Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії

(повна назва факультету)

Програмної інженерії

(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр

(назва освітнього ступеня)

на тему: «Розробка системи координації та визначення критичного шляху переміщення рухомого складу на мові Java і фреймворків Spring, React»

Виконав(ла): студент(ка) _____ курсу, групи _____

спеціальності

121 – Інженерія програмного забезпечення

(шифр і назва спеціальності)

_____ Матвієнко Т.В.

(підпис) (прізвище та ініціали)

Керівник _____ Цуприк Г.Б.

(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.

(підпис) (прізвище та ініціали)

Завідувач _____ Петрик М.Р.

(підпис) (прізвище та ініціали)

Рецензент _____ _____

(підпис) (прізвище та ініціали)

Тернопіль 2022

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: с., рис., табл., джер.

ОБ'ЄКТ, ПРЕДМЕТ, МЕТА, РОЗРОБКА, АРХІТЕКТУРА, SQL, MYSQL, СЕРВЕР, API, JAVA, SPRING, HIBERNATE, JAVASCRIPT.

Метою кваліфікаційної роботи є представлення варіанту програмної реалізації, яка забезпечить можливість визначення критичного шляху переміщення рухомого складу.

Методи розробки базуються на технологіях Java та JavaScript, сервері бази даних MySQL,Hibernate ORM для роботи з базою даних і ReactJS для розробки односторінкового веб-застосунку.

В роботі взято до уваги та застосовано одну із методик проєктування архітектури системи із модульною структурою та при застосуванні сучасних технологій web- програмування.

OBJECT, SUBJECT, PURPOSE, DEVELOPMENT, ARCHITECTURE, SQL, MYSQL, SERVER, API, JAVA, SPRING, HIBERNATE, JAVASCRIPT.

The purpose of the qualification thesis is to presentation a variant of software implementation that will provide the possibility to make best path of transportation.

Development methods are based on Java and Python technologies, MySQL database server, Hibernate ORM for working with the database and ReactJS for website development.

The work takes into account and applies one of the methods of designing the system architecture with a modular structure and using modern web programming technologies.

ЗМІСТ

ВСТУП

На сьогоднішній день глобалізація економіки світу та створення взаємопов’язаних виробничих потужностей змушує якнайбільше зменшувати час на фізичні пересування між різними точками світу. Великі корпорації, та цілі держави витрачають дуже багато ресурсів на те, щоб забеспечувати зростання ВВП, та зменшення затримок.

В такому випадку особливої актуальності набуває задача аналізу та керуванням критичним шляхом для різних моделей, та потребностей. При цьому також потрібно зберегти високу надійність, приватність та ефективність.

Для розв’язання такої задачі будуть використовуватись методи визначення критичного шляху. Такі алгоритми дуже широко використовуються в багатьох галузях. В побудові та підтримці проєктів, машинному навчанні та в бізнесі загалом.

Методи критичного шляху зазвичай здаються дуже простими у використанні. Але тут потрібно правильно визначити вагу кожного пункту, бо на основі цих оцінок метод буде працювати. Якщо оцінки будуть не достатньо влучними, то і сам метод буде теж працювати не так, як було задумано із самого початку.

Однією із вагомих переваг є те, що всі ланцюжки є важливими для шляху. Тобто жоден із них не може бути проігнорованим, або забутим, або протермінованим.

Під час роботи над проєктом було опрацьовано багато матеріалів. Завдяки цьому було сформульовано основне завдання та мета цієї роботи. Спочатку буде представлено розв’язок проблеми у теоретичному вигляді, а потім і у практичному. Проєкт охоплює дуже багато частин людського суспільства і може бути використаним за різними призначеннями. Посилаючись на великий попит до схожих систем, мною було прийняте рішення розробити власну систему із врахуванням усіх недоліків тих систем, які вже існують на ринку.

Дипломний проєкт присвячений розробці системи координацій та визначення критичного шляху рухомого складу на мові Java, з використанням фреймворків Spring, React.

1 Огляд предметної області

1.1 Аналіз предметної області

Як тільки створилось людське суспільство, виникла потреба в координації дій між різними членами цього суспільства. На сьогоднішній день системи координацій широко використовуються в різних галузях життя: освіта, наука, медицина, великий та малий бізнес, соціальна координація та інші. І чим більша та ширша є галузь, тим краще та тендітніше повинна бути налагоджена координація в середині неї.

Існує поширена думка, що координації це марна трату часу та енергії. Яка могла б бути використана у більш ефективному сценарії. Зазвичай ця ідея базується на невдалому досвіді, коли на організаційних заходах дуже багато обговорюється, але реалізується з того дуже мало.

При здоровому підході до координацій та вдалому їх проведені, позитивного результату буде в рази більше, ніж могло би бути без таких координацій. Дуже часто через відсутність узгодженості багато ресурсів може бути витрачено не раціонально, або взагалі дарма без можливості їх повернення.

Координація повинна проводитись на різних рівнях відповідальності. Від найвищого керівництва, яке буде задавати напрямок для подальших дій, до окремих команд та підрозділів. В таких командах, їхні керівники вже повинні донести до членів чіткий план по тому що і як повинно бути зроблено. Також важливо дотримуватись часового ліміту.

Загалом координаційні заходи потрібні для того, щоб кінцева мета була досягнута, а дії окремих осіб були скоординовані та підкріплені планом.

Розподіл відповідальності на маленькі частини дозволяє позбутися стресу, який може з’явитись в людини від несподіваних завдань, про які раніше ніхто і не згадував[1].

Загалом координації покликані полегшити виконання кінцевої мети, та розподіл відповідальності між працівниками.

В протилежному разі, якщо кординації налагоджені слабко, або їх немає взагалі, то дуже велика частина ресурсів може даремно втрачатись. Або витрачатись не з максимальною ефективністю.

Принципи ефективної координації:

-Системність та систематичність

-Чітко визначений критерій позитивного результату(кількісний або якісний)

-Автоматизація

-Глобалізація

-Відслідковування процесу та постійне вдосконалення.

Системи координацій у бізнесі знаходяться на дуже високому рівні. Вони відіграють визначальну роль в тому, щоб бізнес міг існувати і приносити користь людству. За останні кілька десятиліть масова цифровізація та інші чинники зумовленні технічним прогресом людства сприяли тому, що все більше і частіше бізнес залучає до створення та підтримки своїх систем координацій IT сектор. Це значно спрощує та автоматизує велику кількість внутрішніх процесів. Прикладом цього можуть бути логістичні функції бізнесу, які можна автоматизувати. Розв’язок такої проблеми буде освітлено у цій роботі.

Координація в логістичній сфері передбачає широке та клопітне планування: чим більше етапів, тим краще. В процесі розгляду кожного етапу продукту, його розповсюдження та повернення потрібно максимально підвищувати ефективність та збільшувати доходи[2].

Для початку роботи над проєктом потрібно з’ясувати які проблеми він буде вирішувати, та який результат буде вважатись успішним[3]. Система координацій покликана автоматизувати процеси прокладання критичного шляху для рухомих складів, які перевозять різні матеріали. При бажанні цю систему можна буде розширити і використовувати в інших цілях. Для успішного впрвадження системи вона повинна мати якісну архітектурну основу, та зрозумілий інтерфейс для користувачів, адже користуватись нею буде не розробник, а особа, яка має зовсім іншу спеціалізацію.

1.2 Обґрунтування вибору напрямку дослідження та мови програмування

Java - компільована, об’єктно-орієнтована мова програмування зі строгою типізацією[4]. Вперше з’явилася на ринку у 1995 році і до сьогоднішнього дня займає ваговий відсоток використання у порівнянні з іншими мовами програмування. Здобула свою популярність через С-подібний синтаксис який полегшує та пришвидшує процес читання та обслуговування коду, та можливість бути запущеною на великій кількості пристроїв. «Напиши один раз - запускай де завгодно» - гасло цієї крос-платформеної мови. Скомпільований код може бути запущений на будь якій платформі, що підтримує і містить JRE (Java Runtime Environment) - серидовище яке і буде виконувати код.

Java підтримує пакети що дозволяють повторюване використання коду. Також не в останню чергу Java є досі популярною мовою через свою довгу підтримуваність основних пакетів зі сторони розробників. Код написаний у 20 столітті на мові джава може без проблем працювати з кодом який буде написаний у 2022 році.

Експанція мови програмування Java у світі програмування є дуже широкою, і це підтверджується заявою 3 мільярди пристроїв, що запускають Java, яка відображається під час встановлення Java. Java надає багатий і широкий спектр API, який допомагає програмістам розробляти програми. Використовуючи Java, ми можемо розробляти різні програми для різних цілей. Ми можемо використовувати технологію Java для розробки таких програм:

-Розробка мобільних застосунків. Мова програмування Java може вважатися офіційною мовою для розробки мобільних додатків. Більшість додатків для Android будуються за допомогою Java. Найпопулярніша IDE розробки додатків Android Android Studio також використовує Java для розробки додатків Android. Отже, якщо ви вже знайомі з Java, розробляти додатки для Android стане набагато простіше. Найпопулярніші Android-додатки Spotify і Twitter розроблені з використанням Java.

-Десктопні застосунки. Ми також можемо розробити програму графічного інтерфейсу за допомогою Java. Java надає AWT, JavaFX і Swing для розробки настільних програм на основі GUI. Інструменти містять попередньо зібрані компоненти, такі як список, меню, кнопка.

-Веб застосунки. Він також використовується для розробки веб-додатків, оскільки забезпечує широку підтримку веб-розробки через Servlet, JSP і Struts. Це причина того, що Java також відома як мова програмування на стороні сервера. Використовуючи ці технології, ми можемо розробляти різноманітні програми. Найпопулярніші фреймворки Spring, Hibernate, Spring Boot, що використовуються для розробки веб-додатків. LinkedIn, AliExpress, web.archive.org, IRCTC тощо є популярними веб-сайтами, які написані мовою програмування Java.

-Комп'ютерні ігри. Java широко використовується компаніями-розробниками ігор, оскільки вона підтримує найпотужніший 3D-движок із відкритим кодом. Двигун забезпечує неперевершену потужність, коли мова заходить про контекст розробки 3D-ігор. Найпопулярнішими іграми, розробленими на Java, є Minecraft, Mission Impossible III тощо. Для розробки ігор доступні деякі популярні фреймворки та бібліотеки, наприклад LibGDX і OpenGL.

-Big Data технології. Багато мов програмування доступні для технології великих даних, але Java є першим вибором для цього. Інструментальна платформа Hadoop HDFS для обробки та зберігання великих даних додатків написана на Java. У великих даних Java широко використовується в програмах ETL, таких як Apache Camel і Apache Kafka. Він використовується для вилучення та перетворення даних і завантаження в середовищах великих даних.

-Хмарні застосунки. Хмарна програма – це доступність IT-ресурсів через Інтернет на вимогу. Хмарна програма надає послугу за низькою ціною. Java забезпечує середовище для розробки хмарних програм. Ми можемо використовувати Java для розробки SaaS (програмне забезпечення як послуга), LaaS (реєстрація як послуга) і PaaS (платформа як послуга). Хмарна програма, яка широко використовується для обміну даними між компаніями або для віддаленої розробки програм.

-Застосунки інтернету речей. IoT — це технологія, яка з’єднує пристрої у своїй мережі та спілкується з ними. IoT можна знайти майже в усіх невеликих пристроях, таких як приладдя для здоров’я, смартфони, пристрої для носіння, розумне освітлення, телевізори тощо. Для розробки додатків IoT можна використовувати багато мов програмування, але Java пропонує перевагу розробникам, тобто безпрецедентний. Програмісти IoT тяжіють до Java через її безпеку, гнучкість і універсальність.

Порівняно з іншими мовами програмування, Java відрізняється своєю безпекою та функціональністю. Java ізолює себе від інших мов програмування завдяки функціональності та безпеці, і це також актуально. Є й інші причини використовувати Java:

-Масштабованість: масштабованість додає ємності нашій системі. Це покращує потужність системи, додаючи системні ресурси, не впливаючи на архітектуру розгортання. Ми можемо досягти масштабованості, збільшивши такі ресурси, як оперативна пам’ять і процесор, в одній системі. Це чудово справляється з робочим навантаженням, підвищує продуктивність системи та максимізує продуктивність.

-Кросплатформенність: кросплатформенність означає, що скомпільовану програму Java можна запускати на всіх платформах. Пам'ятайте, що система повинна мати JVM. Після компіляції програми Java код Java перетворюється на байт-код, який не залежить від платформи. Цей байт-код розуміє JVM. Ми можемо запустити цей байт-код на будь-якій платформі.

-Управління пам'яттю: Java надає власний механізм керування пам'яттю, відомий як збирач сміття. Нам не потрібно піклуватися про пам'ять і не потрібно впроваджувати це для керування пам'яттю. Він автоматично видаляє об’єкти, коли вони більше не використовуються програмою. Це покращує швидкість програми.

-Багатопотоковість: Потік — це легкий підпроцес. Багатопотоковість в Java дозволяє одночасне виконання двох або більше потоків одночасно. Це максимізує використання ЦП.

JavaScript - інтерпретована, об’єктно-орієнтована мова програмування яка найчастіше використовуєтсья для написання сценаріїв на веб-сторінках, хоче може

бути використана і в інших випадках. Ця мова є динамічною і підтримує такі стилі програмування як об’єктно-орієнтований, функціональний та імперативний. Код написаний на JavScript виконуєтсья на клієнтській стороні. Легкий синтаксис та не строга типізація робить вивчення мови легким та дозволяє легко і швидко підтримувати та розширяти вже наявні застосунки.

До основних переваг JavaScript відносять його асинхронність, що дозволяє продовжувати виконання програмного коду лише після того як будуть отримані всі данні від сервера[5]. Також JavaScript володіє іще декількома перевагами:

-JavaScript надає розробникам HTML інструмент програмування : ті хто верстають на HTML зазвичай не являються програмістами, але JavaScript є мовою сценаріїв, яка використовує дуже простий синтаксис. Кожен хто приділить трошки часу для вивчення - зможе розмістити невеликі фрагменти коду на своїх HTML-сторінках.

-JavaScript здатний розміщувати динамічний текст на HTML-сторінці : такий оператор JavaScript: документ. Write ("<h1>" + ім'я + "</h1>") може писати змінний текст на сторінці HTML.

-JavaScript здатний реагувати на події. JavaScript може бути налаштований на реалізацію, коли щось відбувається, без перезавантаження сторінки.

-JavaScript може читати та писати HTML-елементи. Такожможе читати та змінювати вміст елемента HTML.

-JavaScript можна використовувати для валідації даних: JavaScript можна використовувати для перевірки даних форми перед їх надсиланням на сервер. Це захищає сервер від додаткової обробки.

-Можливість використання JavaScript для виявлення браузера користувача: JavaScript може використовуватися для виявлення браузера відвідувача та, залежно від браузера, завантажувати іншу сторінку, спеціально розроблену для цього браузера.

-JavaScript можна використовувати для створення файлів cookie: JavaScript можна використовувати для зберігання та отримання інформації на комп’ютері відвідувача.

-При таких хороших перевагах також не потрібно забувати і про недоліки, які JavaScript має:

-Оскільки код JavaScript доступний для перегляду користувачеві, інші можуть використовувати його для зловмисних цілей. Ці практики можуть включати використання вихідного коду без автентифікації. Крім того, дуже легко розмістити код на сайті, який порушить безпеку даних на веб-сайті.

-Різні браузери інтерпретують JavaScript по різному. Таким чином, перед публікацією код повинен бути запущений на різних платформах. Старіші браузери не підтримують деякі нові функції.

-Хоча деякі редактори HTML підтримують дебаг, він не такий ефективний, як інші редактори, наприклад редактори Java. Крім того, оскільки браузер не показує жодної помилки, розробнику важко виявити проблему.

-JavaScript підтримує лише одиночне успадкування, а не множинне успадкування. Для деяких програм може знадобитися ця характеристика об’єктно-орієнтованої мови.

-JavaScript зберігає число як 64-розрядне число з плаваючою комою, а оператори працюють з 32-розрядними операндами. Таким чином, JavaScript перетворює число на 32-розрядні цілі числа зі знаком, оперує ними та перетворює їх назад на 64-розрядні числа JavaScript. Це безперервне перетворення займає більше часу для перетворення числа в ціле. Це збільшує час, необхідний для виконання сценарію, і зменшує його швидкість.

-Одна помилка коду може зупинити відтворення всього коду JavaScript на веб-сайті. Користувачеві здається, що JavaScript відсутній. Однак браузери дуже чутливі до цих помилок.

1.2.1 Мови розміток та стилів

HTML (HyperText Markup Language) - мова розмітки гіпертексту. Дозволяє створювати структуровані розділи, абзаци, посилання у веб браузерах. HTML не

надає динамічних можливостей, а отже не є мовою програмування, однак є дуже легким для верстки сайтів, та документів.

Працюючи з HTML використовують відкриваючі «< >» та закриваючі «</ >» теги для розмітки веб-сторінки. Є дуже простим та зручним у вивченні і використанні. Це основа будь-якого застосунку.

CSS (Cascading Style Sheets) - каскадні таблиці стилів. Мова дизайну, яка спрощує прикрашання на коригування веб-сторінки.

Використовуючи CSS можна редагувати текст, відступи, абзаци, анімації. Дозволяє впроваджувати дуже гарні та інтерактивні веб-додатки. Має високу популярність та стрімко розвивається у наш час.

CSS підтримується групою людей у рамках W3C, яка іще має назву Робоча група CSS.

1.2.2 Spring Framework

Spring - фреймворк на базі мови Java. Має код з відкритим доступом. Станом на сьогодні є одним із найпопулярнішим фреймворком для мови програмування Java, якщо не найпопулярніший. Використовується для створення як малих MVC застосунків, так і у великих ентерпрайз проектах.

Основні функції Spring Framework можна використовувати для розробки будь-якої програми Java, але існують розширення для створення веб-програм на основі платформи Java EE[6]. Spring Framework має на меті зробити розробку J2EE простішою у використанні та популяризує ефективні практики програмування, використовуючи модель програмування на основі POJO.

POJO (Plain Old Java Object) - це звичайний об'єкт, який не пов'язаний особливими обмеженнями. POJO тепер широко поширені завдяки простоті обслуговування. Їх легко читати і писати. Клас POJO не має угоди про іменування властивостей і методів. Він не прив'язаний до будь-якого Java Framework, будь-яка

програма Java може використовувати його. Зазвичай POJO клас містит в собі поля і методи гетери і сетери для доступу до цих полів (див. Рисунок 1.1).

Клас POJO — це клас об’єктів, який інкапсулює бізнес-логіку[7]. В архітектурі MVC контролер взаємодіє з бізнес-логікою, яка контактує з класом POJO для доступу до даних (див. Рис. 1.2).

```java
public class Employee {
    private String name;
    private int age;
    private Date birthDate;

    public Employee(String name, int age, Date birthDate) {
        this.name = name;
        this.age = age;
        this.birthDate = birthDate;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }

    public void setAge(int age) { this.age = age; }

    public Date getBirthDate() { return birthDate; }

    public void setBirthDate(Date birthDate) { this.birthDate = birthDate; }
}
```

Рисунок 1.1 - POJO клас

Під час створення застосунку буде використано POJO класи для зв’язку бізнес логіки застосунку із базою даних. Адже Java є об’єктно-орієнтованою мовою, тому все потрібно подавати у вигляді об’єктів.

В аплікаціях які виконані за допомогою фреймворку Spring використовуються Bean класи[8]. Вони є схожими до POJO класів, однак мають декілька особливостей:

-Усі файли Bean можуть бути POJO, але не всі POJO є bean-компонентами.

-Усі файли Bean можуть реалізувати інтерфейс Serializable, але не всі POJO можуть реалізувати інтерфейс Serializable.

-Поля в обох класах мають бути приватними, щоб мати повний контроль над ними.

-Властивості повинні мати гетери та сетери для доступу до них в інших програмах Java.

-Клас Bean є підмножиною класу POJO.

-Немає серйозних недоліків у використанні POJO, але деякі недоліки можуть бути у використанні класу Bean.

-POJO використовується, коли ми хочемо надати повний доступ користувачам і обмежити наших учасників. А Bean використовується, коли ми хочемо надати членам частковий доступ.

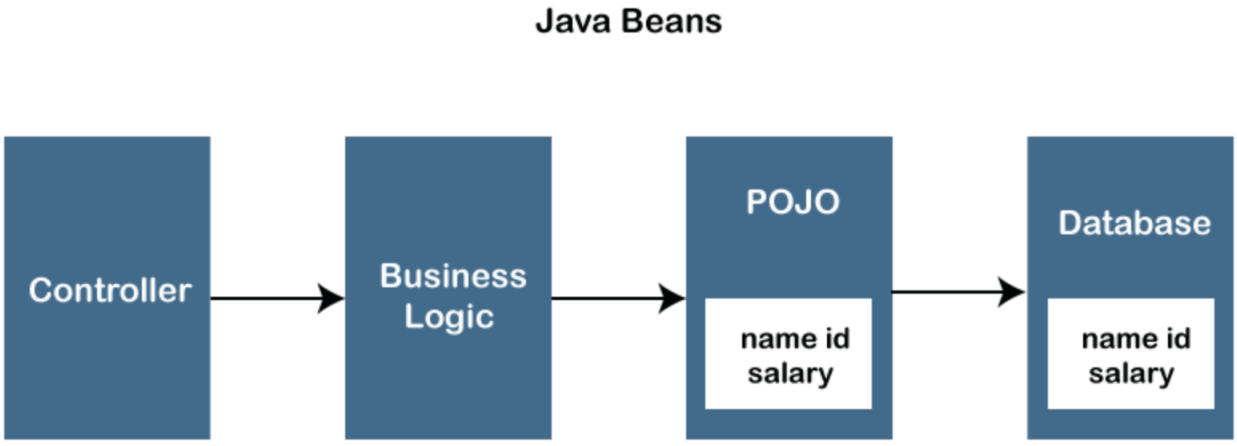

Рисунок 1.2 - Використання POJO класів при побудові архітектури застосунку

Керуючись усіма правилами класу Java Bean він повинен мати структуру зображену на рисунку 1.3.

Як видно з цього рисунку що імплементуєтсья інтерфейс Seriazable та є конструктор класу без аргументів. Цього достатньо, щоб клас вважався Bean класом.

Також відмінності між ними можна побачити у таблиці 1.1 нище.

