Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра мультимедійних систем

РОЗРОБКА КЛІЄНТ-СЕРВЕРНОГО ВЕБ-ЗАСТОСУВАННЯ З ВИКОРИСТАННЯМ ФРЕЙМВОРКІВ SPRING BOOT ТА ANGULAR

Текстова частина до курсової роботи
за спеціальністю 121 «Інженерія програмного забезпечення»

Керівник курсової роботи
ст.в.___Борозенний С.О.___
(прізвище та ініціали)
_____
(підпис)
“_ __ _” _____ 2021 р.

Виконав студент Папроцький І.А.
(прізвище та ініціали)
_____
(підпис)
“_ __ _” _____ 2021 р.

Київ-2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Завідувач кафедри мультимедійних систем,

канд. фіз-мат. наук, доц. – Жежерун О.П.

_____ *(підпис)*

“____” _____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Папроцькому Ігорю Андрійовичу

Факультету інформатики 4 р.н. бакалаврської програми
ТЕМА: Розробка клієнт-серверного веб-застосування з використанням фреймворків Spring Boot та Angular

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Огляд теоретичного матеріалу та здійснення дослідження

Висновки

Список використаної літератури та посилань

Додатки (за необхідністю)

Дата видачі “____” _____ 2020 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Тема: Розробка клієнт-серверного веб-застосування з використанням фреймворків Spring Boot та Angular

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	25.10.2020	
2.	Пошук тематичної літератури	21.11.2020	
3.	Ознайомлення з літературою	25.12.2020	
4.	Вивчення аналогів	15.01.2021	
5.	Проектування та створення бази даних	20.01.2021	
6.	Створення віддаленого директорію та налаштування локального та віддаленого середовищ розробки та розгортання застосунку	30.01.2021	
7.	Створення серверної частини веб-застосування використовуючи фреймворк Spring Boot	20.02.2021	
8.	Створення клієнтської частини веб-застосування використовуючи фреймворк Angular	25.03.2021	
9.	Написання текстової частини	05.04.2021	
10.	Перегляд змісту роботи керівником	10.04.2021	
11.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	10.04.2021	
12.	Створення презентації	10.04.2021	
13.	Захист роботи	17.04.2021	

ЗМІСТ

АНОТАЦІЯ

У роботі розглянуті деякі аспекти розробки веб-застосувань використовуючи фреймворки Spring Boot та Angular. Основна увага зосереджена на деталях реалізації веб-застосунків використовуючи поєднання зазначених інструментів розробки. Практична частина роботи зосереджена на реалізації веб-сайту для мережі магазинів які прагнуть запровадити систему роботи з клієнтами за принципом: клієнт замовляє набір товарів онлайн та приходить на точку видачі товарів, забираючи своє замовлення та зменшуючи можливість контакту з іншими людьми, а також не витрачаючи зайвого часу на черги та рух у скупченні людей всередині магазину.

У першому розділі розглянута предметна область застосування а також проаналізовано ринок та конкурентні веб-застосування. Також у цьому розділі відбувається постановка завдання курсової роботи.

У другому розділі розглянуті деякі теоретичні аспекти та відомості про сучасні тенденції вибору фреймворків для розробки фронтенду та бекенду. Також описано деякі особливості розробки сучасних бізнес-застосувань у контексті вибору архітектури, їх підтримки та розширення. Детально розглянуто використані бібліотеки та принципи побудови архітектури застосунків та їх окремих частин. Аргументовано вибір фреймворків Spring Boot та Angular для реалізації поставленої на практичну частину роботи завдання.

У третьому розділі розглядається практична частина курсової роботи. Детально описуються реалізації як клієнтської частини (реалізованої за допомогою Angular), так і серверної частини (реалізованої за допомогою Spring Boot). Також аналізуються та оглядаються результати роботи і виводяться висновки.

ВИКОРИСТАНІ СКОРОЧЕННЯ

Фреймворк – певна готова інфраструктура програмного рішення, програмний каркас, який дозволяє легше та швидше розроблювати нові програмні рішення.

Spring Boot – фреймворк на мові Java, який використовується для написання веб-застосувань.

Бекенд – серверна частина веб-застосунку.

Фронтенд – клієнтська частина веб-застосунку.

Enterprise - (у пер. з англ. – підприємство, ділова справа) – той, що створюється та функціонує для роботи та прибутку підприємств будь-якої величини.

IT – інформаційні технології.

CRM-система – (Customer Relationship Management System) система управління відносинами з клієнтами. Використовуються компаніями та підприємствами для управлінням процесів продажу товарів клієнтам, а також надання послуг, збирання інформації та статистики та ін.

Стек технологій – набір технологій проекту, який зазвичай визначає – які фреймворки/бібліотеки/мови програмування/СУБД/серверні рішення використовуватимуться командою розробників.

URL – (Uniform Resource Locator) уніфікований локатор ресурсу – стандартизована адреса ресурсу у мережі. У контексті даної курсової роботи буде вживатись майже як синонім до **URI** (Uniform Resource Identifier) – є компактною послідовністю символів що ідентифікує абстрактний або фізичний ресурс. [11]

JSON – (JavaScript Object Notation) – текстовий спосіб передачі даних у вигляді інформації про JavaScript об’єкт.

CRUD – (Create-Read-Update-Delete) – стандартний набір операцій над ресурсом/даними який позначає, відповідно, створення інформації, читання інформації, оновлення даних та видалення.

API – (Application Programming Interface) – прикладний програмний інтерфейс, набір засобів та інструментів які дають можливість звертатись до окремих частин застосування.

DTO (Data Transfer Object) – об’єкт що використовується для передавання інформації між різними рівнями застосунку.

POJO (Plain Old Java Object) – простий Java-об’єкт який не успадковує жодних інтерфейсів окрім необхідних для реалізації бізнес-логіки.

ВСТУП

Актуальність та практичне значення обраної теми

Серед сучасних особливостей та способів розробки програмного забезпечення існує велика різноманітність мов програмування, інструментів, бібліотек та фреймворків які суттєво полегшують життя програміста. Як відомо – «час – це гроші», і цей вислів є надзвичайно важливим для IT-галузі. Тема покращення ефективності розробки програмного забезпечення завжди інтересувала усіх причетних до діяльності в IT. Саме тому люди створюють все новіші та новіші фреймворки які дозволяють розробникам позбавити себе необхідності повторювати шаблонні дії для забезпечення хоча б просто роботоздатності кожного наступного проєкту у якому вони беруть участь. Саме тому для даної курсової роботи вирішено розглянути особливості розробки веб-додатків на основі надзвичайно популярних у сьогоденні фреймворків – Spring Boot для бекенду та Angular для фронтенду. Оскільки найкращим способом розібратись у системі є спроба її реалізації власноруч – цим і заплановано займатись у практичній частині цієї роботи.

Структура роботи

Робота складається із вступу, трьох розділів, висновку, списку використаних джерел та набору додатків.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАВДАННЯ КУРСОВОЇ РОБОТИ

1.1 Базовий огляд предметної області

Серед сучасних проблем світу важко не помітити проблему епідемії COVID-19 та тенденції людства до вимушеності рухатись у напрямку діджиталізації усіх процесів навколо себе (окрім природнього розвитку Інтернету) задля зменшення фізичних контактів людей між собою. Це, в свою чергу, дає безліч додаткових можливостей для росту та розвитку світу IT. Серед цих можливостей варто зазначити величезний попит на створення все більшої кількості застосувань, сайтів, мобільних додатків тощо.

Попри те, що за даними Google Analytics у 2020 році більше половини інтернет-трафіку становив мобільний [1], і, здавалося б, мобільні додатки завойовують все більшу долю трафіку мережі Інтернет, – не варто забувати і про те, що все ще актуальними є веб-сайти, які можуть універсально обслуговувати користувачів з будь-якими пристроями. Це дає можливість підприємствам (а особливо малого та середнього бізнесу) зосередити свою увагу на розробці веб-сайтів для економії часу та грошей на розвиток своєї справи. Не варто також і забувати про можливість реалізації серверних частин веб-застосувань, які можуть обслуговувати одночасно як мобільні додатки, так і так звану «веб-складову» інтернет-бізнесу.

Цей факт є цікавим для нас тому, що з кожним наступним роком у світі розвивається все більше встановлених практик розробки різноманітних додатків, застосувань та програм які виконують найрізноманітніші потреби людства. Багато таких практик включає в себе використання популярних бібліотек та фреймворків які дозволяють відчутно зменшити кількість написання шаблонного коду, який, за відсутності таких бібліотек, був би необхідний, наприклад, лише для можливості мінімальної роботи застосунку. Кількість такого коду можна прямо пропорційно співставити з кількістю часу, а отже і

кількістю втрачених грошей, які могли б компанії-розробники потенційно заробити замість трати часу на написання одного і того ж коду.

Як результат роздумів над цією темою було вирішено дослідити особливості розробки веб-застосування використовуючи одні з надзвичайно популярних у сьогоденні фреймворків Spring Boot та Angular. Кожен з них представляє, відповідно, інструмент створення серверної та клієнтської частин застосунку. Аргументація вибору саме цих інструментів розглядатиметься у наступному розділі.

Для зазначеної цілі було придумано створити веб-застосування у вигляді CRM-системи для мережі магазинів, яке б дозволяло, з однієї сторони, покупцям робити замовлення продукції для подальшого самовивозу товарів із конкретних точок видачі (які можуть знаходитись, наприклад, на вході в магазин), а з іншої сторони, працівникам та адміністрації магазину наповнювати різні категорії продукції наявними товарами, а також обробляти самі замовлення. За належного рівня маркетингу та правильного менеджменту ця ідея може бути дуже успішною в реальному світі. Особливої актуальності цьому застосунку надає ситуація поширеності епідемії COVID-19: багато людей воліли б якомога менше контактувати з іншими персонами або купувати товари які не є «обмацаними» іншими людьми. Також це дозволяє зекономити час стояння у чергах на касі.

Такий сайт був би чудовою можливістю розглянути розробку комплексного застосунку з різними ролями користувачів у системі а також різними типами даних. Назвемо наш застосунок «OrderMe».

1.2 Аналіз існуючих аналогів реалізації предметної області на ринку

Серед існуючих аналогів вищезазначеної системи в Україні одним із найпопулярніших є сайт Zakaz.ua. Власники сайту співпрацюють із деякими крупними магазинами в Україні. Серед переваг сайту можна замітити приємний для користувача інтерфейс, всі необхідні можливості для замовлення та обробки

замовлень а також гарні можливості для показу знижок та акцій, що є немалозначною складовою маркетингової кампанії будь-якого магазину. Можна замітити простоту та інтуїтивність функціоналу, деякі функції якого безумовно необхідні і у реалізації практичної частини цієї курсової роботи.

Також до списку схожих аналогів можна віднести й сайт Cooker. Як і попередній кандидат, вони роблять акцент на доставці товарів до покупця. На відміну від цих систем, ця робота буде орієнтована на можливості самовивозу товарів покупцями. Звісно, це не забороняє додавати функціонал доставки товарів у застосунок у майбутньому.

1.3 Постановка завдання курсової роботи

Отже, відповідно до поставленої цілі можна виокремити такі завдання для цієї курсової роботи:

1. розробити веб-застосунок у вигляді CRM-системи для мережі магазинів;
2. система повинна мати такі елементи функціоналу:
    a. аутентифікація та авторизація, робота з ролями;
    b. можливість додавати, змінювати та видаляти категорії товарів адміністраторами;
    c. можливість додавати, змінювати та видаляти товари, які можна також додавати та переміщувати між категоріями адміністраторами;
    d. можливість управляти статусами замовлень адміністраторами;
    e. можливість переглядати категорії товарів та товари;
    f. можливість додавати товари до кошику у конкретній кількості, управляти своїм кошиком (змінювати його наповнення, створювати замовлення із поточним вмістом кошику) клієнтами (або потенційними клієнтами) магазину;

   g. можливість переглядати свої замовлення, змінювати їх наповнення якщо статус замовлення ще не переведений адміністраторами у статус «В обробці».
3. використати фреймворки Spring Boot та Angular для розробки зазначеного вище веб-додатку;
4. проаналізувати особливості налаштування, розробки та можливості обраних фреймворків виконувати поставлені задачі;
5. виокремити переваги та недоліки обраного підходу;
6. зробити висновки щодо особливостей використання даного підходу базуючись на результатах попередніх пунктів.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Огляд сучасних тенденцій вибору стеку технологій для створення веб-застосунків

Як вже було сказано, у світі програмного забезпечення красується величезна кількість різноманітних рішень розробки та архітектури веб-додатків. Велике значення для ефективності розробки має вибір командою розробників мов програмування, які використовуватимуться на проекті. Це дає не лише можливості наймати в команду нових вже готових розробників-знавців конкретних технологій зв’язаних з цією мовою програмування, а і обирати подальші напрямки розвитку проекту.

Варто підмітити, що вибір стеку технологій в основному залежить від наступних факторів [2]:

- вимоги до проекту та функцій проекту, розмір проекту;
- ресурси та досвід розробників;
- можливості для розширення (Scalability);
- можливості подальшої підтримки проекту;
- необхідного захисту застосунку;
- необхідної швидкості розробки проекту.

2.1.1 Огляд найпопулярніших фреймворків розробки бекенду сьогодення

Перейдемо до теми бекенду. Як відомо, серверною частиною застосунку у клієнт-серверній архітектурі вважається та частина веб-додатку, яка відповідає за обробку даних, надісланих користувачем, та можливого збереження цієї інформації в бази даних, відправки її іншим застосуванням, виконання певних окремих одноразових чи періодичних дій з цими даними тощо.

Звісно, реальні серверні застосування для використання у бізнесі не пишуться програмістами просто звичайними SDK (з англ. – Software

Development Kit – набір стандартних бібліотек для мови програмування). Жодна людина навряд погодиться «винаходити велосипед», і тому у реальних проектах обов'язково використовуються набори бібліотек та інструментів для зменшення кількості так званого шаблонного коду – усього коду необхідного для запуску працездатного веб-застосування.

Серед найпопулярніших фреймворків бекенду, станом на 2020-2021 роки [3], можна виділити Laravel на мові PHP, Express.js на мові JavaScript (платформа Node.js), ASP.NET на мові C#, Ruby on Rails на мові Ruby, фреймворки Django та Flask на мові Python та Spring Boot на мові Java (та Kotlin яка базується на Java). Безумовно кожен з них має свої переваги та недоліки які виражаються у особливостях мов програмування, на яких вони написані, та у різних можливостях, які вони пропонують розробникам для швидкого та зручного створення та налаштування робочого веб-застосунку.

