МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНА ВЕБ-ОРІЄНТОВАНА СИСТЕМА «PANTRY HELPER»

Здобувач освіти гр. ІН – 83 Богдан ГРИЦАЙ

Науковий керівник Олег БЕРЕСТ

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи

здобувача вищої освіти четвертого курсу, групи ІН-83 спеціальності «122 – Комп'ютерні науки» денної форми навчання Грицая Богдана Вікторовича.

Тема: «ІНФОРМАЦІЙНА ВЕБ-ОРІЄНТОВАНА СИСТЕМА PANTRY HELP»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) аналітичний огляд методів побудови подібних інформаційних систем; 2) постановка завдання й формування завдань дослідження; 3) огляд і опис засобів для розробки; 4) проектування та створення бази даних; 5) розробка інформаційної системи Pantry Helper; 6) аналіз результатів.

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____Олег БЕРЕСТ
Завдання прийняв до виконання _____ Богдан ГРИЦАЙ

РЕФЕРАТ

Записка: 64 стор., 41 рис., 5 додатків, 11 джерел.

Об’єкт дослідження — Інформаційна веб-орієнтована система Pantry Helper.

Мета роботи — розробити веб-орієнтовану інформаційну систему Pantry Helper з використанням сучасних технологій: фреймворку Spring, Hibernate, Bootstrap на основі. Спроектувати та створити базу даних на основі системи управління базами даних PostgreSQL, яка буде використана для зберігання всієї необхідної інформації.

Результати — досліджено предметну область, розглянуто аналоги, здійснено постановку завдання та вибір технологій для його виконання, спроектовано та створено базу даних на основі PostgreSQL СУБД, обрано та задіяно для роботи систему для керування процесом розробки програмного забезпечення Atlassian Jira та реалізовано демо-версію веб-орієнтованої інформаційної системи.

ВЕБ-ДОДАТОК, ТОВАРИ, ФРЕЙМВОРК, СУБД, JAVA, SPRING MVC, POSTGRESQL, HTML, CSS, JAVASCRIPT, BOOTSTRAP

ЗМІСТ

ВСТУП

Інформаційні технології - це сучасний помічник в будь-якій галузі діяльності. З їх розвитком виникають веб-технології, які надають змогу виконувати маніпуляції з інформацією, використовуючи лише браузер. Веб-технології стали однією із найбільш впливових ніш у світі інформаційних технологій. Інформаційні веб-ресурси є одним із найголовніших джерел інформації, а також допомагають вирішувати величезну кількість різноманітних повсякденних задач.

Оскільки сьогодні людство намагається перейти до без паперових методів обробки інформації, то веб-технології стають лідером у рішенні цієї задачі. У поєднанні з клієнт-серверною технологією це дозволяє обробляти величезну кількість інформації майже миттєво. Залучення веб-орієнтованих ІС дозволяє здійснювати збереження, обробку та передачу інформації без значних витрат на час, в свою чергу це також дозволяє і позбутися великих матеріальних витрат.

Спираючись на наведені вище твердження, можна зробити висновок, що кожна, без виключення, сфера сучасного життя має потребу у власній інформаційній системі. Виключенням не є і використання подібних ІС у сфері закупівлі господарчих товарів як для дому так і для комерційних потреб, адже саме тут відбувається величезний інформаційний потік, впоратися з яким буває дуже важко.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Дослідження предметної області

Всі інформаційні системи можна представити у вигляді інформаційного довідника й інформаційної бази даних. Кожна з цих систем може включати в себе інші, більш конкретно направлені. Таким чином, для кожної сфери діяльності є своя інформаційна система управління [1].

Головна функція абсолютно кожної такої системи – це збирання, зберігання і пошук інформації. Велика кількість інформації нерідко затрудняє пошук, для цього потрібно багато часу та зусиль. Інформаційні системи управління – це головний помічник для пошуку потрібної інформації. Це дуже швидко, досить зручно і практично. До того ж, інформація в електронному вигляді в найближчому майбутньому замінить паперові документи, оскільки працювати з електронними документами – це в рази простіше, швидше та економніше.

Сьогодні інформаційні системи мають безліч різновидів, але далі слід розглянути саме концепцію веб-орієнтованих ІС оскільки вони є найпопулярнішим різновидом сьогодення. Дані ІС представляють собою клієнт-серверний додаток, у якому клієнтську частину реалізує браузер, який проводить діалог з користувачем і відображає потрібну інформацію. В свою чергу, серверну частину – веб-сервер і сервер додатків, які реалізують основну логіку системи. Через обмежену функціональність клієнта, подібну реалізацію ще називають тонким клієнтом. Більш детально, клієнт-серверну архітектуру зображено на рисунку 1.1.

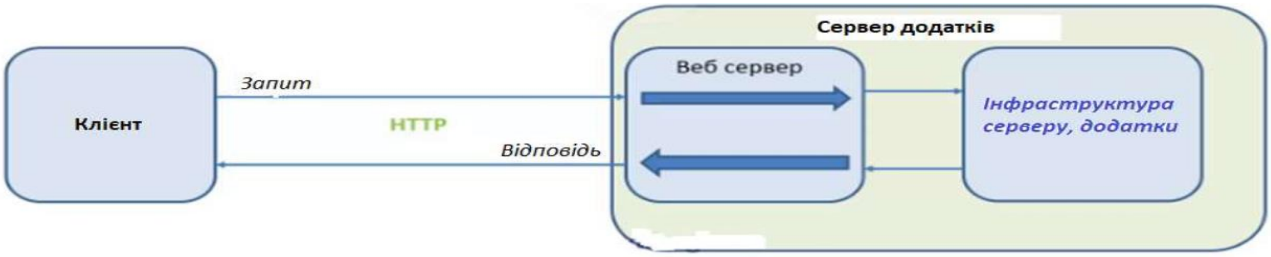

Рисунок 1.1 – Клієнт-серверна архітектура

Безсумнівними перевагами веб-орієнтованої інформаційної системи є наступні речі [2]:

− кросплатформеність – в якості клієнта, як вже було зазначено вище, виступає браузер, а це означає, що для роботи з системою необхідно тільки інтернет-браузер, який входить до складу будь-якої операційної системи, а враховуючи, що сьогодні браузери не залежать від конкретної ОС користувача, веб-орієнтовані системи можна з упевненістю назвати кросплатформеними;

− мобільність – працювати з системою можна в будь-якому місці, де є Інтернет і з будь-якого пристрою, в якому є інтернет-браузер, таким чином клієнт не обмежується вимогами до апаратної частини;

− низька загальна вартість володіння – вартість володіння веб-орієнтованою ІС фактично укладена в розробці, підтримці і розвитку серверної частини (веб-сервера і сервера додатків) і в володінні сервером бази даних системи.

Також, як і будь-які інші інформаційні системи, веб-орієнтована має свої недоліки:

− для роботи з системою необхідно підключення до мережі Інтернет;

− не всі ІС можуть бути замінені на веб-орієнтовані, через обмеженість функціональних можливостей інтернет-браузерів;

− вся інформація, у тому числі особиста інформація користувача, зберігається на сервері.

Для середніх і великих ІС до складу веб-орієнтованої системи зазвичай входить база даних. База даних (БД) – це набір логічно пов’язаних та структурованих даних, які організовані відповідно до концепції та призначених для задоволення інформаційних потреб користувачів. Найпоширеніший різновид – реляційні бази даних. Особливістю даного різновиду БД є те, що в базі наявні взаємозв’язки між елементами.

Розглянувши переваги веб-орієнтованих інформаційних систем можна зробити висновок, що вони є дуже корисними і затребуваними сьогодні. Однією з галузей, у які залучені веб-орієнтовані ІС, є інтернет-магазини та комерція. Ця галузь, як ніхто

інший демонструє потребу в зберіганні величезної кількості інформації та автоматизації управління нею.

На даний момент існує безліч інформаційних веб-систем для комерції і варто розглянути деякі з них, такі як:

– інтернет-магазини – системи, що дозволяють покупцю вручну обрати необхідний товар, додати до кошику та замовити.

– системи управління продажами – системи, призначені для управління товарообігом і призначені в більшій мірі для продавця.