POJO	Bean
У Pojo немає особливих обмежень, окрім угод Java.	Це особливий тип файлів POJO, які мають деякі спеціальні обмеження, окрім угод Java.
Він забезпечує менший контроль над полями порівняно з Bean.	Забезпечує повний захист на полях.
Файл POJO може реалізувати інтерфейс Serializable; але це не є обов'язковим.	Клас Bean повинен реалізовувати інтерфейс Serializable.
До класу POJO можна отримати доступ, використовуючи їхні імена.	Доступ до класу Bean можна отримати лише за допомогою геттерів і сеттерів.
Поля можуть мати будь-який із модифікаторів доступу, наприклад публічний, приватний, захищений.	Поля можуть мати лише закритий доступ.
У POJO необов'язково мати конструктор без аргументів; воно може його мати, а може і не мати.	Він повинен мати конструктор без аргументів.
У використанні POJO немає жодних недоліків	Недоліком використання Bean є те, що конструктор Default і public setter можуть змінювати стан об'єкта, коли він має бути незмінним.

Таблиця 1.1 - порівняння POJO класів та Bean класів

Вибираючи дизайн застосунку треба завжди пам'ятати про ці відмінності. Також не потрібно забувати про недоліки використання JavaBean класів:

Змінність – наші JavaBeans є змінними через їхні методи налаштування – це може призвести до проблем з багатопотоковістю чи узгодженістю.

Шаблон – ми повинні ввести геттери для всіх властивостей і сеттери для більшості, багато з цього може бути непотрібним.

Конструктор з без аргументів – нам часто потрібні аргументи в наших конструкторах, щоб гарантувати, що екземпляр об'єкта буде створено в дійсному стані, але стандарт JavaBean вимагає від нас надання конструктора з без аргументів.

```java
public class Employee implements Serializable {
    private String name;
    private int age;
    private Date birthDate;

    public Employee() {
    }

    public Employee(String name, int age, Date birthDate) {
        this.name = name;
        this.age = age;
        this.birthDate = birthDate;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }

    public void setAge(int age) { this.age = age; }

    public Date getBirthDate() { return birthDate; }

    public void setBirthDate(Date birthDate) { this.birthDate = birthDate; }
}
```

Рисунок 1.3 - Bean клас

Використвуючи усі основні переваги POJO та Bean класів Spring фреймворк дозволяє дуже швидко та ефективно масштабувати систему і так само легко її підтримувати. Серед основних переваг Spring Framework варто виділити наступні:

-Spring дозволяє розробникам розробляти програми корпоративного класу за допомогою POJO. Перевага використання лише POJO полягає в тому, що вам не потрібен продукт-контейнер EJB, наприклад сервер додатків, але ви можете використовувати лише надійний контейнер сервлетів, наприклад Tomcat, або інший комерційний продукт.

-Spring Framework організований за модульним принципом. Незважаючи на те, що кількість пакунків і класів є значною, вам доведеться турбуватися лише про ті, які вам потрібні, і ігнорувати решту.

-Spring не заново винаходить колесо, натомість він справді використовує деякі з існуючих технологій, як-от кілька фреймворків ORM, фреймворків журналювання, таймерів JEE, Quartz і JDK та інших технологій перегляду.

-Тестування програми, написаної за допомогою Spring, просте, оскільки код, що залежить від середовища, переміщується в цю структуру. Крім того, за допомогою JavaBeanstyle POJO стає легше використовувати ін'єкцію залежностей для ін'єкції тестових даних.

-Spring надає зручний API для перекладу специфічних для технології винятків (наприклад, JDBC, Hibernate або JDO) у узгоджені, неперевірні винятки.

-Легкі контейнери IoC (Inversion of Control) мають тенденцію бути легкими, особливо в порівнянні, наприклад, з контейнерами EJB. Це корисно для розробки та розгортання програм на комп'ютерах з обмеженою пам'яттю та ресурсами ЦП.

-Spring забезпечує послідовний інтерфейс керування транзакціями, який можна масштабувати до локальної транзакції (наприклад, за допомогою єдиної бази даних) і масштабувати до глобальних транзакцій (наприклад, за допомогою JTA).

Попри те, що Spring Framework був показаний світові у далекому 2003 році він і досі користуєтсья великою популярністю серед розробників на Java. Також має хорошу підтримку, що дозволяє вчасно та оперативно отриммувати всі найновіші технолоії, та користуватись ними при розробці нових додатків.

Spring Framework зібрав в собі найкращі архітектурні рішення, що для новачків може здатися навіть чимось магічним. Технологія, з якою Spring найбільше ототожнюють, — це інєкція залежностей (Dependency Injection). Інверсія контролю (IoC) є загальною концепцією, і її можна виразити різними способами. Ін'єкція залежностей – лише один конкретний приклад інверсії контролю.

Під час написання складної програми Java - класи програми повинні бути якомога незалежнішими від інших класів Java, щоб збільшити можливість

повторного використання цих класів і тестування їх незалежно від інших класів під час модульного тестування. Ін’єкція залежностей допомагає об’єднати ці класи разом і в той же час зберегти їх незалежність.

Що таке ін’єкція залежності? Давайте розглянемо ці два слова окремо. Тут частина залежності перетворюється на асоціацію між двома класами. Наприклад, клас А залежить від класу В. Тепер давайте подивимося на другу частину, ін’єкцію. Усе це означає, що клас В буде введений у клас А.

Впровадження залежностей може відбуватися шляхом передачі параметрів конструктору або шляхом пост-конструкції за допомогою методів установщика.

В об'єктно-орієнтованому програмуванні клас може покладатися на об'єкти інших класів. Ін’єкція залежностей — це стратегія, яка використовується для відділення створення об’єктів залежностей від класу, який їх потребує[9].

Існує три основні типи ін’єкції залежностей (див. Рисунок 1.4):

- За допомогою конструктора
- За допомогою сет-методів
- За допомогою інтерфейсів

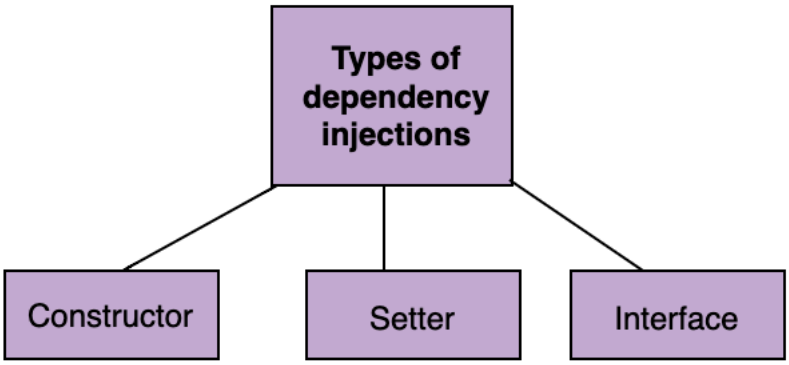

Рисунок 1.4 - три типи ін’єкцій залежностей

Ін’єкція за допомогою конструктора показано на рисунку 1.5. Залежний клас отримує потрібний йому об'єкт як параметр конструктора, а не створює його сам.

Ін’єкція залежності через сет-метод. Залежний клас має загальнодоступний сет-метод (див. Рисунок 1.6), через який впроваджується залежність.

Реалізація залежності за допомогою інтерфейса. Інтерфейс надає метод-ін’єкцію, який відповідає за впровадження залежності до будь-якого класу, якому це може знадобитися. Клас клієнта має реалізувати інтерфейс і перевизначити метод (див. Рисунок 1.7)

```
1  class Question {
2      private Answer answer;
3
4      public Question(Answer ans){
5        this.answer = ans;
6      }
7  }
```

Рисунок 1.5 - ін’єкція залежності через конструктор класу

До основних переваг реалізації ін’єкції залежностей можна віднести наступні пункти:

-Полегшує тестування, дозволяючи використовувати мок об’єкти або заглушки

-Зменшує зв’язок між класом-клієнтом і класами залежностей

-Зменшує шаблонний код, оскільки ініціалізація всіх залежностей виконується інжектором один раз

-Код легше підтримувати та повторно використовувати

```
1  class Building {
2      private Room room;
3
4      public setRoom(Room room){
5          this.room = room;
6      }
7  }
```

Рисунок 1.6 - ін’єкція залежності через публічний сет-метод класу

Також Spring Framework містить пакет аспектно-орієнтованого програмування (AOP). Функції, які охоплюють кілька точок програми, називаються наскрізними, і ці наскрізні функції концептуально відокремлені від бізнес-логіки програми. Існують різні загальні гарні приклади аспектів, включаючи журналювання, декларативні транзакції, безпеку, кешування тощо.

```
1  public interface Injector {
2    public void injectDependency(Dependency object);
3  }
4
5  public class Client implements Injector {
6    private Dependency dependentObject;
7
8    @Override
9    public void injectDependency(Dependency object){
10     this.dependentObject = object;
11   }
12 }
```

Рисунок 1.7 - впровадження залежності через реалізацію інтерфейсу

Ключовою одиницею модульності в ООП є клас, тоді як в АОП одиницею модульності є аспект. Ін’єкція залежностей допомагає вам відокремити об’єкти програми один від одного, тоді як АОП допомагає відокремити наскрізні проблеми від об’єктів, на які вони впливають.

Модуль AOP Spring Framework забезпечує реалізацію аспектно-орієнтованого програмування, що дозволяє вам визначати перехоплювачі методів і точки розрізів, щоб чітко відокремити код, який слід розділити.

Аспекти дозволяють дуже чітко налаштувати перехопювачі і реалізовувати додаткову логіку. Існує 5 основних перехоплювачів, ї можна подивитись у таблиці 1.2.

Назва	Опис
Before	Дозволяє виконувати аспектний метод перед основним.
After	Виконує аспектний метод після основного і не залежно від результату виконання.
After-returning	Виконує аспектний метод лише після успішного завершення основного методу.
After-throwing	Виконує аспектний метод лише після того якщо основний метод викинув помилку.
Around	Запускає аспектний метод до та після виконання основного методу.

Таблиця 1.2 - найпопулярніші анотації аспект-методів

В коді аспекти реалізуються за допомогою анотацій («@Aspect») над класами (див. Рисунок 1.8) та методами (див. Рисунок 1.9). Також для ідентифікації методів і класів над якими будуть виконуватись аспекті методи використовуються анотації поінткати («@Pointcut»), Наведене на рисунку 1.10 визначення pointcut вказує на виконання будь-якого методу в будь-якому класі пакета com.example.myapp.service, де ми збираємося застосувати аспект. Виклик logServiceCalls() пов’язаний із виконанням різних методів у пакеті. Розробникам це дає змогу завжди бачити та розуміти в якій послідовності та що саме буде виконуватись і чи буде взагалі.

```
@Aspect
    public class LoggingAspect {
}
```

Рисунок 1.8 - анотація над аспектним класом.

```
@Pointcut("execution(* com.example.myapp.service.*.*(..))")
public void logServiceCalls() {}
}
@After("serviceCall()") Рисунок 1.10 - анотація поінткат
public void doServiceCall(){
// After Executing serviceCall
}
@AfterReturning(pointcut = "serviceCall()", returning = "result")
public void doAfterReturnningServiceCall(Object result) {
// you can intercept result object.
...
}
@AfterThrowing(pointcut = "serviceCall()", throwing = "ex")
public void doAfterThrowingInServiceCall(Exception ex) {
// you can intercept thrown exception here.
...
}

@Around("serviceCall()")
public void doAroundServiceCall(ProceedingJoinPoint pjp){
```

Рисунок 1.9 - приклад анотацій на аспектними методами

Наведений вище код показує використання JoinPoint у едвайсах. Це дає нам можливість контролювати виконання програми. Виклик методу proceed() JoinPoint ініціює виклик serviceCall(). JoinPoint також має методи для отримання параметрів, переданих до методупозначеним анотацією @Pointcut.

1.2.3 Maven

Описаний вище Spring Framework збирається разом з кодом Java за допомогою Maven. Maven — це інструмент автоматизації та керування, розроблений Apache Software Foundation. Він написаний на мові Java для створення проектів, написаних на C#, Ruby, Scala та інших мовах. Це дозволяє

розробникам створювати проекти, залежності та документацію за допомогою об’єктної моделі проекту та плагінів.

Maven постачаєтсья з більшістю сучасних IDE. Його можна вибрати при створенні нового проєкту і відразу провести початкові налаштування, або видрати вже готові архітипи (див. Рисунок 1.11)

Maven допомагає розробнику легше створити проект на основі Java. Доступність нових функцій, створених або доданих у Maven, можна легко додати до проекту в конфігурації Maven. Це підвищує продуктивність проекту та процесу будівництва.

Основною особливістю Maven є те, що він може автоматично завантажувати бібліотеки залежностей проекту і конектити з поточним проєктом. Внутрішню архітектуру[10] Maven можна побачити на рисунку 1.20, де ми бачимо що на локальній машині міститься проєкт і Maven Core, який і виконує функцію з’єднання з віддаленими репозиторіями.

Використання Maven теж є простим і його можливо поділи на прості пункти:

-Щоб налаштувати Maven у Java, вам потрібно використовувати об’єктну модель проекту, яка зберігається у файлі pom.xml.

-POM містить усі параметри конфігурації, пов’язані з Maven. Плагіни можна налаштовувати та редагувати в тезі <plugins> файлу pom.xml. І розробник може використовувати будь-який плагін без особливої інформації про кожен плагін.

-Коли користувач починає працювати над проектом Maven, він надає налаштування конфігурації за замовчуванням, тому користувачеві не потрібно додавати кожну конфігурацію в pom.xml

За замовчуванням конфігурація проекту Maven виконується за допомогою принаймні одного файлу конфігурації pom.xml. Ця назва файлу pom походить від терміна Project Object Model.

Maven використовує декларативний стиль опису побудови, тобто ви описуєте, що має бути побудова, а не те, як вона побудована. Це дозволяє Maven використовувати за умовчанням для виконання етапів збірки.

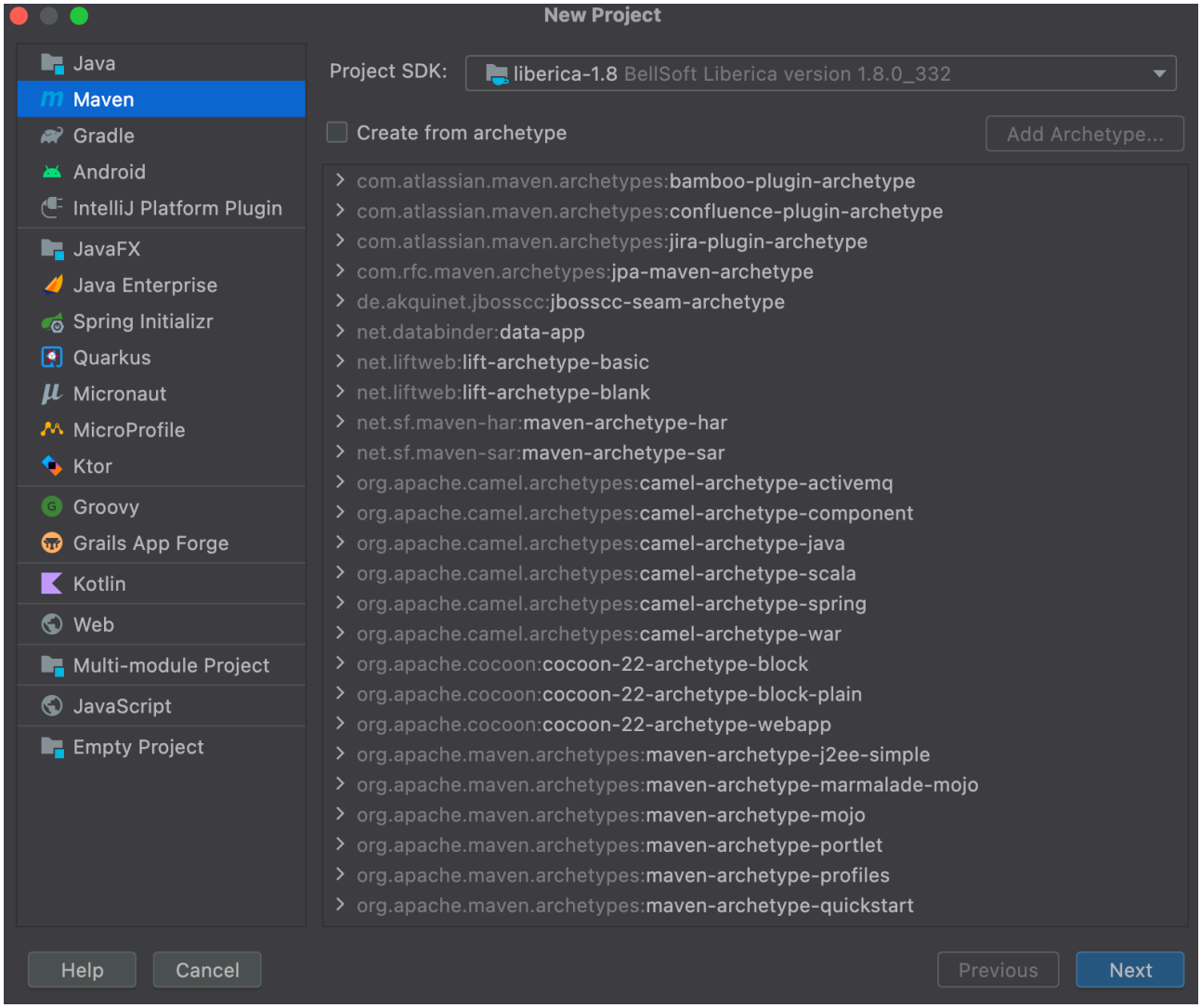

Рисунок 1.11 - вибір Maven при початкових налаштування проєкту в IntellijIDEA

Pom файл визначає:

-Ідентифікатори для проекту, який буде створено.

-Властивості, що стосуються конфігурації збірки.

-Плагіни, які забезпечують функціональність для збірки через розділ збірки.

-Бібліотеки та залежності проекту через розділ залежностей.

Кожен проект має власний файл pom. Ви можете налаштувати Maven для створення одного або кількох проектів. Мультипроект зазвичай визначає файл pom у корені каталогу та перераховує окремі проекти в розділі модулів.

Результат збірки називається артефактом і може бути, наприклад, виконуваним файлом JAR або файлом zip.

Мінімальний pom файл для одного проєкту може мати вигляд зображений на рисунку 1.12. Проєкт не містить жодних залежностей або додаткових плагінів:

1. 4.0.0 — це остання версія структури файлу pom, яку можна використовувати для Maven 2.x або 3.x
2. Унікальний ідентифікатор організації або проекту.
3. Назва проекту, який будується
4. Визначає поточну версію проекту
5. Встановити рівень компілятора Java для проекту Java

Проект Maven використовує groupId, artifactId, версію і властивості пакету для унікальної ідентифікації компонента Maven. Ці атрибути пояснюються в таблиці 1.3.

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
(1)

  <groupId>com.vogella.maven.first</groupId>
(2)
  <artifactId>com.vogella.maven.first</artifactId>
(3)
  <version>1.0-SNAPSHOT</version>
(4)

  <name>com.vogella.maven.first</name>

  <properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
(5)
    <maven.compiler.target>11</maven.compiler.target>
(5)
  </properties>

</project>
```

Рисунок 1.12 - Мінімальне наповнення pom файлу

Maven дозволяє використовувати код з інших проєктів шляхом впровадження залежностей. Репозиторій — це набір бібліотек і артефактів

проекту, що зберігаються в структурі каталогів, подібній до координат Maven проекту. Саме в репозиторії Maven зберігаються найбільші і найпопулярніші бібліотеки, але також можна підключати бібліотеки і з інших місць.

Під час початкової фази збірки Maven перевіряє, чи є у вас указана версія всіх необхідних залежностей артефактів і плагінів Maven. Якщо потрібно, він отримує їх зі сховища Maven.

Якщо необхідно, Maven завантажує ці артефакти та плагіни в локальне сховище. Локальний репозиторій за умовчанням розташований у домашньому каталозі користувача в папці .m2/repository. Якщо артефакт або плагін доступний у локальному сховищі, Maven використовує його для збірки, щоб уникнути непотрібного мережевого трафіку. Під час збирання система Maven намагається вирішити залежності модулів, які збираються. Для усунення залежностей Maven використовує наступні джерела в указаному порядку:

- Проекти, які включені в один цикл Maven (так званий реактор Maven).
- Локальний репозиторій.
- Центральний репозиторій Maven.

```
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-surefire-
plugin</artifactId>
        <version>2.22.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-failsafe-
plugin</artifactId>
        <version>2.22.2</version>
      </plugin>
    </plugins>
  </build>
```

Рисунок 1.13 - додавання плагіну в pom файл

Назва	Опис
GroupId	Визначає унікальне базове ім'я організації або групи, яка створила проект. Зазвичай це зворотне доменне ім’я або назва проекту з відкритим кодом. Для створення нових проектів groupId також визначає пакет головного класу.
ArtifactId	Визначає унікальну назву проекту в групіId. Якщо ви створюєте новий проект за допомогою Maven, це також використовується як коренева папка для проекту.
Version	Це визначає версію проекту. Якщо створено нову версію проекту, цю версію слід змінити, щоб споживачі могли бачити, що використовується інша версія.
Packaging	Визначає спосіб пакування. Це може бути напр. банку, бойову або вушну пилку. Якщо тип упаковки pom, Maven нічого не створює для цього проекту, це лише метадані.

Таблиця 1.3 - основні атрибути Maven компонента

Maven підтримує прямі та транзитивні залежності. Прямі залежності — це ті, які явно включені у pom-файл проекту. Транзитивні залежності — це залежності, необхідні нашим прямим залежностям.

Для цього ви вказуєте у файлі pom зовнішні бібліотеки, від яких залежить ваш проект, використовуючи їх GAV (groupId, artifactId і версію), потім Maven завантажує їх, розміщує у вашому локальному репозиторії Maven і робить їх доступними для збирання проекту. Також доступні транзитивні залежності необхідних бібліотек. Рисунки 1.13 та 1.14 відображають як в pom файлі повинні додаватись залежності для JUnit5.

```
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.7.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.7.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Рисунок 1.14 - додавання залежності JUnit5 до pom файлу

Додавання плагіну як зображено на рисунку 1.13 є обов'язковим, оскільки, починаючи з версії 2.22.0, Maven Surefire та Maven Failsafe забезпечують власну підтримку для виконання тестів на платформі JUnit.

Додавання залежностей на рисунку 1.14 визначає бібліотеку, яка буде використувавтись для тестування програмного забезпечення (через атрибут scope) від JUnit5 (також відомого як JUnit Jupiter).

Maven завантажить визначені залежності та їх транзитивні залежності з центрального репозиторію Maven і додасть їх до локального репозиторію користувача Maven. Якщо залежності вже знайдено у вашому локальному сховищі, Maven використає їх.Для того щоб зібрати проєкт на Maven використовуються спеціальні команди. Повна збірка проєкту поділена на етапи (див. Рисунок 1.15), основних є 8:

-Validate: Цей крок перевіряє правильність структури проекту. Наприклад, він перевіряє, чи всі залежності завантажено та чи доступні вони в локальному сховищі.

