Міністерство освіти і науки України

Національний технічний університет

«Дніпровська

політехніка»

Інститут електроенергетики

(інститут)

факультет інформаційних технологій

(факультет)

Кафедра інформаційних систем та технологій та комп'ютерної інженерії

(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня магістра

(бакалавра, спеціаліста, магістра)

Студента Романова Євгена Ігоровича

(ПІБ)

академічної групи 126м-20-1

(шифр)

спеціальності 126 « Інформаційні системи та технології»

(код і назва спеціальності)

за освітньо-професійною програмою

«Інформаційні системи та технології»

на тему Розробка інформаційної системи управління доставкою вантажу на мові Java

(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційно їроботи	Сергєєва К.Л.			
розділів:				

Рецензент	Алексєєв М.О.			

Нормоконтролер	Коротенко Г.М.			

Дніпро
2022

ЗАТВЕРДЖЕНО:

завідувач кафедри

інформаційних технологій та комп'ютерної інженерії

(Повна назва)

_____ Гнатушенко В.В.

(підпис) (прізвище, ініціали)

«_____» 2022 року

ЗАВДАННЯ

на кваліфікаційну роботу

ступеня магістр

(бакалавра, спеціаліста, магістра)

студенту Романову Є.І. **академічної групи** 126м-20-1

(прізвище та ініціали) (шифр)

спеціальності 126 « Інформаційні системи та технології »

за освітньою-професійною програмою «Інформаційні системи та технології»

на тему Розробка інформаційної системи управління доставкою вантажу на мові Java затверджену наказом ректора НТУ «Дніпровська політехніка» від № 2241-Л

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз стану області рішення задачі	1.10.2021 – 31.10.2021
Розділ 2	Моделі та методи рішення задачі	1.11.2021 – 30.11.2021
Розділ 3	Програмна реалізація розроблених компонентів комп'ютерної інформаційної системи	1.12.2021 – 21.12.2021

Завдання видано _____ Сергєєва К.Л.

(підпис керівника) (прізвище, ініціали)

Дата видачі 1.10.2021 р.

Дата подання до екзаменаційної комісії 23.12.2021 р.

Прийнято до виконання _____ Романов Є.І.

(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 104 стор., 50 рис., 5 додатків, 26 джерел.

Об'єкт дослідження: процеси і алгоритми комунікації клієнтів та приватних вантажоперевізників.

Предмет дослідження: web-додаток для керування вантажоперевезення.

Мета магістерської роботи: вдосконалити методи комунікації клієнтів та приватних вантажоперевізників.

У вступі подано актуальність проблеми на визначення наукової новизни.

У першому розділі зроблено огляд можливої архітектури схожих проектів а також популярні протоколи передачі даних.

У другому розділі наведена програмна складова вирішення завдання, обґрунтувано вибір технологій та бібліотек системи.

У третьому розділі виконано побудову складових системи керування вантажоперевезеннями та обрано платформу для розгортання сервісу.

Наукова новизна полягає у вдосконаленні методів комунікації клієнтів та приватних вантажоперевізників через розробку нового web-додатка з додаванням інтерактивної карти та можливістю автоматичного прорахунку вартості та відстані доставки.

HEROKU, GITHUB, TOMCAT, SERVLET, JAVA, JSP, JSTL, HTML, CSS, JS, MAPBOX, API, POSTGRESQL, AJAX, SCM.

ABSTRACT

Explanatory note: 81 pages, 50 figures, 5 appendices, 25 sources.

Object of research: processes and algorithms of communication of clients and private carriers.

Subject of research: web-application for freight management.

The purpose of the master's work: to improve the methods of communication between customers and private carriers.

The introduction presents the urgency of the problem to determine the scientific novelty.

The first section provides an overview of the possible architecture of similar projects as well as popular data transfer protocols.

The second section presents the software component of the problem,

the choice of technologies and libraries of the system is substantiated.

In the third section, the construction of the components of the freight management system was performed and a platform for the deployment of the service was selected.

The scientific novelty is to improve the methods of communication between customers and private carriers through the development of a new web-application with the addition of an interactive map and the ability to automatically calculate the cost and distance of delivery.

HEROKU, GITHUB, TOMCAT, SERVLET, JAVA, JSP, JSTL, HTML, CSS, JS, MAPBOX, API, POSTGRESQL, AJAX, SCM.

ЗМІСТ

ВСТУП

Актуальність дослідження.

Ринок IT дуже великий — сюди входять і постачальники обладнання (серверів, персональних комп'ютерів, ноутбуків, мобільних пристроїв), і виробники різноманітного програмного забезпечення. При цьому, якщо ринок персональних комп'ютерів в обсязі скорочується, то ринок мобільних пристроїв росте. А ринок програмного забезпечення має прямий взаємозв'язок з ринком обладнання, відповідно, десктоп-додатки вже давно не користуються попитом, а ось ринок мобільних додатків і хмарних сервісів набирає все більших і більших обертів.

1991 рік. Інженер Тім Бернс-Лі створив перший у світі веб-сайт. Таку собі примітивну інформаційну сторінку…аби показати, який вигляд має мова розмітки HTML.

Сучасний світ дає можливість створювати інформаційні системи на стикі технологій, коли програма не тільки працює у віртуальному вимірі а й надає значного впливу на реальне середовище, керує процесами, людьми тощо.

Розробка програмного забезпечення, описаного в кваліфікаційній роботі, є яскравим прикладом такої інформаційної системи, тому що вона безпосередньо допомагає людям виконувати реальні дії за допомогою інформаційної системи, яка здатна вести облік, керування, планування процесів, які будуть виконані безпосередньо в житті.

Предмет дослідження – web-додаток для керування вантажоперевезення

Мета роботи – вдосконалити методи комунікації клієнтів та приватних вантажоперевізників.

Новизна дослідження полягає у вдосконаленні методів комунікації клієнтів та приватних вантажоперевізників через розробку нового web-додатка з додаванням інтерактивної карти та можливістю автоматичного прорахунку вартості та відстані доставки.

1 АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧІ

Які проблеми логістики найбільш типові та болючі для українських підприємств? Вирішення яких проблем логістики здатне принести їм найбільш відчутну користь?

Ось що про це гадають експерти.

1.1. Мінімізація витрат

Однією з найпоширеніших проблем є відсутність кваліфікованих кадрів, особливо в галузі організації сучасних логістичних процесів і технологій.

Ще одна гостра проблема — використання застарілих, а точніше, незастосування сучасних технологій, таких як штрихове кодування, режим реального часу, хвильова обробка, спеціалізовані автоматизовані системи обробки товарів. Адже ринку сьогодні потрібні високоякісні послуги, що надаються на більших обсягах логістичних операцій.

Серйозно стримує розвиток галузі слабка логістична інфраструктура на рівні підприємств, регіонів та країни загалом. На рівні підприємств передусім потрібна мінімізація логістичних витрат. Для цього потрібна ефективна система управління та моніторингу логістик підприємства, впровадження сучасних ІТ для простеження руху продукції та вантажів, підвищення продуктивності. Необхідно також знижувати транспортні витрати за рахунок підвищення якості доріг, створити дешеву, швидку та надійну транспортну інфраструктуру.

1.2. «Дитяча хвороба» керівництва

Багато керівників не має чіткого уявлення у тому, яке місце займає логістика у структурі компанії. Наприклад, важко пояснити виробникам, що

потрібно кожну одиницю товару забезпечити етикеткою із зазначенням його коду (бажано унікального — скажімо штрих-коду). Відсутність етикетки призводить до проблеми ідентифікації, що перетворює роботу складу на жах. Ще одна проблема: у компанії, що виросла з малого бізнесу, часто всі займаються всім, і знайти відповідальних неможливо, тобто відсутня структуризація, за якої певні відділи несуть сувору відповідальність за виконання вузького кола завдань або навіть окремих процесів. Нерідко для підприємства взагалі немає деталізованої системи обліку витрат за логістику (транспортні, складські витрати та ін.).

Часто відсутнє розуміння, навіщо підприємству потрібне дороге програмне забезпечення для керування складом. Є і ціле коло проблем, пов'язаних безпосередньо з вибором та впровадженням інформаційної системи, оскільки слабо розвинений ринок відповідного ПЗ, та й кількість компаній-інтеграторів невелика; нерідкі спроби використовувати ПЗ, призначене для одних цілей, при вирішенні інших завдань; практично не використовується ПЗ для керування транспортною складовою. Окрема проблема — некоректні дії впровадження, які, не вивчивши як слід стан бізнес-процесу, намагаються нав'язати свою систему. Основною ж проблемою українських підприємств у галузі логістики залишається відсутність знань. Тому головне завдання — знайти гідних «вихователів», які терпляче поведуть її шляхом конструктивного розвитку, спираючись на кращий світовий досвід.

1.3. Оптимізація та автоматизація

Найбільш типові для України проблеми логістики — відсутність раціонального управління ланцюжками поставок, нерозвинена транспортна мережа, відсутність мережі термінально-логістичних комплексів, недостатній рівень технічного забезпечення, відсутність знань у сфері підвищення ефективності управління складами і складськими комплексами. Найбільш відчутну користь підприємствам здатні принести оптимізація

ланцюжків постачання та автоматизація складських комплексів за допомогою систем управління складами[1].

1.4. Приватні вантажоперевезення

Існує багато компаній, які надають послуги з вантажних перевезень, але зручного та прозорого сервісу, в якому клієнт міг би чітко розуміти яким маршрутом буде здійснено доставку, немає.

1.5. Принципи розробки ПЗ

Незалежно від розроблюваного додатку слід притримуватися наступних принципів:

1.5.1. Стандартизація процесу розробки. CI/CD

Без стандартизованої системи для розробників кінцевим результатом може бути більш трудомісткі збірки та неефективне обслуговування. Ось чому необхідно встановити послідовний, передбачуваний підхід, якого слідувати розробникам, відповідно до ваших загальних бізнес-цілей і пріоритетів. Цей метод має ряд переваг, зокрема:

− Узгодженість між попередніми та поточними розробниками та командами;

− Стандарти, які керують прийняттям рішень, коли існує безліч альтернатив;

− Швидший розвиток завдяки повторному використанню раніше розгорнутих компонентів;

− Підвищена ефективність від використання одних і тих же технологій у кількох проектах.

Можна вибрати загальну архітектуру веб-розробки та мову програмування для використання з усіма своїми проектами або визначити найбільш підходящий набір інструментів для кожного типу програми. Обидва підходи сприяють узгодженості та дозволяють новим командам розробників легше розпочинати роботу, а розробникам — плавніше переходити від однієї команди до іншої.

Гарним прикладом цього принципу служить **CI/CD**. CI/CD (Continuous Integration/Continuous Delivery) — методологія розробки програмного коду, що забезпечує надійність та швидкість створення продукту. Належить до однієї з типових DevOps-практик.

Її основна відмінність у тому, що тестування та збірка коду проводиться в автоматичному режимі. Такий підхід дозволяє уникнути помилок на ранніх етапах та зводить ризики до мінімуму.

1.5.1.1 Принципи роботи

CI (Continuous Integration) – безперервна інтеграція. Під час написання коду розробники постійно вносять зміни, що підвантажуються до репозиторію. Для автоматичного тестування та перевірки використовують спеціалізовані сервіси (наприклад, Github), що створюють скрипти. Ведеться балка, в якій запротоколовані всі модифікації.

CD (Continuous Delivery) – безперервна доставка. Відповідає за автоматичне розгортання збірки у будь-якому оточенні: продакшн, середовище тестування чи розробки. Наприклад, після редагування коду він автоматично вміщується в область тестування.

Методологія використовується в компаніях, які під час розробки часто модифікують код, але випускають стабільні релізи. Розробники отримують автоматичний процес тестування та розгортання програми, що дозволяє зосередитись на постійному покращенні ПЗ.

1.5.1.2 Етапи методології

Цикл CI/CD умовно розбивають на 7 етапів.

− Створення. Програмісти пишуть певну частину коду, а потім тестують у ручному режимі. Якщо тест пройдено успішно, частини об'єднуються в єдиний код і переносяться в робочу гілку розробки.

− Збірка. На другому етапі використовують спеціальне програмне забезпечення для контролю версії продукту (наприклад, Git). Програма запускає автоматичне складання з внесеними змінами, а потім тестують

отриманий код. Критерії успішного проходження налаштовуються у ручному режимі.

− Ручний тест. Після проходження автотесту у програмному забезпеченні продукт вирушає на перевірку до команди тестувальників. Вони вручну перевіряють написаний код та шукають баги.

− Реліз. Після успішного закінчення третього етапу код надходить у реліз. Створюється робоча збірка для оновлення поточного програмного продукту до актуальної версії.

− Розгортання. Отримане оновлення публікується на офіційних серверах розробників для оновлення програмного забезпечення до останньої версії.

− Підтримка та моніторинг. Розробники відстежують відгуки клієнтів та готують список майбутніх змін.

− Планування. На останньому етапі створюється затверджений перелік змін для наступних версій. Після закінчення цього кроку цикл розробки закінчується і починається наново.

За такого принципу роботи користувачі одержують повністю робочий реліз. Якщо вони виявлять помилку або баг, то повідомляють техпідтримку.

Методологія CI/CD скорочує час розробки програмного продукту, виявляє помилки та недоліки на ранніх етапах створення коду, і навіть зменшує час відновлення у разі створення інциденту.

Зміни протоколюються, що прискорює процес відстеження помилок та додавання нових функцій до програмного продукту.[3]

1.5.2. Більше – не обов’язково краще.

Завдяки постійному використанню мобільних додатків споживачі чекають такої ж чистої простоти в Інтернеті. Щоб задовольнити ці очікування, розробка веб-додатків повинна збалансувати основні функції з простотою та легкістю використання. Заохочуйте розробників спростити

ваш інтерфейс користувача (UI) — тобто все, що користувач бачить і з чим взаємодіє.

Ваші веб-програми повинні давати користувачам відчуття знайомства за допомогою послідовних кольорів, шрифтів, графіки та макетів. Цей підхід може допомогти існуючим користувачам легше звертатися до нових програм, оскільки вони вже досягли рівня комфорту з наявними.

Загальний вигляд має бути чистим, простим і зрозумілим, а не напруженим або захараценим. Однак це не означає, що вони повинні бути простими або нудними. Розробники все ще можуть використовувати креативні ілюстрації, зображення, графіку та анімацію, щоб донести ваше повідомлення зі стилем.

З іншого ж боку, хоча простий дизайн важливий, він не може компенсувати поганий вміст. Ваші програми також мають бути інформативними, розважальними та цікавими з акцентом на очевидні переваги. Використовуйте мову, яка підходить для вашої цільової аудиторії та визначає пріоритети того, як ваші продукти чи послуги приносять цінність і відповідають потребам.

1.5.3. Зручність використання є ключовим.

Одним із ключів до успішного веб-додатка є надання позитивного користувальницького досвіду (UX), тобто те, як користувач відчуває до програми під час роботи з нею. Ця концепція може включати, наскільки простий у використанні (навчання), наскільки швидко користувачі можуть виконувати дії (ефективність), наскільки легко запам’ятати, як використовувати (запам’ятовування), і наскільки добре воно забезпечує очікувану функціональність (корисність).

На початку процесу розробки веб-додатків розробники повинні поговорити з потенційними користувачами, щоб дізнатися про їхні перспективи та цілі. Розробники можуть продовжувати використовувати цей процес протягом усього проекту, створюючи прототипи, дозволяючи користувачам працювати з ними, і вимагаючи зворотного зв’язку, а потім

використовуючи ці дані, щоб виділити слабкі елементи, виправити та покращити.

Розробники можуть використовувати такі можливості та функції, які сприяють зручності використання:

– Посилання: ви можете включити вказівники на інші сторінки, де доступна більш детальна інформація, але переконайтеся, що для цього потрібно лише один крок.

– Можливість спільного доступу. Дозволяє користувачам ділитися екранами з програми, копіюючи URL-адресу та надсилаючи її через загальні канали зв’язку.

– Можливість пошуку: полегшує сканування програми під час пошуку вмісту, пунктів меню та налаштувань. Додайте вікно пошуку до кожної сторінки.

– Швидка доставка: оскільки веб-програми вимагають даних для відображення вмісту, вони повинні використовувати завантажувач під час першого відвідування сторінки. Але під час наступних відвідувань користувачі повинні відразу побачити кешовану інформацію. Розробники можуть використовувати бібліотеки для зберігання даних, коли користувачі переходять з однієї сторінки на іншу.

– Навігація. Кожне веб-додаток має працювати плавно та швидко. Оптимізуйте елементи програми, щоб забезпечити швидкий доступ до всіх розділів.

1.5.4. Безпека – головний пріоритет.

Коли розробники вимагають якнайшвидшого надання рішень веб-додатків, вони можуть пожертвувати безпекою. Намагаючись прискорити процес, вони можуть порушити правильну безпеку, що може призвести до серйозних проблем і витрат у разі порушення програми.

Не зважаючи на те, що розробники мають працювали ефективно та швидко, кінцевий продукт має бути безпечним. Можливі наступні кроки, щоб підтримувати безпеку програми як пріоритет:

1. Постійне навчання з безпеки для внутрішніх розробників.
2. Налаштування системи винагород, щоб стимулювати першокласні функції безпеки додатків.
3. Переконатися, що нові розробники отримують повну інформацію про програми, які вони беруть.
4. Зверніться до досвідченої компанії-розробника з перевіреним послужним списком безпеки.

1.5.5. Забезпечення майбутнього.

В ідеалі веб-додатки повинні відповідатимуть як поточним, так і майбутнім потребам, але ніхто точно не знає, які тенденції будуть наступними, що потрібно буде робити майбутнім додаткам, які нові стандарти та вимоги з’являться або скільки користувачів буде активним у програмі через п'ять років. Щоб вони були надійними, необхідно зробити такі кроки:

− Розділення на яруси. Необхідно вибрати архітектуру розробки, яка розділяє програми на рівні, щоб ви могли підтримувати або покращувати кожен рівень окремо. Завдяки такому підходу ви можете за потреби оновлювати один рівень за раз, не змінюючи всю програму.

− Побудова для зростання. Необхідно переглянути свої початкові мінімальні вимоги і створіть свої програми для роботи з більшим обсягом, якщо в кінцевому підсумку у вас буде більше користувачів, ніж очікувалося. Один сервер може не впоратися з майбутніми потребами, тому розробляйте програми для роботи на кластері серверів.

− Увімкнути інтеграцію. Необхідно створюйти програми, які можуть інтегруватися та добре працювати з іншими типами програм. Ця дія дає вам можливість використовувати комбінацію програм на основі підписки,

внутрішніх і хмарних програм.

− Розробка веб-додатків відіграє важливу роль у бізнесі. Якщо будуть зроблені стратегічні кроки для стандартизації, спрощення та надійності веб-додатків, вони будуть успішними для вашого бізнесу сьогодні та в майбутньому[2].

1.6. Архітектура додатків

1.6.1. Огляд

Архітектура програми описує шаблони та методи, які використовуються для проектування та створення програми. Архітектура дає вам дорожню карту та найкращі методи, яких слід дотримуватися під час створення програми, щоб у підсумку ви отримали добре структурований додаток.

Шаблони проектування програмного забезпечення можуть допомогти вам створити додаток. Шаблон описує повторюване рішення проблеми.

Шаблони можуть бути пов’язані разом для створення більш загальних архітектур додатків. Замість того, щоб повністю створювати архітектуру самостійно, ви можете використовувати існуючі шаблони дизайну, які також гарантують, що все працюватиме так, як має бути.

Як частина архітектури додатка, існуватимуть як зовнішні, так і внутрішні служби. Фронт-енд розробка пов’язана з користувацьким досвідом програми, тоді як розробка серверної частини зосереджена на наданні доступу до даних, служб та інших існуючих систем, які забезпечують роботу програми.

Архітектура є відправною точкою або дорожньою картою для створення програми, але вам потрібно буде зробити вибір реалізації, який не враховано в архітектурі. Наприклад, першим кроком є вибір мови програмування, на якій буде написано програму.

Для розробки програмного забезпечення використовується багато мов програмування. Деякі мови можуть використовуватися для створення певних

типів програм, наприклад Swift для мобільних додатків або JavaScript для розробки інтерфейсу.

JavaScript, який використовується з HTML і CSS, наразі є однією з найпопулярніших мов програмування для розробки веб-додатків.

Інші популярні мови програмування включають Ruby, Python, Swift, TypeScript, Java, PHP та SQL та інші. Мова, яка використовується під час створення програми, буде залежати від типу програми, доступних ресурсів розробки та вимог.

Історично програми були написані як єдина одиниця коду, де всі компоненти мають однакові ресурси та місце в пам'яті. Цей стиль архітектури називають монолітним.

Сучасні архітектури додатків частіше слабо пов'язані, використовуючи мікросервіси та інтерфейси програмного забезпечення (API) для підключення служб, які забезпечують основу для хмарних додатків.

Хмарна розробка — це спосіб прискорити створення нових додатків, оптимізувати існуючі та забезпечити послідовну розробку й автоматизоване керування в приватних, загальнодоступних та гібридних хмарах.

1.6.2. Вибір архітектури програми

Вирішуючи, яку архітектуру програми використовувати для нової програми, або оцінюючи поточну архітектуру, необхідно почати з визначення своїх стратегічних цілей.

Тоді ви можна розробити архітектуру, яка відповідає цілям, замість того, щоб спочатку вибирати архітектуру та намагатися вписати програму в цю структуру.

Також важливо враховувати, як часто будуть випускатися оновлення, щоб задовольнити потреби клієнтів або операцій, а також яку функціональність вимагають або бізнес-цілі, або потреби розвитку.

Здатність швидко надавати нові послуги та нову функціональність клієнтам є одним із ключових конкурентних відмінностей, які може запропонувати компанія. А швидший розвиток дозволяє компаніям частіше

випускати нові функції та впроваджувати оновлення, щойно буде виявлено вразливість.

Існує багато різних типів архітектур додатків, але найпомітнішими сьогодні, заснованими на зв’язках між службами, є: моноліти і N-рівнева архітектура (тісно пов’язані), мікросервіси (відокремлені), а також архітектура, керована подіями, і архітектура, орієнтована на сервіси. (слабко зв'язані).

1.6.3. Багатошарова або N-рівнева архітектура

Багаторівнева або N-рівнева архітектура – це традиційна архітектура, яка часто використовується для створення локальних і корпоративних додатків, і часто асоціюється із застарілими програмами.

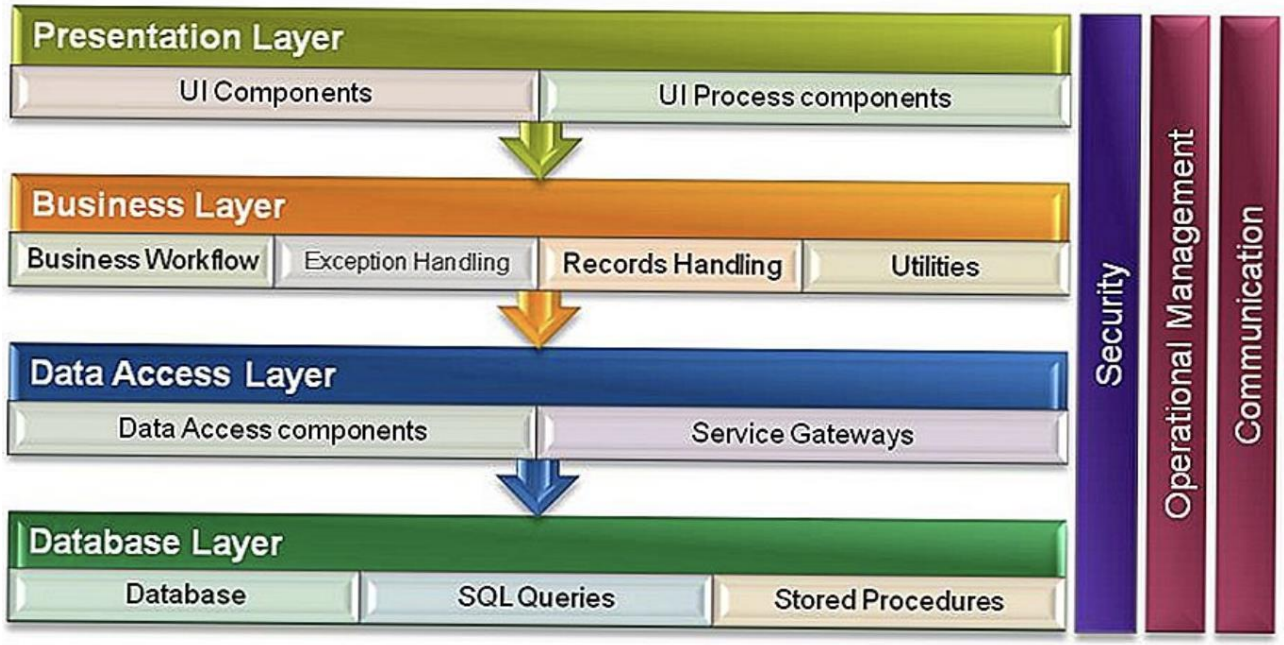


Рис. 1.1. N-рівнева архітектура

У багатошаровій архітектурі є кілька шарів або рівнів, часто 3, але їх може бути більше, що складають програму, кожен з яких несе свою відповідальність.

Шари допомагають керувати залежностями та виконувати логічні функції. У багатошаровій архітектурі шари розташовані горизонтально, тому вони можуть викликати лише нижній шар.

Шар може викликати лише шар безпосередньо під ним, або він може викликати будь-який із шарів під ним.

1.6.4. Монолітна архітектура

Моноліт, інший тип архітектури, пов’язаний із застарілими системами — це один стек додатків, який містить усі функції цієї 1 програми. Це тісно пов’язане як у взаємодії між послугами, так і в тому, як вони розробляються та надаються.

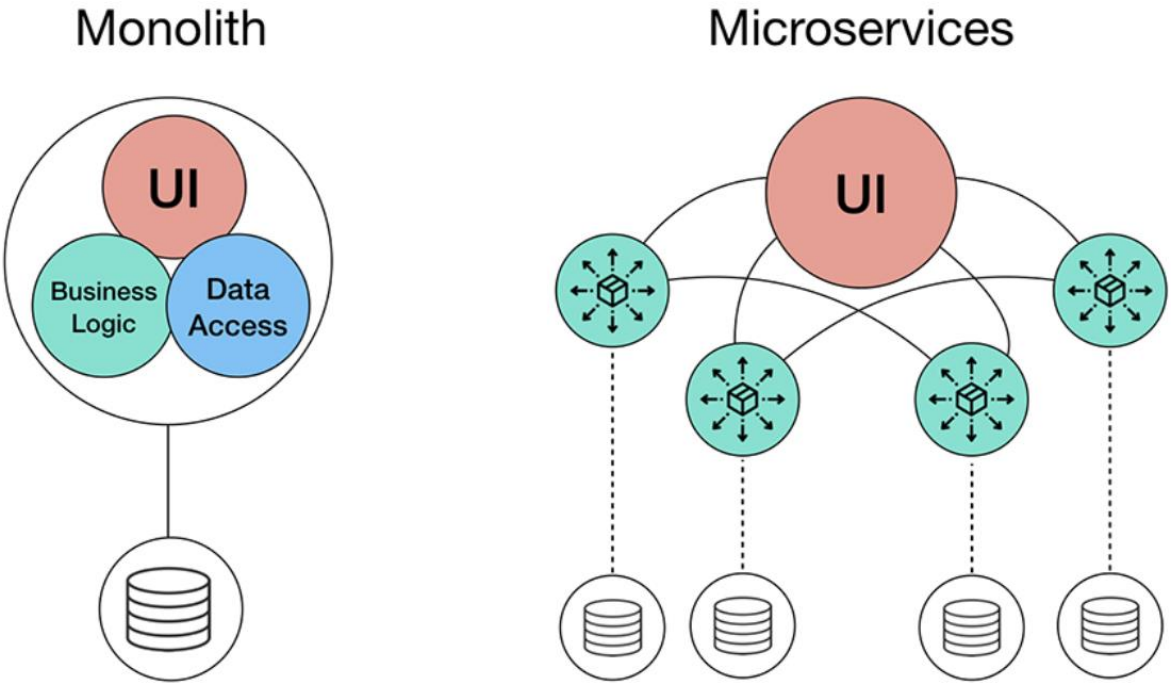


Рис. 1.2. Монолітна архітектура vs мікросервісна

Оновлення або масштабування окремого аспекту монолітної програми має наслідки для всієї програми та її базової інфраструктури.

Одна зміна коду програми вимагає повторного випуску всієї програми. У зв’язку з цим оновлення та нові випуски зазвичай відбуваються лише один-два рази на рік і можуть включати лише загальне обслуговування замість нових функцій.

На відміну від цього, більш сучасні архітектури намагаються виділити послуги за функціональністю або бізнес-можливостями, щоб забезпечити більшу гнучкість.

1.6.5. Мікросервісна архітектура

Мікросервіси — це і архітектура, і підхід до написання програмного забезпечення. За допомогою мікросервісів програми розбиваються на найдрібніші компоненти, незалежні один від одного. Кожен з цих компонентів або процесів є мікросервісом.

Мікросервіси розподілені та слабо пов’язані, тому вони не впливають один на одного. Це має переваги як для динамічної масштабованості, так і для відмовостійкості: окремі послуги можна масштабувати за потреби, не вимагаючи важкої інфраструктури, або можна перемикатися, не впливаючи на інші служби.

Метою використання архітектури мікросервісів є швидша доставка якісного програмного забезпечення. Ви можете розробляти кілька мікросервісів одночасно. А оскільки служби розгортаються незалежно, вам не потрібно перебудовувати або повторно розгортати весь додаток під час внесення змін.

Це призводить до того, що більше розробників працюють над своїми окремими службами одночасно, замість того, щоб оновлювати весь додаток, що призводить до зменшення часу на розробку та можливості випускати нові функції частіше.

Поряд з API та командами DevOps, контейнерні мікросервіси є основою для хмарних додатків.

1.6.6. Архітектура, орієнтована на події

З системою, керованою подіями, фіксація, передача, обробка та збереження подій є основною структурою рішення. Це відрізняється від традиційної моделі на основі запитів.

Подія — це будь-яка значуща подія або зміна стану апаратного чи програмного забезпечення системи. Джерелом події може бути внутрішні або зовнішні вхідні дані.

Архітектура, керована подіями, забезпечує мінімальне з’єднання, що робить її хорошим варіантом для сучасних архітектур розподілених додатків.

