**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**імені ТАРАСА ШЕВЧЕНКА**
Факультет інформаційних технологій
**Кафедра прикладних інформаційних систем**

122 «Комп'ютерні науки»
(шифр і назва спеціальності)

«Прикладне програмування»
(назва освітньої програми)

# Кваліфікаційна робота бакалавра

на тему: «Веб-сайт для замовлення та здійснення доставки вантажів»

Виконав _____
(Підпис)
Шлапак Олег Русланович
(прізвище, ім'я, по батькові)

Керівник доцент Міронова В.Л
(прізвище, ім'я, по батькові)
_____
(Резолюція «До захисту»)

Унікальність тексту 96%
Автор _____ Шлапак О.Р.
(Підпис ) (Прізвище, ініціали)
**Попередній захист:**

_____***23.05.2022***_____
____
(Висновок: "До захисту в екзаменаційній комісії ")

***Завідувач кафедри*** _____ Плескач В.Л.
(Підпис ) (Прізвище, ініціали)
(Дата)

**Київ – 2022**

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

| Номер | Назва етапів кваліфікаційної роботи бакалавра | Термін виконання етапів кваліфікаційної роботи бакалавра | Відмітка про виконання |
|---|---|---|---|
| 1. | Вибір теми та наукового керівника кваліфікаційної роботи бакалавра | 26.10.2021 | виконано |
| 2. | Видача завдання кваліфікаційної роботи бакалавра | 23.11.2021 | заява |
| 3. | Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра | 01.12.2021 | виконано |
| 4. | Затвердження плану кваліфікаційної роботи бакалавра | 18.02.2022 | виконано |
| 5. | Підбір та вивчення літературних та інших джерел з теми дослідження | 25.02.2022 | виконано |
| 6. | Підготовка і подання науковому керівнику першого варіанту І розділу роботи | 05.03.2022 | виконано |
| 7. | Підготовка і подання науковому керівнику першого варіанту ІІ розділу роботи | 09.04.2022 | виконано |
| 8. | Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи | 07.05.2022 | виконано |
| 9. | Подання роботи у першому варіанті | 11.05.2022 | виконано |
| 10. | Оформлення пояснювальної записки кваліфікаційної роботи бакалавра | 12.05.2022 | виконано |
| 11. | Подання кваліфікаційної роботи бакалавра на попередній захист | **23.05.2022** | виконано |
| 12. | Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру | 28.05.2022 | виконано |
| 13. | Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу) | 11.06.2022 | виконано |
| 14. | Захист кваліфікаційної роботи бакалавра | 16.06.2022 | виконано |
| | | 22.16.2022 | виконано |
| | | 23.06.2022 | |
| | | 24.06.2022 | |

Здобувач вищої освіти

Керівник _____

## Анотація

**Шлапак Олег Русланович** виконав випускну кваліфікаційну роботу на тему «Веб-сайт для замовлення та здійснення доставки вантажів» за спеціальністю 122 – «Комп'ютерні науки».

Дипломна робота: 90 с., 28 рис., 15 джерел, 2 дод.

**Метою** дипломної роботи є дослідження та аналіз методик побудови веб-сайтів для замовлення та здійснення доставки вантажів.

Для досягнення мети вирішено такі **завдання**:

- Проведено аналіз існуючих сучасних технологій та методик побудови веб-систем для замовлення та здійснення доставки вантажів.
- На основі проведених досліджень розроблено архітектуру, спроектовано та реалізовано веб-систему вантажних перевезень з використанням MongoDB, NodeJS, Angular, HTML, CSS.
- Проведенно тестування розробленої системи на основі фіктивних даних, згенерованих випадковим чином.

**Об'єкт дослідження**

Процес замовлення та здійснення доставки вантажів

**Предмет дослідження**

Методика побудови веб-систем на основі сучасних технологій.

**Методи дослідження**: теорія системного аналізу та синтезу, теорія систем, теорія баз даних, формальні моделі програмування; мови програмування та мови специфікацій, формальні методи розробки програм тощо.

**Результат дослідження**

Результати дослідження можуть покращити процес розробки веб-систем вантажних перевезень. Розроблена на основі запропонованої технології веб-система, може бути використана для планування та здійснення перевезень невеликого транспортного підприємства.

**Ключові слова**: Веб-система, вантажні перевезення, MongoDB, NodeJS, Angular, HTML, CSS.

## Summary

The degree project: «Website for ordering and goods delivery» has completed by **Oleh Shlapak** speciality 122 – «Computer Science».

The degree project 90 pages

**The purpose** of the degree project is to study and analyze the methods of building websites for ordering and delivery of goods.

To achieve this goal the following tasks was solved:

- An analysis of the development of modern technologies and methods of building web systems for ordering and delivery of goods.
- Based on the research, a system was developed and implemented, using MongoDB, NodeJS, Angular, HTML, CSS.
- Summary testing, developed on the basis of fictitious data, randomly generated systems.

**Object of study**

The object of the study is a web system of ordering and delivery of goods.

**Subject of study**

The subject of study is the methods of building web systems based on modern technologies.

**Research methods**: theory of systems analysis and synthesis, systems theory, database theory, formal programming models; programming languages and specification languages, formal program development methods, etc.

**The result of the study**

The results of the study may improve the process of developing delivery web systems. The web system developed on the offered technology, can be used for planning and implementation of transportations of the small transport enterprise company


**Keywords:** Web system, freight, MongoDB, NodeJS, Angular, HTML, CSS.

**Зміст**

# ВСТУП

У зв'язку із розвитком технологій необхідно все менше контактувати із іншими людьми під час роботи у багатьох сферах, що робить людину яка залучена до даної роботи більш продуктивною та надає їй більше часу для більш якісної роботи, витрачаючи при цьому менше зусиль. Але не дивлячись на велике різноманіття запропонованих інструментів для розробки програмного забезпечення відповідних веб-сайтів, сьогодні немає одноголосного лідера у своїй сфері відповідальності. Саме тому велика кількість різноманітних веб-сайтів побудована на принципово різних підходах та за допомогою різних інструментів. Аналіз існуючих веб-сайтів показав, що у процесі розробки та використання вони дотримуються різних методик, що у свою чергу впливає на якість кінцевого продукту. Також слід зауважити, що різні методики розробки веб-сайту, можуть значно вплинути на його кінцеву вартість, що зробить його розробку не актуальною, або потрібно буде багато часу, щоб даний сайт почав приносити прибуток. То ж потрібно із розумінням вибирати за якою саме методикою розробляти продукт та якими інсрументами користуватись.

**Мета**

Метою даної випускної кваліфікаційної роботи є дослідження та аналіз методик побудови веб-сайтів для замовлення та здійснення доставки вантажів. Розроблений веб-сайт також створить автоматизовані робочі місця для моніторингу зайнятості водіїв, що виконують доставку, та статусу замовлень у реальному часі. Також зменшить загальний штат працівників, за рахунок автоматизованих робочих місць, що у свою чергу збільшить дохід компанії, оскільки створені автоматизовані робочі місця зможуть виконувати більший обсяг робіт у порівнянні із людиною.

Для досягнення мети було виконано наступні завдання

- Провести аналіз існуючих сучасних технологій та методик побудови веб-систем для замовлення та здійснення доставки вантажів.

- На основі проведених досліджень розробити архітектуру, спроектувати та реалізувати веб-систему вантажних перевезень з використанням MongoDB, NodeJS, Angular, HTML, CSS.
- Провести тестування розробленої системи на основі фіктивних даних, що згенеровані випадковим чином.

**Об’єкт дослідження**

Об’єктом дослідження є процес замовлення та здійснення доставки вантажів

**Предмет дослідження**

Предмет дослідження є методика побудови веб-систем на основі сучасних технологій.

**Методи дослідження**: теорія системного аналізу та синтезу, теорія систем, теорія баз даних, формальні моделі програмування; мови програмування та мови специфікацій, формальні методи розробки програм тощо.

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ**

HTML - HyperText Markup Language

CSS - Cascading Style Sheets

SASS - Syntactically Awesome Stylesheets

JS - JavaScript

SQL - Structured query language

PHP - Hypertext Preprocessor, попередня назва: Personal Home Page Tools

SEO - Search Engine Optimization

JSX – Java Script Extension

IDE - Integrated Development Environment

VCS - Version Control System

RDBMS - Relational Database Management System

# РОЗДІЛ 1
# АНАЛІЗ НАЯВНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ СИСТЕМИ ВАНТАЖНИХ ПЕРЕВЕЗЕНЬ

## 1.1 Методи та технології побудови веб-систем

Нині ця сфера дуже швидко розвивається. Що є актуальним сьогодні, може бути вже завтра нікому не потрібним. Тому слід обирати спираючись на ті технології та методи, які відповідають наступним вимогам:

- Відносно нові технології
- Низький поріг входу, тобто освоєння даної технології швидке та легке
- Підтримка даної технології основними специфікаціями
- Популярність технології
- Технологія розвивається

Зараз є популярними такі технології розробки веб-систем:

1. **Прогресивні веб-програми (PWA)** - використовують сучасні веб-API разом зі стратегією прогресивного покращення для створення крос-платформних застосунків. Такі програми запускаються скрізь і мають ряд характеристик, що забезпечують користувачів перевагами, аналогічними тим, що доступні в нативних рішеннях.

- Програми помічаються пошуковими системами.
- Доступні на будь-яких пристроях без потреби в магазині програм.
- Легко поділитись ними через посилання.
- Працюють в офлайні або на низькошвидкісних з'єднаннях.
- працюють для будь-якого користувача.
- Поміщаються на будь-який екран: настільний комп'ютер, смартфон, планшет і т.д.
- Спеціальний спосіб перенесення даних запобігає крадіжці інформації.

2. **Прискорені мобільні сторінки (AMP)** - це технологія прискорених мобільних сторінок із відкритим вихідним кодом, створена для оперативної завантаження сторінок навіть при низькій швидкості мережі.

Якщо подивитися детально, то AMP верстка - це "урізаний" HTML, в якому частина тегів заборонена або замінена на еквівалентні AMP-теги й AMP JS-бібліотеки для швидкого рендеринга (відтворення) AMP-HTML.

Швидкому завантаженню також сприяє кешування сторінок безпосередньо у пошуковій системі Google і Lazy-loading - відмова від завантаження всієї сторінки відразу, тобто елементи сторінки завантажуються за необхідністю при прокручуванні користувачем.

3. **Затосунки однієї сторінки (SPA)** - це веб-застосунок, який повністю знаходиться на сторінці веб-браузера та завантажує необхідні дані у момент завантаження самої сторінки. З'явився порівняно недавно, стрімко набирає популярності . SPA є типовим представником застосунків на HTML5. Тобто, якщо є великий додаток (наприклад із документообігу), що вміщує у собі дані із багатьох файлів, то буде доречно використати саме цю технологію, оскільки завдяки їй дані будуть завантажуватись із серверу тільки тоді, коли буде фактична необхідність у них.

4. **JavaScript фреймворки**. JavaScript є досить старою мовою програмування, але у той же час вона не перестає розвиватись, стаючи при цьому все кращою. Низький поріг входу дозволяє початківцям швидко оволодіти ним на гарному рівні. Сукупність усіх цих факторів робить JavaScript однією із найвикористовуваніших мов програмування на сьогодні, особливо у області веб-розробки

Також слід виділити те, що існує безліч фреймворків, що допомагають користуватись JavaScript більш ефективно. Більшість фреймворків використовують компонентний підхід, що у свою чергу, дозволяє користуватись готовими компонентами та бібліотеками від інших розробників.

5. **Без серверні програми та архітектура.** Використання данного підходу дозволяє уникнути безлічі проблем, наприклад, оптимальне завантаження

ресурсів. Саме цей підхід допомагає витрачати менше бюджету на підтримку серверів, та їх розробку. Базується даний підхід на застосуванні FaaS(тобто функція як послуга).

6. **Адаптивні веб-сайти(RWD).** Концепція підходу у тому, що спочатку розробляється веб-сайт для мобільних пристроїв, після чого, поступово збільшуючи розширення цільових екранів, розробити версію того ж сайту для стандартних(1024 x 1080), а потім і для більших розмірів екрану. При цьому сайт має виглядати однаково добре для усіх пристроїв, незалежно від розмірів. Ключовим моментом є те, що код веб-сторінки повинен максимально залишатись незмінним.

## 1.2. Огляд мов програмування для створення веб-сайтів

Оскільки ми будемо розробляти веб-сайт повністю, то давайте розглядати окремо мови для бекенду та фронтенду, оскільки ще жоден інструмент не дозволяє нам ефективно розробити веб-сайт використовуючи тільки його одного.

### 1.2.1. Розробка Front-end частини

Беззаперечно, основними інструментами будуть HTML та CSS та їхні модифікації як, наприклад, SCSS, SASS та інші. Також увагу слід звернути на JS, оскільки саме на ньому розроблені усі популярні фреймворки Angular, React, VueJS та інші. Також дуже популярною мовою запитів є SQL. Це вузькоспеціалізована мова запитів, якою користуються безліч веб-сайтів та веб-застосунків, саме за допомогою неї звертаються до баз даних моделі.

Далі будемо розглядати фреймворки більш детальніше, оскільки від них напряму залежить успішність та ціна розробки веб-сайту.

**Angular** веб-фреймворк, що дозволяє JavaScript інтегруватися з HTML та CSS. Збудовано понад 400 тисяч сайтів по всьому у яких він використовується. З ним можна розробляти нативні та веб-програми для ПК та мобільних пристроїв. Підходить для корпоративного ПЗ. Його використовують такі компанії як: Google, Microsoft та YouTube.

Кілька переваг:

1. Допомагає створювати прогресивні програми (PWA);
2. Зручно маніпулювати DOM-елементами;
3. Висока швидкість та продуктивність;
4. Вбудований механізм застосування залежностей(Dependency Injection);
5. Підтримка Google та потужна екосистема.

**Aurelia** – набір модулів JavaScript з відкритим вихідним кодом, що написано на ECMAScript. Вони дозволяють розробляти компоненти на JavaScript або

TypeScript. Aurelia потребує менше пам'яті порівняно з іншими JavaScript-фреймворками. Він легко інтегрується зі сторонніми бібліотеками чи фреймворками.

Кілька переваг:

1. Висока продуктивність;
2. Велика спільнота;
3. Адаптивна прив'язка даних;
4. Ефективність пам'яті.

**React** — JavaScript-бібліотека що має просту логіку, та допомагає створювати інтерфейси користувача (UI). Вона дозволяє створювати компоненти інтерфейсу для різних платформ таких як комп'ютери, планшети, мобільні телефони. Він рекомендований до використання у розробці SPA та корпоративних застосунків. На GitHub React займає друге місце серед фреймворків за популярністю. Його використовують: Facebook, Instagram, WhatsApp.

Кілька переваг:

1. Легке об'єднання JavaScript та HTML;
2. Простота розробки динамічних веб-програм;
3. Проста конфігурація;
4. Підтримка спільноти.

**Vue.js** – прогресивний фреймворк, який можна інтегрувати уже з готовими проектами та бібліотеками JS. У 2020 році Vue отримав найбільшу кількість зірок на GitHub, обійшовши при цьому Angular та React. До того ж, за останні декілька років інтерес до фреймворку зріс на 18-20%. Його використовують: Stack Overflow, GitLab, Adobe.

Кілька переваг:

1. Динамічність у використанні;
2. Легкий у вивченні;
3. Підтримка переходів та анімації у CSS;

4. Гнучкість та модульність.

**Meteor** підходить для створення веб-сайтів та мобільних застосунків – у яких можна використовувати один і той же код. Він має ізоморфну систему, яка дозволяє створювати веб-програми у режимі реального часу з нуля. Його можна комбінувати з іншими JavaScript-фреймворками, наприклад Vue, Svelte та Angular. Його використовують: Deloitte, Nordstrom, Accenture.

Кілька переваг:

1. Екосистема ізоморфного розвитку (IDevE);
2. Вбудоване перезавантаження браузера;
3. Користувальницький менеджер пакетів;
4. Потужна хмарна платформа для розгортання, масштабування та моніторингу клієнтських програм;
5. Реактивні шаблони.

**Svelte** — JavaScript-фреймворк з відкритим кодом, створений на основі TypeScript. Він переводить проект у JavaScript під час етапу збирання(build), а не виконання(execute). В результаті розробка програм за допомогою Svelte, швидше ніж з багатьма іншими фреймворками. Його використовують: Codustry, Screeb, Kontist.

Кілька переваг:

1. Швидкість;
2. Висока продуктивність;
3. Легкий і компактний синтаксис;
4. Легкий у вивченні.

### 1.2.2 Розробка Back-end частини

Як із фронтендом існує безліч варіантів вибору інструментів для побудови архітектури бекенду, найпоширеніші із них будуть розглянуті нижче.