-Compile: компілює вихідний код, перетворює файли .java на .class і зберігає класи в папці target/classes.

-Test: запускає модульні тести для проекту.

-Package: цей крок пакує скомпільований код у формат, який можна поширювати, наприклад JAR або WAR.

-Integration test: запускає інтеграційні тести для проекту.

-Verify: на цьому кроці виконуються перевірки, щоб переконатися, що проект дійсний і відповідає стандартам якості.

-Install: цей крок встановлює упакований код до локального репозиторію Maven.

-Deploy: копіює упакований код у віддалений репозиторій для спільного використання з іншими розробниками.

Maven дотримується послідовного порядку виконання команд (див. Рисунок 1.16), де, якщо ви запустите крок n, усі попередні кроки (кроки 1 – n-1) також виконуються[11]. Наприклад, якщо ми запустимо крок інсталяції (крок 7), він перевірить, скомпілює, упакує та перевірить проект, а також запустить модульні та інтеграційні тести (кроки з 1 по 6) перед встановленням зібраного пакета в локальне сховище.

Також є іще один крок «clean», який не є основним, але його можна запускати додатково перед іншими, якщо потрібно переконатися, що попередня версія збірки точно буде видалена. Команда буде мати наступний вигляд: mvn clean install.

Подібним чином, якщо ми хочемо запустити крок у режимі налагодження для отримання детальнішої інформації про збірку та журналів, ми додамо -X до фактичної команди. Отже, крок встановлення з увімкненим режимом налагодження матиме таку команду: mvn -X install.

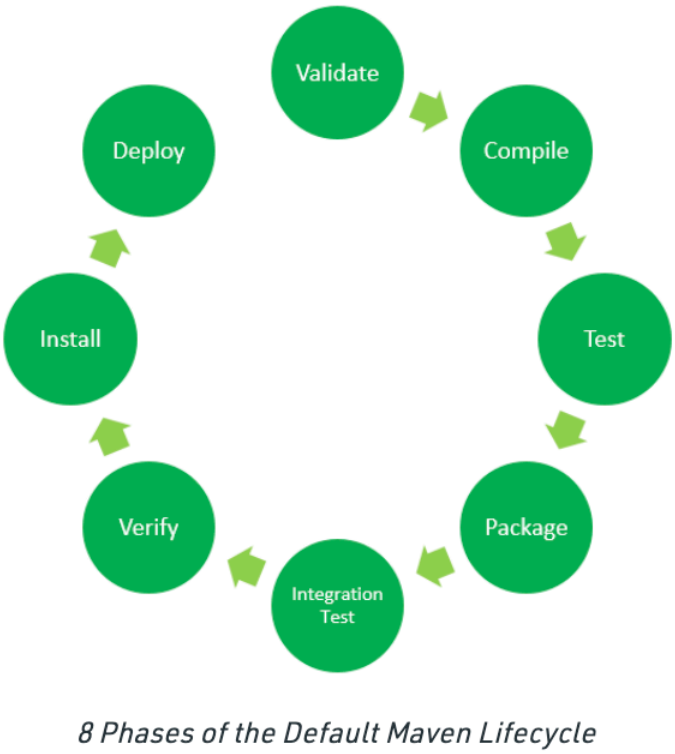

Рисунок 1.15 - основні етапи збірки проєкта Maven

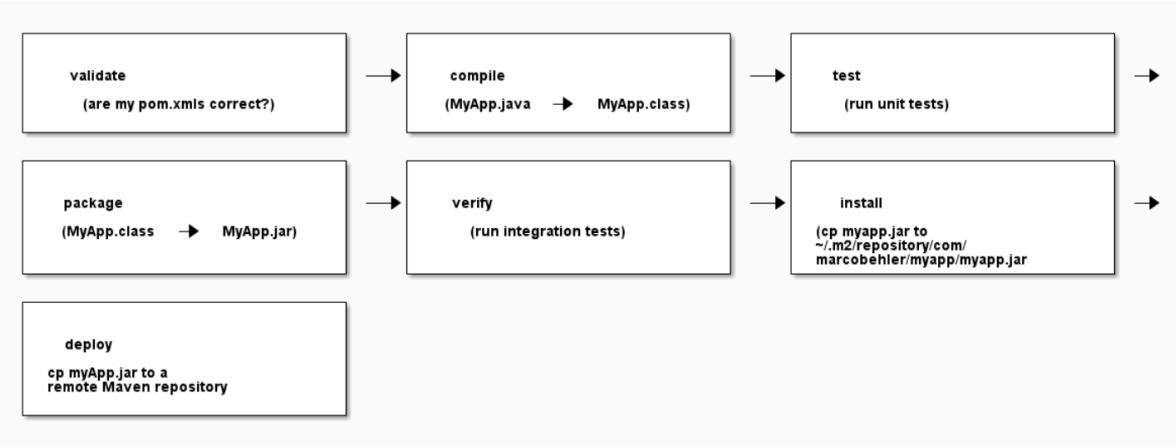

Рисунок 1.16 - Послідовність виконання етапів

Коли ми не хочемо запускати тести під час пакування або встановлення проекту Java. У цьому випадку ми використовуємо -DskipTests разом із фактичною командою. Якщо нам потрібно запустити крок встановлення, пропустивши тести, пов’язані з проектом, команда буде такою: mvn install -DskipTests. Це може заощадити нам якийсь час, щоб не запускати тести. Робити це потрібно дуже обережно, адже тести це вагома складова кожного продукту.

1.2.4 ReactJs

Бібліотека ReactJS широко використовуєтсья у веб розробці. Є дуже популярною як серед новачків, так і серед професіоналів, адже має великий перелік інструментів різної складності і для різного рівня навиків. Побудована бібліотека на основі мови програмування JavaScript, що і спрощує поріг для володіння.

Часто використовується для створення односторінкових програм, але також може бути основою мобільних програм. React.js створено для потреб Facebook. Інженери компанії шукали спосіб побудови динамічних, але високоефективних інтерфейсів, у результаті чого створили цю універсальну бібліотеку. Спочатку React.js призначався лише для внутрішніх проектів, але у 2013 році Facebook зробив його доступним у відкритому доступі. Зараз React.js є найпопулярнішою інтерфейсною бібліотекою JavaScript для веб-розробки[12].

Свою високу ефективність React.js має завдяки декільком складовим:

-Компоненти. Компоненти являють собою автономні біти коду, які можна багаторазово використовувати. Вони використовуються для того ж використання, що й функції JavaScript, але працюють ізольовано та повертають HTML. React дозволяє визначати компоненти як класи або функції за допомогою правильно реалізованих методів і успадкування. Кожен компонент має кілька методів життєвого циклу, які ви можете змінити, щоб ваш код запускався в певний час у вашій програмі. Можна перерахувати кілька типів компонентів, включаючи два основних, функціональний і класовий. Функціональний компонент — це функція, яка приймає атрибути та повертає код JSX для рендерингу в дерево DOM. Вони не мають методів стану або життєвого циклу. Компоненти класу використовуються з синтаксисом класу ES6. Вони складніші, ніж функціональні компоненти, включаючи конструктори, методи життєвого циклу, управління станом тощо. Компоненти класу можуть приймати властивості, якщо це необхідно. У React props і state — це два типи «модельних» даних, простих об’єктів JavaScript.

-Віртуальний DOM. Примітною особливістю React.js є віртуальна об’єктна модель документа або DOM. React.js створює кеш структури даних у пам’яті, обчислює отримані відмінності, а потім ефективно оновлює відображену DOM браузера, що ми називаємо процесом узгодження. Це дозволяє розробнику писати код так, ніби вся сторінка рендериться з кожною зміною, тоді як бібліотеки React рендерять лише субкомпоненти, які змінюються. Цей процес вибіркового рендерингу забезпечує значне підвищення продуктивності, а також економить зусилля, пов’язані з перерахуванням стилю CSS, макета сторінки та рендерингу всієї сторінки.

-JSX. JSX (або JavaScript XML) є розширенням синтаксису JavaScript. На вигляд схожий на HTML, JSX надає спосіб структурувати відтворення компонентів за допомогою синтаксису, знайомого багатьом розробникам. Компоненти React зазвичай пишуться за допомогою JSX, хоча вони також можуть бути написані на простому JavaScript. JSX схожий на синтаксис іншого розширення Facebook для PHP – XHP.

-Одностороння прив'язка даних. ReactJS розроблено для відстеження односторонніх потоків даних або одностороннього зв’язування даних. Односторонній процес зв’язування даних дає вам більший контроль над усією програмою. Оскільки компоненти мають бути незмінними, а дані, які вони містять, не можна змінити, якщо потік даних має інший напрямок, для цього потрібні додаткові функції. Для підтримки цієї концепції в React.js була представлена архітектура під назвою Flux. Flux — це шаблон, який допомагає зберігати ваші дані односпрямованими. Це робить програму більш гнучкою, що призводить до підвищення продуктивності.

React.js — дуже універсальна технологія, яка використовується для створення інтерфейсів, а отже, всього, що дозволяє користувачам взаємодіяти з пристроями. Інтерфейси можна знайти всюди, від мобільних додатків до веб-сайтів, отже існує багато варіантів використання цієї бібліотеки.

Основним прикладом використання Reactjs є односторінкова програма. React SPA працює всередині браузера і не вимагає від користувача перезавантаження сторінки. Крім того, коли користувач переходить через SPA, він

продовжить взаємодіяти з тією самою сторінкою, не взаємодіючи з абсолютно новою сторінкою. Багато відомих програм, які використовуються щодня, є SPA, наприклад Gmail, Google Maps або Facebook.

Оскільки всю програму можна акуратно розбити на менші багаторазово використовувані елементи інтерфейсу, багато розробників вважають React.js найбільш придатним для розробки корпоративних програм. Компонентна архітектура React.js допомагає вирішувати проблеми застарілого стеку за допомогою дуже швидкої адаптації та циклів впровадження. Звичайно, швидкість розробки, стабільність React.js, який підтримується та вдосконалюється Facebook, і популярність цієї технології серед розробників також є причинами, чому компанії вирішують використовувати React.js.

За допомогою фреймворку React Native і, отже, також за допомогою React.js можна створювати кросплатформні мобільні програми. Якщо Reactjs уже використовується у веб-програмі, набагато простіше оновити мобільну версію, а також повністю перенести веб-версію на React Native. Мобільний додаток React Native працює як нативний додаток на IOS та Android. Залучаючи розробників Reactjs, ви також зможете створювати мобільні програми React Native[13].

З наведеного вище можна зрозуміти, що React.js є дуже обожнюваною технологією на ринку. До основних переваг можна віднести:

-Легко освоїти – вивчення ReactJS, на відміну від інших подібних технологій, вважається більш керованим і приємним процесом. Розуміючи ReactJS, ви можете відразу почати створювати високочутливі програми. Це дозволяє легко переглядати та вивчати функції розробки мобільних і веб-додатків, не вимагаючи шаблонів, шаблонів чи архітектури. Він має вбудовані формули та функції, які можна комбінувати для створення ефективних проектів за менший час, ніж це знадобилося б для створення коду рядок за рядком. Знання синтаксису JavaScript може бути корисним, і найчастіше розробники JS досліджують ReactJS, але це не обов’язково.

-Швидший рендеринг і надійна продуктивність – запуск Virtual DOM розробниками Facebook зробив React JS одним із більш ефективних і менш загальмованих технологічних рішень для розробки додатків. DOM (Document

Object Model) має великий вплив на потенційне навантаження на програму завдяки своїй структурі. Методи DOM дозволяють програмний доступ до дерева, і за допомогою них ви можете змінити структуру, стиль або вміст документа. Оскільки DOM сучасних веб-сайтів величезний, оновлення можуть тривати довго, сповільнюючи загальну продуктивність веб-програми. Браузер регулярно перевіряє будь-які зміни в DOM і відповідно оновлює його. Зміна може бути викликана введенням користувача, запитом, отриманням даних тощо. Браузер оновлює DOM щоразу, коли відбувається зміна. Використання віртуального DOM, який є меншим і може швидко оновлюватися, дуже позитивно впливає на продуктивність програми. Лише після взаємодії між веб-сайтом і віртуальною DOM фактична DOM оновлюється. Результатом є краща взаємодія з користувачем, вища продуктивність і чудові можливості.

-Полегшує написання компонентів – серед усіх корисних функцій ReactJS також є ті, які візуалізують те, що відбувається в коді JavaScript, який працює з графічним інтерфейсом. Це JSX. Це додаткове розширення синтаксису JavaScript із можливістю вставляти теги, що значно полегшує написання компонентів. Це також допомагає React відображати кориснішу інформацію про помилки та попередження, приймає HTML-цитування та полегшує візуалізацію підкомпонентів. На практиці це набір ярликів для написання «елементів React» з декількома правилами, щоб зберегти ваш код прозорим і м’яким.

-Повторне використання коду – ReactJS підтримує створення багаторазових компонентів. Після створення такий елемент інтерфейсу можна використовувати в інших частинах вашого коду, у різних інших проектах, іноді з невеликими змінами або без них. Також це поєднується з можливістю використання безкоштовних бібліотек готових компонентів. Це також позитивно впливає на ключові особливості бізнесу, тобто економію часу та грошей завдяки відсутності необхідності писати все з самого початку.

-SEO дружелюбний – завдяки сучасним тенденціям ця функція є надзвичайно корисною. Фреймворки JavaScript зазвичай не дуже сприятливі для SEO, і веб-додатки втрачають позиції в пошукових системах з оптимізованими сторінками. Можливість реагувати на типові проблеми з пошуковими системами є

ще однією перевагою використання фреймворку React JS. Це дозволяє розробникам створювати привабливі інтерфейси користувача, які можна легко переглядати через різні пошукові системи.

-Широко використовуваний і популярний – ReactJS є популярним вибором також серед відомих світових брендів. Успішно реалізовані додатки або веб-сайти таких компаній, як Airbnb, Tesla, Netflix та інші, доводять, що з цією технологією можна досягти багато чого.

-Було перечислено дуже багато переваг і вони є дійсно значущими, але також React.js має і декілька недоліків:

-Високі темпи розробки – це як недолік, так і перевага в залежності від проекту. Однак, оскільки бібліотека постійно і швидко змінюється, деякі розробники можуть почуватися незручно, вивчаючи нові способи реалізації певних елементів. Їм може бути важко працювати з технологіями, які постійно змінюються. Зрештою, це вимагає від них завжди бути в курсі своїх здібностей і вивчати нові способи роботи.

-Погана документація – поширена проблема, коли технології постійно оновлюються. React.js модернізується та пришвидшується настільки швидко, що немає часу робити документацію. Щоб виправити це, розробники самостійно пишуть підручники та інструкції, коли розробляються нові версії.

-Не є повноцінним інструментом – ReactJS охоплює лише рівні інтерфейсу додатка, тому для отримання повного набору інструментів для розробки проекту все ще потрібна технологія.

-Сторонні бібліотеки – при розробці проектів з React.js майже завжди потрібно використовувати додаткові бібліотеки. Якщо хтось хоче мати всі свої інструменти під рукою одразу, не дивлячись далі, він може розчаруватися. Перевагою такої ситуації є легкість бібліотеки React, яка розроблена для невеликої кількості завдань, але справляється з ними добре.

-Проблемний JSX – ReactJS використовує JSX для поєднання HTML і JavaScript, що може мати багато переваг, але деякі розробники вважають JSX перешкодою, особливо для новачків. Вони в основному скаржаться на його складність у навчанні.

1.2.5 Tomcat

Tomcat - По суті, це сервлет Java з відкритим вихідним кодом і контейнер Java Server Page, який дозволяє розробникам реалізувати низку корпоративних програм Java. Tomcat також запускає середовище веб-сервера HTTP, у якому може працювати код Java.

Через три роки після оригінального випуску Java у 1995 році архітектор Sun Microsystems Джеймс Дункан Девідсон розробив довідкову реалізацію сервлету з відкритим кодом для першого Java Servlet API. Сервлети Java — це невеликі програми Java, які визначають, як сервер обробляє відповіді та запити. Розробник написав би свій сервлет або JSP і дозволив Tomcat виконувати всю роботу з маршрутизації та серверної частини.

Tomcat підтримується багатьма розробниками в його спільноті, і його поточною найстабільнішою версією є серія 9.0, яка є першим випуском Apache Tomcat, який підтримує специфікації Servlet 4.0. Tomcat також постачається з двигуном Coyote, веб-сервером, який дозволяє розробникам підключати низку корпоративних додатків і можливостей Java[14].

Якщо ви трохи знайомі з веб-сайтами або маєте деякі базові знання про веб-сайти, ви, мабуть, чули про протокол HTTP або, можливо, знаєте, що це насправді. Якщо ви хочете надати будь-які веб-сервіси, наприклад, ви хочете надати простий статичний вміст, можливо, за допомогою HTML (або мови розмітки гіпертексту), або, можливо, ви просто хочете надіслати дані з сервера, щоб вказати вам, то вам обов’язково потрібен і цим сервером є HTTP (протокол передачі гіпертексту). Отже, як ми всі знаємо, якщо хтось хоче створити простий статичний веб-сайт, йому обов’язково потрібен HTTP-сервер, але якщо він хоче зробити веб-сайт динамічним, він повинен використовувати сервлет. Ми використовуємо HTTP-сервер, якщо хочемо надіслати прості дані. Якщо ми хочемо надіслати динамічні дані або зробити наш веб-сайт динамічним, нам потрібно використовувати сервлет. Отже, нам потрібен HTTP-сервер, а ще нам потрібен контейнер, де ми будемо запускати, або сервлет, тому, коли ми об’єднуємо HTTP-сервер і сервлет

(або, можна сказати, контейнер сервлетів), вони обидва об’єднуються, щоб стати одним сервером. як сервер tomcat.

Контейнер сервлетів — це в основному реалізація специфікації сервлетів Java, яка в основному використовується для розміщення сервлетів Java. Можна сказати, що в центрі Tomcat є JSP (Сторінки сервера Java) і Servlet. JSP — це одна з технологій програмування на стороні сервера, яка дозволяє розробникам створювати незалежний від платформи динамічний вміст і також відома як технологія візуалізації перегляду на стороні сервера. Сервлет — це програмний компонент на основі Java, який допомагає розширити можливості сервера. Однак він також може відповідати на кілька типів запитів і зазвичай реалізовані контейнери веб-серверів для розміщення веб-додатків на веб-серверах. З точки зору розробника, нам просто потрібно написати сторінки сервера Java (або JSP) або сервлет і не потрібно турбуватися про маршрутизацію; Tomcat оброблятиме маршрутизацію.

Tomcat також складається з веб-сервера, відомого як двигун Coyote, завдяки якому можна розширити можливості Tomcat, включивши кілька корпоративних специфікацій Java, включаючи Java Persistence API (JPA). Tomcat також має розширену версію, відому як "TomEE", яка містить більше корпоративних функцій.

Не спроста Tomcat є дуже популярним у світовій спільноті програмістів. Ось основні переваги, за які його цінують[15]:

-Це відкритий код. Це означає, що будь-хто з будь-якого місця може завантажити, встановити та використовувати його безкоштовно, що робить його першим вибором серед нових розробників і нових користувачів.

-Насправді це дуже легка програма, навіть із сертифікацією JavaEE. Однак він забезпечує всі необхідні та стандартні функції, необхідні для роботи сервера, що означає, що він забезпечує дуже швидке завантаження та повторне розгортання порівняно з різними альтернативами.

-Завдяки вбудованим параметрам налаштування, широкій і легкій природі, він пропонує високу гнучкість, користувач може запускати його будь-яким способом, який забажає, і він все одно працюватиме нормально без жодних

проблем. Оскільки це відкритий вихідний код, кожен, хто має знання, може налаштувати його відповідно до своїх вимог.

-Це одна з найстабільніших платформ, доступних на сьогодні, для створення та використання для запуску наших програм. Він неймовірно стабільний, оскільки працює незалежно від нашої інсталяції Apache. Якщо в Tomcat виникне великий збій, через який він перестане працювати, решта нашого сервера працюватиме нормально.

-Оскільки деякі організації зазвичай люблять розміщувати свою установку Tomcat за захистом додаткового брандмауера, доступ до якого можна отримати лише з інсталяції Apache.

-Він містить декілька доступних чудових документів, включаючи широкий спектр безкоштовно доступних онлайн-навчальних посібників, які користувачі можуть завантажити або переглянути безпосередньо в режимі онлайн, що робить його одним із найкращих варіантів для задоволення вимог до сервера додатків у більшості веб-сайтів Java. Незалежно від того, чи шукає користувач інструкції зі встановлення, параметри запуску, примітки щодо конфігурації сервера, уся інформація про Tomcat уже доступна в Інтернеті.

-Згідно з оцінкою, він займає майже 60 відсотків частки ринку майже всіх розгортань серверів додатків Java, що робить його одним із найпопулярніших серверів додатків, які використовуються для веб-додатків Java. Однак ми не можемо сказати, що він реалізує всі функції, необхідні для сервера додатків JavaEE; натомість це дозволяє нам запускати програму Java EE.

-Tomcat існує майже 20 років, що є досить значним часом, протягом якого він стає зрілим з часом. Оскільки Tomcat є програмним забезпеченням з відкритим вихідним кодом, воно оновлюється, і нові випуски виходять майже регулярно, і спільнота відкритих вихідних кодів підтримує його. Зрілість робить його одним із найбільш стабільних серверів додатків для розробки програмного забезпечення, додатків і розгортання додатків Java. Відтепер це надзвичайно стабільний варіант, який стає потужнішим завдяки чудовій підтримці спільноти.

