**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)
червня 2023 р.

# КВАЛІФІКАЦІЙНА РОБОТА
## на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційне і програмне забезпечення системи керування контентом інтернет-магазину. Клієнтська частина»
здобувача групи ІН – 94-1 Крикуненко Михайло Володимирович

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Михайло КРИКУНЕНКО
(підпис)

Керівник,
Кандидат наук, доцент Ігор ШЕЛЕХОВ _____
(підпис)

**Суми – 2023**

**Сумський державний університет**
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)

# ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

**на здобуття освітнього ступеня бакалавра**

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-94-1 Крикуненка Михайла Володимировича

1. Тема роботи: «Інформаційне і програмне забезпечення системи керування контентом інтернет-магазину. Клієнтська частина».

затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*

3. Вхідні дані до кваліфікаційної роботи _____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

*1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, які використовуються для керування контентом інтернет-магазинів 3) Розробка інформаційної системи керування контентом інтернет-магазину. 4) Аналіз результатів.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____

6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис)

Керівник _____ (підпис)

## КАЛЕНДАРНИЙ ПЛАН

| п/п | Назва етапів кваліфікаційної роботи | Термін виконання | Примітка |
|---|---|---|---|
| | *Аналіз проблеми предметної області, постановка й формування завдань дослідження* | | |
| | *Огляд технологій, що використовуються для керування контентом інтернет-магазинів* | | |
| | *Розробка інформаційної системи керування контентом інтернет-магазину* | | |
| | *Аналіз отриманих результатів* | | |
| | *Оформлення пояснювальної записки до кваліфікаційної роботи* | | |

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

# АНОТАЦІЯ

**Записка**: 50 стр., 26 рис., 14 додаток, 20 використаних джерел.

**Обґрунтування актуальності теми роботи** – тема кваліфікаційної роботи є актуальною, оскільки присвячена розв’язанню важливої практичної задачі адміністрування та управління інтернет-магазином.

**Об’єкт дослідження** — процес управління контентом інтернет-магазину.

**Мета роботи** — розробка інформаційної системи управління контентом інтернет-магазину.

**Методи дослідження** — алгоритми аналізу та обробки даних.

**Результати** — розроблено інформаційну систему, для роботи з даними про покупців, їх закази, формує на основі цього статистику.

ІНФОРМАЦІЙНЕ І ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ КЕРУВАННЯ КОНТЕНТОМ ІНТЕРНЕТ-МАГАЗИНУ. КЛІЄНТСЬКА ЧАСТИНА.

# ЗМІСТ

# ВСТУП

Цифрова промисловість швидко розвивається і тенденції у ній змінюються майже щомісяця. Давати тривалі і хоч трохи детальні прогнози щодо неї можна порівняти із прогнозами погоди. І все-таки є основні моменти, які залишатимуться незмінними. До одного з таких відноситься розуміння, що для реалізації продажів за допомогою сайту недостатньо просто зайняти перший рядок у пошуку або забезпечити рясний трафік в інший спосіб. Також необхідно, щоб ресурс, на який залучаються потенційні клієнти, був не абстрактним каркасом з тексту і кнопок, а продуманим з маркетингової точки зору і технологічно опрацьованим веб-рішенням, що володіє високим коефіцієнтом конверсії.

Візуальна складова будь-якого бізнесу стає надзвичайно важливою. Відповідно до спеціального дослідження Forrester, кожен 1$, який бізнес інвестує в дизайн інтерфейсу користувача, повертає 100$. Тобто прибутковість такого роду інвестування становить 9900% [1].

Ігнорування розвитку та тенденцій в області UI-інтерфейсу неминуче призводить до падіння ефективності сайту або мобільного додатку, що спричиняє суттєві втрати прибутку, а в нинішніх умовах часом призводить до загибелі бізнесу.

Розробка web-додатків на даний час є одним із перспективних напрямків діяльності для багатьох компаній, зайнятих у сфері високотехнологічних цифрових та комп'ютерних технологій. Web-додаток є прикладним програмним забезпеченням, логіка якого розподілена між сервером та клієнтом, а обмін інформацією відбувається через мережу. Клієнтська частина реалізує інтерфейс користувача, а серверна – отримує та обробляє запити від клієнта, виконує обчислення, формує веб-сторінку та відправляє її клієнту згідно з протоколом HTTP.

Даний вид додатків має ряд особливостей, які впливають на процеси їх функціонування, при розробці та підтримці:

– відкриті для тестування з віддалених комп'ютерів, що є оптимальним для застосування гнучкої методології розробки;

– виконуються незалежно від операційної системи клієнта. При цьому на нього накладається вимога кросбраузерності, що випливає з різної реалізації браузерами стандартів HTML, CSS та DOM;

– є розподіленою інформаційною системою і мають витримувати максимальну кількість звернень користувача.

Web-додаток, створений із застосуванням сучасних методів розробки, являє собою інформаційний ресурс, завдяки якому можна здійснювати:

– передачу будь-якої інформації про компанію, а також потік новин для користувачів;

– пряма взаємодія з користувачем та його інформаційну підтримку;

– рекламу, так як Web-додаток здатний об'єднати в собі відео-рекламу та банери.

Метою роботи є розробка універсальної CMS-системи для Web-магазину з допомогою фреймворку Angular.

Актуальність роботи полягає в тому що на даний момент, в мережі інтернет немає актуальних готових рішень для запуску інтернет-магазинів, ці рішення є або застарілими з використанням застарілих технологій таких як jQuery, PHP, CMS WordPress, Joomla, Drupal, або є досить дорогими такі, як Tilda, Weblium, Shop-Express.

Для розробки веб-системи необхідно вирішити такі завдання:

– Провести огляд існуючих інструментів та інформаційних технологій для створення веб-сайтів;

– Визначити структуру та дизайн веб-сайтів, що розробляється;
– Реалізувати основний функціонал клієнтської частини;
– Виконати адаптацію веб-сайту під усі популярні пристрої;
– Провести функціональне тестування веб-сайту.

# 1 ІНФОРМАЦІЙНИЙ ОГЛЯД

## 1.1 Базові інструменти розробки клієнтської частини веб-додатків

Базовою частиною веб-сторінки є HTML. Мова гіпертекстової розмітки HTML є стандартною мовою розмітки документів у браузері. Веб-браузери отримують HTML-документи з веб-сервера або локального сховища і перетворюють їх на мультимедійні веб-сторінки. HTML описує структуру веб-сторінки семантично та спочатку включає підказки для зовнішнього вигляду документа [2].

Елементи HTML – це будівельні блоки HTML-сторінок. За допомогою конструкцій HTML зображення та інші об'єкти, такі як інтерактивні форми, можуть бути вбудовані в сторінку, що відображається. HTML надає засоби для створення структурованих документів, визначаючи структурну семантику для тексту, як-от заголовки, абзаци, списки, посилання, цитати та інші елементи [3]. HTML-елементи виділяються тегами, записаними з використанням кутових дужок. Такі теги, як <img /> і <input />, безпосередньо вводять контент на сторінку. Інші теги, такі як <p>, оточують та надають інформацію про текст документа і можуть включати інші теги як піделементи. Браузери не відображають HTML-теги, але використовують їх для інтерпретації вмісту сторінки [4].

HTML може вбудовувати програми, написані мовою сценаріїв, такою як JavaScript, який впливає на поведінку та вміст веб-сторінок. Увімкнення CSS визначає зовнішній вигляд та компонування контенту. Консорціум World Wide Web (W3C), колишній розробник HTML і нинішній розробник стандартів CSS, з 1997 заохочує використання CSS замість явного презентаційного HTML [5].

Каскадні таблиці стилів (CSS) – це мова таблиць стилів, що використовується для опису представлення документа, написаного мовою розмітки, такою як HTML [4].

CSS розроблено для розділення представлення та вмісту, включаючи макет, кольори та шрифти. Даний поділ здатний підвищити доступність контенту, впровадити гнучкість та контроль властивостей подання, дозволяє ряду веб-сторінок спільно використовувати форматування шляхом вказівки відповідного CSS в окремому файлі .css, що знижує складність та повторення структурного контенту, а також дозволяє створювати файл .css, який необхідно кешувати, щоб покращити швидкість завантаження сторінки між сторінками, які спільно використовують файл, та його форматування [6].

JavaScript, часто скорочено JS, є мовою програмування, що відповідає специфікації ECMAScript. JavaScript – це високорівнева, інтерпретована та мультипарадигмальна мова програмування. JavaScript є С-подібною об'єктно-орієнтованою мовою з динамічною типізацією [7].

Поряд з HTML та CSS, JavaScript є однією з основних технологій всесвітньої павутини. JavaScript дозволяє створювати інтерактивні веб-сторінки та є важливою частиною веб-застосунків [8]. Переважна більшість веб-сайтів використовують його для управління поведінкою сторінки на стороні клієнта, і всі основні веб-браузери мають спеціальний механізм JavaScript для виконання [7].

Як мова з кількома парадигмами, JavaScript підтримує керовані подіями, функціональні та імперативні стилі програмування. Він має інтерфейси прикладного програмування (API) для роботи з текстом, датами, регулярними виразами, стандартними структурами даних та об'єктною моделлю документа (DOM) [8].

Спочатку JavaScript використовувалися тільки в веб-браузерах, але тепер вони вбудовані в деякі сервери, зазвичай через Node.js. Вони також вбудовані в різні програми, створені за допомогою таких фреймворків, як Electron та Cordova [9].

На мові JavaScript написано безліч програм для різних сфер та потреб. Для полегшення написання додатків цією мовою існують різноманітні бібліотеки та фреймворки. Фреймворки дозволяють розробнику абстрагуватися від рутинних та довгих операцій та сконцентруватися на розумінні та розробці логіки програми. Однак вибір фреймворку крім очевидних переваг накладає індивідуальні обмеження на розробку і функціонал програми, що розробляється. Тому важливим є вибір відповідного фреймворку для програми, що розробляється [10].

### 1.2 Огляд існуючих рішень

Для того щоб створити конкурентний програмний продукт насамперед необхідно проаналізувати існуючі рішення. В даному огляді розглядаються програми, що дозволяють автоматизувати процес створення інтернет-магазину, а також дозволяють створювати систему управління замовленнями, товарами, клієнтами, співробітниками та торговими точками компанії.

Конструктор WIX.COM. Першим рішенням розглядається конструктор сайтів та різних інтернет-магазинів wix.com. Це один з найпопулярніших конструкторів, який працює з 2006 року. У даного конструктора є багато різних шаблонів красивий дизайн. Візуально робоча область виглядає зручно та зрозуміло. Найчастіше за допомогою даного конструктора створюють прості та красиві сайти-візитки. Основними мінусами цієї програми можна виділити такі моменти:

– через те, що конструктор має безліч шаблонів та вбудованих функцій, додаток не завжди працює швидко, особливо на непотужних комп'ютерах з повільним Інтернет-з'єднанням;

– громіздкий код створеного інтернет-магазину, а також відсутність можливості його редагування. Правда, це може бути плюсом, тому що недосвідчені користувачі можуть змінити код таким чином, що це призведе до непереборних помилок;

– незручний та некрасивий домен, на якому створюється інтернет-магазин. Усі створені за допомогою даного конструктора сайти мають безкоштовний домен наступного виду: *.wix.com/назва ресурсу, де *- логін користувача, який створив сайт;

– незважаючи на простоту в «складанні» інтернет-магазинів, необхідно витратити багато часу на його комплектацію, а також створення форм реєстрації та внутрішньої системи замовлень;

– створені інтернет-магазини мають відмітку про те, що сайт створено з допомогою даного конструктора, прибрати яку можна тільки придбавши один із тарифних планів.

Конструктор NETHOUSE. Наступним інструментом розглядається один із найкращих конструкторів інтернет-магазинів nethouse.ua. Цей конструктор легкий у освоєнні, а також має близько сотні різних шаблонів. Одним з плюсів цього конструктора є можливість підключення сторонніх систем обліку товарів та замовлень. Мінусом даної системи можна виділити складність додавання в систему та налаштування взаємодії співробітників та торгових точок компанії. Даний конструктор є одним із найкращих в огляді.

Конструктор Ukit/ Додаток для створення інтернет-магазинів Ukit є новим на ринку. Він є популярним серед підприємців, власників малого та середнього бізнесу. Він є досить універсальним конструктором, за допомогою якого можна

зробити сайт-візитку, блог, форум, а також потрібний інтернет-магазин. У даного конструктора потужний функціонал, але для його використання потрібен високий професійний рівень. Як плюс у створенні інтернет-магазину на даній платформі можна виділити зручний домен. Основними недоліками даної програми можна виділити наступне:

– довгий процес реєстрації та створення інтернет-магазину, необхідно пройти безліч етапів підтвердження особистих даних перед безпосереднім налаштуванням та комплектацією інтернет-магазину;

– складний інтерфейс, на ознайомлення з яким новачкам потрібно час;

– потрібна велика кількість часу на комплектацію магазину та настроювання системи управління замовленнями, співробітниками та торговими точками.