Для прикладу, фреймворк Django відомий низьким порогом входу у розробку, готовою реалізацією захисту застосування та автоматичною генерацією так званих контроллерів – класів які мають методи, які викликаються при входженні запитів клієнтів у застосунок. Express.js відомий через легку реалізацію та популярність платформи Node.js. Ruby on Rails, в свою чергу, відомий через легкість освоєння новачками та великий потенціал розширення через бібліотеки-залежності. Laravel знаний як фреймворк для однієї з найпопулярніших мов програмування для веб-застосувань – PHP, і через хорошу документацію. [4]

На кінець, поговоримо про Spring Boot. **Spring Boot** – це програмний каркас, який, в свою чергу, побудований на платформі-фреймворку Spring. Однією з головних властивостей Spring Boot є те, що в ньому вже є все готове для дії «просто запусти». Тобто, перевагою Spring Boot є те, що він налаштований на те, щоб користувачу було максимально легко сконфігурувати своє застосування. В ньому вже є вбудовані сервери: широко відомий Tomcat а також інші – і Jetty та

Underflow. «З коробки» в Spring Boot легко додаються так звані залежності-стартери, тобто надбудови над бібліотеками та збірками бібліотек, які дуже часто досить просто додати в список залежностей – і вони вже працюють, маючи автоконфігурації, які, звісно ж, можна перевизначити на власний розсуд та манер. [5]

Деталі та можливості Spring Boot будуть розглянуті в наступних підрозділах

2.1.2 Огляд передових фреймворків побудови клієнтської частини веб-застосувань

Як ми можемо замітити зайшовши на будь-який сучасний якісний сайт (під «якісним» розумітимемо той, який не має дизайну та функціоналу рівню часів початку розвитку мережі Інтернет), світ вже давно використовує динамічні сторінки на вебсайтах замість статичних сторінок. Це означає, що все менше веб-застосунків використовує стару схему роботи вебсайту шляхом відправлення сервером користувачу просто готової статичної HTML сторінки. При цьому не кожна дія або натиснення викликає перезавантаження поточної сторінки браузера. Рішення досягається механізмами асинхронності та реактивності, які широко використовуються у сучасних фреймворках фронтенду.

Отож, на 2021 рік можна виділити найпопулярніші та найвідоміші фронтенд-фреймворки: Angular, React та Vue.js. При цьому React насправді не є фреймворком, а є бібліотекою для фронтенду, хоча багато розробників вважають її саме фреймворком і вона зазвичай використовується для порівняння з іншими у цьому контексті. [4]

React першим з трьох перелічених фреймворків почав адаптуватися до принципу побудови архітектури фронтенд-застосування на основі компонентів. React став популярним через свої можливості швидкої маніпуляції даними на сторінці маючи віртуальний DOM (з англ. – Document Object Model – об’єктна модель яка використовується для побудови веб-сторінки. Також пов’язана з

принципом DOM-дерева, тобто дерева елементів на сторінці), а також через простий і легкий синтакс, представлений бібліотекою JSX. Також він є популярним через те, що його створила компанія Facebook, після чого почала використовувати, наприклад, як на самому сайті Facebook, так і в Instagram. [4]

В свою чергу Vue.js є відносно недавно створеним фреймворком (2014 рік). [6] За роки існування фреймворку він дуже швидко і ефективно розвивався, через що його почали підтримувати багато великих компаній. Він є прогресивним, тобто таким, який може бути легко адаптованим до вже існуючих проектів. Також він має компонентну архітектуру, реалізацію реактивності у вигляді простих JavaScript об’єктів та інші переваги. [4]

І, нарешті, розглянемо Angular. **Angular** – це надзвичайно потужний фронтенд фреймворк, побудований на мові TypeScript – версії JavaScript. Як платформа Angular містить [7]:

- фреймворк що базується на компонентах, який призначений для побудови розширюваних веб-застосувань; [7]
- колекцію добре-інтегрованих бібліотек які покривають широку різноманітність функцій включаючи роутинг (маршрутизація всередині застосунку), менеджмент форм, клієнт-серверну комунікацію та ін.; [7]
- набір інструментів для розробника для допомоги останнім розробляти, тестувати та оновлювати власний код. [7]

До переваг Angular можна віднести також те, що застосунки побудовані на ньому легко створюються як для малих проектів, так і для великих копоративних бізнес-рішень. Точно варто додати про надзвичайно широку і зручну документацію Angular, набір інструкцій та навчальних матеріалів які легко читаються та освоюються. Angular підтримується компанією Google.

Особливості та деталі цього фреймворку будуть розглянуті у наступних підрозділах.

2.2 Огляд клієнт-серверної архітектури

Для цієї курсової роботи використовуватимемо клієнт-серверну архітектуру.

Клієнт-серверна архітектура – це обчислювальна модель в якій сервер розміщує, постачає та управляє більшістю ресурсів та послуг, які споживає клієнт. У цьому типі архітектури один чи більше клієнтських комп'ютерів або пристроїв є підключеними до серверу через мережу або підключення Інтернет. У цій моделі всі запити та сервіси доставляються через мережу. [8]

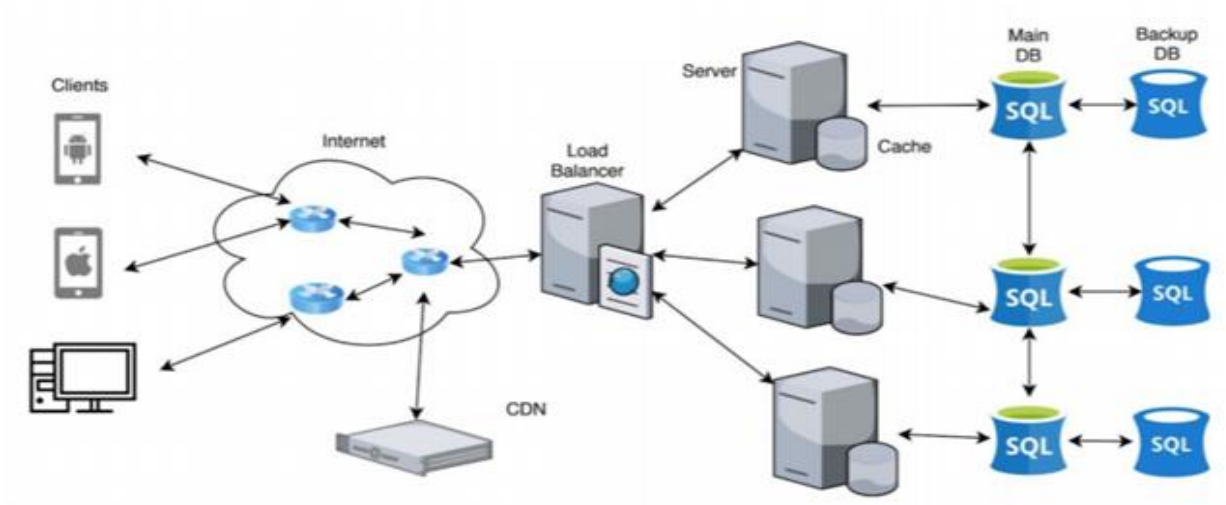


Рисунок 1. Представлення клієнт-серверної архітектури [8]

На Рисунку 1 зображена схема зв'язків між пристроями у клієнт-серверній архітектурі. Веб-застосунок розміщується на сервері (на рисунку – Server), який може мати свої кеші (або можуть бути створені окремі кеш-сервери). Сервер комунікує з серверами баз даних (на рисунку – Main DB (головна БД) та Backup DB (запасна БД)) на яких встановлені системи керування базами даних для збереження інформації необхідної для предметної області у БД. Задля вирішення проблеми навантаженості системи під час великої кількості трафіку на сервері, можуть бути запроваджені додаткові міри – балансувальники навантаження (на рисунку – Load Balancer), горизонтальне та вертикальне розширення. Клієнт (на рисунку - Client) же комунікує із сервером за допомогою запитів через мережу

(зазвичай – Інтернет). При цьому потік даних зазвичай формує цикл – клієнт посилає запит на сервер, на що сервер дає відповідь. Клієнти ніяк не взаємодіють між собою напряму у цій архітектурі. [8]

Горизонтальне розширення – це принцип збільшення кількості серверів задля розподілення навантаження між ними.

Вертикальне розширення – це принцип покращення серверів – заміна їх на більш потужні та швидші екземпляри.

Клієнт-серверна архітектура є надзвичайно поширеною і має багато переваг, серед яких: легка розширюваність, можливості підтримання контролю за мінімізацією часу відмови системи та ін. [8]

У нашій практичній роботі ми базуватимемось на цій моделі, але матимемо деяку зміну до схеми на представленому раніше рисунку: оскільки застосунок Angular запускається як окреме застосування (називатимемо його сервером фронтенду), то ми матимемо взаємодію не «клієнт-мережа-сервер-БД», а «клієнт-мережа-сервер фронтенду-сервер бекенду-БД». Таким чином, клієнт зі свого браузеру посилатиме запит на сервер фронтенду, який буде повертати йому HTML сторінку, стилі та скрипти. Клієнт виконуватиме певні дії на цій веб-сторінці, які будуть викликати ті ж скрипти, які, в свою чергу, будуть посилати запити на сервери бекенду та отримувати відповідь.

2.3 Опис принципів REST

Для реалізації практичної частини курсової роботи використовується принципи REST.

REST – розшифровується як **RE**presentational **S**tate **T**ransfer (у дослівному перекладі з англійської – «передача стану представлення»). REST є архітектурним стилем притаманним для розподілених систем гіпермедіа і

вперше був представлений світу Роєм Філдингом у 2000 році у його відомій дисертації. [9]

У REST система функціонує з елементами даних, які представлені на Рисунку 2:

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Рисунок 2. Елементи даних REST [10]

Найголовнішим елементом абстракції є Ресурс (на Рисунку 2 – resource), який є ціллю звертання до системи. [10] Будь-яка інформація, яка може бути названа може бути ресурсом. Таким чином, ресурсом можуть бути як документи чи зображення, невіртуальні об’єкти, так і колекції ресурсів. [10] Ресурси відрізняють між собою за допомогою спеціальних ідентифікаторів (на рисунку – resource indentifier), прикладами якого є URL. Представлення ресурсу (representation) може бути будь-яким для зручного пересилання інформації – HTML-документ, зображення, або JSON. Разом з інформацією про представлення ресурсу сервером або клієнтом можуть бути надані певні метадані про представлення (representation metadata), сам ресурс (resource metadata) або дані для контролю якості контенту під час пересилання (control data). Варто також зазначити що представлення ресурсу часто іменується як **media type**, тобто тип медіаданих.

Із дисертації Роя Філдінга виділено деякі принципи, головними з яких є [9]:

- клієнт-серверність – принцип призначений головним чином для клієнт-серверної архітектури веб-застосування;
- відсутність контексту стану (stateless) – кожен запит клієнта до сервера повинен містити всю необхідну інформацію для того щоб сервер «зрозумів» запит та міг його правильно опрацювати; запит не може брати переваг з будь-якого збереженого контексту на сервері; стан сесії повинен зберігатись у клієнта;
- уніфікований інтерфейс – повинен використовуватись принцип узагальнення компонентного інтерфейсу; у REST використовується 4 обмеження інтерфейсу: ідентифікація ресурсів, маніпуляція ресурсів через представлення; само-описувані повідомлення та гіпермедіа як рушій стану застосунку;
- багаторівнева (layered) система – стиль багаторівневої системи дозволяє архітектурі бути складеною з ієрархічних рівнів обмежуючи поведінку компонентів таким чином, щоб кожен компонент не міг «бачити» далі ніж найближчий рівень з яким він взаємодіє;
- (необов’язковий) код на вимогу (code on demand) – REST дозволяє функціональності клієнта бути розширеною через завантаження та виконання додаткових скриптів.

Принципами REST закладено використання певних ресурсних методів для досягнення виконання потрібної дії. У даному випадку використовуються методи HTTP. Для стандартних CRUD-операцій існують методи, відповідно, POST, GET, PUT/PATCH, DELETE.

API застосунку, яке створюють за всіма принципами REST називають RESTful API.

2.4 Огляд особливостей фреймворку Spring Boot та платформи Spring

2.4.1 Spring Framework як платформа

Перейдемо до огляду обраного фреймворку для реалізації бекенду. Як вже було сказано, Spring Boot оснований на платформі Spring.

Spring Framework – це фреймворк та платформа яка надає комплексну модель для програмування та конфігурації сучасних корпоративних рішень на будь-якому типі платформи розгортання застосунку. [12]

Ключовим елементом Spring є інфраструктурна підтримка рішення на рівні застосування: Spring забезпечує командам розробки корпоративних застосувань можливість сфокусуватись на розробці бізнес-логіки рівня застосування уникаючи непотрібної витрати часу на занадто громіздкі налаштування підстроювання під специфічні оточення.

Серед ключових функцій Spring виділяється [12]:

- **основні технології**: механізм ін’єкції залежностей, робота з подіями та їх обробкою у застосуванні, а також ресурсами, інтернаціоналізацією (i18n), різними видами валідації даних, зв’язування даних, перетворювання типів, спеціальна мова виразів SpEL (Spring Expressions Language), вбудовані механізми АОП (Аспектно-орієнтованого програмування);
- **інструменти тестування**: інструменти створення mock-об'єктів, фреймворки та бібліотеки TestContext, Spring MVC Test, WebTestClient та інші;
- **робота з даними**: підтримка транзакцій, підтримка DAO (Data Access Object), JDBC, бібліотеки для роботи з ORM (Object-Relational Mapping) та ін.;
- **web-фреймворки**: Spring MVC, Spring WebFlux;
- **інтеграція**: механізми віддаленої роботи з кодом, JMS (Java Message Service – Java API для роботи з Message-Oriented Middleware – механізмом

роботи з використанням асинхронного обміну даними між застосуваннями), фреймворк для роботи з криптографією JCA (Java Cryptography Architecture), технологія управління класами, застосуваннями, пристроями та комп'ютерними мережами JMX (Java Management Extensions), можливості відправки та роботи з email, завданнями (tasks), запланованими завданнями, кешами;

- **підтримка мов програмування**: Java, Kotlin, Groovy, динамічні мови програмування.

Найголовніша особливість Spring – велика кількість модулів для найрізноманітніших потреб розробників. Серед різноманітності Spring-модулів варто виокремити, напевне, найвідоміші:

- **Spring Cloud** – допомагає швидко і гнучко побудувати проект із реалізованими патернами розподілених систем (такі як Configuration Management, Service Discovery, Circuit Breakers, Intelligent Routing, Micro-Proxy, Control Bus, Global Lock, Leadership Election, Distributed Sessions та ін.). [12] Все це допомагає налаштувати роботу сервісу в розподіленій системі уникаючи тисячі рядків шаблонного коду, який потрібно було б написати без наявного фреймворку. Spring Cloud часто використовується в застосунках побудованих на **мікросервісній архітектурі.**
- **Spring Data** – запроваджує легкі в освоєнні та дуже різноманітні інструменти роботи з доступом до даних з різних джерел зберігання інформації. Серед них: робота з реляційними та нереляційними базами даних, технологіями доступу до даних (в тому числі – хмарними). Spring Data є великим проєктом із великою кількістю підпроєктів, специфічних для різних джерел та інструментів роботи з даними. [12]
- **Spring Security** – надає величезну кількість інструментів захисту застосунку, в тому числі – робота з аутентифікацією, авторизацією. Є де-

факто стандартом роботи з захистом у застосуваннях побудованих на платформі Spring.