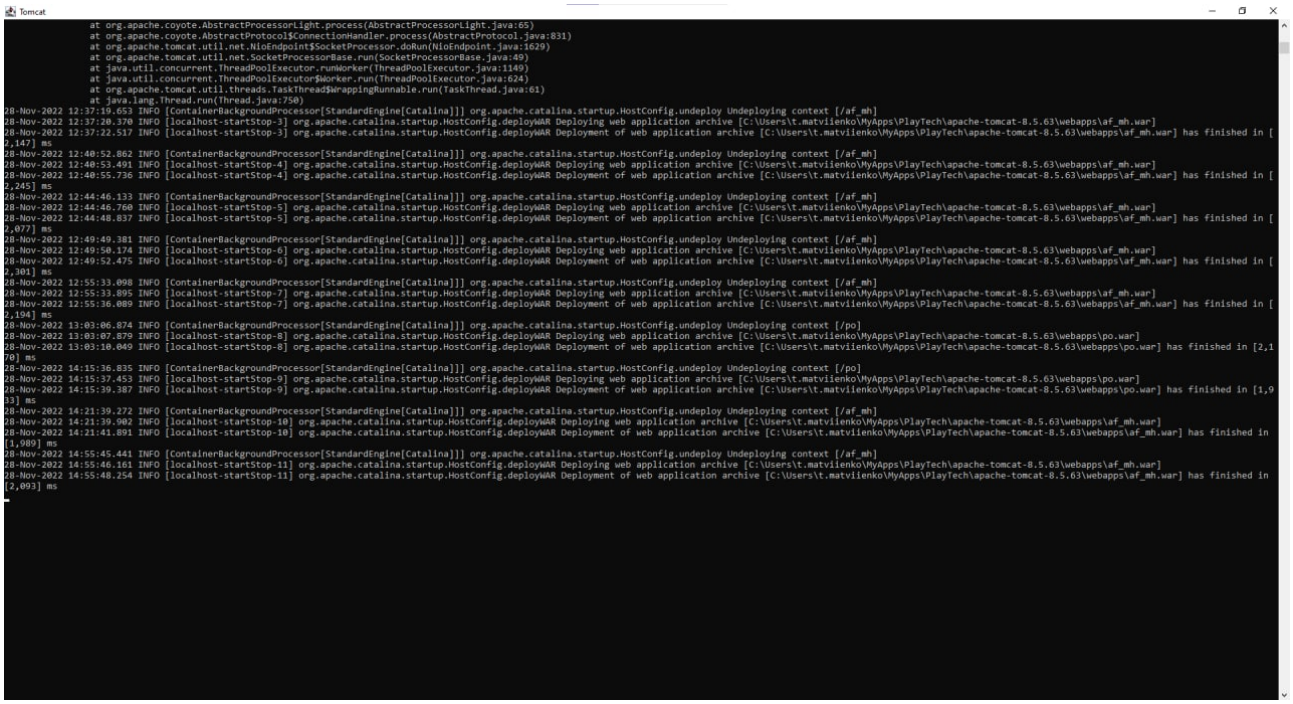

Рисунок 1.17 - вигляд командного рядку запущеного Tomcat серверу

В процесі розробки мого застосунку Tomcat буде використовуватись у якості сервера для обробки інформаціїї, і спілкування клієнта з сервером.

Розгортання Tomcat відбувається різними способами: через командний рядок, за допомогою .bat файлу, який запустить Tomcat. Я буду використовувати саме другий варіант. Запущений Tomcat буде виглядати як зображено на рисунку 1.17.

1.3 Зв’язок між комонентами застосунку

Зв’язок від клієнта до сервера реалізовується веб протоколами. Вони повинні гарантувати надійність, безпеку та швидкість. Для різних потреб можна застосовувати різні методи підходи та протоколи, проте існують найпопулярніші, які використовують у веб розробці.

Протокол передачі гіпертексту (HTTP) - протокол який застосовуєтсья для завантаження даних на веб сторінки. HTTP — це протокол прикладного рівня,

призначений для передачі інформації між мережевими пристроями та працює поверх інших рівнів стеку мережевих протоколів. Типовий потік через HTTP передбачає, що клієнтська машина надсилає запит серверу, який потім надсилає відповідне повідомлення[16].

HTTP має п’ять версії станом на сьогодні - HTTP/0.9, HTTP/1.0, HTTP/1.1, HTTP/2.0 та HTTP/3.0. Сьогодні в загальному корисутванні є версія HTTP/1.1 та HTTP/2.0.

Найстарішим є протокол HTTP/0.9 який з’явився в кінця 20 століття вже майже не використовуєтсья сьогодні. Містив тільки один метод - PUT Хоча деякі популярні сервіси, такі як Apache, Nginx, досі підтримують цей протокол.

Протокол HTTP/1.0 з’явився приблизно за 5 років після свого попередника HTTP/0.9 та приніс багао покращень: заголовки, версія запиту, статус-код, тип контенту що передаєтсья та нові методи - POST та HEAD.

Протокол версії HTTP/1.1 широко використовуєтсья сьогодні. Він приніс виправлення та покращення функцій до попередніх версій - постійні та конвеєрні з’єднання, передачі фрагментів, стиснення/декомпресію, узгодження вмісту, віртуальний хостинг (сервер з однією IP-адресою, на якому розміщено кілька доменів), швидшу відповідь і значну економію пропускної здатності завдяки доданню підтримки кешу.

Також з’явились нові методи[17]: GET , HEAD , POST , PUT , DELETE , TRACE , OPTIONS.

Тип з’єднання змінився на довго-відкритий.

Перед встановленням будь-якого з’єднання відбувається тристороннє рукостискання TCP. Наприкінці, після надсилання всіх даних клієнту, сервер надсилає повідомлення про те, що більше немає даних для надсилання. Потім клієнт закриває з'єднання (TCP teardown).

В HTTP/1.0 TCP з’єднання переривалися «потрійними рукостисканнями» між кожним GET запитом(див. Рисунок 1.18)

У версії HTTP/1.1 лише одне TCP з’єднання буде триматися відкритим і кілька GET запитів зможуть бути виконані без переривання на «потрійне рукостискання» (див. Рисунок 1.19)

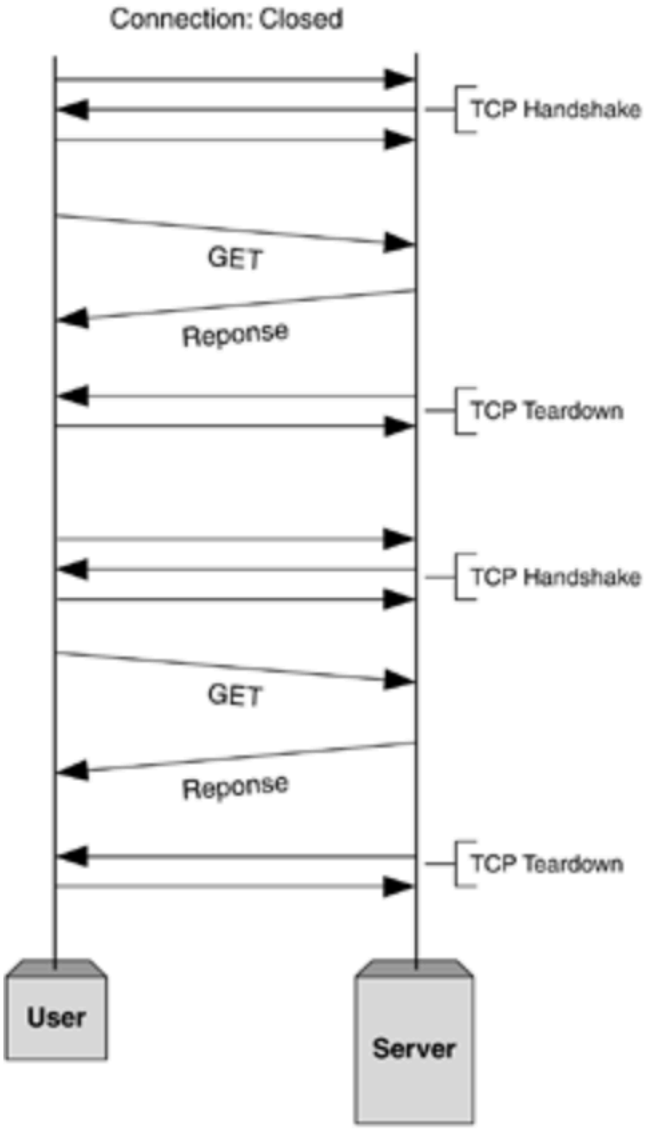

Рисунок 1.18 - GET запит в HTTP/1.0

Keep-Alive заголовок в HTTP/1.1 зробив постійні підключення типовою поведінкою. Заголовок Keep-Alive можна використовувати для визначення політик довготривалого зв'язку між хостами (тобто дозволяє з'єднанню залишатися активним, доки не відбудеться подія). Це заклало основу для стійкості, багаторазових з'єднань, конвеєрної обробки та багатьох інших розширених можливостей у сучасних протоколах веб-зв'язку.

Клієнт, сервер або будь-який посередник можуть незалежно надавати інформацію для заголовка Keep-Alive. Крім того, хост може додати параметри

тайм-ауту та максимального значення, щоб установити час очікування або обмежити максимальну кількість запитів на з’єднання (див. Рисунок 1.6)

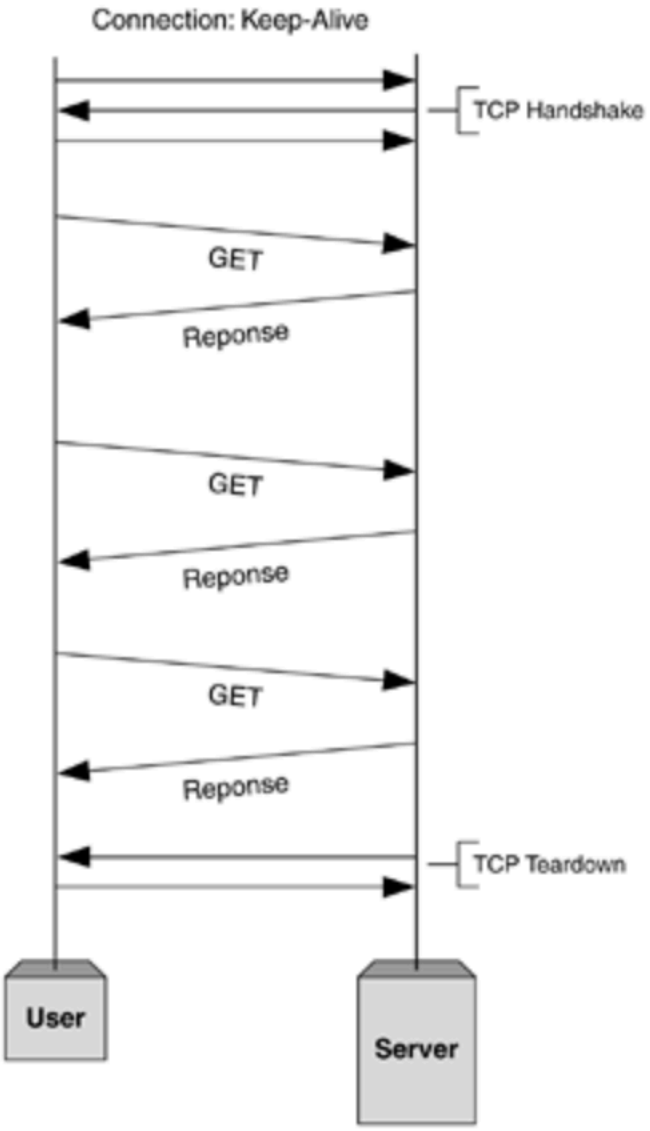

Рисунок 1.19 - GET запит в HTTP/1.1

Удосконалений заголовок який був представлений разом з HTTP/1.1 вніс вагомі зміни. Тепер дозволено запустити підключення за допомогою широко використовуваного протоколу, наприклад HTTP/1.1, а потім запитати, щоб підключення перейшло на розширений тип протоколу, наприклад HTTP/2.0 або WebSockets.

Ще один популярний протокл це HTTPS (Hyper Text Transfer Protocol Secure) захищений варіант звичайного HTTP. Він використовує SSL/TLS для безпечного зашифрованого зв’язку.

З’єднання HTTPS може захистити передачу даних від атак типу «людина посередині» та типових загроз безпеці, забезпечуючи двонаправлене шифрування для зв’язку між клієнтом і сервером.

Хоча протокол HTTPS захищений за своєю схемою, процес встановлення зв’язку SSL/TLS займає значний час перед встановленням з’єднання HTTPS. Зазвичай це займає 1–2 секунди та різко сповільнює запуск веб-сайту.

HTTP версії 2.0 було офіційно випущено в 2015 році. Це бінарний протокол, який замінив попередній текстовий. Зокрема, HTTP 2.0 зосереджений на покращенні продуктивності:

-Мультиплексування запитів. HTTP 1.1 є послідовним протоколом. Таким чином, ми можемо надсилати один запит за раз. HTTP 2.0, у свою чергу, дозволяє надсилати запити та отримувати відповіді асинхронно. Таким чином, ми можемо виконувати кілька запитів одночасно, використовуючи одне з’єднання.

-Пріоритезація запитів. За допомогою HTTP 2.0 ми можемо встановити числовий пріоритет у групі запитів. Таким чином, ми можемо чітко вказати, в якому порядку ми очікуємо відповіді, наприклад отримати CSS веб-сторінки перед її файлами JS

-Автоматичне стиснення. У попередній версії HTTP/1.1 ми повинні явно вимагати стиснення запитів і відповідей. Однак HTTP 2.0 автоматично виконує стиснення GZip

-Скидання з’єднання. Функція, яка дозволяє закривати з’єднання між сервером і клієнтом з певної причини, таким чином негайно відкриваючи нове.

-Сервер push: щоб сервер не отримував багато запитів, HTTP 2.0 запровадив функцію серверу push. При цьому сервер намагається передбачити ресурси, які незабаром будуть затребувані. Отже, сервер активно надсилає ці ресурси в кеш клієнта

Саме за допомогою HTTP запитів система координації та визначення критичного шляху переміщення рухомого складу буде організовувати спілкування

клієнт - сервер. Це значно полегшить роботу для мене, як для розробника, адже такі системи себе добре зарекомендували, та існує безліч цікавих реалізацій систем такого типу.

Кожен HTTP-запит містить в собі закодовані дані, які зберігають різні типи інформації. Типовий HTTP-запит містить:

1.HTTP версія - версія HTTP запиту, про які було згадано вище.

2.URL (Uniform Resource Locator) - стандартизоване іменування адрес в інтернеті.

3.HTTP метод - позначає чи очікує клієнт на якусь відповідь від сервера. Найчаще вживаними методами є «POST» та «GET». «GET» запит очікує на певну відповідь від сервера. А метод «POST» лчше передає дані на сервер для обробки її там, та без повернення назад.

4.HTTP заголовок - містить ключову інформацію у вигляді ключ-значення і передаєтсья у кожному запиті.

Опціонально HTTP запит також може містити тіло запиту, яке зберігає додаткову інформацію що повинна бути передана на сервер. Наприклад інформація, що була введена в певну форму.

Протокол керування передачею (TCP) — це протокол, який використовується у веб розробці для встановлення з'єднання між двома віддаленими застосунками та доставки надійного потоку даних від однієї до іншої[18].

Технологія була розроблена в 1970-80-х роках, власне коли був створений сам Інтернет з такою ідеєю як ми використовуємо його зараз. У статті 1974 року під назвою «Протокол для пакетної мережевої взаємодії» інженери Вінт Серф і Боб Кан вперше описали TCP, його робочу структуру та основні принципи. Він мав два основних компоненти:

-Орієнтовані та з'єднання посилання. Сервер прослуховує запити від клієнта перед тим як встановити з'єднання та почати передавати данні.

-Телеграмна орієнтованість. Інвормація передаєтсья від сервера до клієнта без підтвердження доставки.

Протокол спочатку називали мережевою моделлю Міністерства оборони (DOD), оскільки TCP та інші Інтернет-протоколи в основному використовувалися для військових випадків і досліджень. Протягом багатьох років TCP став частиною загальнодоступної інтернет-інфраструктури, яку ми використовуємо сьогодні, і тепер є синонімом набору протоколів Інтернету (IP). TCP/IP є основою сучасного Інтернету.

Також окрім TCP, мережі використовують різноманітні протоколи, такі як протокол дейтаграм користувача (UDP), простий транспортний протокол пошти (SMTP), протокол передачі файлів (FTP), протокол передачі гіпертексту (HTTP), захищений протокол передачі гіпертексту (HTTPS), і кілька інших.

Протокол TCP працює за принципом - сервер завжди моніторить запити від клієнта:

-«Пасивно відкритий» стан серверу: налаштування за яких сервер чекає на запит про з’єднання від клієнтів. Сервер «прослуховує» з’єднання при цьому самого з’єднання не встановлює.

-Клієнт при цьому знаходиться в стані «активно відкритий». Клієнт повинени надіслати запит про синхронізацію до сервера. Тоді сервер витрачає ресурси на те щоб прийняти з’єднання та обробити його.

-Надійне з’єднання встановлюється через тристороннє рукостискання, що є однією з центральних функцій TCP. Він гарантує, що підключення встановлено безпечно та надійно, гарантуючи, що воно виконує три умови:

-SYN: Клієнт надсилає на сервер повідомлення синхронізації, яке являється унікальним цифровим значенням.

-SYN-ACK: сервер надсилає повідомлення підтвердження синхронізації (або SYN-ACK), яке складається з двох частин – значення цифрового запиту від клієнта SYN + 1 і повідомлення ACK, яке також є числовим значенням. Клієнт отримує SYN-ACK значення.

-ACK: клієнт відповідає власним підтвердженням, яке є цифровим значенням ACK + 1. Цей крок у тристоронньому рукостисканні встановлює з’єднання клієнт-сервер. Програми, розміщені на клієнті, тепер можуть

спілкуватися з програмами, розміщеними на сервері, використовуючи надійне та безпечне з’єднання.

- TCP протокол використовує повторну передачу для підвищення надійності: повторна передача або автоматизований повторний запит (ARQ) — це метод контролю помилок, який повторно надсилає пакети даних, якщо вони втрачені під час передачі по топології мережі. Між тим як сервер надішле дані і клієнт їх отримає може пройти певний проміжок часу. Цей інтервал називається тайм-аутом, і якщо адресат не підтвердить отримання до закінчення тайм-ауту, джерело надішле пакет ще раз.

- Протокол TCP грантує що дані передаютсья правильні, безпечні та без пошкоджень. Для цьооо використовуються три інструменти:

1.Контрольна сума: TCP групує байти в повідомленні на сегменти, і кожен компонент має обов’язкове поле контрольної суми, яке є 16-бітним значенням. Клієнт перевірить дані в полі контрольної суми на цілісність і, якщо вони пошкоджені, не надішле ACK.

2.Тайм-аут: Тайм-аут – це максимальний інтервал, який може пройти між створенням і отриманням даних. Це гарантує, що з’єднання не залишається відкритим надто довго, і мінімізує вплив зловмисників.

3.Підтвердження: сервер і клієнт обмінюються значеннями ACK для підтвердження передачі даних. Якщо потік даних не підтверджено, то протокол намагається повторити передачу. Крім того, якщо три послідовні значення ACK однакові, TCP ініціює повторну передачу.

Загалом стек протоколу TCP складаєтсья з чотирьох рівнів[19]: прикладного рівня, транспортного рівня, мережевого рівня та рівня зв’язку. Перший передає дані з однієї програми в іншу, другий передає дані між процесами, третій забезпечує зв’язок між двома хостами, а останній рівень забезпечує абстракцію, щоб приховати деталі фізичної мережі. Це гарантує, що користувачі можуть отримати переваги від рівнів моделі TCP, не заглиблюючись у складну базову архітектуру комп’ютерної мережі чи функціональні можливості.

Прикладний рівень. Цей рівень керує комунікаціями між веб-додатками або службами на кожному кінці TCP-опосередкованого обміну, тобто між клієнтом і

сервером. Він налаштовує, координує та завершує потоки даних між програмами, дотримуючись правил моделі TCP і синтаксичних протоколів. Це також забезпечує передачу даних у стандартизованому форматі, щоб клієнт зміг зрозуміти інформацію та представити її кінцевому користувачеві для споживання. У випадку програмного забезпечення, розміщеного в Інтернеті, прикладний рівень визначає, чи потребує програмне забезпечення підключення до мережі.

Транспортний рівень. Цей рівень взаємодіє з даними, згенерованими попереднім рівнем, і готує їх для передачі через мережеве програмне забезпечення та апаратну інфраструктуру. Транспортний рівень важливий для виконання правил TCP.

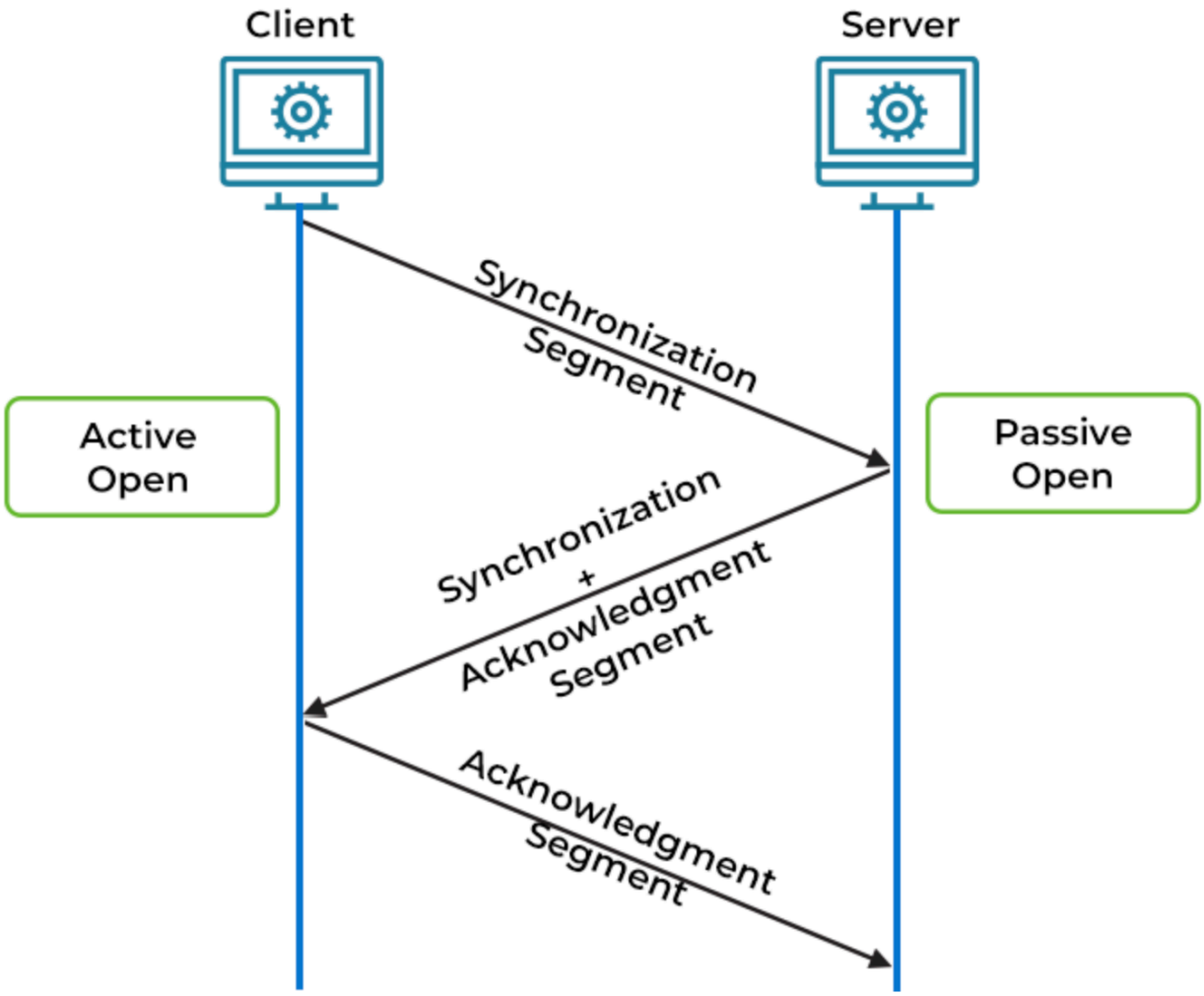

Рис. 1.20 Функціонування протоколу керування передачею (TCP)

Прикладний рівень генерує дані лише у форматі, який підтримує TCP, тоді як саме транспортний рівень забезпечує дотримання всіх правил TCP під час передачі даних. Вбудовані механізми забезпечують надійність і автоматичне відновлення помилок для безперебійного потоку інформації. Клієнт і сервер можуть постійно обмінюватися інформацією без збоїв.