**PHP** – є досить старою, але від того дуже поширеною мовою сценаріїв, яку використовують для написання серверної частини веб-застосунків. Перевагою PHP є можливість web-розробникам швидко створювати динамічні web-сторінки. Відзнакою PHP від якого-небудь коду є те, що він виконується на стороні серверу. Також можливо так конфігурувати свій сервер, щоб HTML-файли оброблялися процесором PHP, а клієнти отримували уже готовий результат виконання скрипту.

Express.js - це фонова платформа веб-дизайну для Node.js. Використовується для створення веб-застосунків та різноманітних API. Фреймворк є досить швидким і надає компоненти маршрутизації, підтримує проміжне програмне забезпечення, різні шаблони та інші функції, які роблять розробку зручніше. Він сумісний із базами даних, наприклад MongoDB та MySQL. Його використовують: PayPal, Uber, IBM.

Декілька плюсів:

1. Швидкість;
2. Проста конфігурація;
3. Дозволяє здійснювати динамічну візуалізацію HTML-сторінок;
4. Сумісність з іншими фреймворками;
5. Велика підтримка спільноти.

**Next.js** дозволяє розробляти Jamstack та серверні програми. Поряд з іншими технологіями він також підтримує CSS та стилізований JSX. Окрім цього, в ньому є можливість динамічно імпортувати модулі JavaScript та компоненти React. А також експортувати повністю статичний сайт із програми. Його використовують: Netflix, Github, Avocode.

Декілька плюсів:

1. автоматичне розподілення коду та маршрутизація;
2. підтримка на високому рівні SEO;
3. серверний рендеринг;
4. підтримка гарячого перезавантаження коду;

**Node.js** — технологія, що дуже стрімко розвивається, а також може бути використана у будь якій частині проекту, що робить її використання та вивчення дуже популярним, оскільки знаючи лише її можна починати створювати комплексні веб-застосунки. Підтримка асинхронного виконання, робить платформу максимально продуктивною, наскільки це можливо для однопотокової моделі.

Кілька переваг:

1. Асинхронна одно-потокова модель виконання коду;
2. Неблокуюче завантаження модулів, та виконання коду;
3. Самодостатня платформа;
4. Система модулів CommonJS;
5. Високопродуктивний рушій V8.

## 1.3 Програмне забезпечення для створення веб-сайтів

Весь процес створення веб-сайту можна поділити на три основні частини:

- Розробка бекендного API
- Створення макету або прототипу сайту
- Розробка лейауту сайту

Нижче наведено список програм, що можуть бути корисними у процесі створення веб-сайту

**WebStorm** - це інтегроване середовище для розробки на технологіях, що відносяться до JavaScript. IDE створена на базі платформи IntelliJ, розробленої JetBrains і випущеної під відкритою ліцензією. Зробить розробку програм швидшою та приємнішою. Особливо слід звернути увагу на дане середовище фронт-енд та фулл-стек розробникам, оскільки у IDE є повний набір інструментів, що можуть знадобитись. А якщо таких немає, то система плагінів дозволить їх завантажити.

WebStorm IDE надає наступні можливості:

**Автодоповнення коду.** Допоможе пришвидшити написання коду за рахунок підказок, що пропонуються по ходу написання. У тому числі ключові слова, назви змінних, методів та класів Наприклад, коли працюєте із CSS будуть пропонуватись імена класів при роботі із .js файлами. Також використовується машинне навчання для кращих пропозицій, для автодоповнення.

**Аналіз якості коду**. WebStorm може допомогти у пошуку помилок та підказок, щодо їх виправлення. Існує вбудована база інспекцій для безлічі мов. Також слід звернути увагу на вбудовану систему перевірки правопису, схожу на ту що використовується у Word. Також є інтеграція зі Stylelint, ESLint та іншими лінтерами – вони запускаються автоматично після їх конфігурації та дозволяють статично перевіряти код, по ходу його написання.

**Безпечний рефакторинг**. WebStorm дозволяє перейменовувати файли, папки та символи, виймати компоненти, методи та змінні та не боятися, що ваші

дії призведуть до помилок. Існує механізм для запобігання помилок, під час рефакторингу.

**Попередній перегляд HTML-файлів**. WebStorm надає можливість попереднього перегляду статичних HTML-сторінок, а також автоматичне збереження та оновлення при їхньому редагуванні

**Юніт-тестування**. Вбудований механізм дебагу та запусу юніт тестів. Що дозволяє у цій же IDE створювати, запускати та слідкувати за проходженням юніт тестів. Стандартні плагіни, дозволяють використовувати Jest, Mocha, Karma, Protractor та Cucumber.js для написання тестів.

**Розширена інтеграція з VCS**. За допомогою вбудованого інтерфейсу у WebStorm можна з легкістю робити усі операції із VCS, забувши про консоль та десятки команд. Наприклад: завантажувати зміни у репозиторій, порівнювати гілки, дивитись історію коммітів, перемикатись між гілками та безліч інших команд

**Postman** – це емуляція HTTP-клієнта для тестування розробленого API. За допомогою нього тестують коректне надсилання запитів від клієнта до серверу та отримання потрібної відповіді назад. Використовує HTTP протокол для надсилання запитів. Зазвичай використовується тестувальниками для перевірки коректної роботи API. За допомогою Postman тестувальник може складати та відправляти HTTP-запити до API, створювати папки із запитами для скорочення часу тестування та багато іншого

**Figma**(Фігма) – один із найпопулярніших це графічних редакторів для створення та удосконалення макетів для веб-сайтів Цим інструментом можуть користуватись: маркетологи, дизайнери, менеджери та розробники. У Фігмі можна відображати елементи інтерфейсу, створити інтерактивний прототип сайту та програми, векторну графіку та різноманітні ілюстрації. Більшість дизайнерів роблять у ній макети сайтів для Тільди.

Рисунок 1.1 - Приклад інтерфейсу у Figma

**Adobe Photoshop** – графічний редактор, що широко використовується для редагування зображень різних розширень, графічного дизайну та інших видів растрової графіки. Використання шарів для забезпечення глибини та гнучкості у процесі проектування та редагування дозволяє швидко та якісно виконувати свою роботу фахівцям. З роками Фотошоп перетворився із звичайної програми для професіоналів на доступний редактор, із функціями якими користуються в багатьох сучасних професіях. Навіть назва програми означати певні видозміни зображення.

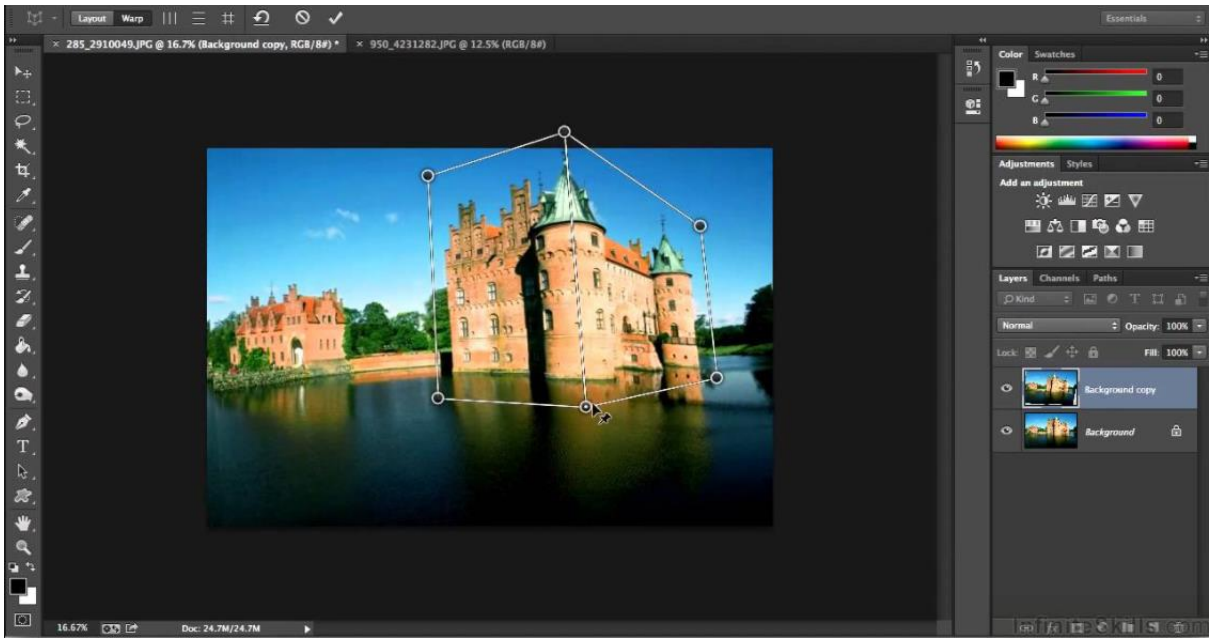



Рисунок 1.2 - Інтерфейс програми Adobe Photoshop

**NPM**. найбільша у світі бібліотека (реєстр) програмного забезпечення. У ньому знаходяться близько мільйона різноманітних пакетів на будь-який смак. Будь-хто може викласти власний пакет із кодом, який буде доступний усім бажаючим. Також багато престижних фірм використовують його для власних потреб, роблячи приватні пакети, які доступні лише певним розробникам, у даному випадку працівникам цих фірм. Бібліотека є повністю безкоштовною, тому можна завантажувати будь-які пакети, проте варто звернути увагу на ліцензію, оскільки деякі із пакетів можна використовувати лише для некомерційних проектів. Перелік усіх встановлених пакетів зазвичай знаходиться у файлі package.json

**Emmet** — це плагін для багатьох популярних текстових редакторів, який значно покращує робочий процес HTML і CSS. Скорочення за допомогою плагіна Emmet дозволяє значно збільшити швидкість верстки за рахунок комбінації команд і абревіатур. EMMET простий в установці, інтегрується в такі редактори як PHPStorm, Sublime Text, Adobe Dreamviewer, Notepad++, WebStorm, Eclipse, CodeMirror, HTML-Kit та багато інших

**Angular Material** — це реалізація специфікації матеріального дизайну Google (2014-2017). Цей проект надає набір багаторазових, добре перевірених і доступних компонентів інтерфейсу користувача для розробників на фреймворку Angular. Основна ідея Angular Material у послідовності та розширенні користувацького інтерфейсу. Також Angular Material дає можливість контролювати поведінку різних компонентів. Використання цього інструменту зменшує час розробки веб-сайту за рахунок використання уже готових компонентів, а також підвищує загальну якість проекту, оскільки компоненти представлені у Angular Material уже протестовані та безпечні до використання. Повний список компонентів доступний за наступним посиланням: https://material.angular.io/components/categories

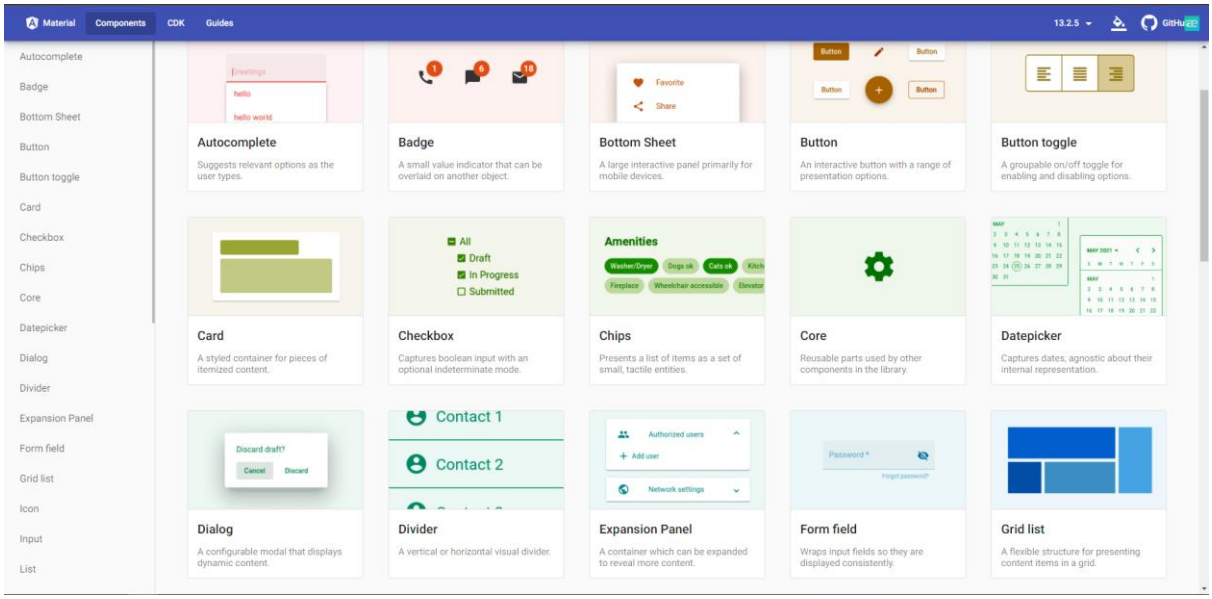



Рисунок 1.3 – Перелік компонентів Angular Material

## 1.4 Сучасні підходи до організації архітектури веб-систем

Провідною тенденцією сьогодні є залучення потенційних клієнтів до веб-сайту за допомогою потужних та красивих додаток веб-застосунків.

Зазвичай вони виглядають так, ніби це звичайні програми, але вони здатні працювати через Інтернет, що значно підсилює інтерес користувачів, оскільки їх не потрібно завантажувати та встановлювати.

Саме тому вибір компонентів та модель розробки програми грає важливу роль у успішності розробленої системи, оскільки може статися так, що програма буде нікому не цікавою, а кошти, що були вкладені у розробку ніхто не поверне.

В архітектурі веб-застосунку описано основні процеси що описують взаємодію між підсистемами. Якщо архітектура зроблена добре, то можна із впевненістю сказати, що кілька програм не будуть заважати одна одній при паралельній роботі.

Зазвичай додаток працює наступним чином:

1. Користувач переходить за URL-адресою
2. Відбувається запит на сервер
3. Сервер у відповідь надсилає відповідний файл чи файли
4 Браузер обробляє їх та показує користувачеві
5 Користувач може взаємодіяти із отриманою сторінкою

Важливо звернути увагу, що у отриманому файлі може не бути інструкцій, що описують реакцію браузера на дії користувача (наприклад різні типи даних користувача)

Роблячи висновок, гарно розроблена архітектура повинна мати підсистеми зовнішнього обміну даними для веб-сайту або застосунку, та включати всі необхідні підкомпоненти для функціонування.

### 1.4.1 Client-side Code та Server-side Code

Архітектура веб-застосунків є основою веб-застосунків в сучасному Інтернеті, оскільки зараз більша частина глобального мережевого трафіку, а також дуже багато програм і пристроїв користуються Інтернет-комунікаціями. Тому архітектура пов’язана не тільки з ефективністю, але й безпекою, масштабованістю та надійністю.

Зазвичай веб-застосунок складається із двох основних підпрограм:

1. **Клієнтський код** – код, який обробляє безпосередньо браузер. Саме із цим кодом взаємодіє користувач через різноманітні поля вводу, кнопки і т.д.

2. **Код на стороні сервера** – код, що відповідає на запити HTTP. Знаходиться на стороні серверу. Взаємодіє лише через визначені API

Поділ коду на серверний та клієнтський відбувається командою розробників, саме вони вирішують, за що буде відповідати сервер, а що буде виконуватись на стороні клієнта(у браузері)

Якщо код, може відповідати на HTTP запити, то його можна використовувати на стороні серверу. Найбільш поширеною є практика, коли серверний код відповідає за створення сторінки, а також за збереження даних, що ввів користувач, а клієнтський код – за реагування на дії користувача. Також, оскільки клієнтський код є відкритим, варто бути обережним із важливими даними, які поточний користувач не повинен бачити.

### 1.4.2 Веб-компоненти застосунку

Характеризуючи веб-компоненти, ми маємо на увазі один із наступних нижче:

1) **Компоненти веб-застосунків типу UI/UX** – це різноманітні частини веб-сайту. До них відносять сповіщення, інформаційні панелі, налаштування, різні види статистики. У своїй природі вони ніяк не перетинаються із архітектурою веб-застосунків. Проте є складовими макета інтерфейсу веб-застосунку

2) **Клієнтський компонент** – це компонент, розроблений на CSS, HTML і JS, або за допомогою фреймворків для JS. Існує лише на стороні клієнта, тобто у браузері, тому не потрібно змінювати налаштування операційної системи або налаштувань самого пристрою, на якому відкритий браузер .

3) **Серверний компонент**. Переважна більшість серверних компонентів включає у себе базу даних та логіку, за якою буде організована взаємодія із нею. Також логіка компоненту визначає шляхи взаємодії між клієнтським та серверним кодом, за допомогою API.

