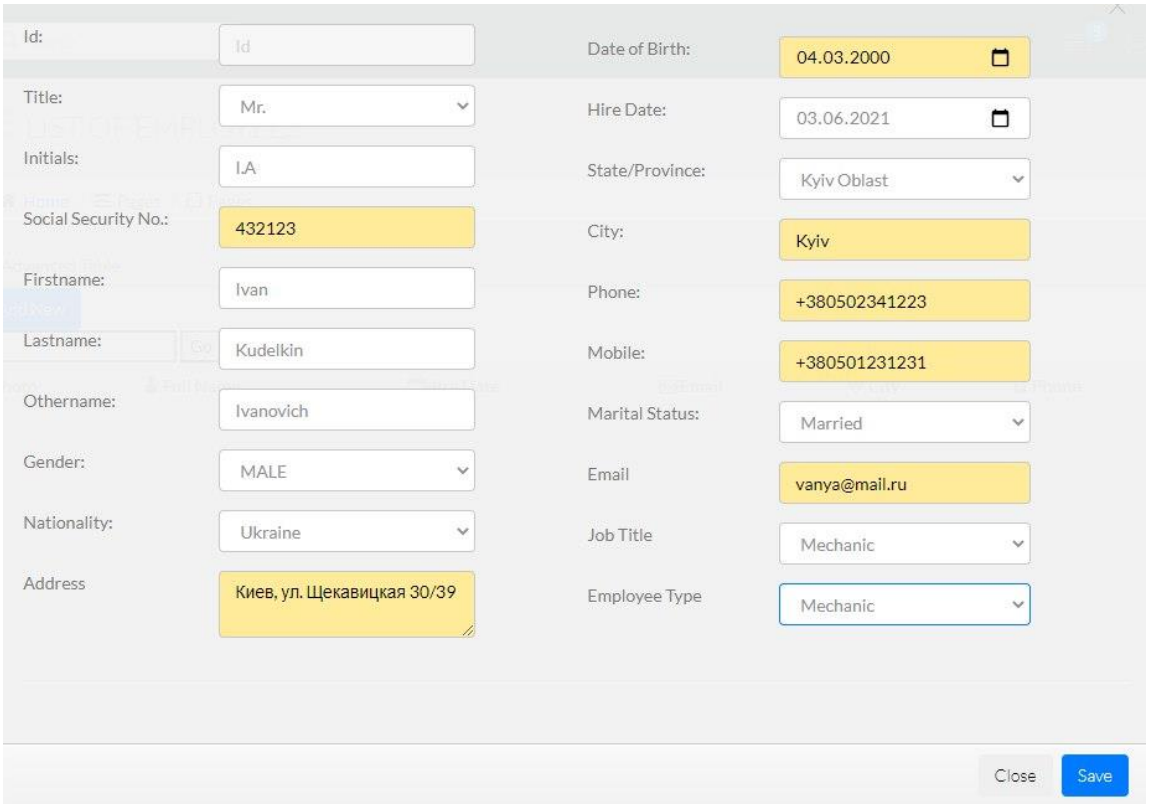

Рис.3.9 Вікно створення замовлення

ВИСНОВКИ

Веб-додатки – одна з найпопулярніших речей на даний момент, зважаючи на те, що весь світ похитнувся від COVID-19, та змушений був перейти онлайн.

Було проаналізовано існуючі рішення для взаємодії з клієнтом без зайвих проблем, перейнято від них найкращі сторони, та зроблено спробу уникнути їх недоліків.

На основі цього аналізу, було спроєктовано додаток, його архітектуру, базу даних. Також, було враховано всі можливі проблеми, які можуть виникнути під час розробки та використання додатку, та заздалегідь продумано засоби, завдяки яким ці проблеми будуть вирішені.

Не менш важливим є то, що під час проєктування системи було використано архітектурні патерни, які найбільше підходять для нашого додатку. Це дало змогу не піклуватися занадто сильно над проблемними місцями додатку, та сфокусуватися над розробкою та імплементацією бізнес-логіки.

До того ж, бізнес-логіка була побудована та спроектована таким чином, що її реалізація була не важкою. Додатком до цього, було використано найсучасніші фреймворки та плагіни, які використовуються на реальних комерційних проектах.

В даній роботі було спроектовано всі прошарки веб-додатку, реалізовано їх. Створено користувацький інтерфейс.

Таким чином у роботі було зроблено наступне:

- Проведено аналіз існуючих рішень, та виділено їх переваги;
- Спроєктовано власний веб-додаток на основі даних, отриманих в ході аналізу;
- Обрано необхідні інструменти та методи розробки, які були використані у ході написання веб-додатку;
- Написано веб-додаток, використовуючи обрані інструменти та методи розробки, враховуючи результати проведеного аналізу існуючих аналогів.

Результати роботи можуть бути застосовані для зручної взаємодії малого бізнесу із клієнтами та ведення власної бази даних по ним. Відштовхуючись від

даного каркасу додатку, вносячи незначні зміні, можна адаптувати його майже під будь-яку сферу та бізнес-модель. Це значно полегшує інтеграцію системи та її адаптацію під різні запити замовника.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Базы данных: Учебник для ВУЗов / Под ред. А.Д.Хомоненко – СПб: Корона принт, 2012. – 416 с.
2. Електронний ресурс. – Режим доступу: https://habr.com/ru/company/trinion/blog/249633/
3. Електронний ресурс. – Режим доступу: https://habr.com/ru/company/hexlet/blog/274675
4. К.Дж.Дейт. Введение в системы баз данных.: Пер. С англ. К.: Диалектика, 2008. – 784с.
5. Блох Джошуа. Effective Java: Учеб. пособие, 2014. – 466 с.
6. Кей Хорстманн. Библиотека профессионала/ Java 11: одиннадцатое издание. 2019. – 966с.
7. Проектирование информационных систем: курс лекций: учеб. Пособие для студентов вузов, обучающихся по специальностям в области информ. технологий / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. – М.: Интернет-Ун-т Информ. технологий, 2015. – 304с.
8. Кей Хорстманн. Библиотека профессионала/ Java 11: девятое издание. 2017. – 926с.
9. Ульман, Д. Введение в системы баз данных /Д.Ульман, Д.Уидом; Пер. с англ. – М.: Лори , 2000. – 512 с.
10. Фаулер М., Скотт К. *UML*. Основы. – Пер. с англ. – СПб.: Символ- Плюс, 2002.
11. Кей Хорстманн. Библиотека профессионала/ Java 11: седьмое издание. 2014. – 936с.
12. Електронний ресурс. – Режим доступу: https://docs.oracle.com/en/java/javase/16/
13. Електронний ресурс. – Режим доступу: https://docs.spring.io/spring-framework/docs/current/reference/html/
14. Електронний ресурс. – Режим доступу: https://dev.mysql.com/doc/

15. Електронний ресурс. – Режим доступу: https://hibernate.org/orm/documentation/5.4/
16. Електронний ресурс. – Режим доступу: https://www.thymeleaf.org/documentation.html