Ще одна з головних особливостей Spring – механізм **Inversion of Control**, тобто інверсія контролю, який використовує патерн Dependency Injection.

Inversion of Control (IoC) – це принцип у розробці програмного забезпечення який передає контроль об’єктами або частинами застосунку спеціальному контейнеру або фреймворку. Зазвичай цей принцип використовується в контексті об’єктно-орієнтованого програмування. На відміну від традиційного програмування, де наш код викликає бібліотеки, IoC дозволяє фреймворку брати контроль за подіями у застосуванні і викликати наш код. Для цього фреймворки використовують абстракцію із готовою вбудованою поведінкою. Для того щоб додати власну поведінку, розробник повинен розширити класи фреймворку або вбудувати свої власні класи. [13]

Dependency Injection («Ін’єкція залежностей») **–** це патерн, який використовується для реалізації IoC. Інвертований контроль над застосунком використовує залежності між частинами застосування для виконання необхідної поведінки програми. [13]

У Spring Framework існує інтерфейс **ApplicationContext**, який представляє собою IoC контейнер. Spring-контейнер відповідальний за ініціалізацію, конфігурацію та управління циклом існування об’єктів, знаних як Bean. Усі необхідні частини застосування у виглядів необхідних класів сервісів, конфігурацій та інших компонент оголошуються як Spring Bean.

2.4.2 Spring Boot як розширення Spring Framework

Перейдемо до Spring Boot. Як вже було сказано, Spring Boot побудований на базі Spring Framework. Після ознайомлення із фреймворком, можна зрозуміти що його завдання – максимізувати стабільність та швидкість розробки за мінімальний час. Де-факто, це надбудова над Spring, яка має багато можливостей

конфігурації «під капотом» застосунку з широким полем для кастомізації та перевизначення конфігурацій.

На відміну від Spring Framework який використовує XML-конфігурації, Spring Boot вживає механізм Java-анотацій для налаштування застосунку. За допомогою останніх виконується надзвичайно багато операцій у застосунку на Spring Boot: оголошення Bean-ів, задання деяких значень конфігурації, робота з профілями та ін.

Також, для створення та ініціалізації нового Spring Boot проекту існує офіційна платформа Spring Initializr, яка дозволяє швидко створити проект із всіма необхідними залежностями, засобом автоматичного збирання (Maven/Gradle), на обраній мові (Java/Kotlin/Groovy), необхідної версії Spring Boot, необхідними метаданими та ін. прямо із вікна браузера. Приклад можна побачити на Рисунку 3:

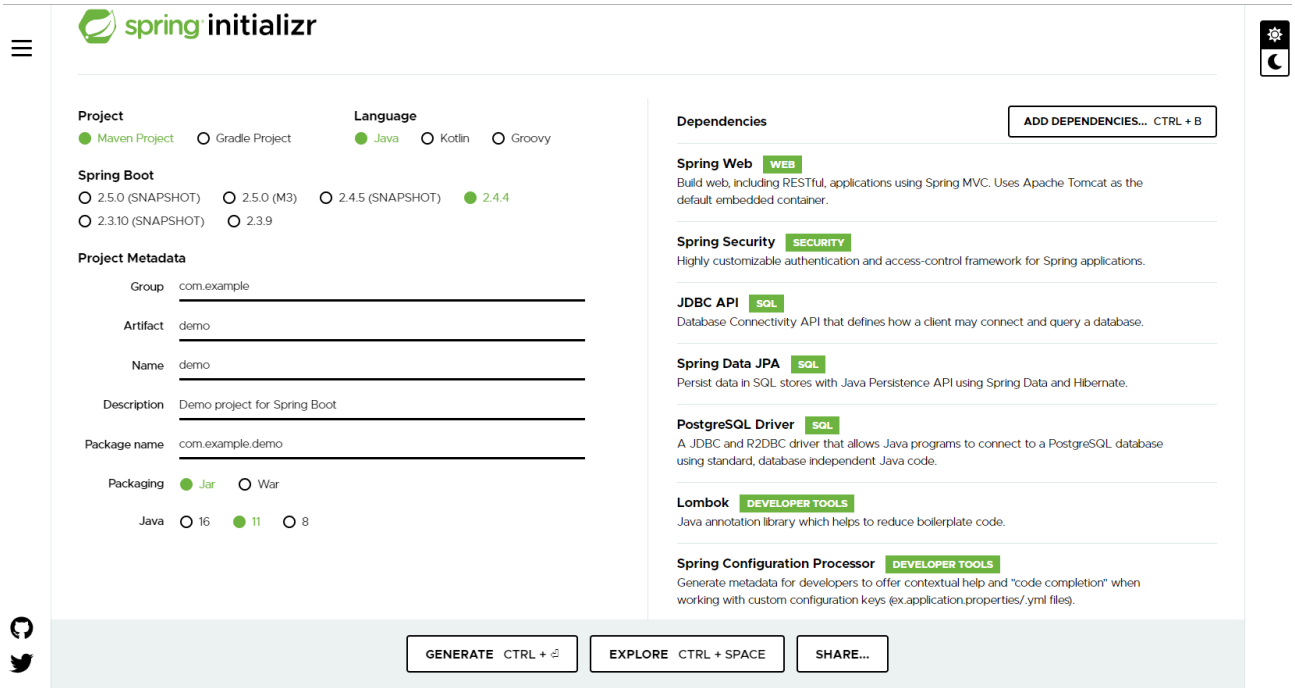


Рисунок 3. Скріншот створення Spring Boot проекту за допомогою утиліти Spring Initializr

Перейдемо до модулів та залежностей. Серед необхідних нам залежностей варто розглянути такі як Spring Web MVC, Spring Security, Spring Boot JDBC та Spring Data JPA.

Розпочнемо із **Spring Web MVC**. Це фреймворк, оснований на моделі Model-View-Controller який оснований навколо інтерфейсу DispatcherServlet. Останній опрацьовує запити до API застосунку яке створюється на основі анотацій @Controller та @RequestMapping. [14] Ці, та інші Java-анотації використовуються для налаштування необхідних методів так званих класів-контроллерів (Controller), в яких і створюються всі необхідні точки входу до API застосунку. Якщо узагальнити, то фреймворк надає всі необхідні можливості для створення RESTful API.

Перейдемо до **Spring Security**. Це фреймворк на базі Spring який дає комплексні можливості по налаштуванню захисту застосунку. Серед можливостей є робота з ролями та фільтрами запитів, різні види захисту від CSRF атак та налаштування CORS. Також є можливості роботи з шифруванням паролів. Досить гнучко налаштовується доступ до різних шляхів API застосунку за різними ролями або іншими обмеженнями, а також різноманітні способи опрацювання відмови доступу. Разом із Spring Security ми використовуватимемо механізм аутентифікації за допомогою JWT.

JWT (Json Web Token) – це стандарт механізму аутентифікації користувачів у веб-застосунку. Особливістю є відсутність стану сесії, тобто при роботі не створюється жодних сесій користувача і не зберігається таких даних на якесь сховище на сервері. Натомість, принцип роботи є наступним: користувач посилає запит із своїми даними для аутентифікації (зазвичай – пароль та пошта/логін/номер телефону тощо), на що сервер створює новий JWT, який, де-факто, є набором зашифрованих символів у стрічці. Всередині зашифрованих даних зберігається деяка інформація яка складається з трьох частин – заголовка (описує медіатип, алгоритм шифрування), вмісту (має дані про того, хто створив

токен, якому суб’єкту призначений, час закінчення токену, час видачі токену та ідентифікатор токену) та підпису (закодовані дані заголовка та вмісту і захешовані алгоритмом вказаним у заголовку). Такий токен повертається користувачу, який повинен його використовувати при кожному наступному запиту на сервер. У разі закінчення строку валідності токена, користувач може запросити новий токен або шляхом нової аутентифікації або маючи деякий Refresh Token, який є тим же JWT, але виданий разом зі звичайним токеном (містить трішки інший набір даних аби їх можна було відрізнити, а також має довший період дії). Refresh Token може використовуватись для зручності роботи користувача та запобігання постійної потреби переаутентифікації.

Перейдемо до Spring Data JPA та Spring Data JDBC. Це фреймворки які призначені для роботи з доступом до даних, які зберігаються у базі даних.

JDBC (Java Database Connectivity) – це механізм роботи з підключенням до баз даних. JDBC має багато різних драйверів які призначені для різних типів сховищ даних. **Spring JDBC** – це бібліотека на платформі Spring яка має в собі засоби роботи з JDBC. [15] Оскільки для нашого застосування ми використовуватимемо систему керування базою даних PostgreSQL, то однією з наших залежностей буде драйвер JDBC для цієї бази даних.

Spring Data JPA – це бібліотека, побудована на специфікації JPA. В свою чергу, JPA (Java Persistence API) – це специфікація, набір концептів які можуть бути імплементовані фреймворками. Ці концепти визначають спосіб роботи з даними таким чином, щоб моделі даних в коді постійно були однаково описані до того, як описані дані в базах даних. Фактично, розмова йде про ORM – Object-Relational Mapping – пропагування моделі даних з бази даних на модель даних в коді і навпаки. Відомим ORM-фреймворком для імплементації JPA являється Hibernate. В свою чергу, Spring Data JPA є надбудовою над Hibernate, яка додає додатковий рівень абстракції для зручної роботи із JPA у Spring Boot. Для

підключення в бібліотеці використовується JDBC, що є найнижчим рівнем роботи з базою даних.

2.5 Підходи Code First та Database First

Важливим моментом є вибір способу написання проекту у контексті роботи з базою даних. Існує два відомих підходи: **Code First** та **Database First** (з англ. – «спершу код» та «спершу база даних»). Не існує єдиної правильної думки щодо того, який з них краще використовувати, а також думки спільноти розробників щодо цього діляться на два великі табори. Проте, що точно відомо, кожен з цих підходів має свої переваги.

Підхід Code First являє собою спосіб створення моделі даних починаючи одразу із коду, і, використовуючи різні інструменти – в тому числі й бібліотеки-провайдери ORM, автоматично пропагувати цю модель на структуру бази даних. Таким чином, при першому запуску застосунку створюються так звані «міграції» - фактично команди до СУБД (наприклад, DDL запити), які створять необхідні таблиці або інші структури для зберігання даних в них. Після цього, міняючи модель даних в коді можна автоматично мігрувати ці зміни на модель у базі даних.

В свою чергу підхід Database First має на увазі абсолютно протилежний спосіб: міграції пишуться програмістом у окремих файлах одразу у потрібному для бази даних вигляді, та вручну або за допомогою спеціальних інструментів (наприклад, бібліотеки Flyway або Liquibase) ці зміни застосовуються прямо до бази даних. Після чого, знову вручну або за допомогою спеціальних інструментів (наприклад, бібліотека Jooq) створюється класи моделей даних у коді.

Як було зазначено, кожен з цих підходів має свої переваги. Наприклад, вважається, що підхід Database First є більш точним з точки зору того, що оскільки база даних містить в собі всю необхідну інформацію, то саме вона повинна «задавати» модель даних. З іншого боку, підхід Code First на практиці

дуже часто виявляється набагато швидшим у реалізації, завдяки широким можливостям бібліотек. Проте, такі бібліотеки мають і свої невеликі недоліки реалізації, заглиблюватись у які потрібно вже на момент їх використання.

2.6 Огляд особливостей фреймворку Angular

Перейдемо до огляду Angular. Як вже було сказано, в підрозділах вище, Angular – це фреймворк побудований на мові програмування TypeScript.

TypeScript – це мова програмування з відкритим кодом побудована над мовою JavaScript шляхом додавання статичних визначень типів. Де-факто, це майже той самий JavaScript але з інструментами для написання більш безпечного для розробки коду. TypeScript трансформується у JavaScript на етапі компіляції своїм компілятором, або за допомогою інструменту Babel. [16] Статичне декларування типів дозволяє суттєво зменшити кількість помилок ще на стадії написання коду. Проте, навіть звичайний JavaScript можна використовувати посеред лістингів коду на TypeScript.

Фреймворк Angular побудований на платформі Node.js, що дозволяє легко і зручно використовувати пакетний менеджер від Node.js – **npm**. Це дає можливість легко додавати нові залежності та бібліотеки до проєкту.

У Angular використовується **реактивне програмування** – парадигма асинхронного програмування, що стосується потоків даних та поширення змін стану об’єктів. Для цього використовується **RxJS** (**Reactive Extensions for JavaScript**) – це бібліотека для реактивного програмування використовуючи спеціальний тип Observable, що дозволяє легше будувати асинхронний та влаштований на так званих пост-викликах (callbacks) код. [17] **Observable** – це такий об’єкт що використовується для слідкуванням за змінами у стані об’єкту.

Фреймворк Angular використовує **принцип компонентів** – головних будівних блоків коду в застосуваннях.

Кожний компонент складається з [17]:

- шаблону HTML;

- класу на TypeScript який визначає поведінку компонентів;

- селектор каскадних таблиць стилів (CSS) який визначає як компонент використовується у шаблоні;

- (опціонально), CSS-стилі які застосовуються до шаблону.

Angular – це фреймворк для створення так званих **Single-Page application**, тобто односторінкових застосувань. Загалом, представлення сторінки складається із деяких будівельних блоків, за які і відповідають компоненти. Для прикладу, сторінка може складатись із заголовної стрічки, деякого контенту та так званого футеру (footer), і кожен з цих елементів можна представити своїм компонентом. Зміна вигляду сторінки та перехід між різними логічними частинами застосунку відбувається заміною компонентів на екрані, що оброблюється повторним рендерингом. Кожен компонент може мати всередині свого шаблону багато інших компонент, тобто їх можна вкладати одна в одну. Також існує багато способів передачі даних між компонентами.

Особливої зручності додає **принцип модульності**. Застосунок можна поділити на модулі, які теж можна вкладати один в інший. Всередині модулів можна визначати необхідні для застосунку компоненти, сервіси та інші класи.

Окрім компонентів у застосунку використовуються **сервіси**. Сервіси у Angular – це класи, в які прийнято виносити якусь загальну логіку для застосунку, яка може бути використана багатьма компонентами. У сервісах зазвичай виконують методи звертання до серверу, роботу з аутентифікацією, будь-які періодичні дії або обробка даних. Вважається гарним тоном якомога більше загального коду виносити у сервіси тому що це сприяє можливостям перевикористання функцій.