### 1.4.3 Моделі компонентів веб-застосунків

Враховуючи сумарну кількість серверів і баз даних, які використовуються для веб-застосунку, вибирається модель веб-застосунку. Це може бути одна із наступних моделей відношення баз даних до серверів:

1. Один до одного

Є найвразливішою моделлю, але у той же час найлегша у розумінні та розробці. Відповідно до назви, існує лише одна база даних та сервер. Веб-додаток, побудований на такій моделі, вийде з ладу, як тільки сервер вийде з ладу. Отже, він не дуже надійний.

Модель компонента веб-програми з одним веб-сервером і однією базою даних зазвичай не використовується для реальних веб-застосунків. Зазвичай не

використовується у реальних проектах, оскільки має багато мінусів, але саме те, для вивчення розуміння роботи веб-застосунків.

2. Один до багатьох

Принцип роботи полягає у наступному, після отримання веб-сервером інформації, вона не зберігається на ньому, лише обробляється, а передається за його межі, де записується у базу даних, якою керують за межами цього сервера Це іноді також називають архітектурою без стану. Для цієї моделі компонента веб-програми потрібні щонайменше 2 веб-сервери. Це все для того, щоб уникнути невдач. Навіть коли один із веб-серверів виходить з ладу, інший бере на себе відповідальність.

Усі зроблені запити будуть автоматично перенаправлені на новий сервер, і веб-додаток продовжить виконання. Таким чином, надійність краща в порівнянні з одним сервером з притаманною моделлю бази даних. Однак, якщо база даних аварійно завершує роботу, веб-додаток також завершить роботу.

3. Багато до багатьох

Вважається найстабільнішою за рахунок альтернативних точок взаємодії, тобто якщо один сервер вийде з ладу, інформація буде збережена на іншому. Існує два варіанти цієї моделі. Або зберігати ідентичні дані у всіх використаних базах даних або розподіляти їх між ними рівномірно.

У першому випадку зазвичай потрібно не більше 2 баз даних, тоді як в другому при поломці бази даних, буде втрачено цінні дані, що не можуть бути відновлені. При великому обсягу даних рекомендується використовувати не менше п’яти екземплярів баз даних та веб-серверів, а також потрібно встановити балансувальники навантаження

## Висновки

У цьому розділі було описано та проаналізовано існуючі рішення проектування систем вантажних перевезень. Було розглянуто та описано існуюче програмне забезпечення, що використовується сьогодні для побудов відповідних веб-систем, а саме – найпопулярніші інструменти розробки, проектування та масштабування.

Під час аналізу існуючих технологій веб-систем я дізнався про найпопулярніші підходи розробки, їхні переваги та недоліки, а також вимоги до них, що допоможуть обрати кращий підхід відповідно до вимог системи, що буде розроблятись.

Також було розглянуто популярні на сьогодні інструменти для розробки Front-end та Back-end частин системи, інструменти для керування базою даних, та інше програмне забезпечення, що може полегшити розробку веб-системи вантажних перевезень.

У процесі дослідження існуючих інструментів підходів та мов програмування я значно розширив та поглибив свої знання у цих сферах та навчився підібрати відповідні інструменти під поставлену задачу.

# РОЗДІЛ 2
# РОЗРОБКА АРХІТЕКТУРИ ТА РЕАЛІЗАЦІЯ ВЕБ-СИСТЕМИ ВАНТАЖНИХ ПЕРЕВЕЗЕНЬ

## 2.1 Розробка архітектури веб-систем для замовлення та здійснення доставки вантажів

Оскільки передбачається, що веб-система має бути доступна на різних платформах, таких як комп'ютер, телефон, планшет та ін. Ми можемо побудувати архітектуру на принципі клієнт-сервер.

Архітектура клієнт-сервер характеризується, як мережа, що використовує більшу частину ресурсів знаходиться на сервері, та обслуговує клієнтів, які до нього звертаються. Така архітектура дає визначення наступним компонентам:

**Сервер**, що надає інформацію або інші послуги абонентам, які звертаються до них;

**Клієнт**, який звертається до серверу за необхідною інформацією, що міститься на ньому;

**Мережа**, структура ,що забезпечує зв'язок міз сервером та клієнтом.

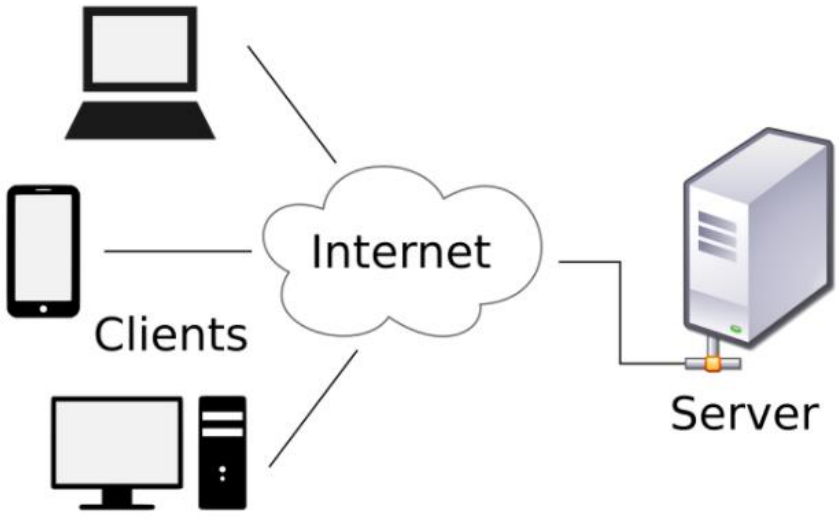



Рисунок 2.1 - Представлення клієнт-серверної архітектури

Основна ідея такої моделі - це зв’язок із клієнта із сервером, що обробляє усі запити та надсилає результат роботи назад

Зазвичай сервери є багато потоковими, що дозволяє обробити декілька запитів одночасно. Якщо приходить більше одного запиту на сервер, то вони виконуються сервером послідовно у тому порядку, що прийшли. Деякі запити можуть мати пріоритети, що пришвидшують або сповільнюють їх обробку.

Функціональні вимоги серверу:

1. Зберігання, доступ, захист і резервне копіювання даних;
2. Обробка клієнтського запиту;
3. Відправлення результату (відповіді) клієнту.

Функціональні вимоги клієнта:

1. Надання інтерфейсу користувача;
2. Формування запитів до сервера і його відправлення;
3. Отримання результату від сервера;

Така архітектура визначає точки взаємодії між сервером та клієнтом, правила ж визначені в мережевому протоколі.

Існують наступні мережеві протоколи:

**TCP/IP** – це вся мережа , що працює за допомогою двох протоколів – TCP та IP.

**TCP** (Transfer Control Protocol) – протокол, що служить для встановлення надійного з'єднання між двома клієнтами та передачі інформації про підтвердження її отримання.

**IP** (Internet Protocol) – інтернет протокол, відповідає за правильність доставки повідомлень за певною адресою. Під час доставки дані розбиваються на пакети, що можуть доставлятись по різному.

**MAC** (Media Access Control) – протокол, що відповідає за ідентифікацію, кожен користувач мережі має власну унікальну MAC адресу

**ICMP** (Internet control message protocol) – відповідає за обмін інформацією, проте не відповідає за передачу даних.

**UDP** (User datagram protocol) – інформація, отримана через даний протокол не перевіряється при отриманні, але оскільки він є швидшим ніж TCP, має право на існування .

**HTTP** (Hyper Text Transfer Protocol) – протокол за допомогою якого працюють всі веб-сайти, являє собою протокол передачі гіпертексту

**SSH** (Secure Shell) – захисний протокол, для віддаленого керування по захищеному каналу.

Також існують дворівневі та трьохрівневі архітектури на принципі клієнт-сервер:

**Дворівнева архітектура складається з двох вузлів**:

1) Сервер, відповідальний за отримання запитів та відправлення відповіді клієнту. Використовує лише власні ресурси;

2) Клієнт, представляє користувацький інтерфейс, що формує та надсилає запити. Також обробляє відповіді від серверу.

Принцип роботи: робота серверу базується обробці та відповіді на отриманий запит. Не використовує інші ресурси.

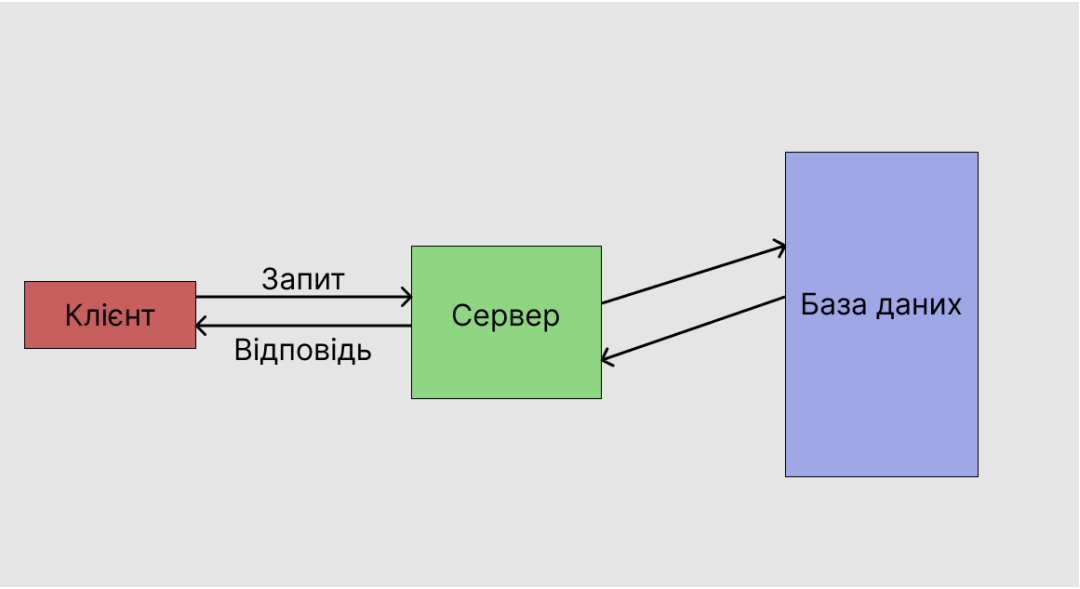



Рисунок 2.2 - Дворівнева клієнт-серверна архітектура

**Трирівнева архітектура складається з трьох вузлів, що наведені нижче**:

1) Прикладний компонент – сервер застосунків;
2) Представлення даних – інтерфейс користувача;
3) Керування ресурсами – компонент відповідальний за надання інформації.

Працює наступним чином: декілька серверів обробляють запит клієнта, що знижує навантаження на сервер, дозволяючи збільшити кількість оброблених запитів за той же час. Також даний підхід можна розширювати до багаторівневої архітектури.

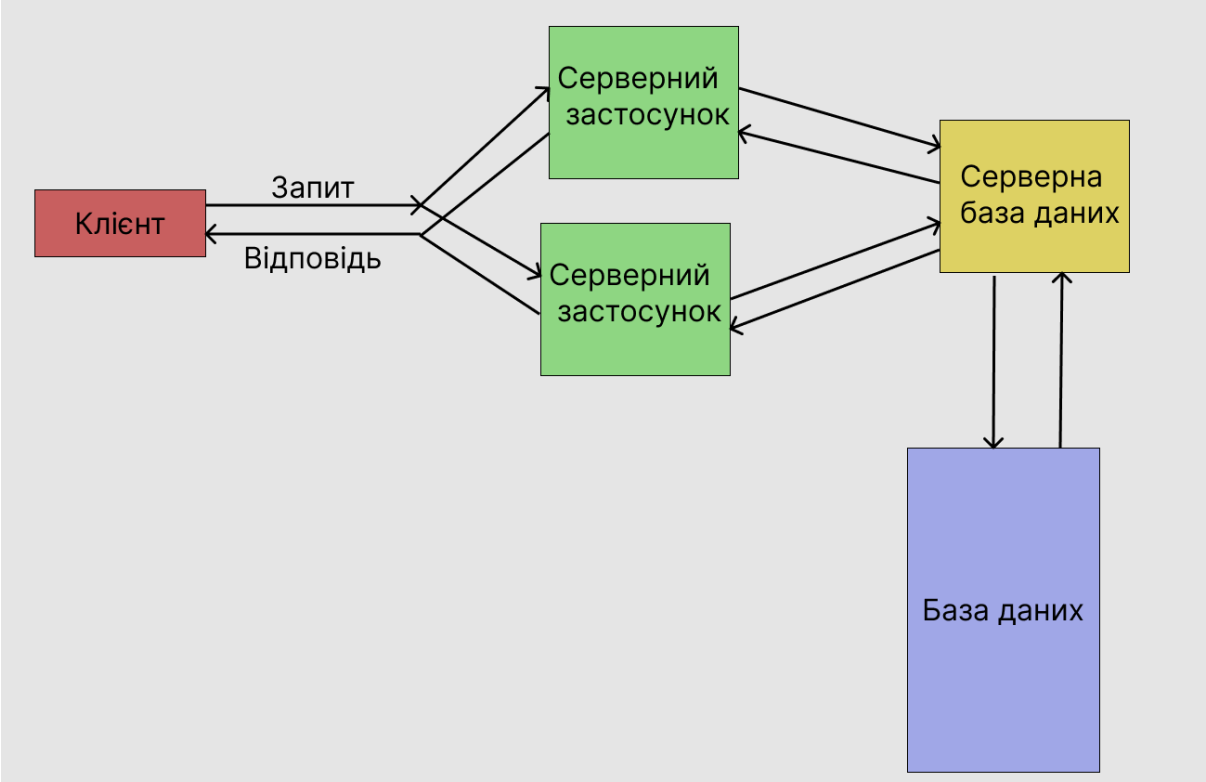



Рисунок 2.3 - Трирівнева клієнт-серверна архітектура

## 2.2 Розробка архітектури бази даних

Технологія баз даних змінювалася та розвивалася з роками. Реляційні, NoSQL, ієрархічні є найпопулярнішими на сьогодні, але не варто забувати й про інші види, що допоможуть кожному підібрати певну базу даних для його потреб. Давайте розглянемо наступні типи баз даних

**1) Реляційні бази даних**

Реляційні бази даних існують з 1970-х років. У таблицях дані зберігаються в рядках і стовпцях. Найбільш поширеною мовою, яка використовується під час роботи з реляційними базами даних є мова структурованих запитів (SQL). ЇЇ використовують для усіх дій із базою даних(створення, видалення, вибірка, редагування, та інші). Реляційні бази даних показали себе дуже надійними. Вони сумісні із принципом ACID (атомність, послідовність, ізоляція, довговічність), що є стандартним набором властивостей для надійних операції з базами даних. Реляційні бази даних добре працюють зі структурованими даними. Організації, які мають багато неструктурованих або напівструктурованих даних, не повинні розглядати реляційну базу даних.

Приклади: Microsoft SQL Server, Oracle Database, MySQL, PostgreSQL та IBM Db2.

**2) Бази даних NoSQL**

NoSQL — це широка категорія, яка включає будь-яку базу даних, яка не використовує SQL як свою основну мову доступу до даних. На відміну від реляційних баз даних, дані в базі даних NoSQL не повинні відповідати попередньо визначеній схемі, тому ці типи баз даних чудово підходять для організацій, які прагнуть зберігати неструктуровані або напівструктуровані дані. Варто зазначити, що у NoSQL базу даних можна вносити зміни прямо на льоту, не впливаючи на програми, які використовують базу даних.

Приклади: Apache Cassandra, MongoDB, CouchDB і CouchBase.

### 2.2.1 Cloud MongoDB

Для бази даних було використано Cloud MongoDB, тому що вона дає наступні переваги:

1. Надання високодоступних кластерів MongoDB через UI, CLI або API.
2. Вбудована можливість автоматичного резервного копіювання.
3. Автоматичне оновлення кластерів з мінімальними перешкодами для проекту.

4. Зводить до мінімуму час простою та робить базу даних високодоступною за допомогою резервних вузлів. Ці вузли додають надлишковість і негайно починають обслуговувати запити, якщо ваш основний вузол виходить з ладу.
5. Допомагає легко справлятися зі зростанням трафіку, масштабуючи свої кластери, коли це необхідно.
6. Всі кластери розміщено в мережі VPC, що робить їх захищеними та зашифрованими.