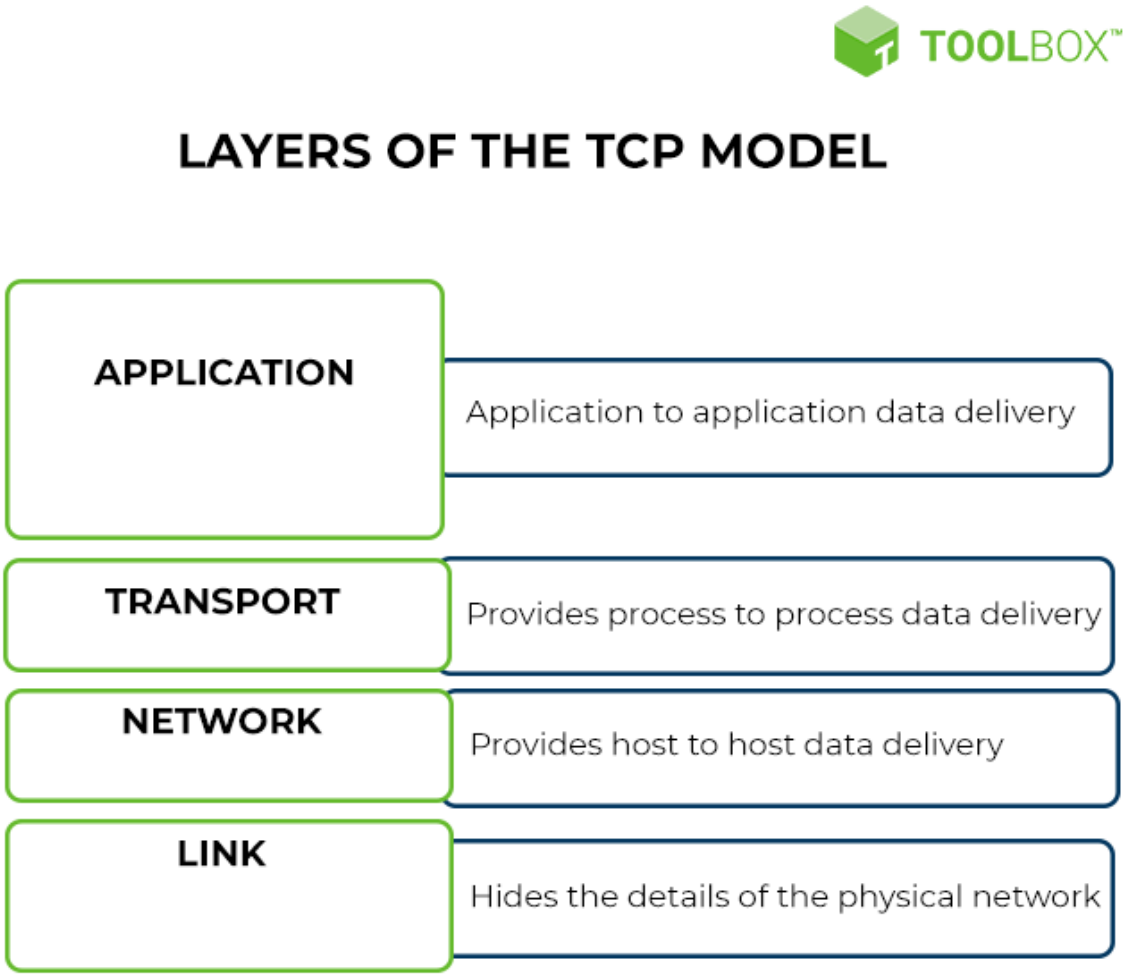

Рис. 1.21 Рівні TCP моделі

Мережевий рівень. Цей стековий рівень протоколу керування займається взаємодією даних між кількома сегментами та компонентами мережі. Він створює логічні шляхи між двома хостами (клієнтом і сервером), вибираючи найбільш надійний і ефективний шлях.

Пакети даних можуть подорожувати безліччю шляхів, перш ніж досягти місця призначення, і вибраний маршрут визначатиме затримку, вразливість системи безпеки та вплив на цілісність. Мережевий рівень оцінює фізичну інфраструктуру, яка живить Інтернет, щоб визначити ідеальний шлях передачі, дотримуючись правил TCP.

Рівень посилань. Рівень зв’язку працює на рівні локального пристрою або локального середовища. Це забезпечує підключення до Інтернету для кінцевих пристроїв і програмних додатків, зберігаючи належну абстракцію. Абстракція в обчислювальній техніці — це процес представлення лише релевантних атрибутів процедури або досвіду, щоб уникнути перевантаження користувача.

Рівень зв’язку стеку TCP використовує топологію мережі, встановлені драйвери та інші локальні компоненти для встановлення підключення до Інтернету. Цей рівень також включає мережеві інтерфейси, які адміністратори можуть використовувати для більш детального контролю. Ці рівні працюють разом, щоб забезпечити підключення до Інтернету через протоколи керування передачею. Пам’ятайте, що кілька протоколів можуть поєднуватися та співіснувати на різних рівнях для виконання окремих функцій.

У цьому розділі було розглянуто засоби розробки програмного продукту. Розкрито основні інструменти, які будуть використовуватись. Надано опис інструментів та перераховані їхні переваги, що дозволяє підтвердити доцільність використання саме цих засобів для розробки сучасного веб-застосунку.

2 Проєктування системи координації та визначення критичного шляху переміщення рухомого складу.

2.1 Вхідні дані

Вхідні данні до застосунку будуть приходити від Інтерфейсу користувача за допомогою реалізації API.

API(Application programming interface) - інтерфейс, який декларує методи і способи зв’язку між користувацьким інтерфейсом та системою, яка ці запити обробляє. Тобто реалізує спосіб та методи для спілкування між різними частинами ПЗ.

Існує декілька протоколів API (див. Рисунок 2.1) які широко використовуються у світі і до сьогодні[20]. Серед них:

-SOAP (simple object access protocol) - використовує XML файли для передавання інформації між користувачами. Використовується у давніх проєктах, та не є дуже популярним сьогодні, через новітні вимоги до API.

-gRPC (google remote procedure call) - протокол що базуєтсья на відкритому коді та може бути використаний у багатьох розробках. Має кілька вагомих переваг, за що і здобув свою популярність.

-GraphQL (graph query language) - розроблена у 2015 році компанією Facebook, а зараз Meta, мова для баз даних. Є орієнтованою на користувача та легка у освоєнні.

-REST (representational state transfer) - протокол який буде використовуватись у цій роботі. В основному використовує файли типу JSON для передачі даних. Може використовувати HTTP та HTTPS протоколи передачі. Має змогу кешування даних в браузері через що і є найпопулярнішим протоколом для веб застосунків.API дозволяють масштабним системам розростатись та ставати ще більшими. При цьому можливість використовувати будь який патерн побудови, мікросервісний або монолітний.

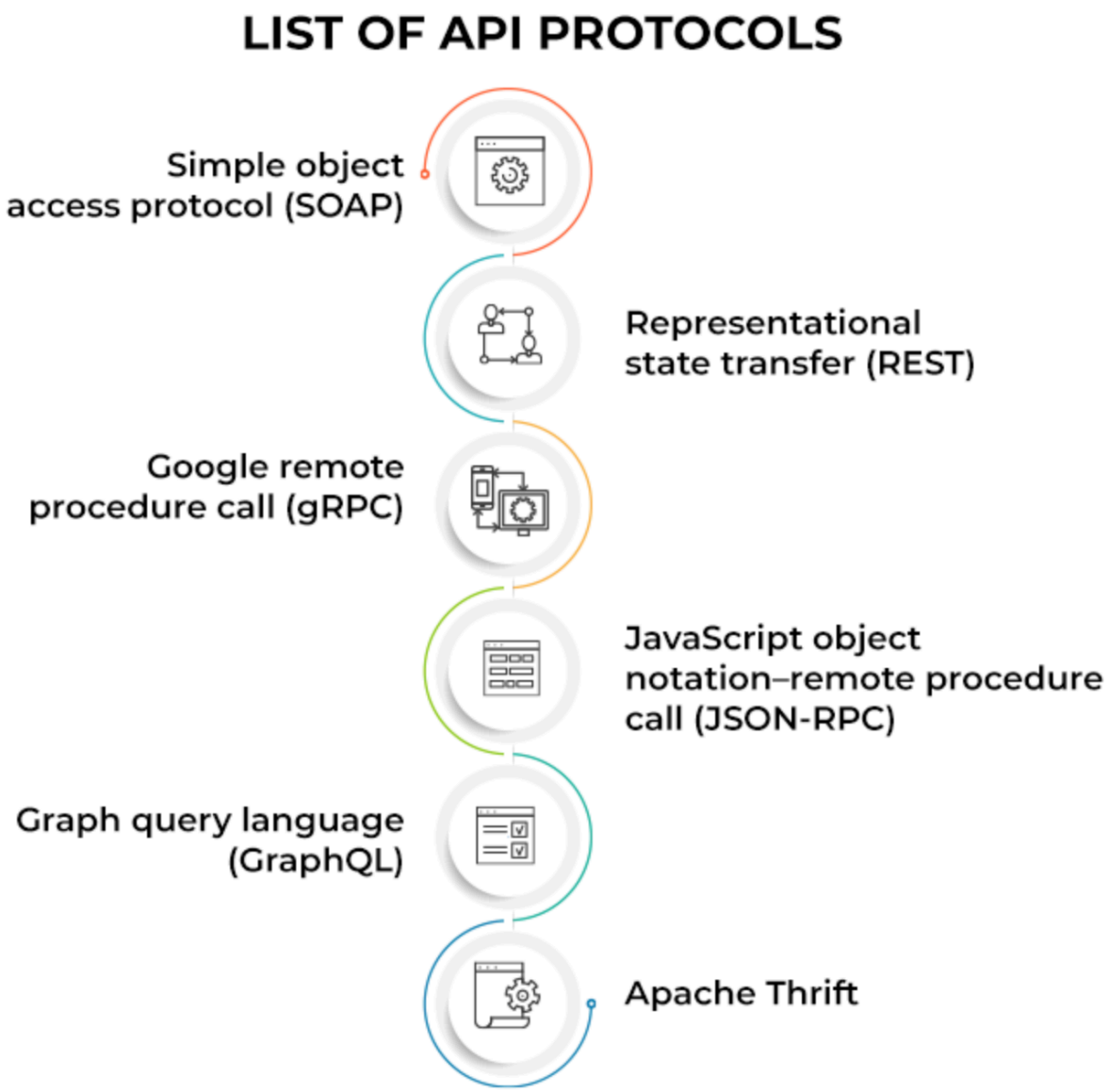

Рис. 2.1 - Перелік протоколів API

Тож навіть мала система яка робить прості обрахунки може використовувати потужності будь якої іншої за допомогою API протоколу, маючи лише доступ до неї.

Існує кілька видів API, такі як: приватні, партнерські, публічні та інші[21]. В системі, яка розробляється в ході виконання цієї роботи буде використано саме публічний API. Це дозволить системі розростатись та залучати сторонніх розробників, якщо вони захочуть теж нею скористатися.

Інтерфейс приймає в систему дані, які називаються запитом, та повертає оброблені дані назад - відповідь.

Ці данні будуть приходити у вигляді об’єктів типу JSON та оброблятися вже самим застосунком.

JSON(JavaScript Object Notation) - важлива частина будь якого проєкту. Текстовий формат обміну повідомленнями між клієнтом та сервером. Він має велику кількість переваг, порівняно з іншими способами передачі інформації по мережі, за що і здобув таку велику і заслужену популярність. Одна з основиних, як і зазначалось раніше, це текстовий формат, тобто такий, що легко може бути прочитаний людиною і може бути зрозумілий контекст без додаткових оброблень на стороні ПЗ. Також типи даних, які може зберігати JSON теж є легко читаємі та зрозумілі - це: числа, null, текст, масив, булеве значення та асоціативний масив(ключ - значення).

Не зважаючи на своє походження з мови JavaScript він має широке застосуваннях і в інших мовах програмування.

На цьому етапі ми визначили яким способом будемо отримувати дані, тепер треба вирішити що саме ми будемо отримувати від клієнта. Оскільки сама система передбачає що нею зможуть користуватись кілька людей, то потрібно щоб був налагоджений процес аутентифікації та авторизації.

2.2 Реєстрація

Процес реєстрації нового користувача з роллю USER буде проходити в окремому вікні. Тут потрібно буде вказати логін та пароль, які в подальшому будуть використовуватись, щоб ідентифікувати користувача в системі, та зберігати певний користувацький досвід в базу даних.

Якщо користувач повинен мати роль ADMIN, то його повинен зареєструвати інший користувач з такою самою роллю. Так само буде запропоновано вказати логін і пароль для подальшого користування системою.

Пароль буде зберігатись в базі у зашифрованому вигляді. Буде використана Bcrypt хеш генератор, який перетворить пароль користувача у зашифрований

набір символів. Його буде неможливо прочитати і зрозуміти людині. Також вкрасти зашифрований пароль теж немає сенсу, адже при вході пароль кожного разу буде хешуватись за допомогою Bcrypt футкції і будуть порівнюватись зашифрований пароль з тим, що є у базі. Тобто якщо просто буде введений зашифрований хеш пароль, то він не буде відповідати тому, що є у базі.

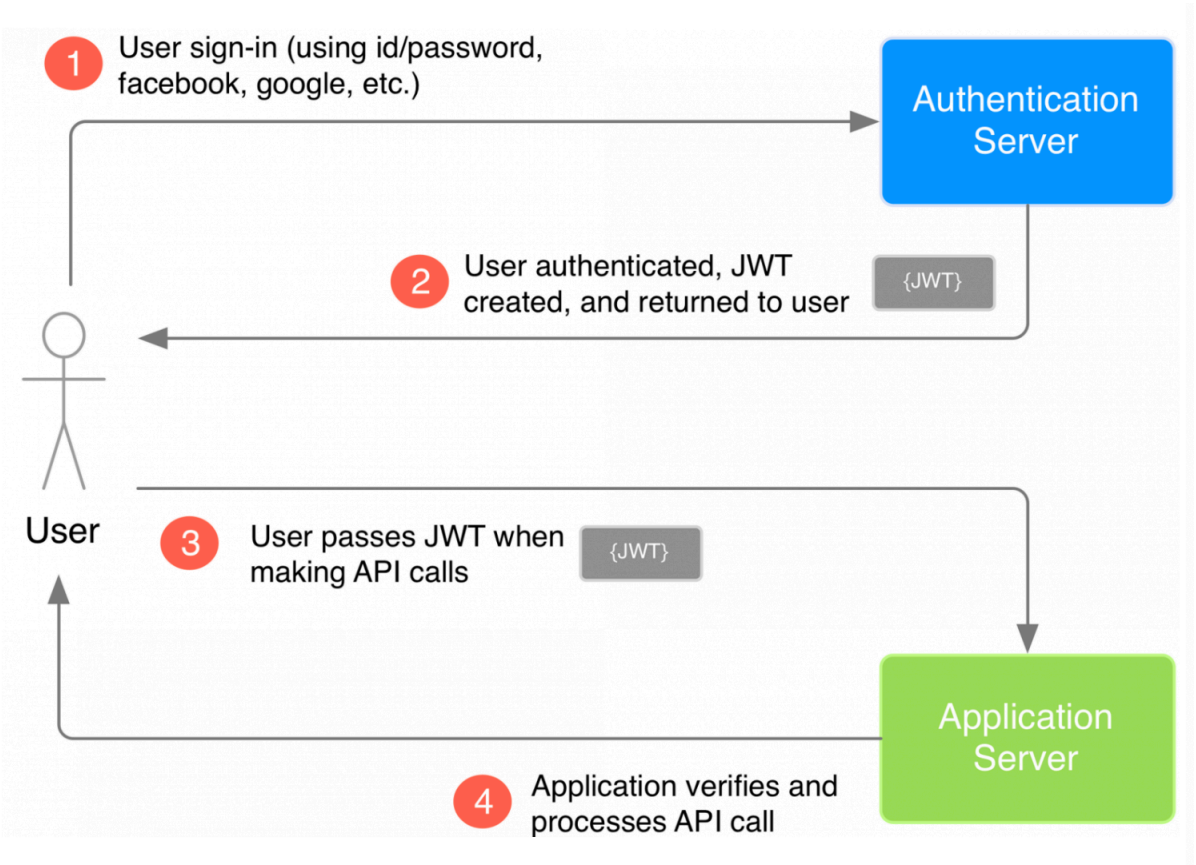

Рис. 2.2 - Порядок створення та використання JWT

Аутентифікація буде проходити за допомогою пари логіна та пароля користувача. Сервер буде перевіряти введені користувачем дані та співставляти їх з даними в базі від вже зареєстрованих користувачів. Якщо логін і пароль співпадають з тими що наявні в базі даних, то користувач є дійсно тим, ким він себе назвав. Після цього для нього почнеться процес авторизації.

Цей процес надання і перевірки прав на вчинення будь яких дій. Наша система передбачає дві ролі для користувачів: адміністратор(admin) та користувач(user). Відповідно адміністратор буде мати набагато більше прав та можливостей ніж звичайний користувач. Це буде реалізовано для того, щоб

мінімізувати вплив простого користувача на роботу системи, надаючі йому всі можливості для повноцінного використання. При цьому зберігаючи можливість коригування роботи самої системи адміністраторами, без змін у коді програми.

Раніше зазначалось, що дані між клієнтом і сервером будуть передаватись у текстовому форматі. Передавати пароль між різними системами у просто відкритому форматі не є безпечним. Тому буде застосовуватись JWT спосіб.

JWT(JSON Web Token) - JSON об’єкт, який вважається одним із найбезпечнішим способом передавати дані між двома учасниками[22]. В собі він містить три складові:

1. Заголовок(header) - загальна інформація по токену.
2. Корисні дані(payload) - дані про ролі користувача, його id, та інші дані.

3.Підпис(signature) - певно основна частина цього токена. Вона відповідає за те, щоб система в яку цей токен приходить змогла визначити його оригінальність, та не підробність.

Система генерує та використовує JWT наступним чином:

1.Спершу користувач заходить на сервер аутентифікації за допомогою пари логін пароль.

2.Після того сервер перевіряє наявність такого користувача, та в позитивному випадку створює та повертає JWT до користувача.

3.Коли користувач виконує запити до сервера, він повинен додати отриманий JWT.

4.Коли на сервер приходить запит від користувача, то сервер перевіряє по отриманому JWT чи можливо повернути якісь дані і які саме дані користувач може отримати. Саме в підписі JWT знаходиться сееретна фраза яка відома тільки на сервері. І за її допомогою стає відомо чи справді користувач є легальним у системі.

Формат і спосіб прийому данних від клієнта відомий. Це буде запит, який повинен містити токен від користувача, з усіма необхідним данними та підписом. JSON об’єкт із вказаними місцями, між якими потрібно буде прокласти маршрут на сервері, а також додатковими параметрами, такими як: кількість зупинок,

максимальний час у дорозі для одного переїзду та денний, чи нічний переїзд. Вже прокладений маршрут буде повернутий назад для користувача теж як JSON об’єкт.

2.3 Вихідні дані

Вихідні дані будуть повертатись на клієнт від сервера також у вигляді JSON об’єкта. В разі якщо користувач знайдений у системі, то сервер поверне прорахований маршрут, який буде відображено на мапі користувача. Це будуть дані у вигляді координат точок. Він буде містити інформацію, яка потрібна для того, щоб відобразити на клієнті. Це будуть такі дані: координати початкової точки, координати кінцевої точки та координати проміжних точок, через який буде прокладено маршрут. Оскільки Варіативність маршруту залежиться від розгалуженості шляхів на місцевості, то і додатковиз точко може бути різна кількість, від однієї до кількох сотень, якщо цього потребує маршрут.

2.4 Структура бази даних

При розробці системи буде використовуватись реліаційна база даних. Реліаційна БД використовує певну структуру, яка дозволяє нам пов’язувати дані між собою. Зазвичай такі дані знаходяться в різних таблицях. В кожній таблиці може міститись безліч полів з даними різних типів. Кожній колонці в таблиці відповідає певний тип даних, який вона може зберігати.

Серед основних переваг реліаційних БД виділяють наступні:

-Типізація. Реліаційні БД дозволяють користувачам типізувати дані, які зберігаються. Такий підхід спрощує сортування та зберігання інформації.

-Проста модель. Оскільки така система не потребує складних процесів чи запитів, вона може бути просто реалізована, та не навантажувати сильно систему.

-Точність даних. Оскільки таблиці пов’язуються за допомогою первинного та зовнішнього ключа, то дані не можуть бути повторюваними і унеможливлює дублювання даних.

-Легкий доступ до даних. Щоб отримати доступ до таблиці потрібно просто до неї звернутись. Не потрібно проходити дерево запитів, щоб отримати якийсь результат. Можна об’єднувати кілька таблиць, фільтрувати та сортувати дані, щоб отримати саме те, що потрібно одним запитом.

-Цілісність даних. Типізовані записи даних та підтвердження легітимності гарантують що всі дані обмежені певними домовленостями та існують усі потрібні дані для створення зв’язків між таблицями.

-Гнучкість. Реліаційна БД має здатність до розширення оскільки має гнучку структуру. Технічно немає жодних обмежень по кількості записів в одній таблиці. Також дозволяє змінювати конфігурацію самої БД без збоїв у роботі системи.