Архітектура, орієнтована на події, складається з виробників подій і споживачів подій. Виробник події виявляє або відчуває подію і представляє подію у вигляді повідомлення. Він не знає споживача події чи результатів події.

Після виявлення події вона передається від виробника події до споживачів події через канали подій, де платформа обробки подій обробляє подію асинхронно.

Архітектура, орієнтована на події, може бути заснована на моделі pub/sub або на моделі потоку подій.

Модель pub/sub заснована на підписках на потік подій. З цією моделлю після того, як подія відбувається або публікується, вона надсилається передплатникам, яких потрібно повідомити.

Це відрізняється від моделі потокової передачі подій, коли споживачі подій не підписуються на потік подій. Натомість вони можуть читати з будь-якої частини потоку та можуть приєднатися до потоку в будь-який час.

Події фіксуються в міру їх виникнення з джерел подій, таких як пристрої Інтернету речей (IoT), програми та мережі, що дозволяє виробникам подій і споживачам подій обмінюватися інформацією про статус та відповіді в режимі реального часу.

1.6.7. Сервісно-орієнтована архітектура

Сервісно-орієнтована архітектура (SOA) — це усталений стиль проектування програмного забезпечення, схожий на стиль архітектури мікросервісів.

SOA структурує програми на окремі, багаторазові послуги, які спілкуються через корпоративну службову шину (ESB).

У цій архітектурі окремі служби, кожна з яких організована навколо певного бізнес-процесу, дотримуються протоколу зв’язку (наприклад, SOAP, ActiveMQ або Apache Thrift) і відкривають себе через платформу ESB. У сукупності цей набір послуг, інтегрований через ESB, використовується

інтерфейсною програмою, щоб забезпечити цінність для бізнесу або клієнта[4].

1.7. Протоколи обміну інформації

REST проти SOAP. Це вже деякий час є проблемою. І насправді це лише дві відповіді на одне й те саме питання: як отримати доступ до веб-сервісів.

Але визначити одне над іншим може бути напрочуд важко.

SOAP (Simple Object Access Protocol) — це заснований на стандартах протокол доступу до веб-сервісів, який існує протягом тривалого часу. Спочатку розроблений Microsoft, SOAP не такий простий, як можна було б припустити.

REST (Representational State Transfer) — ще один стандарт, створений у відповідь на недоліки SOAP. Він прагне виправити проблеми з SOAP і забезпечити простіший метод доступу до веб-сервісів.

А як щодо GraphQL?

Звичайно, GraphQL нещодавно зробив величезний фурор. Але він все ще не такий стандартизований, як REST і SOAP, тому для цієї кваліфікаційної роботи наразі він не підходить.

І SOAP, і REST мають проблеми, які слід враховувати, вирішуючи, який протокол використовувати.

1.7.1. Подібність

Хоча SOAP і REST мають схожість щодо протоколу HTTP, SOAP є більш жорстким набором шаблонів обміну повідомленнями, ніж REST. Правила в SOAP важливі, тому що без них ми не можемо досягти жодного рівня стандартизації. REST як стиль архітектури не вимагає обробки і, природно, є більш гнучким. І SOAP, і REST покладаються на добре встановлені правила, яких усі погодилися дотримуватися в інтересах обміну інформацією.

1.7.2. Огляд SOAP

SOAP покладається виключно на XML для надання послуг обміну повідомленнями. Спочатку Microsoft розробила SOAP, щоб замінити старі технології, які погано працюють в Інтернеті, такі як модель розподіленого компонента (DCOM) і архітектура посередника загального об'єктного запиту (CORBA). Ці технології зазнають невдачі, оскільки вони покладаються на двійкові повідомлення. Повідомлення XML, які використовує SOAP, краще працюють в Інтернеті.

Після початкового випуску Microsoft представила SOAP до Інженерної групи з розробки Інтернету (IETF), де він був стандартизований. SOAP розроблено для підтримки розширення, тому він має всілякі інші абревіатури та абревіатури, пов'язані з ним, такі як WS-Addressing, WS-Policy, WS-Security, WS-Federation, WS-ReliableMessaging, WS-Coordination, WS-AtomicTransaction , і WS-RemotePortlets. Насправді, можна знайти цілий список цих стандартів у стандартах веб-служб.

Справа в тому, що SOAP дуже розширюваний, але ви використовуєте лише ті частини, які вам потрібні для певного завдання. Наприклад, коли користуєтеся загальнодоступною веб-службою, яка є безкоштовно доступною для всіх, вам дійсно не потрібна WS-Security.

1.7.2.1 Складність залежить від мови програмування

XML, який використовується для запитів і отримання відповідей у SOAP, може стати надзвичайно складним. У деяких мовах програмування вам потрібно створювати ці запити вручну, що стає проблематичним, оскільки SOAP не терпить помилок. Однак інші мови можуть використовувати ярлики, які надає SOAP. Вони можуть допомогти вам зменшити зусилля, необхідні для створення запиту та аналізу відповіді. Насправді, працюючи з мовами .NET, ви навіть не бачите XML.

Частиною магії є мова опису веб-сервісів (WSDL). Це ще один файл, пов'язаний із SOAP. Він дає визначення того, як працює веб-сервіс, так що, коли ви створюєте посилання на нього, IDE може повністю автоматизувати

процес. Отже, складність використання SOAP значною мірою залежить від мови, яку ви використовуєте.

1.7.2.2 Вбудована обробка помилок

Однією з найважливіших функцій SOAP є вбудована обробка помилок. Якщо з вашим запитом виникла проблема, відповідь містить інформацію про помилку, яку можна використати для вирішення проблеми. З огляду на те, що ви можете не володіти веб-сервісом, ця особливість надзвичайно важлива; інакше ви залишилися б здогадуватися, чому все не працює. Звіт про помилки навіть надає стандартизовані коди, щоб можна було автоматизувати деякі завдання обробки помилок у вашому коді.

Цікавою функцією SOAP є те, що вам не обов'язково використовувати її з транспортом HTTP. Існує фактична специфікація використання SOAP за протоколом простї передачі пошти (SMTP), і немає причин, чому ви не можете використовувати його для інших транспортних засобів. Насправді, розробники деяких мов, таких як Python і PHP, роблять саме це.

1.7.3. Огляд REST

REST забезпечує більш легку альтернативу. Багато розробників знайшли SOAP громіздким і важким у використанні. Наприклад, робота з SOAP в JavaScript означає написання тонни коду для виконання простих завдань, оскільки ви повинні щоразу створювати необхідну структуру XML.

Замість використання XML для запиту, REST (зазвичай) покладається на просту URL-адресу. У деяких ситуаціях потрібно надати додаткову інформацію, але більшість веб-сервісів, які використовують REST, покладаються виключно на використання URL-підходу. REST може використовувати чотири різні дієслова HTTP 1.1 (GET, POST, PUT і DELETE) для виконання завдань.

На відміну від SOAP, REST не повинен використовувати XML для надання відповіді. Ви можете знайти веб-сервіси на основі REST, які виводять дані у значенні, розділеному командами (CSV), нотації об'єктів JavaScript (JSON) і дійсно простому поширенні (RSS). Справа в тому, що ви

можете отримати потрібний результат у формі, яку легко проаналізувати мовою, яку ви використовуєте для своєї програми.

1.7.4. Вибір між SOAP і REST

Якщо ви не плануєте створити власну веб-службу, рішення про те, який протокол використовувати, можливо, вже прийнято за вас. Вкрай мало веб-сервісів, таких як Amazon, підтримують обидва. Ваші рішення часто зосереджуються на тому, який веб-сервіс найкраще відповідає вашим потребам, а не на тому, який протокол використовувати.

Переваги SOAP

SOAP надає такі переваги в порівнянні з REST:

− Незалежність від мови, платформи та транспорту (REST вимагає використання HTTP)

− Добре працює в розподілених корпоративних середовищах (REST передбачає прямий зв'язок «точка-точка»)

− Стандартизовані

− Забезпечує значну розширюваність перед збіркою у формі стандартів WS*

− Вбудована обробка помилок

− Автоматизація при використанні з певними мовними продуктами

Переваги REST

REST здебільшого легший у використанні та більш гнучкий. Він має такі переваги перед SOAP:

− Для взаємодії з веб-сервісом не потрібні дорогі інструменти.

− Менша крива навчання.

− Ефективний (SOAP використовує XML для всіх повідомлень, REST може використовувати менші формати повідомлень).

− Швидкий (не вимагає тривалої обробки).

− Ближче до інших веб-технологій у філософії дизайну[5].

1.8. Що таке API?

API — це абревіатура від Application Programming Interface, який є програмним посередником, який дозволяє двом додаткам спілкуватися один з одним. Кожного разу, коли ми використовуємо програму, як-от Facebook, надсилаємо миттєве повідомлення або перевіряєте погоду на своєму телефоні, ми використовуємо API. Іншими словами — це те, за допомогою чого ми можемо взаємодіяти з тою чи іншою програмою.

1.9. Геодані

Для додаткового функціоналу та навігації вантажоперевезень на сайті має бути розташована область з картою, яка буде давати можливість створювати перевезення та бачити інтерактивно за яким маршрутом буде виконана доставка. Це і можливо зробити за допомогою API програм, які можуть надати данні за допомогою яких стає можливим використання карт на сайті. До прикладу можна навести такі сервіси: Google Maps API, MapQuest, Mapbox, ESRI та ін. Для задоволення потреб проекту перш за все необхідно зупинитися на одному, найбільш зручному сервісі. За допомогою аналізу буде вибрано найбільш зручний сервіс.

1.10. Висновки розділу

− Була розглянута тема проекту, занурення у тематику та специфіку вантажоперевезень, проблеми, які наразі присутні в цій сфері.

− Розглянуті популярні протоколи обміну інформації, SOAP та REST, їх плюси та мінуси. Перевагу віддано REST, тому що його простота менша кількість обмежень більше підходить для реалізації кваліфікаційної роботи.

− Виконано огляд на можливі архітектурні підходи для реалізації кваліфікаційної роботи.

− Короткий аналіз можливих постачальників API для картографічних даних. Для виконання картографічних цілей проекту обрано використання API MapBox, адже він має найбільш доступну та зрозумілу документацію, зрозумілий користувацький інтерфейс, зручний моніторинг користування API, достатній безкоштовний період користування для тестування MVP

проекту.

− Визначені головні пріоритети при розробці додатка

2. МОДЕЛІ ТА МЕТОДИ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1. Використані технології

Для виконання кваліфікаційної роботи було вибрано Java як основну серверну мову програмування. Для реалізації фронтенду було обрано зв'язку HTML, CSS, JS та Bootstrap для пришвидшення верстки. Також для спрощення деяких функцій на клієнтській стороні буде використано JQuery. Зокрема асинхронний доступ до серверних API за допомогою AJAX технології. В якості персистентного сховища вибрано PostrgreSQL.

2.1.1. Java

Java — суворо типізована об'єктно-орієнтована мова програмування загального призначення, розроблена компанією Sun Microsystems (надалі придбаною компанією Oracle). Розробка ведеться спільнотою, організованою

через Java Community Process; мова та основні технології, що реалізують її, поширюються за ліцензією GPL. Права на торгову марку належать корпорації Oracle.

Програми Java зазвичай транслюються у спеціальний байт-код, тому вони можуть працювати на будь-якій комп'ютерній архітектурі, для якої існує реалізація віртуальної Java-машини. Дата офіційного випуску – 23 травня 1995 року. Займає високі місця у рейтингах популярності мов програмування (2-е місце у рейтингах IEEE Spectrum (2020) та TIOBE (2021)) [6].

2.1.2. HTML

HTML (від англ. HyperText Markup Language — "мова гіпертекстової розмітки") — стандартизована мова розмітки документів для перегляду веб-сторінок у браузері. Веб-браузери отримують HTML документ від сервера за протоколами HTTP/HTTPS або відкривають з локального диска, далі інтерпретують код в інтерфейс, який відображатиметься на екрані монітора.

Елементи HTML є будівельними блоками сторінок HTML. За допомогою HTML різні конструкції, зображення та інші об'єкти такі як інтерактивна веб-форми можуть бути вбудовані в сторінку, що відображається. HTML надає засоби для створення заголовків, абзаців, списків, посилань, цитат та інших елементів. Елементи HTML виділяються тегами, записаними з використанням кутових дужок. Такі теги, як <img/> та <input/>, безпосередньо вводять контент на сторінку. Інші теги, такі як <p>, оточують і оформляють текст у собі і можуть включати інші теги як поделементів. Браузери не відображають HTML-теги, але використовують їх для інтерпретації вмісту сторінки.

Мова XHTML є суворішим варіантом HTML, він слідує синтаксису XML і є додатком мови XML в області розмітки гіпертексту.

У HTML можна вбудувати програмний код мовою програмування JavaScript, для управління поведінкою та змістом веб-сторінок. Також включення CSS в HTML описує зовнішній вигляд та макет сторінки[7].

2.1.3. CSS

CSS (/siːɛsɛs/ англ. Cascading Style Sheets «каскадні таблиці стилів») — офіційна мова, описана зовнішнього вида документа (веб-страницы), написаного з використанням мови розметки (чаще всього HTML або XHTML). Також може застосовуватися будь-який XML-документ, наприклад, SVG або XUL.

CSS використовується для створення веб-сторінок для завдання кольорів, шрифтів, стилів, розташування окремих блоків та інших аспектів представлення зовнішнього вигляду цих веб-сторінок. Основна ціль розробки CSS є відділенням опису логічної структури веб-страниці (що виробляється за допомогою HTML або інших мовних розділів) від опису зовнішнього виду цієї веб-страниці (що тепер виробляється за допомогою формальної мови CSS). Таке розділення може збільшити доступність документа, забезпечити велику гнучкість і можливість керування його представленням, а також зменшити складність і повторність у структурному вмісті.

Крім того, CSS дозволяє відображати один і той самий документ у різних стилях чи методах виводу, таких, як екранне представлення, друковане представлення, читання голосом (спеціальним голосовим браузером або програмою читання з екраном) або при виводі пристроїв, які використовують шрифт Брайля[8].

2.1.4. JS

JavaScript (/ˈdʒɑːvɑːˌskrɪpt/; аббр. JS /ˈdʒeɪ.ɛs./) — мультипарадигменна мова програмування. Підтримує об'єктно-орієнтований, імперативний та функціональний стилі. Є реалізацією специфікації ECMAScript (стандарт ECMA-262).

JavaScript зазвичай використовується як вбудована мова для програмного доступу до об'єктів програм. Найширше застосування знаходить у браузерах як мову сценаріїв надання інтерактивності веб-страницам.

Основні архітектурні риси: динамічна типізація, слабка типізація, автоматичне керування пам'яттю, прототипне програмування, функції як об'єкти першого класу.

На JavaScript вплинули багато мов, при розробці була мета зробити мову схожою на Java. Мовою JavaScript не володіє будь-яка компанія або організація, що відрізняє його від низки мов програмування, що використовуються у веб-розробці.

Назва «JavaScript» є зареєстрованим товарним знаком корпорації Oracle США.

У 1992 році компанія Nombas (згодом придбана Openwave[en]) почала розробку вбудовуваної скриптової мови Cmm (Сі-мінус-мінус), яка, за задумом розробників, повинна була стати досить потужною, щоб замінити макроси, зберігаючи при цьому схожість із Сі, щоб розробникам не важко було вивчити его. Головною відмінністю від Сі була робота з пам'яттю. У новій мові все управління пам'яттю здійснювалося автоматично: був необхідності створювати буфера, оголошувати змінні, здійснювати перетворення типів. В іншому мови сильно скидалися одна на одну: зокрема, Cmm підтримував стандартні функції та оператори Сі. Cmm був перейменований в ScriptEase, оскільки вихідна назва звучала надто негативно, а згадка в ньому Сі відлякувало людей. На основі цієї мови було створено пропрієтарний продукт CEnvi. Наприкінці листопада 1995 року Nombas розробила версію CEnvi, яка впроваджується у веб-сторінки. Сторінки, які можна було змінювати за допомогою скриптової мови, отримали назву Espresso Pages — вони демонстрували використання скриптової мови для створення гри, перевірки введення в форми і створення анімації. Espresso Pages позиціонувалися як демоверсія, покликана допомогти уявити, що станеться, якщо в браузер буде впроваджено мову Cmm. Працювали вони лише у 16-бітовому Netscape Navigator під керуванням Windows.

Найперша реалізація JavaScript була створена Бренданом Ейхом (Brendan Eich) у компанії Netscape, і з того часу оновлюється, щоб відповідати ECMA-262 Edition 5 та пізнішим версіям. Цей двигун називається SpiderMonkey і реалізований мовою C/C++. Двигун Rhino створений Норрісом Бойдом (Norris Boyd) і реалізований мовою Java. Як і SpiderMonkey, Rhino відповідає ECMA-262 Edition 5.

2.1.5. Bootstrap

Bootstrap — це елігантний, інтуїтивно зрозумілий і потужний mobile first front-end фреймворк для швидшої та простішої веб-розробки. Він використовує HTML, CSS і Javascript.

2.1.5.1 Переваги

− Mobile first підхід — фреймворк складається зі стилів mobile first у всій бібліотеці, а не в окремих файлах. Mobile First – це принцип, згідно з яким спочатку розробляється UI/UX інтерфейс сайту для розширень, які встановлюються на смартфони. В подальшому дизайн-макети масштабуються під планшетні та desktop-пристрої. Цей підхід дозволяє якісно опрацювати зручність мобільної версії сайту, а не за залишковим принципом, як при типовому підході, коли робота над макетами починається з desktop-версії[10].

− Підтримка браузерів — підтримується всіма популярними браузерами.

− Легко розпочати роботу — лише володіючи знаннями HTML і CSS, кожен може розпочати роботу з Bootstrap. Також на офіційному сайті Bootstrap є хороша документація.

− Адаптивний дизайн – адаптивний CSS Bootstrap налаштовується на настільні комп’ютери, планшети та мобільні пристрої.

− Забезпечує чисте й уніфіковане рішення для створення інтерфейсу для розробників.

− Він містить красиві та функціональні вбудовані компоненти, які легко налаштувати.

− Він також забезпечує веб-налаштування.

− І найкраще це відкритий вихідний код.

2.1.5.2 Пакети, з яких складається фреймворк

− Scaffolding – Bootstrap надає базову структуру із системою сіток, стилями посилань та фоном. Це докладно розглянуто в розділі Основна структура Bootstrap

− CSS – Bootstrap поставляється з функцією глобальних налаштувань CSS, основними елементами HTML, стилізованими та покращеними за допомогою розширюваних класів, а також розширеною системою сіток. Це докладно описано в розділі Bootstrap з CSS.

− Components – Bootstrap містить понад дюжину компонентів для багаторазового використання, створених для надання іконок, спадних меню, навігації, сповіщень, спливаючих вікон та багато іншого. Це детально описано в розділі Компоненти макета.

− Плагіни JavaScript – Bootstrap містить понад дюжину спеціальних плагінів jQuery. Ви можете легко включити їх усі або по одному. Це докладно описано в розділі Bootstrap Plugins.

− Customize — ви можете налаштувати компоненти Bootstrap, змінні LESS та плагіни jQuery, щоб отримати свою власну версію[9].

2.1.6. jQuery

jQuery — це швидка, невелика і багатофункціональна бібліотека JavaScript. Це значно спрощує такі речі, як обхід і маніпуляції з документами HTML, обробка подій, анімація та Ajax, завдяки простому у використанні API, який працює в багатьох браузерах. Завдяки поєднанню універсальності та розширюваності jQuery змінив спосіб написання JavaScript мільйонами людей[11].

2.1.7. AJAX

AJAX, Ajax (ˈeɪdʒæks, від англ. Asynchronous Javascript and XML — «асинхронний JavaScript і XML») — підхід до побудови інтерактивних

інтерфейсів користувача веб-додатків, що полягає в «фоновому» обміні даними браузера з веб-сервером. В результаті, при оновленні даних веб-сторінка не перезавантажується повністю, і веб-програми стають швидше і зручнішими.

AJAX – не самостійна технологія, а концепція використання кількох суміжних технологій. AJAX базується на двох основних принципах:

− використання технології динамічного звернення до сервера «на льоту», без перезавантаження всієї сторінки повністю, наприклад, з використанням XMLHttpRequest (основний об'єкт);

- через динамічне створення дочірніх фреймів;
- через динамічне створення тега <script>;
- через динамічне створення тега <img>, як це було реалізовано в Google Analytics.

− використання DHTML для динамічного зміни змісту сторінки;

Дії з інтерфейсом перетворюються на операції з елементами DOM (англ. Document Object Model), з допомогою яких обробляються дані, доступні користувачеві, у результаті представлення їх змінюється. Тут проводиться обробка переміщень і клацань мишею, і навіть натискань клавіш. Каскадні таблиці стилів, або CSS забезпечують узгоджений зовнішній вигляд елементів програми та спрощують звернення до DOM-об'єктів. Об'єкт XMLHttpRequest (або подібні механізми) використовується для асинхронної взаємодії з сервером, обробки запитів користувача та завантаження у процесі роботи необхідних даних.

Три з цих чотирьох технологій – CSS, DOM та JavaScript – складають DHTML (англ. Dynamic HTML). На думку деяких авторів, механізми DHTML, винайдені у 1997 року, подавали великі надії, але не виправдали їх.

Як формат передачі даних можуть використовуватися фрагменти простого тексту, HTML-коду, JSON або XML[26].

2.1.8. JSON

JSON (англ. JavaScript Object Notation) — текстовий формат обміну даними, заснований на JavaScript. Як і багато інших текстових форматів, JSON легко читається людьми. Формат JSON був розроблений Дугласом Крокфордом.

Незважаючи на походження від JavaScript (точне, від підмноження мови стандарту ECMA-262 1999 року), формат вважається незалежним від мови і може використовуватися практично з будь-яким мовою програмування. Для багатьох мов існує готовий код для створення та обробки даних у форматі JSON[12].

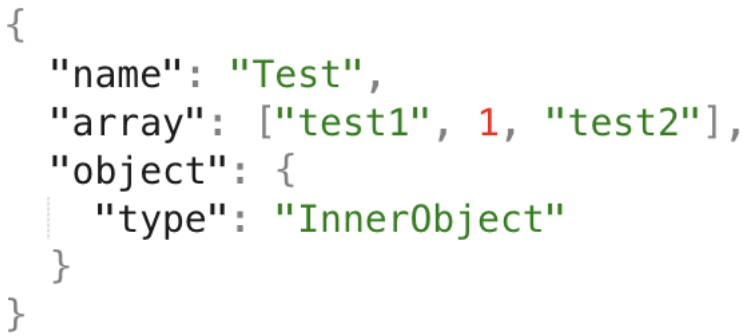


Рис. 2.1. Приклад формату JSON

2.1.9. PostgreSQL

PostgreSQL – це потужна об’єктно-реляційна система баз даних з відкритим вихідним кодом з більш ніж 30-річною активною розробкою, завдяки якій вона заслужила міцну репутацію за надійність, стійкість функцій і продуктивність.

PostgreSQL базується на мові SQL і підтримує багато можливостей стандарту SQL:2011.

У PostgreSQL версії 12 є такі обмеження:

Максимальний розмір бази даних	Немає обмежень
Максимальний розмір таблиці	32 Тбайт

Максимальний розмір поля	1 Гбайт
Максимум записів у таблиці	Обмежено розмірами таблиці
Максимум полів у записі	250-1600, залежно від типів полів
Максимум індексів у таблиці	Немає обмежень

Табл. 2.1. Характеристики СУБВ PostgreSQL

Сильними сторонами PostgreSQL вважаються:

− високопродуктивні та надійні механізми транзакцій та реплікації;

− система вбудованих мов програмування, що розширюється: у стандартній поставці підтримуються PL/pgSQL, PL/Perl, PL/Python і PL/Tcl; додатково можна використовувати PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Scheme, PL/sh та PL/V8, а також є підтримка завантаження модулів розширення мовою C[10];

− успадкування;

− можливість індексування геометричних (зокрема, географічних) об'єктів і наявність розширення PostGIS, що базується на ній;

− вбудована підтримка слабоструктурованих даних у форматі JSON з можливістю їхньої індексації;

− розширюваність (можливість створювати нові типи даних, типи індексів, мови програмування, модулі розширення, підключати будь-які зовнішні джерела даних).

2.1.10. Servlet API

Jakarta Servlet (раніше Java Servlet) — це програмний компонент Java, який розширює можливості сервера. Хоча сервлети можуть відповідати на багато типів запитів, вони найчастіше реалізують веб-контейнери для розміщення веб-додатків на веб-серверах і, таким чином, кваліфікуються як веб-API сервлетів на стороні сервера. Такі веб-сервлети є аналогом Java для інших технологій динамічного веб-контенту, таких як PHP і ASP.NET.

Сервлети підтримуються віртуальною машиною JVM, що запобігає витоку пам'яті та забезпечує функціонування garbage collection. Кожному клієнту сервлет виділяє незалежний потік виконання. Клієнт надсилає

додатку HTTP-запит, сервлет генерує відповідь та повертає його клієнту у вигляді html-документа.

Сервлет:

− компонент програм Java Enterprise Edition;

− Завантажується веб-сервером в контейнер;

− Виконується на стороні сервера;

− Опрацьовує клієнтські запити;

− динамічно генерує відповіді на запити;

− перебуває у стані очікування, якщо запити відсутні;

− приймає запити від інших сервлетів (Servlet Chaining);

− Підтримує з'єднання з ресурсами.

Найбільшого поширення набули сервлети, що обробляють клієнтські запити за протоколом HTTP. Технологія сервлетів є оболонкою протоколу HTTP та підтримує його як транспорт передачі даних від клієнта серверу та назад. Контейнер сервлетів підтримує також протокол HTTPS (HTTP і SSL) для запитів, що захищаються.

Сервлети у промисловому програмуванні використовуються для:

− прийому вхідних даних від клієнта;

− взаємодії з бізнес-логікою системи;

− динамічної генерації відповіді клієнту.

Усі сервлети реалізують спільний інтерфейс Servlet із пакету javax.servlet. Для обробки HTTP-запитів у web можна скористатися як базовий клас абстрактним класом HttpServlet з пакету javax.servlet.http.

Життєвий цикл сервлета починається з його ініціалізації та завантаження в пам'ять контейнером сервлетів при старті контейнера або у відповідь на перший запит клієнта. Сервлет готовий до обслуговування будь-якої кількості запитів. Завершення існування відбувається при розвантаженні його з контейнера.

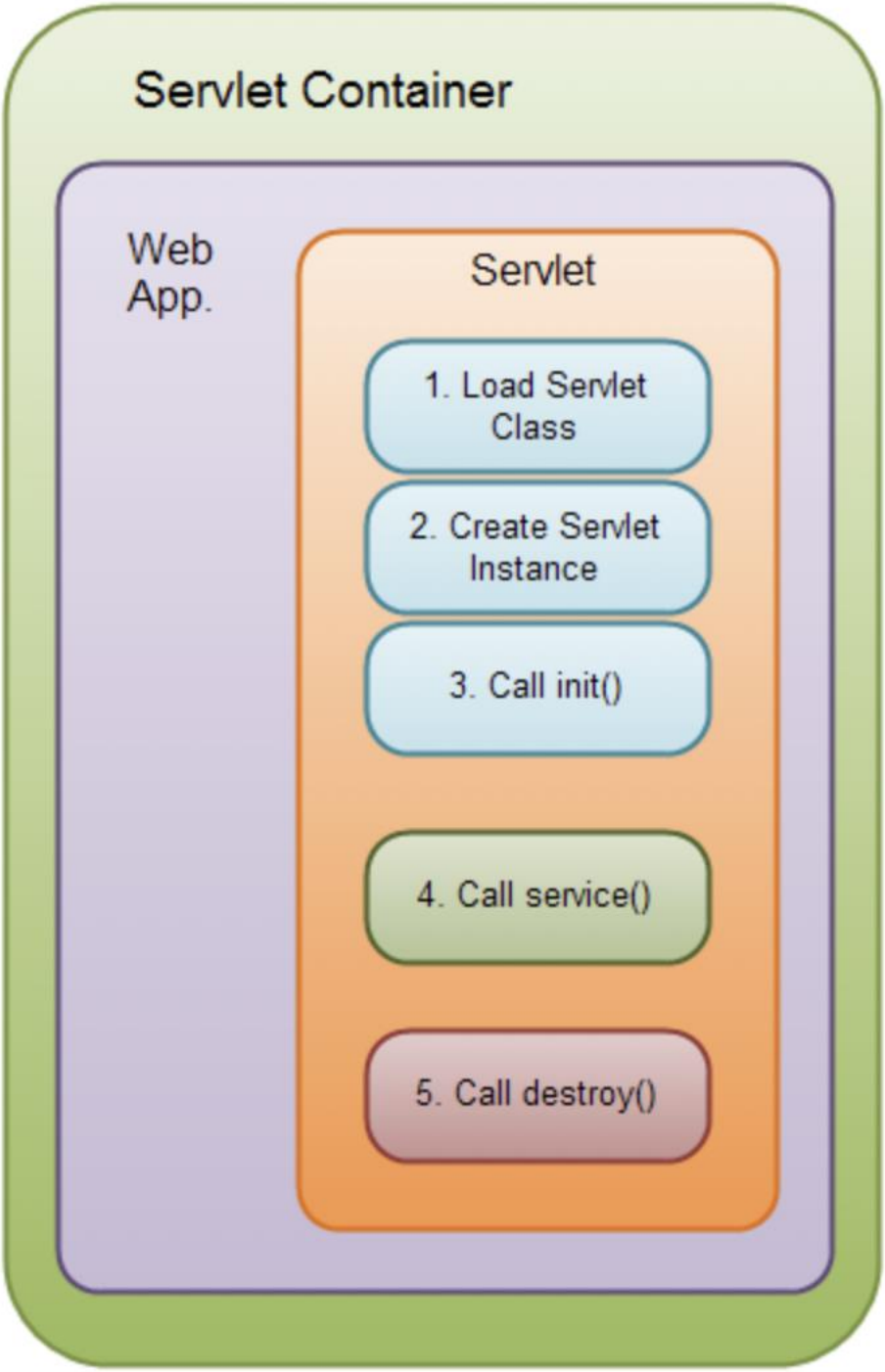


Рис. 2.2. Життєвий цикл сервлета

Першим викликається метод init(). Він дає сервлету можливість ініціалізувати дані та підготуватися для обробки запитів. Найчастіше у цьому методі розміщується код, що кешує дані фази ініціалізації.