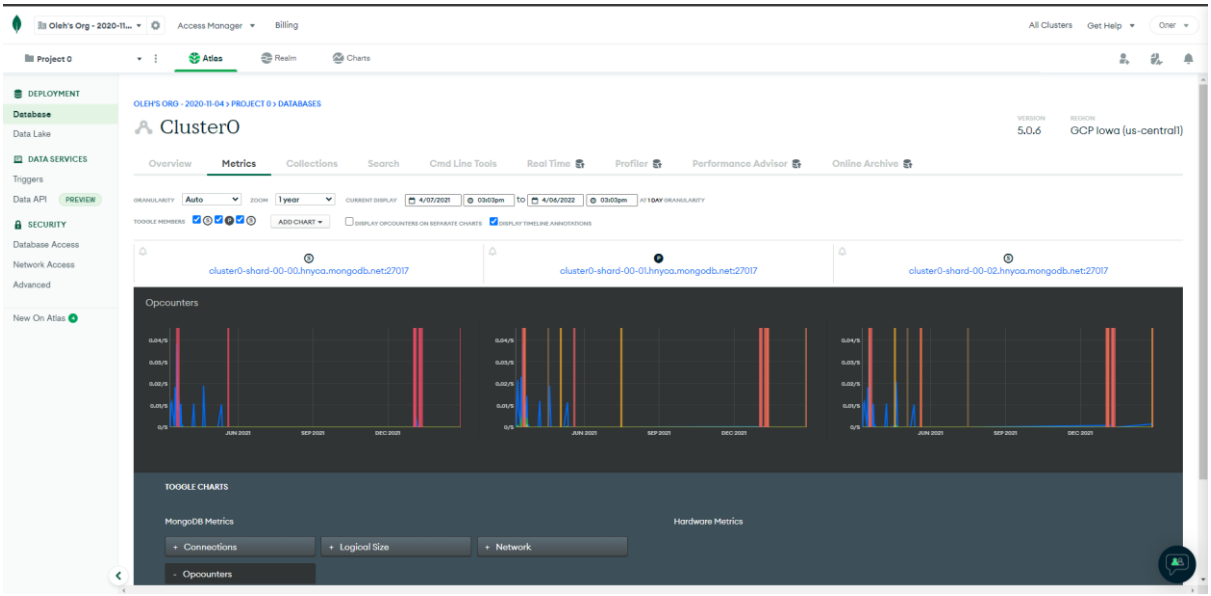



Рисунок 2.4 - Інтерфейс Cloud MongoDB

### 2.2.2 Моделі та колекції у базі даних

Усі дані будуть розміщуватись у 4 колекціях:

**loads** – у цій колекції зберігаються дані про замовлення

**registrationcredentials** - у цій колекції зберігаються конфіденційні дані про користувача

**trucks** - у цій колекції зберігаються дані про вантажівки

**users** - у цій колекції зберігаються загальні дані про користувача

У кожній колекції дані будуть мати певну модель:

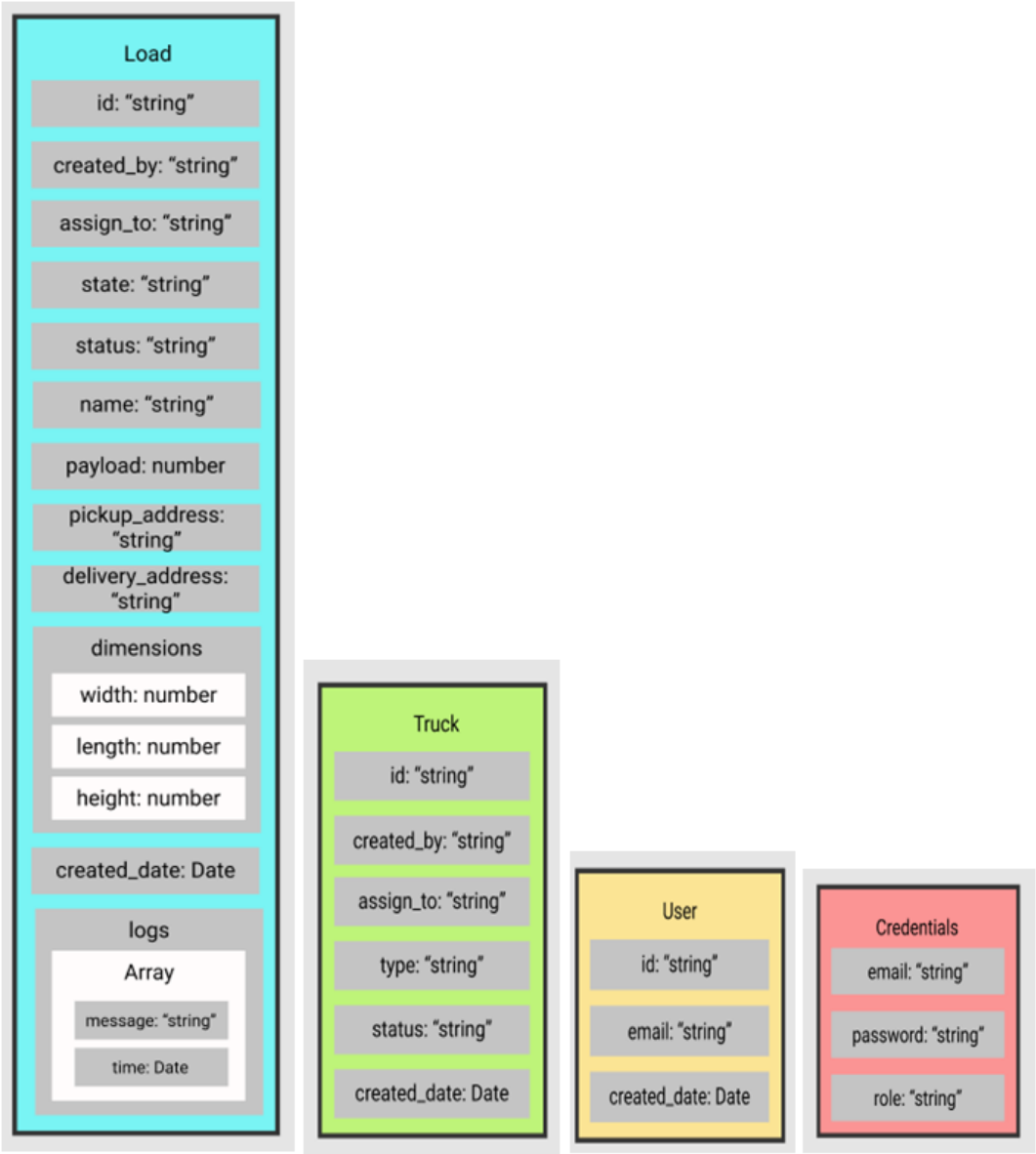



Рисунок 2.5 - Моделі даних у колекціях

Для управління даними, використовувався веб-сайт MongoDB Atlas, що дозволяє ефективно управляти процесами створення нових та моніторинг існуючих колекцій, об’єктів у них.

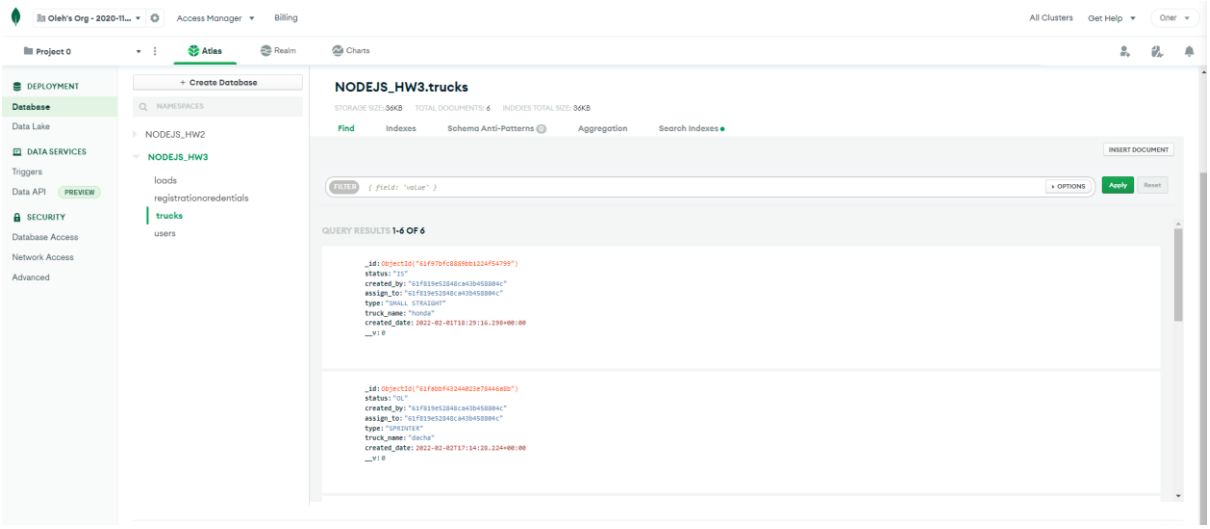



Рисунок 2.6 - Приклад відображення колекції у середовищі MongoDB Atlas

### 2.3 Розробка Back-end частини (код в додатку А)

Під час розробки Back-end частини було дотримано підходу під назвою REST API. Комунікацію клієнтської частини із базою даних було розроблено за допомогою веб-сервісу.

Основна задача веб-сервісу – надання ресурсів у потрібному форматі, що буде використовуватись іншими користувачами у власних цілях. Коли говорять про веб-сервіси, зазвичай мають на увазі запити та відповіді на них, тобто спілкування між сервером та клієнтом.

Зараз найбільш популярним підходом до розробки веб-сервісів є REST і SOAP.

Слід зазначити, що мова, на якій написано веб-сервіс та мова клієнтської частини не мають значення, оскільки передача даних відбувається за допомогою протоколу HTTP, тобто якщо у вас веб-сервіс написано на Ruby, а фронт-енд частина, на Angular, то це ніяк не завадить обміну даними між ними, оскільки спочатку запит, а потім відповідь будуть конвертовані у стандарт для передачі даних через HTTP

#### 2.3.1 Створення API-інтерфейсів

Для зручності усі ендпоінти або API-інтерфейси було розділено на 4 категорії відповідно до колекцій у базі даних, цим самим визначено для кожної категорії зону відповідальності.

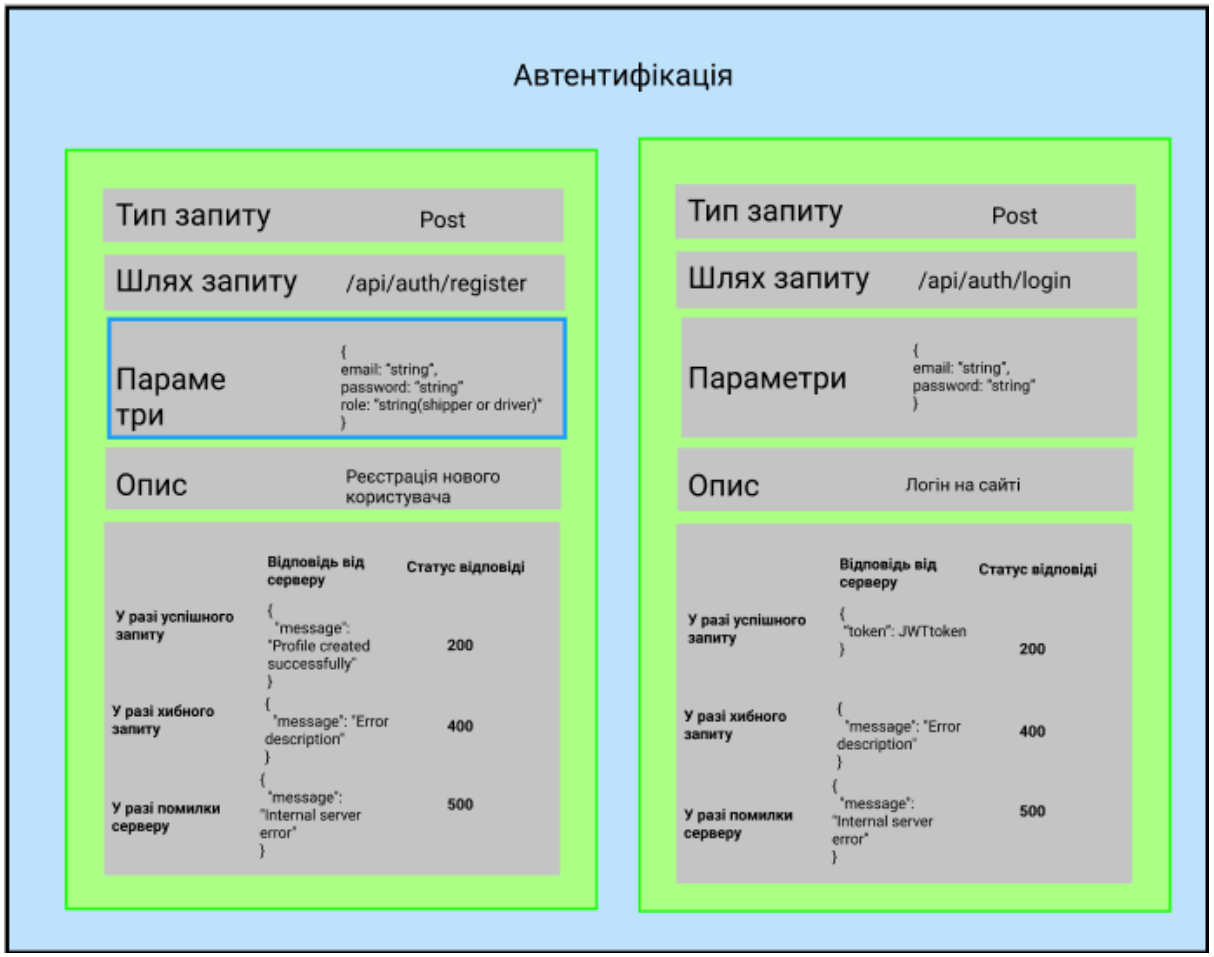
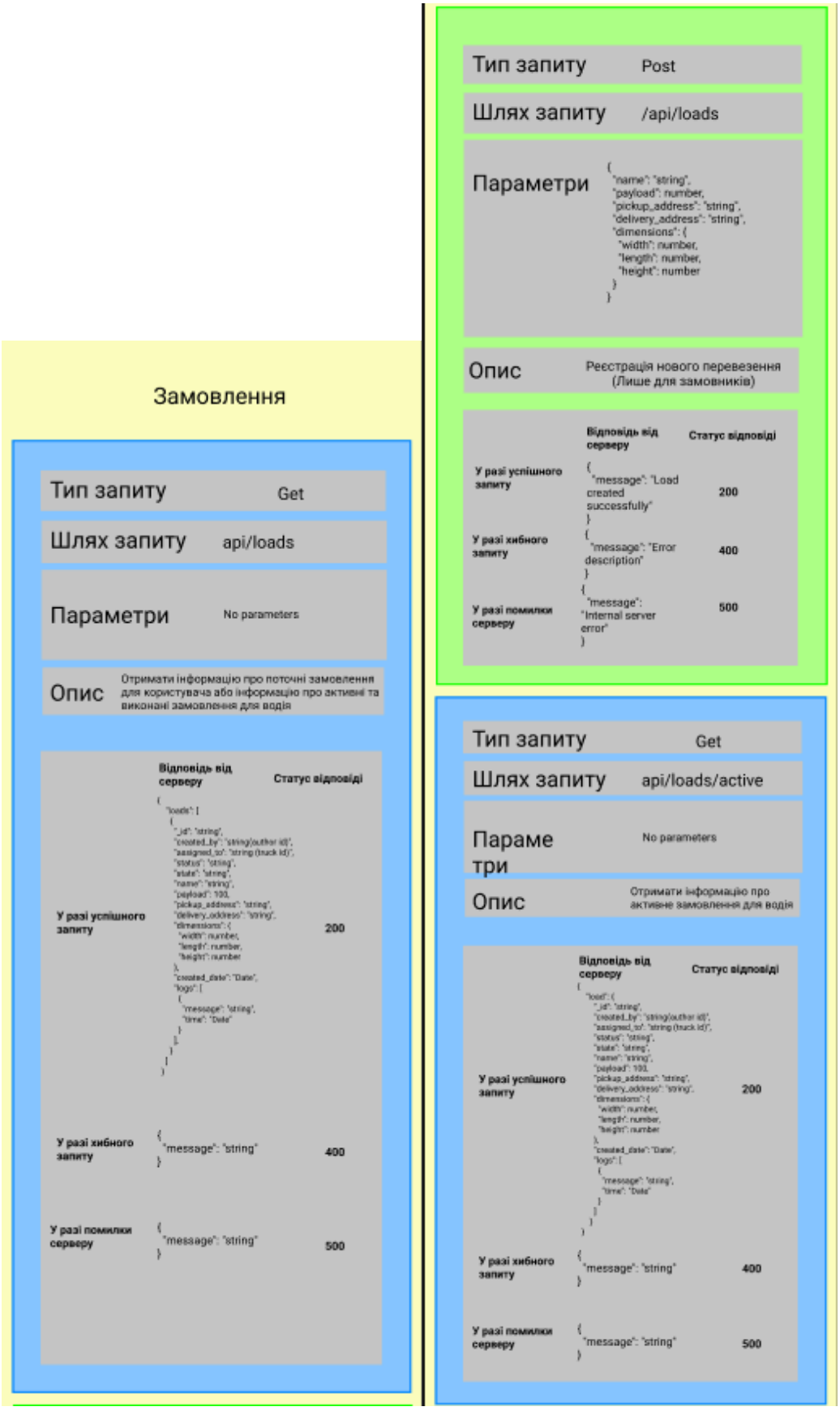