Сервіси використовуються компонентами за допомогою вбудованого механізму **Dependency Injection** (**DI**), про який було розказано у підрозділах вище. Огляд практичного використання DI у застосунку буде розглянуто у третьому підрозділі.

Важливою частиною Angular є робота з **роутингом** (тобто маршрутизація між компонентами) всередині застосунку, що часто являється досить складною темою у фреймворках та бібліотеках фронтенд. В Angular використовується модуль app-routing-module для настройки роутингу всередині застосунку. У цьому модулі можна визначати – які шляхи URL будуть використовуватись для відповідних їм компонент. Також можна додавати так звані **guards** (з англ. дослівно – «захисник»). Це спеціальні класи які дозволяють забороняти доступ до шляху та до компоненти якщо не виконуються певні умови (наприклад, неавторизованому користувачу не можна відкривати сторінку для адміністраторів).

Також цікавим інструментом Angular являються **інтерсептори**. Це такі сервіси які реалізовують інтерфейс HttpInterceptor які дозволяють перехоплювати будь-які HTTP-запити та виконувати певні дії з ними. Таким чином легко реалізувати автоматичне розміщення користувацького JWT у заголовку запиту або видалення поточного токену із Local Storage користувача у разі відповіді від серверу HTTP-статусом «401 Unauthorized», що означатиме що у токена користувача завершився термін дії і його потрібно оновити. Все це буде використано у практичній частині цієї курсової роботи.

2.7 Аргументація вибору поєднання Spring Boot та Angular у одному стеку технологій

Важливою особливістю що поєднує дві обрані технології є стабільність та можливості для розширення. Java відома своєю стабільністю через що часто

використовується IT-компаніями у якості мови для корпоративних (enterprise) бізнес-рішень.

Досить відомим є широке використання застосування на Java у банківській сфері. Банки та фінансові установи, що є логічною причиною, потребують від систем стабільності. Саме тому в цій сфері широко використовується Spring Boot та інші фреймворки на Java. [18] Проте, звісно, це не єдина сфера, де використовується Spring.

Також відомо що Angular підтримується компанією-гігантом Google, що доводить не тільки наявність постійних оновлень, а і широка відомість цього інструменту по всьому світу. Відомо, що Angular використовується такими проектами як PayPal, Upwork, Forbes, Gmail, Mixer, Deutsche Bank а також Microsoft Office. [19]

Поєднання цих двох фреймворків робить можливим легко та ефективно розширювати застосунок, що забезпечують такі механізми як модульність, правильна робота з залежностями, швидкість розробки а також чудовий рівень реалізації безпеки застосунку та даних користувачів.

У якості деталі реалізації також варто відмітити, що у разі необхідності розгорнути монолітний застосунок, Angular можливо запускати і на вбудованому сервері Tomcat разом із застосунком Spring Boot на спільному порті. Для цього використовується плагін maven-resource-plugin. При збірці проекту відбувається помодульна компіляція як серверної частини, так і клієнтської, які оформлюють як два великі модулі у Maven-проекті. Цю можливість і буде використано у практичній частині цієї курсової роботи для розгортання на хмарній платформі Heroku.

Підсумовуючи, використання фреймворків Spring Boot та Angular дозволяє сконцентруватись на реалізації бізнес-логіки застосунку, та уникнути занадто громіздких та важких конфігурацій застосунку. Оскільки більшість функцій

шаблонного коду покриваються бібліотеками які входять до складу цих інструментів, можна просто насолоджуватись розробкою веб-застосунку маючи вигоду від не тільки зручності та швидкості реалізації, а і стабільності та легкого розширення системи.

РОЗДІЛ 3. ОПИС ПРАКТИЧНОГО ДОСЛІДЖЕННЯ

3.1 Аналіз технічного завдання

Відповідно до завдання курсової роботи, ціллю є створення онлайн системи для клієнтів мережі магазинів. Головна суть застосування є у тому що вона повинна використовуватись для одночасно кількох ролей – адміністрації магазину та клієнтів. Відповідно, потрібен механізм додавання адміністрації магазину, тому додамо і третю роль – адміністратор сайту.

Адміністратор сайту може виконувати всі ті ж функції, що й персонал магазину а також реєструвати в системі нових адміністраторів магазину. Адміністратор магазину може переглядати, додавати, видаляти та змінювати товари та категорії товарів, переглядати та змінювати статуси замовлень користувачів у конкретних магазинах мережі. Користувач може переглядати товари та категорії товарів, створювати замовлення, переглядати та змінювати свої замовлення

Як вже було згадано, для складання застосунку та роботи на одному порті було використано maven-resource-plugin. Було створено проект на базі інструменту для управління та складання програм Maven. Він матиме в собі два підмодулі які призначені для проектів бекенду та фронтенду відповідно. У файлах pom.xml, які містять у зовнішньому модулі та двох підмодулях була описана конфігурація для правильного складання проекту в один пакет jar, який знаходитиметься у модулі бекенду. Загальну структуру проекту можна побачити на рисунку 4:

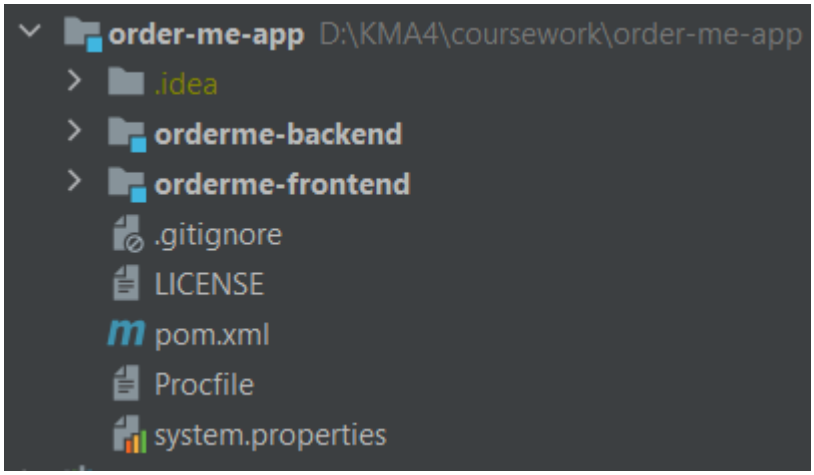


Рисунок 4. Структура монолітного проєкту "OrderMe"

Для зручного зберігання версій застосунку використана система керування версіями Git а також віддалений репозиторій у хмарному сховищі проєктів Github. Для конфігурації запуску застосунку на Heroku було написано файл під назвою «Procfile» із необхідними інструкціями, а також «system.properties» в якій вказана версія Java (було використано 11 версію SDK) для правильного деплойменту.

3.2 Проєктування моделі даних

Перейдемо до реалізації. Варто розпочати саме з моделі даних, адже на ній буде основана уся бізнес-логіка.

Для імплементації сховища даних у застосунку використано систему керування базами даних – PostgreSQL. **PostgreSQL** – це потужна об’єктно-реляційна база даних з відкритим кодом з більш ніж 30 роками активної розробки, яку використовують розробники по всьому світі і дотепер. Вона заслужила репутацію надійності та ефективності. [20] ЇЇ було запущено на хмарній платформі Heroku.

Відповідно до необхідних функцій, описаних у технічному завданні була складена ER модель. Її зображення можна побачити на рисунку 4:

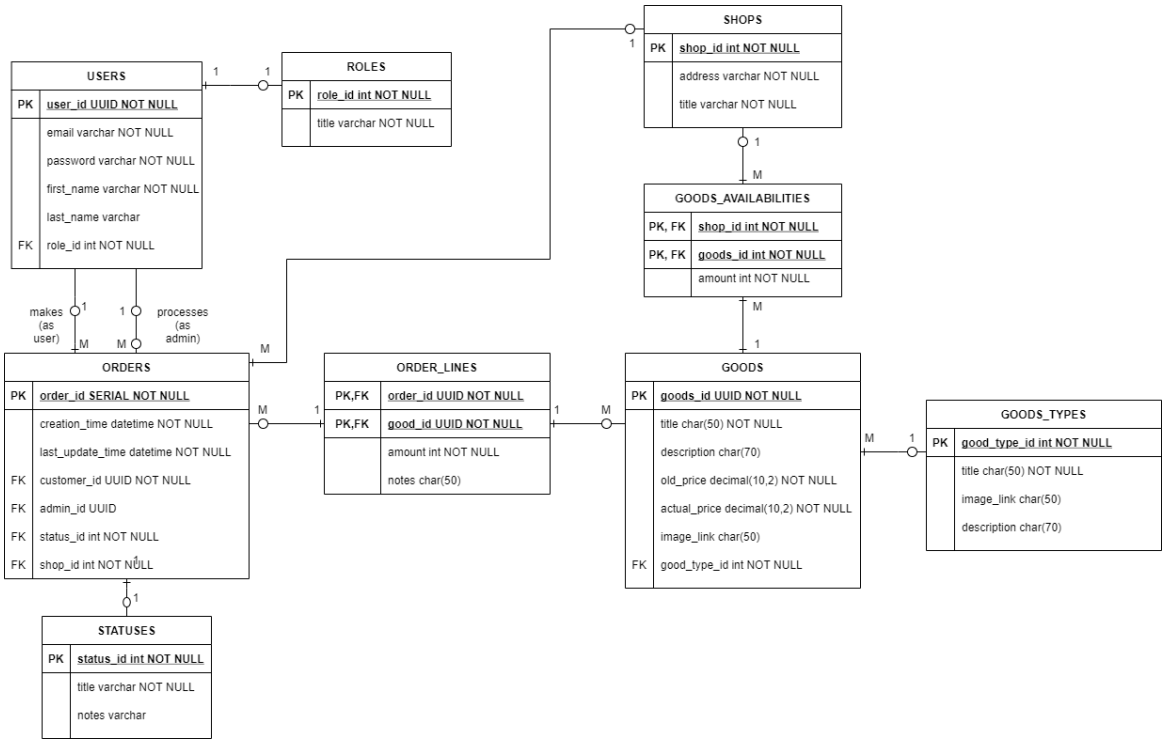


Рисунок 5. ER-модель застосунку "OrderMe"

Потрібно також відмітити що в реляціях GOODS та GOODS_TYPES, які слугують таблицями для зберігання, відповідно, товарів та категорій товарів, є атрибути image_link, в яких зберігатимуться посилання на зображення у хмарному сервісі Firebase Storage. Саме на цій платформі і будуть зберігатись зображення, адже вона довзволяє швидко та зручно завантажувати та вивантажувати файли.

3.3 Реалізація серверної частини

Перейдемо до імплементації серверної частини. На Рисунку 6 можна побачити структуру проєкту бекенду створеної на Spring Boot:

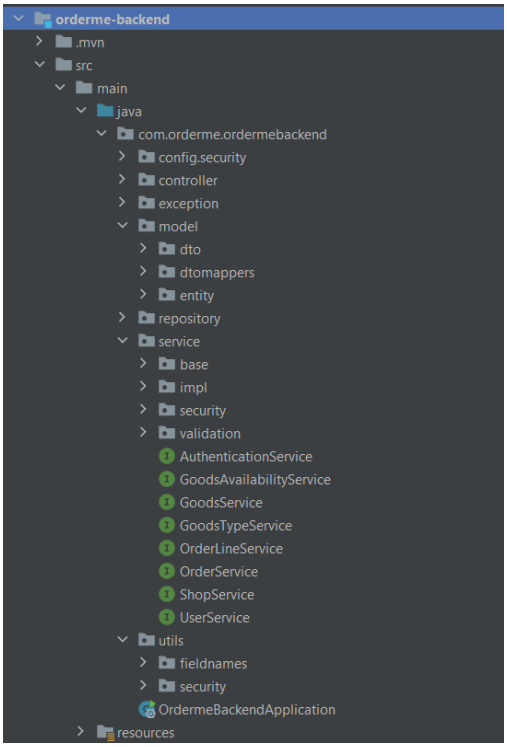


Рисунок 6. Структура проєкту бекенду "OrderMe"

Звісно, частина підмодулів було сховано задля адекватності розміру скріншоту. Структуру проєкту можна умовно поділити на такі чистини: конфігурації (папка config), класи контроллерів (controller), робота з помилками (exception), класи моделей – включають в себе класи DTO, класи POJO для представлення сутностей та класи перетворювачів – мапери з DTO у POJO (папка model), класи репозиторіїв (папка repository) – інтерфейсів для роботи з базою даних, класи сервісів (папка service), та папка з утилітами (utils). Центральною точкою запуску застосунку (аналог Main) у фреймворку Spring Boot та у даному випадку являється клас OrdermeBackendApplication, який викликає SpringApplication.run з необхідними аргументами.

Отже, розглянемо деякі з названих частин детальніше.

3.3.1 Конфігурація Spring Security

Відповідно до технічного завдання та необхідної бізнес-логіки у практичній системі використовується три наступні ролі: USER, ADMIN та SUPER_ADMIN.

Перша відповідає за клієнтів, друга за адміністраторів магазину, третя за адміністратора сайту.

Варто розглянути **Додаток 1**. У цьому додатку зображені методи налаштування захисту під назвою configure. Верхній додає у конфігурацію створений нами сервіс UserDetailsService, який надає доступ до даних користувача, які будуть використовуватись для авторизації та аутентифікації. Нижній, в свою чергу, задає параметри відкритості/закритості різних частин API застосунку для різних ролей користувачів, а також дає можливість настроїти додану розробником нестандартну обробку помилок, параметри сесій та фільтрів.

У конфігурації захисту застосунку також присутній клас JwtAuthenticationFilter. Він додається як окремий фільтр у попередньо згаданому методі та використовується для перевірки JWT, надісланих від користувачів. У разі невалідності токену, невідповідності ролі користувача одній з дозволених або відсутності токену – запит по захищеному шляху відхиляється.

3.3.2 Зв'язок класів контролерів та сервісів

Як відомо, у застосунках Spring Boot основаних на роботі із Spring Web MVC використовуються так звані контролери – це класи які мають анотацію @Controller (у даному випадку - @RestController). Такі класи є компонентами у Spring Boot в яких розробник може об'являти методи які будуть викликатись при вході запиту до API у застосунок. Як видно на прикладі контролеру **у Додатку 2**, який відповідає за роботу з товарами (Goods), ми можемо об'являти методи для різних CRUD операцій з різними параметрами запитів, частин URL-шляху, та тіла запиту за допомогою анотацій над методами @GetMapping, @PostMapping, @PatchMapping та @DeleteMapping. Ці анотації позначають HTTP-методи які будуть обслуговуватись цими методами. Також можемо побачити один з видів згаданого у попередньому розділі DI всередині

контролеру (за допомогою ін’єкції як параметр контструктора): у додатку видно як у клас контролеру додано залежності на сервіси GoodsService та DtoValidationService<GoodsDto> (типізований сервіс для валідації DTO які створюються з даних які передаються запитами ззовні). Всередині показаних методів контролера виконуються запити спочатку на методи сервісу валідації для перевірки правильності даних), а потім і методи сервісу GoodsService.