Після цього сервлет можна вважати запущеним, він чекає запитів від клієнтів. З'явившийся запит обслуговується методом service(HttpServletRequest request, HttpServletResponse response) сервлета,

викликаний контейнером, проте параметри запиту упаковуються в екземпляр request інтерфейсу HttpServletRequest, переданий в сервлет. Ще одним параметром цього методу є екземпляр response інтерфейсу HttpServletResponse, в який завантажується інформація для передачі клієнта.

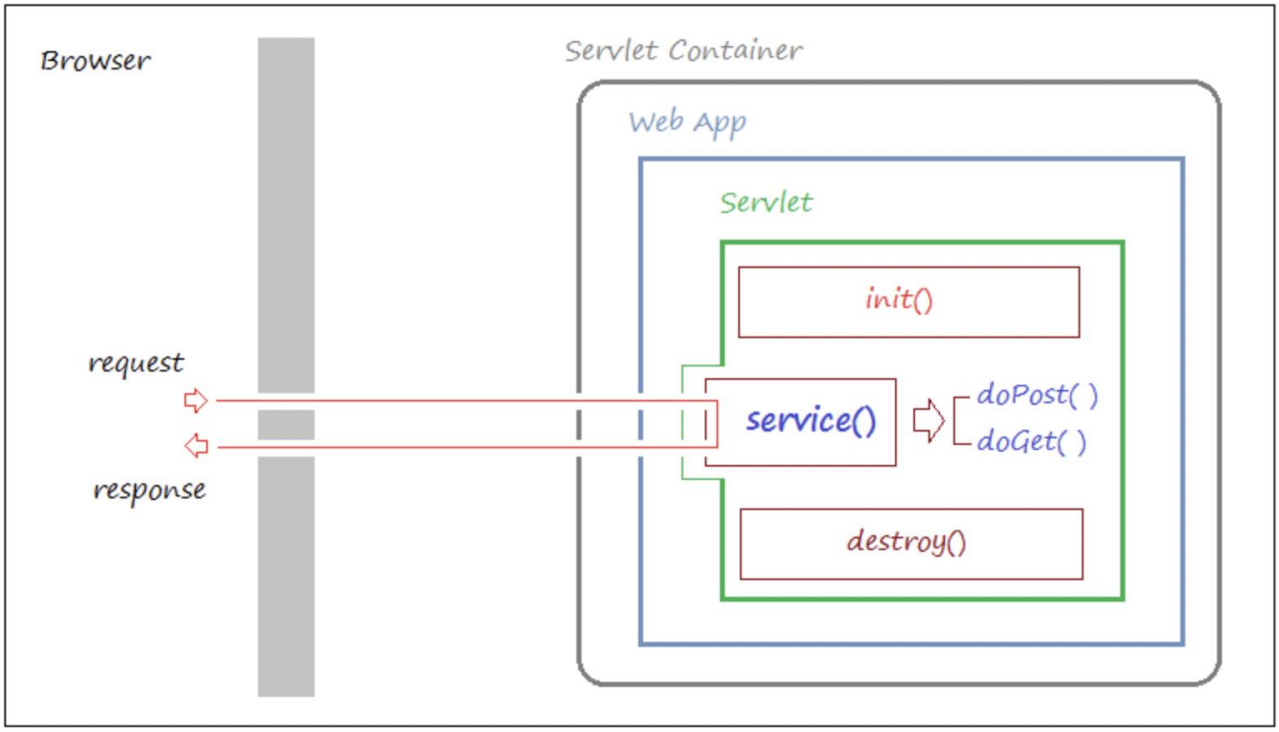


Рис. 2.3. Архітектура роботи сервлету

Для кожного нового клієнта при зверненні до сервлета створюється незалежний потік, у якому здійснюється виклик методу service(). Метод service() призначений для одночасної обробки множини запитів.

При вивантаженні програми з контейнера, тобто після закінчення життєвого циклу сервлета, викликається метод destroy(), у тілі якого слід поміщати код визволення зайнятих сервлетом ресурсів.

Перевагою сервлетів перед CGI або ASP є швидкодія, переносимість на різні платформи, використання об'єктно-орієнтованої мови високого рівня Java, яка розширюється великою кількістю класів та програмних інтерфейсів.

Сервлети підтримуються серверами програм WebSphere, Weblogic, JBoss, Oracle OC4J Server, Glassfish, Geronimo, Apache Tomcat, а також контейнерами сервлетів tomcat, jetty, grizzly та є частиною платформи JEE.

Сервлети реалізують інтерфейс Servlet, у якому крім розглянутих вище методів service(), init(), destroy() передбачено реалізацію методу ServletConfig getServletConfig() — він повертає об'єкт, що містить параметри конфігурації сервлета і дає доступ до середовища виконання.

При розробці сервлетів як суперклас у більшості випадків використовується не інтерфейс Servlet, а абстрактний клас HttpServlet, який відповідає за обробку запитів HTTP.

Метод service() класу HttpServlet служить диспетчером інших методів, кожен із яких обробляє методи доступу до відповідних ресурсів. У специфікації HTTP визначено методи: GET, HEAD, POST, PUT, DELETE, OPTIONS та TRACE. Найчастіше використовуються методи GET і POST, що передають на сервер запити, а також параметри для їх виконання.

Метод GET (method="GET") використовується для запиту вмісту вказаного ресурсу, зображення або гіпертекстового документа. Разом із запитом можуть передаватися додаткові параметри як частина URI, значення можуть вибиратися з полів форми або передаватись безпосередньо через URL. При цьому запити кешуються та мають обмеження на розмір. Цей метод є основним методом взаємодії браузера клієнта та веб-сервера.

Метод POST використовується для передачі даних у вмісті HTTP-запиту на сервер. Користувальницькі дані упаковані в тіло запиту відповідно до поля заголовка Content-Type та включені в URI запиту. При використанні методу POST під URI мається на увазі ресурс, який оброблятиме запит.

Метод PUT схожий з методом POST за тим винятком, що тут URI має на увазі ресурс, який буде створено або збережено на сервері в результаті виконання PUT-запиту.

Метод DELETE призначений видалення цільового ресурсу.

Обидві ці дії на деяких серверах можуть заборонятися через загрозу внутрішній безпеці.

Метод HEAD передбачає повернення сервером такої самої відповіді, як і при використанні GET, але без тіла відповіді. Метод зазвичай

використовується для того, щоб перевірити існування ресурсу або дізнатися, чи змінився запитуваний ресурс з останнього звернення.

OPTIONS повинен повертати інформацію про можливості веб-сервера або параметри з'єднання для конкретного ресурсу.

Метод TRACE повертає клієнту запит у тому вигляді, в якому він прийшов на сервер — використовується для налагодження, визначаючи заголовки, що додаються проміжними серверами, а також тестування налаштувань з'єднання.

Методи HEAD, OPTIONS і TRACE, як правило, реалізуються сервером веб-програми прозоро для програміста і не вимагають будь-якої спеціальної обробки. Тому зазвичай при розробці веб-застосунків вони не розглядаються.

Методи GET і HEAD, за прийнятими угодами, не повинні мати іншого призначення, крім передачі інформації клієнту — вони повинні бути «безпечними» («safe»). Тобто в результаті виконання на сервері не повинно відбуватися жодних «побічних ефектів», які вплинуть на стан ресурсів сервера — таких, як створення, зміна, видалення даних. Також мається на увазі, що запити з безпечними методами можуть бути виконані браузером клієнта в автоматичному режимі, без спеціальної дозволу користувача. Оскільки посилання через елемент <a> у коді HTML за умовчанням мають на увазі використання браузером GET-запитів, то досить часто можна зустріти порушення «безпеки» GET-запитів через використання елемента <a> для виконання будь-яких дій, що змінюють стан сервера, наприклад, посилання виду:

```
<a href="http://example.com/do?action=delete">Delete database</a>
```

Не завжди таке порушення тягне за собою негативні наслідки, проте оскільки інфраструктура глобальних мереж передбачає дотримання угоди щодо «безпеки» GET-запитів, то при взаємодії з наступними факторами можуть виникати негативні наслідки.

− Кешуючі проксі. Оскільки GET-запит (на відміну POST) може

бути кешований проміжним кешуючим проксі-сервером, то при обробці запиту проксі може не виконати запит до цільового сервера, а відразу повернути результат. Таким чином, якщо запит мав на увазі якусь дію над даними на сервері, то ця дія не буде виконана.

− Передзавантаження. Деякі браузери для прискорення навігації виконують завантаження сторінок та інших ресурсів за посиланнями ще до того, як користувач виконав перехід за цими посиланнями. У цьому випадку при порушенні "безпеки" браузер виконає запит, який не був ініційований користувачем. Наприклад, якщо на сторінці розміщені посилання, які видаляють якісь об'єкти через GET-запити, то при відкритті цієї сторінки браузер може викликати видалення всіх цих об'єктів.

− Пошукові системи та інші клієнти, що діють в автоматичному режимі. Різноманітні автоматичні клієнти не можуть відрізнити запити, які виконують будь-які дії від запитів, які повертають інформацію. Тому вони зазвичай орієнтуються за методом запиту: GET-запити виконуються і використовуються для подальшого аналізу, а запити POST ігноруються. При порушенні «безпеки» GET-запитів виникає ситуація, аналогічна до завантаження: виконуються ті дії, які при «нормальній» взаємодії з додатком не повинні були б виконуватися.

− Порушення авторизації. Якщо виконання дії передбачає наявність певних повноважень, то використання GET-запиту для цього надає зловмиснику можливість скористатися привілеями іншого користувача (наприклад, шляхом розміщення посилання з потрібним GET-запитом у елементі <img> на довільному сайті, який може відвідати користувач , який має потрібні повноваження з ідентифікацією через cookies) і тим самим виконати неавторизовану дію.

Тому важливо розуміти різницю між методами GET і POST і використовувати GET тільки там, де виконується лише безпосередня передача даних без виконання активних дій.

Методи GET, HEAD, PUT і DELETE повинні бути ідемпотентними («idempotence») у тому сенсі, що повторне (більше одного разу) виконання запитів за допомогою цих методів матиме той самий ефект, що й одноразове виконання.

До завдання методу service() класу HttpServlet входить аналіз отриманого через запит методу доступу до ресурсів та виклик методу зі схожим ім'ям, де перед ім'ям додається префікс do: doGet(), doPost(), doHead(), doPut(), doDelete (), doOptions() та doTrace(). Розробник повинен перевизначити потрібний метод, розмістивши у ньому функціональну логіку.

Механізм протоколу HTTP не передбачає збереження інформації про свого клієнта. Проте веб-додаток може вимагати інформації про клієнта, особливо таке, реалізація якого передбачає аутентифікацію клієнта без необхідності її підтвердження при зміні сторінок. Окрім підтримки розподіленої сесії, про яку йтиметься у розділі 21, існують і примітивні механізми отримання відомостей про клієнта:

− файли cookie — текстові файли, асоційовані з додатком та зберігаються на комп'ютері клієнта.

− додати до рядка GET запит додаткових параметрів.

− приховані HTML-теги виду

```
<input type="hidden" name="userType" value="admin">
```

які необхідно додавати до кожної сторінки програми для збереження цілісності її архітектури.

2.1.11. JSP

Технологія Java Server Pages (JSP) забезпечує поділ динамічної та статичної частин сторінки, результатом чого є можливість зміни дизайну сторінки, не торкаючись динамічного змісту. Ця властивість використовується при розробці та підтримці сторінок, оскільки дизайнерам не потрібно знати, як працювати з динамічними даними.

JSP-код складається зі спеціальних тегів та виразів, які вказують контейнеру відповідність між тегами та java-кодом для генерації сервлета або його частини. Таким чином підтримується документ, який одночасно містить і статичну сторінку, і теги Java, що керують сторінкою. Статичні частини HTML-сторінок посилаються як рядків метод write(). Динамічні частини включаються у код сервлета. З моменту формування відповіді сервера сторінка поводиться як звичайна HTML-сторінка з асоційованим сервлетом.

Щоб створити найпростішу JSP, достатньо взяти HTML-сторінку та замінити розширення html на jsp. Тільки в цьому випадку для запуску сторінки необхідно спеціальний application server з контейнером сервлетів і особливе розміщення самої сторінки.

2.1.11.1 Взаємодія сервлета та JSP

Сторінки JSP та сервлети ніколи не слід використовувати в інформаційних системах одна без одної. Причиною є принципово різні ролі, які відіграють ці компоненти у додатку. Сторінка JSP відповідальна за формування інтерфейсу користувача та відображення інформації, переданої з сервера. Сервлет виконує роль контролера запитів та відповідей, тобто приймає запити від усіх пов'язаних з ним JSP-сторінок, викликає відповідну бізнес-логіку для їх (запитів) обробки та в залежності від результату виконання вирішує, яку JSP поставити цьому результату в відповідність.

Обмін інформацією між JSP та сервлетом найчастіше здійснюється за допомогою атрибутів об'єктів HttpServletRequest, HttpSession, HttpServletContext.

Для виклику JSP відносною дорогою застосовується метод forward(), для звернення до JSP абсолютною дорогою використовується метод sendRedirect(). Відмінність цих методів у тому, що з методом forward() передається вже існуючий об'єкт запиту request, а виклику методу sendRedirect() формується новий запит. Інформацію в останньому випадку

слід надсилати з іншими об'єктами. До того ж, метод forward() спрацьовує швидше.

2.1.12. JSTL

Повторне використання коду – звичайна техніка програмування. У більшості випадків такий код інкапсулюється в методі, при створенні JSP — стандартного тегу. Найбільш стандартні та затребувані рішення були визначені та зібрані до бібліотеки JSTL.

JSP-сторінки, що включають елементи action (стандартні дії) та теги користувача, не можуть бути технологічними без використання JSTL (JSP Standard Tag Library). Створення сторінок із застосуванням JSTL дозволяє спростити розробку та відмовитись від вживлення java-коду в JSP. Як було показано раніше, сторінки зі скриптлетами важко читати, що викликає проблеми як у програміста, так і у веб-дизайнера, що не має глибоких знань у Java.

Бібліотека тегів JSTL складається з п'яти груп тегів: основні теги – core, теги форматування – formatting, теги для роботи з SQL – sql, теги для обробки XML – xml, функції-теги для обробки рядків – functions.

JSTL надає такі можливості:

− Підтримку Expression Language, що дозволяє розробнику писати про-

− сті вирази всередині атрибутів тега і надає «прозорий» до-

− ступ до змінних у різних областях видимості сторінки;

− організацію умовних переходів та циклів, засновану на тегах, а не на

− скриптовою мовою;

− Просте формування доступу (URL) до різних ресурсів;

− Просту інтернаціоналізацію JSP;

− взаємодія з базами даних (не рекомендується);

− Обробку XML;

− Форматування та розбір рядків за допомогою бібліотеки функцій[13][25].

2.1.13. Servlet Filters

Сервлетний фільтр, відповідно до специфікації, це Java-код, придатний для повторного використання і дозволяє перетворити вміст HTTP-запитів, HTTP-відповідей та інформацію, що міститься в заголовках HTML. Сервлетний фільтр займається попередньою обробкою запиту, перш ніж той потрапляє в сервлет, та/або подальшою обробкою відповіді, що виходить з сервлета.

Сервлетні фільтри можуть:

− перехоплювати ініціацію сервлета перш, ніж сервлет буде ініційовано;

− визначити зміст запиту, перш ніж сервлет буде ініційований;

− модифікувати заголовки і дані запиту, в які упаковується запит, що надходить;

− модифікувати заголовки і дані відповіді, в які упаковується відповідь, що отримується;

− перехоплювати ініціацію сервлета після звернення до сервлета.

2.1.13.1 Інтерфейс javax.servlet.Filter

Сервлетний фільтр може бути налаштований так, що він буде працювати з одним сервлетом або групою сервлетів. Основою для формування фільтрів служить інтерфейс javax.servlet.Filter, який реалізує три методи:

```
void init (FilterConfig config) throws ServletException;

void destroy();

void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException;
```

Метод init викликається, перш ніж фільтр починає працювати, і визначає конфігураційні параметри фільтра. Метод doFilter безпосередньо виконує роботу фільтра. Таким чином, сервер викликає init один раз, щоб запустити фільтр в роботу, а потім викликає doFilter стільки разів, скільки

запитів буде зроблено безпосередньо до цього фільтра. Після того, як фільтр закінчує свою роботу, викликається метод destroy.

Найчастіше фільтри можуть використовуватися для автентифікації чи авторизації користувача або логування запитів. Зазвичай не прийнято навантажувати фільтр складною бізнес-логікою задля зменшення часу обробки запитів.

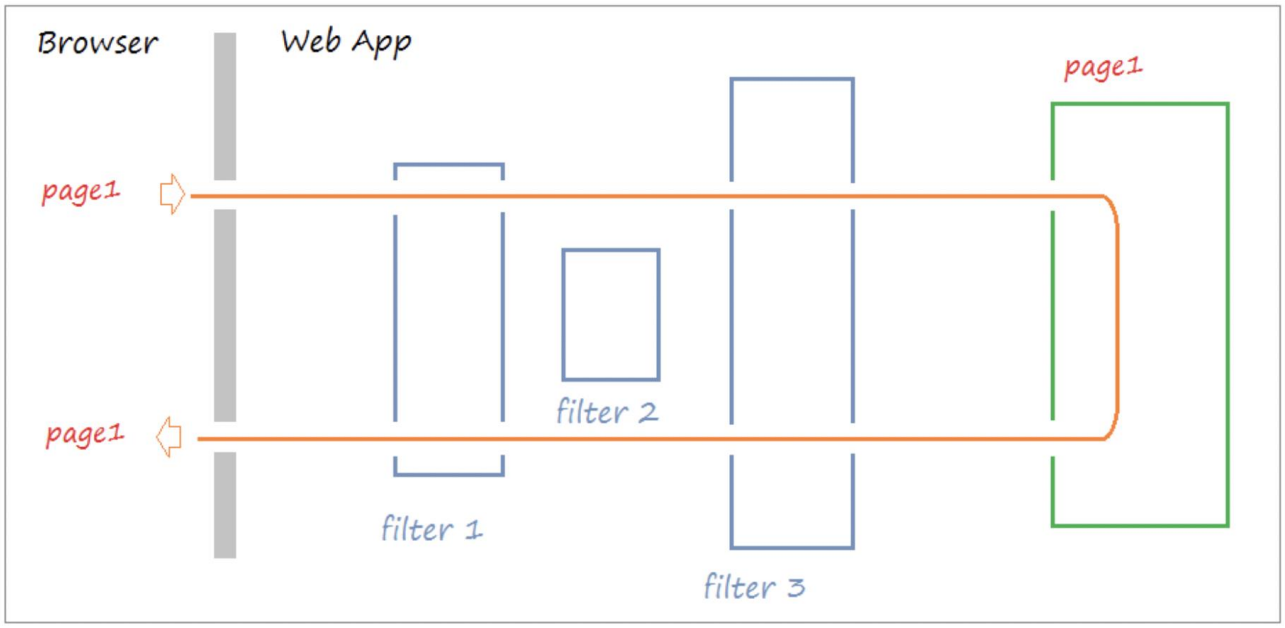


Рис. 2.4. Схема роботи фільтрів в Servlet API.

Ситуація, коли користувач надсилає запит однієї сторінки (page1), цей запит проходить через Filter, у певного filter запит буде перенаправлено на іншу сторінку (page2).

Приклад ситуації:

– Користувач надсилає запит, щоб переглянути сторінку з особистою інформацією.

– Запит буде надіслано на сервер.

– Вона проходить Filter, яка записує інформацію log.

– Йде до Filter і перевіряє, чи ввійшов користувач у систему, filter перевірив, і побачив, що користувач не увійшов до системи, то перенаправляє запит на сторінку входу в систему користувача (page2) [14].

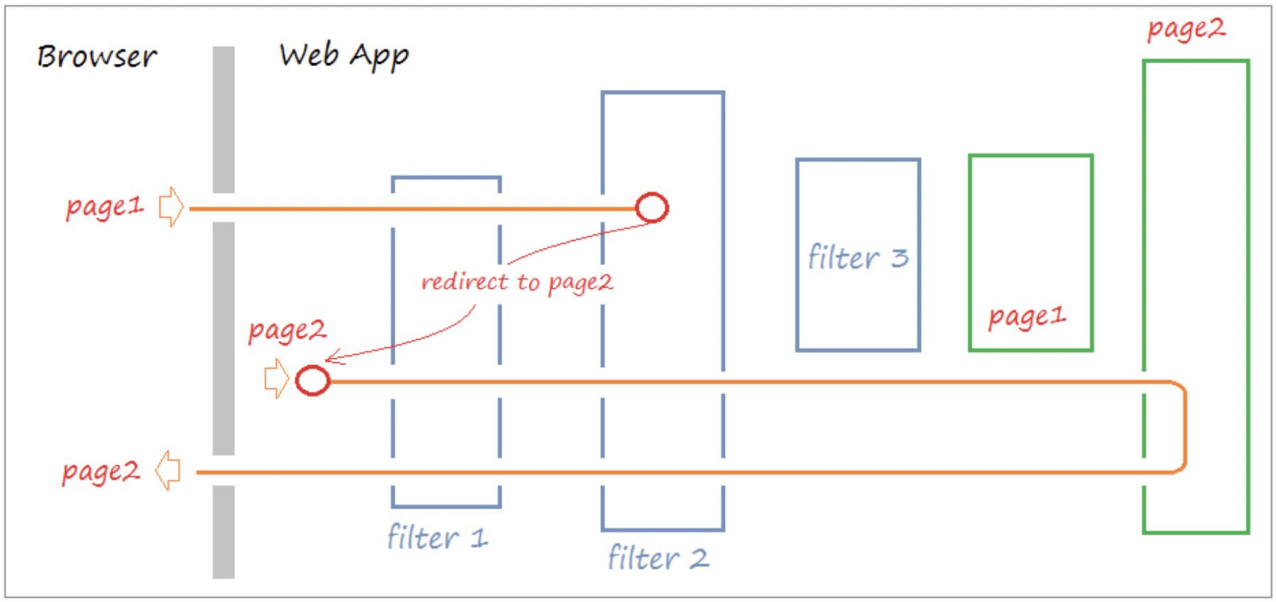


Рис. 2.5. Автентифікація користувача через фільтри.

2.1.14. Jackson

Jackson — це високопродуктивний процесор JSON для Java. Його розробники високо оцінюють поєднання швидких, правильних, легких та ергономічних атрибутів бібліотеки.

Джексон надає кілька підходів до роботи з JSON, включаючи використання анотацій прив’язки до класів POJO для простих випадків використання.

2.1.15. OpenPDF

OpenPDF — це безкоштовна бібліотека Java для створення та редагування PDF-файлів з ліцензією на вільне програмне забезпечення Mozilla Public License та GNU Library General Public License. Це форк iText, створений через те, що ліцензія iText була змінена з LGPL / MPL на подвійну AGPL і проприетарну ліцензію, щоб оригінальні автори продавали запатентовану версію програмного забезпечення. Версія 1.3.26 була випущена 3 травня 2021 року.

В додатку ця бібліотека використана для генерації квитанцій по доставкам[15].

2.1.16. Apache POI

Apache POI являє собою API, який дозволяє використовувати файли MS Office у додатках Java. Данна бібліотека розробляється і розширюється Apache Software Foundation і носить відкритий характер. Apache POI включає класи та методи для чтення та запису інформації в документи MS Office.

Бібліотека використовується для генерації звітів згідно встановлених фільтрів

HSSF	Horrible Spreadsheet Format	омпонент читання та записи файлів MS-Excel, формат XLS
XSSF	XML Spreadsheet Format	Компонент читання та запису файлів MS-Excel, формат XLSX
HPSF	Horrible Property Set Format	Компонент отримання наборів властивостей файлів MS-Office
HWPF	Horrible Word Processor Format	Компонент для читання та запису файлів MS-Word, формат DOC
XWPF	XML Word Processor Format	Компонент читання та запису файлів MS-Word, формат DOCX
HSLF	Horrible Slide Layout Format	Компонент для читання та запису файлів PowerPoint, формат PPT
XSLF	XML Slide Layout Format	Компонент для читання та запису файлів PowerPoint, формат PPTX
HDGF	Horrible DiaGram Format	Компонент роботи з файлами MS-Visio, формат VSD
XDGF	XML DiaGram Format	Компонент роботи з файлами MS-Visio, формат VSDX

Табл. 2.1. Опис компонентів бібліотеки[16].

2.1.17. Lombok

Проект Lombok — це плагін компілятора, який додає в Java нові ключові слова і перетворює анотації в Java-код, зменшуючи зусилля на розробку і забезпечуючи деяку додаткову функціональність. Головна ціль цього плагіна це автоматизація генерації геторів, сеторів, конструкторів, білдерів та інших корисних методів для классів, які використовуються для маніпулювання даними, наприклад:

− DTO (Data Transfer Object) – Об’єкти як правило без бізнес логіки, служать лише для передачі даних між шарами додатка

− Entity – об’єкти, які підтримують персистентність даних у пам’яті програми через зв’язок з базою даних

− POJO (Plain Old Java Object) – "старий добрий Java-об'єкт", простий Java-об'єкт, не успадкований від якогось специфічного об'єкта і не реалізує жодних службових інтерфейсів понад тих, які потрібні для бізнес-моделі.

Цей плагін дуже зручний, адже це стандартна річ коли під час розробки функціоналу ми змінюємо поля класів і разом з тим те ж саме необхідно робити з методами, але Lombok робить це за розробника автоматично.

2.1.18. Gradle

Інструменти збірки – це програми, які автоматизують створення виконуваних програм із вихідного коду (наприклад, .apk для програми Android). Побудова включає в себе компіляцію, зв’язування та пакування коду в придатну для використання або виконувану форму.

В основному автоматизація збірки – це сценарії або автоматизація різноманітних завдань, які розробники програмного забезпечення виконують у своїй повсякденній діяльності, як-от:

− Завантаження залежностей.

− Компіляція вихідного коду в двійковий код.

− Упакування цього двійкового коду.

− Запуск тестів.

− Розгортання в виробничих системах.

Gradle — це інструмент автоматизації збірки з відкритим кодом, який розроблений так, щоб бути достатньо гнучким для створення практично будь-якого типу програмного забезпечення. Нижче наведено короткий огляд деяких з його найважливіших функцій:

Висока ефективність

Gradle уникає непотрібної роботи, запускаючи лише ті завдання, які потрібно виконати, оскільки змінилися їхні вхідні або вихідні дані. Також може використовувати кеш збірки, щоб увімкнути повторного використання результатів завдань з попередніх запусків або навіть з іншої машини (зі спільним кешем збірки).

Фундамент JVM

Gradle працює на JVM, і для його використання необхідно встановити Java Development Kit (JDK). Це також полегшує запуск Gradle на різних платформах.

Конвенції

Gradle має багато спільного з Maven і робить звичайні типи проектів, наприклад проекти Java, легко створюваними шляхом впровадження конвенцій. Можна застосовувати відповідні плагіни і легко отримати скрипти тонкої збірки для багатьох проектів. Але ці домовленості не обмежують вас: Gradle дозволяє перевизначати їх, додавати власні завдання та вносити багато інших налаштувань до збірок на основі конвенцій.

Розширюваність

Є можливість легко розширити Gradle, щоб надати власні типи завдань або навіть створити модель. Приклад цього дивіться у підтримці збірки Android: вона додає багато нових концепцій збірки, таких як варіанти та типи збірки.

Підтримка IDE

Кілька основних IDE дозволяють імпортувати збірки Gradle та взаємодіяти з ними: Android Studio, IntelliJ IDEA, Eclipse і NetBeans. Gradle

також підтримує створення файлів рішень, необхідних для завантаження проекту в Visual Studio.

Прозорість

Сканування збірки надають вичерпну інформацію про запуск збірки, яку можна використовувати для виявлення проблем зі створенням. Вони особливо добре допомагають вам визначити проблеми з продуктивністю збірки. Ви також можете поділитися скануваннями збірки з іншими, що особливо корисно, якщо вам потрібно попросити поради щодо вирішення проблеми зі збіркою[17][18].

2.1.19. JDBC

JDBC (англ. Java DataBase Connectivity - з'єднання з базами даних на Java) - платформно незалежний промисловий стандарт взаємодії Java-додатків з різними СУБД, реалізований у вигляді пакета java.sql, що входить до складу Java SE.

− JDBC вирішує такі завдання:

− Створення з'єднання із БД.

− Створення SQL виразів.

− Виконання SQL-запитів.

− Перегляд та модифікація отриманих записів.

Якщо говорити загалом, то JDBC – це бібліотека, що забезпечує цілий набір інтерфейсів доступу до різних БД.

Для доступу до кожної конкретної БД необхідний спеціальний JDBC драйвер, який є адаптером Java додатки до БД.

2.1.19.1 Будова JDBC

У загальному вигляді, JDBC і двох шарів:

− JDBC API. Забезпечує з'єднання програми – JDBC Manager.

− JDBC Driver API. Забезпечує з'єднання “JDBC Manager – драйвер”.

JDBC API використовує менеджер драйверів та спеціальні драйвери БД для забезпечення підключення до різних баз даних.

JBDC Manager перевіряє відповідність драйвера та конкретної БД. Він підтримує можливість використання кількох драйверів для одночасної роботи з декількома видами БД.