Основною метою впровадження інформаційних систем у продаж є економія дорогоцінного часу, підвищення якості контролю за продажами та спрощення процесу придбання товарів.

1.2 Визначення актуальності роботи

За останніх декілька десятиліть у світі було створено незліченну кількість інформаційних систем для комерції. Найбільш поширеними серед них є інтернет-магазини. Незважаючи на безліч переваг даних систем, є також речі, які дані системи не в змозі контролювати. Для повного розуміння проблеми, варто навести приклад. Наприклад, існує невелике підприємство з фіксованою кількістю робітників (або ж родина). Навіть невелика група людей, не може обійтися без різних господарчих товарів, наприклад: мило, антисептики, миючі засоби, серветки, туалетний папір тощо. Все це відноситься до класу витратних матеріалів, які потрібно замовляти регулярно. За допомогою інтернет-магазину ми можемо створити вручну замовлення, або ж, в більш розвинутих системах навіть зберегти шаблон замовлення, але не можемо розрахувати проміжок часу використання товару і період через який, нам потрібно буде замовити цей товар повторно. Таким чином доречним є створення системи, котра буде запам’ятовувати одиниці товару, період використання, а також створюватиме попереднє замовлення для покупця.

1.3 Аналіз аналогів

У світі існує безліч інтернет-магазинів, які дозволяють обрати потрібні товари, але, як вже було зазначено в попередньому розділі, подібним функціоналом вони не володіють. Це означає, що дана система має бути унікальною та єдиною у своєму роді. Тож варто розглянути інтернет-магазин і знайти спільне з даною системою.

Першим прикладом досить-таки потужної веб-орієнтованої системи інтернет-магазину є Розетка. Головна сторінка сайту відображена на рисунку 1.2.

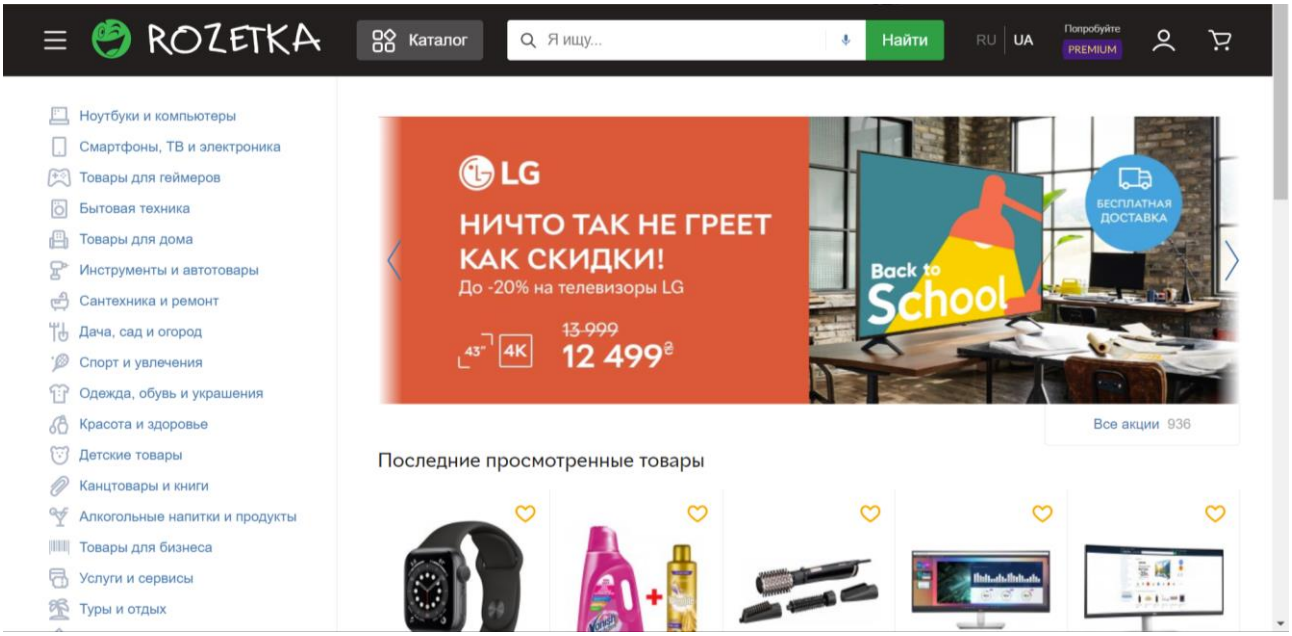

Рисунок 1.2 – Головна сторінка веб-сайту Розетка

Структура сайту є типовою для сайтів цієї ніші. Вгорі розташовано шапку сайту з назвою, а також поле для пошуку по сайту. Зліва розташовано вертикальне навігаційне меню, за допомогою якого здійснюється перехід по сторінкам веб-ресурсу. Основна область, тіло сайту включає в себе інформацію про найбільш актуальні або ж акційні товари.

Для більш детального огляду функціоналу варто переглянути можливості особистого кабінету користувача.

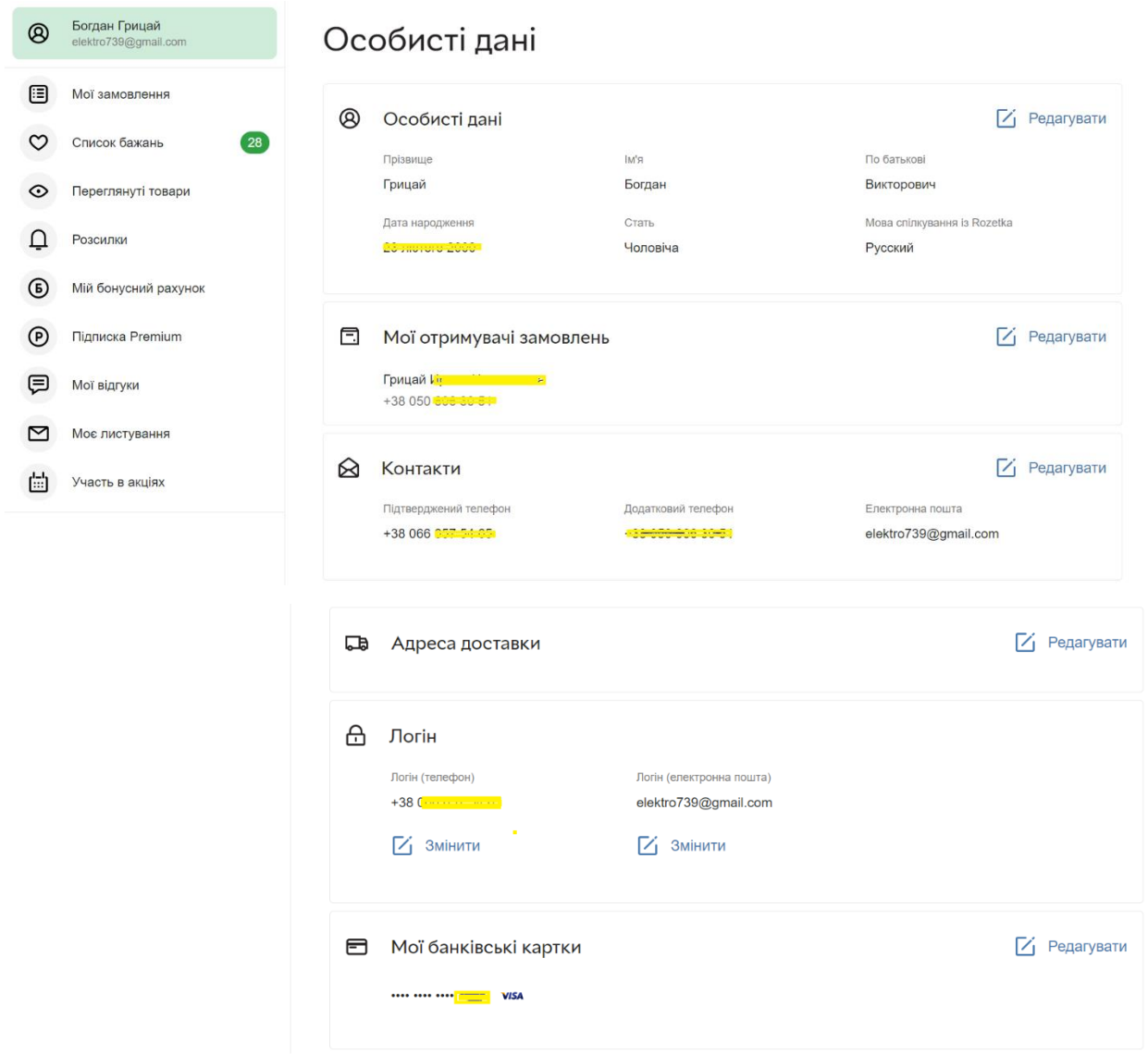

Рисунок 1.3 – Особистий кабінет користувача

Варто звернути увагу на те, що система містить досить широкий функціонал з точки зору користувача:

- особисті дані користувача;
- інші особи для отримання замовлень;
- контакти;
- адреси для доставки;
- банківські карти для оплати замовлень;
- захоплення користувача;
- домашні тварини;

- додаткова інформація (володіння автомобілем можливо вказати ВІН-код для підбору аксесуарів або запчасте саме для даної моделі, наявність дитини, юридична особа);

- можливість пов'язати акаунт з акаунтами в соціальних мережах (Facebook, Google).