Розглянемо частину сервісу GoodsService у **Додатку 3**. На рисунку зображений лістинг коду з двома методами із сервісу GoodsServiceImpl, який реалізовує інтерфейс GoodsService, методи якого теж зображені у цьому додатку. Варто підмітити анотацію @Transactional яка використовується механізмами аспектно-орієнтованого програмування у Spring Boot. Усередині методів виконується перетворення даних з об’єкту DTO (який передав клас контролеру) у POJO (необхідного для збереження даних у базу даних), а також виклики методів класів репозиторію GoodsRepository, залежність на який було додано в цей сервіс тими ж механізмами DI, що і в попередньому додатку. Згадана анотація слугує маркером для програмного створення транзакції для того щоб у разі виникнення помилки під час дії запиту до бази даних відбувалась відміна змін у рамках даної транзакції.

За аналогією до коду у розлянутих додатках 2 та 3 у проєкті створені та налагоджені всі необхідні CRUD операції у класах відповідних сервісів та контролерів для обробки даних користувачів, товарів, категорій товарів, їх наявностей, магазинів у мережі а також усього непотрібного для обробки замовлень користувачів. У класах сервісів також містяться необхідні приватні методи для реалізації необхідної бізнес-логіки.

3.3.3 Огляд репозиторіїв з використанням Spring Data JPA

Перейдемо до реалізації вже згаданих раніше репозиторіїв. Як було сказано, в роботі використаний фреймворк Spring Data JPA. Для коректної роботи та

реалізації підходу Code First, фреймворком використовуються анотації пакету javax.persistence. У POJO-класах сутностей потрібно об’явити необхідні поля та позначити необхідні анотації @Entity, @Id, @Column та ін. для об’явлення класу як джерело для нової реляції, поля як джерело ідентифікатора, настроювання параметрів (constraints) поля відповідно для використання їх у роботі ORM-інструментів фреймворку. Приклад класу сутності замовлень з такими анотаціями представлений у **Додатку 4**. Також у додатку можна замітити анотації @ManyToOne та @OneToMany які позначають необхідність зробити із полів класу Foreign Key, тобто зовнішнім ключем у різних типах зв’язків (багато до одного, та один до багатьох).

Також варто звернути увагу на зручність написання репозиторіїв із фреймворком Spring Data JPA. Цікавим є те що необхідно лише створити інтерфейс та вказати типізацію – клас сутності та клас типу ідентифікатора сутності. Інтерфейс необхідно унаслідувати від інтерфейсу JpaRepository<Entity, Id> а реалізація згенерується автоматично під час компіляції застосунку. Усі CRUD операції вже є у інтерфейсі тому в основному такі інтерфейси залишаються майже пустими або зовсім пустими. Також в інтерфейсі можна вказувати необхідні для бізнес-логіки методи лише правильно вказуючи назву методу із необхідними іменами параметрів із класу сутності, а також самі параметри і результат методу. Приклад такого сервісу можемо побачити на Рисунку 7:

```
package com.orderme.ordermebackend.repository;

import ...

public interface GoodsRepository extends JpaRepository<Goods, UUID> {

    Page<Goods> findAllByGoodsType(GoodsType goodsType, Pageable pageable);

}
```

Рисунок 7. Приклад репозиторію - репозиторій для товарів

3.4 Реалізація клієнтської частини

Розглянемо модуль orderme-frontend в якому міститься проект на Angular. Як вже було сказано, Angular – це фреймворк для побудови односторінкових веб-застосувань за допомогою компонентів, тому головний HTML-файл виглядає наступним чином:

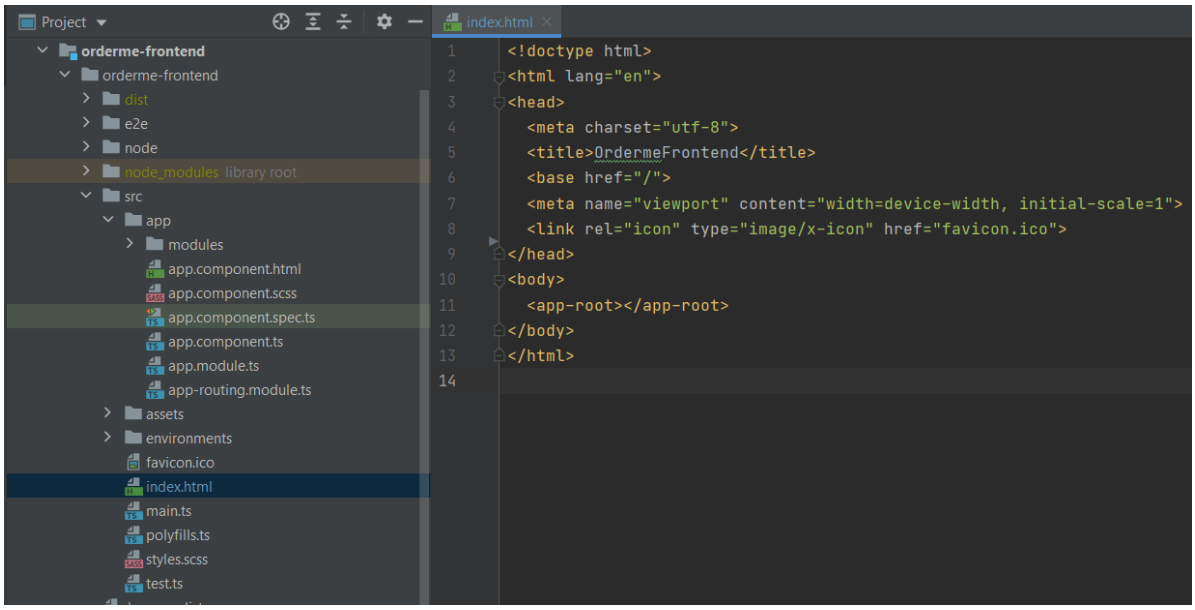


Рисунок 8. Лістинг коду з головного HMTL-файлу

Як можна замітити з Рисунку 8, весь застосунок будується покомпонентно, і головним зовнішнім компонентом є app-root. Як вже було сказано, компоненти можути містити всередині себе інші компоненти, що і відбувається у нашому випадку.

3.4.1 Перелік використаних бібліотек

Для роботи були використані бібліотеки:

- вже згадана раніше RxJS для використання асинхронності;
- бібліотека AngularFire, для підключення та обміну даних зі згаданим у попередньому розділі Firebase Storage;
- бібліотеки Bootstrap та ng-bootstrap – перша є бібліотекою стилів та скриптів для швидкого та зручного оформлення стилів сторінок, а друга є

доповненням до Bootstrap у Angular із готовим набором компонент, які можна зручно використовувати для усіляких форм, пажинації та інших елементів інтерфейсу;

- бібліотека Angular-fontawesome з готовими картинками та значками які можна широко використовувати для покращення досвіду користувача;
- @angular/core, @angular/common/http та інші – бібліотеки із фреймворку Angular в яких містяться всі необхідні типи для роботи із усіма аспектами застосунку на Angular (наприклад, із http підключенням).

3.4.2 Робота створених модулів, сервісів, компонентів, гвардів та інтерсепторів

Отже, розпочнемо з модулів. На Рисунку 9 зображено структуру проєкту зі всіма створеними модулями:

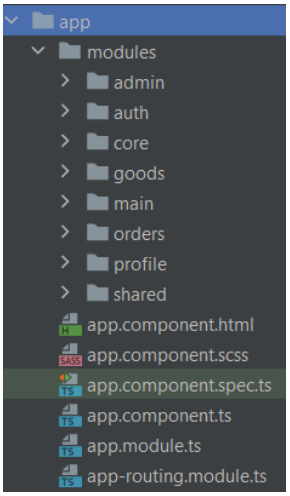


Рисунок 9. Модульна структура Angular застосунку

Як можна здогадатись, вони названі відповідно до тематики компонент що в них розташовані. Найголовнішими є app.component шаблон якого і викликається всередині тегу <body> на Рисунку 8.

Варто одразу розглянути файл app-routing.module.ts, який зображено у **Додатку 5**. У масиві routes маємо список об’єктів, які визначають шлях до компоненти, клас компоненти, з яким його треба зв’язати а також поля

canActivate та data в яких використовується гвард AuthenticationGuard а також вказуються ролі для нього. Таким чином, при зміні компоненти за вказаним шляхом спочатку клас гварду буде перевіряти чи можна дозволити відкрити цю компоненту користувачу, поточна роль якого є однією з тих, що в вказані у об’єкті data для цього шляху. Реалізація гварду є тривіальною та її можна легко знайти в документації Angular.

Перейдемо до головного модуля у файлі app.module.ts. У ньому, як і в будь-якому іншому модулі містяться декларації усіх компонент з нього, а також імпортуються всі інші модулі. Також тут об’являються ті самі інтерсептори, про які було розказано у Розділі 2. Їх у застосунку два: UnauthorizederrorInterceptor та AuthenticationInterceptor. Перший виконує функцію видалення поточного токену у разі отримання від серверу помилки про невалідність токену. Другий кладе у заголовок кожного запиту поточний JWT токен, якщо він присутній у локальному сховищі (local storage) браузеру. Їх реалізація є також тривіальною та знаходиться у документації Angular.

Оскільки створених компонентів для застосування виявилось надзвичайно багато – розглянемо тільки повний приклад роботи одного з них та його зв’язку з сервісом. Отже, обраним компонентом буде компонент categories. Він знаходиться у модулі goods та має в собі, як і інші компоненти файли: typescript файл класу, html-файл шаблону та css-файл стилів. Також є файл для тестування компоненту з уже згенерованим кодом, він має розширення «.spec.ts».

Розглянемо лістинг коду з **Додатку 6**. У ньому показаний файл на мові typescript, у якому і відбувається робота з даними, описуються необхідні функції а також виконується DI у конструкторі (за допомогою параметрів конструктору). Як можемо замітити, в цей клас було додано залежності на класи сервісів: GoodsTypeService – сервіс роботи з категоріями товарів, а саме для відправлення запитів на сервер, клас роутеру – за допомогою нього відбувається навігація в застосунку з коду, сервіс ToastsService – має в собі функціонал для показу

повідомлень користувачу зверху екрану (реалізація є тривіальною для Angular та її можна побачити в документації Angular), а також сервіс AuthenticationService, в якому відбувається вся робота з аутентифікацією у застосунку. У конструкторі ми бачимо перевірку ролі поточного користувача та ініціалізація булевої змінної isAdministrator, яка необхідна для визначення чи показувати користувачу деякі елементи інтерфейсу, які має бачити лише адміністратор. Також імплементує інтерфейси OnInit та OnDestroy, які дають можливість автоматично робити необхідні дії при ініціалізації компоненти та її знищення. Цей механізм використовуватимемо щоб викликати функцію fetchGoodsTypes() при ініціалізації компоненти а також для знищення підписок на об’єкти Observable, що вважається правилом хорошого тону у Angular (необхідно піклуватись про такі підписки коли вони нам вже не потрібні, аби вони не залишали «завислих» об’єктів у пам’яті). У функції fetchGoodsTypes можемо бачити створення підписки на об’єкт Observable який повертається сервісом, який викликається в цьому методі. Тут визначаються необхідні дії у разі позитивного результату від серверу (повернення даних, які треба показати користувачу) або негативного результату (у разі помилки потрібно показати якесь повідомлення користувачу аби він знав що щось пішло не так).

Також розглянемо лістинг коду з **Додатку 7.** Тут зображений html-шаблон, який буде показаний на сторінці у браузері. Він наповнюється даних за допомогою спеціальних директив *ngFor, *ngIf та інших. Вони дозволяють генерувати шаблон у «живому режимі» під час оновлення даних у класі компоненту. Перша директива дозволяє ітеруватись по колекції об’єктів, а друга дозволяє виконувати дії в залежності від певного булевого значення. Також тут можна побачити всі необхідні класи CSS, та інші елементи розмітки.

Як результат при роботі застосунку маємо наступний вигляд сторінки з категоріями товарів для поточного користувача з роллю адміністратора (для

звичайного користувача відсутні кнопки створення та редагування категорій), дані про які прийшли з сервера:

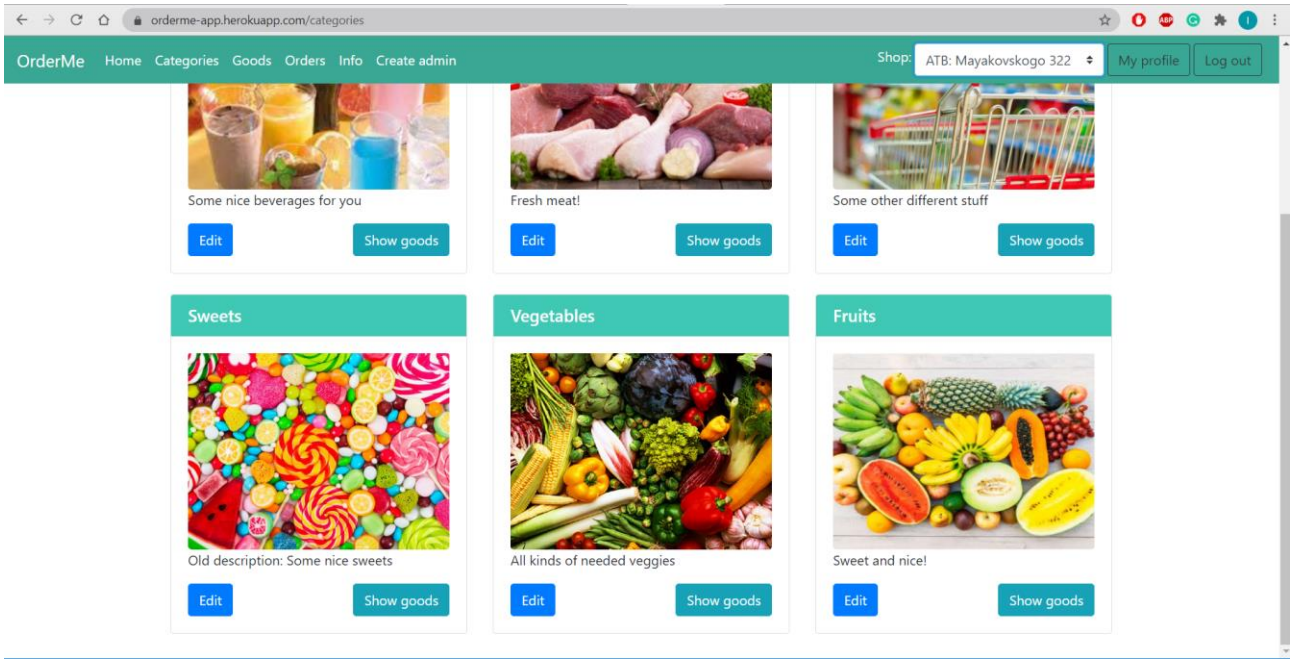


Рисунок 10. Результат роботи компоненту - категорії товарів

Також у **Додатку 7** зображений лістинг коду частини класу сервісу GoodsTypeService. В ньому зображений конструктор (в якому за допомогою DI було зроблено ін'єкцію HttpClient – сервісу бібліотеки @angular/core для відправлення запитів по протоколу HTTP. Відповідно, він використовується для посилання запитів різних типів (get, post, patch, delete та ін.) а також для отримання результату запиту у вигляді об'єкту Observable. Таким чином компонент при виклику методу сервісу отримує об'єкт одразу і в момент коли в ньому з'являється значення – наповнює даними шаблон за допомогою колбеків (післявикликів), які описані у класі компоненти.