Схематично, JDBC можна представити у такому вигляді:

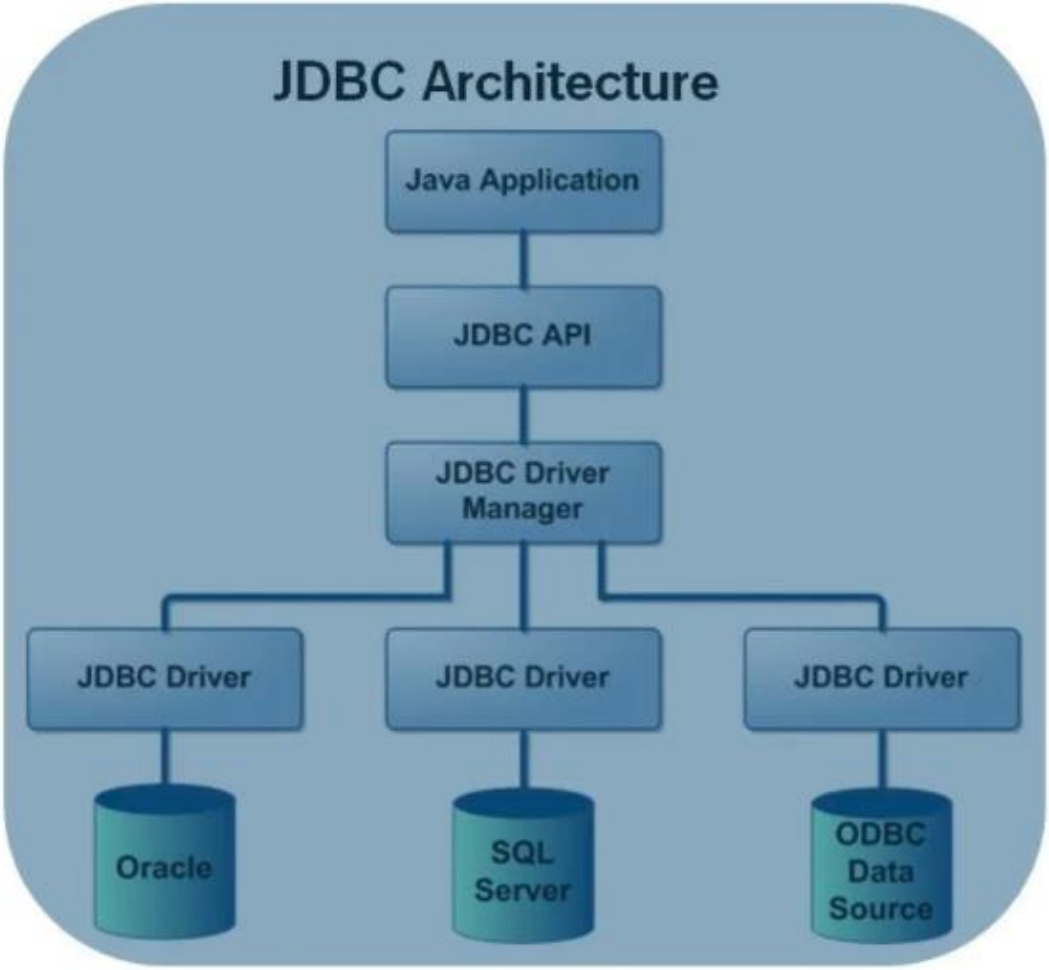


Рис. 2.6. Схема роботи JDBC.

2.1.19.2 Елементи JDBC

JDBC API складається з наступних елементів:

− Менеджер драйверів (Driver Manager). Цей елемент керує списком драйверів БД. Кожний запит на підключення вимагає відповідного драйвера. Перший збіг дає нам з'єднання.

− Драйвер (Driver). Цей елемент відповідає за зв'язок із БД. Працювати з ним доводиться дуже рідко. Натомість частіше використовується об'єкти DriverManager, які керують об'єктами цього типу.

− З'єднання (Connection). Цей інтерфейс забезпечує нас методами роботи з БД. Усі взаємодії з БД відбуваються лише через Connection.

− Вираз (Statement). Для підтвердження SQL-запитів ми використовуємо об'єкти, створені за допомогою цього інтерфейсу.

− Результат (ResultSet). Примірники цього елемента містять дані, отримані в результаті виконання SQL - запиту. Він працює як ітератор та “пробігає” за отриманими даними.

− Винятки (SQL Exception). Цей клас обробляє всі помилки, які можуть виникнути під час роботи з БД[19].

2.1.20. Tomcat

Програмне забезпечення Apache Tomcat® є реалізацією з відкритим вихідним кодом Jakarta Servlet, Jakarta Server Pages, Jakarta Expression Language, Jakarta WebSocket, Jakarta Annotations та Jakarta Authentication специфікацій. Ці специфікації є частиною платформи Jakarta EE.

Платформа Jakarta EE є еволюцією платформи Java EE. Tomcat 10 і новіші реалізують специфікації, розроблені як частина Jakarta EE. Tomcat 9 і раніше реалізують специфікації, розроблені як частина Java EE.

Програмне забезпечення Apache Tomcat розробляється у відкритому середовищі з участю учасників і випускається під ліцензією Apache версії 2. Проект Apache Tomcat призначений як спільна робота найкращих розробників з усього світу.

Програмне забезпечення Apache Tomcat підтримує численні масштабні, критично важливі веб-додатки в різних галузях і організаціях.

Цей контейнер сервлетів був обраний як реалізація серверу для даного проекту. Він повністю підходить для виконання заданої бізнес логіки, загалом для приймання HTTP запитів від користувачів та обробки їх на серверній частині мовою Java згідно специфікації Jakarta Servlet.

2.1.20.1 Компоненти Tomcat

− Catalina. Це контейнер сервлетів Tomcat. Catalina реалізує

специфікації Sun Microsystems для сервлетів і сторінок JavaServer (JSP). У Tomcat елемент Realm представляє «базу даних» імен користувачів, паролів і ролей (подібно групам Unix), призначених цим користувачам. Різні реалізації Realm дозволяють інтегрувати Catalina в середовища, де така інформація аутентифікації вже створюється та підтримується, а потім використовувати цю інформацію для реалізації безпеки, керованої контейнером, як описано в специфікації сервлетів.

− Coyote. Це компонент Connector для Tomcat, який підтримує протоколи HTTP 1.1 і 2 як веб-сервер. Це дозволяє Catalina, номінально Java Servlet або JSP-контейнер, також діяти як звичайний веб-сервер, який обслуговує локальні файли як HTTP-документи. Coyote прослуховує вхідні підключення до сервера на певному порту TCP і пересилає запит до Tomcat Engine, щоб обробити запит і надіслати відповідь клієнту, що запитує. Інший конектор Coyote, Coyote JK, слухає аналогічно, але замість цього пересилає свої запити на інший веб-сервер, такий як Apache, використовуючи протокол JK. Зазвичай це забезпечує кращу продуктивність.

− Jasper. Це двигун JSP Tomcat. Jasper аналізує файли JSP, щоб скомпілювати їх у код Java як сервлети (які може оброблятися Catalina). Під час виконання Jasper виявляє зміни до файлів JSP і перекомпілює їх. Починаючи з версії 5, Tomcat використовує Jasper 2, який є реалізацією специфікації JSP 2.0 компанії Sun Microsystems. Від Jasper до Jasper 2 були додані важливі функції:

− Об’єднання бібліотек тегів JSP – кожна розмітка тегів у файлі JSP обробляється класом обробника тегів. Об'єкти класу обробника тегів можна об'єднати в пул і повторно використовувати в усьому сервлеті JSP.

− Фонова компіляція JSP – під час перекомпіляції модифікованого коду JSP Java старіша версія все ще доступна для запитів сервера. Старіший сервлет JSP видаляється, як тільки новий сервлет JSP закінчиться перекомпілювати.

− Перекомпілюйте JSP, коли включені зміни сторінки – сторінки можна вставляти та включати в JSP під час виконання. JSP буде перекомпільовано не тільки зі змінами файлів JSP, але й із включеними змінами сторінок.

− Компілятор Java JDT – Jasper 2 може використовувати компілятор Java Eclipse JDT (Java Development Tools) замість Ant і javac[20].

2.1.21. JUnit

JUnit — це платформа модульного тестування для мови програмування Java. JUnit відігравав важливу роль у розробці методології, керованої тестуванням (Test-Driven Development (TDD)), і є однією з сімейств платформ модульного тестування, які спільно відомі як xUnit, що виникли разом із SUnit.

JUnit пов’язується як JAR під час компіляції. Остання версія фреймворка, JUnit 5, знаходиться в пакеті org.junit.jupiter. Попередні версії JUnit 4 і JUnit 3 були в пакетах org.junit і junit.framework відповідно.

Дослідження, проведене в 2013 році серед 10 000 проектів Java, розміщених на GitHub, показало, що JUnit (у зв’язку з slf4j-api) був найбільш поширеною зовнішньою бібліотекою. Кожну бібліотеку використовували 30,7% проектів.[21]

2.2. Архітектурні рішення

В розробці додатку були використані серверні технології Java без використання веб-фреймворків, таких як Angular, React, тощо. Тому серверний код знаходиться на одній машині, разом з фронтенд частиною. Також особливістю є те, що HTML генерується на серверній стороні за допомогою технології JSP, таким чином фактично проект написаний повністю серверним годом та генерація HTML ведеться на льоту та віддається у відповідь на запит користувача. Також реалізовано декілька ендпоінтів з підтримкою AJAX запитів, які не генеруються HTML код, а віддають відповідь в форматі JSON. Зважаючи на це можна сказати, що додаток виконаний в монолітній архітектурі.

На перших етапах розробки така архітектура виграє в порівнянні з мікросервисною за наступних причин:

− Витрачається значно менше часу на розрубку та для цього достатньо одного Full-stack інженера.

− Додаток збираєьтся в 1 файл формату .war та розвертається в контейнері сервлетів Tomcat. Це забезпечує швидкий деплоймент на сервер.

− Простота в маштабуванні. Просто розмістити додаток за балансувальником навантаження і змінювати кількість екземплярів додатка в залежності від навантаження.

Однак, як тільки додаток стає великим, а команда збільшується в розмірах, цей підхід має ряд недоліків, які стають все більш істотними:

− Велика монолітна база коду лякає розробників, особливо тих, хто є новичком у команді. Програму може бути важко зрозуміти та змінити. Як наслідок, розвиток зазвичай сповільнюється. Крім того, оскільки немає жорстких меж модулів, модульність з часом руйнується. Більше того, оскільки буває важко зрозуміти, як правильно реалізувати зміну, якість коду з часом знижується. Це низхідна спіраль.

− Перевантажена IDE – чим більша база коду, тим повільніше IDE та менш продуктивні розробники.

− Перевантажений веб-контейнер – чим більше програма, тим більше часу потрібно для її запуску. Це дуже вплинуло на продуктивність розробників, оскільки час витрачався на очікування запуску контейнера. Це також впливає на розгортання.

− Постійне розгортання є складним – велика монолітна програма також є перешкодою для частих розгортань. Щоб оновити один компонент, потрібно повторно розгорнути всю програму. Це призведе до переривання фонових завдань незалежно від того, чи вплинуть вони на них, і, можливо, спричинить проблеми. Також існує ймовірність, що компоненти, які не були оновлені, не запустяться належним чином. У результаті зростає ризик,

пов’язаний з перерозподілом, що перешкоджає частим оновленням. Це особливо проблема для розробників користувальницького інтерфейсу, оскільки їм зазвичай потрібно швидко виконувати ітерації та часто перерозгортати.

− Масштабування програми може бути складним – монолітна архітектура полягає в тому, що вона може масштабуватися лише в одному вимірі. З одного боку, він може масштабуватися зі збільшенням обсягу транзакцій, запускаючи більше копій програми. Деякі хмари можуть динамічно регулювати кількість екземплярів залежно від навантаження. Але з іншого боку, ця архітектура не може масштабуватися зі збільшенням обсягу даних. Кожна копія екземпляра програми матиме доступ до всіх даних, що робить кешування менш ефективним і збільшує споживання пам’яті та трафік введення-виводу. Крім того, різні компоненти програми мають різні вимоги до ресурсів – один може завантажувати процесор, а інший – пам’ять. З монолітною архітектурою ми не можемо масштабувати кожен компонент незалежно

− Перешкода на шляху розвитку масштабування. Монолітний додаток також є перешкодою для розвитку масштабування. Як тільки програма досягає певного розміру, корисно розділити інженерну організацію на команди, які зосереджені на певних функціональних областях. Наприклад, ми можемо захотіти мати команду UI, команду бухгалтерії, команду інвентаризації тощо. Проблема з монолітною програмою полягає в тому, що вона заважає командам працювати незалежно. Команди повинні координувати свої зусилля з розвитку та передислокації. Команді набагато складніше внести зміни та оновити виробництво.

− Потрібна довгострокова прихильність до стеку технологій — монолітна архітектура змушує вас бути одруженим із технологічним стеком (а в деяких випадках і з конкретною версією цієї технології), який ви вибрали на початку розробки. При монолітному застосуванні може бути важко

поступово впроваджувати новішу технологію. Наприклад, уявімо, що ви вибрали JVM. У вас є певний вибір мов, оскільки, крім Java, ви можете використовувати інші мови JVM, які добре взаємодіють з Java, наприклад Groovy і Scala. Але компоненти, написані мовами, які не є JVM, не мають місця у вашій монолітній архітектурі. Крім того, якщо ваша програма використовує фреймворк платформи, який згодом стає застарілим, то поступово переносити програму на новішу та кращу платформу може бути складно. Можливо, для того, щоб прийняти нову платформу, доведеться переписати весь додаток, що є ризикованою справою.[22]

Незважаючи на ці недоліки, для демонтрації готового MVP ця архітектура підходить. У майбутньому при розширенні функціоналу перехід на окреме розгортання фронтенду та бекенту не забере багато часу.

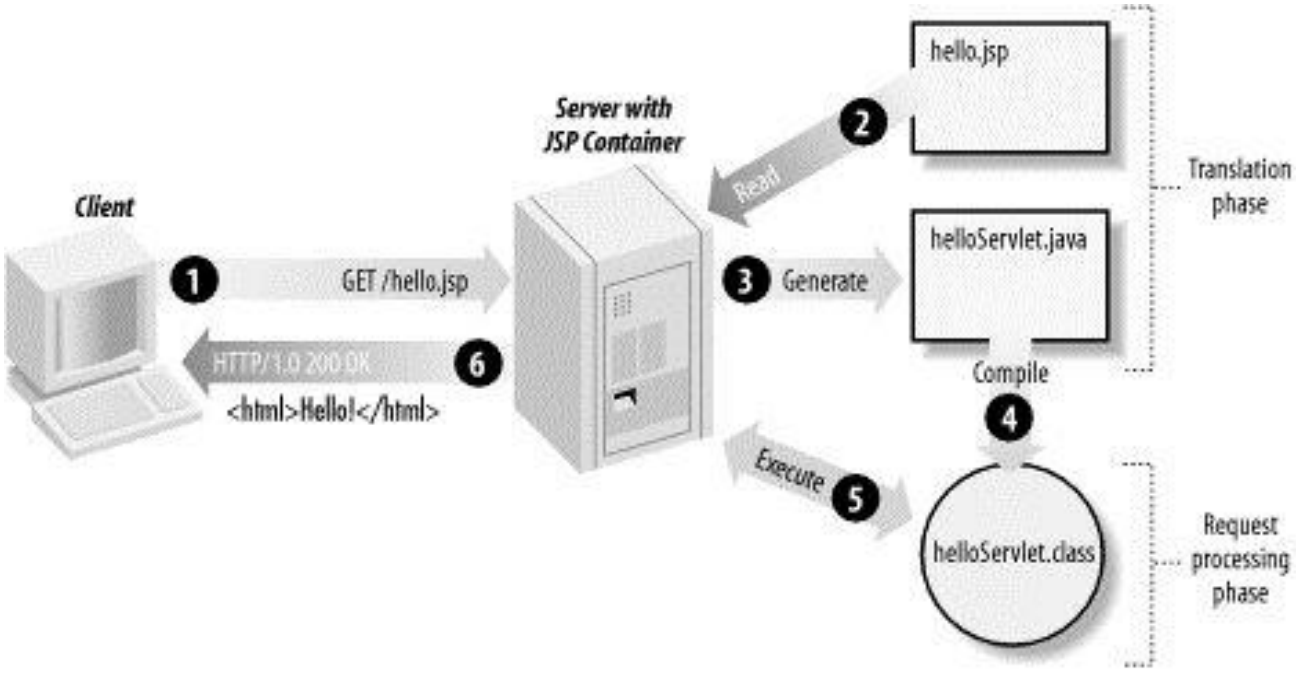


Рис.2.7. Схема роботи серверу

2.3. Висновки розділу

В цьому розділі були виконані такі завдання:

– В якості основного стеку техологій вибрана мова Java та реалізвція специфікації JSR-000369 Java Servlet 4.0 а якості фреймворку Servlet API.

– В якості контейнера сервленів обрано Tomkat як найперевіреніший сервер додатків на Java.

− Для з’єднання з БД обраний протокол JDBC адже він є найгнучкішим з усіх запропонованих стандартів

− Обрані додаткові бібліотеки для генерування звітів та квитанцій

3. ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗРОБЛЕНИХ КОМПОНЕНТІВ КОМП’ЮТЕРНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

Всі вихідні файли з кодом доступні за посиланням на систему SCM GitHub: https://github.com/EvgenRomanov98/cargoDelivery. Загалом система спроектована за клієнт-серверною архітектурою. Система розроблена для Unix систем, тож всі команди наведені нижче будуть описані процеси для роботи на таких системах (Linux, MacOS).

3.1. Структура проекту

На Рис.3.1. зображено загальну структуру проекту.

− ua/nmu/cargo_delivery/clients – тут зберігаються клієнтський код для з’єднання з сторонніми API, наприклад MapBox.

− ua/nmu/cargo_delivery/dto – каталог для DTO классів, скі служать для передачі даних між прошарками програми.

− ua/nmu/cargo_delivery/exceptions – стандартні помилки програми

− ua/nmu/cargo_delivery/filters – фільтери HTTP запитів

− ua/nmu/cargo_delivery/listeners – слухачі подій програми, зокрема подій підняття контексту.

− ua/nmu/cargo_delivery/model – файли-сутності для репрезентації таблиць в базі данних, допоміжні класи для створення звітів та квитанцій.

− ua/nmu/cargo_delivery/servlets – безпосереднь сервлети, які приймають та оброблюють HTTP запити до серверу. Також у каталозі ua/nmu/cargo_delivery/model/commands реалізовано патерн command, для динамічної валідації номера телефону і пошти через ValidationServlet. Цей сервлет приймає AJAX запити та перевіряє email та номер телефону користувача на унікальність через БД.

− ua/nmu/cargo_delivery/tags – кастомні теги для оптимізації даних на JSP сторінці.

− webapp – структура фронтенд частини проекту з JSP файлами. Вона схожа з структурою на Рис. 3.6, яка генерується під час деплоя на сервер.

− test – JUnit тести проекту.

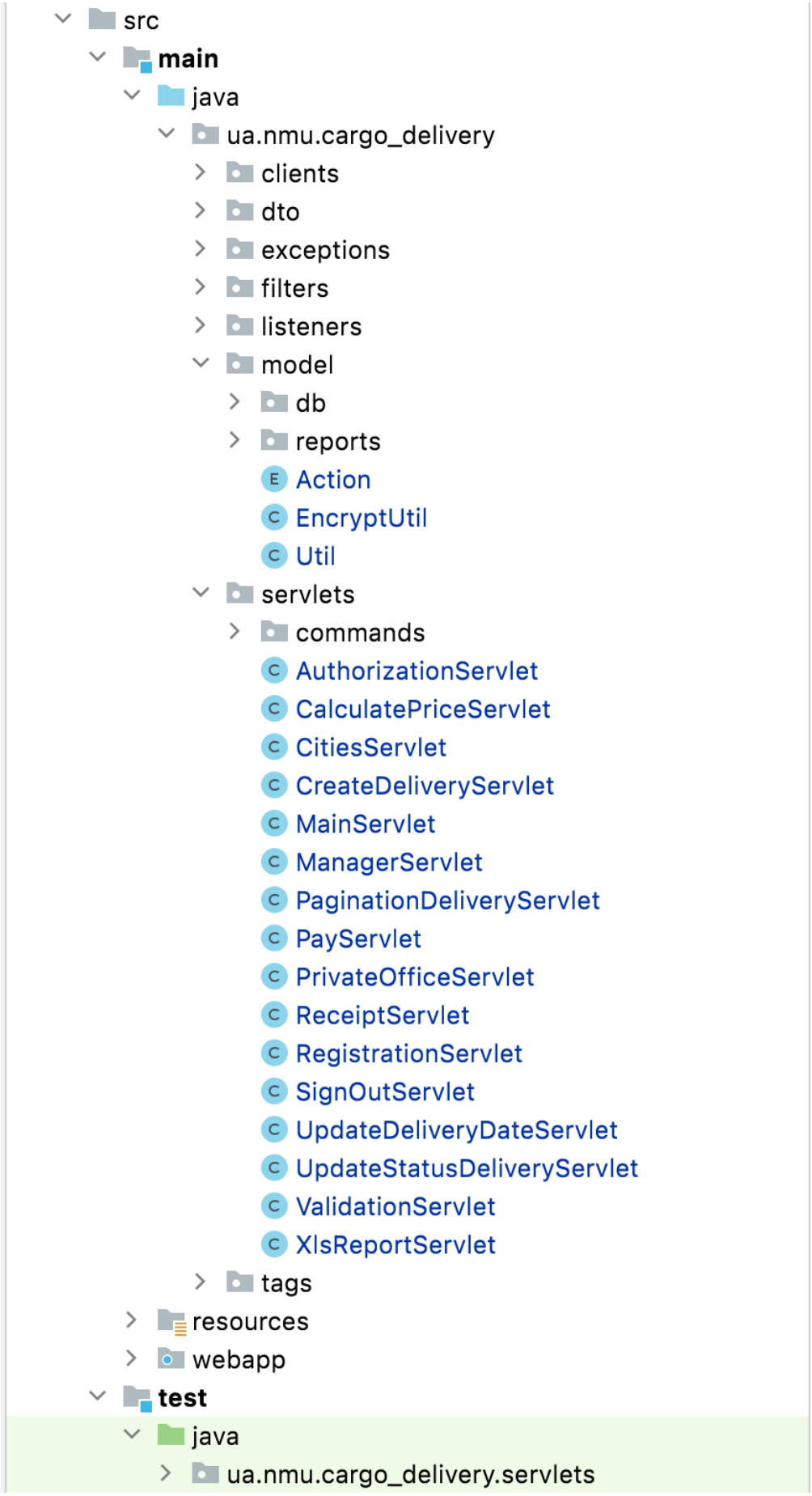


Рис. 3.1. Структура вихідного коду проекту

3.1.1. Патерн Команда(Command)

Команда — це поведінковий патерн проектування, який перетворює запити на об’єкти, дозволяючи передавати їх як аргументи під час виклику методів, ставити запити в чергу, логувати їх, а також підтримувати скасування операцій.[23]

То ж цей патерн служить для уніфікації повторюваних дій в користувацькому інтерфейсі. В програмі є функціонал валідації реєстраційної форми для якого як раз підходить цей патерн, адже виконуються схожі дії для різних данних. Тому для того, щоб не створювати зайвих HTTP шляхів використовується 1 і в ньому передається в параметрах вид команди. Він використовується у ValidationSrvlet для визначення дії. Приклад команди:

https://cargo-delivery-service.herokuapp.com/validate?check=email&param=evrom98%40gmail.com

Тут можна побачити, що параметр запиту check=email, це і є визначним фактором в логікі цього патерну. Далі в сервлеті за цим параметром визначається яка саме команда буде запущена для обробки запиту.

3.2. Heroku

Розгортання сервісу виконано на базі Heroku – це хмарна PaaS-платформа, яка дозволяє легко розгортати свої аплікації для загального доступу. Сервіс розгорнуто на базі стеку Heroku-20, який базується на Ubuntu версії 20.04. Для інстансу було виділено 500МБ пам’яті, з яких зараз аплікація займає 126.7МБ. Регіон розгортання – Європа.

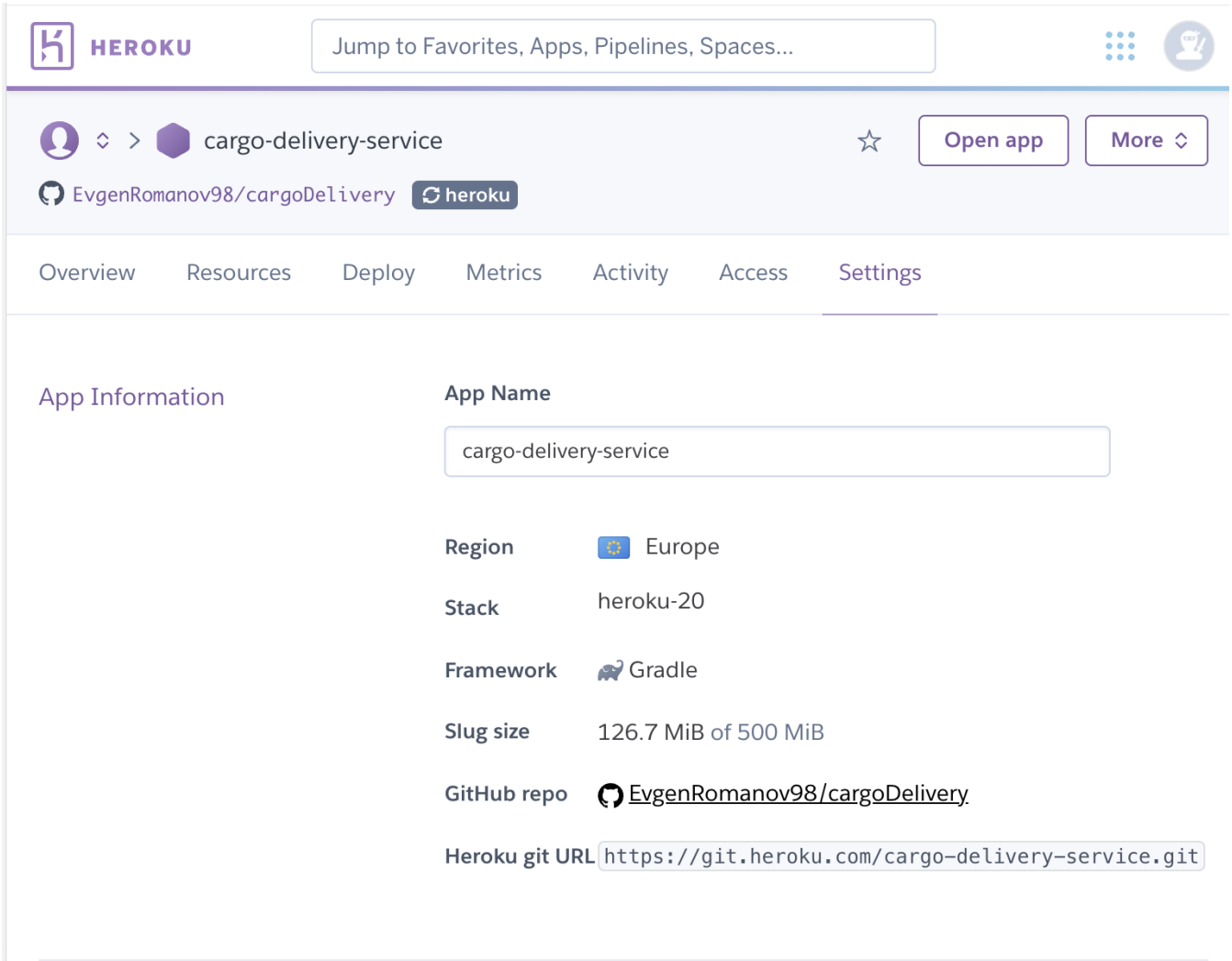


Рис. 3.2. Налаштування серверу

Для підключення до бази даних були інстальовані наступні змінні:

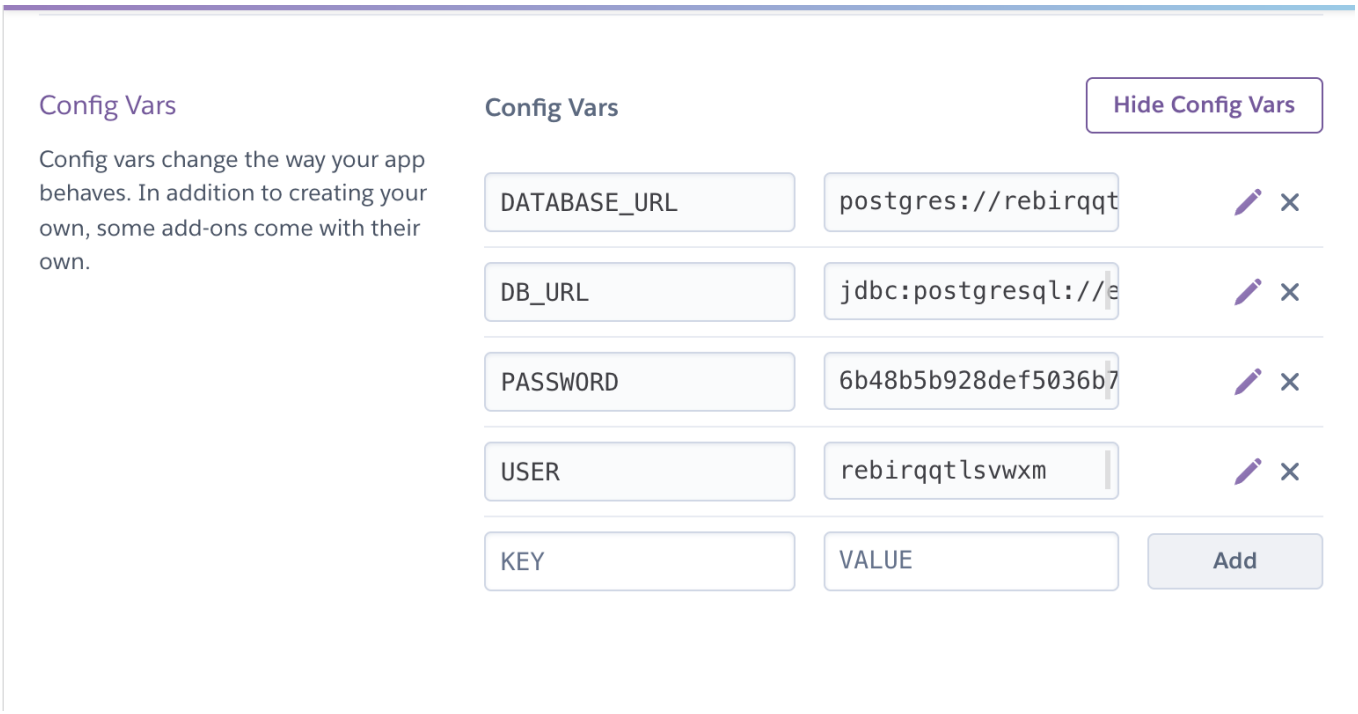


Рис. 3.3. Конфігурація системних змінних

Для розгортання сервісу обраний heroku/gradle BuildPack:

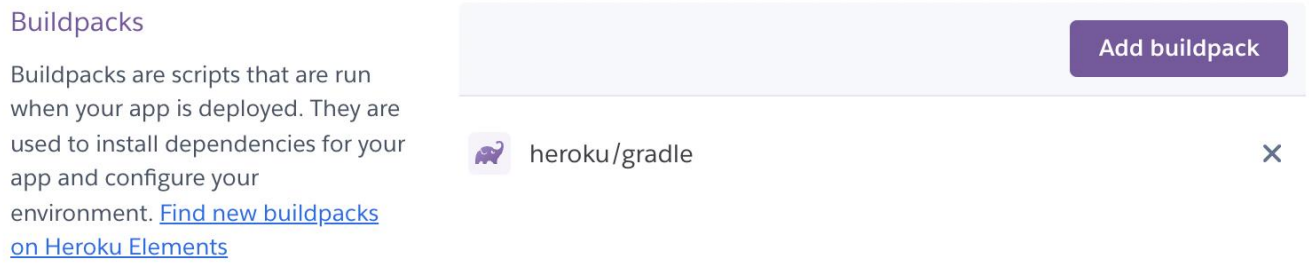


Рис. 3.4. BuildPack

Buildpacks — це сценарії, які запускаються під час розгортання програми. Вони використовуються для встановлення залежностей для вашої програми та налаштування середовища. Він був вибраний у відповідності з використовуваним фреймворком для збірки вихідного коду Gradle.