Рисунок 2.7 - Запити із категорії автентифікації


Рисунки 2.7 та 2.8 - Запити із категорії вантажівки




Рисунки 2.9 та 2.10 - Запити із категорії замовлення (частина 1)


Рисунки 2.11 та 2.12 - Запити із категорії замовлення (частина 2)


Рисунки 2.13 - Запити із категорії замовлення (частина 3)


Рисунки 2.13 - Запити із категорії користувач

Розроблення відбувалось на мові програмування JavaScript на платформі Node.js, що може використовуватись для розроблення Back-end частини. Також у процесі використовувались наступні бібліотеки:

- Express
- Mongoose
- Cors

Для прикладу візмемо модуль авторизації та розглянемо його детальніше

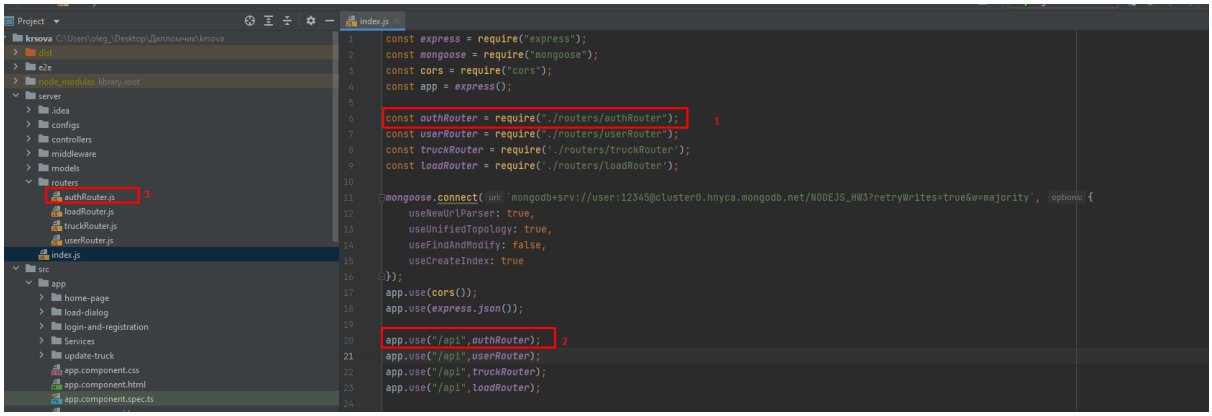



Рисунок 2.14 - Вміст файлу index.js

Для початку було створено файл authRouter.js, (3) на рисунку 2.14

Потім присвоєно його відносний шлях змінній (1) на рисунку 2.14

Після цього за допомогою бібліотеки Express додано його до використання у якості API (2) на рисунку 2.14.

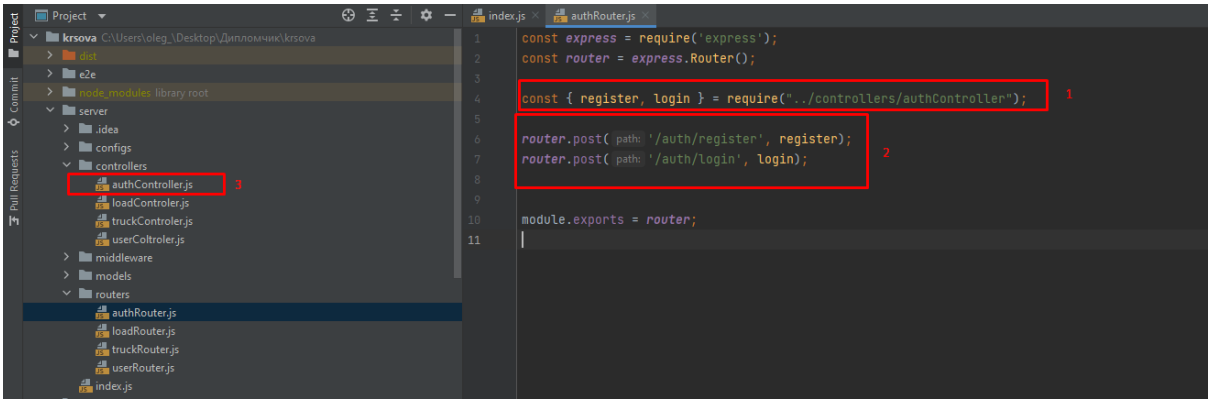



Рисунок 2.15 - Вміст файлу authRouter.js

У файлі authRouter.js ми бачимо що на певні запити URL викликаються відповідні функції (2) на рисунку 2.15, що знаходяться та експортуються із файлу authController.js (1) на рисунку 2.15, також видно знаходження самого файлу (1)

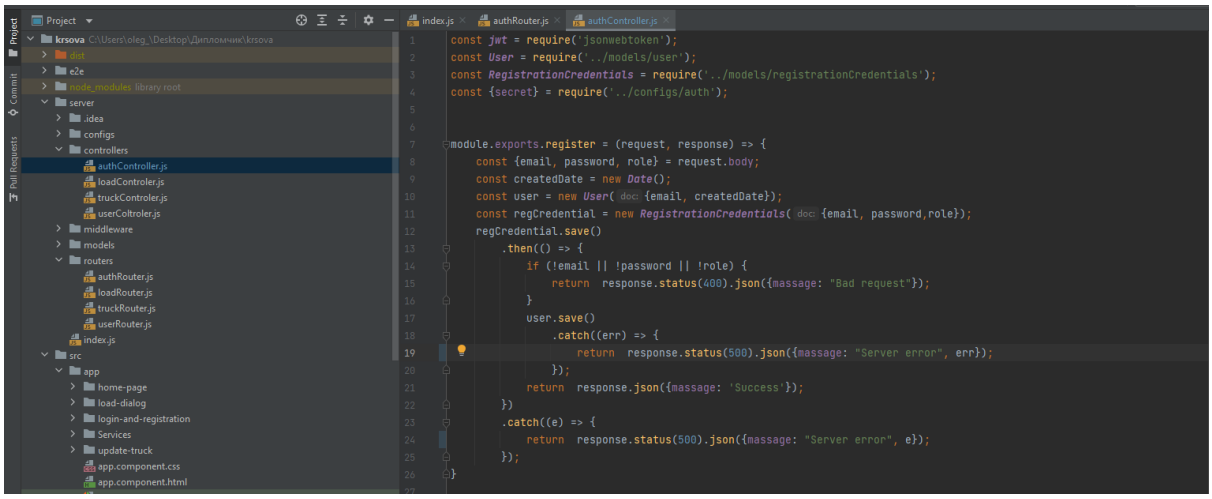



Рисунок 2.16 - Вміст файлу authController.js

Як видно на рисунку вище, файл експортує метод register(), що відповідає за запит, описаний на Рисунку 2.7. Для перевірки чи користувач дійсно існує, метод login() використовує jsonwebtoken, із однойменної бібліотеки, що є зашифрованим ключем авторизації та є унікальним для кожного користувача.

## 2.4 Розробка Front-end частини (код у додатку Б)

Розробка Front-end частини відбувалась за допомогою фреймворку Angular та веб-компонентів із бібліотеки Angular Material. Ця бібліотека є безкоштовною. Її використання вважаю доречним, оскільки вона має безліч плюсів:

1. Висока якість компонентів, оскільки її творці – професіонали, та займаються підтримкою та вдосконаленням своєї бібліотеки.
2. Надає прості API для користування.
3. Можливість легко модифікувати та дороблятиі існуючі компоненти, що збереже час.
4. Легка інтеграція до поточних проектів, що також є чудовим

### 2.4.1 Архітектура компонентів Front-end частини

Для розроблення архітектури нам необхідно зрозуміти, як саме користувач зможе взаємодіяти із сайтом, та які дії від нього нам потрібно обробляти. Тож було побудовано наступну схему взаємодії (User WorkFlow)

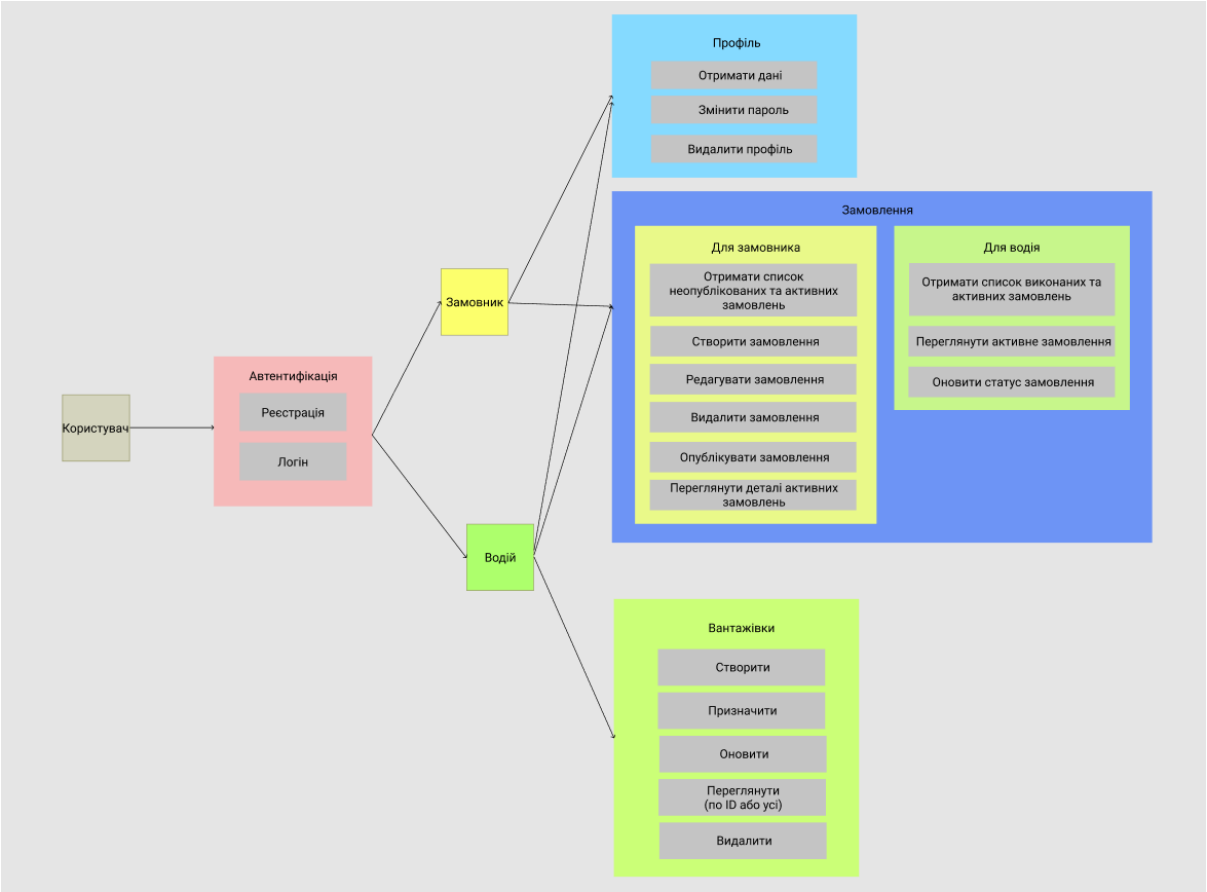



Рисунок 2.17 - Схема взаємодії із користувачем

Як можна побачити із рисунку вище, після аутентифікації та авторизації користувач отримує одну із наступних ролей:

Замовник – це людина, що може створити та опублікувати замовлення для компанії.

Водій – це людина, що виконує доставку вантажів від замовника.

Кожна роль має свої особливості та різні можливості взаємодіяти із базою даних через веб-сайт

Для реалізації поставлених цілей було побудовано наступну файлову структуру:

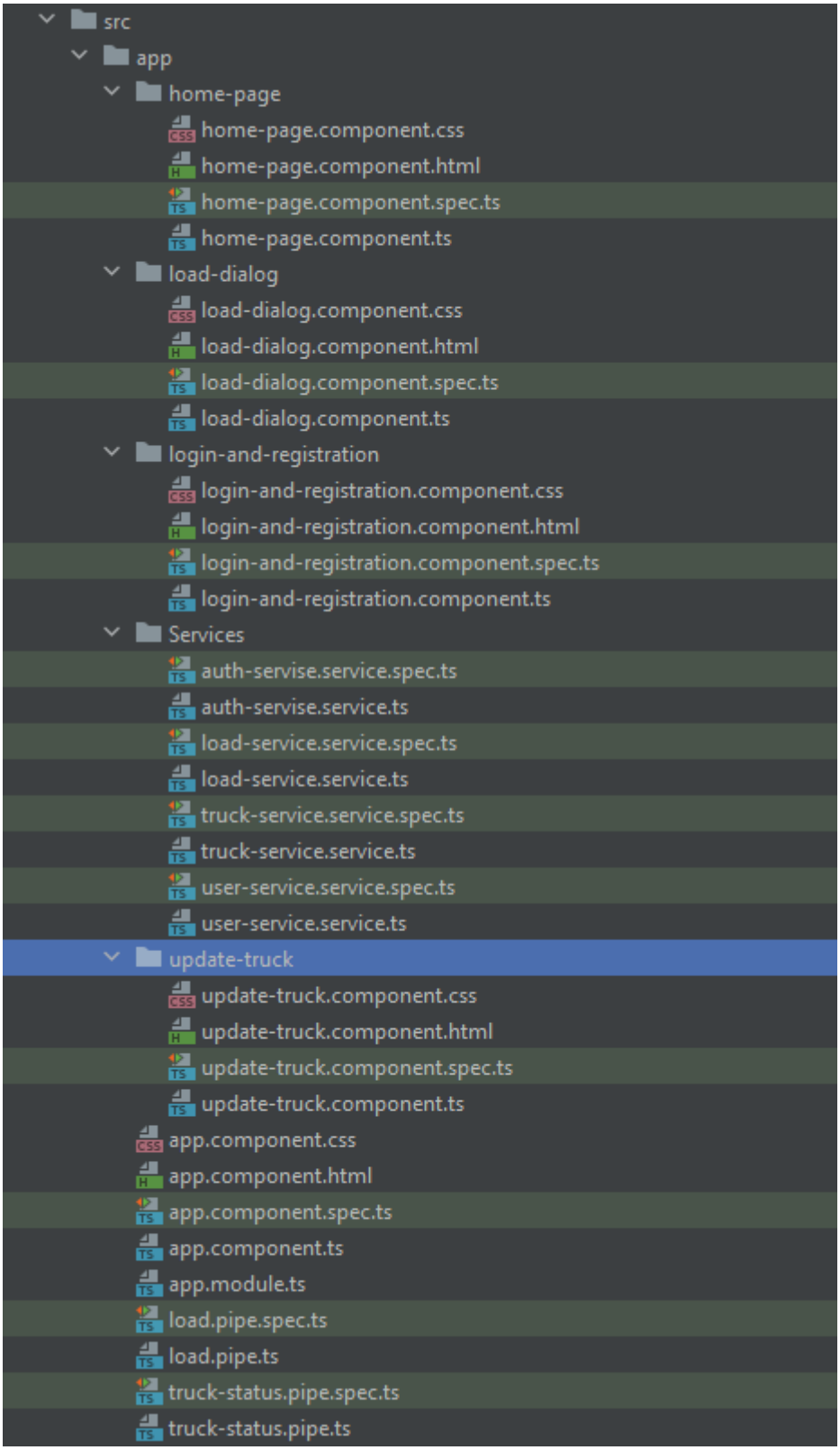



Рисунок 2.18 - Файлова структура Front-end частини

Як можна побачити на рисунку вище проект складається із 4 компонентів вищого рівня:

- login-and-registration – компонент що відповідає за форму реєстрації та логіну
- home-page - компонент що відповідає за надання інтерфейсу зареєстрованим користувачам
- load-dialog - компонент що відповідає за модальне вікно при редагуванні замовлення
- update-truck - компонент що відповідає за модальне вікно при редагуванні вантажівки

### 2.4.2 Розробка проекту на прикладі форми реєстрації та логіну

Розберемо на прикладі форми реєстрації та логіну, тож спочатку потрібно встановити необхідні компоненти із бібліотеки, для цього необхідно у терміналі проекту прописати наступний скрипт, що підключить нам саму бібліотеку

```
ng add @angular/material
```

Після цього потрібно імпортувати необхідні компоненти в необхідний модуль проекту, як це зроблено на рисунку нижче.