Наступним аналізованим додатком є Umi. Це проста і легка в освоєнні система, що володіє великим функціоналом, в тому числі прив'язкою до систем управління складом і 1С. Конструктор має багато готових шаблонів, які не дуже складно редагувати. Крім цього, система досить популярна, що підтверджується більшою кількістю сайтів, створених за допомогою даної платформи (близько 1,5 млн.). Мінусами даної системи можна назвати таке:

– за весь функціонал необхідно платити;

– потрібен тривалий час для створення магазину, а також підключення сторонніх систем;

– довгий процес реєстрації адміністратора майбутнього сайту в системі;

– вартість мінімальної оплати для роботи сайту складає 550$ на місяць.

Останнім конструктором, що розглядається, є Tilda. З її допомогою можна створювати сайти компанії, різні візитки та невеликі інтернет-магазини. Усі дії відбуваються у внутрішньому візуальному редакторі, але не для професіонала, адже буде складно освоїтися зі створенням блоків та налаштування їхньої взаємодії.

Головним плюсом Tilda є дизайн розроблених сайтів, бо усі шаблони розроблені професійними дизайнерами. Мінусами даної системи є таке:

– складність створення інтернет-магазину та системи управління замовленнями, товарами та співробітниками;

– висока вартість тарифного плану, який необхідний у зв'язку з тим, що базовий дозволяє створювати лише кілька сторінок;

– націленість програми на створення сайтів-візиток;

– потрібна велика кількість часу для оволодіння даною системою, а також для створення інтернет-магазину.

Нижче представлена таблиця 1.1, що містить результати порівняльного аналізу програм-конструкторів. Під час їхнього порівняння для всіх параметрів, крім «Кількість етапів, які необхідно пройти при реєстрації нового магазину», запроваджено умовні оцінки – лінгвістичні змінні, що градуюються на числовій осі [11]. Так, числу 1 відповідає змінна «дуже погано», 2 – «погано», 3 – «нормально», 4 - «добре», 5 – «відмінно». Оцінка * у параметрі «Наявність внутрішньої системи для підтримки замовлень, користувачів та товарів» означає, що така система відсутня, але може бути інтегрована стороння.

Таблиця 1.1 – Порівняння існуючих програм

| **Параметр** | **Додаток, що розглядається** | | | | |
|---|---|---|---|---|---|
| | WIX | Nethouse | Ukit | Umi | Tilda |
| Швидкість роботи програми | 2 | 5 | 5 | 4 | 4 |
| Кількість етапів, яке необхідно пройти за реєстрації нового магазину | 3 | 2 | 4 | 4 | 3 |
| Вибір безкоштовного домену | 3 | 5 | 5 | 5 | 5 |
| Кінцева ціна | 3 | 4 | 4 | 3 | 2 |
| Кількість та якість шаблонів для створення інтернет-магазинів | 5 | 5 | 5 | 5 | 5 |
| Наявність внутрішньої системи для підтримки замовлень, користувачів та товарів | 1 | * | 1 | * | 1 |
| Простота та швидкість створення елементарного інтернет-магазину | 4 | 5 | 5 | 5 | 3 |
| Простота та швидкість налаштування створеного інтернет-магазину | 4 | 5 | 4 | 4 | 3 |

Провівши аналіз існуючих програмних продуктів, можна висунути низку вимог до створення конкурентоспроможного веб-сайту. Веб-сайт повинен забезпечувати виконання таких умов:

– простота та швидкість створення нового інтернет-магазину, тобто він повинен створюватися максимально швидко за адресою, що задається;

– мінімальний час для реєстрації в системі. Користувач не повинен витрачати багато часу на верифікацію своїх даних, до того ж не всі користувачі хочуть ділитися своїми особистими даними, такими як, наприклад, телефон;

– наявність зручного, універсального, сучасного, інтуїтивно зрозумілого та адаптивного шаблону інтернет-магазину;

– мати автоматично створювані форми реєстрації для клієнтів та співробітників, а також особисті кабінети. Користувач не повинен розбиратися в методах створення авторизації та витрачати на це свій час;

– магазини, що створюються в системі, повинні розташовуватися на доменах другого рівня, це необхідно для краси та стислості адреси, на якому будуть розташовуватися магазини. До того ж, користувач повинен мати можливість вибору цієї адреси, для впізнаваності створеного магазину;

– кінцева ціна для користувача має бути максимально низькою;

– наявність інтуїтивно зрозумілого інтерфейсу для роботи з системою замовлень, товарів, торгових точок та співробітників;

– наявність зручної системи вивантаження товарів у систему, наприклад, одним файлом, де міститься вся необхідна інформація.

## 1.3 Огляд актуальних фреймворків для розробки клієнтської частини веб-додатків

У сфері програмування немає чіткого визначення, яким критеріям має відповідати програмне забезпечення щоб достовірно вважатися фреймворком. Іноді фреймворки плутають із бібліотеками [12].

Бібліотека зазвичай складається з попередньо написаного коду, класів, процедур, скриптів, конфігурацій та багато іншого. В основному бібліотеку можна легко інтегрувати в існуючий проект для скорочення часу розробки. Це пов'язано з тим, що багато питань, що стосуються основних алгоритмів та функцій, вже було вирішено іншими досвідченими програмістами. Бібліотека

економить час для розробників, що дозволяє зосередитися на проблемах, пов'язаних з логікою. Але використання сторонньої бібліотеки також може становити потенційний ризик для програми [13].

Порівняно з бібліотеками, фреймворки пропонують повний стек корисних функцій і беруть він відповідальність за архітектурні рішення у процесі розробки. Фреймворки можуть включати стратегії для маршрутизації URL-адрес у додатку, управління станом, об'єднання та інших. Крім того, фреймворки забезпечують покращення робочого процесу, які включають передові практики для основних аспектів розробки, таких як загальна структура програми або створення шаблонів [12].

### 1.3.1 Огляд фреймворку Angular

Angular - це повнофункціональний фреймворк, що охоплює більшу частину потреб розробників. З іншого боку, Angular може здатися складним щодо та використання фреймворку, оскільки потрібно освоїти великий набір взаємодіючих елементів (служби, використання залежностей тощо).

Angular містить усе, що може знадобитися для розробки великомасштабних інтерфейсів. Для порівняння, React покладається на плагіни, розроблені спільнотою (наприклад, підтримки маршрутизаторів). Зрештою розробник знаходиться всередині машини кодового мислення, яка хоче, щоб він відповідав її ідеалам і угодам. Більше того, великомасштабні проекти з великою кількістю членів команди можуть виграти від жорсткішого та чітко визначеного архітектурного стилю, що є присутнім у Angular.

Angular адаптував класи ECMAScript. Ці класи використовують вбудований стан як стан компонента. Вони прикрашені інструкціями, що визначають їх метадані. Уявлення в Angular схожі на уявлення в Vue в тому, що вони є простими HTML-шаблонами з додатковою прив'язкою даних і

підтримкою логіки за допомогою вбудованих директив. В той час, як AngularJS був побудований в основному на архітектурі Model-View-Controller (MVC), починаючи з версії 2 Angular вважається компонентним, що дуже схоже на MVC, але забезпечує більш високу можливість повторного використання компонентів у додатку. Це дозволяє створювати інтерфейси користувача з безліччю інтерактивних частин і в той же час спрощує процес розробки [13]. Основні переваги цієї архітектури - це можливість повторного використання. Компоненти аналогічної природи добре інкапсульовані, тобто самодостатні. Розробники можуть повторно використовувати їх у різних частинах програми. Це особливо корисно в корпоративних додатках, де різні системи сходяться воєдино, але можуть мати багато схожих елементів, таких як поля пошуку, засоби вибору дати, списки сортування, а також можуть краще читати код і зрештою швидше виходити на плато продуктивності. На додаток незалежний характер компонентів спрощує юніт-тести, процедури забезпечення якості, спрямовані на перевірку продуктивності найдрібніших частин програми (юнітів). Компоненти, які легко відокремлюються один від одного, можуть бути легко замінені найкращими реалізаціями себе.

Вбудований TypeScript: покращує інструментарій, і робить код більш чистим і масштабованим Angular написаний з використанням мови TypeScript, яка по суті є підмножиною JavaScript. Він повністю компілюється в JavaScript, але допомагає виявляти та усувати типові помилки під час кодування. У той час як невеликі проекти JavaScript не вимагають такого підходу, програми корпоративного масштабу вимагають від розробників робити свій код чистішим і частіше перевіряти його якість. Внесення політики Angular, заснованої на TypeScript, у розділі переваг, є спірним моментом для багатьох інженерів [13]. Скарги, пов'язані з TypeScript, постійно з'являються серед спільноти розробників. Розробникам доводиться вивчати ще одну мову.

Віктор Савкін, колишній розробник команди Google Angular, пояснює, що перехід від JavaScript до TypeScript виправданий інструментами для великих проектів корпоративного масштабу. TypeScript має покращені служби навігації, автозаповнення та рефакторингу. В даний час TypeScript вважається базовою для Angular, і для TypeScript.

Також необхідно згадати і RxJS. RxJS — це бібліотека, яка зазвичай використовується з Angular для обробки асинхронних викликів даних. Це дозволяє обробляти події незалежно паралельно та продовжувати виконання, не чекаючи настання якоїсь події та не залишаючи веб-сторінку без відповіді. В принципі, це працює як складальна лінія, де виконання розбите на окремі та взаємозамінні частини, а не прив'язане до однієї людини. Очевидно, що асинхронне програмування існувало і до RxJS, але ця бібліотека спростила багато речей. Хоча багато інженерів скаржаться на криву навчання RxJS. А також на повідомлення про помилки, які занадто загадкові, щоб їх можна було зрозуміти без додаткових досліджень, що супроводжуються маніпуляціями методом спроб та помилок. Бібліотека працює з Observables, свого роду схемами, які описують, як поєднуються потоки даних і як додаток реагує на змінні в цих потоках [18].

Філософія, яка залежить від платформи Angular була розроблена з урахуванням підходу, орієнтованого на мобільні пристрої. Ідея полягає в тому, щоб поділитися базою коду та, зрештою, набором інженерних навичок у веб-додатках із додатками для iOS та Android. Щоб реалізувати це амбітне позиціонування, у 2015 році розробники Angular співпрацювали з командою, яка стоїть за фреймворком NativeScript (який орієнтований на створення близьких до рідних мобільних додатків). Не тільки сам код, а й концепції Angular, такі як використання залежностей, прив'язка даних, служби та маршрутизація, аналогічні як для NativeScript, так і для Angular. Проте цей

агностицизм поширюється не так на повторне використання коду, але в той самий набір інженерних навичок. Іншими словами, розробники повинні використовувати компоненти інтерфейсу користувача NativeScript для створення мобільних інтерфейсів, але вони будуть працювати в знайомих середовищах JavaScript і Angular.

Основним фактором, що забезпечує продуктивність додатків на Angular відбувається за рахунок впровадження ієрархічних залежностей та підтримки Angular Universal. Використання ієрархічної залежності. Angular використовує покращену ієрархічну ін'єкцію залежностей порівняно з AngularJS. Цей метод відокремлює фактичні компоненти від своїх залежностей, виконуючи їх паралельно одне одному. Angular будує окреме дерево інжекторів залежностей, яке можна змінювати без переналаштування компонентів.

Кожному дереву компонентів призначено дерево інжекторів, які містять інформацію про залежність. Такий підхід забезпечує високу продуктивність програм Angular. Як стверджує команда Angular, Angular 2 був у 5 разів швидше, ніж Angular 1.x.

Angular Universal - це сервіс, який дозволяє відображати вид програм на сервері, а не в клієнтських браузерах. Google надає набір інструментів для попередньої візуалізації програми або її повторної візуалізації для кожного запиту користувача. В даний час набір інструментів адаптований до серверних фреймворків Node.JS та підтримує ASP.NET Core. Google стверджує, що вони збираються додати підтримку PHP, Python та Java. Засіб візуалізації – це механізм, який переводить шаблони та компоненти у JavaScript та HTML, які браузери можуть розуміти та відображати. Ivy – це третя ітерація рендерера Angular після вихідного компілятора та рендерера. Крім інших оновлень, Ivy застосовує техніку струшування дерева, що означає, що він видаляє фрагменти

коду, що не використовуються, роблячи програми меншими і дозволяє швидше їх завантажувати.

Деякі інженери-програмісти вважають сам факт підтримки Angular із боку Google є головною перевагою цієї технології. Хоча це може здатися виправданим, самого Google недостатньо. Хорошою ознакою є те, що Google оголосив довгострокову підтримку (LTS) для цієї технології. Ігор Мінар та Стівен Фуїн, інженери Angular, підтвердили цю відданість у Keynote ng-conf 2017. По суті, це означає, що Google планує дотримуватись екосистеми Angular та розвивати її, намагаючись утримувати лідируючі позиції серед інструментів фронтенд-інжинірингу. Оскільки Angular існує вже давно, він перевантажений пакетами, плагінами, надбудовами та інструментами розробки. Також можна вивчити частину роботи спільноти, переглянувши список ресурсів Angular. До них відносяться IDE, інструменти, середовища інтерфейсу користувача, Angular Universal для рендерингу на стороні сервера, інструменти аналітики, засоби для ASP.NET, бібліотеки даних тощо.