Для запуску проекту на сервері додано спеціальна бібліотека com.github.jsimone:webapp-runner:9.0.27.1. Вона дозволяє виконати деплой зібраної версії програми з розширенням .war безпосередньо в контейнер

сервлетів Tomcat. Для перевірки працездатності розширення необхідно завантажити інструмент Heroku CLI на локальну машину за допомогою команд:

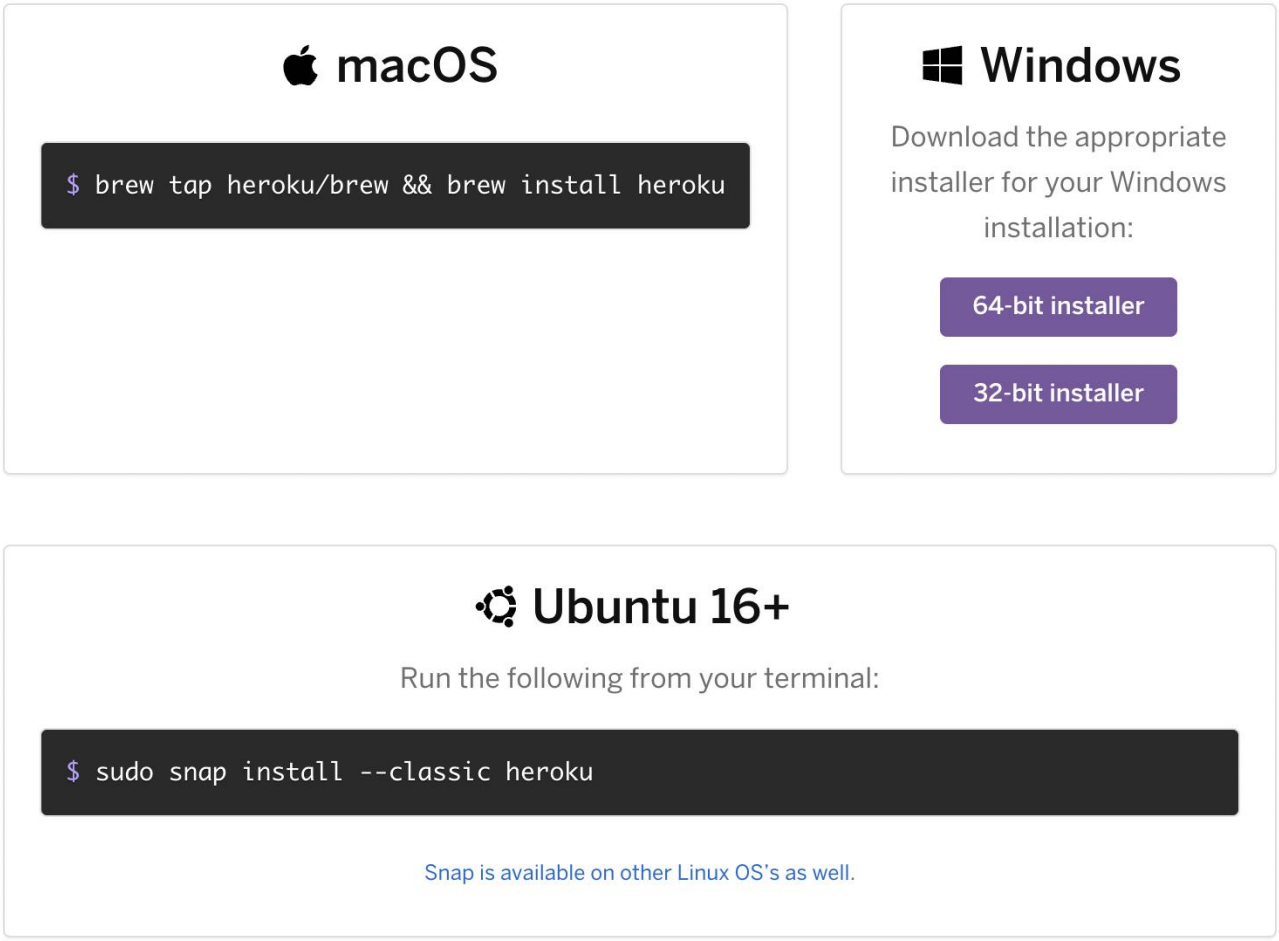


Рис. 3.5. Методи встановлення Heroku CLI.

Далі необхідно запустити наступне знаходячись в каталозі з проектом:

```
./gradlew build && ./gradlew copyToLib && heroku local
```

Це зробить 3 дії:

– Збере додаток в .war файл

– Запустить copyToLib задачу. Її фукція сворити робочий каталог для Tomcat та перемістити збірку в цей каталог

– Запуск додатка

3.3. Структура робочого каталогу Tomcat

Розширення генерує структуру як на малюнку Рис. 3.6. В каталозі webapps знаходяться скомпільовані файли класів (webapps/expanded/WEB-INF/classes), бібліотеки (webapps/expanded/WEB-INF/lib), jsp сторінки. Каталоги tld та tags містять в собі кастомні теги, які розроблені програмістом власноруч. Вони зберігають в собі опис цих тегів а також яким классом оброблювати запити до цих тегів. Для коректної роботи додатку та з цілями безпеки основні файли jsp були приховані в WEB-INF, адже сервер не дозволяю прямий доступ до цих файлів, коли вони знаходяться в цьому каталозі.

Каталог work містить в собі компільовані файли JSP, які конвертуються в звичайні сервлети, Java класси, за допомогою механізму Jasper.

Важлива особливість, що файли JSP конвертуються одноразово в рантаймі, тобто динамічно, коли запит був вперше адресований до конкретної сторінки. Далі вона зберігається і перевикористовується в наступні рази.

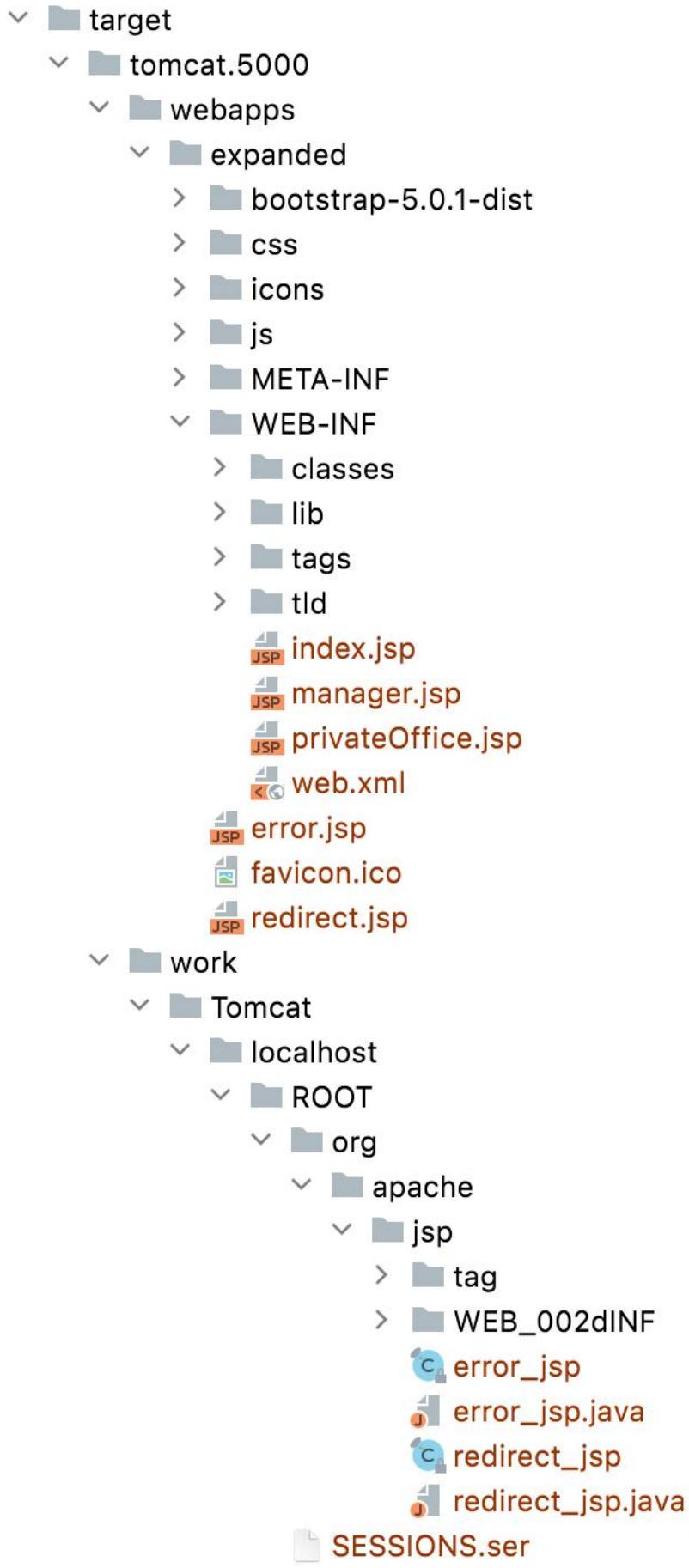


Рис. 3.6. Структура робочого каталогу.

3.4. Структура PostgreSQL

Важлива особливість для цього проекту, для неймінгу таблиць в продакшн базі даних, яка була згенерована у Heroku, було додано префікс cargo_, тому що в проекті використовується безкоштовна версія БД і в ній присутні недоліки, зокрема таблиці мають бути з унікальною назвою не тільки для окремого профілю, а й взагалі не повинні повторюватися у загальному плані для безкоштовного варіанту використання.

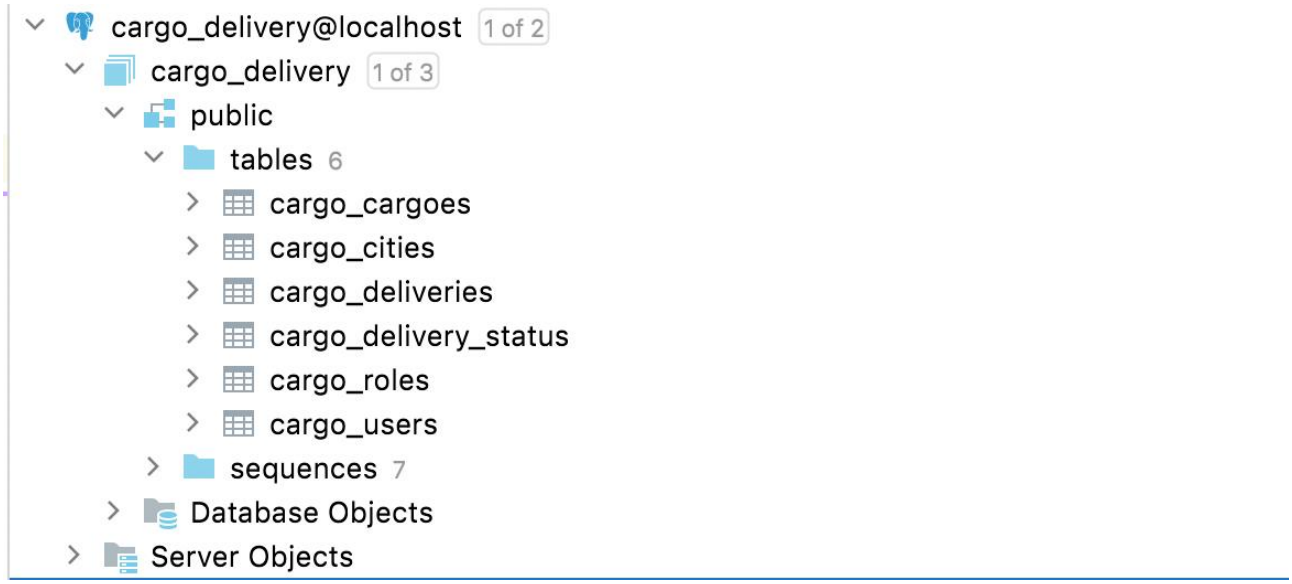


Рис. 3.7. Оглядова структура.

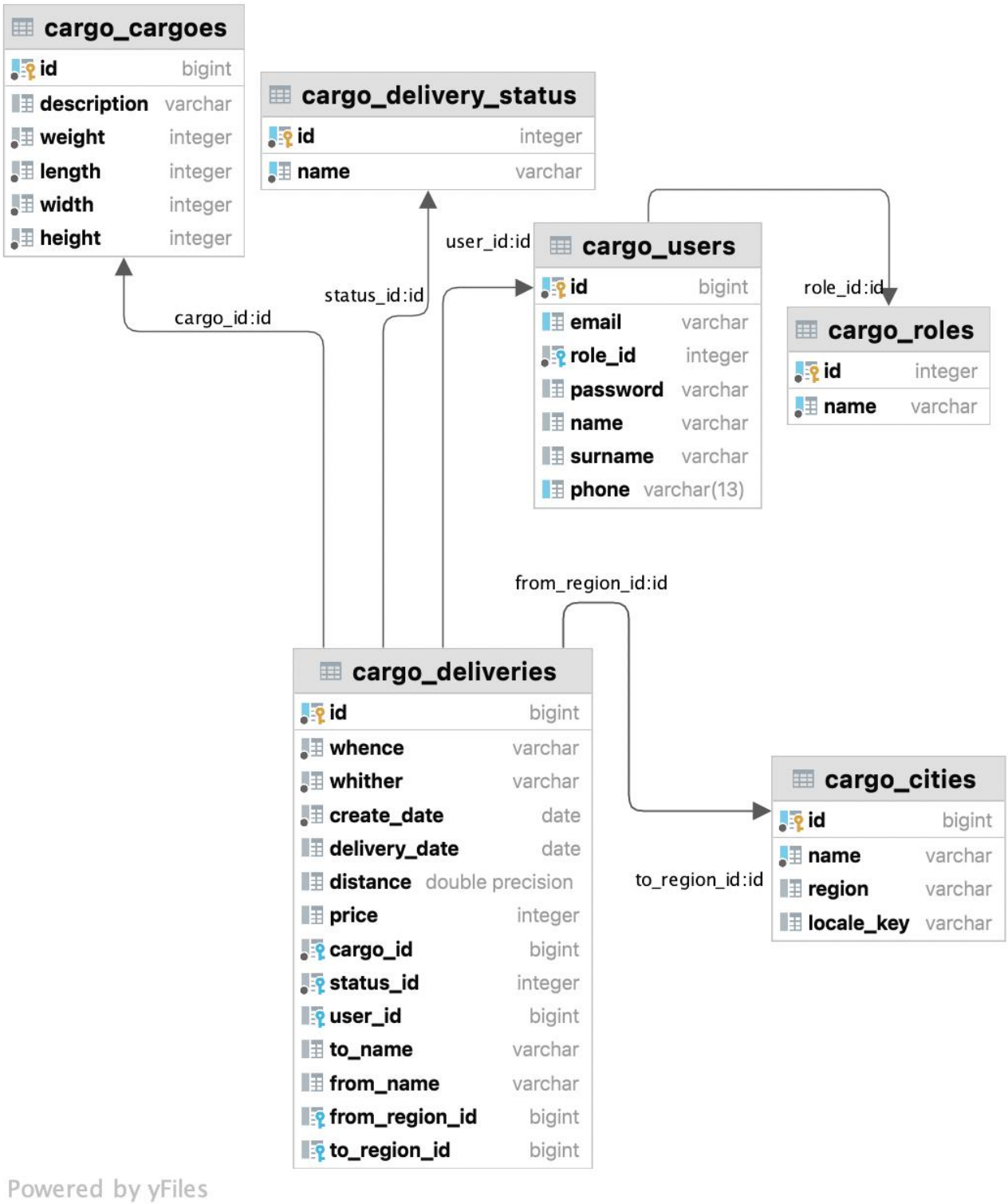


Рис. 3.8. Детальна структура БД.

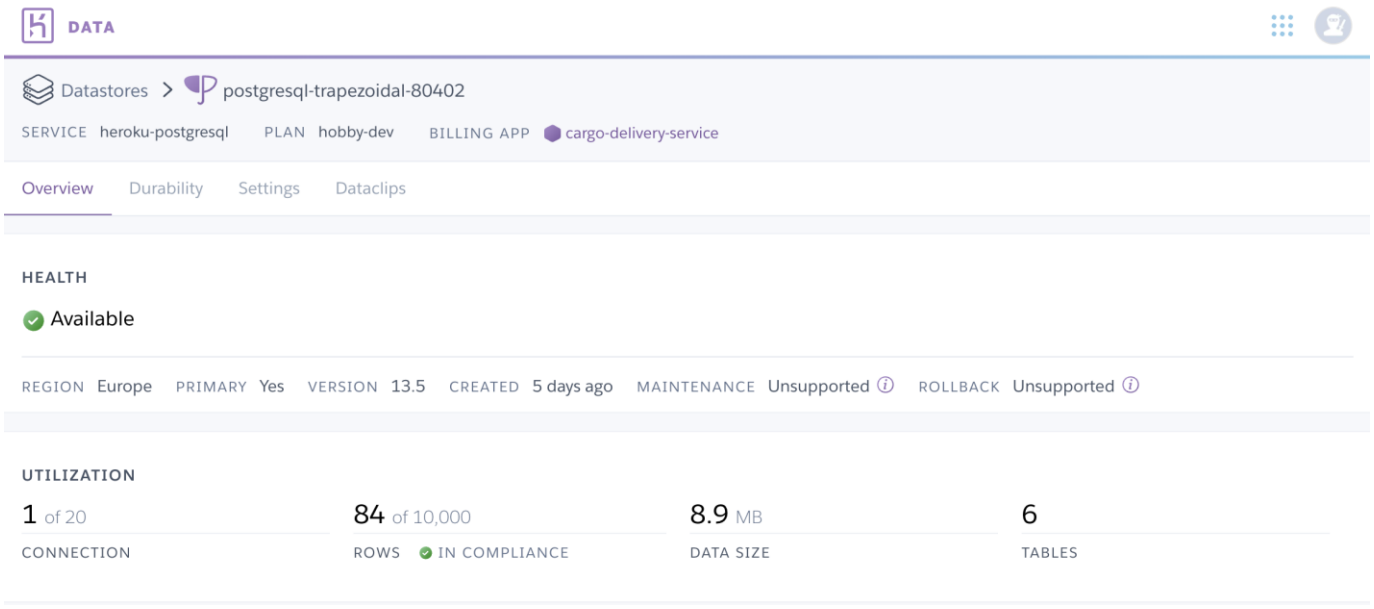


Рис. 3.9. Статус бази даних.

3.5. CI/CD

Heroku пропонує дуже зручний сервіс для деплою та доставки коду. При виборі цього варіанту автоматизації Heroku створює webhook, який сигналізує про push подію в систему GitHub і автоматизована система збірки забирає нову версію коду.

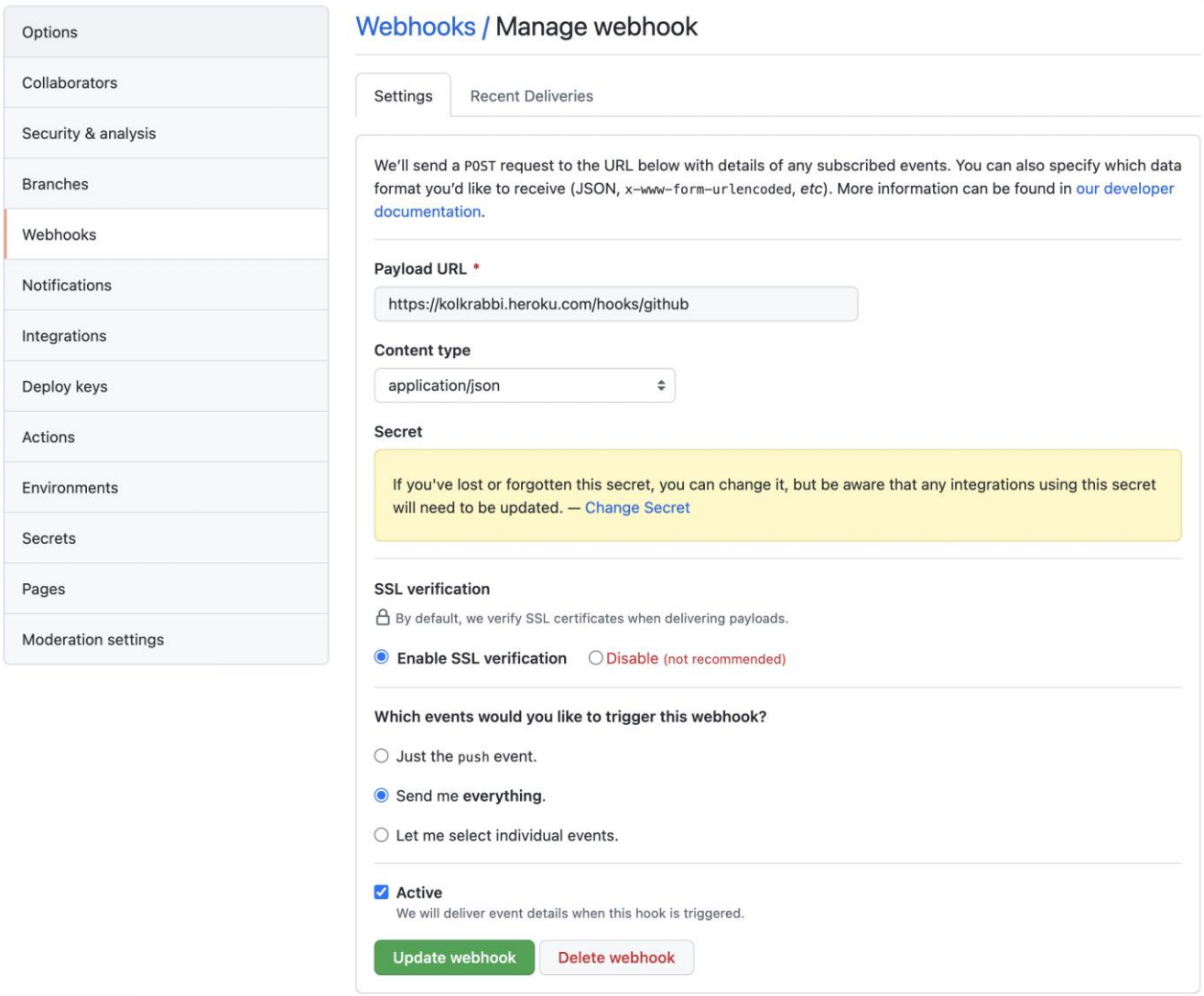


Рис. 3.10. Налаштування Webhook.

Було налаштовано наступний процес:

– Розробник зберігає код на гілці heroku в сервісі GitHub.

– Heroku забирає нову версію з GitHub після активації webhook-ом.

– Запускається процес збірки.

– Деплой на сервер .war файлу в Tomcat контейнер.

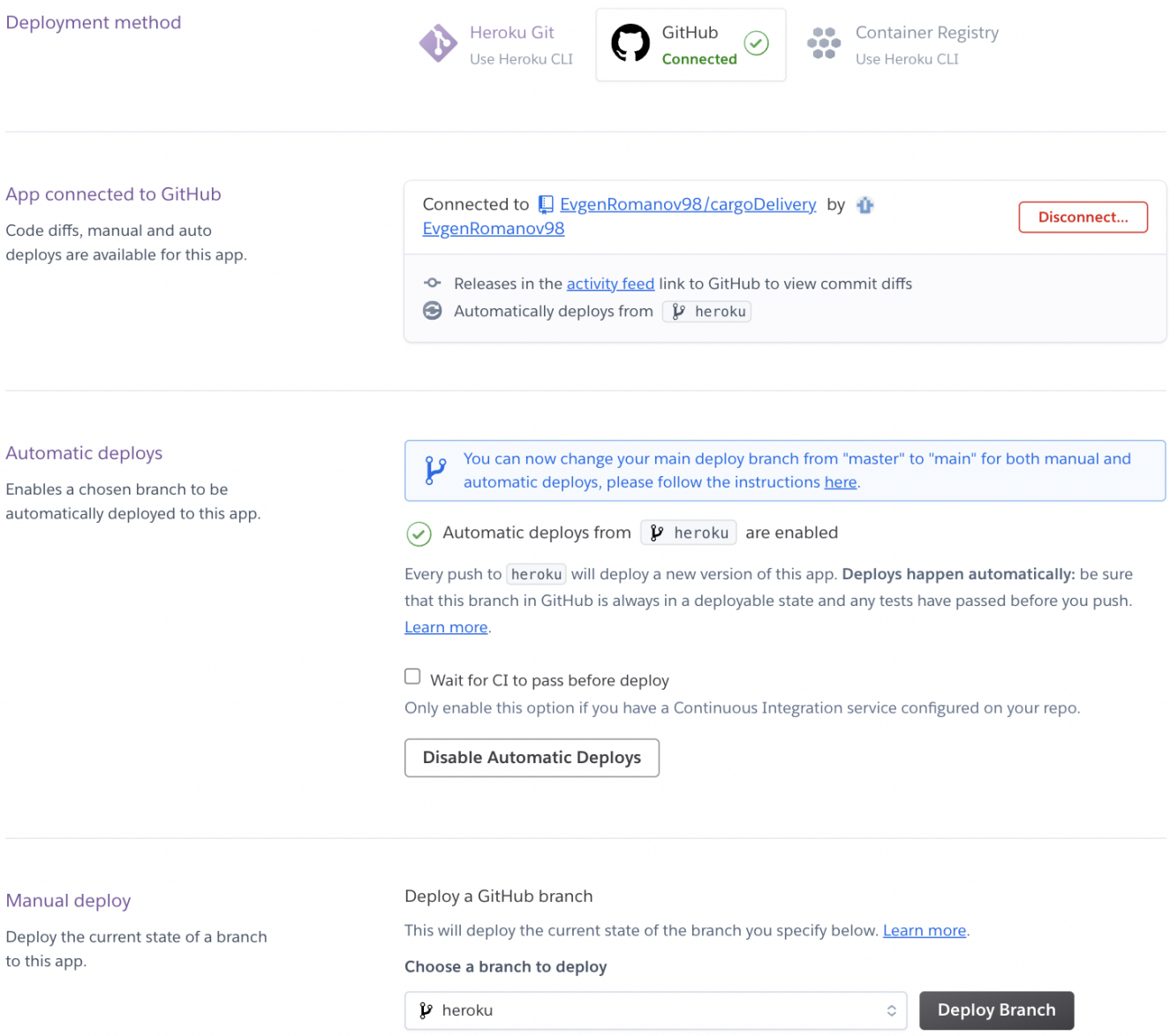


Рис. 3.11. Налаштований процес CI/CD.

3.6. Інтерфейс системи

Програма має браузерний інтерфейс і наразі виконує наступні функції:

– Реєстрація користувача.

– Авторизація користувача.

– Інформаційне табло по доставкам.

– Побудова маршруту для доставки з точки в точку.

– Реалізоване обмеження доступних міст для доставки.

– Витяг звітів для менеджера.

– Генерація квитанції по оплаті.

– Сторінка менеджера.

– Сортування при клікі на шапці таблиці, фільтрація у відповідному полі та пагінація на інформаційному табло.

– Локалізація (UA, RU, EN)

3.6.1. Реєстрація

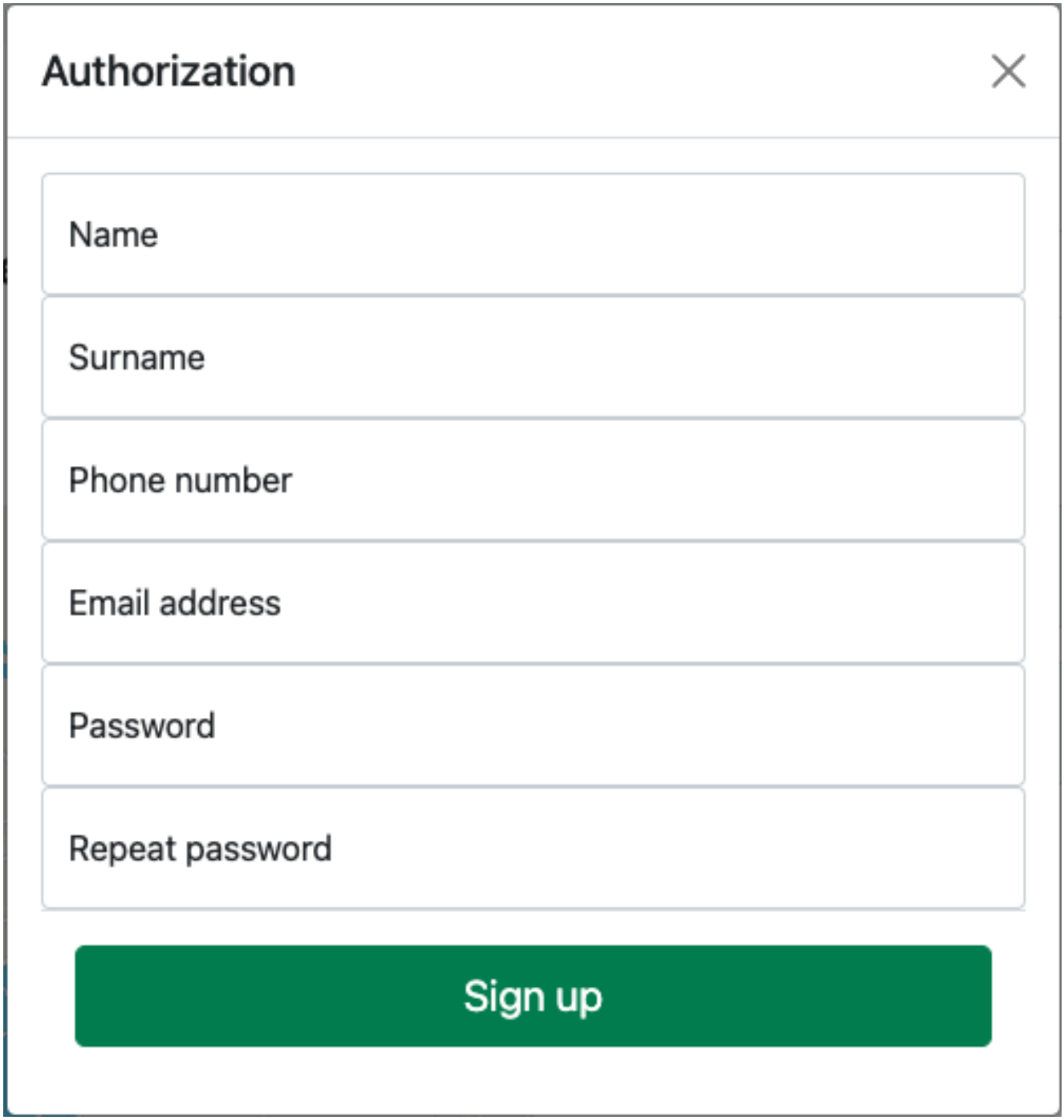


Рис. 3.12. Реєстраційна форма.