-Безпечність. Оскільки дані розподіляютсья між різними таблицями, то доступ до цих таблиць теж можна регулювати. Це легко реалізуєтсья за допомогою СУБД

Отримувати доступ до бази даних будемо за допомогою мови SQL(Structured Query Language). Це мова запитів до БД, вона дуже схожа до англійської мови, що спрощує її написання, читання, та підтримку.

Система управління яка буде використовуватись - MySql. Популярна СУБД для розробки веб додатків. Відома своєю швидкістю, легкістю в користуванні.

Таблиця «users» буде зберігати основні дані про користувачів. Буде містити логін користувача у вигляді тексту, зашифрований пароль - текст, унікальний ідентифікатор - ціле число, що буде також первинним ключем, дата створення користувача - тип дата.

Атрибути таблиці «user» (рис. 2.3): «id» (primary key, int), «user_name» (varchar), «password» («varchar», bcrypt), «email» («varchar»), «creation_date» («date»).

Field Types

#	Field	Schema	Table	Type	Character Set	Display Size	Precision	Scale
1	id	ukeess_spa_app	user	INT	binary	11	1	0
2	username	ukeess_spa_app	user	VARCHAR	utf8mb4	45	5	0
3	password	ukeess_spa_app	user	VARCHAR	utf8mb4	100	60	0
4	email	ukeess_spa_app	user	VARCHAR	utf8mb4	45	0	0
5	creation_date	ukeess_spa_app	user	DATETIME	binary	19	0	0

Рисунок 2.3 - Таблиця «user»

Таблиця «roles» (рис. 2.4) буде містити дані про ролі в сисемі. Зберігає унікальний ідентифікатор та первинний ключ - ціле число, назва ролі - текстове поле.

Оскільки один користувач може мати кілька ролей, то буде створенна допоміжна таблиця, яка буде містити два зовнішні ключі. Таблиця «user_role» (рис. 2.5) буде містити зовнішній ключ «user_id» - ціле число, та зовнішній ключ «role_id» - ціле число. Застосування проміжної таблиці спростить відносини між таблицями «users» та «roles» і дозволить легко вносити зміни у ці відносини.

#	Field	Schema	Table	Type	Character Set	Display Size	Precision	Scale
1	role_id	ukeess_spa_app	roles	INT	binary	11	0	0
2	role_name	ukeess_spa_app	roles	VARCHAR	utf8mb4	45	0	0

Рисунок 2.4 - Таблиця «roles»

Таблиця «route» (рис. 2.6) буде зберігати маршрути, які створював користувач раніше. Вона буде пов’язана з користувачем за допомогою зовнішнього ключа «user_id» - тип даних ціле число, який буде вказувати на користувача, якому належить цей маршрут. Буде присутнє поле «route_name», яке буде зберігати назву маршруту, який може вказати користувач. Поле «start_position» містить дані про

Field Types

#	Field	Schema	Table	Type	Character Set	Display Size	Precision	Scale
1	role_id	ukeess_spa_app	user_roles	INT	binary	11	0	0
2	user_id	ukeess_spa_app	user_roles	INT	binary	11	0	0

Рисунок 2.5 - Таблиця «user_roles»

початкову точку маршруту. Поле «finish_position» зберігає інформацію про кінцеву точку маршруту.

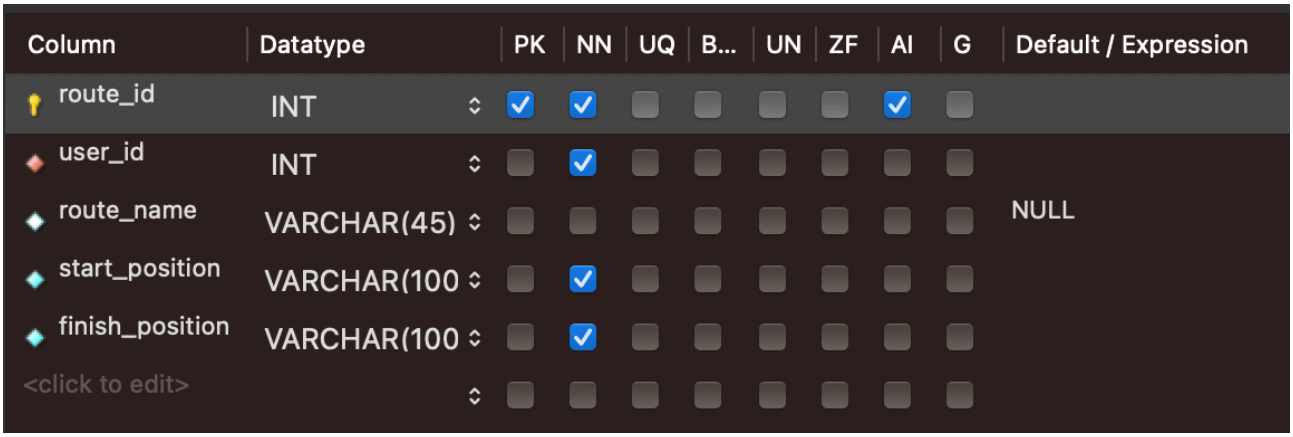

Рисунок 2.6 - Таблиця «route»

В цьому розділі були окреслені характеристики системи та базу даних з якою система буде працювати. Також було описано дані, які система буде приймати на вході та дані які буде повертати до користувача. Також описано процес захисту даних користувача за допомогою шифрування ключів доступу та веб-токенів які збільшують захищеність передачі даних в мережі.

3 Опис програмної реалізації

3.1 Побудова системи

Система пошуку критичного шляху буде використовувати координатну систему встановлення початкової та кінцевої точки. Такий підхід дозволить максимально точно відобразити точки, навіть якщо їх не буде в адресному пошуку Google.

Побудова критичного шляху буде базуватись на кількох методах: метод критичного шляху(CPM - critical path method), та метод , який носить ім’я нідерландсього вченого, метод Дейкстри. Також буде застосовуватись функцію отримання даних з інтернету, якщо це буде потрібно для точнішого позиціонування.

Система будувалася за наступними принципами:

- Швидкодія.
- Можливість оптимізації.
- Можливість масштабування.
- Можливість перенесення на інші платформи.

Для того, щоб перетворювати числові координати в точку на мапі, був створений окремий пакет, що брав вхідну інформацію від користувача та парсив її в дані, які могли вже відображатись і з якими могла працювати програма.

Також програма сканує найблищі 300 точок і обирає ту, яка підходить найкраще для того щоб побудувати оптимальний шлях. Критичний шлях будується з огляду на швидкість та час, який буде затрачено на пересування із точки А в точку Б.

На рисунку 3.1 зображено процес сканування місцевості я прокладання оптимального маршруту між початком і кінцем шляху із врахуванням 300 наблищих точок.

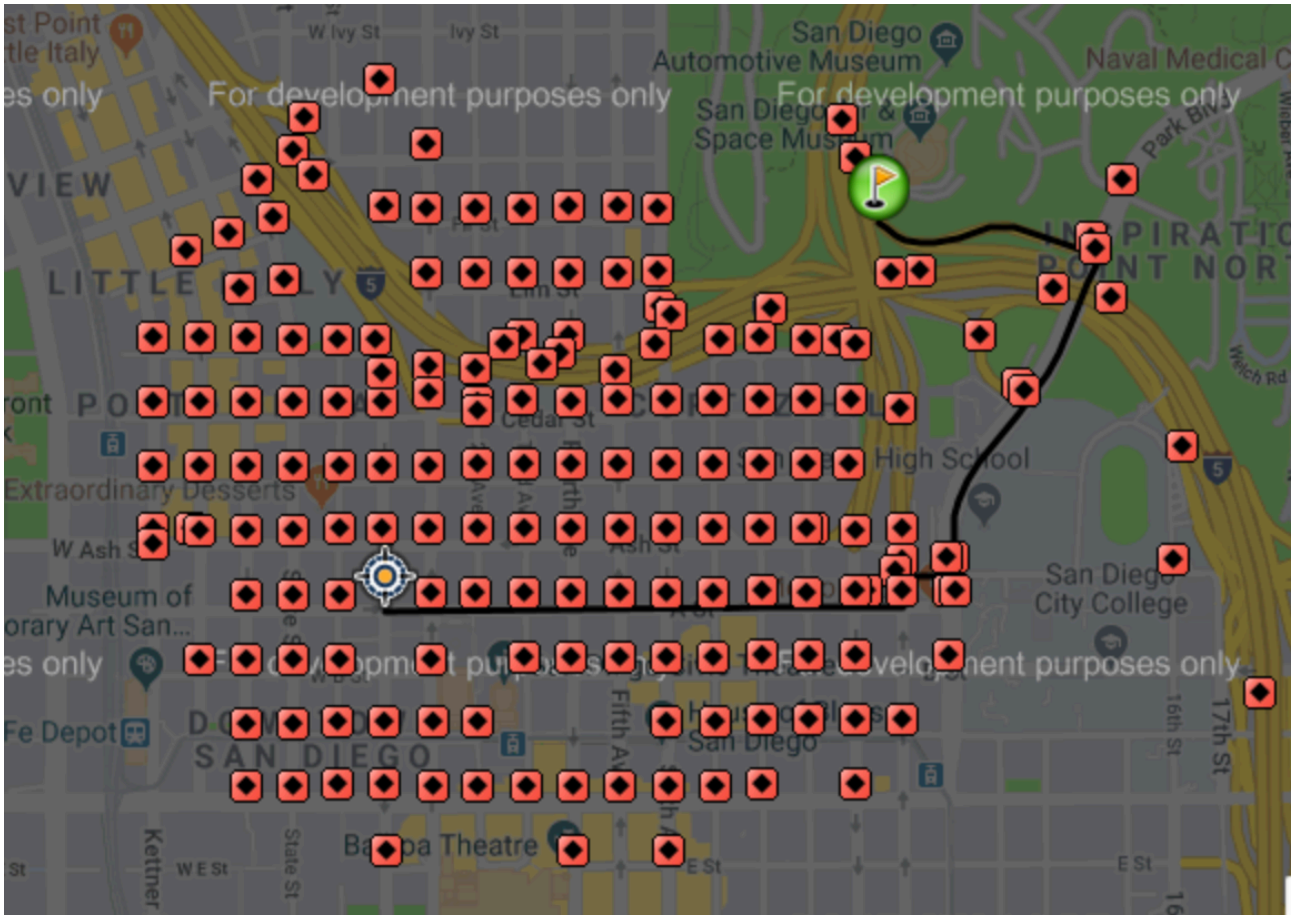

Рисунок 3.1 - побудова маршруту за методом Дейкстри

Мобільність та легкість застосунку була досягнута через використання розробки SPA (single page application) - що дозволяє на одній сторінці застосунку виконувати всі необхідні дії з користувацького боку.

```
Length: 1.1351738276623071 | Speed Limit: 65.0 | Time: 0.017464212733266263
Length: 1.2596596198708847 | Speed Limit: 65.0 | Time: 0.019379378767244382
Length: 1.4441699039116194 | Speed Limit: 25.0 | Time: 0.057766796156464774
Length: 1.3848441159359914 | Speed Limit: 65.0 | Time: 0.021305294091322945
Length: 1.333314816687634 | Speed Limit: 65.0 | Time: 0.020512535641348216
Length: 1.2544363162168959 | Speed Limit: 65.0 | Time: 0.019299020249490706
Length: 4.436404827984578 | Speed Limit: 30.0 | Time: 0.14788016093281928
Length: 1.1795922207789649 | Speed Limit: 65.0 | Time: 0.018147572627368 69
Length: 1.1534031469597443 | Speed Limit: 65.0 | Time: 0.017744663799380 68
Length: 1.2591951663980683 | Speed Limit: 25.0 | Time: 0.050367806655922734
```

Рисунок 3.2 - логування обрахунку часу

Алгоритм також враховує максимально допустиму швидкість руху між двома сусідніми точками у графі для того щоб обрахувати час, який буде затрачено на проходження цієї відстані. Процес підрахунку відбувається на серверній частині і може логуватись. Для наглядності це було виведено в лог системи і зображено на рисунку 3.2.

3.2 Послідовність роботи алгоритму

Алгоритм Декстри який застосовується в системі для пошуку критичного шляху буде дуже ефективним, адже він відрізняється від інших схожих алгоритмів тим, що він може знаходити не найкоротші шляхи, але й найшвидші шляхи, чому може надаватися більша перевага з боку користувача.

Алгоритм базується на 4 основних правилах[23]:

1.Алгоритм в основному починається з вузла, який ви вибрали (вихідний вузол), і він аналізує графік, щоб знайти найкоротший шлях між цим вузлом та всіма іншими вузлами на графіку.

2.Алгоритм відстежує відому найкоротшу відстань від кожного вузла до вихідного вузла та оновлює ці значення, якщо знаходить коротший шлях.

3.Коли алгоритм знаходить найкоротший шлях між вихідним вузлом та іншим вузлом, цей вузол позначається як «відвіданий» і додається до шляху.

4.Процес триває, доки всі вузли в графі не будуть додані до шляху. Пункт 2 і 3 повторюються, таким чином, ми отримаємо шлях, який з’єднує вихідний вузол з усіма іншими вузлами, дотримуючись найкоротшого шляху для досягнення кожного вузла.

Також важливо уточнити що цей алгоритм може працювати лише з графами які мають додатню вагу. Оскільки відстань між двома точками в просторі не може бути від’ємною то цей алгоритм ідеально підходить для нашої системи.

В системі з пошуку критичного шляху для рухомих складів буде збудований граф, де вершинами будуть обиратися точки в найблищому радіусі, а відстань між двома сусідніми точками буде «вагою».

На рисунку 3.3 зоюражено схему роботи алгоритму яка була описана вище.

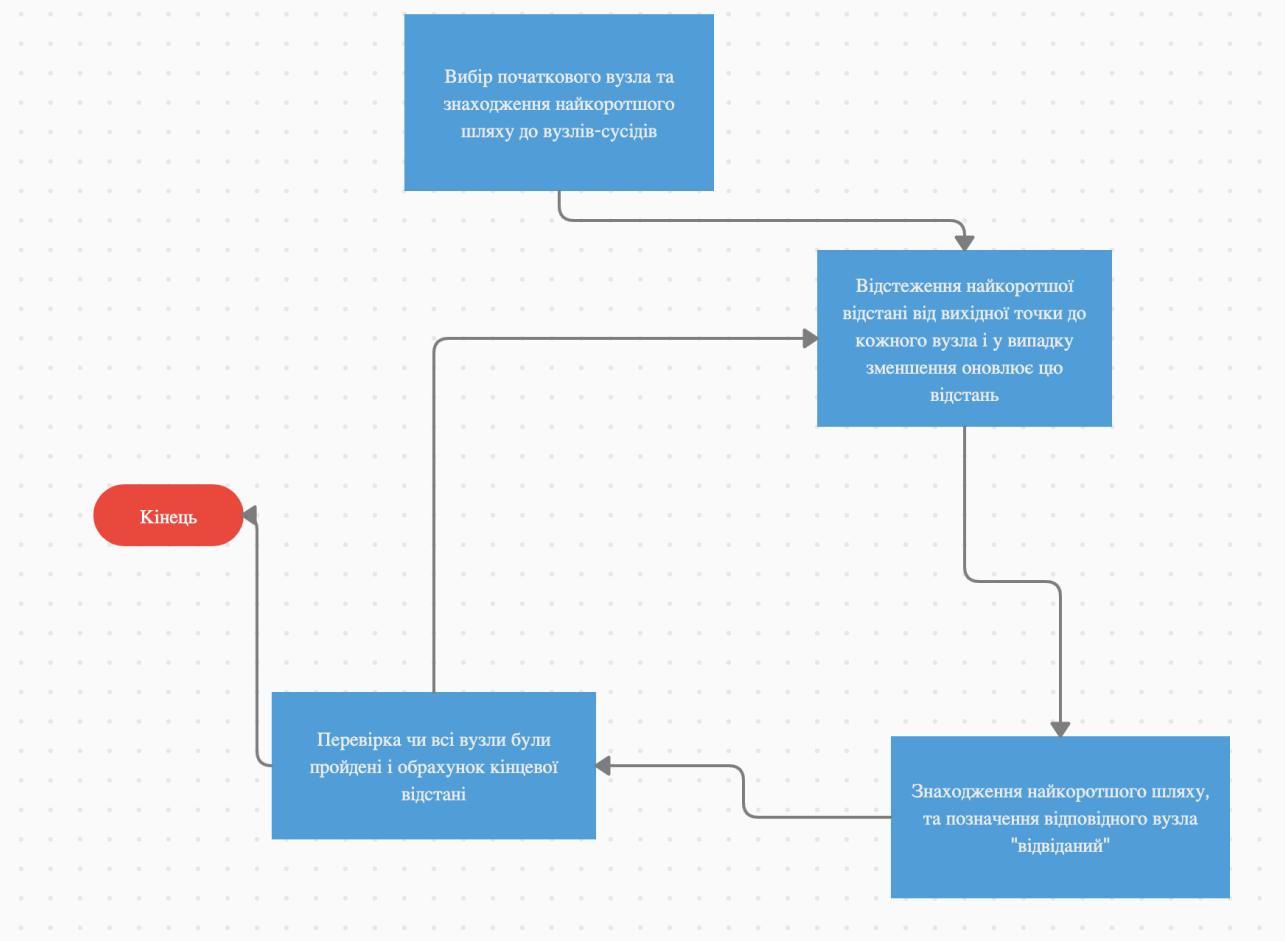

Рисунок 3.3 - схема роботи алгоритму

3.3 Архітектура додатку

Користувач спілкується з системою при допомозі архітектури клієнт - сервер (див. Рисунок 3.4). Це означає що сервер надсилає дані для клієнта тільки отримавши запит від останнього. Оскільки сервер не зберігає жодної інформації про клієнта, то це дозволяє серверу одночасно обробляти дані від багатьох

клієнтів. З іншого боку клієнт повинен бути побудований так, щоб кожного разу надсилати всі данні, які потрібні на сервері для правильної роботи. Це робить систему легко масштабованою та гнучкою для тестування та подальших змін.

Рисунок 3.4 - архітектура клієн-сервер[20]

Основна бізнес-логіка обробляється на стороні сервера. Як було зазначено раніше, сервер не зберігає жодних даних, а лише працює з ними. Для зберігання данних використовуєтсья база даних MySQL. Вона дуже швидко обробляє структуровані дані, тож це не сильно впливає на швидкодію застосунку.

MVC(Model View Controller) - модель дизайну програмного забезпечення (див. Рисунок 3.5), яка складаєтсья з трьох основних компонентів: модель, вигляд та контролер.

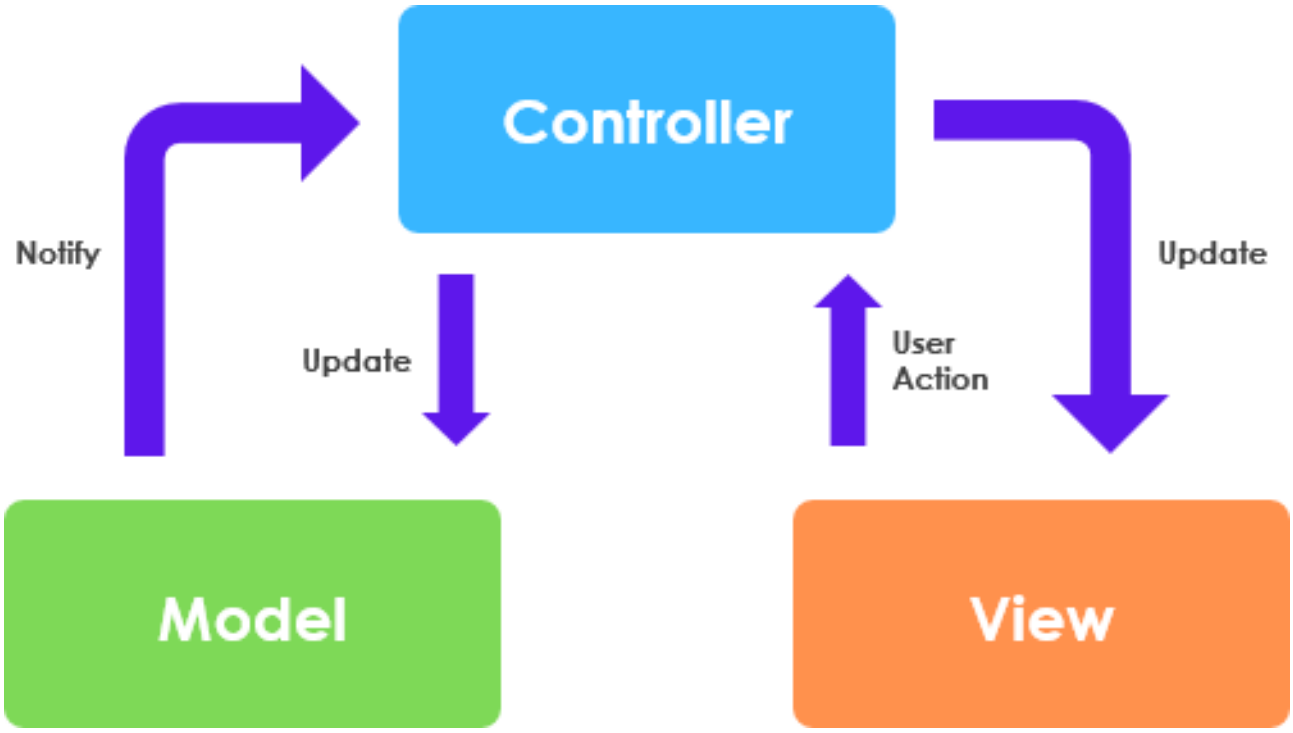

Рисунок 3.5 - MVC архітектура

Модель - частина застосунку, яка відповідає за бізнес-логіку. Вона спілкуєтсья з базою даних, а також проводить усі маніпуляції та обрахунки. Це ядро застосунку.

Контролер - шар застосунку, який дозволяє приймати запити від клієнта, а також повертати відповіді від сервера назад до клієнта у зрозумілому для нього вигляді. Реалізується шляхом HTTP запитів і є загалом дуже використовуваним у розробці подібних застосунків. Використання Spring фреймворку дозволяє значно спростити роботу з цією частиної застосунку, адже Spring вже має реалізацію багатьох інтерфейсів, що можуть і будуть використаними.

Вигляд - частина, що відображає ввесь користувацький інтерфейс, та реалізує запити від кінцевого користувача до серверу. За частину вигляд в застосунку буде відповідати React.js. Фреймворк на основі мови програмування

JavaScript. Оскільки це SPA застосунок, то ця частина теж буде містити частину бізнес-логіки, яка буде безпечною і зможе полегшити роботу для сервера.