Окрім особистих даних система має достатню кількість інших розділів для користувача, серед них:

- замовлення;
- список бажань;
- переглянуті товари;
- розсилки;
- бонусний рахунок;
- відгуки користувача;
- листування користувача з продавцями;
- участь в акціях.

Отже, проаналізувавши можливості особистого кабінету інтернет-магазину Розетка, можна зробити висновок, що для користувача представлено безліч можливостей для менеджменту своїми замовленнями та підбору товарів за інтересами. Але не дивлячись на всі переваги даної системи, вона не має можливості періодично автоматично створювати замовлення хоча б навіть за певним шаблоном і це є основним недоліком даного інтрнет-магазину.

В свою чергу спільним є те, що в веб-системі теж повинні бути такі розділи, як наприклад: особисті дані (ПІБ, дата народження), контактна інформація, адреси доставки, історія замовлень тощо.

1.4 Постановка задачі

Під час виконання даної роботи потрібно реалізувати інтернет-магазин, з наступним функціоналом:реєстрація користувачів у системі з підтвердженням за вказаною адресою електронної пошти;

– авторизація користувачів та можливість запам'ятати користувача на 2 тижні;

– додавання, перегляд та видалення господарчих товарів з продуктового каталогу;

– додавання та видалення товарів з кошику;

– фільтрація товарів за критерієм;

– автоматичне додавання товару шляхом розрахунку часу використання;

– керування особистим кабінетом користувача (зміна паролю, імені, адреси, номеру телефону і т.д.).

– перегляд замовлень та товарів, що доставляються;

– прегляд історії замовлень;

– відправка повідомлення з оформленим замовленням на пошту після підтвердження;

– нагадування про закінчення товару.

Результатом роботи має бути інтернет-магазин інтуїтивно зрозумілий для користучача, з привабливим дизайном, включати все необхідне для створення та роботи з замовленням. ІС має бути швидкою, надійною та мати можливість дял подальшої підтримки. Вся інформація має зберігатися в базі даних та може бути додана системним адміністратором вручну шляхом написання запитів.

2 ВИБІР МЕТОДІВ ВИРІШЕННЯ ЗАДАЧІ

2.1 Визначення середовища планування роботи

Розробка будь-якої інформаційної системи насамперед потребує створення певного плану роботи, задач, термінів та розподілу завдань між собою на певні групи. Більшість цих проблем вирішують саме методології розробки програмного забезпечення. Іншими словами, це деякі принципи, ідеї, способи та засоби, які визначають розробку ПЗ, або ж це ще можна назвати стандартами, яким рекомендовано керуватися під час створювати програмне забезпечення.

Існує багато видів методологій, які в свою чергу поділяються за різними ознаками, наприклад, прогнозовані, гнучкі. Адаптивні і т. д. До методологій розробки ПЗ належать:

– Agile;
– Waterfall;
– V-model;
– Incremental Model;
– RAD Model;
– Iterative Model;
– Spiral Model.

Кожна з наведених вище методологій має свої певні ознаки, переваги та недоліки. Під час виконання аналізу було вирішено використовувати саме методологію Agile Scrum для розробки інформаційної системи. Ця методологія відноситься до гнучких. Деякі принципи та переваги Agile:

– готовність до змін – важливіше за слідування за планом (оскільки інформаційна система є експериментальною і має змінюватися під час тестування: дизайн, додатковий функціонал і т. д.);
– працюючий продукт – важливіший за написання вичерпної документації;
– взаємодія між людьми – важливіше за процеси й інструменти;
– гнучкість.

Існує два найбільш популярні підходи Agile: Scrum та Kanban. Було обрано саме Scrum через те що цей підхід є більш знайомий. Не дивлячись на те, що Scrum є гнучким він є достатньо структурованим. Деякий проміжок часу за, який повинна бути виконанна деяка робота у «Скрамі» називається спринтом. Тривалість спринту залежить від особливостей команди та «скоупу» (масштабу) задач. Зазвичай він складає від 2 тижнів до 1 місяця.

Для інтеграції методології Agile та підходу Scrum у розробку проєкту було обрано безкоштовну версію Atlassian Jira. Саме ця інформаційна система дозволить сторювати задачі, тест кейси, плани для розробки веб-орієнтованої ІС.

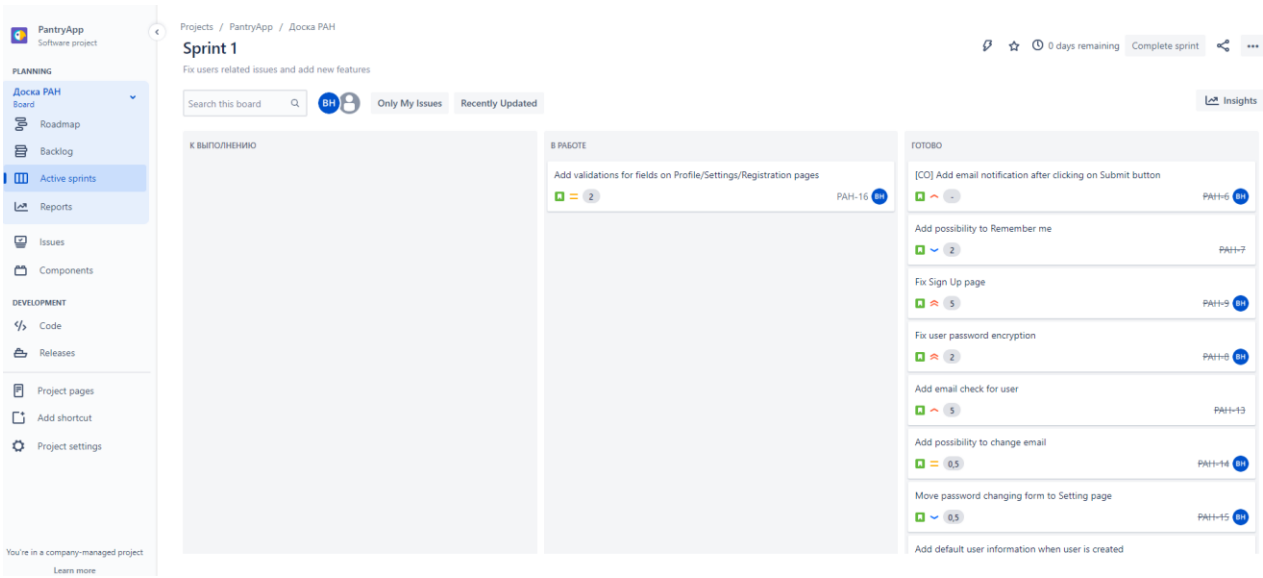

Рисунок 2.1 – Приклад дошки з завданнями поточного спринта

Jira містить дуже багато інформації про проєкт тому варто бути дуже обережним передаючи доступ до проєктів, які керуються нею. На Рисунку 2.1 зображено дошку з завданнями поточного спринта розділених на три колонки по статусам: TO DO, IN PROGRESS та DONE. Зліва від дошки розміщено меню:

– Roadmap – дорожня карта проєкту, яка допомогає відстежити як рухається його розробка, тестування чи підтримка.