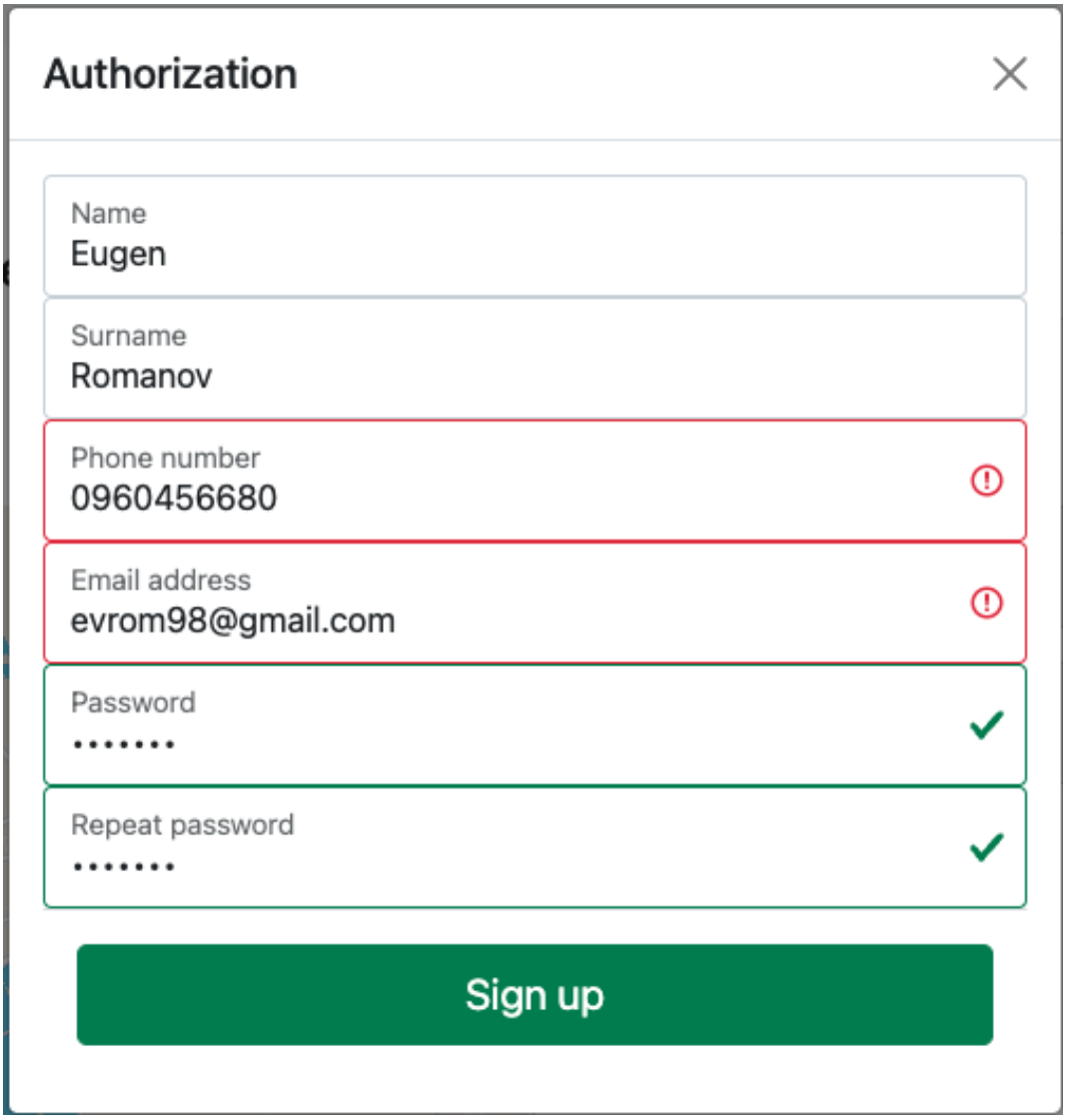


Рис. 3.13. Форма реєстрації з відображенням валідації.

Валідація номера телефона та пошти відбувається через протокол AJAX, без перезавантаження сторінки і в той самий час, коли користувач закінчив ввід цих даних. Це надзвичайно зручно і надає моментальний відгук користувачу про те, що щось не так і необхідно змінити вхідні дані. Також на клієнтській стороні працює перевірка пароля для запобігання неправильного повторного вводу.

3.6.2. Авторизація

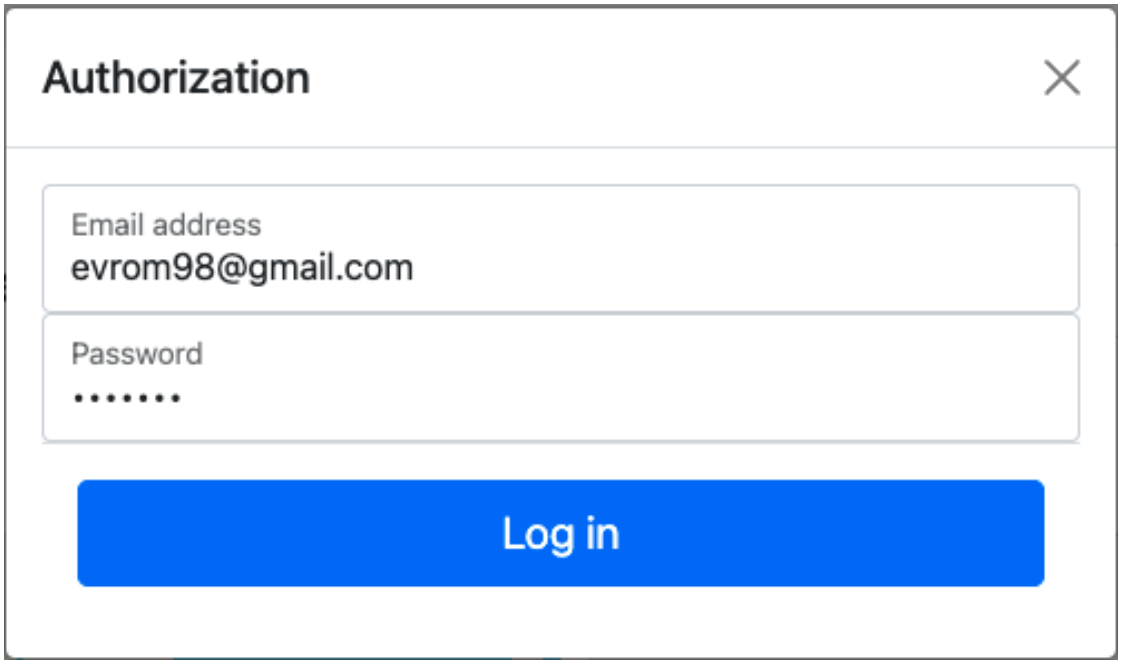


Рис. 3.14. Авторизаційна форма.

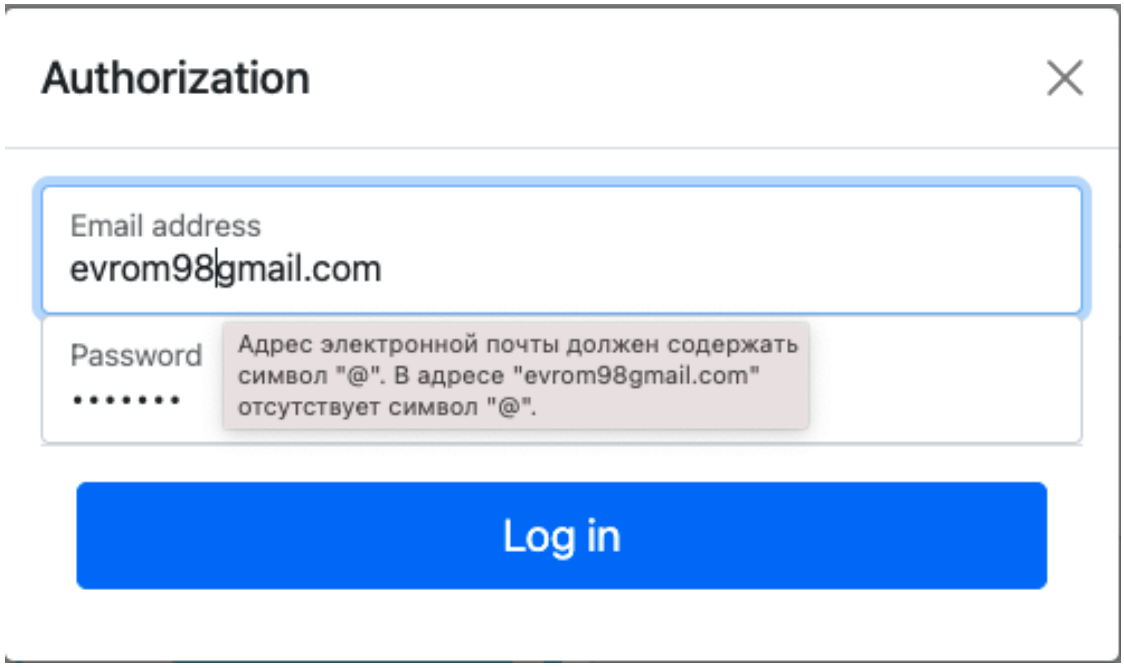


Рис. 3.15. Валідація на авторизаційній формі.

На формі для авторизації присутня стандартна браузерна валідація пошти, а також використана валідація за допомогою підходу регулярних виразі. Регулярний вираз (скорочений як regex або regexp; також згадується як раціональний вираз) — це послідовність символів, яка визначає шаблон

пошуку в тексті. Зазвичай такі шаблони використовуються алгоритмами пошуку рядків для операцій "знайти" або "знайти і замінити" над рядками або для перевірки введених даних. Це методика, розроблена в теоретичній інформатиці та теорії формальної мови.[24]

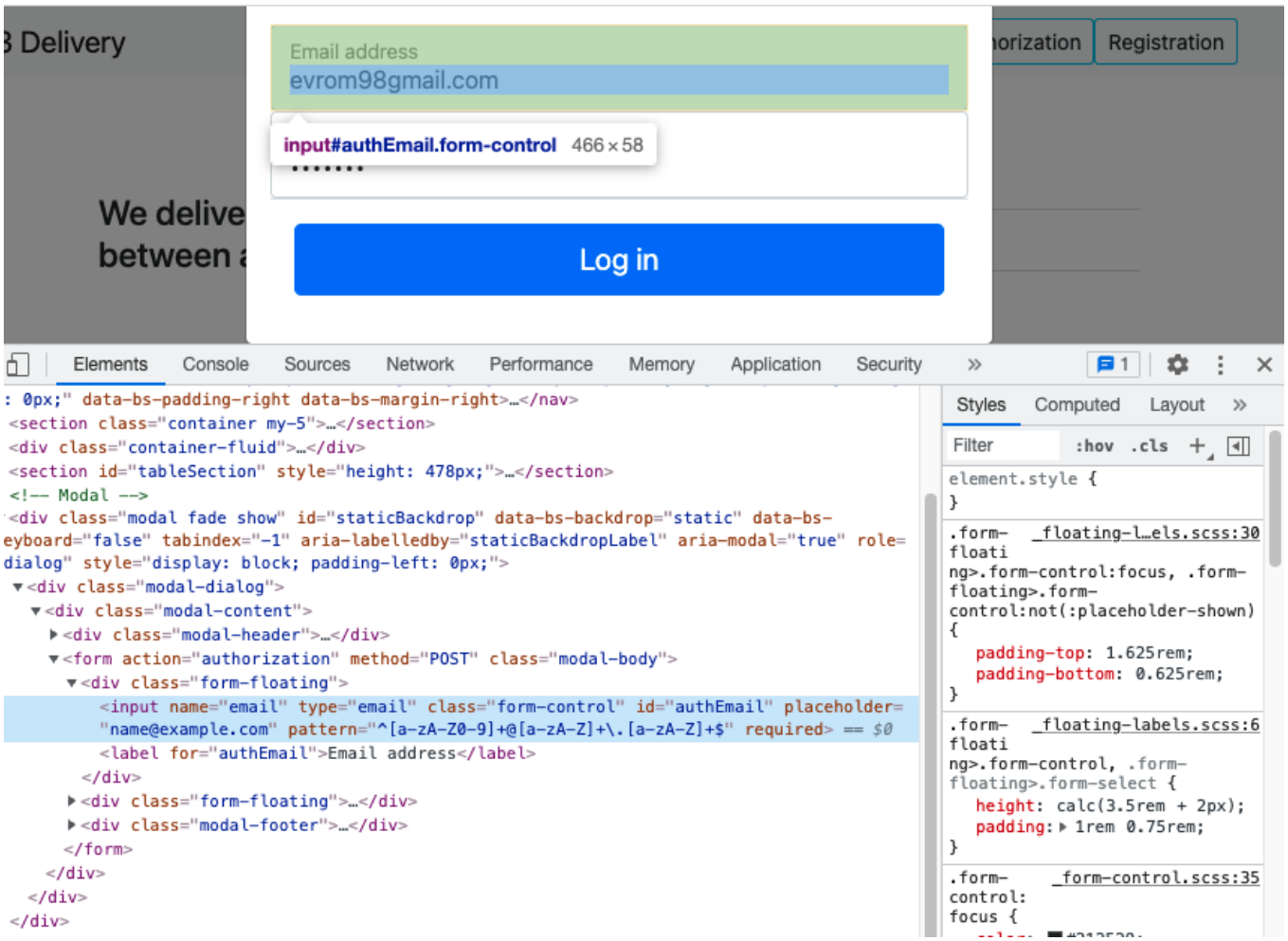


Рис. 3.16. Демонстрація роботи regex через інтерфейс розробника браузера.

Також після авторизації кнопка змінюється на ім’я та прізвище користувача і дає йому можливість перейти до власного кабінету.

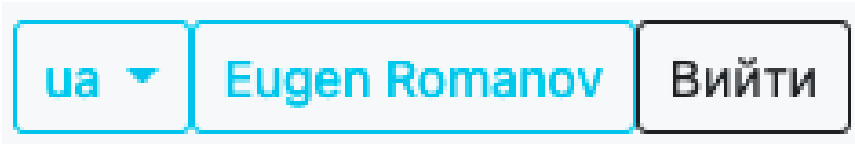


Рис. 3.17. Вхід до особистого кабінету.

3.6.3. Інформаційне табло доставок

Приклади доставок:

Звідки	Куди	Відстань, м	Ціна, грн
вул. Лісова, 1, Капітанівка, Киевская область, Украина 30.199817343570544, 50.44149062484871	Днепропетровская область, Украина 34.66198362006514, 48.2809033592298	500391.28	2692
Днепропетровская область, Украина 34.601598214288714, 48.445474470483305	Киевская область, Украина 30.137725381260537, 50.385950194003215	532136.5	3281
Запорожская область, Украина 36.3714898361217, 46.99737297788022	Днепропетровская область, Украина 34.879146500179274, 48.48454477343992	276638.06	1663
Днепропетровская область, Украина 34.885359780807335, 48.37436588647378	Запорожская область, Украина 35.38417599626396, 47.74363667688692	114379.3	1202
просп. Маяковского, 19, Запорожье, Запорожская область, Украина 35.11604117324106, 47.82806262361166	Волгоградская ул., 37, Киев, Киев, Украина 30.48818787415192, 50.424726146107446	564873.75	3144

« 1 2 3 ... 7 »

Рис. 3.18. Табло історії доставок.

При кліках на шапку таблиці, вона буде відсортована по тому критерію, на який було натиснуто. Фільтрація відбувається у відповідному полі у шапці таблиці по тому критерію, напроти якого розташовано це поле. Номер сторінки пагінації можна вибрати у правому нижньому куту таблиці.

3.6.4. Локалізація

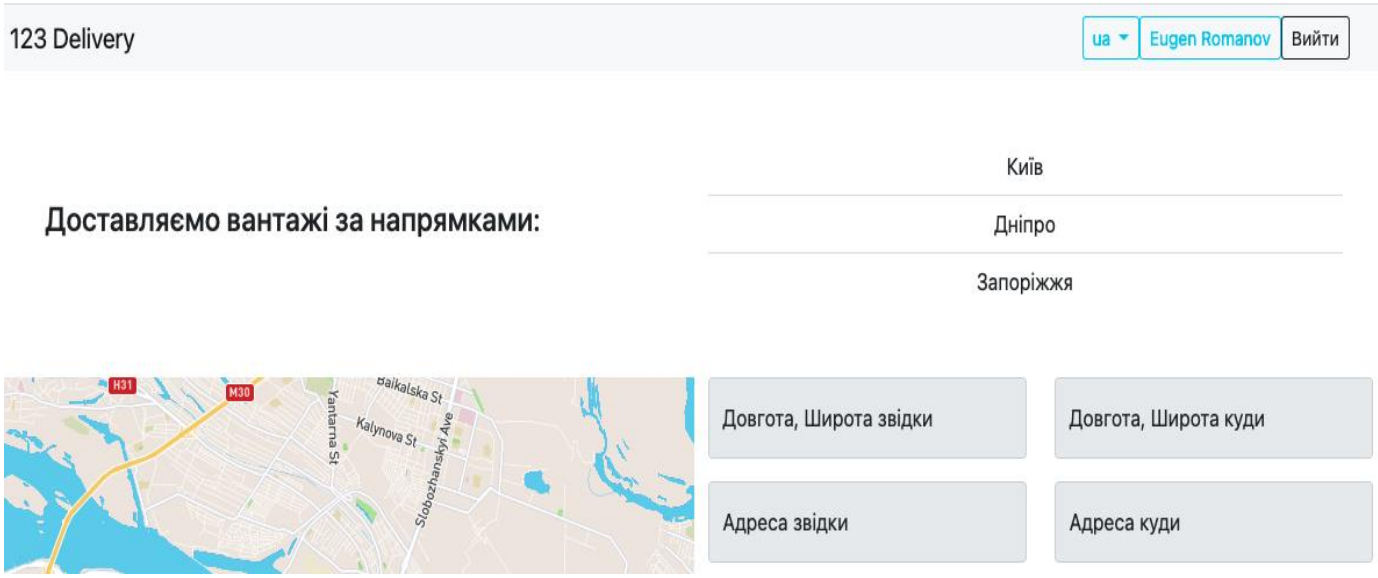


Рис. 3.19. Зміна мови інтерфейсу.

При натисканні на кнопку вгорі сайту з відображенням локалі можна змінити мову на українську, російську на англійську. Це зроблено за допомогою тегу fmt у JSP сторінках. До прикладу наступний тег

```
<fmt:message key="address.from"/>
```

При транслюванні JSP в сервлет Jasper візьме відповідне повідомлення з Resource Bundle у встановленій зараз локалі і виведе на сторінку користувача. Resource Bundle розташований у каталозі resources.

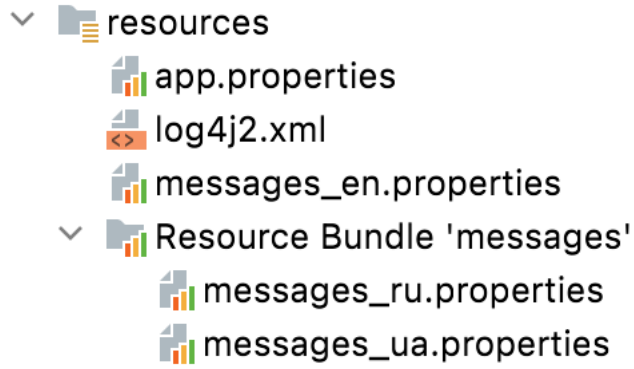


Рис. 3.20. Resource Bundle.

3.6.5. Побудова маршруту

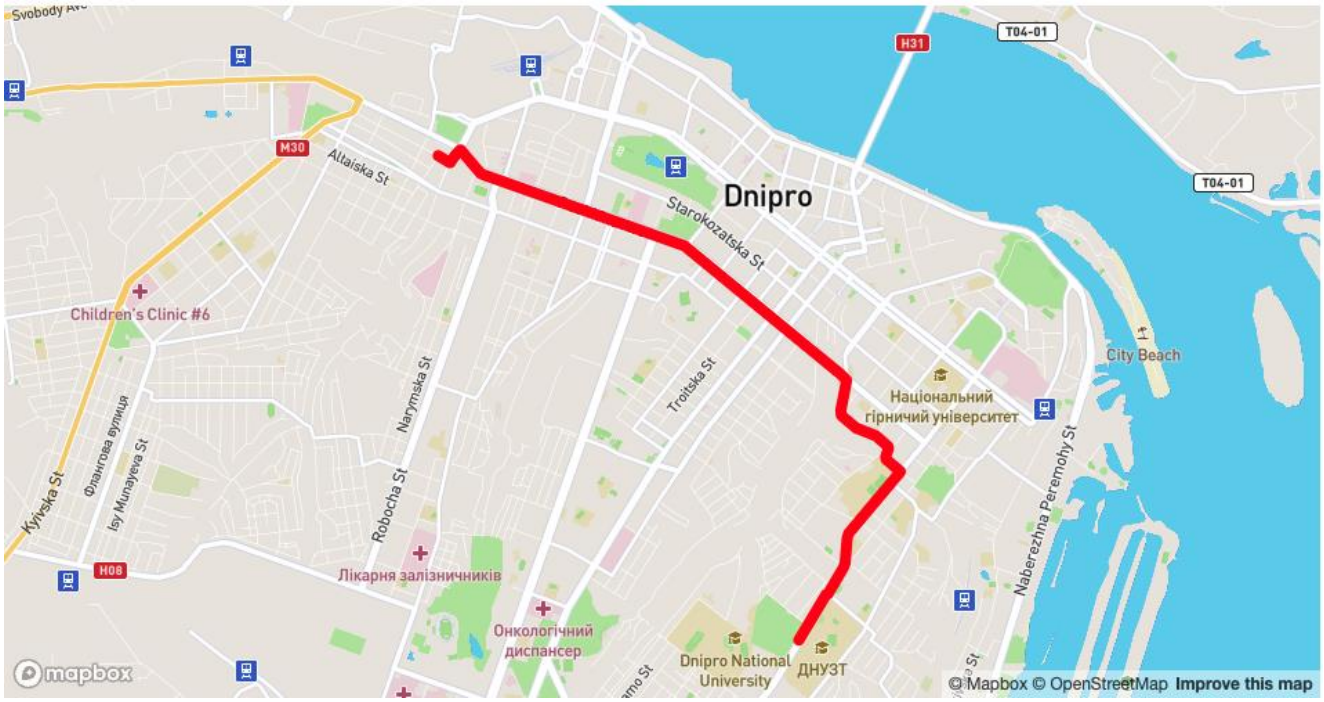


Рис. 3.21. Побудова маршруту на карті.

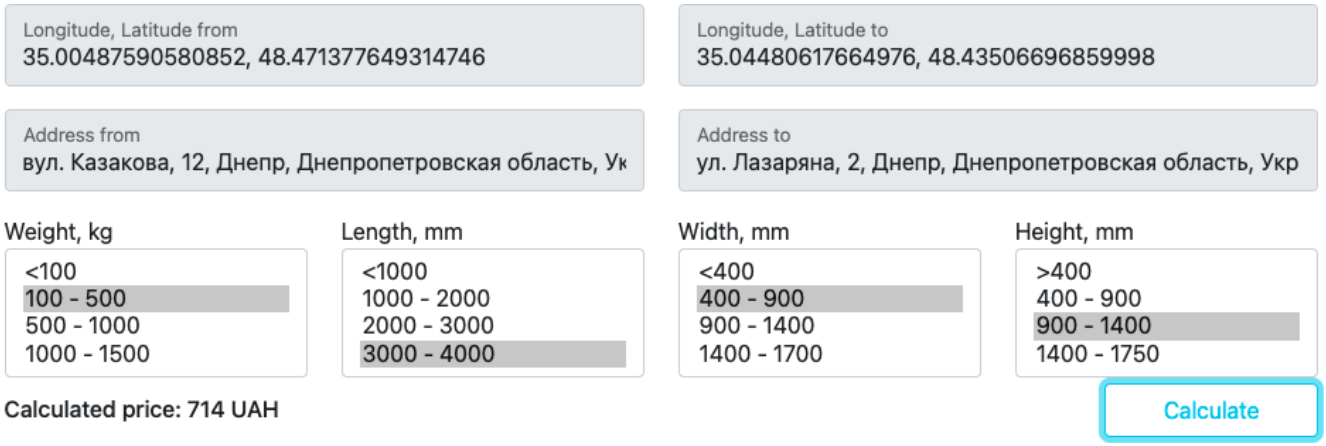


Рис. 3.22. Відображення інформації за маршрутом.

Алгоритм дії користувача:

– Вибрати точку відправки та прибуття. Для цього потрібно на інтерактивній карті клікнути перший раз на точку відправки, наступний на прибуття. Автоматично підтягнеться інформація по адресі, в яке місце відбувся клік мишкою.

– Вибрати вагу та габарити вантажу.

– Натиснути кнопку Розрахувати(Calculate)

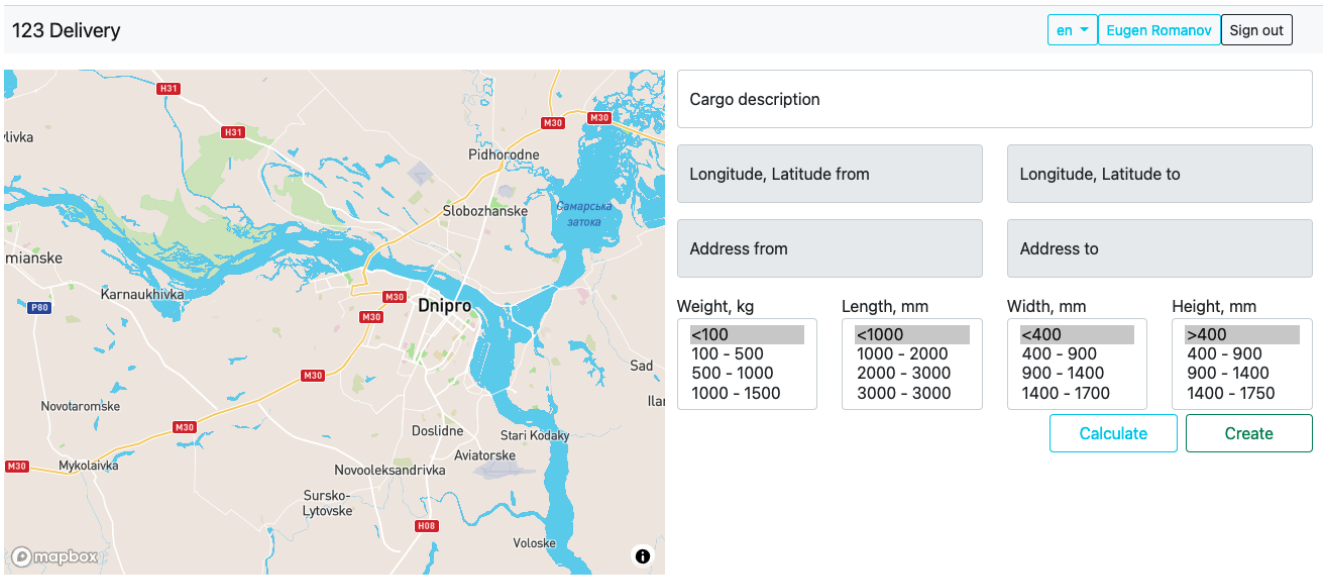
Після цього на інтерактивній карті з’явиться червона лінія маршруту доставки та ціна за доставку.

3.6.6. Наукова новизна

Наукова новизна проекту полягає саме в комунікації клієнтів та виконавців вантажних перевезень через цю інтерактивну карту. Це дозволяє і клієнтам і виконавцям бути в одному середовищі при замовленні послуг та при їх виконанні.

3.6.7. Особистий кабінет

При вході в особистий кабінет для користувача інтерфейс майже не змінюється. Лише змінюється табло по доставкам, воно містить лише інформацію доставок авторизованого клієнта. Також для нього доступно більше інформації.



Your deliveries:

Description	From	To	Create date	Delivery date	Distance, m	Price, UAH	Weight, kg	Length, mm	Width, mm	Height, mm	Status	Info
test7	Днепр, Днепропетровская область, Украина 35.00715336460058,	просп. Карла Маркса, 19, Днепр, Днепропетровская область, Украина	2021-06-14		4775.965	769	1000	3000	1700	1750	CREATED	Wait manager's approve

Рис. 3.23. Особистий кабінет.