Незалежно від заяв LTS, саме спільнота навколо будь-якої технології робить її сильною на ринку. І історія спільноти Angular досить суперечлива. Згідно з опитуваннями розробників StackOverflow 2020 року, Angular (як Angular 1.x, так і Angular) є другою технологією, що найчастіше використовується в категорії Frameworks, Libraries і Other Technologies, що непогано.

Компанії, що використовують Angular: Microsoft, Autodesk, MacDonald's, UPS, Cisco Solution Partner Program, AT&T, Apple, Adobe, , Clarity Design System, Upwork, Udemy, YouTube, Paypal, Nike, Google, Telegram, AWS.

Отже, можна виділити такі переваги фреймворку Angular:

– Фреймворк Angular має вбудовану функціональність для миттєвого оновлення змін, внесених до моделі, з використанням уявлення та навпаки.

– Архітектура, заснована на компонентах, забезпечує більш високу якість коду.

– Компоненти багаторазово використовуються і легко управляються за допомогою ін'єкції залежностей.

– Розробники можуть легко виявити помилки, зберегти код чітким і зрозумілим, а також усунути помилки в міру їхнього введення, оскільки Angular побудований на TypeScript.

– Асинхронні виклики даних можуть бути легко оброблені за допомогою RxJS (бібліотека, яка широко використовується в Angular).

– Довгострокова Підтримка Google (LTS) гарантує, що Google планує і надалі розвивати фреймворк, тому розробники мають доступ до широкої спільноти для підтримки та навчання.

Недоліки Angular:

– Різноманітність різних структур (Injectables, Components, Pipes, Modules тощо) ускладнює вивчення порівняно з React і Vue.js, які мають лише «Component».

Відносно повільна продуктивність з огляду на різні показники. З іншого боку, це можна легко вирішити, використовуючи так званий ChangeDetectionStrategy, який допомагає вручну контролювати процес рендерингу компонентів.

### 1.3.2 Огляд бібліотеки React

Сильна сторона React полягає в тому, що він органічно виріс у результаті використання у світі та продовжує розвиватися у відповідь на інтенсивне використання. Він зазнав значного зростання, але його коріння та постійні переваги полягають у тому, що він є бібліотекою, який використовується Facebook для власних додатків [14].

Можна побачити відданість Facebook до просування інновацій у цій структурі за допомогою перспективних функцій, таких як Concurrent Model. React також активно розробив так звані чисті або функціональні компоненти і хуки, щоб розширити їх можливості. Ці компоненти дозволяють уникнути деяких накладних витрат, пов'язаних із компонентами на основі класів. Vue має деяку підтримку функціональних компонентів, і їх можна створювати в Angular, але React – явний лідер у цій галузі.

Angular узгоджено обробляє великі бази коду і може запропонувати переваги порівняно з React у сфері великомасштабних проектів. У React можна визначати елегантні великомасштабні програми, але сама бібліотека не робитиме стільки, скільки Angular, щоб забезпечити дотримання цього визначення [15].

React використовує незмінні стани об'єкта, доступні тільки через setState(), для представлення стану компонента. Це відрізняється від Vue та Angular, які використовують більш вбудований підхід JavaScript до стану даних [16].

React використовує JSX для своїх шаблонів уявлень. JSX – цікавий підхід, оскільки він схожий на HTML з надздібностями JavaScript (або JavaScript з надздібностями HTML). Під час першого вивчення фреймворку JSX може здатися, що він є надто важким для засвоєння. У довгостроковій перспективі він працює досить добре, і його неважко вивчити. Централізоване керування даними React за умовчанням здійснюється через Redux [17].

Переваги React:

– Економічна ефективність - оскільки немає необхідності витрачати великі гроші від початку створення продукту, завдяки чому скорочується час виведення продукту на ринок.

– Відмінний досвід користувача - ніхто не любить стояти в черзі і чекати, поки веб-сайт повністю завантажиться. У React є функції, які роблять програми React неймовірно швидкими: Concurrent Mode, React Fiber, Suspense, virtual DOM та багато інших.

– Популярність React не є тимчасовим сезонним трендом. Це стабільна та зріла технологія.

– Підвищення продуктивності та швидкості - коли справа доходить до таких технологій, як бібліотеки або фреймворки, їхня продуктивність та швидкість є ключем до успіху. Оскільки React може впоратися з Facebook, що найчастіше використовується додатком, він може впоратися з чим завгодно. І причина того, що React такий швидкий – це віртуальний DOM. Замість того, щоб кожен раз генерувати всі HTML-файли, він шукає відмінності між старим DOM і поточним файлом і оновлює його відповідним чином [14].

– Оптимізація SEO. У той час як пошуковим системам важко працювати з програмами з важким JavaScript, для програм React все по-іншому. Це тому, що вони можуть працювати на сервері, в той час як віртуальна модель DOM дбає про рендеринг та малювання в браузер.

– Скорочений час виходу на ринок (TTM) – час – гроші, особливо для стартапів та інших компаній, які прагнуть створювати цифрові продукти. Оскільки React прискорює весь процес розробки, можна швидше протестувати MVP на ринку і внести відповідні зміни, не витрачаючи на це великих грошей.

– Зворотна сумісність – це те, що деякі фреймворки вимагають переписування коду після оновлення фреймворку до останньої версії. Але не у випадку з React.js. Загальнодоступний API завжди залишається практично незмінним, тому Facebook та інші компанії простіше оновлювати код, використовуючи його старі частини.

– Час розробки – React – це швидкість, і це означає не лише швидкість завантаження сторінки, а й швидкість розробки. Багато речей прискорюють процес розробки: 1. готові рішення 2. створені компоненти можна використовувати повторно 3. можливість налаштувати веб-додаток за допомогою «Create React App», який є стартовим комплектом, що надається Facebook [14].

– Легко масштабується - завдяки модульності програми React легко масштабуються. Наприклад, можна почати зі створення простого MVP, а потім зробити його односторінковим додатком таким же всеосяжним, як Facebook.

– Корисні інструменти – розробники React мають у своєму розпорядженні набір зручних інструментів під назвою React Developer Tools. Це розширення для Google Chrome та Mozilla Firefox, яке дозволяє перевіряти ієрархію компонентів у віртуальній DOM. Це також дозволяє редагувати окремі компоненти.

– Велике співтовариство - Співтовариство React з моменту свого заснування в 2013 році швидко зростало і продовжує зростати. Велика ймовірність, що проблема, яку намагається розв'язати розробник, вже вирішена чи легко знайде того, хто йому допоможе. Це тому, що є багато розробників, які роблять React краще та краще [12].

– Багаторазові компоненти, також відомі як модульність, тут дуже допомагає компонентний підхід, тому що програми React складаються з окремих компонентів, які можна розділити на підкомпоненти. Кожен компонент та підкомпоненти відповідають за одну невелику частину всієї програми та можуть бути повторно використані у будь-який час та в будь-якому місці. Завдяки цьому можна створити складну програму з простих блоків.

– Зручність тестування та налагодження - ReactJS використовує власні інструменти для тестування та налагодження компонентів перед фактичним використанням.

Недоліки:

– React не однозначний і залишає розробникам можливість вибирати найкращий спосіб розвитку. Це може бути вирішено сильним лідерством проєкту та добрими процесами.

– Спільнота ділиться за способами написання CSS у React, які поділяються на традиційні таблиці стилів (CSS Modules) та CSS-in-JS (тобто Emotion та Styled Components).

– React відходить від компонентів на основі класів, що може стати на заваді розробникам, яким комфортніше працювати з об'єктно-орієнтованим програмуванням (ООП).

– Змішування шаблонів з логікою (JSX) може спантеличити деяких розробників при перших знайомствах з React.

В умілих технічних руках React – потужний інструмент для створення надійних, продуктивних та масштабованих цифрових продуктів. Власники бізнесу особливо оцінять його рентабельність, скорочення часу виведення ринку та оптимізацію SEO. З іншого боку, розробникам сподобається компонентний підхід та скорочений час розробки [18].

Компанії, що використовують React: Facebook, Instagram, Netflix, Yahoo, Whatsapp, Dropbox, Airbnb, Atlassian, Microsoft та багато інших.

### 1.3.3 Огляд фреймворку Vue

У певному сенсі Vue знаходиться десь між Angular та React, компромісом між низхідним дизайном Angular та органічним зростанням React. Незважаючи на те, що Vue є новим фреймворком і немає підтримки з боку великої

корпорації, він йде в ногу з розробками та надає повністю життєздатну структуру. Також існує ряд якісних плагінів та комплектів для Vue (наприклад, Quasar та Vuetify)[19].

Vue має репутацію найпростішого в освоєнні. Ймовірно, це походить з його моделі даних JSON та визначень уявлення (проти JSX React). Шаблони Vue можуть також включати вбудовані функції JavaScript, на відміну від JSX. Vue пропонує Vuex як вбудоване централізоване рішення для управління станом [20].

Переваги Vue [23]:

– Цей фреймворк є досить простим для вивчення. Також цьому сприяє широка та якісна документація.

– Універсальність дозволяє Vue масштабуватися між бібліотекою та повнофункціональним фреймворком, що спрощує розробникам створення програм.

– Vue полегшує розробку, оскільки готовий до розгортання проект важить 20 КБ після min + gzip. Це призводить до прискорення виконання, а також стимулює розробку та дозволяє розробникам відокремлювати віртуальну модель DOM від шаблону та компілятора. Більше того, коли мінімальний розмір проєкту не потрібно докладати додаткових зусиль для надмірної оптимізації [15].

– Будучи доступним, універсальним та продуктивним, Vue допомагає розробникам створювати програми, які можна супроводжувати та тестувати. Його можна використовувати для створення як SPA (односторінкових додатків), так і дуже складних веб-застосунків, оскільки розробники можуть вільно інтегрувати дрібніші частини в існуючу інфраструктуру, не зачіпаючи всю систему [16].

– На відміну від інших фреймворків, Vue.js можна використовувати як бібліотеку та як повноцінний фреймворк. Його можна модифікувати, змінювати та масштабувати відповідно до вимог. У той же час, якщо необхідно виконати будь-яку інтеграцію або розробити веб-додаток, вона може використовувати REST API, розробка WordPress або WooCommerce [17].

Недоліки Vue:

– Нестача ресурсів. Vue.js, як і раніше, займає досить невелику частку ринку в порівнянні з React або Angular, що означає, що обмін знаннями в цьому середовищі все ще знаходиться на початковій стадії.

– Ризик надмірної гнучкості. Іноді у Vue.js можуть виникнути проблеми при інтеграції у величезні проекти, і поки що немає досвіду можливих рішень, але вони обов'язково з'являться найближчим часом.

– Vue швидко розвивається. Те, що працює сьогодні у Vue, завтра може застаріти. Це найбільший недолік Vue,

– Оскільки розробники знаходять його досить складним, і їм доводиться досить часто вивчати фреймворк.

– Невелика спільнота. В результаті він не такий популярний у порівнянні з іншими фреймворками, такими як React та Angular.

Крім того, оскільки фреймворк був створений китайською компанією, більшість коду була написана китайською мовою, що створює деякі проблеми для англомовних користувачів. Більшість членів спільноти не говорять англійською, що теж у невеликій кількості, що може не підтримати.

Компанії, які використовують Vue.js: Xiaomi, Alibaba, WizzAir, EuroNews, Grammarly, Gitlab та Laracasts, Adobe, Behance, Codeship, Reuters.

### 1.4 Постановка задачі

Інтернет-магазин - це звичний кожному користувачеві Інтернету веб-сайт, що представляє інформацію про товари в зручному структурованому вигляді.

Інтернет-магазини створюються із застосуванням систем управління вмістом сайтів, оснащених необхідними модулями. З технічної точки зору інтернет-магазин – це сукупність веб-вітрини і торгової системи – фронт системи і бек-офісу. Веб-вітрина надає інтерфейс до бази даних товарів, що продаються у вигляді каталогу, прайс-листа, працює з віртуальним торговим візком, оформляє замовлення і реєструє покупця, надає допомогу покупцеві в онлайновому режимі, передає інформацію в торговельну систему і забезпечує безпеку особистої інформації покупця.

У загальному випадку основні функції інтернет-магазину – це інформаційне обслуговування покупця, обробка замовлень, проведення платежів, а також збір та аналіз різної статистичної інформації. Програмний комплекс управління інтернет-магазином дозволяє формувати і інтерфейс з покупцем, і функціональні можливості інтернет-магазину.