Користувач в частині вигляд буде вводити свій запит. Запит буде містити координати точок, а також дані про користувача. Дані користувача потрібні для його ідентифікації та подального збереження в базі даних. Контролер буде перехоплювати цей запит та передавати до моделі. Модель відповідає за те, щоб числові координати обробити, провести розрахунки згідно з алгоритмом Дейкстри і повернути назад до користувача у вигляді точок на карті. Також модель буде вносити до бази даних відомості про його запити, щоб у подальшому можна було їх знову використовувати без нових обрахунків.

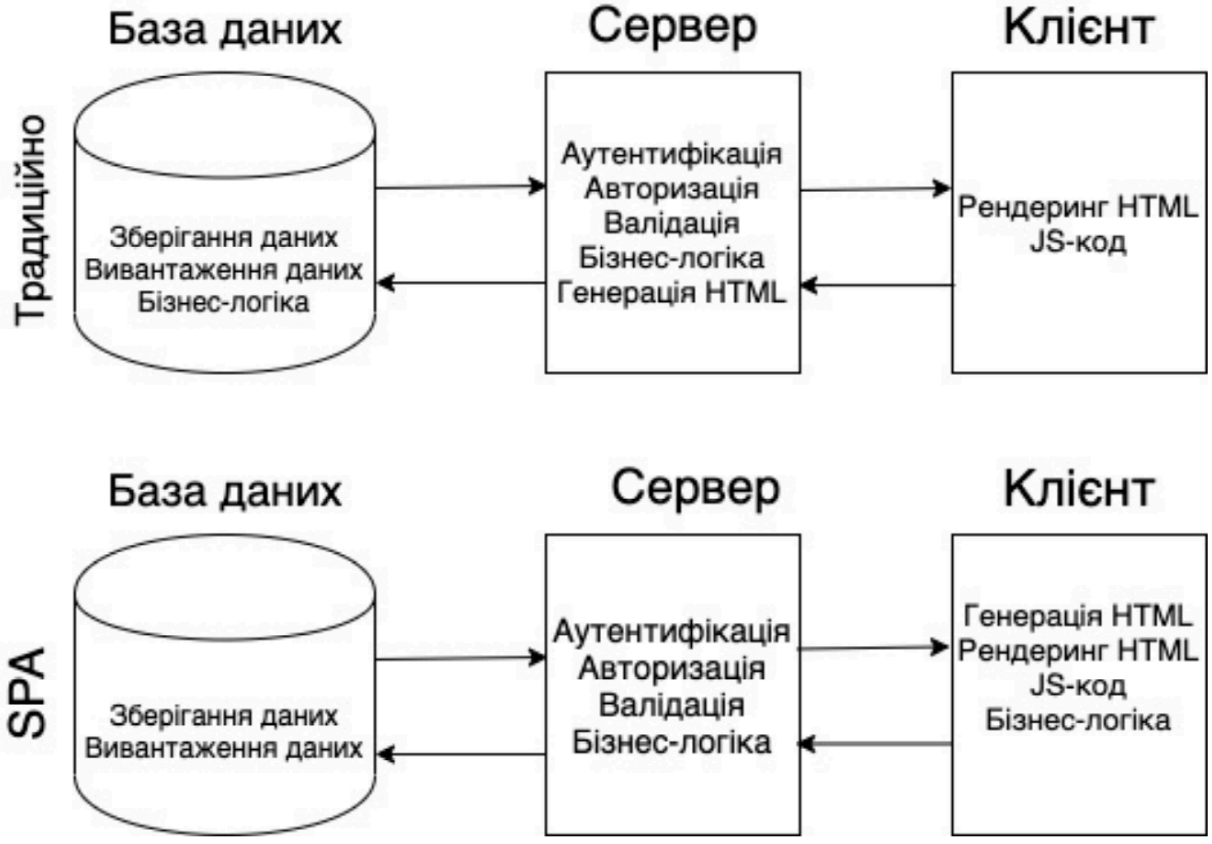

Рисунок 3.6 - Відмінності між односторінковими застосунками та стандартним підходом[22]

Односторінковий додаток працює в браузері і дозволяє оновлювати дані для користувача без перезавантаження самої сторінки браузера. Дуже багато веб-техтологій можуть бути використані при побудові такого застосунку.

На рисунку 3.6 видно різницю між стандартним підходом до побудови веб-застосунків та односторінкомим застосунком. Як видно, генерація HTML та частина бізнес-логіки переходить саме на сторону клієнта у браузері.

Для взаємодії серверного коду з базою даних буде використовуватись ORM система Hibernate. Hibernate входить до складу Spring Framework, хоча являється повністю незалежним фреймворком.

Hibernate — це служба об’єктно-реляційної стійкості та запитів з відкритим кодом для будь-якої програми Java. Hibernate зіставляє класи Java з таблицями бази даних і з типів даних Java на типи даних SQL і звільняє розробника від більшості поширених завдань програмування, пов’язаних із збереженням даних.

Hibernate розміщується між традиційними об’єктами Java і сервером бази даних, щоб виконати всю роботу зі збереження цих об’єктів на основі відповідних механізмів (див. Рисунок 3.7).

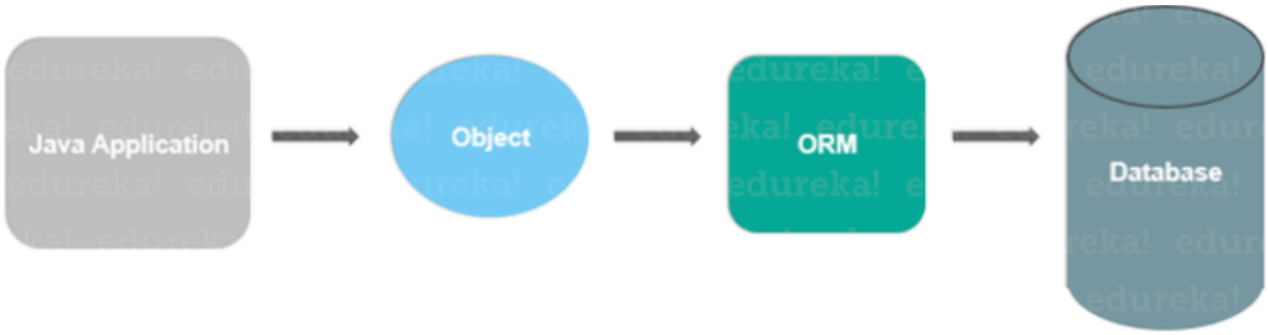

Рисунок 3.7 - схема роботи ORM систем

Hibernate є незалежним від баз даних і може працювати з різними, це залежить від налаштувань фреймворку.

```
spring.datasource.url = jdbc:mysql://localhost:3306/Ukeess_SPA_App?useSSL=false&serverTimezone=UTC
spring.datasource.password=bestuser
spring.datasource.username=bestuser
```

Рисунок 3.8 - вміст файлу, необхідного для зв’язку з абазою даних

Для того щоб поєднати системний код і дані з бази в мові Java є JDBC. Hibernate[24] використовує теж цей інструмент, але значно спрощує роботу для

розробника. Зв’язок може бути налаштованим за допомогою конфігураційного файлу як зображено на рисунку 3.8.

Java це об’єктно-орієнтована мова програмування і реалізація таблиць з бази даних, теж відбуваєтсья за допомогою об’єктів. Для того будуть використовуватись POJO об’єкти. Про них було розказано у попередніх розділах. В коді вони були реалізовані з застосуванням Hibernate. Hibernate дозволяє використовувати анотації, та легко налаштовувати класи.

Лістинг 2.1 - POJO-Клас User для зв’язку з базою даних.

```
@Entity
@Table(name = "user")
@DynamicUpdate
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
     private int id;

    @Column(name = "username")
    private String username;

    @Column(name = "password")
    private String password;

    @Column(name = "email")
    private String email;

    @Column(name = "creation_date")
    private Date creationDate;

    @OneToMany(mappedBy = "user")
    private List<Route> routeList;

    @OneToOne(cascade = CascadeType.REFRESH)
```

```
@JoinTable(name = "user_roles",
            joinColumns =
                    { @JoinColumn(name = "user_id", referencedColumnName =
"id") },
            inverseJoinColumns =
                    { @JoinColumn(name = "role_id", referencedColumnName =
"id") })
    private Roles role;

    public User() {
    }

public User(String username, String password, String email, Date
creationDate) {
    this.username = username;
    this.password = password;
    this.email = email;
    this.creationDate = creationDate;
}

public User(String username, String password, String email, Date
creationDate, List<Route> routeList) {
    this.username = username;
    this.password = password;
    this.email = email;
    this.creationDate = creationDate;
    this.routeList = routeList;
}

public Roles getRole() {
    return role;
}
```

```
public void setRole(Roles role) {
    this.role = role;
}

public List<Route> getRouteList() {
    return routeList;
}

public void setRouteList(List<Route> routeList) {
    this.routeList = routeList;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
```

```
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public Date getCreationDate() {
    return creationDate;
}

public void setCreationDate(Date creationDate) {
    this.creationDate = creationDate;
}
}
```

Клас User зберігає в собі всі поля з бази даних, а також має допоміжні поля які за допомогою анотацій та Hibernate дозволяє реалізувати відносини між таблицями.

Анотація @OneToMany дозволяє зберегти масив шляхів для кожного окремого юзера.

Анотація @JoinTable реалізує застосування допоміжної таблиці для зв’язку між таблицями User та Roles.

Лістинг 2.2 - POJO-Клас Roles для зв’язку з базою даних.

```
@Entity
@Table(name = "roles")
@DynamicUpdate
public class Roles {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "role_id")
```

```
private int id;

@Column(name = "role_name")
private String roleName;

@OneToOne(mappedBy = "role")
private User user;

public Roles() {
}

public Roles(String roleName) {
    this.roleName = roleName;
}

public User getUser() {
    return user;
}

public void setUser(User user) {
    this.user = user;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getRoleName() {
    return roleName;
}
```

```
    public void setRoleName(String roleName) {
        this.roleName = roleName;
    }
}
```

Цей клас містить анотація @OneToOne та дозволяє поєднати дві таблиці Roles та User.

Лістинг 2.3 - POJO-Клас Routes для зв’язку з базою даних.

```
@Entity
@Table(name = "route")
@DynamicUpdate
public class Route {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "route_id")
    private int id;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    @Column(name = "route_name")
    private String routeName;

    @Column(name = "start_position")
    private String startPosition;

    @Column(name = "finish_position")
    private String finishPosition;

    public Route() {
    }

    public Route(String routeName, String startPosition, String
finishPosition) {
```

```
        this.routeName = routeName;
        this.startPosition = startPosition;
        this.finishPosition = finishPosition;
    }

    public Route(User user, String routeName, String startPosition, String
finishPosition) {
        this.user = user;
        this.routeName = routeName;
        this.startPosition = startPosition;
        this.finishPosition = finishPosition;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }

    public String getRouteName() {
        return routeName;
    }

    public void setRouteName(String routeName) {
        this.routeName = routeName;
```

```
    }

    public String getStartPosition() {
        return startPosition;
    }

    public void setStartPosition(String startPosition) {
        this.startPosition = startPosition;
    }

    public String getFinishPosition() {
        return finishPosition;
    }

    public void setFinishPosition(String finishPosition) {
        this.finishPosition = finishPosition;
    }
}
```

3.4 Робота користувача з системою

Операціяна система користувача може бути MacOS, Windows 7 і вище, Linux. Також повинен бути встановлений браузер з підтримкою веб-технологій та JavaScript. Застосунок інсталювати не потрібно, адже система працює в режимі онлайн через браузер.

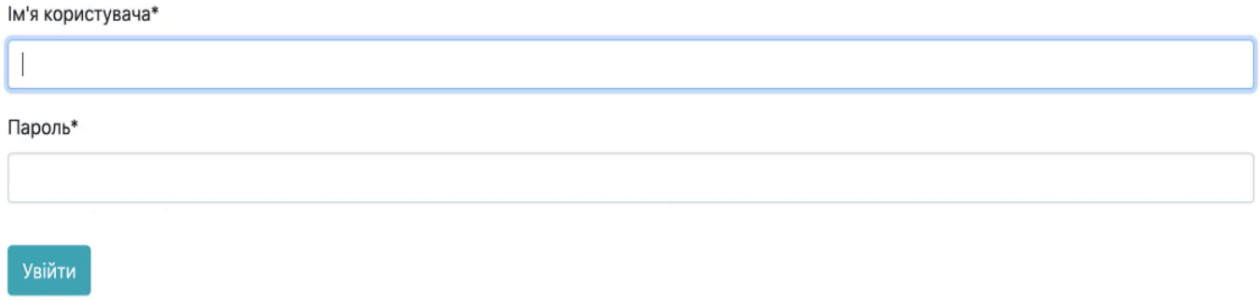

Рисунок 3.9 - Вікно входу в систему

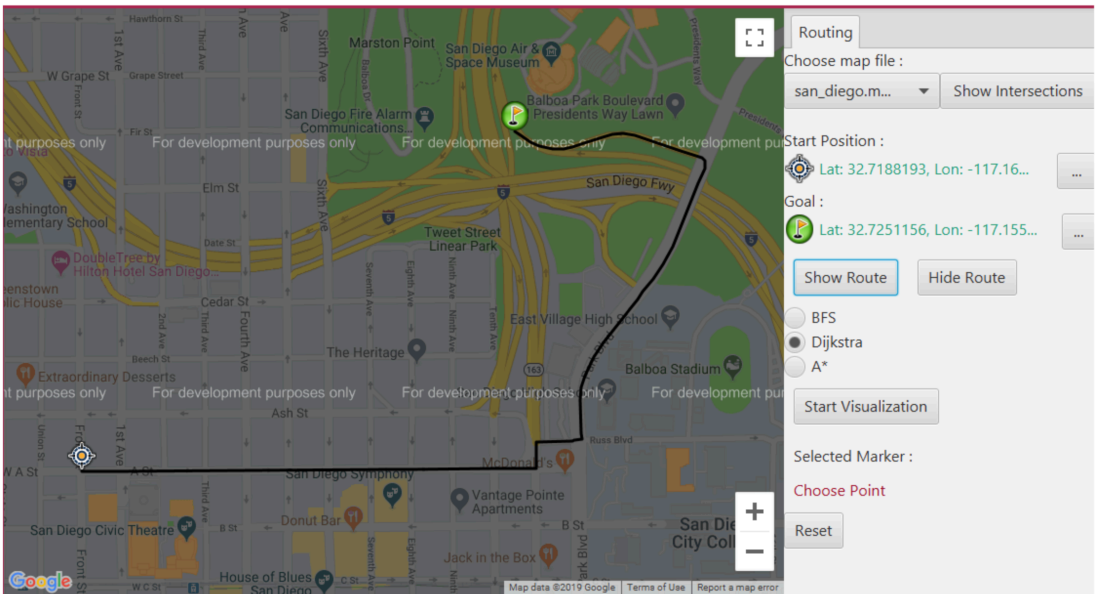

Рисунок 3.10 - Головне вікно роботи користувача з системою

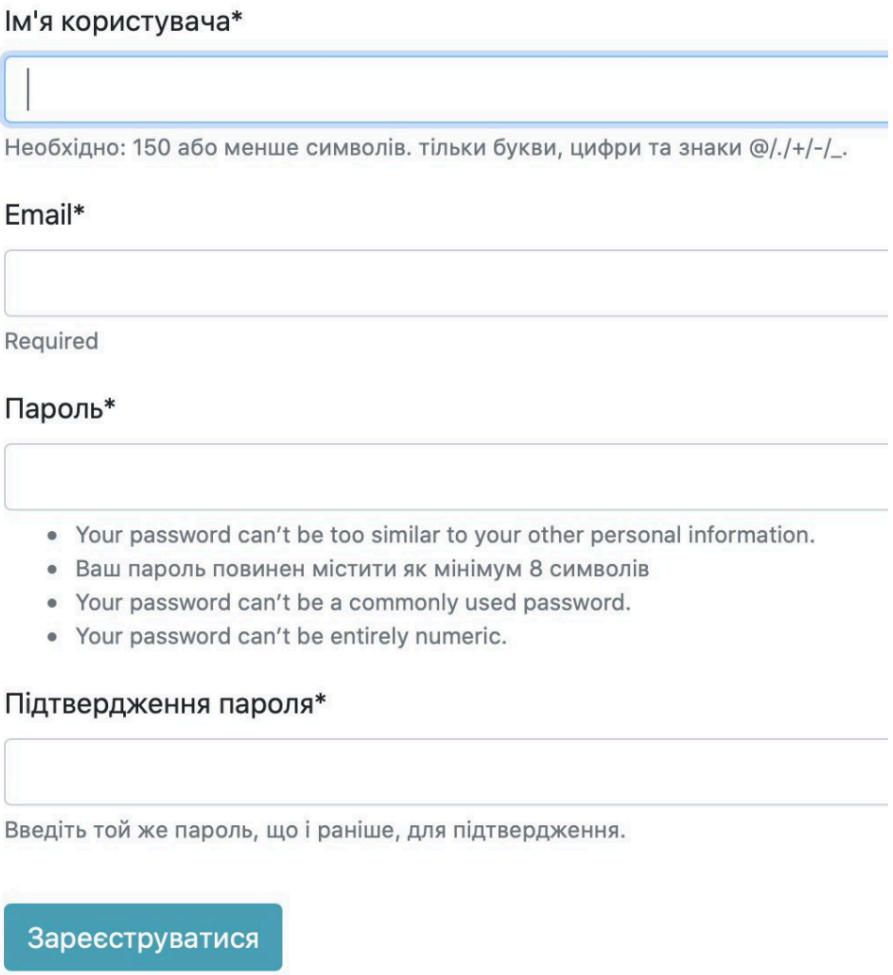

Рисунок 3.11 - реєстрація нового користувача

Вхід в систему для зареєстрованого користувача відбувається за допомогою введення логіна і пароля як показано на рисунку 3.9.

Для нових користувачів передбачена процедура реєстрації в системі. Можна створити новий обліковий запис. Щоб це зробити потрібно ввести ім’я користувача, електронну пошту, пароль, а також підтвердити пароль це вікно зображено на рисунку 3.11.

На рисунку 3.10 можна побачити головне вікно взаємодії користувача з системою. Тут можна поставити початкову та кінцеву точку. Також є можливість обрати алгоритм, яким буде користуватись система під прокладання критичного шляху. Як видно з рисунку, це не є дуже складний етап. Точки на карті переводяться у географічні координати з якими система може легко працювати.

3.5 Тестування програмного забезпеченння

Забезпечення якості програмного забезпечення це дуже важлива складова у процесі розробки. Важливо надати замовнику найкращий продукт, або послугу.

Щоб якість продукту була найвищою потрібно проводити тестування у кілька етапів: функціональне тестування за участі розробників та провести користувацьке тестування при участі потенційних юзерів. Ці етапи зображено на рисунку 3.12.

Рисунок 3.12 - етапи тестування при розробці продукту

Функціональне тестування — це тип тестування, який перевіряє систему на відповідність функціональним вимогам та специфікаціям. Метою функціональних тестів є тестування кожної функції програмного додатку шляхом надання відповідних вхідних даних і перевірки вихідних даних на відповідність функціональним вимогам.

Такий вид тестування перевіряє функції через подачу вхідної інформації та перевірки вихідної. Цей процес буде гарантувати належне виконання вимог, поставлених зі сторони замовника. Відбувається імітація фактичного використання системи але без тестування структури системи.

Для функціонального тестування можуть бути використаний метод[25] Box Testing, який буде використовувати Black Box testing та White Box testing, які зображено на рисунку 3.13

White Box testing дозволяє тестувати внутрішню структуру , дизайн і код. Основний акцент тестування приділяється перевірці вхідних і вихідних даних, поліпшенню внутрішньої структури та зручності у використанні а також посиленню безпеки додатку. Цей етап виконується розробниками на етапі розробки.

White Box testing виконуєтсья для виявлення наступних чинників:

- Проблеми у внутрішній безпеці.
- Не працюючий, або погано структурований код.
- Витік динних.
- Окреме тестування кожної футкції та об’єкта.

Цей етап перевіряє відповідність вихідних данних до очікуваного результату. Якщо вони не збігаються, то тестувальник зіткнувся з помилкою.

Black Box testing виконує тестування без урахування структури коди та деталей реалізації. Виконується окремо від розробки, та може виконуватись людиною, яка ніколи не бачила програмного коду. Цей тип створений для того, щоб протестувати основні вимоги та специфікації програмної системи. Black Box testing дає змогу тестувати систему, працюючи лише з інформацією на вході і виході, не знаючи внутрішньої реалізації.

Зазвичай функціональне тестування поділяється на наступні етапи:

- Визначення функцій, які будуть виконані системою.
- Створення вхідних даних для тестування на основі характеристик функції.
- Виконання тестів.
- Порівняння очікуваних та отриманих результатів.

Тетсування буде найбільш ефективним, якщо вимоги тестування будуються виходячи з запитів користувача або бізнесу.

Бета-тестування, або користувацьке тестування - це тестування проведено кінцевими користувачами для того, щоб визначити чи може продукт піти в реалізацію чи потребує доповнень. Це кінцевий етап тестування який виконуєтсья після функціонального тестування, безпосередньо перед релізом продукту. Метою являється перевірка відповідності програмного продукту до вимог поставленим зі сторони бізнесу.

Число етапів, які будуть проводитись під час бета-тестування може мінятись і залежить від того на скільки детально потрібно тестувати систему, а також від того як багато часу на коштів виділено на цей етап. Зазвичай включені наступні етапи:

- Етап планування, де окреслюються основні вимоги від бізнесу.
- Створення тестових сценаріїв, які будут охоплювати якомога більшу кількість можливих варіантів використання системи.
- Обирається група для тестування з кількох осіб, або відкриваєтсья тест для багатьох шляхом розміщення в інтернеті безкоштовної пробної версії продукту.
- Етап внесення кінцевих змін на основі отриманих результатів тестування.

При розробці системи координації та визначення критичного шляху переміщення рухомого складу теж було проведено тестування.

Для проведення функціонального тестування була використана додаткова бібліотека JUnit5, яка дозволяє розробляти користувацькі тести та покривати ними свою систему. Цими тестами було покрито основну частину системи.