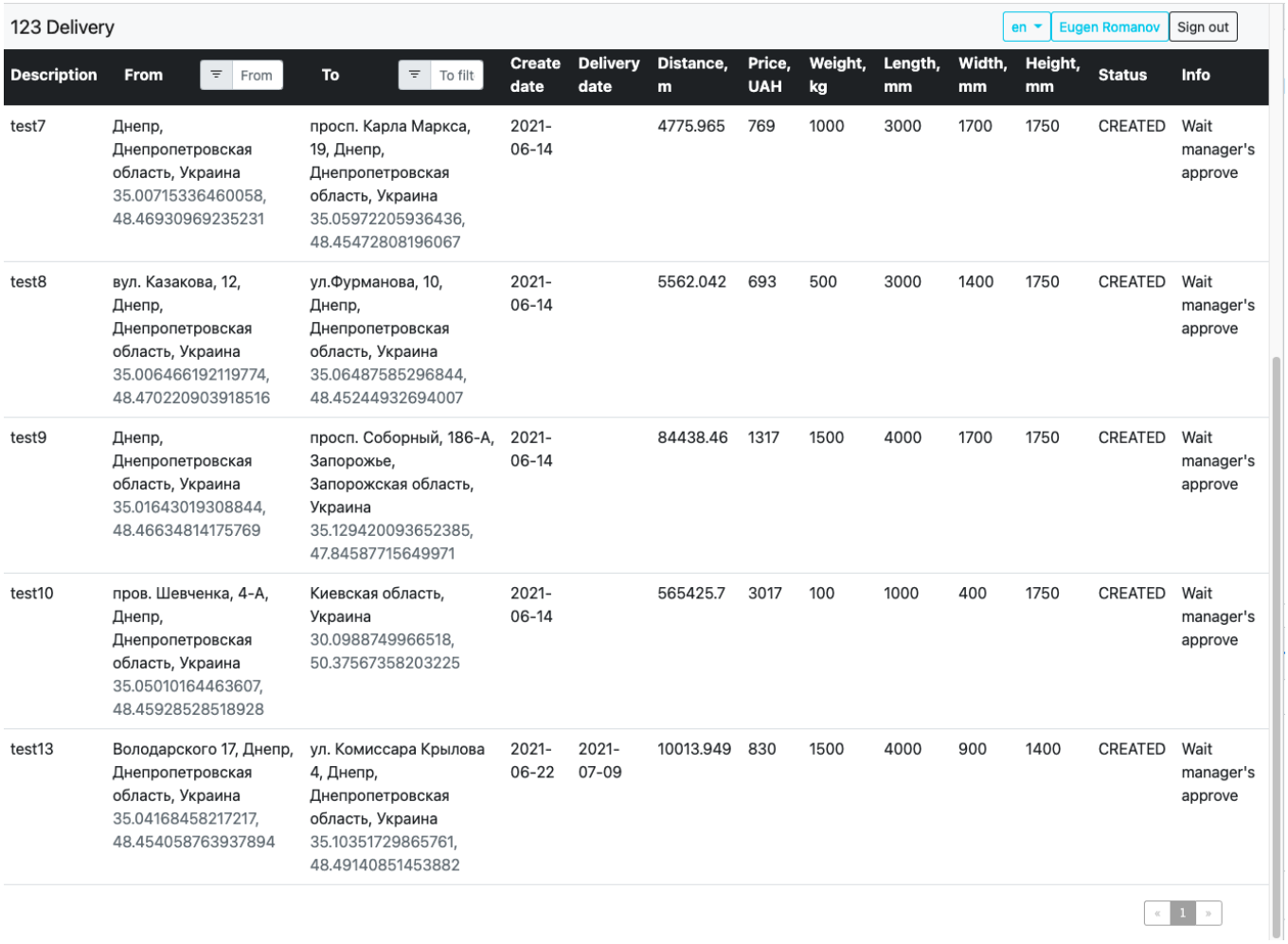


Рис. 3.24. Інформаційне табло в особистому кабінеті.

Додається колонка статусу замовлення. Можливі статуси:

– Створений – означає, що замовлення збережено в системі та надалі буде розглянуто менеджером.

– Підтверджений – менеджер розглянув замовлення, провів комунікацію з клієнтом, виставив дату доставки та підтвердив її. Після підтвердження клієнт в особистому кабінеті має можливість оплати доставки. Також якщо доставок декілька, є можливість оплатити все одним кліком внизу таблиці

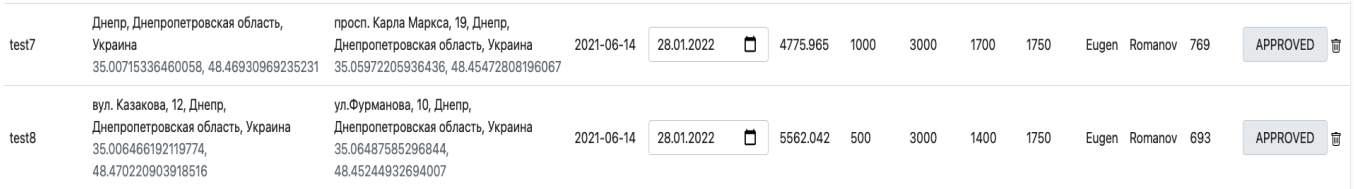


Рис. 3.25. Менеджер підтвердив замовлення.

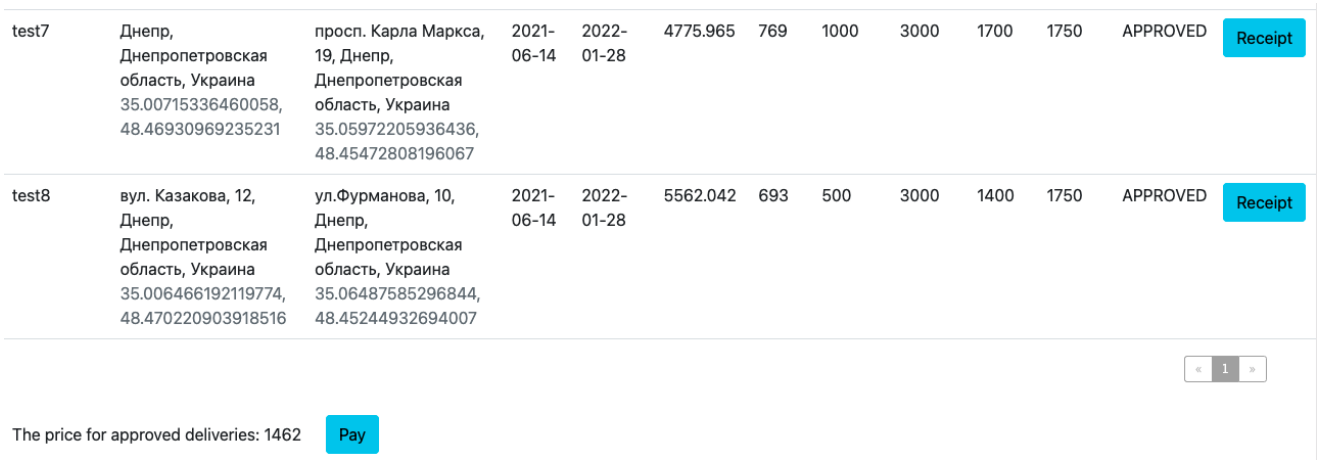


Рис. 3.26. Підтверджене замовлення.

– Оплачено – користувач оплатив замовлення, статус змінюється автоматично. Після цього користувач може згенерувати для себе квитанцію про оплату.

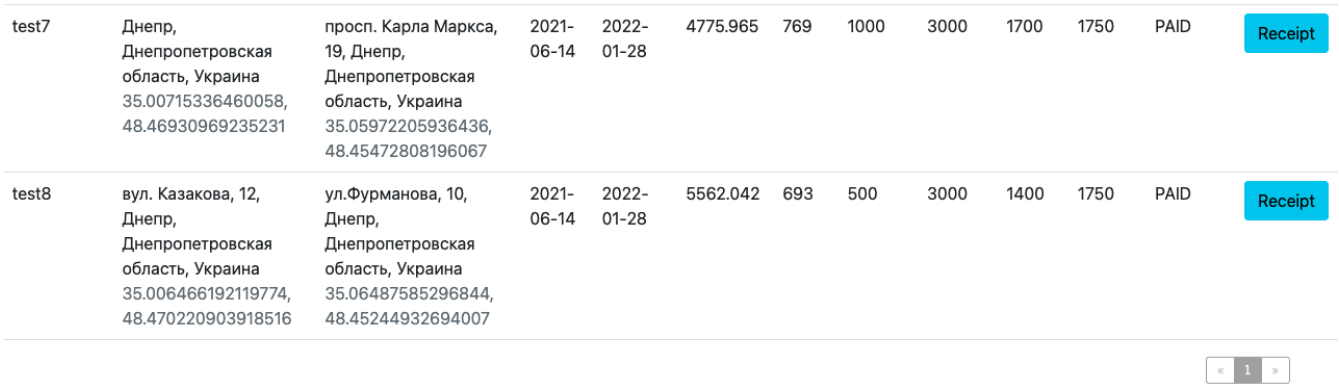


Рис. 3.27. Оплачені замовлення.

Рис. 3.28. Інтерфейс менеджера. Оплачені замовлення.

123 Delivery

Receipt from: 2022-01-16 22:33:00

Customer: Eugen Romanov

Delivery id: 52

status	PAID
price	769
Distance	4775.965
DeliveryDate	2022-01-28
From	Днепр, Днепропетровская область, Украина
To	просп. Карла Маркса, 19, Днепр, Днепропетровская область, Украина
From (latitude, longitude)	35.00715336460058, 48.46930969235231
To (latitude, longitude)	35.05972205936436, 48.45472808196067

Thanks for choosing us!

Рис. 3.29. Згенерована квитанція в форматі PDF.

Квитанція формується за допомогою бібліотеки OpenPDF.

– Доставляється – менеджер виставляє статус коли вантаж знаходиться в шляху до адресата. Також це можна спостерігати в особистому кабінеті.

test7	Днепр, Днепропетровская область, Украина 35.00715336460058, 48.46930969235231	просп. Карла Маркса, 19, Днепр, Днепропетровская область, Украина 35.05972205936436, 48.45472808196067	2021-06-14	2022-01-28	4775.965	769	1000	3000	1700	1750	TRANSFER	Receipt
test8	вул. Казакова, 12, Днепр, Днепропетровская область, Украина 35.006466192119774, 48.470220903918516	ул.Фурманова, 10, Днепр, Днепропетровская область, Украина 35.06487585296844, 48.45244932694007	2021-06-14	2022-01-28	5562.042	693	500	3000	1400	1750	TRANSFER	Receipt

Рис. 3.30. Доствка в статусі TRANSFER

– Доставлено – менеджер виставляє статус, коли вантаж доставлений.

123 Delivery

en | Eugen Romanov | Sign out

Description	From	To	Create date	Delivery date	Distance, m	Price, UAH	Weight, kg	Length, mm	Width, mm	Height, mm	Status	Info
test9	Днепр, Днепропетровская область, Украина 35.01643019308844, 48.46634814175769	просп. Соборный, 186-А, Запорожье, Запорожская область, Украина 35.129420093652385, 47.84587715649971	2021-06-14		84438.46	1317	1500	4000	1700	1750	CREATED	Wait manager's approve
test10	пров. Шевченка, 4-А, Днепр, Днепропетровская область, Украина 35.05010164463607, 48.45928528518928	Киевская область, Украина 30.0988749966518, 50.37567358203225	2021-06-14		565425.7	3017	100	1000	400	1750	CREATED	Wait manager's approve
test13	Володарского 17, Днепр, Днепропетровская область, Украина 35.04168458217217, 48.454058763937894	ул. Комиссара Крылова 4, Днепр, Днепропетровская область, Украина 35.10351729865761, 48.49140851453882	2021-06-22	2021-07-09	10013.949	830	1500	4000	900	1400	CREATED	Wait manager's approve
test7	Днепр, Днепропетровская область, Украина 35.00715336460058, 48.46930969235231	просп. Карла Маркса, 19, Днепр, Днепропетровская область, Украина 35.05972205936436, 48.45472808196067	2021-06-14	2022-01-28	4775.965	769	1000	3000	1700	1750	DELIVERED	Receipt
test8	вул. Казакова, 12, Днепр, Днепропетровская область, Украина 35.006466192119774, 48.470220903918516	ул.Фурманова, 10, Днепр, Днепропетровская область, Украина 35.06487585296844, 48.45244932694007	2021-06-14	2022-01-28	5562.042	693	500	3000	1400	1750	DELIVERED	Receipt

« 1 »

https://cargo-delivery-service.herokuapp.com

Рис. 3.31. Вантаж доставлено.

– Видалений – також замовлення може бути видалено менеджером.

Для цього йому необхідно натиснути на іконку смітника напроти замовлення. Воно перестає відображатися в особистому кабінеті користувача і менеджера, але залишається в базі даних для статистики в статусі DELETED.

Description	From	To	Create date	Delivery date	Distance, m	Price, UAH	Weight, kg	Length, mm	Width, mm	Height, mm	Status	Info
test9	Днепр, Днепропетровская область, Украина 35.01643019308844, 48.46634814175769	просп. Соборный, 186-А, Запорожье, Запорожская область, Украина 35.129420093652385, 47.84587715649971	2021-06-14		84438.46	1317	1500	4000	1700	1750	CREATED	Wait manager's approve
test10	пров. Шевченка, 4-А, Днепр, Днепропетровская область, Украина 35.05010164463607, 48.45928528518928	Киевская область, Украина 30.0988749966518, 50.37567358203225	2021-06-14		565425.7	3017	100	1000	400	1750	CREATED	Wait manager's approve
test13	Володарского 17, Днепр, Днепропетровская область, Украина 35.04168458217217, 48.454058763937894	ул. Комиссара Крылова 4, Днепр, Днепропетровская область, Украина 35.10351729865761, 48.49140851453882	2021-06-22	2021-07-09	10013.949	830	1500	4000	900	1400	CREATED	Wait manager's approve

« 1 »

Рис. 3.32. Видалені замовлення.

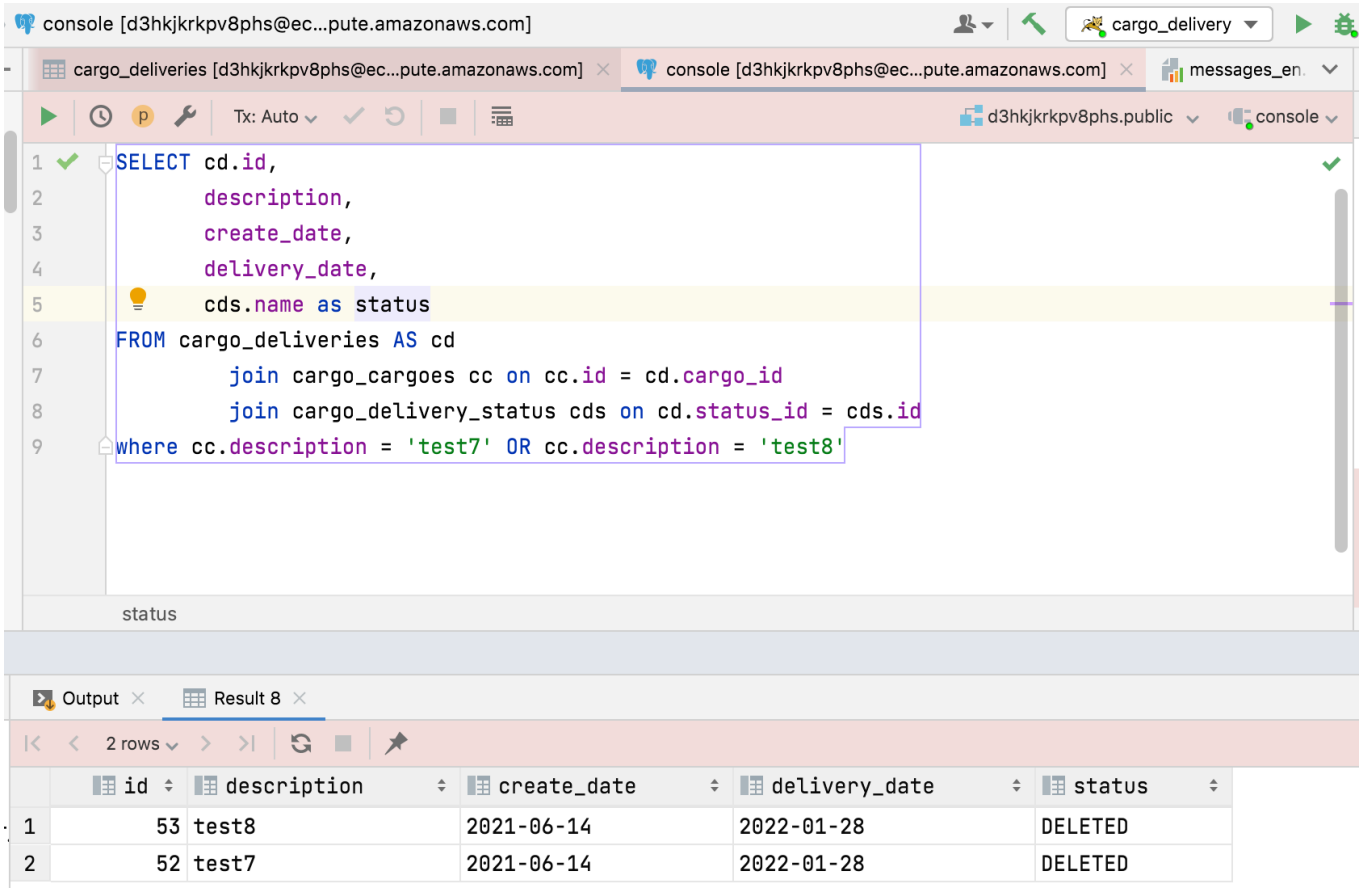


Рис. 3.33. Записи в БД.

3.6.8. Інтерфейс менеджера

Зараз в БД зареєстрований 1 менеджер з логіном m@m.com. На рисунку 2.33. зображений інтерфейс менеджера.

Особливості роботи інтерфейсу:

– Менеджер не зможе змінити статус доставки поки не виставить дату доставки.

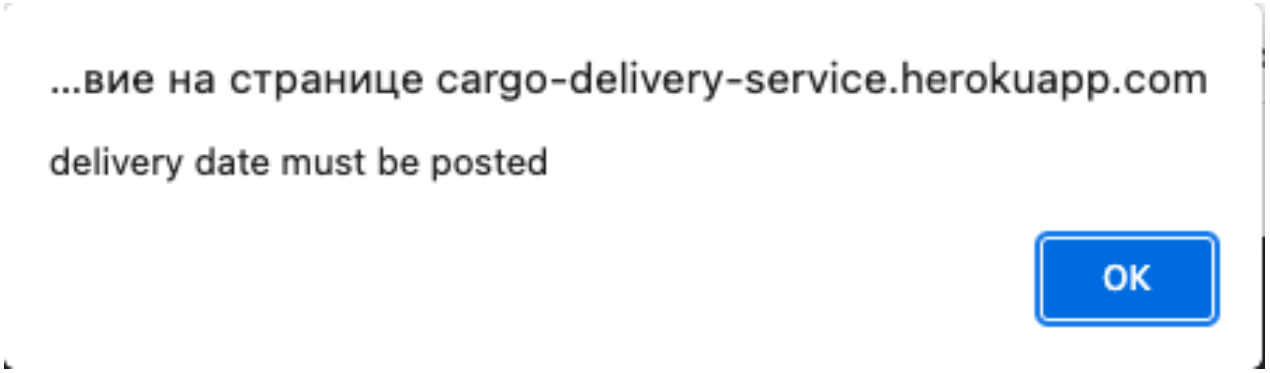


Рис. 3.34. Помилка при спробі змінити статус без дати.

– Для запобігання випадкових натискань менеджеру необхідно 2 рази натиснути по клавіщі зміни статусу для її активації.

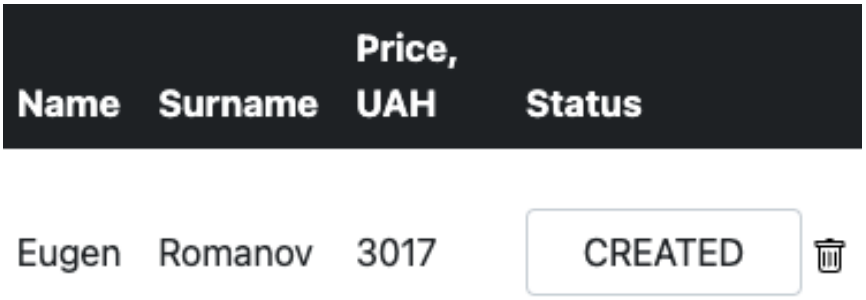


Рис 3.31. Активована клавіша зміни статусу.

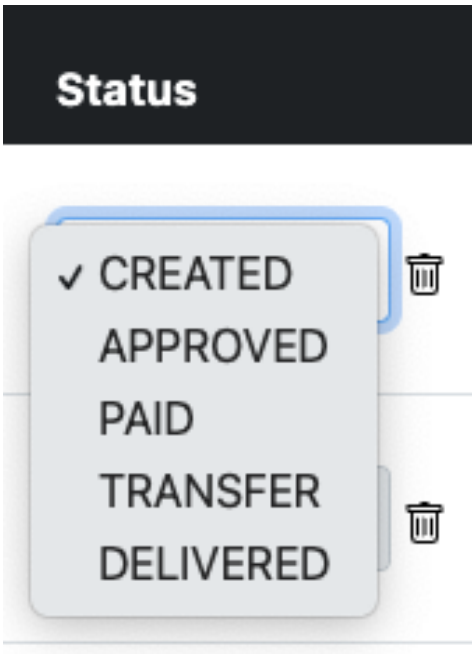


Рис. 3.35. Варіанти зміни статусу.

– Після вибору дати доставки менеджеру необхідно підтвердити замовлення вибравши відповідний статус замовлення. Статус замовдення оновлюється за допомогою AJAX, то ж йому не потрібно робити додаткових дій для зберігання оновлення.

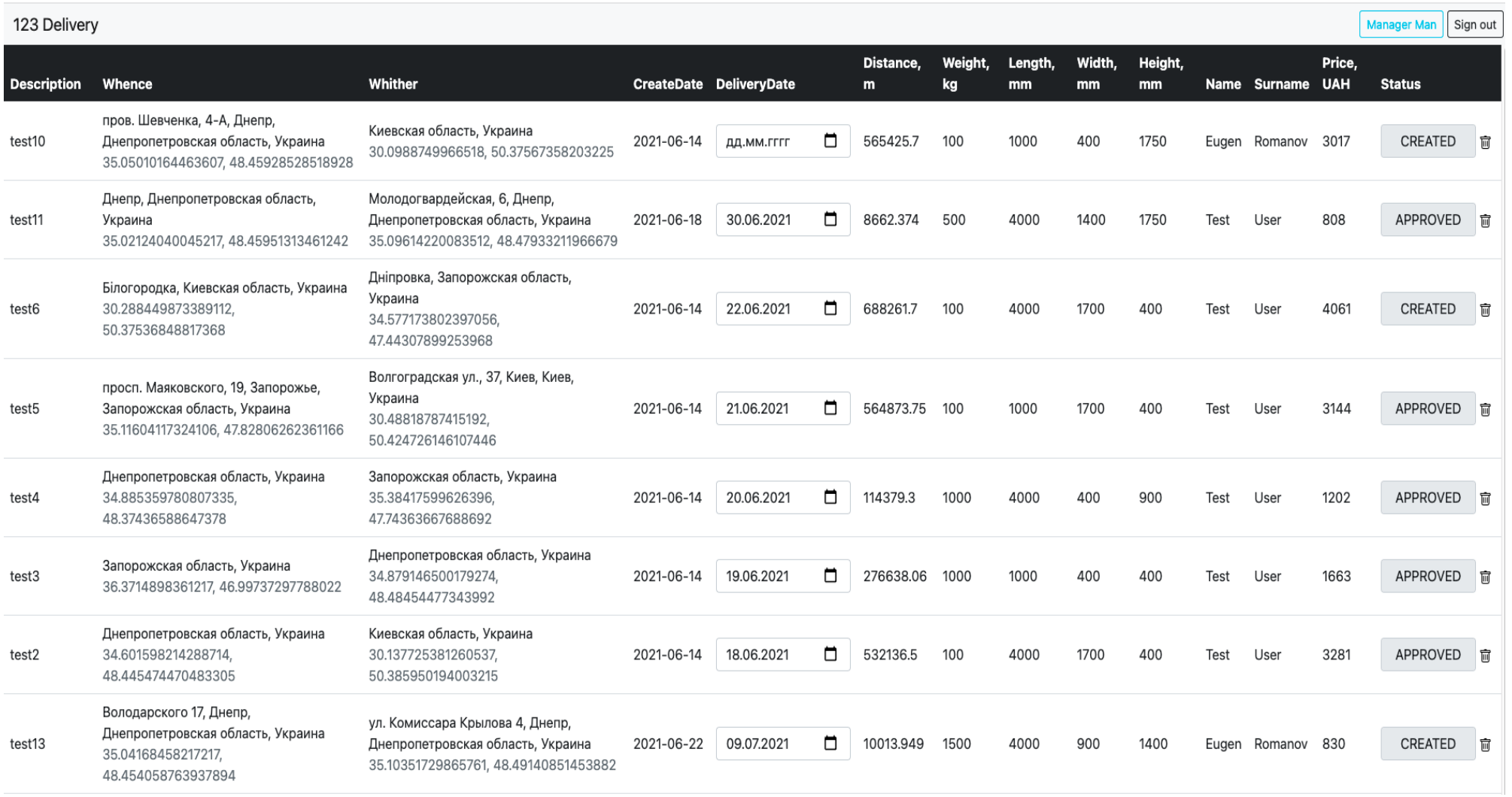


Рис. 3.36. Інтерфейс менеджера.

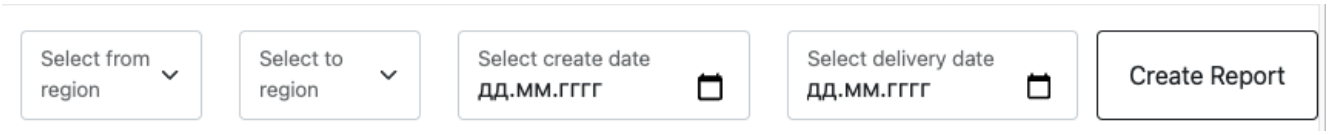


Рис. 3.37. Інтерфейс генерації звітів для менеджера.

Менеджер може генерувати звіти за допомогою інтерфейсу, який розташований внизу сторінки. Йому необхідно вибрати 4 варіанти фільтрації:

– Куди були відправлення

– Звідки

– Дата створення заявки

– Дата доставки

За бажанням можна вибрати декілька параметрів, не обов’язково всі. Якщо не вибрати жодного, то буде вивантажена вся таблиця:

D14 | 34.99547143243086, 48.454500211060775

	A	B	C	D	E
1	Id	Description	From	From location	To
2	56	test11	Днепр, Днепропетровская область, Украина	35.02124040045217, 48.45951313461242	Молодогвардейская, 6, Днепр, Днепропетровс
3	50	test5	просп. Маяковского, 19, Запорожье, Запорожская область, Украина	35.11604117324106, 47.82806262361166	Волгоградская ул., 37, Киев, Киев, Украина
4	49	test4	Днепропетровская область, Украина	34.885359780807335, 48.37436588647378	Запорожская область, Украина
5	48	test3	Запорожская область, Украина	36.3714898361217, 46.99737297788022	Днепропетровская область, Украина
6	47	test2	Днепропетровская область, Украина	34.601598214288714, 48.445474470483305	Киевская область, Украина
7	61	test13	Володарского 17, Днепр, Днепропетровская область, Украина	35.04168458217217, 48.454058763937894	ул. Комиссара Крылова 4, Днепр, Днепропетр
8	60	test	вул. Михайла Грушевського, 65, Днепр, Днепропетровская область, Украина	35.03498395644206, 48.45167992531097	Днепр, Днепропетровская область, Украина
9	62	rrt	ул. Молодогвардейская 6, корпус 29, Днепр, Днепропетровская область, Украина	35.10729596466456, 48.48252905325492	Днепр, Днепропетровская область, Украина
10	46	test1	вул. Лісова, 1, Капітанівка, Киевская область, Украина	30.199817343570544, 50.44149062484871	Днепропетровская область, Украина
11	65	тест на русском	ул. Набережная Заводская, 53, Днепр, Днепропетровская область, Украина	34.95340120363517, 48.48047972601873	Днепр, Днепропетровская область, Украина
12	59	description	ул. Орловская, 1Б, Днепр, Днепропетровская область, Украина	34.99634059008295, 48.4734202984512	Клары цеткин, 12, Днепр, Днепропетровская о
13	57	test12	Киевская область, Украина	30.509541966792682, 50.32521163903144	Днепропетровская область, Украина
14	58	test12	Днепр, Днепропетровская область, Украина	34.99547143243086, 48.454500211060775	Днепр, Днепропетровская область, Украина
15	54	test9	Днепр, Днепропетровская область, Украина	35.01643019308844, 48.46634814175769	просп. Соборный, 186-А, Запорожье, Запорож
16	63	rrrr	вул. Леніна, 46, Днепр, Днепропетровская область, Украина	35.03481428034098, 48.46157647207664	Національний Гірничий Університет, Днепр, Д
17	64	кпап	ул. Маршала Малиновского, 58, Днепр, Днепропетровская область, Украина	35.08668505917004, 48.48207365435729	ул. Благоева, 7, Днепр, Днепропетровская обл
18	66	dffdfdfdfdf	вул. Артільна, 11, Днепр, Днепропетровская область, Украина	35.08187584788803, 48.48913187780482	Днепропетровская область, Украина
19	51	test6	Білогородка, Киевская область, Украина	30.288449873389112, 50.37536848817368	Дніпровка, Запорожская область, Украина
20	53	test8	вул. Казакова, 12, Днепр, Днепропетровская область, Украина	35.006466192119774, 48.470220903918516	ул.Фурманова, 10, Днепр, Днепропетровская о
21	55	test10	пров. Шевченка, 4-А, Днепр, Днепропетровская область, Украина	35.05010164463607, 48.45928528518928	Киевская область, Украина
22	67	dfdasqs222	Днепр, Днепропетровская область, Украина	35.193601992411914, 48.48060357006213	Днепр, Днепропетровская область, Украина
23	68	test14	вул. Кам'янська, 40, Днепр, Днепропетровская область, Украина	34.9656598436556, 48.46519090717848	Запорожская область, Украина
24	69	test15	Киевская область, Украина	30.147263503379776, 50.46834884630351	М Славутич, Киев, Киев, Украина
25	70		Білогородка, Киевская область, Украина	30.22631841240363, 50.40823809975561	Запорожская область, Украина
26	81		Леваневского, 13, Днепр, Днепропетровская область, Украина	35.00252386173139, 48.470915104153335	вул. Сірка, 45, Днепр, Днепропетровская обла
27	78	eee	просп. Дмитра Яворницького, 19, Днепр, Днепропетровская область, Украина	35.061341973633915, 48.45500820099815	Днепр, Днепропетровская область, Украина
28	77	testdd	вул. Казакова, 12, Днепр, Днепропетровская область, Украина	35.00503704199309, 48.469578550380874	вул. Криворізька, 1, Днепр, Днепропетровская
29	79		Днепр, Днепропетровская область, Украина	35.031472894043674, 48.46092790986455	Днепр, Днепропетровская область, Украина
30	80		Молодогвардейская, 6, Днепр, Днепропетровская область, Украина	35.094300958009114, 48.48232417525327	Днепр, Днепропетровская область, Украина
31	82	test	просп. Пушкина, 77а, Днепр, Днепропетровская область, Украина	35.00870713337986, 48.468637547459224	Ул.Янтарная, Днепр, Днепропетровская облас
32	52	test7	Днепр, Днепропетровская область, Украина	35.00715336460058, 48.46930969235231	просп. Карла Маркса, 19, Днепр, Днепропетро

Рис. 3.38. Вивантаження всіх записів.