Метою даної роботи є розробка веб-сайту для підтримки управління асортиментом товарів, списку замовлень, переліком клієнтів, переліком категорій та співробітників магазину. Для досягнення поставленої мети необхідно послідовно вирішити такі завдання:

– проаналізувати роботу веб-магазинів;

– провести аналіз існуючих засобів розробки веб-сайтів та здійснити вибір тих, які є найкращими для досягнення поставленої мети;

– розробити та описати функціонал усіх можливих користувачів у системі;

– провести опис усіх алгоритмів, що використовуються під час розробки програмного продукту;

– провести тестування створеного програмного рішення.

# 2 ПРОГРАМНА РЕАЛІЗАЦІЯ

## 2.1 Створення базового проєкту

Для створення базового проєкту використаємо Angular CLI. За допомогою команди “ng new”, з параметрами “--prefix” для задання префіксу за замовчуванням, “--style” для вибору пре процесору CSS, та “--skip-tests” для пропускання створення початкових тестів.

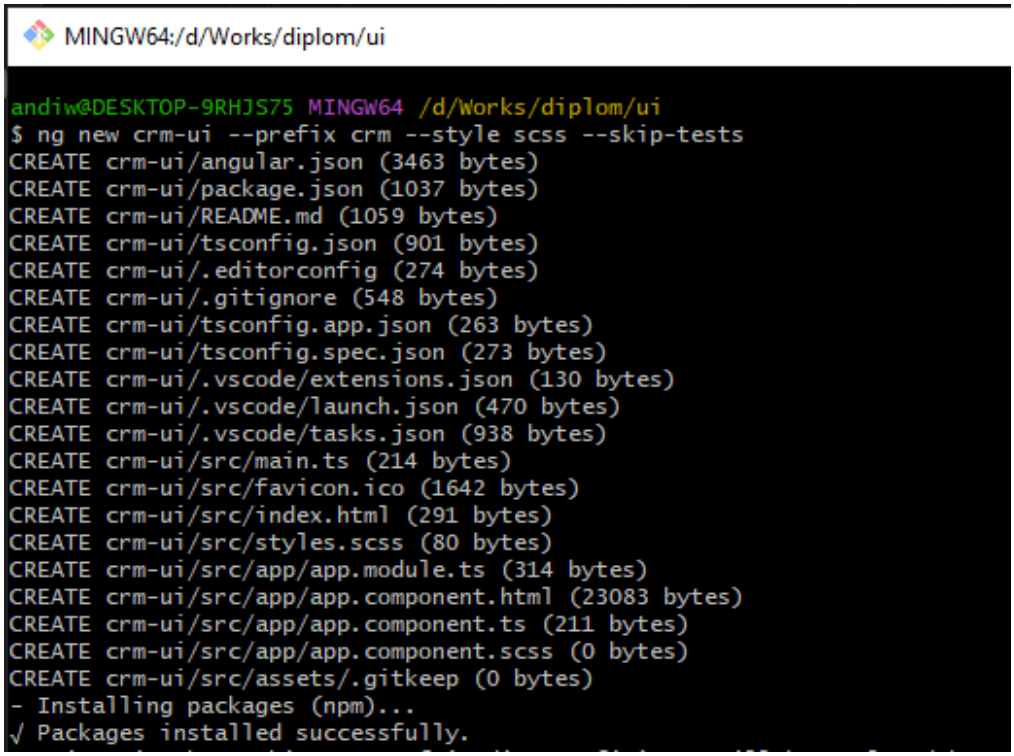


Рисунок 1 – Створення початкового проєкту

Після того як Angular CLI згенерував проєкт та встановив всі початкові залежності, відкриваємо проєкт в редакторі коду, наприклад Visual Studio Code, та вводимо в терміналі команду “npm start” для запуску проєкту, та перевірки того що все встановилось та працює.

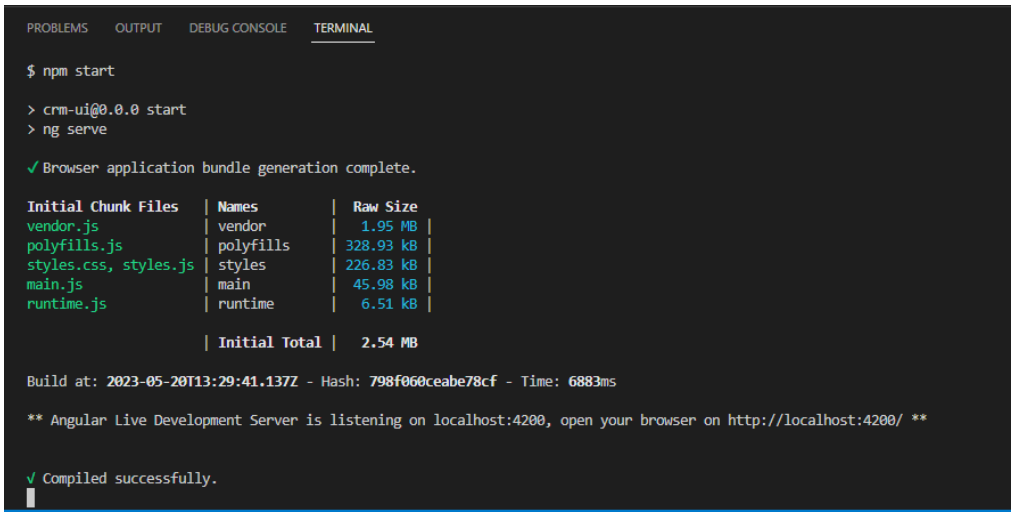


Рисунок 2 – Запуск проєкту

Після успішно збирання проєкту, сайт буде доступний за адресом “localhost:4200”.

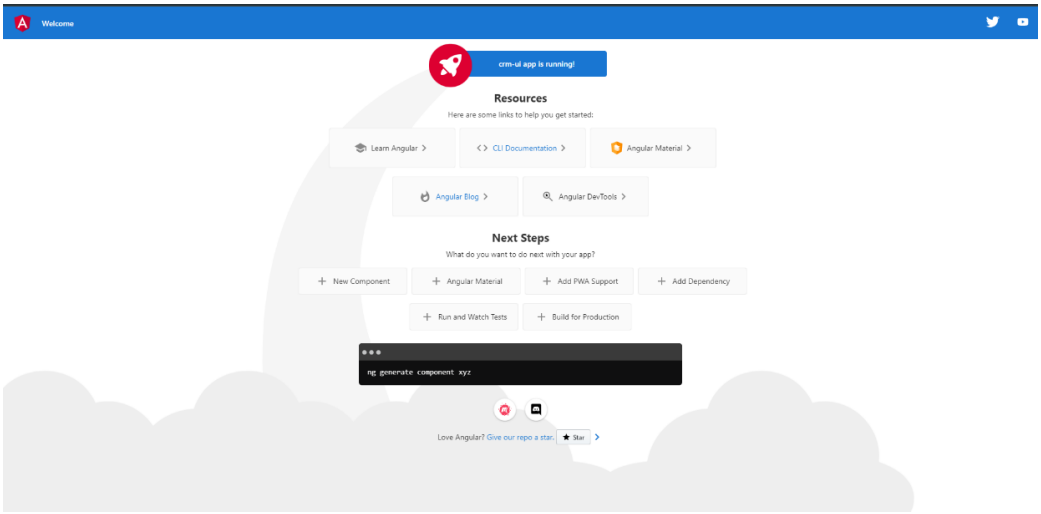


Рисунок 3 – Початкова сторінка

Для подальшої роботи, необхідно встановити додаткові залежності проєкту. Додамо бібліотеки для стилізування додатку.

PrimeNG – це популярна бібліотека компонентів для розробки інтерфейсів користувача на основі Angular. Вона надає багатий набір готових компонентів, таких як таблиці, форми, меню, діалогові вікна та інші, що допомагають швидко побудувати сучасні та функціональні додатки. Для встановлення виконаємо команду “npm install primeng”.

Додатково до PrimeNG встановимо PrimeFlex та PrimeIcons.

PrimeFlex – це роздільний пакет CSS класів, розроблений спеціально для використання з PrimeNG і Angular. Він надає потужні можливості для розташування, вирівнювання та розміщення елементів на сторінці.

PrimeIcons – це набір векторних іконок, які можна використовувати в PrimeNG та інших проектах для надання додаткової графічної інформації та покращення візуального дизайну.

Для встановлення цих бібліотек використовуємо команду “npm install primeflex” та “ npm install primeicons”.

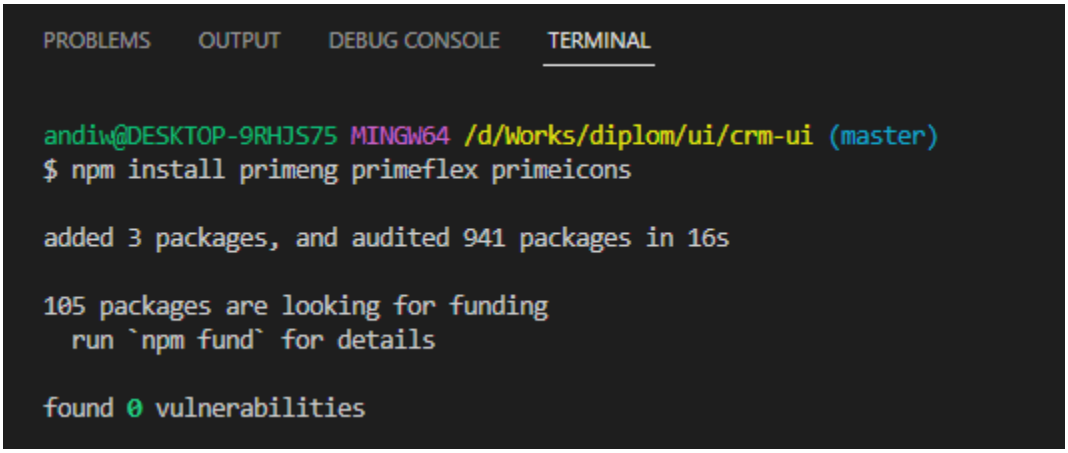


Рисунок 4 – Встановлення пакетів для стилізації

Після їх встановлення, потрібно провести базове налаштування.


Рисунок 5 – Підключення PrimeNG до проєкту

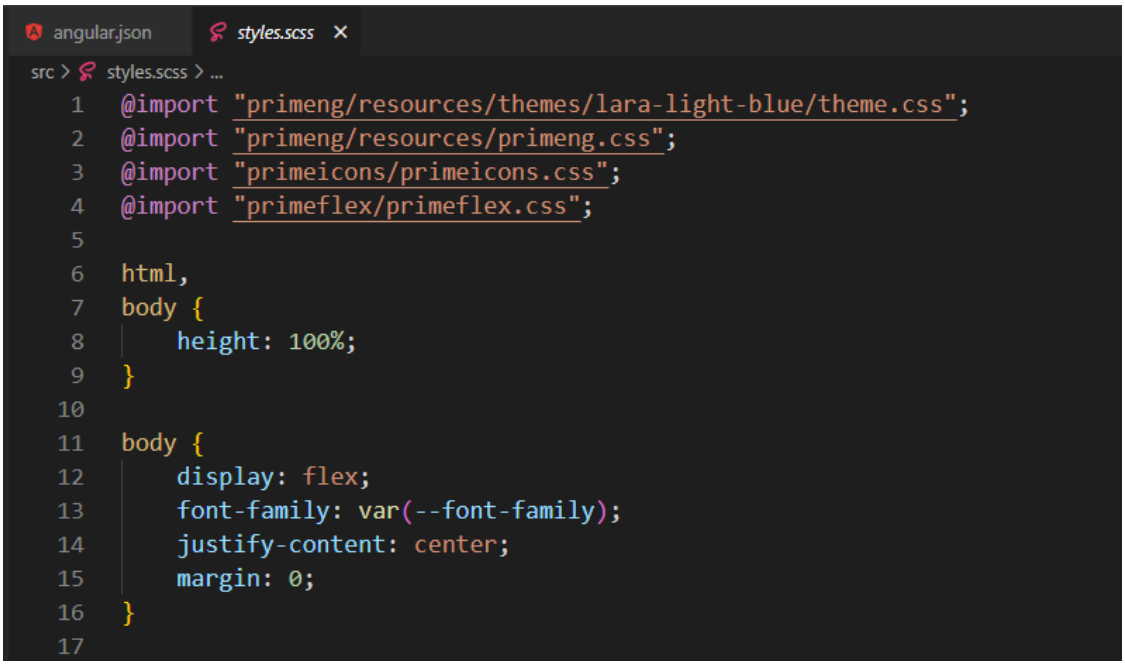


Рисунок 6 – Задання базових стилів, підключення PrimeFlex та PrimeIcons

Для підтримання форматування коду в єдиному стилі, встановимо додаткові залежності розробки, такі як Prettier та ESLint.