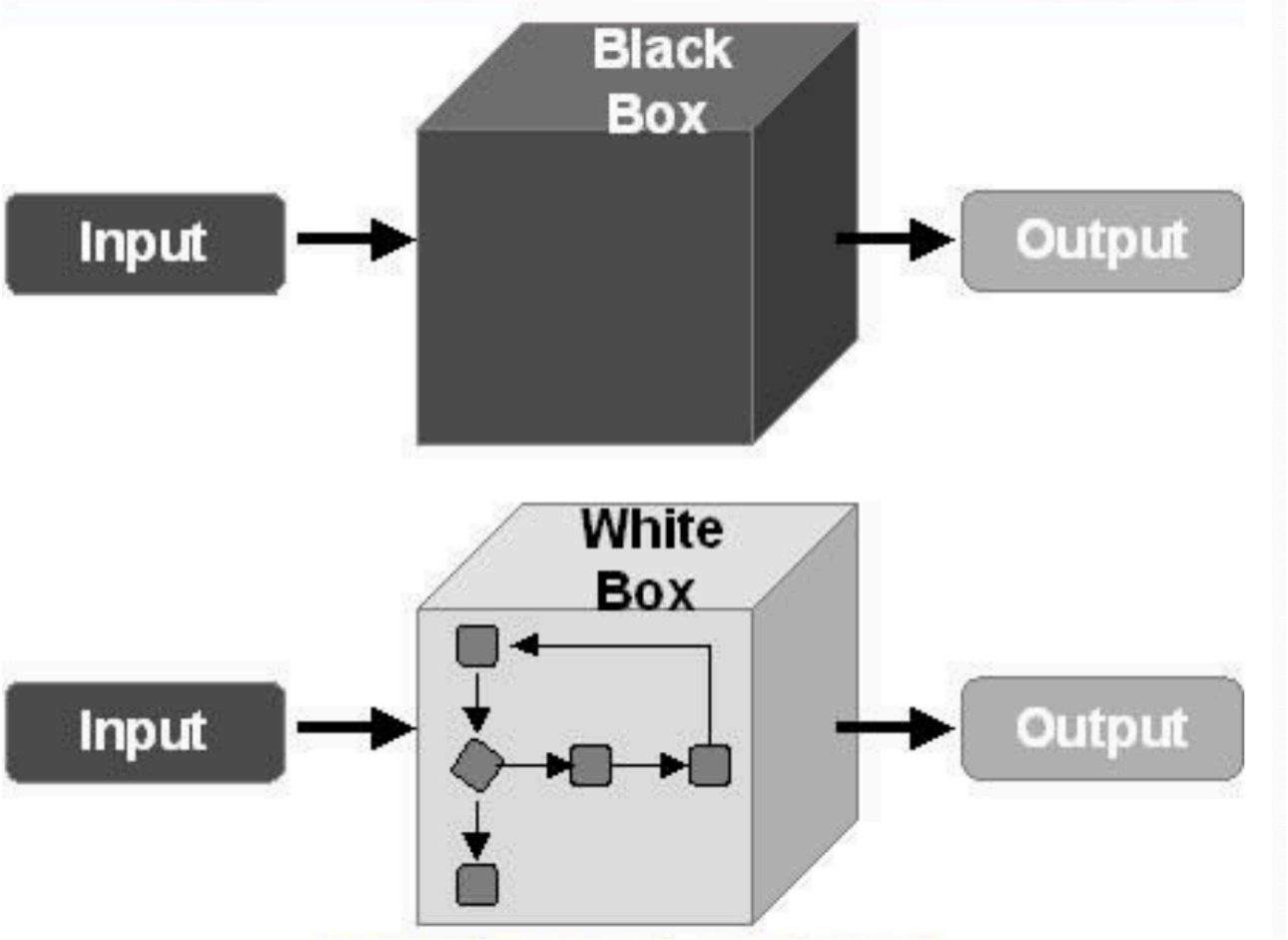

Рисунок 3.13 - White Box та Black Box методи тестування

Також було передбачено вибірку з кількох точок на карті, які будуть передаватись у систему для обрахунку. На рисунку 3.14 зображено пари координат точок, між якими потрібно прокласти маршрут.

Після того як система опрацює вхідні дані, можна спостерігати вихідні дані, які будуть залоговані системою як це зображено на ристунку 3.15.

Оскільки ці дані співпадають з даними з інших ресурсів, які можуть обраховувати відстань, то можна зробити висновки що система коректно обраховує відстані.

Також було перевірено коректність додавання користувачів у базу даних, та можливість входу під своїм логіном і паролем.

Для проведеня користувацького тестування було розроблено кілька кейсів для перевірки якості користувацького інтерфейсу та перевірити чи всі функціх працюють задовільно.

Приклади користувацьких тест-кейсів:

```
40.745624 -73.9726034 40.7456612 -73.9724536 "" motorway
40.7571333 -73.9970778 40.7571134 -73.9968704 "" motorway_link
40.7571134 -73.9968704 40.7570959 -73.9967529 "" motorway_link
40.7570959 -73.9967529 40.757071 -73.9966216 "" motorway_link
40.757071 -73.9966216 40.7570489 -73.9965335 "" motorway_link
40.7570489 -73.9965335 40.7570255 -73.9964434 "" motorway_link
40.7570255 -73.9964434 40.7569929 -73.9963454 "" motorway_link
40.7569929 -73.9963454 40.7568953 -73.9960804 "" motorway_link
40.7568953 -73.9960804 40.7568281 -73.9959146 "" motorway_link
40.7568281 -73.9959146 40.7568078 -73.9958422 "" motorway_link
```

Рисунок 3.14 - координати початвої та фінальної точки для тестування

- Перевірка правильності входу та реєстрації в системі.
- Перевірка правильності роботу карти.
- Перевірка правильносіт вводу даних.

Також було проведено кілька тестів на валідність введених даних, щоб унеможливити в подальшому введення неіснуючих, або таких даних, з якими система не зможе коректно працювати.

Користуваць ке та функціональне тестування було проведено успішно на всіх своїх етапах. Помилки було відстежено та усуното, щоб система працювала коректно.

```
Length: 1.1351738276623071 | Speed Limit: 65.0 | Time: 0.017464212733266263
Length: 1.2596596198708847 | Speed Limit: 65.0 | Time: 0.019379378767244382
Length: 1.4441699039116194 | Speed Limit: 25.0 | Time: 0.057766796156464774
Length: 1.3848441159359914 | Speed Limit: 65.0 | Time: 0.021305294091322945
Length: 1.333314816687634 | Speed Limit: 65.0 | Time: 0.020512535641348216
Length: 1.2544363162168959 | Speed Limit: 65.0 | Time: 0.019299020249490706
Length: 4.436404827984578 | Speed Limit: 30.0 | Time: 0.14788016093281928
Length: 1.1795922207789649 | Speed Limit: 65.0 | Time: 0.01814757262736869
Length: 1.1534031469597443 | Speed Limit: 65.0 | Time: 0.01774466379938068
Length: 1.2591951663980683 | Speed Limit: 25.0 | Time: 0.050367806655922734
```

Рисунок 3.15 - Логування системи

У цьому розділі було описано програмну реалізацію системи пошуку критичного шляху для рухомого складу. Показано структуру системи та архітектуру додатку. Також описано роботу алгоритму.

Також було розглянуто методи Black Box testing та White Box testing та наведено ключові відмінносіт між ними. Також була проведена перевірка якості програмного продукту за стандартами тестування. Цей етам підтвердив, що система координації та визначення критичного шляху переміщення рухомого складу працює коректно та без збоїв на поточному етапі.

4 Охорона праці та безпека життєдіяльності

4.1 Охорона праці.

Враховуючи те, що більшість часу при використанні системи координації та визначення критичного шляху переміщення рухомого складу користувачі проводять за комп'ютером, необхідно дотримуватись правил охорони праці при роботі з персональним комп'ютером.

Питання охорони праці регулюються певними законодавчими та нормативно-правовими актами, які, зокрема, визначають обов'язки роботодавця із забезпечення робітникам комфортних та безпечних умов для здійснення роботи. Ці обов'язки, а також права робітників на таких умовах праці передбачені частиною 2 ст.2 і частиною 1 ст.21 КЗпП, а також ст.13 Закону України «Про охорону праці», у яких визначаються основні положення з реалізації конституційного права робітників.

В залежності від розміру компанії та кількості працівників які задіяні в розробці системи знаходження критичного шляху служба охорони праці може мати різні розміри та форми.

Підпорядковується служба охорони праці згідно із законодавством безпосередньо роботодавцеві.

Проте роботодавець може доручити функціональне управління (кураторство) діяльністю служби іншій посадовій особі, скажімо, головному інженерові, заступникові директора з охорони праці тощо.

Функції служби охорони праці:

1. Підготовка проектів наказів (розпоряджень) з питань охорони праці і внесення їх на розгляд роботодавцю. Проведення спільно з представниками інших структурних підрозділів і за участю представників професійної спілки підприємства.

2. Проведення з працівниками вступного інструктажу з питань охорони праці та супутніх інструктажів.

3. Ведення обліку та проведення аналізу причин виробничого травматизму, професійних захворювань, аварій на виробництві, заподіяної ними шкоди.

4. Забезпечення належного оформлення і зберігання документації з питань охорони праці, а також своєчасної передачі її до архіву для тривалого зберігання згідно з установленим порядком.

5. Складання звітності з охорони праці за встановленими формами.

6. Складання за участю керівників підрозділів підприємства переліків професій, посад і видів робіт, на які повинні бути розроблені інструкції з охорони праці, що діють в межах підприємства, надання методичної допомоги під час їх розроблення.

7. Інформування працівників про основні вимоги законів, інших нормативно-правових актів та актів з охорони праці, що діють в межах підприємства.

8. Розгляд питань про підтвердження наявності небезпечної виробничої ситуації, що стала причиною відмови працівника від виконання дорученої роботи відповідно до законодавства (у разі необхідності).

9. Організація - забезпечення підрозділів нормативно-правовими актами з охорони праці та актами з охорони праці, що діють в межах підприємства, посібниками, навчальними матеріалами з цих питань.

Конструкція робочого (згідно ДСТУ 7234:2011) місця повинна забезпечувати можливість зручного регулювання його розмірів та експлуатаційних параметрів з урахуванням специфіки виконуваної роботи та відповідно до антропометричних характеристик працівника.

Елементи робочого місця за характером формоутворення мають бути зручними та відповідати специфіці роботи людини, а також композиційно організованими в робочому просторі. Кольорографічне та кольорофактурне вирішення робочого місця і його елементів потрібно розробляти з урахуванням психофізіологічного стану оператора під час виконання робіт, та вони мають бути кольористично згармонізованими між собою.

Робоче місце повинно забезпечувати можливість зручного виконання робот у положенні сидячи, стоячи або і сидячи, і стоячи.

Відносно вікон робоче місце необхідно організовувати так, щоб природне світло було з лівого боку (п. 4.3 ДСанПіН 3.3.2.007-98). Робоче місце необхідно розміщувати таким чином, щоб уникнути попадання прямого світла в очі. Штучне освітлення приміщення має здійснюватись системою загального рівномірного освітлення (п. 3.2.2 ДСанПіН 3.3.2.007-98). У приміщеннях при переважній роботі з документами допускається використання системи комбінованого освітлення, тобто встановлення світильників місцевого освітлення додатково до загального.

Як джерела штучного освітлення необхідно використовувати люмінесцентні лампи. Згідно з п. 3.2.5 ДСанПіН 3.3.2.007-98 система загального освітлення має бути у вигляді суцільних або переривчатих ліній світильників, що розташовані збоку від робочих місць (зазвичай ліворуч) паралельно лінії зору працівників.

Шум несприятливий для людини, особливо при тривалому впливі. В оператора це виражається в зниженні працездатності (наприклад, швидкість обробки тексту зменшується на 10-15%), у прискоренні розвитку зорового стомлення, зміну відчуття кольору, підвищенні витрати енергії (на 17%). Тривалий та інтенсивний шум значно знижує продуктивність праці і призводить до зростання кількості помилок у роботі. У відділі головного економіста шум може створюватися телефонними дзвінками та розмовами, системними блоками та клавіатурою ПЕОМ. Так само джерелами шуму можуть бути системи кондиціювання та вентилювання повітря, існують і зовнішні джерела шуму

4.2 Підвищення стійкості роботи підприємств логістики у воєнний час

Реалії війни накладають обмеження на всі сфери життя, але на доставляння та сполучення – у першу чергу. Небезпека використання транспорту, ризик втратити товар під час складського зберігання, необхідність перебудувати маршрути безпосередньо під час руху – це тільки мала частина того, з чим

стикається логістика під час війни. Залишається лише два шляхи – оптимізувати процеси та адаптуватися, або визнати неможливість цього для компанії.

Оскільки в Україні зараз введено воєнний стан, то аналізувати підвищення стійкості роботи логістичних підприємств логістики буду на основі українських підприємств.

Не можна сказати, що виклики для галузі постали лише під час війни. Але зараз рішення треба приймати швидко, ефективно, а ситуація змінюється із кожною хвилиною. З якими проблемами стикається логістика у мирний час:

- управління запасами – що їх більше, то вищий об'єм «заморожених» фінансових ресурсів;

- закупівлі та постачання на склад – типові конфлікти, пов'язані з асортиментом, умовами доставляння, вибором постачальника;

-базисні умови постачання – ризики, витрати, обмін документацією та інші питання між сторонами договору.

Логістика – динамічна галузь, яка для ефективної роботи потребує досвідченого менеджменту. Додайте до перелічених проблем умови, які накладає воєнний стан:

-Відмова від зберігання і накопичення. Зараз немає можливості довго тримати товар на складі – у разі атаки він буде втрачений.

-Зміна складських умов. Зазвичай розгортання хабу для зберігання займає приблизно 3 місяці, наразі ж треба бути готовими організувати безпечний та інтегрований склад за 7–14 днів.

-Ускладнення логістичних операцій. Блокпости, огляди, непрозоре регулювання проїзду під час комендантської години.

-Різкі зміни маршрутів. Необхідність враховувати нові атаки, планувати запасні маршрути заздалегідь.

Перша та найголовніша зміна, що сталася в українській логістиці, – це усунення «центру тяжіння». Як відомо, найбільший складський хаб в Україні – 70–80% усіх професійних складських площ – був у Київській області.

З початком війни великі компанії, а за ними середні та дрібні, були змушені перевезти свої складські залишки та товари на захід України. Відбувся

колосальний відтік до Львівської, Тернопільської, Івано-Франківської областей, Закарпаття, де такого обсягу складських площ не існувало в принципі. Приміщень на всіх не вистачало, а ті, які компаніям вдавалося зайняти, на порядок поступалися колишнім площею, організацією простору та рівнем обслуговування.

Таким чином, бізнес був змушений змінити складський ланцюжок і тим самим збільшити складність та вартість цих операцій. Можна назвати три основні фактори, що вплинули на бізнес-процеси, пов'язані з логістикою.

-Відмова від накопичення та зберігання товарів.

Якщо раніше товар міг довгий час перебувати на складах, звідки йшло відвантаження, то зараз бізнес почав відвантажувати «з коліс», намагаючись мінімально накопичувати залишки, щоб у разі можливої атаки на склади не було втрат товару.

-Різка та швидка зміна складських умов.

Зазвичай запуск складу забирає приблизно три місяці: переїзд складу, розгортання IT-системи та IT-інтеграція, налаштування систем безпеки, відеоспостереження і т. д. Зрозуміло, що стільки часу в компаній не було, вони були змушені мігрувати за лічені тижні, а то й дні на невідомі площі. Десь «кульгала» безпека, десь – операційні процеси. Логістиці, особливо складській, було, м'яко кажучи, складно, але при цьому компанії впоралися досить добре. Включився режим виживання, запуску з нуля, закривалися базові потреби, такі як пошуки складів, водіїв, складського персоналу.

-Ускладнення логістичних операцій.

З цим була велика проблема, особливо спочатку. Це величезна кількість блокпостів та оглядів. Це відсутність чітких правил пересування під час комендантської години: яким логістам можна було їхати вночі, а яким не можна. Це ті чи інші дії з боку територій, які не завжди адекватно реагували на нічні пересування транспорту, що доставляє продукти.

У принципі, логістика як така в мирний та воєнний час не дуже відрізняється. Якщо говорити про звичайні продовольчі товари, то і зараз все йде своєю чергою, все доставляється. Головне – це забезпечення безпеки зон та точок відвантаження, пунктів доставки, транспорту.

Зміни відбулися, але локально чи в рамках окремих бізнес-процесів. Була низка проблем з імпортом, які вже вирішені. Є затримки доставки товарів в окремі райони через окупацію або високі ризики. Зараз нульова авіаційна та воднотранспортна логістика. Є певні складності в'їзду та виїзду залізницею через відсутність залізничних вагонів для експорту з України товарів аграрної групи та відсутність необхідного обсягу залізничних вагонів по імпорту палива – це питання активно вирішується останніми тижнями.

Усе це, звичайно, впливає на зростання ціни на товар для кінцевого споживача. Сьогодні середній товар подорожчав на 1–3%. Чому так? Тому що в середньому в структурі товару логістика складає орієнтовно 12–15% (це світові усереднені цифри). Але треба зазначити, що це збільшення не є таким суттєвим, як, наприклад, при інфляції, яка набагато більше впливає на зростання вартості товару, ніж складнощі з логістикою.

Корпоративні потреби в складах сьогодні досить великі. Ще до війни у Київському регіоні був попит у межах 400 000 кв. м. Зараз попит не такий високий, і це пов'язано здебільшого з великим відтоком логістичних процесів на захід України та загальним зниженням товаропотоку.

Центр тяжіння при побудові нових складів сильно зміниться. Якщо раніше левова частка всіх складів створювалася в Київській області, то зараз компанії схильні зміщувати фокус на південну частину Київської області та південний захід України, вважаючи їх більш безпечною зоною.

При створенні нових складських приміщень ми маємо чітко розуміти ризик-профіль країни, розуміти загрози.

Також важливо мати склади держрезерву, як це організовано у великих європейських країнах: Німеччині, Чехії, Польщі. Велика кількість гуманітарної допомоги, яку ми зараз отримуємо з Європи, – саме з держрезерву. На жаль, у нас інфраструктура складів держрезерву була, м'яко кажучи, не дуже підготовлена.

Ситуація, що сталася в Україні, є показовою для всього світу. Очевидно, що ми не були готові до такого розвитку подій. І зрозуміло чому: жодній розсудливій людині не могло спасти на думку, що в XXI столітті в центрі Європи може статися така трагедія. Найважливіше, що можна з неї винести щодо логістики, – це те, що країна повинна вміти керувати своїми логістичними потоками в екстрених ситуаціях. Має бути ризик-менеджмент, який допоможе великою мірою покривати логістику.

ВИСНОВКИ

Розвиток інформаційних технологій станом на сьогодні дає можливість удосконалювати вже наявні системи та принципи. Також дозволяє приносити нові інструменти та технології в щоденне користування.

Мета роботи полягає в представленні програмної реалізації системи координації та визначення критичного шляху переміщення рухомого складу.

Робота створює рішення для стпрощення багатьох логістичних операцій. Може бути використана як в межах великих корпорацій для побудови голістичних маршрутів, так і окремими користувачами для задоволення власних потреб.

В роботі використовуються різні мови програмування, що дозволяє будувати застосунок з міцною бізнес логікою на стороні серверу, та зробити зручний і зрозумілий інтерфейс для користувачів. Поєднання нових фреймворків в галузі веб-програмування дозволяє розробити систему максимально швидко і залишити її відкритою для розширення в майбутньому.

Описано основні етапи взаємодії між сервером і клієнтом та між сервером та базою даних. Реалізовано класи для організації даних в базі даних та для взаємодії з клієнтом. Обрано базу даних, яка буде максимаьно швидко працювати з поставленими завданнями та дозволить легко масштабувати систему в майбутньому.

Описано основні технології для розробки веб-застосунків за допомогою фреймворку ReactJS. Описано його можливості та основні його переваги.

Також описано роботу з фреймворком ORM Hibernate, та основні переваги його вибору.

Система відкрита для розширення та модернізації, а також готова працювати з різними клієнтами через публічні. API.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. [Електронний ресурс] – Режим доступу: URL: https://www.toppr.com/guides/business-management-and-entrepreneurship/direction-and-coordination/concept-and-features-of-coordination/

2. [Електронний ресурс] – Режим доступу: URL: https://www.trans.eu/ua/blog/lohistyka-4-0/3-kroky-do-ocyfruvannia-biznesu/

3. М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – “Інженерія програмного забезпечення” для усіх форм навчання [Текст] – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя – 2020 – 27 с.\

4. [Електронний ресурс] – Режим доступу: URL: https://dev.java/learn/getting-started-with-java/4.

5. [Електронний ресурс] – Режим доступу: URL: https://data-flair.training/blogs/advantages-disadvantages-javascript/

6. [Електронний ресурс] – Режим доступу: URL: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html

7. [Електронний ресурс] – Режим доступу: URL: https://www.javatpoint.com/pojo-in-java

8. [Електронний ресурс] – Режим доступу: URL: https://www.baeldung.com/spring-bean

9. [Електронний ресурс] – Режим доступу: URL: https://www.educative.io/answers/what-is-dependency-injection-in-java

10. [Електронний ресурс] – Режим доступу: URL: https://www.vogella.com/tutorials/ApacheMaven/article.html

11. [Електронний ресурс] – Режим доступу: URL: https://www.geeksforgeeks.org/maven-lifecycle-and-basic-maven-commands/

12. [Електронний ресурс] – Режим доступу: URL: https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs

13. [Електронний ресурс] – Режим доступу: URL: https://uk.reactjs.org/docs/react-dom.html

14. [Електронний ресурс] – Режим доступу: URL: https://www.javatpoint.com/what-is-tomcat

15. [Електронний ресурс] – Режим доступу: URL: https://techmonitor.ai/hardware/what-is-apache-tomcat

16. [Електронний ресурс] – Режим доступу: URL: https://medium.com/platform-engineer/evolution-of-http-69cfe6531ba0

17. [Електронний ресурс] – Режим доступу: URL: https://www.extrahop.com/resources/protocols/http/

18. Wikipedia [Електронний ресурс] – Режим доступу: URL: https://uk.wikipedia.org/wiki/TCP

19. [Електронний ресурс] – Режим доступу: URL: https://www.ionos.com/digitalguide/server/know-how/introduction-to-tcp/

20. [Електронний ресурс] – Режим доступу: URL: https://www.upwork.com/resources/what-is-an-api

21. [Електронний ресурс] – Режим доступу: URL: https://www.spiceworks.com/tech/devops/articles/application-programming-interface/

22. JWT.IO [Електронний ресурс] – Режим доступу: URL: https://jwt.io/introduction

23. Aditya Y. Bhargava. Grokking Algorithms – Manning– 2021 – 288 с.\

24.[Електронний ресурс] – Режим доступу: URL: https://www.theserverside.com/definition/Hibernate

25.[Електронний ресурс] – Режим доступу: URL: https://www.guru99.com/black-box-testing.html

ДОДАТКИ

Додаток А Диск

Додаток Б Слайди презентації

Додаток В Апробація результатів роботи

Додаток Г Відгук та рецензія