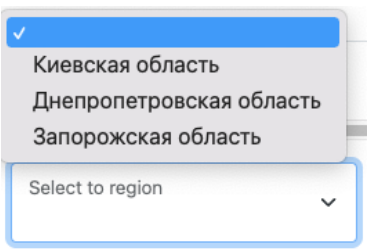


Рис. 3.39. Вибір регіонів для сортування

Регіони для сортування підгружаються динамічно з БД.

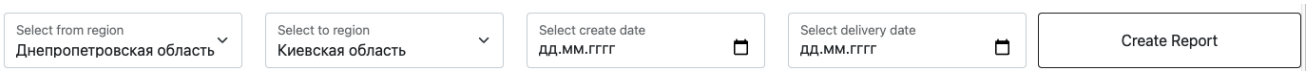
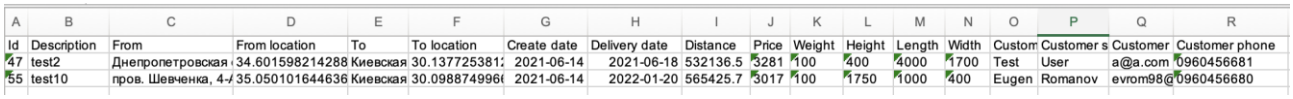


Рис. 3.40. Витяг замовлень з Дніпропетровської в Київську область.



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Id	Description	From	From location	To	To location	Create date	Delivery date	Distance	Price	Weight	Height	Length	Width	Custom	Customer s	Customer	Customer phone
47	test2	Днепропетровская	34.601598214288	Киевская	30.1377253812	2021-06-14	2021-06-18	532136.5	3281	100	400	4000	1700	Test	User	a@a.com	0960456681
55	test10	пров. Шевченка, 4-А	35.050101644636	Киевская	30.0988749966	2021-06-14	2022-01-20	565425.7	3017	100	1750	1000	400	Eugen	Romanov	evrom98@	0960456680

Рис. 3.41. Результат витягу

3.7. Висновки за розділом

– Розроблено та описано користувацький інтерфейс.

– Визначено алгоритм роботи користувача та менеджера.

– Досліджено архітектурний підхід програми.

– Вибрано реалізацію серверу.

– Додаток розгорнуто в хмарному сервісі Heroku.

ВИСНОВКИ

В кваліфікаційній роботі був розроблений логістичний сервіс для реєстрації замовлень, а також побудови оптимальних маршрутів. Виконано огляд та опис апаратно, програмної та архітектурної частин проекту.

Ключові компоненти розроблюваної системи:

− Використання СУБД PostgreSQL для підтримання персистентності даних на користувацькому інтерфейсі.

− Сервер на базі PaaS Heroku

− JSP сторінки, які відображаються кліжнтську чатину

− Бізнес-логіка додатку пов’язана з MapBox API та вирахуванням оптимального маршруту.

Наукова новизна отриманих результатів кваліфікаційної роботи визначається вдосконаленням методів комунікації клієнтів та приватних вантажоперевізників через розробку нового web-додатка з додаванням інтерактивної карти та можливістю автоматичного прорахунку вартості та відстані доставки.

Практична цінність результатів визначається як підвищення зручності планування у вантажних перевезеннях.

Подальші дослідження будуть направлені на пошуки можливості повної автоматизації процесів пошуку клієнтів і виконавців доставок, додаванню можливості створенню багатьох доставок для одного вантажоперевізника, зв’язування доставок одного виконавця в один ланцюг доставок відповідно по дням і прорахування альтернативного маршруту для ланцюга доставок, створення алгоритму пошуку доставок відповідно до транспортних обмежень виконавця, вподобань, вже вибраних доставок тощо.

1. https://www.osp.ru/cio/2007/01/3923760
2. https://www.bairesdev.com/blog/web-application-development-tips/
3. https://itglobal.com/ru-ru/company/glossary/ci-cd/
4. https://www.redhat.com/en/topics/cloud-native-apps/what-is-an-application-architecture
5. https://smartbear.com/blog/soap-vs-rest-whats-the-difference/
6. https://ru.wikipedia.org/wiki/Java
7. https://ru.wikipedia.org/wiki/HTML
8. https://ru.wikipedia.org/wiki/CSS
9. https://www.tutorialspoint.com/bootstrap/bootstrap_overview.htm
10. https://www.site2b.ua/web-blog/mobile-first-website-design.html
11. https://jquery.com/
12. https://ru.wikipedia.org/wiki/JSON
13. https://betacode.net/10169/java-servlet
14. https://betacode.net/10395/java-servlet-filter
15. https://github.com/LibrePDF/OpenPDF
16. https://java-online.ru/java-excel.xhtml
17. https://stackoverflow.com/questions/7249871/what-is-a-build-tool
18. https://docs.gradle.org/current/userguide/what_is_gradle.html#1_gradle_is_a_general_purpose_build_tool
19. https://proselyte.net/tutorials/jdbc/introduction/
20. https://en.wikipedia.org/wiki/Apache_Tomcat
21. https://en.wikipedia.org/wiki/JUnit
22. https://microservices.io/patterns/monolithic.html
23. https://refactoring.guru/uk/design-patterns/command
24. https://en.wikipedia.org/wiki/Regular_expression
25. *Герберт Шилдт*. Полный справочник по Java SE 6 = Java: The Complete Reference. — 7-е изд. — М.: «Вильямс», 2007. — С. 1040. — ISBN 0-07-226385-7.

26. https://developer.mozilla.org/ru/docs/Web/Guide/AJAX

ДОДАТОК А

		Позначення	Найменування	Кільк. аркушів	Примітка
1					
2			Документація		
3					
4		**ІТКІ.КР 19.03.ДА.ПЗ**	Пояснювальна записка	81	
5					
6			Презентація		
7					
8			Диск CD с презентацией	1	

Зм.	Ар-куш	№ докум	Підпис	Дата	**ІТКІ.КР 19.03.ДА.ПЗ**			
Розроб.		Є.І. Романов			**Матеріали кваліфікаційної**	Літ.	Аркуш	Аркушів
						Н	1	1
Керівник		К.Л.Сергєєва						
Рецензент		М.О. Алексєєв						

Н.контр.	Г.М. Коротенко			**роботи**	НТУ «ДП», 126м-20-1
Зав.каф.	В.В.Гнатушенко				

ДОДАТОК Б

Фрагменти вихідного коду програми

Index.jsp

```
<!DOCTYPE html>
<%@ page import="ua.epam.cargo_delivery.model.Action" %>
<%@ page contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c" %>
<%@ taglib prefix="fmt"
uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="d" uri="http://cargo_delivery.epam.ua"
%>
<%@ taglib tagdir="/WEB-INF/tags" prefix="tf" %>

<fmt:setLocale value="${param.lang == null ? 'en' :
param.lang}"/>
<fmt:setBundle basename="messages"/>
<c:set scope="page" value="${param.lang == null ? 'en' :
param.lang}" var="lang"/>
<html lang="${param.lang == null ? 'en' : param.lang}">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <!-- Bootstrap CSS -->
    <link href="<c:url value="/bootstrap-5.0.1-
dist/css/bootstrap.min.css"/>" rel="stylesheet">
    <link href="https://api.mapbox.com/mapbox-gl-
js/v2.3.0/mapbox-gl.css" rel="stylesheet">
    <link href="<c:url value="/css/delivery.css"/>"
rel="stylesheet">
    <link href="<c:url value="/css/map.css"/>"
rel="stylesheet">
    <link href="<c:url value="/css/pagination.css"/>"
rel="stylesheet">
    <link rel="icon" href='<c:url value="/favicon.ico"
/>' type="image/x-icon">
    <title>123 Delivery</title>
```

```
</head>
<body>
<nav class="navbar sticky-top navbar-light bg-light">
    <div class="container-fluid row px-2">
        <div class="col-6">
            <a id="homeLocation" class="navbar-brand"
href="<c:url value="/"/>">123 Delivery</a>
        </div>
        <div class="col d-flex justify-content-end">
            <div class="dropdown">
                <button class="btn btn-outline-info btn-
sm dropdown-toggle" type="button"
id="dropdownMenuButton1"
                        data-bs-toggle="dropdown"
                        aria-expanded="false">
                    <c:if test="${param.lang == null}">
                        en
                    </c:if>
                    <c:if test="${param.lang != null}">
                        ${param.lang}
                    </c:if>
                </button>
                <ul class="dropdown-menu" aria-
labelledby="dropdownMenuButton1">
                    <li><a class="dropdown-item"
href="?lang=en">en</a></li>
                    <li><a class="dropdown-item"
href="?lang=ru">ru</a></li>
                    <li><a class="dropdown-item"
href="?lang=ua">ua</a></li>
                </ul>
            </div>
            <c:if test="${sessionScope.loggedUser.role ==
'USER'}">
                <button type="button" class="btn btn-
outline-info btn-sm text-dark" data-bs-toggle="modal"
                        data-bs-target="#staticBackdrop">
                    <fmt:message key="authorization"/>
                </button>
                <button type="button" class="btn btn-
outline-info btn-sm text-dark" data-bs-toggle="modal"
                        data-bs-
target="#registrationStaticBackdrop">
                    <fmt:message key="registration"/>
                </button>
```

```
            </c:if>
            <tf:auth/>
        </div>
    </div>
</nav>
<section class="container my-5">
    <div class="row align-items-center">
        <h4 class="col"><fmt:message
key="available.regions"/></h4>
        <div class="col">
            <ul class="list-group list-group-flush">
                <c:forEach
items="${sessionScope.availableRegions}" var="region">
                    <li class="list-group-item text-
center"><d:region/></li>
                </c:forEach>
            </ul>
        </div>
    </div>
</section>
<div class="container-fluid">
    <div class="row">
        <div id="map" class="col"></div>
        <form class="col container-fluid"
              action="<c:url
value="/privateOffice?lang=${param.lang == null ? 'en' :
param.lang}"/>" method="post">
            <section class="row">
                <div class="form-floating mb-3 col">
                    <input name="from" class="form-
control" id="from" placeholder="<fmt:message
key="location.from"/>"
                           readonly>
                    <label for="from" style="left:
auto"><fmt:message key="location.from"/></label>
                </div>
                <div class="form-floating mb-3 col">
                    <input name="to" class="form-control"
id="to" placeholder="<fmt:message key="location.to"/>"
                           readonly>
                    <label for="to" style="left:
auto"><fmt:message key="location.to"/></label>
                </div>
                <label><input name="fromRegionId"
id="fromRegionId" hidden></label>
```

```
                <label><input name="toRegionId"
id="toRegionId" hidden></label>
            </section>
            <section class="row">
                <div class="form-floating mb-3 col">
                    <input name="fromName" class="form-
control" id="fromName"
                           placeholder="<fmt:message
key="address.from"/>" readonly>
                    <label for="fromName" style="left:
auto"><fmt:message key="address.from"/></label>
                </div>
                <div class="form-floating mb-3 col">
                    <input name="toName" class="form-
control" id="toName" placeholder="<fmt:message
key="address.to"/>"
                           readonly>
                    <label for="toName" style="left:
auto"><fmt:message key="address.to"/></label>
                </div>
            </section>
            <section class="row">
                <label class="col"><fmt:message
key="display.weight"/>
                    <select name="weight" id="weight"
class="form-select overflow-hidden" multiple>
                        <option value="100"
selected><100</option>
                        <option value="500">100 -
500</option>
                        <option value="1000">500 -
1000</option>
                        <option value="1500">1000 -
1500</option>
                    </select>
                </label>
                <label class="col"><fmt:message
key="display.length"/>
                    <select name="length" id="length"
class="form-select overflow-hidden" multiple>
                        <option value="1000"
selected><1000</option>
                        <option value="2000">1000 -
2000</option>
                        <option value="3000">2000 -
```

```
3000</option>
                        <option value="4000">3000 -
4000</option>
                    </select>
                </label>
                <label class="col"><fmt:message
key="display.width"/>
                    <select name="width" id="width"
class="form-select overflow-hidden" multiple>
                        <option value="400"
selected><400</option>
                        <option value="900">400 -
900</option>
                        <option value="1400">900 -
1400</option>
                        <option value="1700">1400 -
1700</option>
                    </select>
                </label>
                <label class="col"><fmt:message
key="display.height"/>
                    <select name="height" id="height"
class="form-select overflow-hidden" multiple>
                        <option value="400"
selected>>400</option>
                        <option value="900">400 -
900</option>
                        <option value="1400">900 -
1400</option>
                        <option value="1750">1400 -
1750</option>
                    </select>
                </label>
            </section>
            <section class="row align-items-center pt-1">
                <label hidden><input id="inputPrice"
name="price"></label>
                <h6 id="price" class="col-6 my-0"></h6>
                <div class="col-6 d-flex flex-row-
reverse">
                    <c:if
test="${sessionScope.loggedUser.role.checkPermission(Acti
on.CREATE_DELIVERY)}">
                        <button type="submit"
                                class="btn btn-outline-
```

```
success col-5 col-xxl-4 ms-2">
                                <fmt:message
key="button.create"/>
                            </button>
                        </c:if>
                        <button class="btn btn-outline-info
col-5 col-xxl-4" type="button"

onclick="calculatePrice('<c:url
value="/calculatePrice"/>')">
                            <fmt:message
key="button.calculate"/>
                        </button>
                    </div>
                </section>
            </form>
        </div>
    </div>
    <section id="tableSection">
        <table id="deliveryTable" class="table caption-top">
            <caption><fmt:message
key="table.caption"/></caption>
            <thead class="table-dark">
            <tr class="align-middle">
                <th scope="col">
                    <div class="container-fluid row m-0 p-0
align-items-center">
                        <span class="col-7"
col="fromName"><fmt:message key="table.from"/></span>
                        <div class="col">
                            <div class="input-group input-
group-sm">
                                <label for="filterWhence"
class="input-group-text">
                                    <img src="<c:url
value="/icons/filter.svg"/>" alt="Filter">
                                </label>
                                <input id="filterWhence"
col="fromName" type="text" class="form-control"

placeholder="<fmt:message
key="filter.from.description"/>">
                            </div>
                        </div>
                    </div>
```

```
            </th>
            <th scope="col">
                <div class="container-fluid row m-0 p-0
align-items-center">
                    <span class="col-7"
col="toName"><fmt:message key="table.to"/></span>
                    <div class="col">
                        <div class="input-group input-
group-sm">
                            <label for="filterWhither"
class="input-group-text">
                                <img src="<c:url
value="/icons/filter.svg"/>" alt="Filter">
                            </label>
                            <input id="filterWhither"
col="toName" type="text" class="form-control"

placeholder="<fmt:message key="filter.to.description"/>">
                        </div>
                    </div>
                </div>
            </th>
            <th scope="col"><span col="distance"
class="d-flex"><fmt:message
key="table.distance"/></span></th>
            <th scope="col"><span col="price" class="d-
flex"><fmt:message key="table.price"/></span></th>
        </tr>
        </thead>
        <tbody id="data-container" class="container-
fluid">
        <c:forEach items="${sessionScope.deliveries}"
var="delivery">
            <tr>
                <td class="table-tb-width-40">
                    <div class="d-flex flex-column">
                        <div class="flex-row flex-
wrap">${delivery.fromName}</div>
                        <div class="flex-row flex-wrap
text-muted fs-6">${delivery.whence}</div>
                    </div>
                </td>
                <td class="table-tb-width-40">
                    <div class="d-flex flex-column">
                        <div class="flex-row flex-
```

```
wrap">${delivery.toName}</div>
                            <div class="flex-row flex-wrap
text-muted fs-6">${delivery.whither}</div>
                        </div>
                    </td>
                    <td>${delivery.distance}</td>
                    <td>${delivery.price}</td>
                </tr>
            </c:forEach>
            </tbody>
        </table>
        <div id="pagination-container"></div>
</section>

<!-- Modal -->
<div class="modal fade" id="staticBackdrop" data-bs-
backdrop="static" data-bs-keyboard="false" tabindex="-1"
     aria-labelledby="staticBackdropLabel" aria-
hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title"
id="staticBackdropLabel">Authorization</h5>
                <button type="button" class="btn-close"
data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <form action="authorization" method="POST"
class="modal-body">

                <div class="form-floating">
                    <input name="email" type="email"
class="form-control" id="authEmail"
                           placeholder="name@example.com"
pattern="^[a-zA-Z0-9]+@[a-zA-Z]+\.[a-zA-Z]+$" required>
                    <label for="authEmail">Email
address</label>
                </div>
                <div class="form-floating">
                    <input name="password"
type="password" class="form-control" id="authPassword"
                           placeholder="Password"
required>
                    <label
for="authPassword">Password</label>
```

```
                </div>

                <div class="modal-footer">
                    <button class="w-100 btn btn-lg btn-
primary" type="submit">Log in</button>
                </div>

            </form>
        </div>
    </div>
</div>

<!-- Modal -->
<div class="modal fade" id="registrationStaticBackdrop"
data-bs-backdrop="static" data-bs-keyboard="false"
tabindex="-1"
     aria-labelledby="staticBackdropLabel" aria-
hidden="true">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title"
id="registrationStaticBackdropLabel">Authorization</h5>
                <button type="button" class="btn-close"
data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <form action="registration" method="POST"
class="modal-body">

                <div class="form-floating">
                    <input name="name" class="form-
control" id="name"
                           placeholder="Name" required>
                    <label for="name">Name</label>
                </div>
                <div class="form-floating">
                    <input name="surname" class="form-
control" id="surname"
                           placeholder="Surname"
required>
                    <label for="surname">Surname</label>
                </div>
                <div class="form-floating">
                    <input name="phone" type="tel"
pattern="^(\+38)?(0\d{9})$" required class="form-control"
```

```
id="phone"
                                placeholder="+380961111111 or
0661111111" title="For example:+380961111111 or
0661111111"

onchange="checkUniquePhone('<c:url
value="/validate"/>')">
                        <label for="phone">Phone
number</label>
                    </div>
                    <div class="form-floating">
                        <input name="email" type="email"
required class="form-control" id="email"
                                placeholder="name@example.com"
                                pattern="^[a-zA-Z1-9 ]+@[a-zA-
Z]+\.[a-zA-Z]+$"

onchange="checkUniqueEmail('<c:url
value="/validate"/>')">
                        <label for="email">Email
address</label>
                    </div>
                    <div class="form-floating">
                        <input name="password"
type="password" class="form-control"
id="registerPassword"
                                placeholder="Password"
required>
                        <label
for="registerPassword">Password</label>
                    </div>
                    <div class="form-floating">
                        <input type="password" class="form-
control" id="passwordCheck"
                                placeholder="Password"
required>
                        <label for="passwordCheck">Repeat
password</label>
                    </div>

                    <div class="modal-footer">
                        <button class="w-100 btn btn-lg btn-
success" type="submit"
                                 onclick="return
validateForm();">Sign up
```

```
                    </button>
                </div>

            </form>
        </div>
    </div>
</div>

<script src="<c:url value="/bootstrap-5.0.1-
dist/js/bootstrap.bundle.min.js"/>" async></script>
<script src="https://api.mapbox.com/mapbox-gl-
js/v2.3.0/mapbox-gl.js"></script>
<script src="<c:url value="/js/jquery-
3.6.0.min.js"/>"></script>
<script src="<c:url value="/js/delivery.js"/>"
async></script>
<script src="<c:url value="/js/map.js"/>" async></script>
<script src="<c:url
value="/js/pagination.js"/>"></script>
<script src="<c:url value="/js/index.js"/>"
async></script>
<script>
    sessionStorage.setItem("totalNumber",
${sessionScope.totalNumber});
</script>
</body>
</html>
```

ValidationServle.java

```
package ua.epam.cargo_delivery.servlets;

import ua.epam.cargo_delivery.dto.Response;
import ua.epam.cargo_delivery.exceptions.AppException;
import
ua.epam.cargo_delivery.servlets.commands.CheckEmail;
import
ua.epam.cargo_delivery.servlets.commands.CheckPhone;
import ua.epam.cargo_delivery.servlets.commands.Command;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import java.io.IOException;
import java.util.HashMap;
import java.util.Optional;

@WebServlet(name = "ValidationServlet", urlPatterns =
"/validate")
public class ValidationServlet extends HttpServlet {
    private final HashMap<String, Command> commands = new
HashMap<>();

    @Override
    public void init(ServletConfig config) throws
ServletException {
        super.init(config);
        commands.put("phone", new CheckPhone());
        commands.put("email", new CheckEmail());
    }

    @Override
    protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        try {
            String ajax = req.getHeader("x-requested-
with");
            if (!"XMLHttpRequest".equals(ajax)) {
                throw new AppException("Only ajax
support");
            }

            String command =
Optional.ofNullable(req.getParameter("check")).filter(s -
> !s.isBlank()).orElse(null);
            Command c;
            if (command == null || (c =
commands.get(command)) == null) {
                throw new AppException("Command not
found");
            }
            String param =
Optional.ofNullable(req.getParameter("param")).filter(s -
> !s.isBlank()).orElse(null);
            if (param == null) {
                throw new AppException("Empty search
parameter");
```

```
            }
            resp.getWriter().write(c.execute(param));
        } catch (Exception e) {

resp.setStatus(HttpServletResponse.SC_BAD_REQUEST);

resp.getWriter().write(Response.builder().message(e.getMe
ssage()).build().toString());
        }
    }
}
```

PaginationDeliveryServlet.java

```
package ua.epam.cargo_delivery.servlets;

import ua.epam.cargo_delivery.dto.DeliveryDTO;
import ua.epam.cargo_delivery.model.db.Delivery;
import ua.epam.cargo_delivery.model.db.DeliveryManager;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;
import java.util.Optional;

@WebServlet(name = "PaginationDeliveryServlet",
urlPatterns = "/paginationDelivery")
public class PaginationDeliveryServlet extends
HttpServlet {
    private static final Integer DEFAULT_LIMIT = 5;
    private static final Integer DEFAULT_PAGE = 0;
    private static final String DEFAULT_ORDER_BY = "id";

    @Override
    protected void service(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        Integer limit =
Optional.ofNullable(req.getParameter("pageSize")).filter(
s ->
!s.isBlank()).map(Integer::parseInt).orElse(DEFAULT_LIMIT
```

```
);
        Integer page =
Optional.ofNullable(req.getParameter("pageNumber")).filte
r(s ->
!s.isBlank()).map(Integer::parseInt).orElse(DEFAULT_PAGE)
;
        String orderBy =
Optional.ofNullable(req.getParameter("orderBy")).filter(s
-> !s.isBlank()).orElse(DEFAULT_ORDER_BY);
        boolean asc =
Optional.ofNullable(req.getParameter("ascending")).filter
(s ->
!s.isBlank()).map(Boolean::parseBoolean).orElse(true);
        String filterFrom =
Optional.ofNullable(req.getParameter("filter[fromName]"))
.orElse("");
        String filterTo =
Optional.ofNullable(req.getParameter("filter[toName]")).o
rElse("");
        Long userId =
Optional.ofNullable(req.getParameter("userId")).filter(s
-> !s.isBlank()).map(Long::parseLong).orElse(null);
        List<Delivery> deliveries =
DeliveryManager.findDeliveries(limit, page, orderBy, asc,
filterFrom, filterTo, userId);
        resp.setCharacterEncoding("UTF-8");

resp.getWriter().write(DeliveryDTO.builder().deliveries(d
eliveries).build().toString());
    }
}
```

PrivateOfficeServlet.java

```
package ua.epam.cargo_delivery.servlets;

import ua.epam.cargo_delivery.model.db.Delivery;
import ua.epam.cargo_delivery.model.db.DeliveryManager;
import ua.epam.cargo_delivery.model.db.DeliveryStatus;
import ua.epam.cargo_delivery.model.db.User;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;

@WebServlet(name = "PrivateOfficeServlet", urlPatterns =
"/privateOffice")
public class PrivateOfficeServlet extends HttpServlet {
    private final int limit = 5;
    private final int page = 0;

    @Override
    protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        User user = (User)
req.getSession().getAttribute("loggedUser");
        req.setAttribute("userDeliveries",

DeliveryManager.findDeliveriesForUser(limit, page,
user));
        List<Delivery> deliveriesForPay =
DeliveryManager.findDeliveriesWithStatus(DeliveryStatus.A
PPROVED, user);
        req.getSession().setAttribute("deliveriesForPay",
deliveriesForPay);
        req.setAttribute("commonPrice",
deliveriesForPay.stream().map(Delivery::getPrice)
                .reduce(Integer::sum).orElse(null));

req.getSession().setAttribute("totalNumberForUser",
DeliveryManager.getTotalNumberForUser((User)req.getSessio
n().getAttribute("loggedUser")));
        req.getRequestDispatcher("/WEB-
INF/privateOffice.jsp").forward(req, resp);
    }
}
```

MainServlet.java

```
package ua.epam.cargo_delivery.servlets;

import ua.epam.cargo_delivery.model.db.*;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;

@WebServlet(name = "MainServlet", urlPatterns = "/main")
public class MainServlet extends HttpServlet {
    private static final String AVAILABLE_REGIONS =
"availableRegions";
    private static final String LOGGED_USER =
"loggedUser";
    private static final int LIMIT = 5;
    private static final String DELIVERIES =
"deliveries";
    private static final String TOTAL_NUMBER =
"totalNumber";

    @Override
    protected void doGet(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException {
        if (((User)
req.getSession().getAttribute(LOGGED_USER)).getRole() ==
Role.MANAGER) {

req.getRequestDispatcher("/manager").forward(req, resp);
            return;
        }
        initDisplayedData(req);
        setUpPagination(req);
        setSupportRegions(req);
        req.getRequestDispatcher("/WEB-
INF/index.jsp").forward(req, resp);
    }

    private void initDisplayedData(HttpServletRequest
```

```
req) {
        if (req.getSession().getAttribute(DELIVERIES) ==
null) {
            List<Delivery> deliveries =
DeliveryManager.findDeliveries(LIMIT, 0);
            req.getSession().setAttribute(DELIVERIES,
deliveries);
        }
    }

    private void setUpPagination(HttpServletRequest req)
{
        req.getSession().setAttribute(TOTAL_NUMBER,
DeliveryManager.getTotalNumber());
    }

    private void setSupportRegions(HttpServletRequest
req) {
        if
(req.getSession().getAttribute(AVAILABLE_REGIONS) ==
null) {

req.getSession().setAttribute(AVAILABLE_REGIONS,
CityManager.getCities());
        }
    }
}
```

ДОДАТОК В

ВІДГУК

на кваліфікаційну роботу магістра

"Розробка інформаційної системи управління доставкою вантажу на мові Java"

студента групи 126м-20-1 Романова Євгена Ігоровича

1. Метою кваліфікаційної роботи є створення складових інформаційної системи обліку, планування та управління процесами доставки вантажу засобами WEB-технологій.

2. Завдання та зміст кваліфікаційної роботи відповідають основній меті – оцінці знань і ступеня підготовленості студента за спеціальністю 126 "Інформаційні системи та технології".

3. Тема роботи представляється актуальною і спрямована на мінімізацію витрат та оптимізацію параметрів вантажоперевезень між двома пунктами з урахуванням клієнт-орієнтованого підходу. Використання у роботі базової карти дозволяє користувачеві взаємодіяти у інтерактивному режимі з розробленою інформаційною системою, візуалізувати маршрути доставки та відстежувати у режимі реального часу географічне розташування вантажу.

4. Для досягнення мети кваліфікаційної роботи Романов Є.І. застосував протокол обміну інформації REST, спроектував клієнт-серверну архітектуру інформаційної системи, реалізував базу даних системи засобами СУБД PostgreSQL, застосував API картографічного сервісу Mapbox для візуалізації даних.

5. Оформлення пояснювальної записки виконано, в основному, відповідно до діючих стандартів і нормативних вимог.

6. У якості зауваження слід відзначити недостатнє порівняння створеної інформаційної системи з існуючими аналогами, відсутність детального опису переваг практичного результату роботи.

Зважаючи на зазначені недоліки, кваліфікаційна робота заслуговує оцінки "_____".

Керівник,

доцент кафедри ІТКІ К.Л. Сергєєва

ДОДАТОК Г

РЕЦЕНЗІЯ

на кваліфікаційну роботу магістра
"Розробка інформаційної системи управління доставкою вантажу на мові Java"
студента групи 126м-20-1 Романова Євгена Ігоровича

1. Тема кваліфікаційної роботи, присвячена створенню інформаційної системи управління доставкою вантажу є актуальною і спрямована на вирішення проблеми створення зручного програмного інструментарію взаємодії менеджера і замовника послуг перевезення вантажу на базі єдиної програмної платформи.
2. У рецензованій роботі Романов Є.І. спроектував та наповнив базу даних інформаційної системи, реалізував методи оптимізації параметрів перевезень, застосував інтерфейс взаємодії користувача та функціональної складової системи, забезпечив можливість інтерактивної візуалізації результати розрахунків на основі картографічного сервісу Mapbox.
3. Наукова новизна роботи полягає у створенні складових інформаційної технології та системи управління доставкою вантажу методами оптимізації (мінімізації) параметрів часу, відстані та вартості перевезень.
4. Практична значимість результатів кваліфікаційної роботи Романова Є.І. полягає у використанні мови програмування Java, набору інструментів Bootstrap, СУБД PostgreSQL та ін. для реалізації повнофункціональної web-орієнтованої інформаційної системи надання послуг управління вантажоперевезеннями та генерування супровідної документації.
5. Робота цілком відповідає вимогам, що пред'являються до кваліфікаційних робіт ступеня магістр.

Недолік: методи оптимізації вантажоперевезеннями описані недостатньо. Не у повній мірі розкрита й обґрунтована наукова новизна роботи.

Зважаючи на зазначений недолік, кваліфікаційна робота в цілому може бути відзначена оцінкою "_____".

Рецензент,
декан факультету інформаційних технологій,
д.т.н., професор М.О. Алексєєв