ESLint – це інструмент статичного аналізу коду для JavaScript, який допомагає виявляти та виправляти потенційні проблеми, стилістичні неузгодженості та помилки у вашому коді, допомагаючи забезпечити високу якість програмного забезпечення.

Prettier – це інструмент для автоматичного форматування коду, який дозволяє автоматично розташовувати його відповідно до зазначених стандартів форматування. Він забезпечує одноманітний та консистентний стиль коду, спрощуючи процес розробки та поліпшуючи читабельність вашого програмного забезпечення.

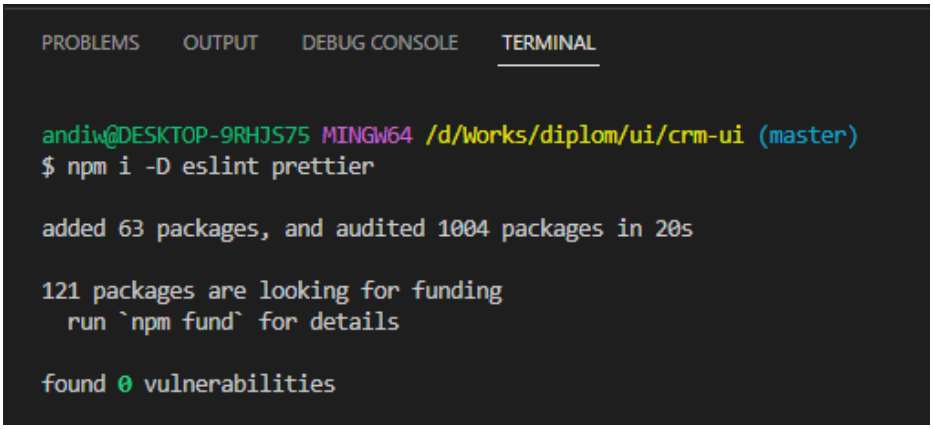


Рисунок 7 – Встановлення Prettier та ESLint


Рисунок 8 – Налаштування Prettier

Для локалізації проєкту, встановимо та налаштуємо бібліотеку Transloco.

Transloco – це бібліотека для інтернаціоналізації (i18n) в Angular, яка дозволяє розширити можливості додатку для підтримки різних мов та локалізації. За допомогою Transloco можна зручно управляти перекладами, динамічно завантажувати локалізаційні ресурси та забезпечити вибір мови користувача.

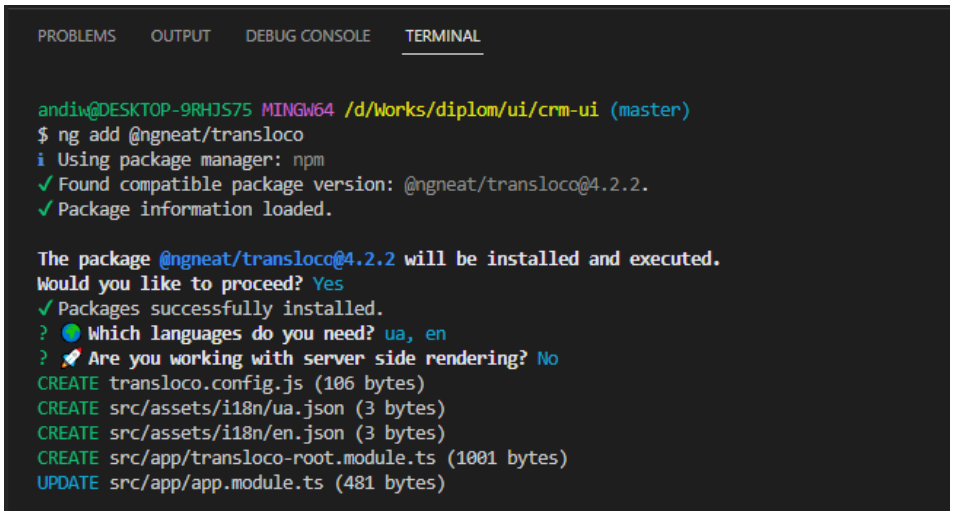


Рисунок 9 – Встановлення та налаштування Transloco

Після того як всі необхідні бібліотеки встановлено та налаштовано, за допомогою Angular CLI, згенеруємо декілька модулів та компонентів. Для цього використаємо команди “ng generate module” та “ng generate component” відповідно.

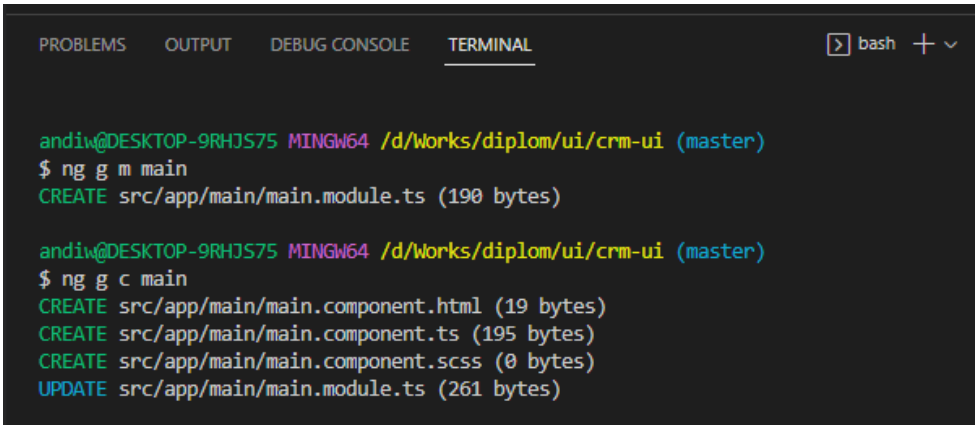


Рисунок 10 – Генерація “main” модуля та компоненту

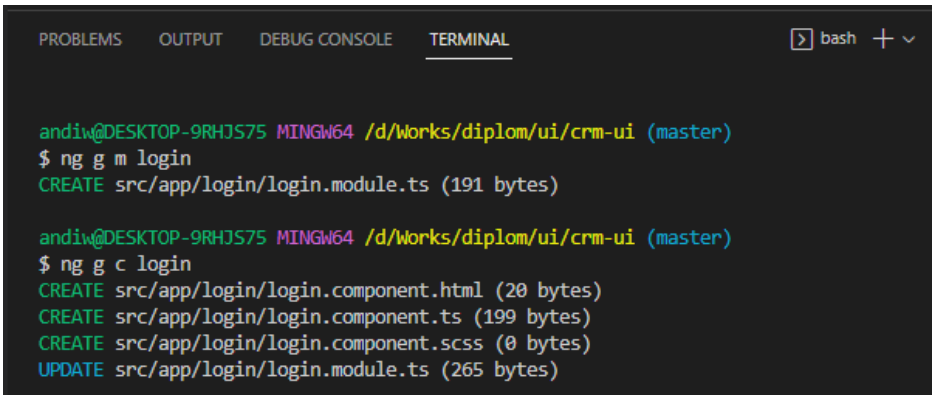


Рисунок 11 – Генерація “login” модуля та компоненту

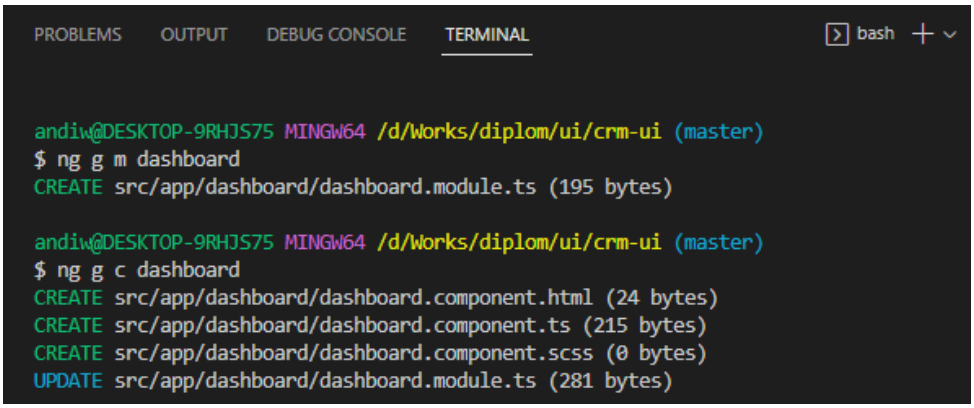


Рисунок 12 – Генерація “dashboard” модуля та компоненту

Після того як всі модулі та компоненти згенеровані, створимо та налаштуємо головний роутинг файл.


Рисунок 13 – Створення “app.routing.ts”

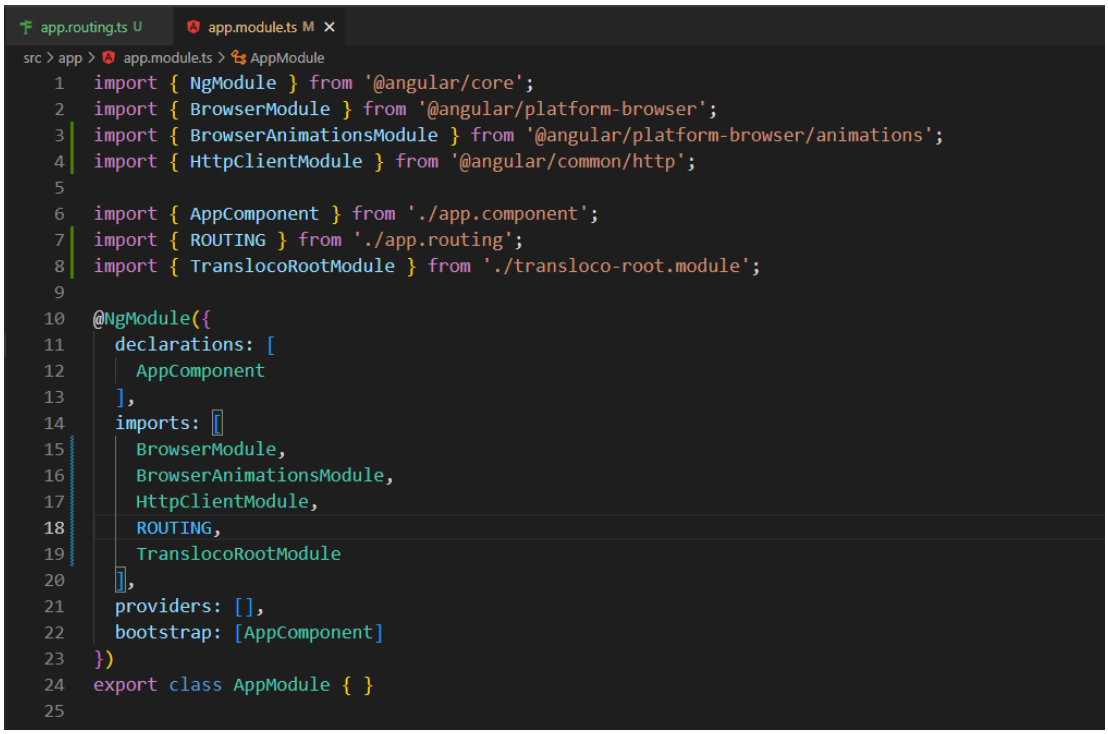


Рисунок 14 – Підключення “app.routing” до головного модулю

Аналогічні дії, проводимо для інших модулів. Наступним кроком буде зміна головного компоненту, в “app.component.html” змінюємо згенеровано сторінку на “<router-outlet></router-outlet>”, та додаємо такий тег в “main.comoponent.html”. “Main” модуль виконує роль “обгортки” над іншими компонентами, в ньому будуть додані блоки які використовуються на всіх інших сторінках такі як “menu”, “sidebar” та інші.

Для перевірки результату роботи запустимо застосунок за допомогою команди “npm start”.