– Backlog – усі завдання по проєкту сгруповані по приналежності до певного спринта.

– Active Sprints – вдображення дошки з завданнями поточного спринту (рисунок 2.1).

– Reports – звіти по проєкту.

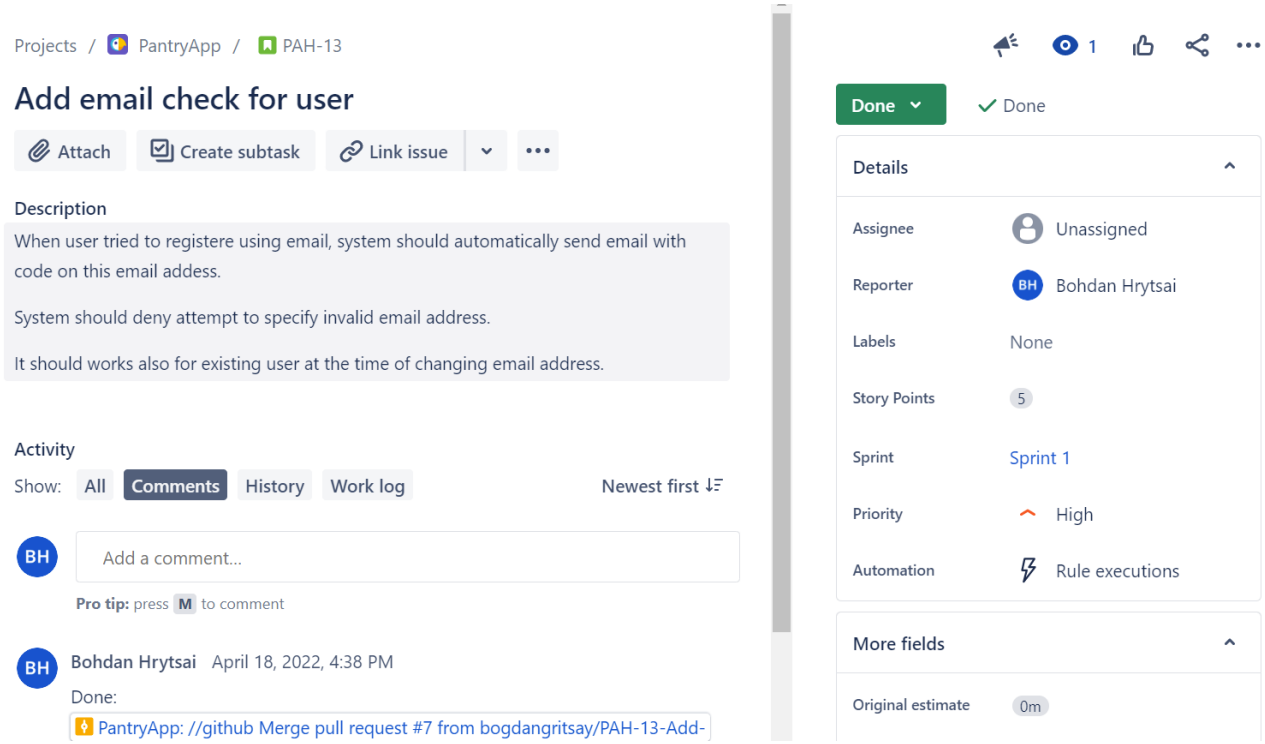

Рисунок 2.2 – Приклад створеної та виконаної задачі в Jira

Як уже було зазначено вище, Jira містить дуже багато інформації про задачі, на рисунку 2.2 можна побачити приклад створеної задачі та інформацію яка наявна про неї (відображення певних відомостей теж можна налаштувати окремо).

2.2 Визначення середовища розробки

Одним з найважливіших етапів створення web-сайту є розробка дизайну. Дуже важливо правильно підібрати гамму для всього сайту і грамотно продумати його структуру. Якісний дизайн повинен створити атмосферу, при якій, представлена на його сторінках інформація буде сприйматися максимально комфортно. Зображення та інші графічні елементи налаштовують відвідувача на потрібний лад, гарно продумана структурна побудова веб-ресурсу допомагає гостю вільно орієнтуватися у його просторі, а текстові блоки легко читаються - таким повинен бути дизайн веб-

сайту. З огляду на те, що веб-сайти створюються для абсолютно різних цілей, напрошується висновок - до розробки веб-дизайну слід підходити індивідуально.

Наступним не менш важливим моментом є вибір движка. Движок - це спеціалізована платформа, на базі якої і будується майбутній веб-ресурс. Ще їх називають CMS (Content management system) системами. Саме за допомогою цих систем здійснюється управління веб-сайтом. На сьогоднішній день існує дуже багато різних систем управління. Серед них є як платні, так і безкоштовні платформи. Вартість більшої частини комерційних (платних) платформ є чисто символічною (виручені від цього кошти йдуть на вдосконалення продукту).

Характеристики CMS:

– універсальність і багатозадачність - здатність створювати проекти різних типів;

– зручність - деякі движки дуже прості в роботі (Wordpress, Opencart), а інші потребують додаткового часу на вивчення;

– ціна – є багато безкоштовних, але є і дорогі платні рішення;

– гнучкість - CMS не вимагає великих ресурсів сервера, добре працює з кешем;

– структура - бувають монолітні системи, які практично не допускають змін, і модульні движки, де легко додати потрібний функціонал через різні плагіни і доповнення;

– безпека - в різних CMS до питання надійного захисту підходять по-різному.

– оптимізація - наскільки швидко працює система і її адаптованість під критерії пошукових систем.

При виборі системи CMS для свого сайту звертають більше уваги не на її вартість або вільне (безкоштовне) поширення, а також і на зручність роботи з нею. Сучасні системи CMS представлені на рисунку 2.3.

Рисунок 2.3 – Сучасні системи CMS

Для створення сучасних сайтів зазвичай використовуються: HTML, CSS, JavaScript, PHP, Java і, як правило, реляційні бази даних SQL.

Клієнтське середовище (браузер) є переднім краєм роботи програми. У цьому середовищі, відображаються HTML-сторінки у вікні і обслуговуються історії сеансів HTML-сторінок, що відображаються в браузері протягом сесії. Об'єкти цього середовища, зобов'язані мати можливість маніпулювати сторінками, вікнами та історією.

HTML - це не мова програмування і не мова оформлення документів. Це, в першу чергу, засіб розмітки тексту. Мова HTML містить достатню кількість елементів, що дозволяють оформити документ на будь-який смак. Якщо не влаштовує спосіб оформлення документа браузером (хочеться змінити шрифт, запропонований за умовчанням, або зробити його подрібніше) – потрібно скористатися таблицями стилів (CSS) [10].

Динамічний HTML - це комерційний термін, придуманий для опису технологій, які були введені в четвертій версії веб-браузерів і дозволяли обходити обмеження HTML.

DHTML являє собою комбінацію таких веб-стандартів:

CSS + JavaScript + Java + DOM + HTML = DHTML, де:

– CSS - визначає атрибути об'єктів,

– JavaScript - допоміжна мова роботи з блоками;

– Java – відповідає за динамічне наповнення блоків різноманітним контентом, а також за весь backend веб-системи.

– DOM - визначає ієрархію об’єктів;

– HTML - створює об'єкти (виконує розмітку тексту).

Наведені вище технології є досить потужними, але в XXI столітті цього замало для створення дійсно сучасного веб-сайту з можливістю подальшої підтримки. Для цього існують веб-фреймворки. Веб-фреймворк - інструмент, який полегшує процес написання і запуску веб-додатку. Використовуючи них не потрібно самостійно писати купу повторюваного коду і витрачати час на пошук потенційних помилок.

У фреймворків є дві основні функції: робота на серверній стороні (backend) і робота на клієнтській стороні (frontend).

Frontend-фреймворки пов'язані з зовнішньою частиною програми. Простими словами, вони відповідають за зовнішній вигляд програми. Backend відповідає за внутрішній устрій додатку.