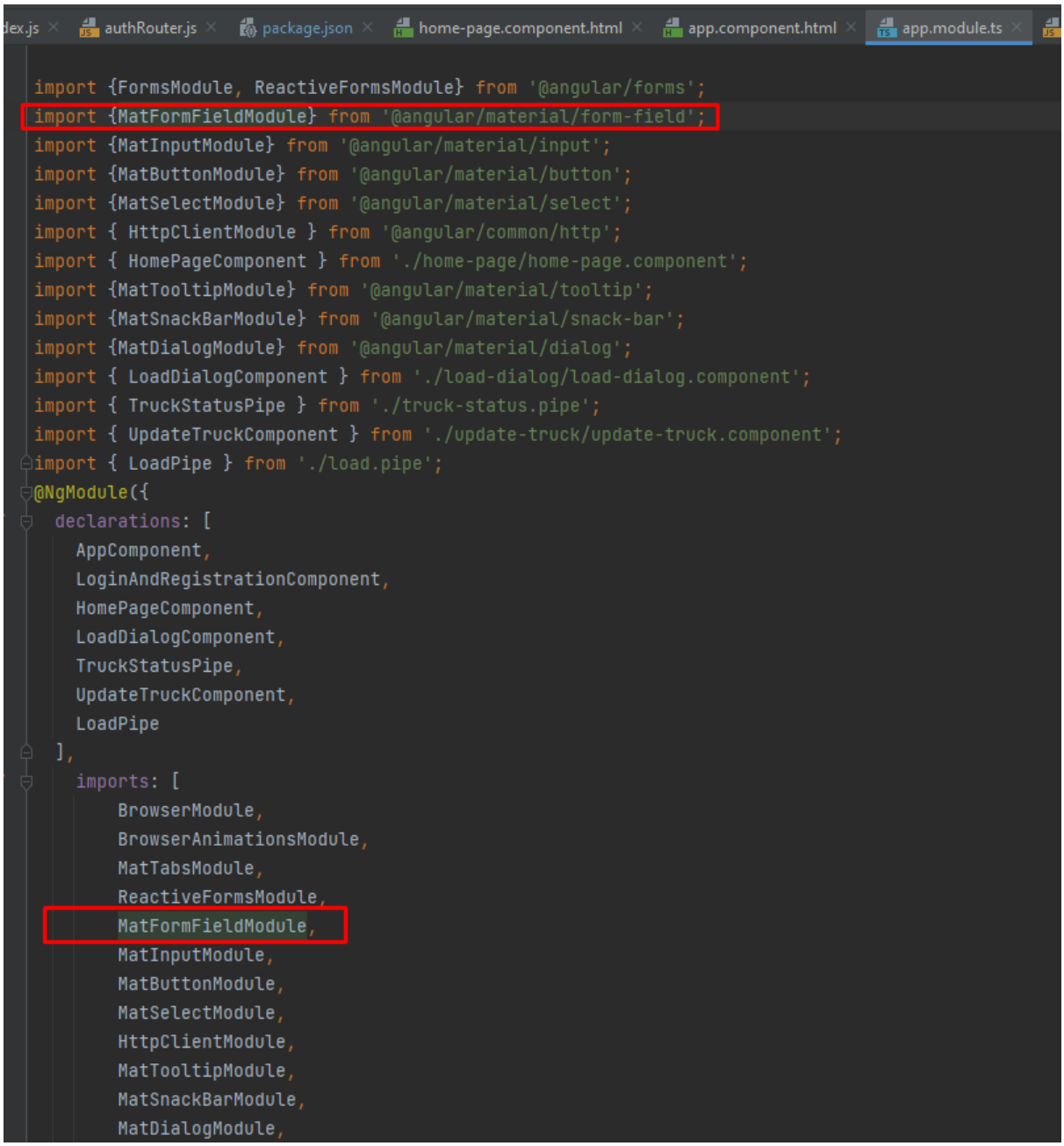



Рисунок 2.17 - Імпорт компонента із бібліотеки Angular Material

Так потрібно підключити усі необхідні компоненти, у нашому випадку вони будуть наступними:

- mat-tab-group
- mat-tab
- mat-form-field
- mat-label
- mat-error
- mat-select

- mat-option

Після підключення компонентів їх можна використовувати, для прикладу наведено HTML код форми логіну.

```
<form class="login-form form" [formGroup]="loginForm">
  <h2>Login form</h2>
  <mat-form-field class="full-width form-field">
    <mat-label>Email</mat-label>
    <input type="email" matInput formControlName="email" placeholder="Email" onfocus="this.setAttribute('autocomplete', 'none');">
    <mat-error *ngIf="loginForm.controls.email.errors">
      Please enter a valid email address
    </mat-error>
  </mat-form-field>
  <mat-form-field class="full-width form-field">
    <mat-label>Password</mat-label>
    <input type="text" matInput formControlName="password" placeholder="password" onfocus="this.setAttribute('autocomplete', 'none');">
    <mat-error *ngIf="loginForm.controls.password.errors">
      Minimum password length 6 symbols
    </mat-error>
  </mat-form-field>
  <button mat-raised-button  (click)="login()" color="primary">Login</button>
</form>
```

Рисунок 2.18 - HTML код для форми реєстрації

Як можна побачити на рисунку вище, виділено кнопку, при натисканні на яку визветься функція **login()**, сама функція знаходиться у **.ts** файлі та виглядає наступним чином.

```
constructor(private fb: FormBuilder, private authService: AuthServiseService) {  2
}

ngOnInit(): void {
  this.loginForm = this.fb.group( controlsConfig: {
    email: ['', Validators.email],
    password: ['', Validators.minLength( minLength: 6)]
  });
  this.registerForm = this.fb.group( controlsConfig: {
    email: ['', Validators.email],
    password: ['', Validators.minLength( minLength: 6)],
    role: ['', Validators.required]
  });
  this.authService.IsLogged.subscribe(value => this.isLogged = value);
}

login(): void {
  if (this.loginForm.valid) {
    const value = this.loginForm.value;
    this.authService.login(value.email, value.password);  1
    this.loginForm.reset();
  }
}
```

Рисунок 2.19 - login-and-registration.component.ts

Як можна побачити функція перевіряє форму на валідність, а також викликає функцію login із **authService**, передаючи туди 2 параметри логін та пароль. Сам сервіс підключений у конструкторі та має наступний вигляд.

```
export class AuthServiseService {
  IsLogged ;
  serverPath;
  UserToken;
  constructor(private http: HttpClient) {
    this.serverPath = environment.serverPath;
    if (localStorage.getItem( key: 'token') !== null){
      this.IsLogged = new BehaviorSubject<boolean>( _value: true);
      this.UserToken = new BehaviorSubject(localStorage.getItem( key: 'token'));
    }
    else {
      this.IsLogged = new BehaviorSubject<boolean>( _value: false);
      this.UserToken = new BehaviorSubject( _value: null);
    }
  }

  login(email: string, password: string): void {
    this.http.post( url: `${this.serverPath}/api/auth/login`,  body: {email, password})
      .subscribe( next: (res: any) => {
        if (res.token) {
          this.IsLogged.next(true);
          localStorage.setItem('token', res.token);
          this.UserToken.next(res.token);
          console.log('login success');
        }
      });
  }
```

Рисунок 2.20 AuthServiseService.ts

Як видно на рисунку вище функція **login()**, відправляє на сервер запит, щоб перевірити чи є користувач у базі даних, та якщо він є присвоїти змінній **IsLogged** значення true. Що у свою чергу допустить користувача на сайт, із відповідною роллю, що знаходиться у змінній **UserToken**.

## Висновки

У розділі було розроблено архітектуру та реалізовано веб-систему для замовлення та здійснення доставки вантажів. Під час роботи над розробкою архітектури я поглибив свої знання про архітектуру виду «клієнт-сервер». Дізнався багато нового про основні мережеві протоколи а також їх види. Отримав уявлення про різно рівневі види архітектур, їхні переваги та недоліки.

Під час розробки архітектури бази даних я поглибив свої знання із різновидів БД, їхні переваги та недоліки відносно інших, а також дізнався який тип бази даних краще використовувати у тому чи іншому проекті. Поглибив свої знання та навички при роботі із нереляційною базою даних.

Під час реалізації веб-системи я вдосконалив свої навички роботи із наступними технологіями: Angular , MongoDB,  Js, Css, Ts, HTML.

# РОЗДІЛ 3
# ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ ДЛЯ ЗАМОВЛЕННЯ ТА ЗДІЙСНЕННЯ ДОСТАВКИ ВАНТАЖІВ

## 3.1 Формування Unit тестів системи вантажних перевезень

Юніт тестування - це написання тестів для окремого блоку коду, що перевіряє правильність його виконання. Юніт тести перевіряють виконання окремих функцій, невеликих класів, сервісів та ін. Загалом всього що можна назвати одиницею програмного коду. Юніт тестування проводиться під час розробки програми розробниками. Модульні тести виділяють розділ коду та перевіряють його правильність. Блоком може бути окрема функція, метод, процедура, модуль або об'єкт.

Перш ніж починати розробку юніт тестів потрібно визначитись із тим, чи потрібні вони взагалі, а також визначити параметри, яким вони будуть відповідати.

Тож, оскільки ми розробляємо систему для замовлення та здійснення доставки вантажів, планується подальша підтримка цього коду у майбутньому, а також розширення функціоналу. Тому без юніт тестування не обійтись, адже юніт тести забезпечать гарну якість коду та його легку масштабованість.

Також важливо, щоб тести відповідали наступним вимогам:

1. Бути достовірними
2. Не залежати від оточення, на якому вони виконуються
3. Легко підтримуватись
4. Легко читатись і бути простими для розуміння (навіть новий розробник повинен зрозуміти що саме тестується)
5. Дотримуватися єдиної конвенції іменування

Для прикладу демонстрації юніт тестування було взято форму логіну та реєстрації. Самі юніт тести знаходяться у файлах із розширенням **.spec.ts**, та мають наступний вигляд.

```
describe( description: 'login', specDefinitions: function() {
  it( expectation: 'should call login service method with login and password params', assertion: function() {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    app.loginForm = { valid: true, value: { email: 'test@email', password: "testPassword" }, reset: () => {}}

    app.login();
    expect(authService.login).toHaveBeenCalledWith( params: 'test@email', 'testPassword');
    expect(authService.login).toHaveBeenCalled();
  });
});

describe( description: 'register', specDefinitions: function() {
  it( expectation: 'should call register service method with login, role and password params', assertion: function() {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    app.registerForm = { valid: true, value: { email: 'test@email', password: 'testPassword', role: 'SHIPPER'}, reset: () => {}}

    app.register();
    expect(authService.register).toHaveBeenCalledWith( params: 'test@email', 'testPassword', 'SHIPPER');
    expect(authService.register).toHaveBeenCalled();
  });
});
});
```

Рисунок 3.1- login-and-registration.component.spec.ts

Функція **describe()** – визначає блок або функцію, що тестується

Функція **It()** – визначає сам юніт тест

Функція **Expect()** – перевіряє отриманий результат із очікуваним

Функція **toHaveBeenCalledWitht()** – приймає параметри, що були передані у функцію, що перевіряється. Якщо ці параметри не були передані, то повертає false значення. Функція **toHaveBeenCalled()** – перевіряє чи функція була викликана.\

Для запуску юніт тестів на фреймворку Angular потрібно запустити команду **ng test** у терміналі, після чого вбудована бібліотека **karma** запустить браузер, у якому автоматично виконаються усі юніт тести у проекті.

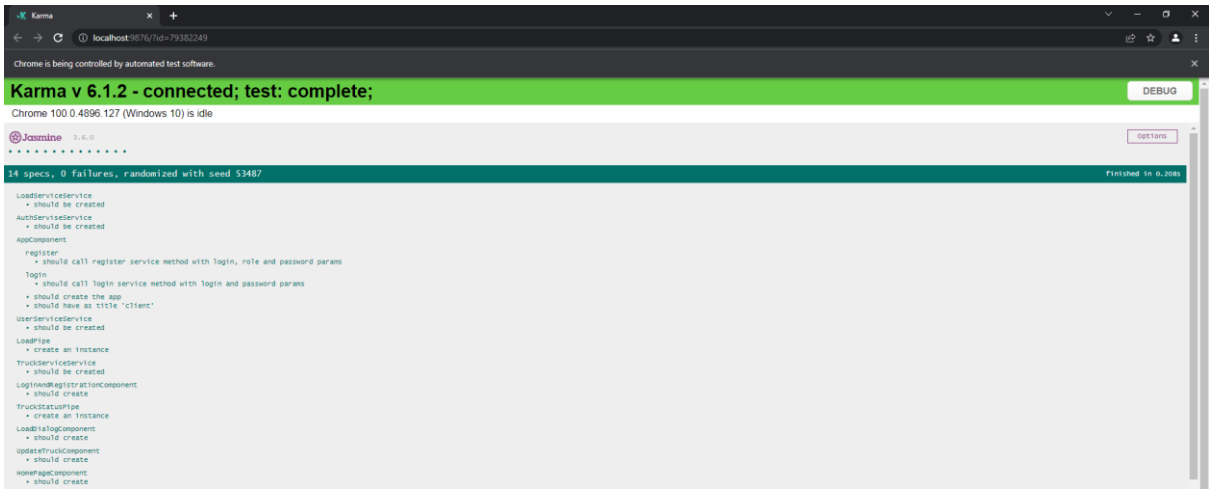



Рисунок 3.3 - Приклад виконання юніт тестів за допомогою **Karma**

Також можна подивитись загальне покриття коду юніт тестами за допомогою звіту від цієї ж бібліотеки, він формується автоматично у проекті. Для того, щоб його переглянути потрібно перейти у папку проекту та запустити файл, що знаходиться за наступним шляхом

coverage/{Назва проекту}/index.html

| File | | Statements | | Branches | | Functions | | Lines | |
|---|---|---|---|---|---|---|---|---|---|
| src | | 100% | 3/3 | 100% | 0/0 | 100% | 0/0 | 100% | 3/3 |
| src/app | | 100% | 36/36 | 100% | 23/23 | 100% | 10/10 | 100% | 31/31 |
| src/app/Services | | 100% | 76/76 | 100% | 8/8 | 100% | 27/27 | 100% | 69/69 |
| src/app/home-page | | 100% | 84/84 | 100% | 12/12 | 100% | 42/42 | 100% | 81/81 |
| src/app/load-dialog | | 53.33% | 8/15 | 76.67% | 23/30 | 100% | 7/7 | 50% | 7/14 |
| src/app/login-and-registration | | 100% | 17/17 | 100% | 5/5 | 100% | 8/8 | 100% | 51/51 |
| src/app/update-truck | | 47.37% | 9/19 | 25% | 1/4 | 25% | 2/8 | 44.44% | 8/18 |
| src/environments | | 100% | 1/1 | 100% | 0/0 | 100% | 0/0 | 100% | 1/1 |

Рисунок 3.2 - Покриття коду юніт тестами

Як можна побачити на рисунку вище компонент login-and-registration повністю покритий юніт тестами, а саме:

- 51/51 ліній коду
- 17/17 виразів
- 5/5 розгалужень
- 8/8 функцій

## 3.2 Опис функціональних можливостей та інструкція використання розробленої веб-системи

Розроблена веб-система виконує допомагає налагодити зв’язок замовника із людиною, що буде займатись перевезенням, а також автоматизувати цей процес.

Передбачено 2 основних сценарії поведінки користувачів:

1. Замовлення вантажного перевезення.
2. Здійснення вантажного перевезення.

Для здійснення замовлення необхідно виконати наступні кроки:

1. Ввійти у особистий кабінет як замовник.