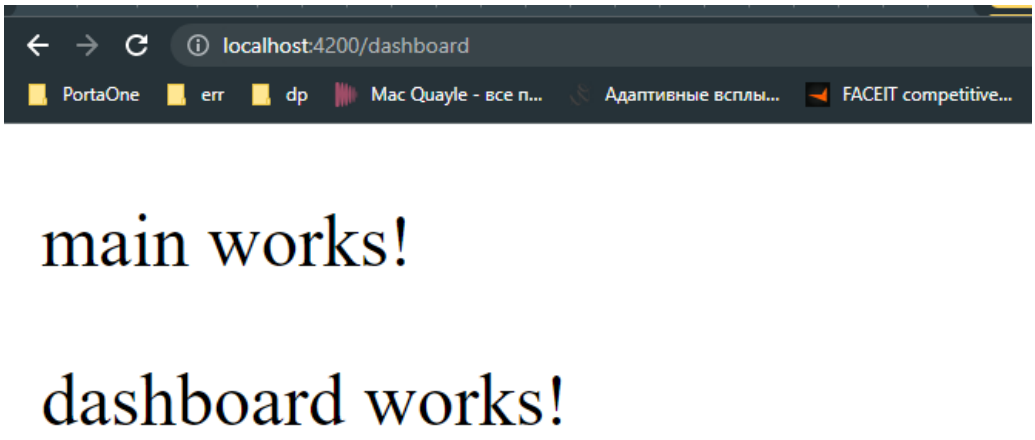


Рисунок 15 – Перевірка роботи головної сторінки

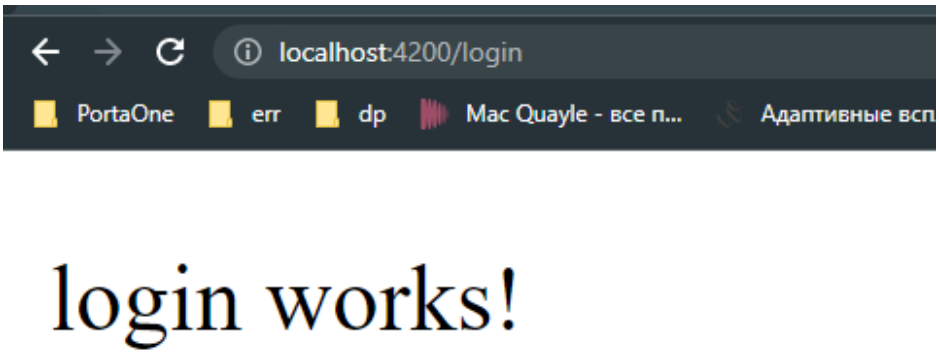


Рисунок 16 – Перевірка роботи сторінки “login”

Як ми бачимо на рисунках 15, 16 все працює правильно, роутинг працює як очікувалось, бібліотеки зі стилями підключились.

### 2.2 Створення сервісів

Сервіси Angular, використовуються для розділення для організації та обробки бізнес-логіки, спільних ресурсів і функціональності, які не пов'язані напряму з компонентами. Основна функція сервісів – надання способу доступу

до даних, виконання HTTP-запитів, обробки подій, маніпуляції з даними та забезпечення спільної функціональності між різними компонентами. Сервіси можуть бути використані в будь-якому компоненті або іншому сервісі в рамках додатку.

Для більш зручної роботи з запитами до API, було створено “api.service.ts”. Цей сервіс має один публічний метод “request”, який виконує API запити, додає необхідні заголовки, cookie, та обробляє помилки і приводить їх до заданого формату.

```
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable, catchError, of } from 'rxjs';

import { environment } from 'src/environments/environment';
import { ApiResponse } from '../interfaces';

type RequestMethod = 'POST' | 'GET';

@Injectable({ providedIn: 'root' })
export class ApiService {
    private apiUrl: string = environment.apiUrl;

    constructor(private http: HttpClient) {}

    request(method: RequestMethod, body: object, entity: string, action?: string):
Observable<ApiResponse> {
        return this.http
            .request<ApiResponse>(method, `${this.apiUrl}/api/${entity}${action ? `/${action}` : ''}`, {
                body,
                withCredentials: true
            })
            .pipe(catchError((err: HttpErrorResponse) => this.prepareError(err)));
    }

    private prepareError(error: HttpErrorResponse): Observable<ApiResponse> {
        const contentType: string = error.headers.get('content-type');
        let response: ApiResponse;

        if (contentType && contentType.includes('application/json')) {
            const body: object = error.error;
            const isServerError: boolean = !!body['faultCode'];
            response = {
                faultString: isServerError ? 'Internal server error' : body['faultString'],
                faultCode: isServerError ? '_internal_server_error' : body['faultCode'],
                success: false,
                data: undefined,
                validationResult: isServerError ? '_internal_server_error' : body['validationResult']
            };
        } else {
            response = {
                faultString: error.statusText,
                faultCode: '_unclear_server_response',
                success: false,
                data: undefined
            };
        }
        return of(response);
    }
}
```

Рисунок 17 – Код “api.service.ts”

Наступний сервіс має назву “auth.service.ts”, та відповідає за авторизацію користувачів. Має такі методи як:

– “login” – відправляє запит до API, якщо відповідь від сервера успішна, виконує метод “start”;
– “logout” – видаляє cookie файли та направляє користувача на сторінку авторизації;
– “sendResetLink” – відправляє запит до API, для відновлення паролю;
– “resetPassword” – відправляє запит до API, для задання нового паролю, параметром приймає новий пароль та підтвердження нового паролю;
– “refresh” – відправляє запит до API, для поновлення терміну дії “access” та “refresh” токенів;
– “start” – параметром приймає час, з періодичністю до якого виконує метод “refresh” щоб користувач завжди був авторизований в системі;
– “stop” – зупиняє процес метода “start”.

```ts
import { Injectable, NgZone } from '@angular/core';
import { tap, Subscription, interval, switchMap, map, Observable, of } from 'rxjs';

import { ApiService } from './api.service';
import { ApiResponse } from '../interfaces';

@Injectable({ providedIn: 'root' })
export class AuthService {
    authorized: boolean = false;
    private jwtSub: Subscription = Subscription.EMPTY;

    constructor(private api: ApiService, private ngZone: NgZone) {}

    login(param: { username: string; password: string }): Observable<ApiResponse> {
        return this.api.request('POST', { username: param.username, password: param.password }, 'auth', 'signin').pipe(
            tap(res => {
                if (res.success) {
                    this.start(Number.parseInt(res.data?.expires_in));
                }
            })
        );
    }

    sendResetLink(param: { username: string; email: string }): Observable<ApiResponse> {
        return this.api.request('POST', { username: param.username, email: param.email }, 'auth', 'reset-request');
    }

    resetPassword(param: { password: string; passwordConfirm: string }): Observable<ApiResponse> {
        return this.api.request(
            'POST',
            { password: param.password, passwordConfirm: param.passwordConfirm },
            'auth',
            'change-password'
        );
    }

    logout(): Observable<any> {
        return this.api.request('POST', {}, 'logout');
    }

    refresh(): Observable<ApiResponse> {
        return this.api.request('POST', {}, 'auth', 'refresh');
    }

    isAuthorized(): Observable<boolean> {
        if (this.authorized) {
            return of(true);
        }

        return this.refresh().pipe(
            map(res => {
                if (res.success) {
                    this.start(Number.parseInt(res.data?.expires_in));
                    return true;
                }

                return false;
            })
        );
    }

    start(tokenLifetime: number): void {
        this.stop();
        this.authorized = true;
        this.ngZone.runOutsideAngular(() => {
            this.jwtSub = interval(tokenLifetime * (3 / 4) * 1000)
                .pipe(switchMap(() => this.refresh()))
                .subscribe((res: ApiResponse) => {
                    if (res.success) {
                        this.authorized = true;
                        tokenLifetime = Number.parseInt(res.data?.expires_in);
                    } else {
                        this.stop();
                    }
                });
        });
    }

    stop(): void {
        this.authorized = false;
        this.jwtSub.unsubscribe();
    }
}
```

Рисунок 18 – Код “auth.service.ts”

Сервіс “main-guard.service.ts” перевіряє чи авторизований користувач, якщо авторизований – дозволяє перехід на захищені сторінки, а в іншому випадку перенаправляє користувача на сторінку авторизації.

```
import { Injectable } from '@angular/core';
import type { CanActivateChild, ActivatedRouteSnapshot, UrlTree } from '@angular/router';
import { Router } from '@angular/router';
import { Observable, tap } from 'rxjs';
import { of, share, mergeMap, map } from 'rxjs';

import type { ApiResponse } from '../interfaces';
import { AuthService } from './auth.service';
import { UserService } from './user.service';

@Injectable({ providedIn: 'root' })
export class MainGuardChildService implements CanActivateChild {
    private activateObs: Observable<UrlTree | null> | null;

    constructor(private router: Router, private authSrv: AuthService, private userSrv: UserService) {}

    canActivateChild(_childRoute: ActivatedRouteSnapshot): Observable<UrlTree | boolean> {
        if (!this.activateObs) {
            const chain: Observable<ApiResponse> = of({ success: true });

            this.activateObs = chain.pipe(
                mergeMap(res => {
                    return res.success ? this.authSrv.isAuthorized() : of({ success: false });
                }),
                mergeMap(success => (success ? this.userSrv.isDataLoaded() : of(false))),
                map(success => {
                    this.activateObs = null;
                    return success ? null : this.router.parseUrl('/login');
                }),
                share()
            );
        }

        return this.activateObs;
    }
}

```

Рисунок 19 – Код “main-guard.service.ts”

Останній загальний сервіс “user.service.ts”, він містить інформацію про авторизованого користувача, завантажує данні про нього.

```ts
import { Injectable } from '@angular/core';
import { Observable, of, tap, map } from 'rxjs';

import { ApiService } from './api.service';
import type { ApiResponse, User } from '../interfaces';

@Injectable({ providedIn: 'root' })
export class UserService {
    myData: User;
    constructor(private api: ApiService) {}

    private dataLoaded: boolean = false;

    isDataLoaded(): Observable<boolean> {
        return this.dataLoaded ? of(true) : this.loadData().pipe(map(res => res.success));
    }

    private loadData(): Observable<ApiResponse<boolean>> {
        return this.api.request('GET', {}, 'auth', 'get-my-info').pipe(
            tap(res => {
                if (res.success) {
                    this.myData = res.data;
                    this.dataLoaded = true;
                }
            }),
            map(res => (res.success ? { success: true } : { success: false }))
        );
    }
}

```

Рисунок 20 – Код “user.service.ts”

### 2.3 Розробка базової сторінки

Всі сторінки будуть мати однакову структуру, загальний список елементів, та сторінку редагування окремого елементу. Розглянемо цю структуру на прикладі сторінки “Категорії”.

Для початку за допомогою Angular CLI згенеруємо модуль “category”, після чого генеруємо однойменний компонент “category”, в цьому компоненті підключаємо роутинг та загальні компоненти. Наступним кроком є створення компонентів “list” та “info”, вони будуть відображати список всіх категорій та інформацію про конкретну категорію відповідно.

Для  налаштування роутингу в цього модулю, створюємо окремий файл “category.routing.ts”.

Робота з даними буде відбуватися завдяки сервісу “category.service.ts” в рамках цього модуля.

Після генерації та створення файлів маємо наступну файлову структуру, така структура буде в усіх інших модулях.

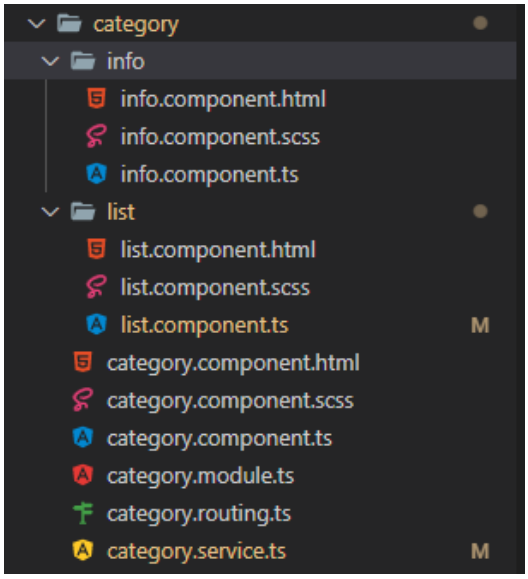


Рисунок 21 – Файлова структура модуля “category”

Розглянемо більш детально файл роутингу, в ньому ми описуємо можливі маршрути:

– “domen-name.com/category” – в цьому випадку користувача перенаправляє на сторінку “list” з усіма категоріями;
– “domen-name.com/category/list” – сторінка з усіма категоріями;
– “domen-name.com/category/3” – сторінка конкретної категорії;
– “domen-name.com/category/qwerty” – якщо користувач переходить по неіснуючому посиланню, відкриється сторінка “list”.

```typescript
import type { ModuleWithProviders } from '@angular/core';
import { RouterModule } from '@angular/router';
import type { Routes } from '@angular/router';

import { CategoryComponent } from './category.component';
import { ListComponent } from './list/list.component';
import { InfoComponent } from './info/info.component';

const routes: Routes = [
    {
        path: '',
        component: CategoryComponent,
        children: [
            {
                path: '',
                pathMatch: 'full',
                redirectTo: 'list'
            },
            {
                path: 'list',
                component: ListComponent
            },
            {
                path: 'info/:id',
                component: InfoComponent
            },
            {
                path: '**',
                redirectTo: 'list'
            }
        ]
    }
];

export const ROUTING: ModuleWithProviders<RouterModule> = RouterModule.forChild(routes);

```

Рисунок 22 – Код “category.routing.ts”