Правила та архітектура серверних фреймворків не дає можливості створити веб-додаток з багатим інтерфейсом. Вони обмежені в своїй функціональності, проте ви все одно можете створювати прості сторінки і різні форми. Також вони можуть формувати вихідні дані і відповідати за безпеку в разі атак. Все це може суттєво спростити процес розробки. Серверні фреймворки в основному відповідають за окремі, але критично важливі частини програми, без яких додаток не зможе нормально працювати. До найпопулярніших серверних фреймворків належать: Django (Python), Zend (PHP), Node.js (JavaScript) та Spring (Java).

На відміну від серверних, клієнтські фреймворки ніяк не пов'язані з логікою програми. Цей тип фреймворків працює в браузері. З їх допомогою можна поліпшити і впровадити нові призначені для користувача інтерфейси. Frontend-фреймворки

дозволяють створювати різну анімацію і односторінкові додатки. Всі клієнтські фреймворки відрізняються по функціоналу. Наприклад, Angular, Ember.js, React, Vue.js, Bootstrap.

Незважаючи на те, що всі фреймворки відрізняються один від одного і вибрати який-небудь з них може бути дуже складно, є кілька речей, загальних для них усіх. Мова йде про архітектуру і особливості, які так само важливі, як і функції.

Архітектура майже всіх фреймворків заснована на декомпозиції декількох окремих шарів (додатки, модулі і т.д.), це означає, що можна розширювати функціональність виходячи зі своїх потреб і використовувати змінену версію разом з кодом фреймворка або використовувати сторонні додатки. Така гнучкість є ще одною ключовою перевагою фреймворків. Існує безліч open-source спільнот і комерційних організацій, які створюють програми або розширення для популярних фреймворків, наприклад, Django REST Framework, Bootstrap і т.д.

Для розробки веб-орієнтованої інформаційної системи було обрано Java-фреймворк Spring MVC [3]. Особливістю даного фреймворку є підтримка Spring архітектури модель-уявлення-контролер (model-view-controller).

Головною метою MVC є поділ об'єктів, бізнес-логіки й зовнішнього вигляду програми. Всі ці компоненти слабо пов'язані між собою і при бажанні ми можемо змінити, наприклад, зовнішній вигляд програми, не вносячи суттєвих змін в інші два компонента [11].

Модель містить всі дані і рівні бізнес-логіки, її правила і функції. На практиці це - POJO-класи (Plain Old Java Objects).

Модуль уявлення відповідає за виведення даних користувачеві. Зазвичай це JSP-файл, який може бути пізнаний і інтерпретований браузером на клієнтській машині або ж у даному випадку – це шаблонізатор Freemaker (для HTML) [8].

Контролер відповідає за обробку запитів користувачів і передачу даних модулю View для обробки. Дані складові є залежними один від одного, тому важливо як слід у всьому розібратися, щоб уникнути помилок під час роботи програми.

В основі Spring MVC Framework лежить DispatcherServlet [9], завдання якого - обробка всіх HTTP запитів і відповідей. Архітектуру Spring MVC веб-додатку представлено на рисунку 2.4.

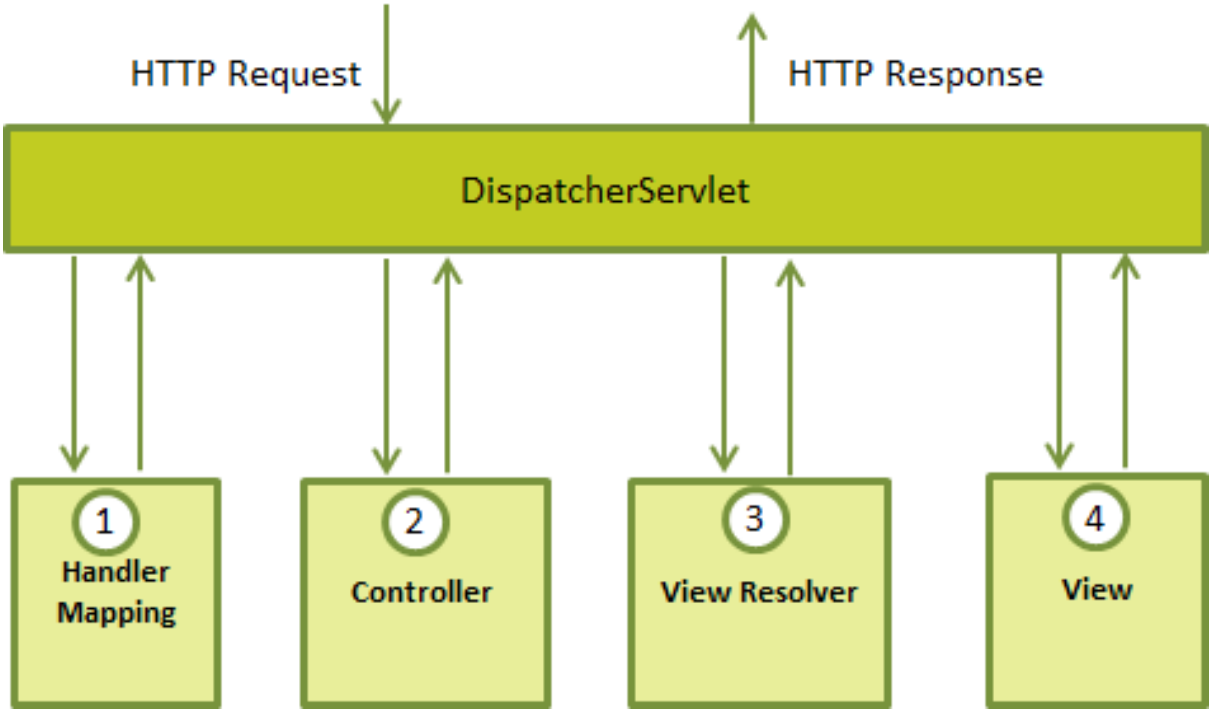

Рисунок 2.4 – Архітектура Spring MVC додатку

Для створення привабливого дизайну веб-системи буде використано фреймворк Bootstrap. Bootstrap – це вільний набір інструментів для створення сайтів і веб-додатків. Включає в себе HTML- і CSS-шаблони оформлення для веб-форм [7], кнопок, міток, блоків навігації та інших компонентів веб-інтерфейсу, включаючи JavaScript-розширення. Це значно спростить написання CSS коду та зменшить його кількість, шляхом уже прописаних шаблонів. Не дивлячись на використання Bootstrap, для більшої привабливості, стилі CSS фреймворку будуть перевизначені в деяких місцях в той час як сторінки HTML (Thymeleaf) розмітка буде створена вручну [4].

Очевидно, що жодна сучасна інформаційна система не обходиться без використання баз даних. В даному випадку буде використано мову SQL, а саме діалекту PostgreSQL. Рішення використовувати цей діалект, прийнято з полглядом на

те, що не потрібно налаштовувати складні конфігураціїї (наприклад, як в Oracle), Postres займає відносно невелику кількість місця, а також підтримується майже всіма сучасними платформами для розгортання веб-орієнтованих інформаційних систем на мові Java. Оскільки даний проєкт буде оптимізуватися, доповнюватися та підтримуватися в майбутньому, то варто звернути увагу на інші переваги й недоліки даної субд. До переваг належить:

PostgreSQL – безкоштовне ПЗ з відкритим вихідним кодом. Вона випускається на умовах ліберальної ліцензії PostgreSQL. СУБД не може бути монополізовано та придбано, що дає компаніям наступні переваги:

- ліцензування є безкоштовним;
- кількість розгорнутих екземплярів PostgreSQL не є обмеженою;
- вигідніша бізнес-модель;
- нескладна для освоєння, а перенесення коду з іншого СУБД коштує недорого;
- сумісна зі стандартами SQL, тому неважко знайти професійних розробників;
- адміністрування PostgreSQL легко автоматизувати, що дозволяє суттєво заощадити на зарплаті персоналу;
- добре масштабується та забезпечує високу продуктивність;
- є кроссплатформеним ПЗ;
- надійна;
- підтримує ACID.

Оскільки однозначної думки щодо недоліків Postgres SQL не існує, то варто навести лише ті, які є більш значимими.