Для цього необхідно скористатись однією із наступних форм, для нових користувачів необхідно у полі “Your role” вибрати “Shipper”

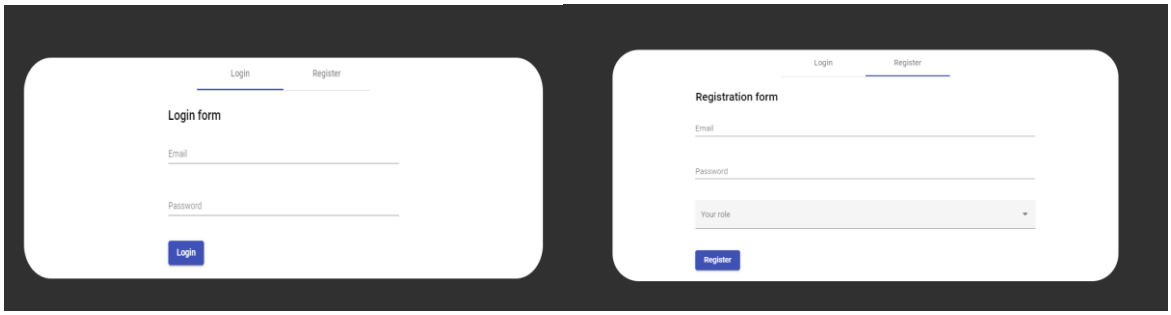



Рисунок 3.4 та 3.5 - Форми логіну та реєстрації

2. Перейти на сторінку створення замовлення, заповнити усі необхідні дані та натиснути на кнопку “Save the load” для збереження у якості чернетки, для подальшої публікації або “Save and post” для безпосередньої публікації замовлення

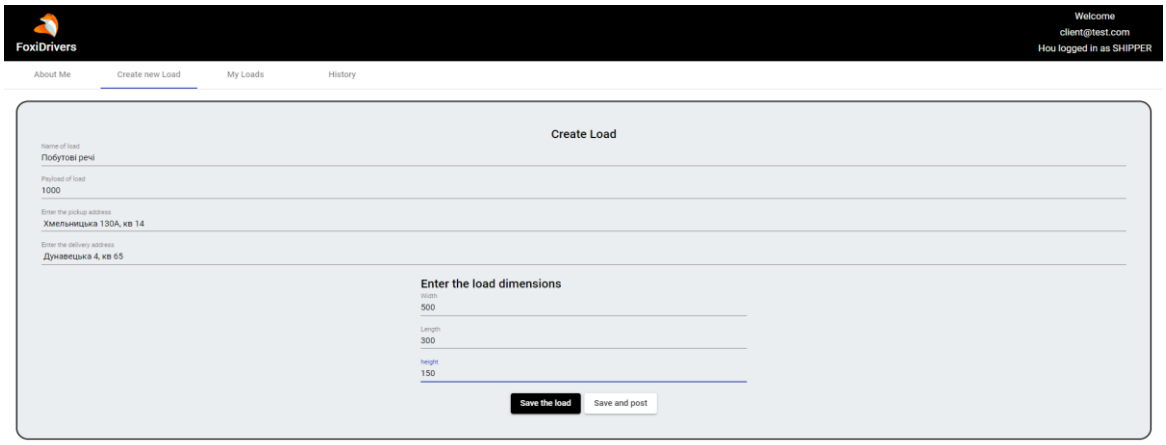



Рисунок 3.6 - Приклад заповненої форми завмовлення

3. У разі успішного назначення водія буде показано наступне повідомлення, після чого статус замовлення буде вважатись “У процесі виконання”, а за поточним його станом можна буде слідкувати на сторінці “Мої замовлення”

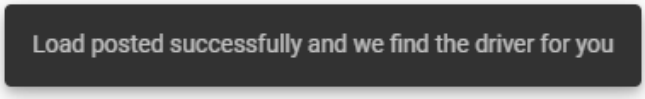



Рисунок 3.7 - Повідомлення про успішне назначення водія

4. Після здійснення перевезення, замовлення автоматично перейде у архів, після чого буде вважатись закритим

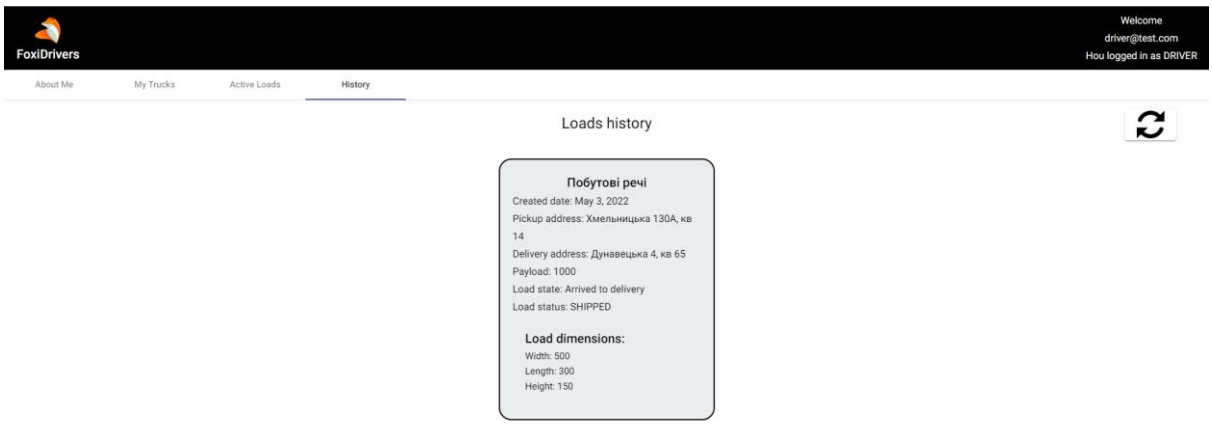



Рисунок 3.8 Сторінка архіву

**Висновки**

У розділі було описано побудову та архітектуру юніт тестування на прикладі форми реєстрації та логіну, а також наведено приклади юніт тестів. Також описано функціональні можливості розробленої веб-системи.

Під час роботи над частиною я дізнався що таке юніт тестування, навіщо воно потрібне, та чим воно може бути корине у проекті. Також дізнався про основні критерії доцільності використання бібліотек для створення та запуску юніт тестування.

У процесі розробки юніт тестів я дізнався які функції надає бібліотека Karma, основні її функції та класи, а також багато нового про методи та підходи для тестування веб-систем.

Можна стверджувати, що вони створюють додаткову роботу розробникам, але у той же час, юніт тести допомагають швидко та відносно недорого забезпечити швидку перевірку основного функціоналу розробленої системи.

# ВИСНОВОК

Сьогодні все частіше можна побачити автоматизовані робочі місця у тій чи іншій сфері діяльності. Тенденція до цього лише збільшується. У ході виконання дипломної роботи було проведено аналіз існуючих сучасних технологій та методик побудови веб-систем для замовлення та здійснення доставки вантажів. На основі проведених досліджень розроблено архітектуру, спроектувано та реалізовано веб-систему вантажних перевезень з використанням MongoDB, NodeJS, Angular, HTML, CSS.

Також проведено тестування розробленої системи на основі фіктивних даних, що згенеровані випадковим чином. Реалізована веб-система зможе виконувати функції оператора зв’язку замовника із адміністратором компанії, що займається вантажними перевезеннями. Також розроблений код системи покритий юніт тестами, що допоможе у майбутньому легше масштабувати та удосконалювати дану веб-систему.

Практичне значення полягає у подальшому використанні методик побудови веб-систем вантажних перевезень, що дозволять будувати високопродуктивні веб-системи різного рівня складності, але у той же час будуть легкими для розширення та розуміння.

## ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Прогресивні веб застосунки [Електронний ресурс] // 2022р. Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps
2. Створення AMP сторінок [Електронний ресурс] // 2022р. Режим доступу до ресурсу: https://evergreens.com.ua/ua/articles/amp-technology.html
3. 15 провідних тенденцій веб розробки у 2020 році [Електронний ресурс] // - 2020р. Режим доступу до ресурсу: https://web4u.in.ua/blog/15-prov-dnih-tendenc-y-veb-rozrobki-u-2020-roc-31
4. Офіційний сайт компанії JetBrains [Електронний ресурс] // 2022р. Режим доступу до ресурсу: https://www.jetbrains.com/
5. Графічний редактор Figma [Електронний ресурс] // 2022р. Режим доступу до ресурсу: https://www.figma.com/
6. Що таке NPM ? [Електронний ресурс] // - 2022р. - Режим доступу до ресурсу: https://ivanmalaniak.pp.ua/webdev-blog/nodejs/shcho-take-npm/
7. Бібліотека Karma [Електронний ресурс] // 2022р. Режим доступу до ресурсу: https://karma-runner.github.io/latest/index.html \
8. Клієнт-серверна Архітектура [Електронний ресурс] // 2022р. – Режим доступу до ресерсу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/
9. Офіційний сайт фреймворку Angular [Електронний ресурс] // 2022р. Режим доступу до ресурсу: https://angular.io/
10. Адам Фрімен Pro Angular 9: Build Powerful and Dynamic Web Apps.2020р. 809 с.
11. Anton Moiseev та Yakov Fain Angular Development with TypeScript 2018р. 1288 с.
12. Nate Murray, Ari Lerner, Felipe Coury, Carlos Taborda ng-book 2: The Complete Book on Angular 2 (Volume 2). 2016. 626 с.
13. Джессі Пальмер, Корінна Кон, Крейг Нішіна, Майкл Джамбальво Testing Angular Applications. 2018 р. 240 с.
14. Jeremy Wilken Angular in Action. 2018р. 749 с.
15. Dan Maharry TypeScript Revealed. 2013р. 104 с.

16. Яновицька А.В. Договір міжнародного перевезення вантажів автомобільним транспортом. 2019 р. 180 с.

**ДОДАТОК А**

**Лістинг Back-end частини**

**index.js**

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const app = express();

const authRouter = require("./routers/authRouter");
const userRouter = require("./routers/userRouter");
const truckRouter = require('./routers/truckRouter');
const loadRouter = require('./routers/loadRouter');

mongoose.connect(`mongodb+srv://user:12345@cluster0.hnyca.mongodb.net/NODEJS_HW3?retryWrites=true&w=majority`, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useFindAndModify: false,
    useCreateIndex: true
});
app.use(cors());
app.use(express.json());

app.use("/api",authRouter);
app.use("/api",userRouter);
```

```
app.use("/api",truckRouter);
app.use("/api",loadRouter);



app.use(express.static('dist/client'));
app.get('/*', (req, res) => {
  res.sendFile(`index.html`, {root: `dist/client/`});
});

app.listen(8080||process.env.PORT , () => console.log("listening"));
```

**controllers => authController**

```
const jwt = require('jsonwebtoken');
const User = require('../models/user');
const RegistrationCredentials = require('../models/registrationCredentials');
const {secret} = require('../configs/auth');



module.exports.register = (request, response) => {
    const {email, password, role} = request.body;
    const createdDate = new Date();
    const user = new User({email, createdDate});
    const regCredential = new RegistrationCredentials({email, password,role});
    regCredential.save()
        .then(() => {
```

```
            if (!email || !password || !role) {
                return  response.status(400).json({massage: "Bad request"});
            }
            user.save()
                .catch((err) => {
                    return  response.status(500).json({massage: "Server 2 error", err});
                });
            return  response.json({massage: 'Success'});
        })
        .catch((e) => {
            return  response.status(500).json({massage: "Server 1 error", e});
        });
}

module.exports.login = async (request, response) => {

    const {email, password} = request.body;
    let user;
     await User.findOne({email}).exec()
        .then(selectedUser =>{
            user = selectedUser;
        })
        .catch((err) => response.status(500).json({massage: err} ));

    await RegistrationCredentials.findOne({email, password}).exec()
```

```
        .then(selectedUser => {
            if (!selectedUser) {
                return response.status(400).json({massage: "Wrong email or password"});
            }

            let jwtObj = {
                regCred: {
                    _id : selectedUser._id,
                    role: selectedUser.role,
                    password: selectedUser.password,
                    email: selectedUser.email
                },
                user:{
                    _id: user._id,
                    createdDate: user.createdDate,
                    email: user.email
                }
            }
            return  response.status(200).json({massage: 'success', token:
jwt.sign(JSON.stringify(jwtObj), secret)});
        })
        .catch((err) => {
            return  response.status(500).json({massage: err + "SSS"});
        });
}
```

**middleware => authMiddleware**

```
const jwt = require('jsonwebtoken');
const { secret } = require('../configs/auth');

module.exports = (request, response, next) => {

  const authHeader = request.headers['authorization'];

    if(!authHeader) {
        return response.status(400).json({status: 'No authorization header found'});
    }
    const [, jwtToken] = authHeader.split(' ');
    try {
        request.token = jwt.verify(jwtToken, secret);
        next();
    } catch (err) {
        return response.status(400).json({status: 'Invalid JWT'});
    }
};
```

**models => registrationCredentials**

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
module.exports = mongoose.model('registrationCredentials', new Schema({
    email: {
        required: true,
```

```
        type: String,
        unique: true
    },
    password: {
        required: true,
        type: String
    },
    role: {
        required: true,
        type: String,
        enum: [ "SHIPPER", "DRIVER" ]
    }
}));
```

**routers=>authRouter**

```
const express = require('express');
const router = express.Router();

const { register, login } = require("../controllers/authController");

router.post('/auth/register', register);
router.post('/auth/login', login);


module.exports = router;
```

**ДОДАТОК Б**

**Лістинг Front-end частини**

**app.module.ts**

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { LoginAndRegistrationComponent } from './login-and-
registration/login-and-registration.component';
import { BrowserAnimationsModule } from '@angular/platform-
browser/animations';
import {MatTabsModule} from '@angular/material/tabs';

import {FormsModule, ReactiveFormsModule} from '@angular/forms';
import {MatFormFieldModule} from '@angular/material/form-field';
import {MatInputModule} from '@angular/material/input';
import {MatButtonModule} from '@angular/material/button';
import {MatSelectModule} from '@angular/material/select';
import { HttpClientModule } from '@angular/common/http';
import { HomePageComponent } from './home-page/home-page.component';
import {MatTooltipModule} from '@angular/material/tooltip';
import {MatSnackBarModule} from '@angular/material/snack-bar';
import {MatDialogModule} from '@angular/material/dialog';
import { LoadDialogComponent } from './load-dialog/load-dialog.component';
import { TruckStatusPipe } from './truck-status.pipe';
import { UpdateTruckComponent } from './update-truck/update-
truck.component';
import { LoadPipe } from './load.pipe';
@NgModule({
  declarations: [
```

```
    AppComponent,
    LoginAndRegistrationComponent,
    HomePageComponent,
    LoadDialogComponent,
    TruckStatusPipe,
    UpdateTruckComponent,
    LoadPipe
  ],
    imports: [
      BrowserModule,
      BrowserAnimationsModule,
      MatTabsModule,
      ReactiveFormsModule,
      MatFormFieldModule,
      MatInputModule,
      MatButtonModule,
      MatSelectModule,
      HttpClientModule,
      MatTooltipModule,
      MatSnackBarModule,
      MatDialogModule,
      FormsModule
    ],
   providers: [],
   bootstrap: [AppComponent]
  })
export class AppModule { }
```

**home-page.component.css**

```
  .header {
   display: flex;
```

```
  justify-content: space-between;
  background: black;
  color: white;
  height: 100px;
}
.logo-info{
  padding: 0 20px;
  display: flex;
  align-items: center;
  justify-content: center;
  flex-direction: column;
}
.logo{
  height: 50px;
}
.user-info{
  padding: 0 20px;
  margin: 0;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

h1, h2, h3, h4 {
  margin: 0;
}
.welcome{
  padding-top: 20px;
  font-size: 40px;
```

```
  text-align: center;
}
.form{
  padding: 40px;
  border: black 2px solid;
  border-radius: 20px;
  background: #ebeef0;
  margin: 20px;
}
.load-form {
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}
.full-width {
 width: 100%;
}
.loads {
  display: flex;
  flex-wrap: wrap;
  align-items: center;
  justify-content: center;
}
.load-item {
  width: 300px;
  padding: 20px;
  margin: 20px;
  background: #ebeef0;
  border: black 2px solid;
```

```
  border-radius: 20px;
}
.my-loads-header{
  display: flex;
  justify-content: center;
  font-size: 40px;
  padding: 20px;
}
.load-name {
  text-align: center;
}
.load-dimensions {
  padding: 20px;
}
.my-loads-header__refresh{
  height: 50px;
}
.update-load-button {
  position: absolute;
  top: 10px;
  right: 50px;
}
.truck{
  width: 400px;
  padding: 20px;
  margin: 20px;
  background: #ebeef0;
  border:black 2px solid;
  border-radius: 20px;
}
```

```
.trucks {
  display: flex;
  flex-wrap: wrap;
  align-items: center;
  justify-content: center;
}
.create-truck{
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 400px;
  height: 130px;
  padding: 20px;
  margin: 20px;
  background: #ebeef0;
  border: black 2px solid;
  border-radius: 20px;
}

.create-truck:hover {
  cursor: pointer;
}

.add-truck-icon {
  width: 125px;
}
.no-load {
  text-align: center;
}
```

```
.no-load-button{
  margin-top: 20px;
}

button {
  margin: 3px;
}

.black-button {
  color: white;
  background-color: black;
}
.red-button {
  color: red;
  background-color: black;
}
```

**home-page.component.html**

```
<header class="header">
  <div class="logo-info">
    <img src="assets/Logo.png" class="logo" alt="logo">
    <h2>FoxiDrivers</h2>
  </div>
  <div class="user-info">
    <h3>Welcome</h3>
    <h3 class="user-name">
      {{user?.email}}
    </h3>
    <h3>Hou logged in as {{user?.role}}</h3>
  </div>
</header>
```

```
<mat-tab-group >
  <mat-tab label="About Me">
    <h1 class="welcome">Welcome</h1>
    <form class="form" action="">
      <h2 class="form-label">Info about user</h2>
      <h3 class="form-field">Email: {{user?.email}}</h3>
      <h3 class="form-field">Created date: {{user?.date | date}}</h3>
      <h3>Register as: {{user?.role}}</h3>
    </form>

    <form class="form" action="">
      <h2>You can exit from this profile</h2>
      <h3>Just press the button :)</h3>
      <button mat-raised-button (click)="exit()">Sign Out</button>
    </form>

    <form class="form" action="">
      <h2>You can delete this profile</h2>
      <button mat-raised-button  matTooltip="Are you sure about this?"
(click)="deleteUser()">Delete me
      </button>
    </form>