Після налаштування роутингу, розпочнемо роботу по розробці “list” компоненту. PrimeNG має готовий компонент таблиці “p-table” він буде головним елементом сторінки. Додатково до нього додамо кнопку для створення нових категорій. При натисканні на неї буде відображатися діалоговий компонент “p-dialog”, в якому буде два поля: назва категорії, та “p-dropdown” для обирання батьківської категорії.

```
<div class="card h-full" *transloco="let t">
    <p-table
        #dt
        class="custom-table"
        [value]="categorys"
        dataKey="id"
        [rows]="25"
        [showCurrentPageReport]="true"
        [rowsPerPageOptions]="[25, 50, 75, 100]"
        [loading]="loading"
        [paginator]="true"
        [currentPageReportTemplate]="t('_table.showing')"
        [globalFilterFields]="['name', 'parent_id']"
        [tableStyle]="{ 'min-width': '60rem' }"
    >
        <ng-template pTemplate="caption">
            <div class="flex">
                <button pButton icon="pi pi-plus" (click)="showDialog()" [label]="t('_addCategory')"></button>
                <span class="p-input-icon-left ml-auto">
                    <i class="pi pi-search"></i>
                    <input
                        pInputText
                        type="text"
                        [placeholder]="t('_search')"
                        (input)="applyFilterGlobal($event, 'contains')"
                    />
                </span>
            </div>
        </ng-template>
        <ng-template pTemplate="header">
            <tr>
                <th style="width: 50%">{{ t('_categoryName') }}</th>
                <th style="width: 50%">{{ t('_parrentCategory') }}</th>
            </tr>
            <tr>
                <th>
                    <p-columnFilter type="text" field="name"></p-columnFilter>
                </th>
                <th>
                    <p-columnFilter type="text" field="parent_id"></p-columnFilter>
                </th>
            </tr>
        </ng-template>
        <ng-template pTemplate="body" let-category>
            <tr>
                <td>
                    <span class="link" [routerLink]="['/category', 'info', category.id]">{{ category.name }}</span>
                </td>
                <td>
                    {{ getParrentName(category.parent_id) }}
                </td>
            </tr>
        </ng-template>
        <ng-template pTemplate="emptymessage">
            <tr>
                <td colspan="2">{{ t('_table.notFound') }}</td>
            </tr>
        </ng-template>
    </p-table>

    <p-dialog
        [header]="t('_addCategory')"
        [(visible)]="dialogVisible"
        [modal]="true"
        [style]="{ width: '50vw' }"
        [draggable]="false"
        [resizable]="false"
        [formGroup]="addNewForm"
        (onHide)="closeDialog()"
    >
        <div class="flex flex-column pt-1">
            <input class="flex mb-3" [placeholder]="t('_categoryName')" pInputText formControlName="name" />
            <p-dropdown
                styleClass="flex"
                optionLabel="name"
                optionValue="id"
                appendTo="body"
                [options]="categoryOptions"
                formControlName="parent_id"
            ></p-dropdown>
        </div>

        <ng-template pTemplate="footer">
            <button pButton class="p-button-danger" (click)="closeDialog()" [label]="t('_cancel')"></button>

            <button
                pButton
                icon="pi pi-check"
                class="p-button-success"
                (click)="addCategory()"
                [disabled]="addNewForm.invalid"
                [label]="t('_save')"
            ></button>
        </ng-template>
    </p-dialog>
</div>
```

Рисунок 23 – Код “list.component.html”

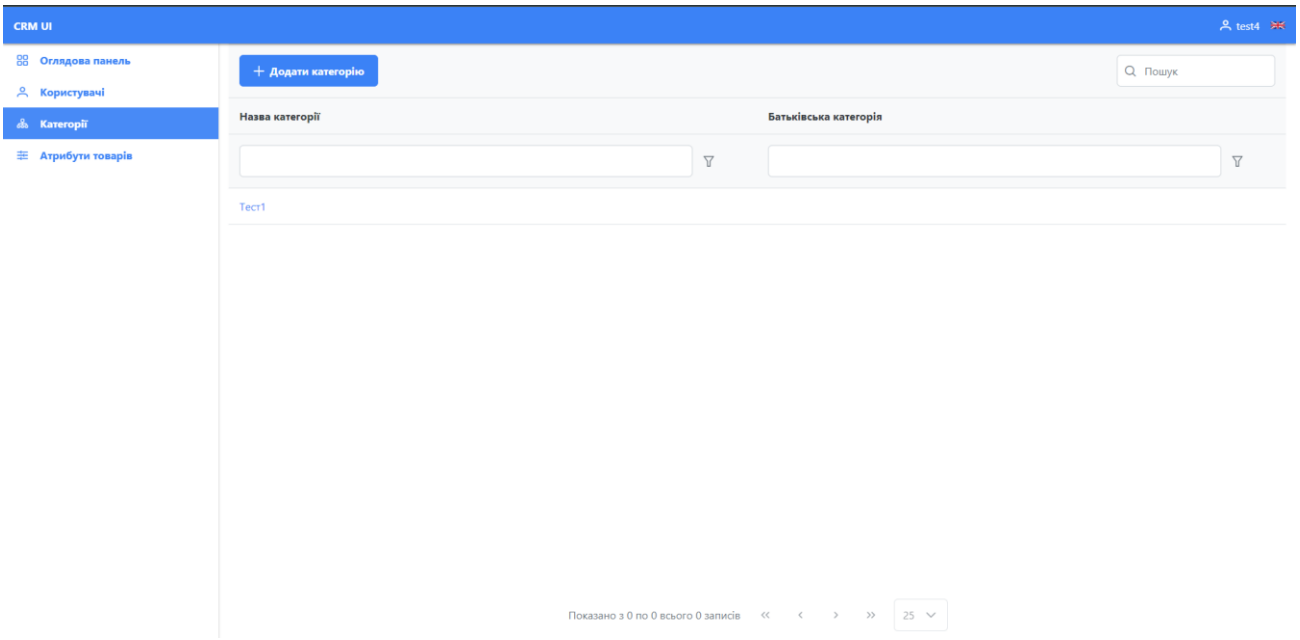


Рисунок 24 – Сторінка списку категорій


Рисунок 25 – Форма додавання нової категорії

Сторінка редагування, “info” компонент, дозволяє змінювати категорію, та дозволяє видалити її.

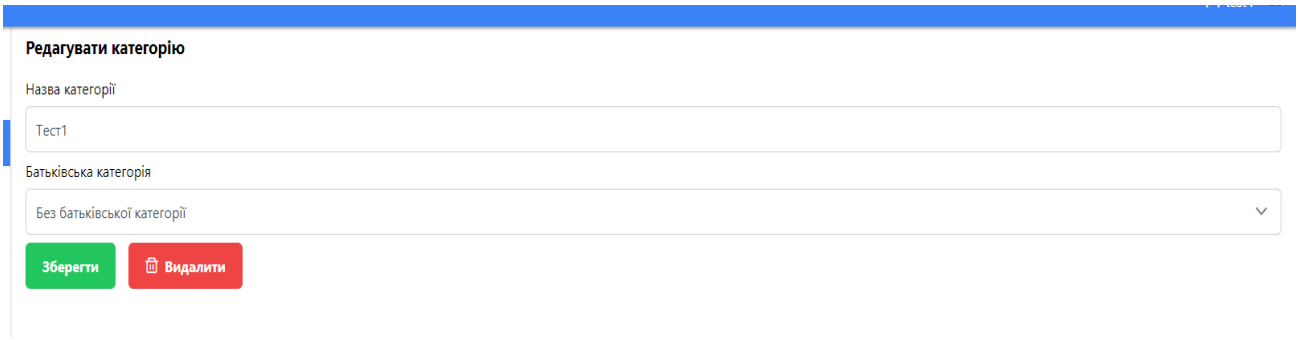


Рисунок 26 – Сторінка редагування

Всі інші модулі (сторінки), мають подібну структуру, вона є достатньо зручною та покриває CRUD операції.

### 2.4 Інструкція користувача

Для збірки проєкту потрібно встановити NodeJS версії 14.20, або вище.

Після клонування репозиторію, для встановлення всіх залежностей, виконується команда “npm install”, далі в файлах “environment.ts” та “environment.prod.ts” потрібно змінити параметр “apiUrl” на актуальну адресу сервера.

Запуск проєкту в режимі розробки – “npm start”.

Збірка проєкту виконується за допомогою команди “npm run build”, після збірки проєкту, в каталозі створюється новий каталог з назвою “dist”, далі файли цього каталогу завантажуються на сервер, та сайт готовий до роботи.

# ВИСНОВКИ

У результаті виконаної роботи було розроблено клієнтську частину системи керування контеном інтернет-магазину. Було проаналізовано різні інструменти розробки веб-застосунків такі як React, Vue та Angular. У результаті розробки було створено та налаштовано проєкт на фреймворкі Angular, розроблені всі необхідні сторінки для керування інтернет-магазином.

Розроблений додаток задовольняє всім поставленим вимогам.

# СПИСОК ЛІТЕРАТУРИ

1. The Forrester research: https://neo4j.com/whitepapers/forrester-wave-graph-data-platforms/

2. Розробка WEB додатка: https://webcase.com.ua/uk/razrabotka-web-prilozhenij/

3. Fickers A., Balbi G. (ed.). History of the International Telecommunication Union: Transnational techno-diplomacy from the telegraph to the Internet. – Walter de Gruyter GmbH & Co KG, 2020. – Т. 1.

4. Mohanty M. S., nanda D. R. B. E-commerce: evolution, present status and future prospects.

5. Li X. et al. Measuring ease of use of mobile applications in e-commerce retailing from the perspective of consumer online shopping behaviour patterns //Journal of Retailing and Consumer Services. – 2020. – Т. 55. – С. 102093.

6. JavaScript web-фреймворк https://brander.ua/technologies/expressjs

7. Найпопулярніші фреймворки 2020 року https://katerinka1200.github.io/index4.html

8. Dao C. The Nature And Evolution Of JavaScript. – 2020.

9. Mitropoulos D. et al. Time present and time past: analyzing the evolution of JavaScript code in the wild //2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). – IEEE, 2019. – С. 126-137.

10. Chatzimparmpas A. et al. Analyzing the Evolution of Javascript Applications //ENASE. – 2019. – С. 359-366.

11. Fuzzy Systems Modeling in Environmental and Health Risk Assessment. Ashok Deshpande, Rehan Sadiq, Boris Faybishenko. – 2023.

12. Функціональне програмування в JavaScript https://dou.ua/forums/topic/37548/

13. Огляд популярних javascript фреймворків https://conf.ztu.edu.ua/wp-content/uploads/2019/06/42.pdf

14. React спільнота https://www.facebook.com/react/

15. Огляд фреймворків JavaScript: https://dou.ua/forums/topic/34739/

16. Топ 18 фреймворков и библиотек JavaScript в 2021 https://www.plerdy.com/ru/blog/top-javascript-frameworks-and-libraries/

17. How Single Page Application works? https://www.droptica.com/blog/single-page-application-spa-what-it-and-when-should-you-use-it/

18. How to make React SEO-friendly: https://yalantis.com/blog/search-engine-optimization-for-react-apps/

19. Quasar Framework vs Vuetify https://stackshare.io/stackups/quasar-framework-vs-vuetify

20. Why not quasar support JSX? https://github.com/quasarframework/quasar/issues/7672

# ДОДАТОК

## main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
    enableProdMode();
}

platformBrowserDynamic()
    .bootstrapModule(AppModule)
    .catch(err => console.error(err));
```

## app.module.ts

```
import { NgModule } from '@angular/core';
import { A11yModule } from '@angular/cdk/a11y';
import { PortalModule } from '@angular/cdk/portal';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { ProgressSpinnerModule } from 'primeng/progressspinner';
import { ToastModule } from 'primeng/toast';
import { MessageModule } from 'primeng/message';

import { MainGuardChildService } from './shared/services/main-guard.service';
import { CRMComponentsModule } from './shared/components/components.module';
import { ApiService } from './shared/services/api.service';

import { AppComponent } from './app.component';
```

```
import { ROUTING } from './app.routing';
import { TranslocoRootModule } from './transloco-root.module';

@NgModule({
    declarations: [AppComponent],
    imports: [
        BrowserModule,
        BrowserAnimationsModule,
        HttpClientModule,
        ROUTING,
        TranslocoRootModule,
        ProgressSpinnerModule,
        A11yModule,
        PortalModule,
        ToastModule,
        MessageModule,
        CRMComponentsModule
    ],
    providers: [MainGuardChildService, ApiService],
    bootstrap: [AppComponent]
})
export class AppModule {}
```

## app.component.html

```
<crm-snackbar></crm-snackbar>
<router-outlet></router-outlet>
<div *cdk-portal class="global-loadmask-container" cdkTrapFocus>
    <p-progressSpinner></p-progressSpinner>
</div>
```

## app.component.ts

```
import { ChangeDetectionStrategy, ChangeDetectorRef, Component, ViewChild,
ViewEncapsulation } from '@angular/core';
import type { OverlayRef } from '@angular/cdk/overlay';
import { Overlay } from '@angular/cdk/overlay';
import { CdkPortal } from '@angular/cdk/portal';
```

```
    import { NavigationCancel, NavigationEnd, NavigationError, NavigationStart,
Router } from '@angular/router';
    import { AppUtils } from './shared/utils/app-utils';