У найпростіших операціях читання продуктивність цієї СУБД поступається іншим. Також недоліком можна вважати складність системи, її налаштування. Якщо це нескладний проект із відносно невеликою базою даних, за якою планується проводити складну аналітику, PostgreSQL краще не використовувати. Також PostgreSQL не має надійного способу оновлення до основних випусків. Ця СУБД не

для новачків, але сьогодні вона є одною із найпопулярніших та майже кожен досвідчений спеціаліст по бекенду на Java має досвід з Postgres.

2.3 Розробка прототипу дизайну

Дизайн – перше, на що звертає увагу відвідувач веб-сайту. Саме тому він має бути інтуїтивно зрозумілим, сучасним та не повинен заважати інформативності контенту на сторінках веб-ресурсу. Розробка дизайну - один з найважливіших етапів розробки інформаційної системи, що включає в себе:

– побудову макету;

– побудову прототипу сторінок веб-сайту;

– верстку та стилізацію згідно прототипу.

Для зручності верстки та впровадження динамічного контенту було вирішено взяти за основу сучасний серверний механізм Java-шаблонів для веб-середовищ – Freemaker у поєднанні з HTML, CSS, JS-скриптами та фреймворком Bootstrap.

Загальний макет сторінок веб-сайту зображено на рисунку 2.5.

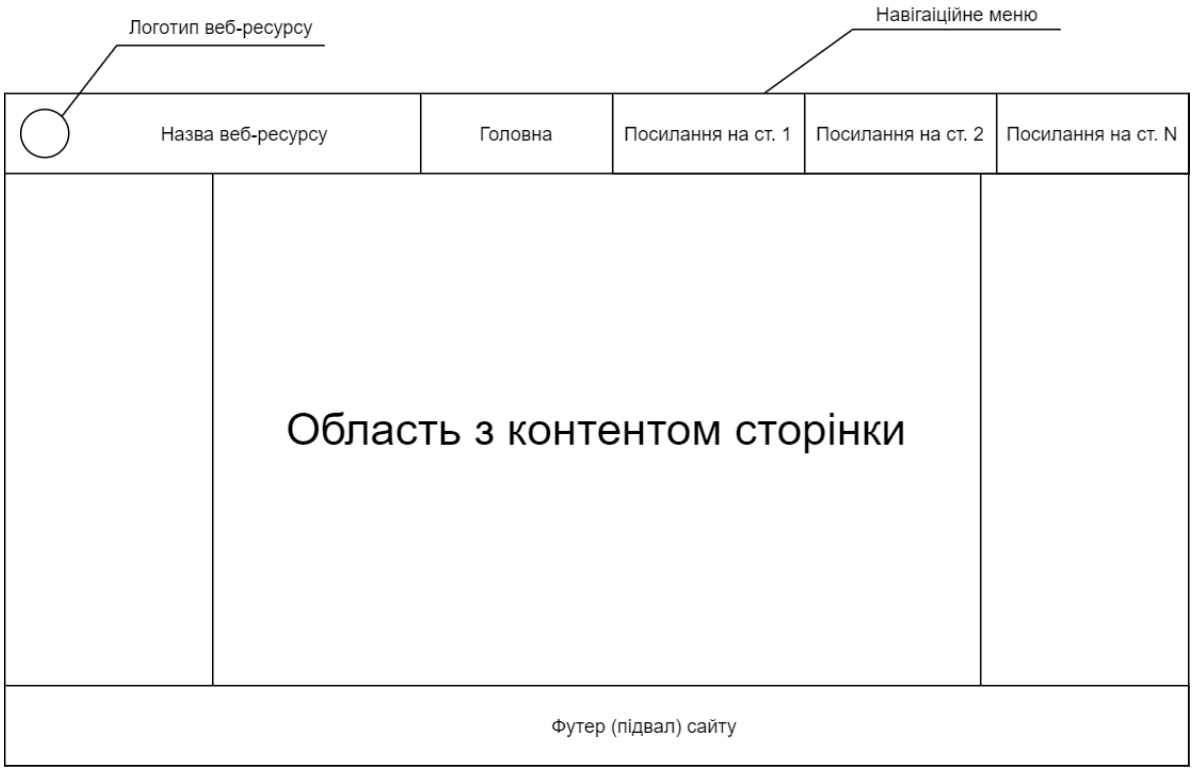

Рисунок 2.5 – Загальний макет сторінки веб-сайту

Варто зазначити, що наведений вище дизайн є лише прототипом і може бути змінений у ході реалізації даної веб-системи.

Для створення дизайну веб-сайту буде також використано фреймворк Bootstrap. Фреймворк допомогає створювати сучасні та адаптивні сторінки. Вбудовані стилі дозволяють оформити елементи лише додаванням декількох класів, але в більшості випадків потрібно підлаштовувати таблицю стилів під свої потреби шляхом написання властивостей для класів.

Однією з багатьох переваг даного фреймворку є те, що він надає свої класи оформлення для створення безлічі різних компонентів, ось деякі з них:

- кнопки;
- картки;
- підказки;
- списки (таблиці);
- форми;
- навігаційні меню;
- модальні вікна.

Для демонстрації переваг та можливостей фреймворку слід розглянути деякі приклади класів для оформлення елементів.

Для більш повного розуміння поняття дизайну та прототипу, а також безпосередньо самої інформаційної системи варто переглянути прототип дизайну даної інформаційної системи.

Прототип головної сторінки інформаційної системи буде виглядати приблизно так як показано на рисунку 2.6.

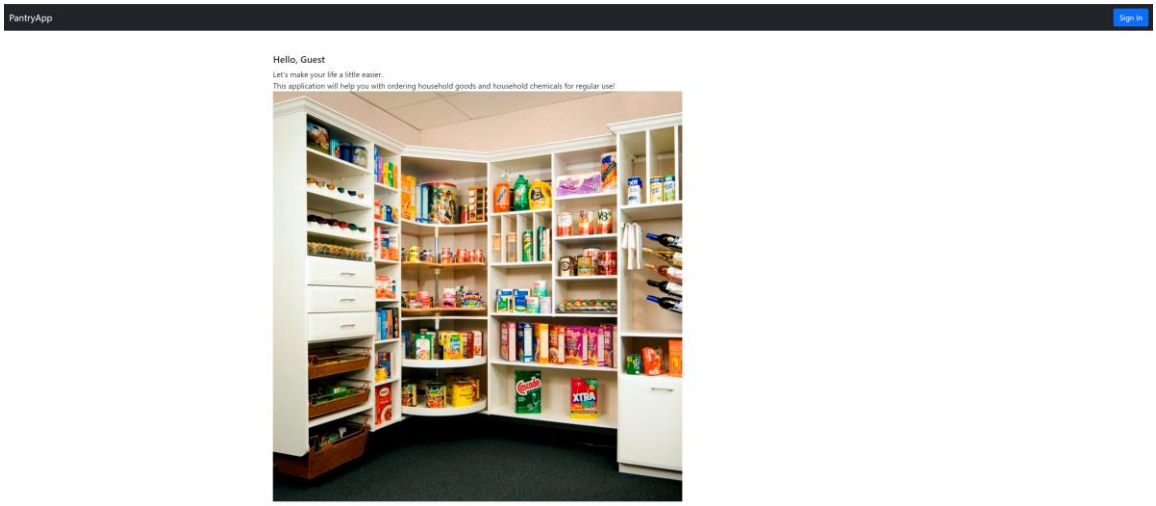

Рисунок 2.6 – Прототип головної сторінки ІС

Жоден веб-сайт не може існувати без привітання. Адже потрапляючи на сторінку ІС користувач мусить знати та розуміти де він знаходиться. Саме для цього було створену таку сторінку, де користувач зможе побачити назву ресурсу, коротке привітання, навігаційне меню, зображення для більшої привабливості системи та кнопки для можливості авторизації.

Після натискання на кнопку Sign In, гість потрапляє на форму авторизації, де повинен ввести коректний логін та пароль (рисунок 2.7). У разі заповнення полей невірними даними, користувач повинен побачити повідомлення про неуспішну авторизацію.