  </mat-tab>
  <mat-tab *ngIf="user?.role ==='DRIVER'" label="My Trucks">
    <section class="trucks">
      <div *ngFor="let truck of userTrucks" class="truck">
        <h2>{{ truck?.truck_name }}</h2>
```

```
        <h3>Truck type: {{ truck.type }}</h3>
        <h3>Created date: {{ truck.created_date | date }}</h3>
        <h3>Truck Status: {{ truck.status | truckStatus }}</h3>

        <button mat-raised-button (click)="assignTruck(truck._id)"
[disabled]="truck.status !== 'OS'">
          Assign
        </button>
        <button mat-raised-button (click)="unAssignTruck(truck._id)"
[disabled]="truck.status !=='IS'">
          Unassigned
        </button>
        <button mat-raised-button (click)="updateTruck(truck._id, truck.type)"
              [disabled]="truck.status !=='OS'">Change Type
        </button>
        <button mat-raised-button class="red-button"
(click)="deleteTruck(truck._id)" [disabled]="truck.status !=='OS'">
          Delete
        </button>
      </div>
      <div (click)="createTruck()" class="create-truck">
        <img class="add-truck-icon"
            src="https://icons-for-
free.com/iconfiles/png/512/circle+more+plus+icon-
1320183136549593898.png" alt="">
        <h2>Add new truck</h2>
      </div>
    </section>
  </mat-tab>
  <mat-tab *ngIf="user?.role ==='DRIVER'" label="Active Loads">
```

```
<div class="my-loads-header">
  <h1 class="title">Active and New loads</h1>
  <button mat-raised-button class="update-load-button">
    <img class="my-loads-header__refresh"
        (click)="update()"
        src="http://simpleicon.com/wp-content/uploads/refresh.png"
        alt=""
        matTooltip="Update my loads"
        matTooltipPosition="left"
    >
  </button>

</div>
<section *ngIf="( userLoads| load : 'ACTIVE').length === 0" class="no-load">
  <h1>You don`t have any active loads</h1>
  <h1>For newest information press update button</h1>
  <button mat-raised-button color="basic" class="no-load-button">
    <img class="my-loads-header__refresh "
        (click)="update()"
        src="http://simpleicon.com/wp-content/uploads/refresh.png"
        alt=""
        matTooltip="Update my loads"
        matTooltipPosition="left"
    >
  </button>
</section>
<section *ngIf="userLoads" class="loads">
  <div *ngFor="let load of userLoads| load : 'ACTIVE'" class="load-item">
    <h2 class="load-name">{{load?.name}}</h2>
```

```
        <h3>Created date: {{load?.created_date | date }}</h3>
        <h3>Pickup address: {{load?.pickup_address}}</h3>
        <h3>Delivery address: {{load?.delivery_address}}</h3>
        <h3>Payload: {{load?.payload}}</h3>
        <h3>Load state: {{load?.state}}</h3>
        <div class="load-dimensions">
          <h2>Load dimensions:</h2>
          <h4>Width: {{load.dimensions.width}}</h4>
          <h4>Length: {{load.dimensions.length}}</h4>
          <h4>Height: {{load.dimensions.height}}</h4>
        </div>

        <button mat-raised-button color="primary"
(click)="activeLoadChangeState(load._id)"
                [disabled]="load.state === 'Arrived to delivery'">Change to next
state
        </button>
      </div>
    </section>
  </mat-tab>
  <mat-tab *ngIf="user?.role ==='SHIPPER'" label="Create new Load">
    <form class="load-form form" autocomplete="off"
[formGroup]="loadForm">
      <h2>Create Load</h2>
      <mat-form-field  class="full-width form-field">
        <mat-label>Name of load</mat-label>
        <input type="text" matInput formControlName="name"
onfocus="this.setAttribute('autocomplete', 'none');">
        <mat-error *ngIf="loadForm.controls.name.errors">
          *required field
```

```
        </mat-error>
      </mat-form-field>
      <mat-form-field class="full-width form-field">
        <mat-label>Payload of load</mat-label>
        <input type="number" matInput formControlName="payload">
        <mat-error *ngIf="loadForm.controls.payload.errors">
          *required field
        </mat-error>
      </mat-form-field>
      <mat-form-field class="full-width form-field">
        <mat-label>Enter the pickup address</mat-label>
        <input type="text" matInput formControlName="pickup_address"
onfocus="this.setAttribute('autocomplete', 'none');">
        <mat-error *ngIf="loadForm.controls.pickup_address.errors">
          *required field
        </mat-error>
      </mat-form-field>
      <mat-form-field class="full-width form-field">
        <mat-label>Enter the delivery address</mat-label>
        <input type="text" matInput formControlName="delivery_address"
onfocus="this.setAttribute('autocomplete', 'none');">
        <mat-error *ngIf="loadForm.controls.delivery_address.errors">
          *required field
        </mat-error>
      </mat-form-field>
      <div class="dimensions">
        <h2>Enter the load dimensions</h2>
        <mat-form-field class="full-width form-field">
          <mat-label>Width</mat-label>
          <input type="text" matInput formControlName="width">
```

```
          <mat-error *ngIf="loadForm.controls.width.errors">
            *required field
          </mat-error>
        </mat-form-field>
        <mat-form-field class="full-width form-field">
          <mat-label>Length</mat-label>
          <input type="text" matInput formControlName="length"
onfocus="this.setAttribute('autocomplete', 'none');">
          <mat-error *ngIf="loadForm.controls.length.errors">
            *required field
          </mat-error>
        </mat-form-field>
        <mat-form-field class="full-width form-field">
          <mat-label>height</mat-label>
          <input type="text" matInput formControlName="height">
          <mat-error *ngIf="loadForm.controls.height.errors">
            *required field
          </mat-error>
        </mat-form-field>
      </div>
      <div class="load-form-buttons">
        <button mat-raised-button (click)="saveLoad()" class="black-button"
>Save the load</button>
        <button mat-raised-button (click)="saveAndPostLoad()" >Save and
post</button>
      </div>
    </form>
  </mat-tab>
  <mat-tab *ngIf="user?.role ==='SHIPPER'" label="My Loads">
    <div class="my-loads-header">
```

```
  <h1></h1>
  <h1>Active and New loads</h1>
  <button mat-raised-button color="basic" class="update-load-button">
    <img class="my-loads-header__refresh"
        (click)="updateLoads()"
        src="http://simpleicon.com/wp-content/uploads/refresh.png"
        alt=""
        matTooltip="Update my loads"
        matTooltipPosition="left"
    >
  </button>

</div>

<section class="loads">
  <div *ngFor="let load of userLoads | load : 'ACTIVE' " class="load-item">
    <h2 class="load-name">{{load.name}}</h2>
    <h3>Created date: {{load.created_date | date }}</h3>
    <h3>Pickup address: {{load.pickup_address}}</h3>
    <h3>Delivery address: {{load.delivery_address}}</h3>
    <h3>Payload: {{load.payload}}</h3>
    <h3>Load state: {{load.state}}</h3>
    <h3>Load status: {{load.status}}</h3>
    <div class="load-dimensions">
      <h2>Load dimensions:</h2>
      <h4>Width: {{load.dimensions.width}}</h4>
      <h4>Length: {{load.dimensions.length}}</h4>
      <h4>Height: {{load.dimensions.height}}</h4>
    </div>
```

```
        <button mat-raised-button color="primary"
(click)="MyLoadsPostLoad(load._id)"
                [disabled]="load.status !== 'NEW'">POST LOAD
        </button>
        <button mat-raised-button color="primary" (click)="editLoad(load)"
[disabled]="load.status !=='NEW'">Edit LOAD
        </button>
        <button mat-raised-button color="warn" (click)="deleteLoad(load._id)"
[disabled]="load.status !=='NEW'">Delete
        </button>
      </div>
    </section>
  </mat-tab>
  <mat-tab label="History">
    <div class="my-loads-header">
      <h1></h1>
      <h1>Loads history</h1>
      <button mat-raised-button color="basic" class="update-load-button">
        <img class="my-loads-header__refresh"
            (click)="updateLoads()"
            src="http://simpleicon.com/wp-content/uploads/refresh.png"
            alt=""
            matTooltip="Update my loads"
            matTooltipPosition="left"
        >
      </button>

    </div>

    <section class="loads">
```

```
        <div *ngFor="let load of userLoads| load : 'HISTORY'" class="load-
      item">
          <h2 class="load-name">{{load.name}}</h2>
          <h3>Created date: {{load.created_date | date }}</h3>
          <h3>Pickup address: {{load.pickup_address}}</h3>
          <h3>Delivery address: {{load.delivery_address}}</h3>
          <h3>Payload: {{load.payload}}</h3>
          <h3>Load state: {{load.state}}</h3>
          <h3>Load status: {{load.status}}</h3>
          <div class="load-dimensions">
            <h2>Load dimensions:</h2>
            <h4>Width: {{load.dimensions.width}}</h4>
            <h4>Length: {{load.dimensions.length}}</h4>
            <h4>Height: {{load.dimensions.height}}</h4>
          </div>

        </div>
      </section>
    </mat-tab>
  </mat-tab-group>
```

**home-page.component.ts**

```
  import {Component, OnDestroy, OnInit} from '@angular/core';
  import {UserServiceService} from '../Services/user-service.service';
  import {AuthServiseService} from '../Services/auth-servise.service';
  import {LoadServiceService} from '../Services/load-service.service';
  import {FormBuilder, Validators} from '@angular/forms';
  import {map} from 'rxjs/operators';
  import {MatSnackBar} from '@angular/material/snack-bar';
  import {MatDialog} from '@angular/material/dialog';
  import {LoadDialogComponent} from '../load-dialog/load-dialog.component';
```

```
import {TruckServiceService} from '../Services/truck-service.service';
import {UpdateTruckComponent} from '../update-truck/update-
truck.component';

const DELAY_TIME = 1000;

@Component({
  selector: 'app-home-page',
  templateUrl: './home-page.component.html',
  styleUrls: ['./home-page.component.css']
})
export class HomePageComponent implements OnInit, OnDestroy {

  user;
  userLoads;
  loadForm;
  userTrucks;
  cache;
  constructor(private userService: UserServiceService,
              private authService: AuthServiseService,
              private loadService: LoadServiceService,
              private truckService: TruckServiceService,
              private fb: FormBuilder,
              private snackBar: MatSnackBar,
              public dialog: MatDialog) {

  }

  ngOnDestroy(): void {
    clearInterval(this.cache);
```

```
}

ngOnInit(): void {
  this.cache = setInterval(() => this.update(), DELAY_TIME);
  this.loadForm = this.fb.group({
    name: ['', Validators.required],
    payload: ['', Validators.required],
    pickup_address: ['', Validators.required],
    delivery_address: ['', Validators.required],
    width: ['', Validators.required],
    length: ['', Validators.required],
    height: ['', Validators.required],
  });

  this.userService.getUser().pipe(
    map((user: any) => {
      this.user = user.res;
      if (user.res.role === 'DRIVER') {
        this.truckService.getMyTrucks().subscribe(res => {
          this.userTrucks = res.trucks;
        });
      }
      this.loadService.getAllLoads().subscribe(res => {
        this.userLoads = res.loads;
      });
    })
  ).subscribe(res => {
    this.user = res.res;
  });
}
```

```
  deleteUser(): any {
    this.userService.deleteUser();
  }

  exit(): any {
    this.authService.signOut();
  }

  saveLoad(): any {
    const value = this.loadForm.value;
    const load = {
      name: value.name,
      payload: value.payload,
      pickup_address: value.pickup_address,
      delivery_address: value.delivery_address,
      dimensions: {
        width: value.width,
        height: value.height,
        length: value.length,
      }
    };
    this.loadService.postLoad(load).subscribe(res => {
      if (res.massage === 'Load created successfully') {
        this.openSnackBar(res.massage + ' You can check load status in the next
tab');
      } else {
        this.openSnackBar(res.massage + ' Please try one more time');
      }
    });
```

```
    this.updateLoads();
  }

  saveAndPostLoad(): void {
    const value = this.loadForm.value;
    const load = {
      name: value.name,
      payload: value.payload,
      pickup_address: value.pickup_address,
      delivery_address: value.delivery_address,
      dimensions: {
        width: value.width,
        height: value.height,
        length: value.length,
      }
    };
    this.loadService.postLoad(load).pipe(
      map((res: any) => {
        if (res.massage === 'Load created successfully') {
          this.loadService.getAllLoads().subscribe((loads) => {
            const currentLoad = loads.loads.find(l => l.name === load.name);
            this.loadService.postLoadForId(currentLoad._id).subscribe(loadForId
=> {
              if (loadForId.driver_found) {
                this.openSnackBar('Load posted successfully and we find the driver
for you');
              } else {
                this.openSnackBar('We can`t find driver for this load, you can try
later from the next tab');
              }
```

```
          });
        });
      }
      return res;
    })
  ).subscribe(res => {
    this.updateLoads();
    console.log(res);
  });
 }

 openSnackBar(message: string): void {
  this.snackBar.open(message, '', {
    duration: 4000,
  });
 }

 MyLoadsPostLoad(id: string): void {
  this.loadService.postLoadForId(id).subscribe(res => {
    console.log(res);
    if (res.driver_found) {
      this.openSnackBar('Load posted successfully and we find the driver for
you');
    } else {
      this.openSnackBar('We can`t find driver for this load, please try later');
    }
    this.updateLoads();
  });
 }
```

```
editLoad(load): void {
  const dialogRef = this.dialog.open(LoadDialogComponent, {
    width: '800px',
    data: {load}
  });

  dialogRef.afterClosed().subscribe(() => {
    this.updateLoads();
  });
}

updateLoads(): void {
  this.loadService.getAllLoads().subscribe(res => {
    this.userLoads = res.loads;
  });
}

deleteLoad(id): void {
  this.loadService.deleteLoadForId(id).subscribe(res => {
    this.openSnackBar(res.massage);
    this.updateLoads();
  });
  this.updateLoads();
}

updateTrucks(): void {
  this.truckService.getMyTrucks().subscribe(res => {
    this.userTrucks = res.trucks;
  });
}
```

```
assignTruck(id): void {
  this.truckService.assignTruck(id)
    .subscribe(res => {
      this.openSnackBar(res.massage);
      this.updateTrucks();
    });
}

unAssignTruck(id): void {
  this.truckService.unAssignTruck(id)
    .subscribe(res => {
      this.openSnackBar(res.massage);
      this.updateTrucks();
    });
}

updateTruck(id, currentType): void {
  const dialogRef = this.dialog.open(UpdateTruckComponent, {
    width: '800px',
    data: {currentType, id}
  });

  dialogRef.afterClosed().subscribe(() => {
    this.updateTrucks();
  });

}

deleteTruck(id): void {
```

```
  this.truckService.deleteTruck(id).subscribe(res => {
    this.openSnackBar(res.massage);
    this.updateTrucks();
  });
}

createTruck(): void {
  const dialogRef = this.dialog.open(UpdateTruckComponent, {
    width: '800px',
    data: {}
  });

  dialogRef.afterClosed().subscribe(() => {
    this.updateTrucks();
  });

}
activeLoadChangeState(id): void {
  this.loadService.changeState(id)
    .subscribe(res => {
      this.openSnackBar(res.massage);
      this.update();
    });

}
update(): void {
  if(this.user.role === 'DRIVER') {
    this.updateTrucks();
  }
  this.updateLoads();
```

```
    }
}
```

**auth-service.service.ts**

```
import {Injectable} from '@angular/core';
import {environment} from '../../environments/environment';
import {HttpClient} from '@angular/common/http';
import {BehaviorSubject, } from 'rxjs';


@Injectable({
  providedIn: 'root',
})

export class AuthServiseService {
  IsLogged ;
  serverPath;
  UserToken;
  constructor(private http: HttpClient) {
    this.serverPath = environment.serverPath;
    if (localStorage.getItem('token') !== null){
      this.IsLogged = new BehaviorSubject<boolean>(true);
      this.UserToken = new BehaviorSubject(localStorage.getItem('token'));
    }
    else {
      this.IsLogged = new BehaviorSubject<boolean>(false);
      this.UserToken = new BehaviorSubject(null);
    }
  }

  login(email: string, password: string): void {
```

```
    this.http.post(`${this.serverPath}/api/auth/login`, {email, password})
      .subscribe((res: any) => {
        if (res.token) {
          this.IsLogged.next(true);
          localStorage.setItem('token', res.token);
          this.UserToken.next(res.token);
          console.log('login success');
        }
      });
  }

  register(email: string, password: string, role: string): void {
    this.http.post(`${this.serverPath}/api/auth/register`,
      {email, password, role})
      .subscribe((res: any) => {
        if (res.massage === 'Success') {
          console.log('registration success');
          this.login(email, password);
        }
      });
  }

  signOut(): void {
    this.IsLogged.next(false);
    localStorage.removeItem('token');
    this.UserToken.next(null);
  }
}
```