    @Component({
        selector: 'crm-root',
        templateUrl: './app.component.html',
        styleUrls: ['./app.component.scss'],
        changeDetection: ChangeDetectionStrategy.OnPush,
        encapsulation: ViewEncapsulation.None
    })
    export class AppComponent {
        private overlayRef: OverlayRef;

        @ViewChild(CdkPortal) private portal: CdkPortal;

        constructor(private overlay: Overlay, private router: Router, private
cdRef: ChangeDetectorRef) {
            AppUtils.setRootComponent(this);

            this.router.events.pipe().subscribe(e => {
                if (e instanceof NavigationStart) {
                    this.showLoadmask();
                    return;
                }
                if (e instanceof NavigationEnd || e instanceof NavigationCancel
|| e instanceof NavigationError) {
                    this.hideLoadmask();
                }
            });
        }

        showLoadmask(): void {
            if (!!this.overlayRef && this.overlayRef.hasAttached()) {
                this.overlayRef.dispose();
            }
            this.overlayRef = this.overlay.create({
                positionStrategy: this.overlay.position().global(),
```

```
            hasBackdrop: true,
            backdropClass: null
        });
        this.overlayRef.attach(this.portal);
        this.cdRef.markForCheck();
    }

    hideLoadmask(): void {
        if (this.overlayRef) {
            this.overlayRef.dispose();
        }
        this.cdRef.markForCheck();
    }
}
```

## main.component.ts

```
import { ChangeDetectionStrategy, Component } from '@angular/core';
import { TranslocoService } from '@ngneat/transloco';

import { Constants } from '../shared/constants';
import { UserService } from '../shared/services/user.service';

interface MenuItems {
    icon: string;
    label: string;
    url: string;
}

@Component({
    selector: 'crm-main',
    templateUrl: './main.component.html',
    styleUrls: ['./main.component.scss'],
    changeDetection: ChangeDetectionStrategy.OnPush
})
export class MainComponent {
    readonly menuItems: MenuItems[] = [
        {
            icon: 'pi pi-th-large',
```

```
                label: '_dashboard',
                url: '/dashboard'
            },
            {
                icon: 'pi pi-user',
                label: '_users',
                url: '/users'
            },
            {
                icon: 'pi pi-sitemap',
                label: '_category',
                url: '/category'
            },
            {
                icon: 'pi pi-sliders-h',
                label: '_productAttributes',
                url: '/attributes'
            }
        ];

        get flagImg(): string {
            return this.activeLang === 'ua' ? Constants.UK_FLAG_URL :
Constants.UA_FLAG_URL;
        }

        get activeLang(): string {
            return this.translocoSrv.getActiveLang();
        }

        get userName(): string {
            return this.userSrv.myData.username;
        }

        constructor(public translocoSrv: TranslocoService, private userSrv:
UserService) {}

        changeLanguage(): void {
```

```
            this.translocoSrv.setActiveLang(this.activeLang === 'ua' ? 'en' :
'ua');
        }
    }
```

## main.routing.ts

```
import type { Routes } from '@angular/router';
import { RouterModule } from '@angular/router';
import type { ModuleWithProviders } from '@angular/core';

import { MainComponent } from './main.component';
import { MainGuardChildService } from '../shared/services/main-guard.service';

const lazy: Routes = [
    {
        path: 'dashboard',
        loadChildren: () => import('../dashboard/dashboard.module').then(m =>
m.DashboardModule)
    },
    {
        path: 'users',
        loadChildren: () => import('../users/users.module').then(m =>
m.UsersModule)
    },
    {
        path: 'category',
        loadChildren: () => import('../category/category.module').then(m =>
m.CategoryModule)
    },
    {
        path: 'attributes',
        loadChildren: () => import('../attributes/attributes.module').then(m =>
m.AttributesModule)
    }
];

const routes: Routes = [
    {
        path: '',
```

```
        component: MainComponent,
        canActivateChild: [MainGuardChildService],
        children: [
            {
                path: '',
                pathMatch: 'full',
                redirectTo: 'dashboard'
            },
            ...lazy.map(route => ({
                ...route
            })),
            {
                path: '**',
                redirectTo: 'dashboard'
            }
        ]
    }
];

export const ROUTING: ModuleWithProviders<RouterModule> =
RouterModule.forChild(routes);
```

## main.component.html

```
<ng-container *transloco="let t">
    <p-menubar class="w-full fixed" styleClass="bg-primary-500 border-none
border-noround p-3">
        <ng-template pTemplate="start">
            <div routerLink="/" class="font-bold text-white">CRM UI</div>
        </ng-template>
        <ng-template pTemplate="end">
            <div class="flex align-items-center">
                <div class="h-1rem flex align-items-center text-white mr-3">
                    <span class="mr-2">
                        <i class="pi pi-user"></i>
                    </span>
                    <p>{{ userName }}</p>
                </div>
```

```
                <img      class="flag      cursor-pointer"      [src]="flagImg"
[alt]="activeLang" (click)="changeLanguage()" />
            </div>
        </ng-template>
    </p-menubar>

    <p-sidebar    [visible]="true"    [modal]="false"    [showCloseIcon]="false"
[closeOnEscape]="false">
        <nav>
            <button
                *ngFor="let item of menuItems"
                pButton
                routerLinkActive="active"
                class="w-full p-button-text border-noround"
                [routerLink]="item.url"
                [label]="t(item.label)"
                [icon]="item.icon"
            ></button>
        </nav>
    </p-sidebar>
    <div class="content">
        <router-outlet></router-outlet>
    </div>
</ng-container>
```

## login.component.html

```
<crm-snackbar></crm-snackbar>
<div class="wrapper" *transloco="let t">
    <p-card class="w-full" [formGroup]="activeForm">
        <ng-container *ngIf="isLoginMode">
            <h3 class="text-center mt-0">{{ t('_singIn') }}</h3>
            <div class="p-inputgroup mb-2">
                <span class="p-inputgroup-addon">
                    <i class="pi pi-user"></i>
                </span>
                <input           formControlName="username"           pInputText
[placeholder]="t('_login')" />
            </div>
```

```
            <div class="p-inputgroup">
                <span class="p-inputgroup-addon">
                    <i class="pi pi-key"></i>
                </span>
                <p-password
                    styleClass="w-full"
                    formControlName="password"
                    [placeholder]="t('_password')"
                    [toggleMask]="true"
                    [feedback]="false"
                >
                </p-password>
            </div>
            <p-button      styleClass="w-full      mt-2"      (click)="login()"
[disabled]="loginForm.invalid" [label]="t('_singIn')">
            </p-button>
        </ng-container>

        <ng-container *ngIf="isRecoverMode">
            <h3 class="text-center mt-0">{{ t('_passwordRecovery') }}</h3>
            <div class="p-inputgroup mb-2">
                <span class="p-inputgroup-addon">
                    <i class="pi pi-user"></i>
                </span>
                <input           formControlName="username"           pInputText
[placeholder]="t('_login')" />
            </div>
            <div class="p-inputgroup mb-2">
                <span class="p-inputgroup-addon">
                    <i class="pi pi-envelope"></i>
                </span>
                <input    formControlName="email"    pInputText    type="email"
[placeholder]="t('_email')" />
            </div>
            <p-button
                styleClass="w-full mt-2"
                (click)="sendResetLink()"
                [disabled]="recoveryForm.invalid"
```

```
            [label]="t('_sendResetLink')"
        >
        </p-button>
    </ng-container>

    <ng-container *ngIf="isChangePasswordMode">
        <h3 class="text-center mt-0">{{ t('_newPassword') }}</h3>
        <div class="p-inputgroup mb-2">
            <span class="p-inputgroup-addon">
                <i class="pi pi-key"></i>
            </span>
            <p-password
                formControlName="password"
                styleClass="w-full"
                [placeholder]="t('_password')"
                [toggleMask]="true"
                [feedback]="false"
            >
            </p-password>
        </div>
        <div class="p-inputgroup mb-2">
            <span class="p-inputgroup-addon">
                <i class="pi pi-key"></i>
            </span>
            <p-password
                formControlName="passwordConfirm"
                styleClass="w-full"
                [placeholder]="t('_repeatPassword')"
                [toggleMask]="true"
                [feedback]="false"
            >
            </p-password>
        </div>
        <p-button
            styleClass="w-full mt-2"
            (click)="resetPassword()"
            [disabled]="changePasswordForm.invalid"
            [label]="t('_changePassword')"
```

```
            ></p-button>
        </ng-container>

        <p      *ngIf="!isChangePasswordMode"      class="text-center      link"
(click)="changeRecoverMode()">
            {{  isRecoverMode  ?  t('_rememberedPassword')  :  t('_forgotPassword')
}}
        </p>

        <p  (click)="changeLanguage()"  class="flex  align-items-center  justify-
content-center link">
            {{  t('_changeLanguage')  }} <img class="h-3rem ml-2" [src]="flagImg"
[alt]="activeLang" />
        </p>
    </p-card>
</div>
```

## login.component.ts

```
import { ChangeDetectionStrategy, Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { ActivatedRoute, Router } from '@angular/router';
import { TranslocoService } from '@ngneat/transloco';

import { AuthService } from '../shared/services/auth.service';
import { AppUtils } from '../shared/utils/app-utils';
import              {              SnackbarService              }              from
'../shared/components/snackbar/snackbar.service';
import { ValidationUtils } from '../shared/utils/validation-utils';
import { Constants } from '../shared/constants';

enum PageMode {
    Login,
    Recovery,
    ChangePassword
}

@Component({
    selector: 'crm-login',
```

```
    templateUrl: './login.component.html',
    styleUrls: ['./login.component.scss'],
    changeDetection: ChangeDetectionStrategy.OnPush
})
export class LoginComponent implements OnInit {
    activeMode: PageMode = PageMode.Login;

    loginForm: FormGroup = this.formBuilder.group({
        username: ['', Validators.required],
        password: ['', [Validators.required, Validators.minLength(6)]]
    });

    recoveryForm: FormGroup = this.formBuilder.group({
        username: ['', Validators.required],
        email: ['', [Validators.required, Validators.email]]
    });

    changePasswordForm: FormGroup = this.formBuilder.group(
        {
            password: ['', [Validators.required, Validators.minLength(6)]],
            passwordConfirm:              ['',              [Validators.required,
Validators.minLength(6)]]
        },
        {
            validators:                 ValidationUtils.passwordMatch('password',
'passwordConfirm')
        }
    );

    get isLoginMode(): boolean {
        return this.activeMode === PageMode.Login;
    }

    get isRecoverMode(): boolean {
        return this.activeMode === PageMode.Recovery;
    }

    get isChangePasswordMode(): boolean {
```

```
        return this.activeMode === PageMode.ChangePassword;
    }

    get activeForm(): FormGroup {
        return this.isLoginMode ? this.loginForm : this.isRecoverMode ?
this.recoveryForm : this.changePasswordForm;
    }

    get flagImg(): string {
        return this.activeLang === 'ua' ? Constants.UK_FLAG_URL :
Constants.UA_FLAG_URL;
    }

    get activeLang(): string {
        return this.translocoSrv.getActiveLang();
    }

    constructor(
        private formBuilder: FormBuilder,
        private authSrv: AuthService,
        private snackbarSrv: SnackbarService,
        private router: Router,
        private route: ActivatedRoute,
        private translocoSrv: TranslocoService
    ) {}

    ngOnInit(): void {
        this.route.queryParams.subscribe(param => {
            if (param?.['recover']) {
                this.activeMode = PageMode.ChangePassword;
            }
        });
    }

    changeRecoverMode(): void {
        this.activeMode = this.isLoginMode ? PageMode.Recovery : PageMode.Login;
    }
```

```
    login(): void {
        AppUtils.showLoadmask();
        this.authSrv.login(this.loginForm.value).subscribe(res => {
            AppUtils.hideLoadmask();

            if (res.success) {
                this.router.navigate(['/']);
            } else {
                this.snackbarSrv.error(res.faultString);
            }
        });
    }

    sendResetLink(): void {
        AppUtils.showLoadmask();
        this.authSrv.sendResetLink(this.recoveryForm.value).subscribe(res => {
            AppUtils.hideLoadmask();

            if (res.success) {
                this.snackbarSrv.success('_resetSuccess');
            } else {
                this.snackbarSrv.error(res.faultString);
            }
        });
    }

    resetPassword(): void {
        AppUtils.showLoadmask();
        this.authSrv.resetPassword(this.changePasswordForm.value).subscribe(res
=> {
            AppUtils.hideLoadmask();

            if (res.success) {
                this.snackbarSrv.success('_passwordChanged');
                this.activeMode = PageMode.Login;
            } else {
                this.snackbarSrv.error(res.faultString);
            }
```

```
        });
    }

    changeLanguage(): void {
        this.translocoSrv.setActiveLang(this.activeLang === 'ua' ? 'en' : 'ua');
    }
}
```