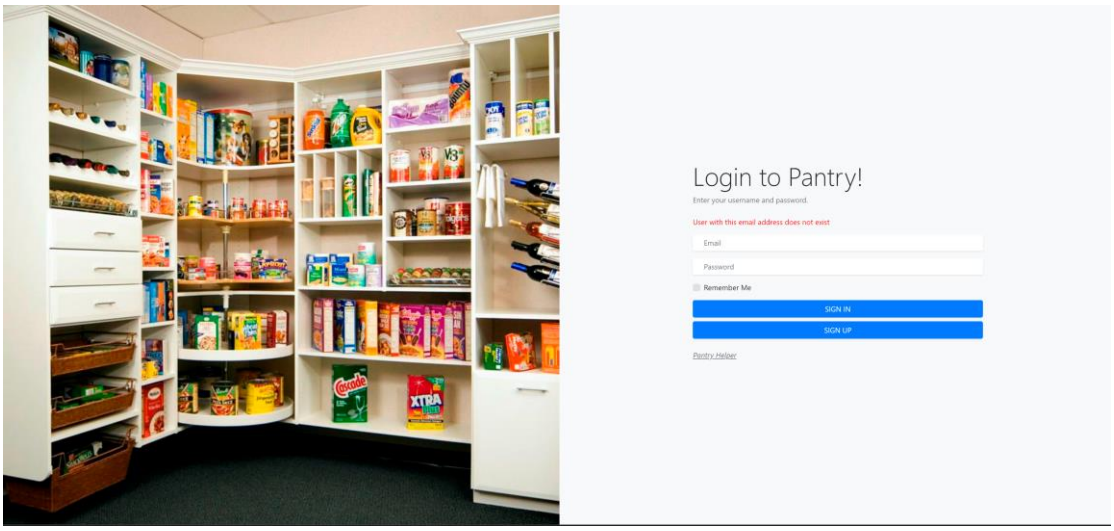

Рисунок 2.7 – Форма авторизації

Система призначена для багатьох юзерів, які можуть асинхронно користуватись функціоналом тому має бути сторінка реєстрації з полями для введення необхідних даних і відповідною валідацією до цих полей (рисунок 2.8).

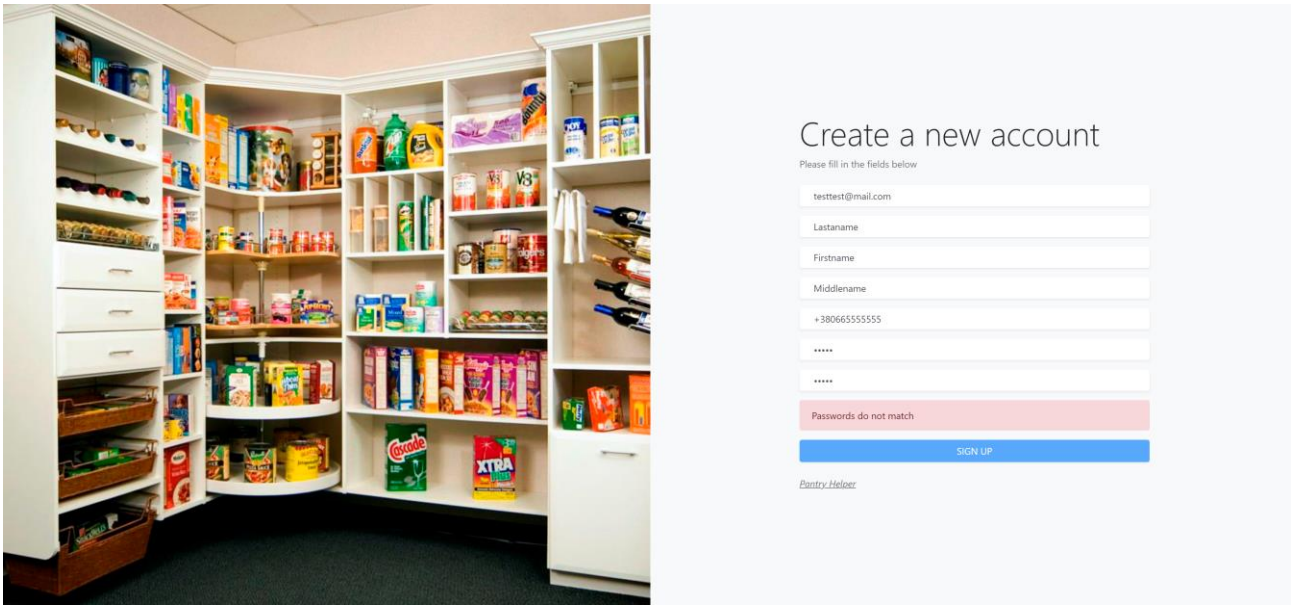

Рисунок 2.8 – Сторінка реєстрації

Після успішної авторизації користувач потрапляє на ту ж саму сторінку з привітанням, але з навігаційним меню веб-додатка.(рисунок 2.9). При натисканні на гіперпосилання з назвою розділу, юзер може потрапити на сторінку відповідну сторінку.

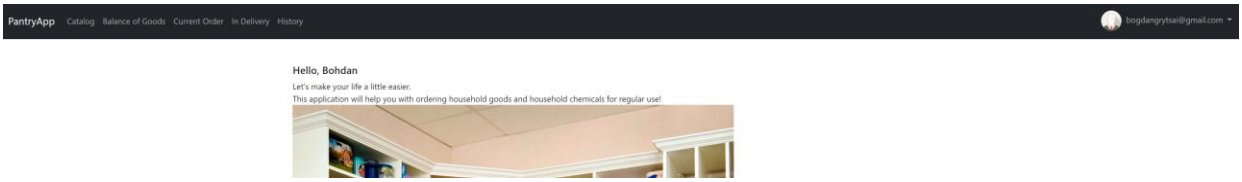

Рисунок 2.9 – Сторінка з залишком товару

Розділ з каталогом продукції (рисунок 2.10) містить коротку інформацію про товари, список яких можна відфільтрувати за обраним критерієм.

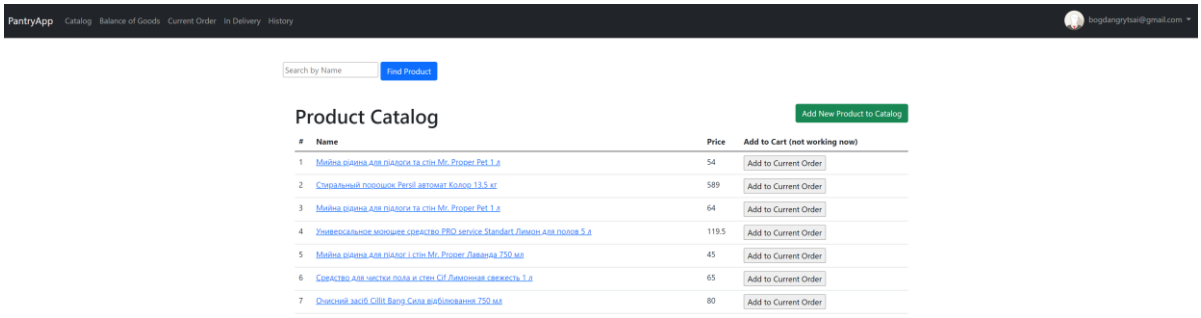

Рисунок 2.10 – Сторінка з каталогом товарів

Також можна додати новий товар до списку натиснувши на кнопку «Add New Product to Catalog». Перед користувачем з’явиться форма для додавання нового товару (рисунок 2.11).

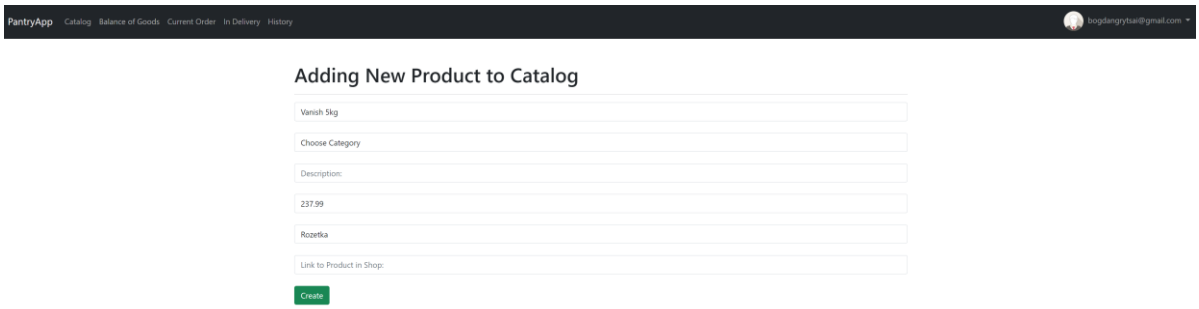

Рисунок 2.11 – Сторінка додавання нового товару

У наступному розділі «Balance of Goods» користувач може ознайомитися з товарами, які замовлялися раніше і уже наявні в його «коморі». Також тут відображено прогнозований час, за який буде використано конкретний товар та кількість днів до закінчення (рисунок 2.12).