Коротко кажучи, були створені аналогічні сервіси для аутентифікації, роботи з товарами, категоріями товарів, замовленнями користувача, віртуальним кошиком товарів для замовлення, профілем користувача. Також були створені сервіси-утиліти для перетворення дат, роботи зі сховищем Firebase Storage (за допомогою бібліотеки AngularFire запити на зберігання картинок та їх

скачування відправляються одразу на фронтенд, а на бекенд відправляються лише посилання на збережені картинки, які зберігаються у базу даних), обробки помилок, роботи з модальними вікнами, локальним сховищем браузера, відображення повідомлень користувачу а також для роботи з вікном (наприклад, прокручування екрану). Всі ці сервіси, які і класи DTO та класи представлення сутностей було поміщено у модуль modules/core у відповідних директоріях services та model.

3.5 Огляд результату практичної частини

Результатом практичної частини курсової роботи став повноцінний сайт для замовлень у мережі магазинів (до того ж, будь-яких товарів, адже наповнювати сайт можна адміністрацією динамічно). Будь-яка людина може зайти на головну сторінку сайту, побачити там деякі пропозиції товарів, які зараз мають акції, або просто випадковий перелік товарів. Людина може прочитати інформацію про призначення та можливості сайту або контакти адміністрації магазину на відповідних вкладках. Також неавторизований користувач може зайти на сторінку перегляду категорій товарів (див. Рисунок 10), або одразу на сторінку перегляду усіх товарів, де є пошук та необхідна фільтрація. Користувач може переглянути конкретний товар, обрати конкретний магазин із списку магазинів у мережі та додати його до свого кошика у необхідній йому кількості якщо товар наявний у цьому магазині. Після цього користувач може додати ще більше товарів або зайти у свій кошик, приклад якого зображено на Рисунку 11. Користувач може також доналаштовувати кількість товарів у елементах кошику або видаляти звідти додані товари. У разі спроби неавторизованого користувача зробити замовлення (Checkout), користувачу показується повідомлення із проханням авторизуватись або зареєструватись на сайті для продовження створення повідомлення:

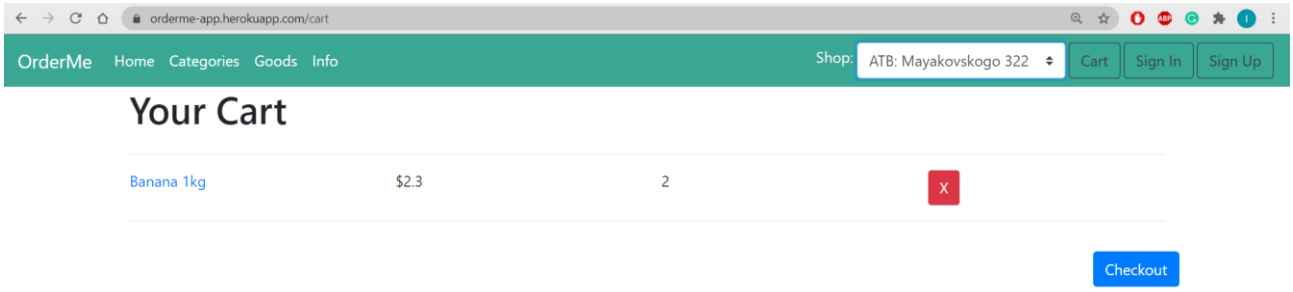


Рисунок 11. Скріншот сторінки кошика користувача

Модальні вікна авторизації та реєстрації які викликаються відповідними кнопками у верхньому меню справа є тривіальними (див. Рисунки 12 та 13):

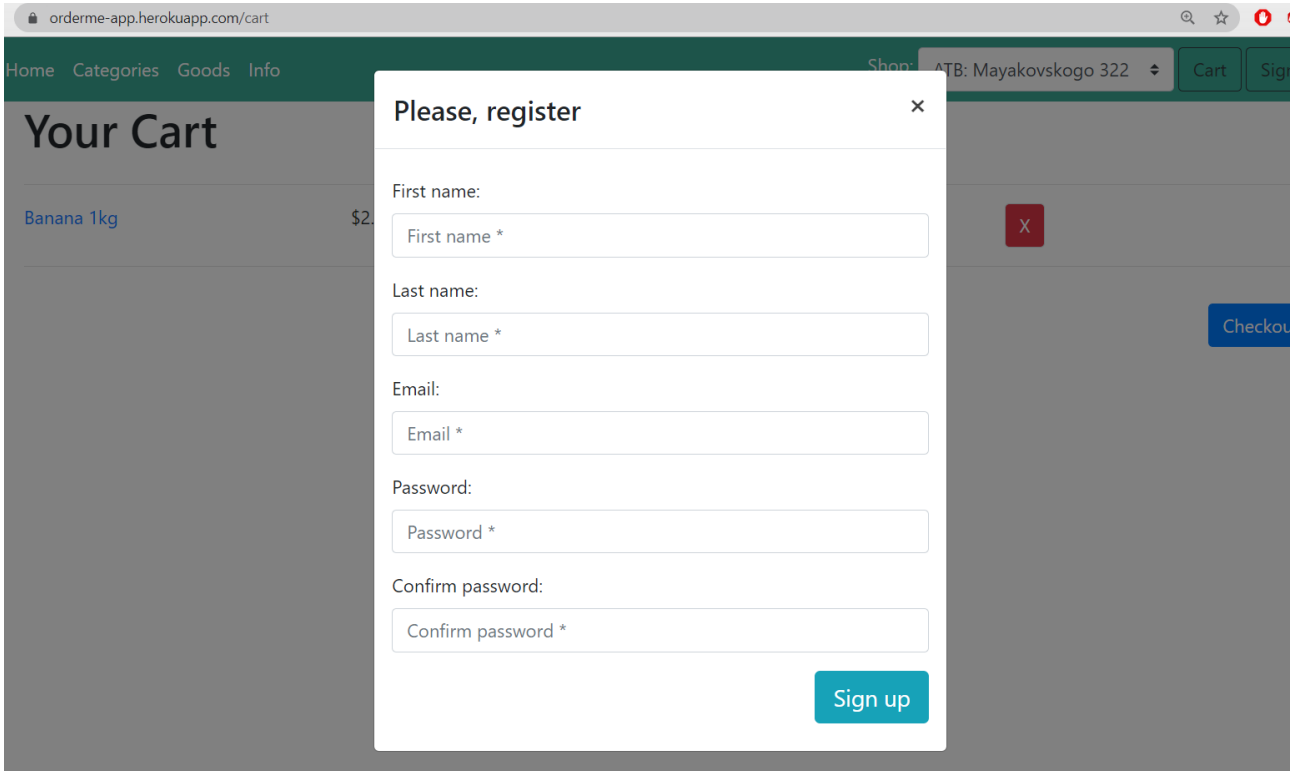


Рисунок 12. Модальне вікно реєстрації

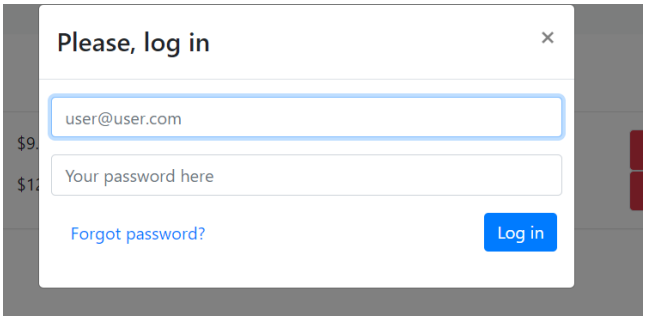


Рисунок 13. Модальне вікно авторизації

Авторизований користувач може створити замовлення та після успішного результату його перенаправляє на сторінку його замовлень. Тут користувач може бачити список своїх замовлень (Рисунок 14), їх деталі та статуси, може заходити на сторінку сторінку огляду конкретного замовлення та має можливість його відмінити якщо його ще не почали обробляти адміністратори. Приклад відміненого замовлення можемо бачити на скріншоті на Рисунку 15.

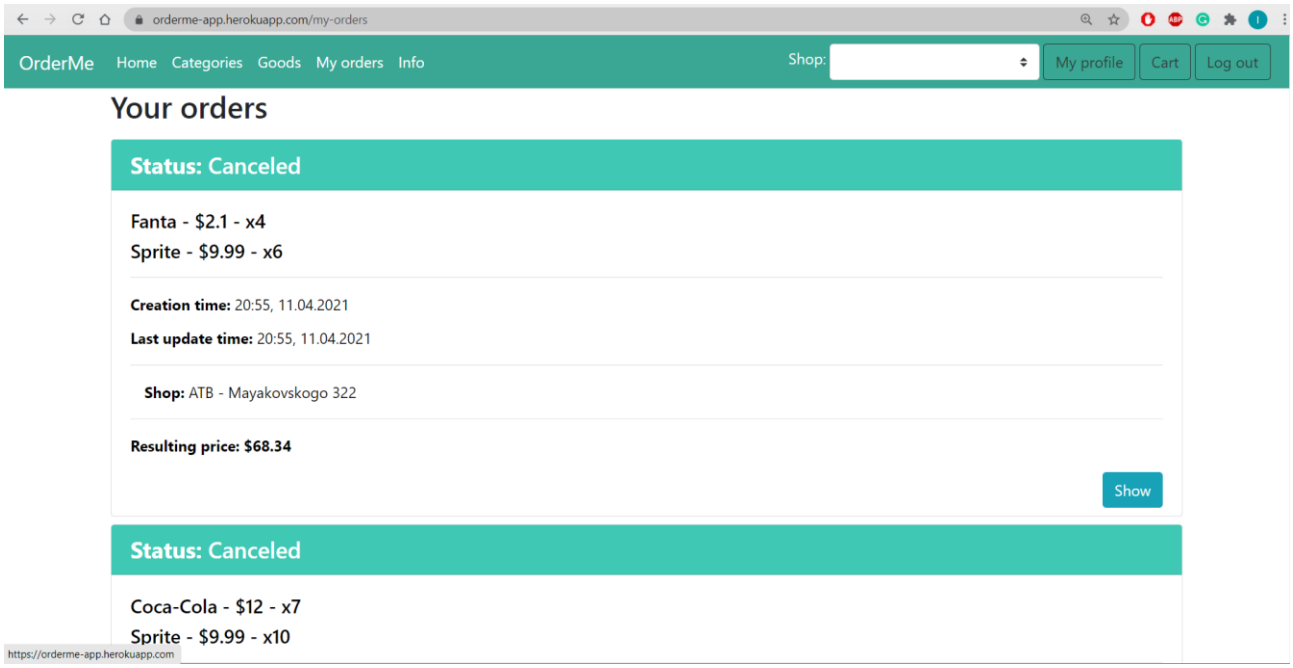


Рисунок 14. Сторінка замовлень користувача

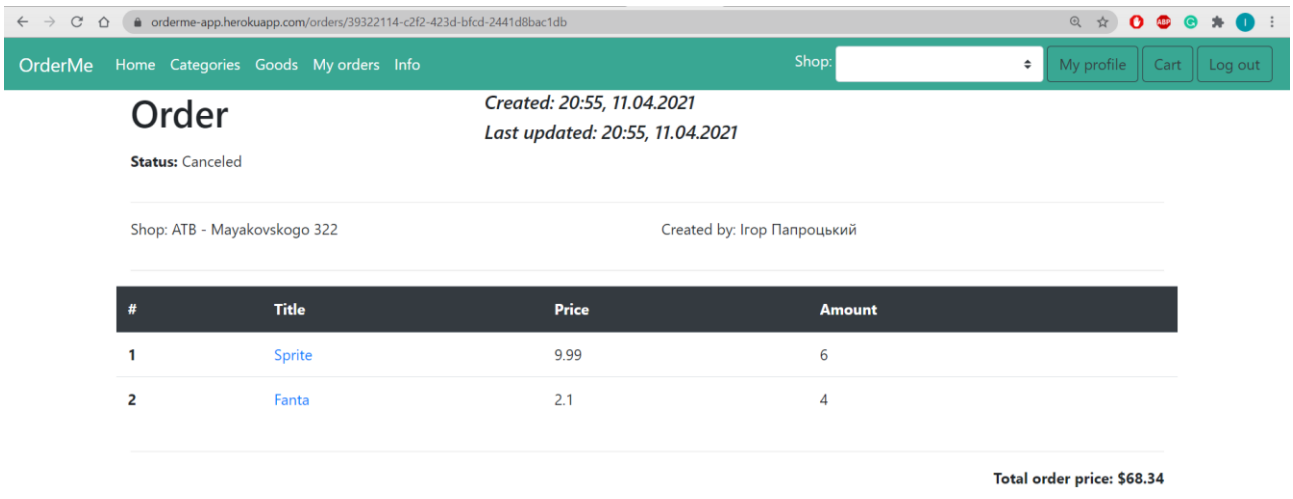


Рисунок 15. Сторінка огляду замовлення - скасоване замовлення

Усі замовлення можуть оброблятися персоналом магазину на відповідній сторінці обробки замовлень, приклад такої сторінки можна побачити на Рисунку 16. Адміністрація може фільтрувати замовлення за статусом, шукати замовлення не прийняті в обробку (до речі, після початку обробки замовлення, воно закріпляється за адміністратором який його прийняв у обробку, тому надалі керувати ним може тільки той адміністратор, який має такі права), або шукати замовлення закріплені за поточним адміністратором:

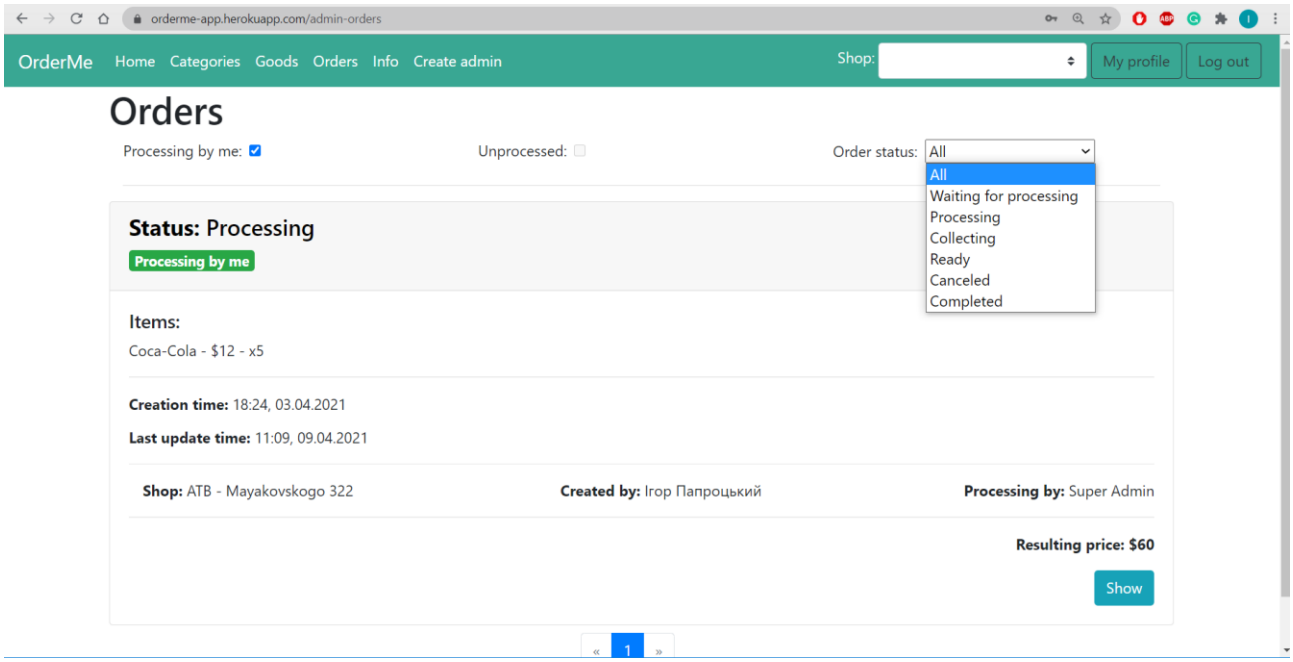


Рисунок 16. Скріншот сторінки обробки замовлень для адміністрації магазину

У разі початку обробки адміністратором замовлення (кнопка Start processing), він може управляти ним на сторінці, аналогічній сторінці на Рисунку 15.

Варто побачити також приклад успішної валідації полів форми на фронтенді за допомогою модуля Angular «ReactiveForms». Оскільки бібліотека надає чудові можливості автоматично обробляти поля на рахунок порушення доданих розробником обмежень, це дуже пришвидшує створення таких форм. Приклад такої форми можна побачити на сторінці реєстрації нового адміністратора магазину, яка доступна лише адміністратору сайту:

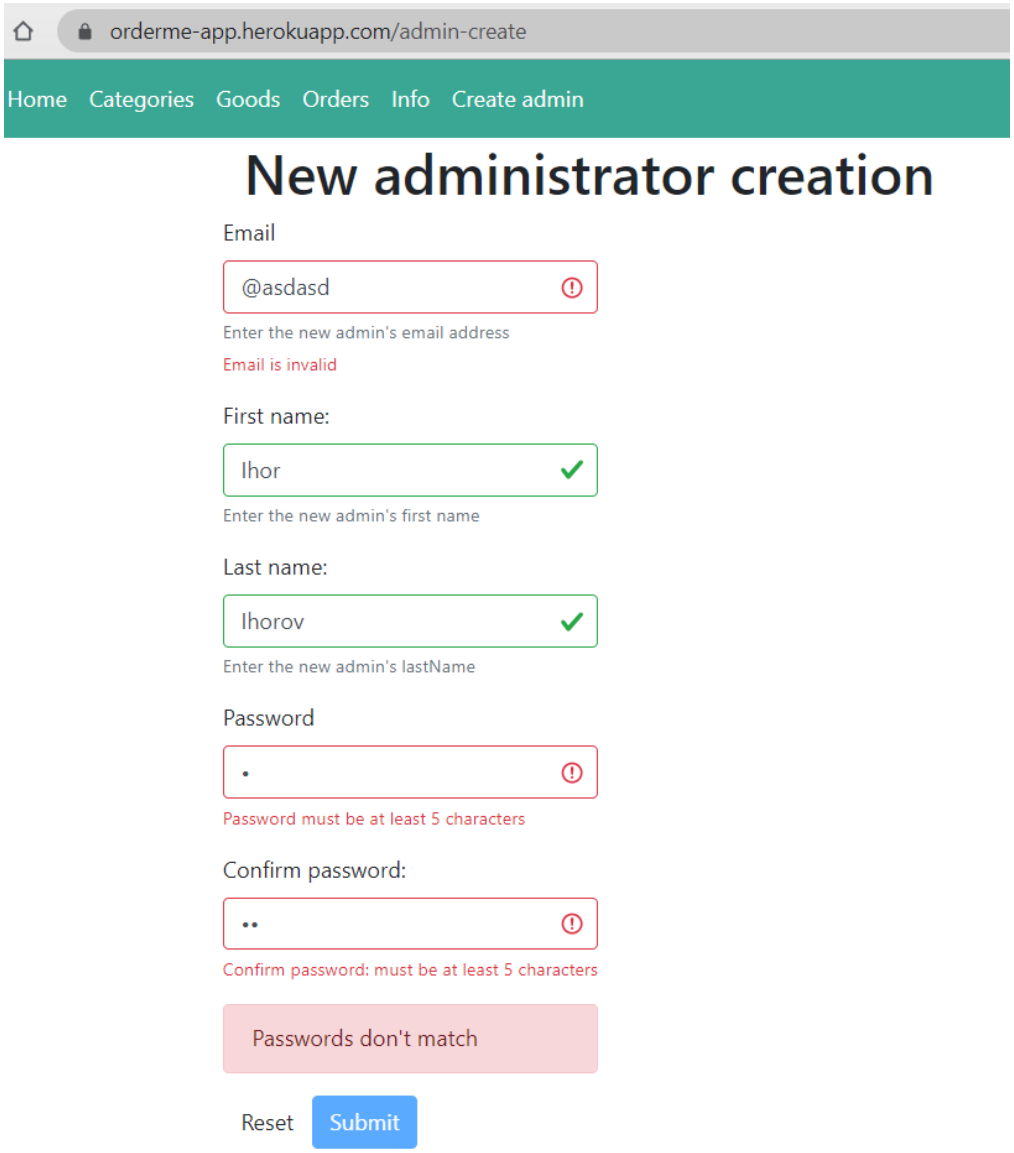


Рисунок 17. Скріншот сторінки реєстрації адміністратора. Приклад валідації форми

Переходячи до висновків, було повністю реалізовано запланований застосунок.

ВИСНОВКИ

Отже, результатом роботи стало плідне дослідження можливостей фреймворків Angular та Spring Boot а також різних бібліотек що входять до їх складу або можуть бути до них додані. Були розглянуті багато особливостей влаштування роботи системи, використання тих чи інших бібліотек та підходів до вирішення задач необхідних для розробки високоякісних веб-застосувань. Як наслідок, було створено повноцінну веб-систему, яка може задовільнити мережу магазинів як у вигляді CRM-системи з різними ролями користувачів, так і у вигляді користувацького сайту. Систему було вдало задеплоєно на хмарну платформу а також під'єднано до хмарних бази даних та сховища файлів. На кінець, було обговорені деякі деталі реалізації вищезазначеного веб-застосунку.

Як результат дослідження, яке було описано в цій курсовій роботі, можна виокремити такі переваги та недоліки обраного підходу до використання обраних технологій у розробці веб-додатків:

- **серед переваг**: широкі можливості для налаштувань оточення, конфігурацій, роботи з даними та зовнішньою взаємодією елементів системи які розробники отримують готовими одразу при підключенні фреймворків до проєкту; економію часу та кількості коду необхідного для створення правильного осередку роботи з даними; надійність, ефективність розробки та роботи, а також чудові можливості для розширення системи; сучасний підхід та величезна кількість встановлених підходів та кращих практик від спільноти розробників з усіх куточків планети; можливість розробки як малих проєктів, так і проєктів корпоративного рівня; а також багато іншого.
- **серед недоліків**: необхідність хорошого розуміння усіх процесів та практик чистоти та ефективності коду з використанням зазначених фреймворків а також необхідність витрати деякого часу на опанування усіх бібліотек та інструментів, які пропонуються фреймворками.

Як висновок, можна із впевненістю сказати, що поєднання фреймворків Angular та Spring Boot надає безліч ефективних можливостей для втілення ідей бізнесу як малого рівня, так і корпоративного. Обидва фреймворки добре працюють як в складі єдиної системи, так і легко виконують поставлені на них задачі окремо. Доказом цього є їх широке використання компаніями різного рівня по всьому світу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Perficient – Mobile vs Desktop Usage In 2020 – Eric Enge (2021) [Електронний ресурс] https://blogs.perficient.com/2021/03/23/mobile-vs-desktop-usage-in-2020/
2. Medium – What Is a Technology Stack? Choosing the Right Technology Stack For Your Web Project – Andrei Suschevich [Електронний ресурс] https://medium.com/swlh/what-is-a-technology-stack-choosing-the-right-tech-stack-for-your-web-project-3f295cf60f10
3. Kadamtech – Top backend frameworks in 2020 [Електронний ресурс] https://www.kadamtech.com/top-backend-frameworks-in-2020/
4. Hackr.io – 10 Best Web Development Frameworks – Aman Goel [Електронний ресурс] https://hackr.io/blog/web-development-frameworks
5. Spring.io – Spring Boot [Електронний ресурс] https://spring.io/projects/spring-boot
6. Блог розробника Vue.js – First week of launching vue.js – Evan You, 2014 [Електронний ресурс] https://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project/
7. Angular.io – What is Angular [Електронний ресурс] https://angular.io/guide/what-is-angular
8. University of Waterloo – Client-server architecture [Електронний ресурс] https://cs.uwaterloo.ca/~m2nagapp/courses/CS446/1195/Arch_Design_Activity/ClientServer.pdf
9. Restfulapi – What is REST – REST API Tutorial [Електронний ресурс] https://restfulapi.net/
10. University Of California, Irvine – Dissertation – Roy Thomas Fielding, 2000 [Електронний ресурс] https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
11. ietf.org – Uniform Resource Identifier (URI): Generic Syntax – Network Working Group (January 2005) [Електронний ресурс] https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

12. Spring.io – Spring Framework – [Електронний ресурс] https://spring.io/projects/spring-framework

13. Baeldung.com – Inversion of Control and Dependency Injection [Електронний ресурс] https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring

14. Docs.Spring.io – Web MVC framework - [Електронний ресурс] https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

15. Docs.Spring.io – Spring Data JDBC - [Електронний ресурс] https://spring.io/projects/spring-data-jdbc

16. Typescript.org – What is TypeScript - [Електронний ресурс] https://www.typescriptlang.org/

17. Angular.io – The modern web developer's platform [Електронний ресурс] https://angular.io/

18. Redhat.com – 25 years and going strong: Why Java matters to the future of banks – Leon Matthews [Електронний ресурс] https://www.redhat.com/en/blog/25-years-and-going-strong-why-java-matters-future-banks

19. Trio.dev – Who uses Angular in 2021 [Електронний ресурс] https://trio.dev/blog/companies-use-angular

20. Postgresql.org – PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс] https://www.postgresql.org/

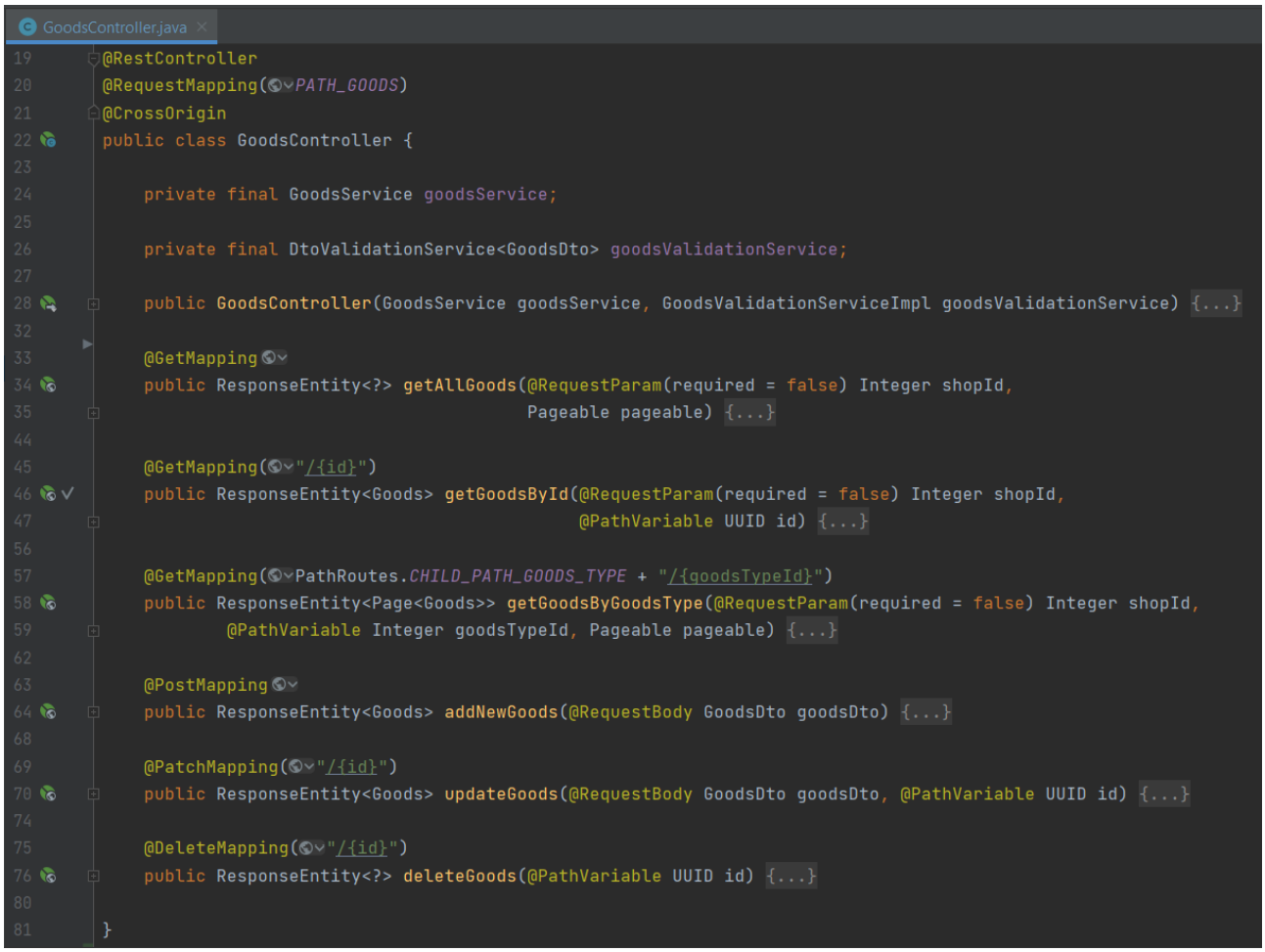
ДОДАТКИ

Додаток 1. Налаштування Spring Security

SecurityConfiguration.java

```
49      @Override
50  @   protected void configure(HttpSecurity http) throws Exception {
51          http.csrf().disable().cors() CorsConfigurer<HttpSecurity>
52                  .and().exceptionHandling().authenticationEntryPoint(new HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED)) ExceptionHandlingConfigurer<HttpSecurity>
53                  .and() HttpSecurity
54                      .authorizeRequests() ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
55                          .antMatchers( ...antPatterns: PathRoutes.PATH_AUTH + PathRoutes.CHILD_PATH_ADMIN_REGISTER).hasAuthority(AUTHORITY_SUPER_ADMIN)
56                          .antMatchers(PathRoutes.PATH_ORDERS).hasAnyAuthority( ...authorities: AUTHORITY_SUPER_ADMIN, AUTHORITY_ADMIN, AUTHORITY_USER)
57                          .antMatchers(PathRoutes.PATH_ORDER_LINES).hasAnyAuthority( ...authorities: AUTHORITY_SUPER_ADMIN, AUTHORITY_ADMIN, AUTHORITY_USER)
58                          .antMatchers(HttpMethod.POST,
59                                  ...antPatterns: PathRoutes.PATH_GOODS_AVAILABILITIES + "/**",
60                                  PathRoutes.PATH_GOODS + "/**",
61                                  PathRoutes.PATH_GOODS_TYPES + "/**",
62                                  PathRoutes.PATH_SHOPS + "/**") ExpressionUrlAuthorizationConfigurer<H>.AuthorizedUrl
63                              .hasAnyAuthority( ...authorities: AUTHORITY_SUPER_ADMIN, AUTHORITY_ADMIN) ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
64                          .antMatchers(HttpMethod.PATCH,
65                                  ...antPatterns: PathRoutes.PATH_GOODS_AVAILABILITIES + "/**",
66                                  PathRoutes.PATH_GOODS + "/**",
67                                  PathRoutes.PATH_GOODS_TYPES + "/**",
68                                  PathRoutes.PATH_SHOPS + "/**") ExpressionUrlAuthorizationConfigurer<H>.AuthorizedUrl
69                              .hasAnyAuthority( ...authorities: AUTHORITY_SUPER_ADMIN, AUTHORITY_ADMIN) ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
70                          .antMatchers(HttpMethod.DELETE,
71                                  ...antPatterns: PathRoutes.PATH_GOODS_AVAILABILITIES + "/**",
72                                  PathRoutes.PATH_GOODS + "/**",
73                                  PathRoutes.PATH_GOODS_TYPES + "/**",
74                                  PathRoutes.PATH_SHOPS + "/**") ExpressionUrlAuthorizationConfigurer<H>.AuthorizedUrl
75                              .hasAnyAuthority( ...authorities: AUTHORITY_SUPER_ADMIN, AUTHORITY_ADMIN) ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
76                      .anyRequest().permitAll()
77                  .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.NEVER) SessionManagementConfigurer<HttpSecurity>
78                  .and().addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class) HttpSecurity
79                  .addFilterBefore(exceptionHandlerChainFilter, JwtAuthenticationFilter.class);
80      }
```

Додаток 2. Приклад контролера у Spring Boot



```java
@RestController
@RequestMapping(PATH_GOODS)
@CrossOrigin
public class GoodsController {

    private final GoodsService goodsService;

    private final DtoValidationService<GoodsDto> goodsValidationService;

    public GoodsController(GoodsService goodsService, GoodsValidationServiceImpl goodsValidationService) {...}

    @GetMapping
    public ResponseEntity<?> getAllGoods(@RequestParam(required = false) Integer shopId,
                                         Pageable pageable) {...}

    @GetMapping("/{id}")
    public ResponseEntity<Goods> getGoodsById(@RequestParam(required = false) Integer shopId,
                                              @PathVariable UUID id) {...}

    @GetMapping(PathRoutes.CHILD_PATH_GOODS_TYPE + "/{goodsTypeId}")
    public ResponseEntity<Page<Goods>> getGoodsByGoodsType(@RequestParam(required = false) Integer shopId,
            @PathVariable Integer goodsTypeId, Pageable pageable) {...}

    @PostMapping
    public ResponseEntity<Goods> addNewGoods(@RequestBody GoodsDto goodsDto) {...}

    @PatchMapping("/{id}")
    public ResponseEntity<Goods> updateGoods(@RequestBody GoodsDto goodsDto, @PathVariable UUID id) {...}

    @DeleteMapping("/{id}")
    public ResponseEntity<?> deleteGoods(@PathVariable UUID id) {...}

}
```

Додаток 3. Лістинг коду частини сервісу GoodsServiceImpl а також інтерфейсу GoodsService

GoodsController.java | GoodsService.java | GoodsServiceImpl.java

```
61
62      @Override
63      @Transactional
64      public Page<Goods> getByGoodsTypeId(Integer shopId, int goodsTypeId, Pageable pageable) {
65          GoodsType goodsTypeRef = goodsTypeRepository.getOne(goodsTypeId);
66          Page<Goods> goodsPage = goodsRepository.findAllByGoodsType(goodsTypeRef, pageable);
67          if (shopId != null) {
68              filterGoodsAvailabilitiesByShop(goodsPage, shopId);
69          }
70          return goodsPage;
71      }
72
73      @Override
74      @Transactional
75      public Goods create(GoodsDto goodsDto) {
76          GoodsType goodsTypeRef = goodsTypeRepository.getOne(goodsDto.getGoodsTypeId());
77          Goods goodsToSave = Goods.builder()
78                  .title(goodsDto.getTitle())
79                  .description(goodsDto.getDescription())
80                  .oldPrice(goodsDto.getOldPrice())
81                  .actualPrice(goodsDto.getActualPrice())
82                  .goodsType(goodsTypeRef)
83                  .imageLink(goodsDto.getImageLink())
84                  .build();
85          return goodsRepository.save(goodsToSave);
86      }
```

GoodsController.java | GoodsService.java | GoodsServiceImpl.java

```
1   package com.orderme.ordermebackend.service;
2
3   import ...
11
12  public interface GoodsService extends PaginatedCrudService<UUID, Goods, GoodsDto> {
13
14      Page<Goods> getByGoodsTypeId(Integer shopId, int goodsTypeId, Pageable pageable);
15
16      Page<Goods> getAllByPageable(int shopId, Pageable pageable);
17
18      Goods getByIdAndShopId(Integer shopId, UUID id);
19  }
20
```

Додаток 4. Лістинг частини POJO-класу Order

```java
@Entity(name = "orders")
@Builder
public class Order implements AbstractEntity<UUID> {

    @Id
    @GeneratedValue(generator = "UUID")
    @GenericGenerator(
            name = "UUID",
            strategy = "org.hibernate.id.UUIDGenerator"
    )
    @Column(unique = true, updatable = false, nullable = false)
    private UUID orderId;

    private LocalDateTime creationTime;

    private LocalDateTime lastUpdateTime;

    @ManyToOne
    @JoinColumn(name="createdBy", referencedColumnName = "userId")
    private User createdBy;

    @ManyToOne
    @JoinColumn(name="processingBy", referencedColumnName = "userId", insertable = false)
    private User processingBy;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private OrderStatus orderStatus;

    @OneToMany(mappedBy = "salesOrder", fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @Column(insertable = false)
    private Set<OrderLine> orderLines;
```

```java
    @ManyToOne
    @JoinColumn(name="shop", referencedColumnName = "shopId")
    private Shop shop;

    @JsonInclude
    @Transient
    private BigDecimal fullPrice;
```

Додаток 5. Основна частина файлу роутинг-модуля в Angular

index.html | core.module.ts | app.module.ts | app-routing.module.ts

```
const routes: Routes = [
  {path: '', component: LandingPageComponent},
  {path: 'categories', component: CategoriesComponent},
  {path: 'my-orders',
    component: MyOrdersComponent,
    canActivate: [AuthenticationGuard],
    data: {roles: [UserRole.USER, UserRole.ADMIN, UserRole.SUPER_ADMIN]}},
  {path: 'sign-in', component:SigninComponent},
  {path: 'categories/:id', component: GoodsByCategoryComponent},
  {path: 'goods/:id', component: ViewGoodsComponent},
  {path: 'goods', component: AllGoodsComponent},
  {path: 'cart', component: CartComponent},
  {path: 'orders/:id', component: ViewOrderComponent},
  {path: 'profile', component: ProfileComponent},
  {path: 'admin-create',
    component: AdminCreationComponent,
    canActivate: [AuthenticationGuard],
    data: {roles: [UserRole.SUPER_ADMIN]}},
  {path: 'admin-orders',
    component: AdminOrdersComponent,
    canActivate: [AuthenticationGuard],
    data: {roles: [UserRole.ADMIN, UserRole.SUPER_ADMIN]}},
  {path: 'category-create',
    component: CategoryCreateComponent,
    canActivate: [AuthenticationGuard],
    data: {roles: [UserRole.ADMIN, UserRole.SUPER_ADMIN]}},
  {path: 'goods-create',
    component: GoodsCreateComponent,
    canActivate: [AuthenticationGuard],
    data: {roles: [UserRole.ADMIN, UserRole.SUPER_ADMIN]}},
  {path: 'category-edit/:id',
    component: CategoryEditComponent,
    canActivate: [AuthenticationGuard],
    data: {roles: [UserRole.ADMIN, UserRole.SUPER_ADMIN]}},
  {path: '**', redirectTo: ''}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Додаток 6. Лістинг частини класу компоненту Categories

categories.component.ts

```
@Component({
  selector: 'app-categories',
  templateUrl: './categories.component.html',
  styleUrls: ['./categories.component.scss']
})
export class CategoriesComponent implements OnInit, OnDestroy {

  subscriptions: Subscription = new Subscription();
  goodsTypeSet: GoodsType[] = [];
  isLoading: boolean = false;
  isEmpty: boolean = false;
  isAdministrator: boolean = false;
  readonly noImagePath: string = './assets/img/no-image.jpg';

  constructor(private goodsTypeService: GoodsTypeService,
              private router: Router,
              private toastsService: ToastsService,
              private authenticationService: AuthenticationService) {
    if (authenticationService.isAuthenticated()){
      this.isAdministrator = (authenticationService.currentUserValue.userRole === UserRole.ADMIN
        || authenticationService.currentUserValue.userRole === UserRole.SUPER_ADMIN);
    }
  }

  ngOnInit(): void {
    this.fetchGoodsTypes();
  }

  fetchGoodsTypes(): void {
    this.isLoading = true;
```

CategoriesComponent

```
    this.subscriptions.add(
      this.goodsTypeService.getAllGoodsTypesList()
        .subscribe(
          next: data => {
            this.goodsTypeSet = data;
            if (this.goodsTypeSet.length === 0) {
              this.isEmpty = true;
            }
          },
          error: error => {
            console.error(error);
            this.toastsService.toastAddDanger( textOrTpl: "Something went wrong during categories fetching. " +
              "Please, contact the administrator");
          }
        )
    );
    this.isLoading = false;
  }

  ngOnDestroy(): void {
    this.subscriptions.unsubscribe();
  }

  navigate(event: Event, goodsTypeId: string, title: string) {
    event.preventDefault();

    this.router.navigateByUrl( url: `/categories/${goodsTypeId}`, extras: {state: { categoryTitle: title}});
  }
}
```

Додаток 7. Шаблон компоненту Categories, а також сервіс для категорій

categories.component.ts | categories.component.html

```
<div class="container">
  <div *ngIf="isAdministrator" class="row">
    <div class="col text-right">
      <button routerLink="/category-create" class="btn btn-info">Create category</button>
    </div>
  </div>
  <hr>
  <div class="row">
    <div *ngFor="let item of goodsTypeSet" class="col-md-4 mt-4">
      <div class="card">
        <h5 class="card-header">{{item.title}}</h5>
        <div class="card-body">
          <div class="embed-responsive embed-responsive-4by3">
            <img class="img-thumbnail card-img-top  embed-responsive-item"
                 [src]="!(item.imageLink) ? this.noImagePath : (item.imageLink | getDownloadURL)" alt="Current photo"/>
          </div>
          <p class="card-text">{{item.description}}</p>
          <button *ngIf="isAdministrator" routerLink="/category-edit/{{item.goodsTypeId}}" class="btn btn-primary">
            Edit
          </button>
          <button (click)="navigate($event, item.goodsTypeId, item.title)" class="btn btn-info float-right">Show goods
          </button>
        </div>
      </div>
    </div>
  </div>
</div>
```

categories.component.ts | categories.component.html | goods-type.service.ts

```
  private GOODS_TYPES_URL = `${environment.apiUrl_v1}goodsTypes`

  private httpOptions = {
    headers: new HttpHeaders( headers: {
      'Content-Type': 'application/json'
    })
  };

  constructor(private http: HttpClient,
              private handleErrorsService: HandleErrorsService) {}

  /**
   * Get a list of goods types
   */
  public getAllGoodsTypesList(): Observable<GoodsType[]> {
    return this.http.get<GoodsType[]>(this.GOODS_TYPES_URL, this.httpOptions)
      .pipe(catchError(this.handleErrorsService.handleError<GoodsType[]>( operation: 'getAllGoodsTypesList',  result: [])));
  }

  /**
   * Get a goods type by id
   */
  public getGoodsTypeById(goodsTypeId: number): Observable<GoodsType> {
    return this.http.get<GoodsType>( url: this.GOODS_TYPES_URL + '/' + goodsTypeId,
      this.httpOptions).pipe(
        catchError(this.handleErrorsService.handleError<GoodsType>( operation: 'getGoodsTypeById')));
  }

  /**
   * Create goods type
```

GoodsTypeService