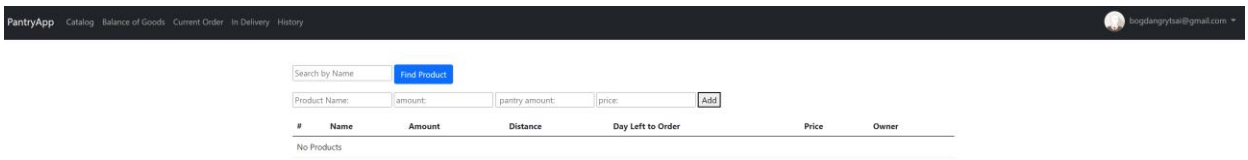

Рисунок 2.12 – Сторінка з наявними у користувача товарами

У розділі Current Order подано інформацію про додані до кошика товари як вручну так і автоматично на основі розрахунку часу використання товару. Звідси

можна створити замовлення натиснувши відповідно кнопку або ж видалити деякі товари з кошика (рисунок 2.13).

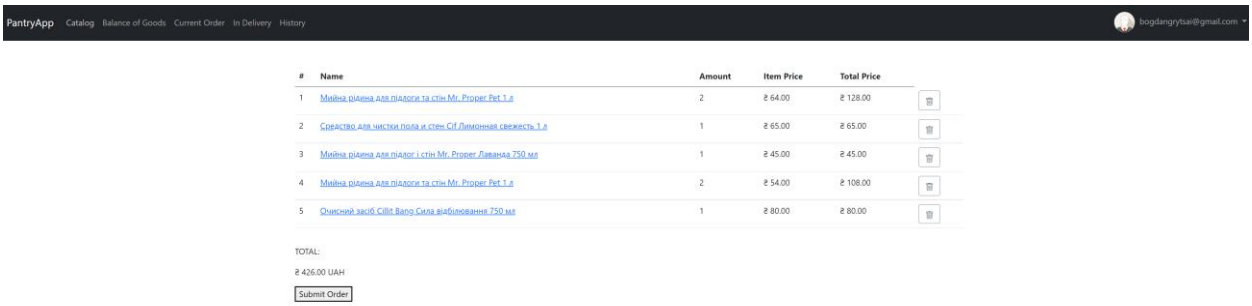

Рисунок 2.13 – Сторінка з поточним замовленням

Для перегляду інформації про раніше замовлені товари, які ще не дійшли до користувача, потрібно перейти на сторінку «In Delivery», де можна побачити вартість товару за одиницю, його назву, загальну вартість, розраховану дистанцію використання, а також власника (рисунок 2.14). Також тут користувач знайде загальну вартість замовлення та кнопку, яку повинен натиснути після успішного отримання товару.

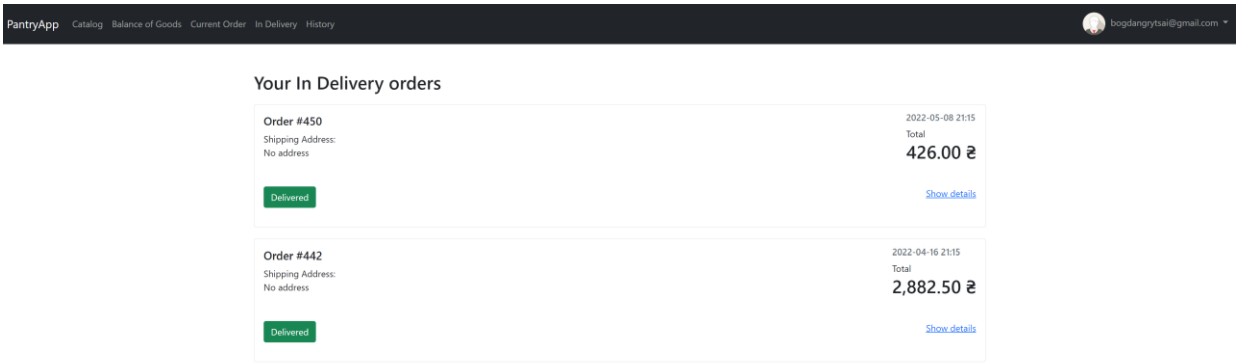

Рисунок 2.14 – Сторінка з товарами, які доставляються

Як будь-яка інформаційна система для керування замовлення, користувач повинен мати змогу переглянути історію своїх замовлень. Сторінка з історією замовлень користувача зображена на рисунку 2.15.

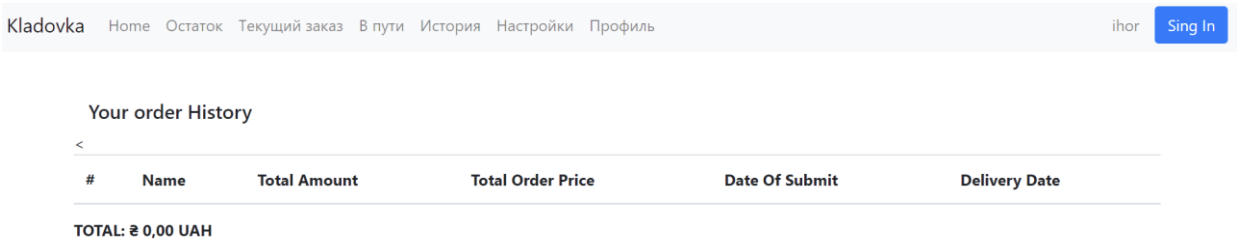

Рисунок 2.15 – Історія замовлень

На наведеній вище сторінці можна дізнатися таку інформацію як: назву замовлення, загальну кількість одиниць, загальну вартість, дату створення замовлення і дату отримання замовлення користувачем.

Будь-яка інформаційна система, котра має розбиття на користувачів повинна надавати змогу налаштовувати обліковий запис та змінювати пароль. Сторінка з налаштуванням облікового запису (рисунок 2.16 – 2.17) має містити електронну адресу користувача, адресу та номер телефону. Це мінімальний набір характеристик, котрий повинна містити дана сторінка.

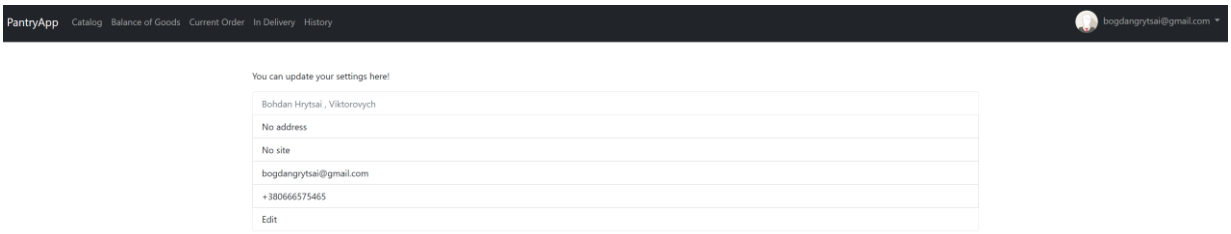

Рисунок 2.16 Налаштування користувача

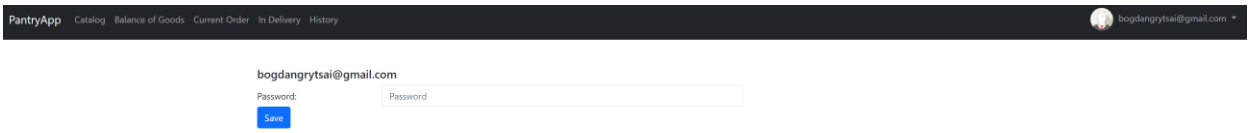

Рисунок 2.17 – Сторінка зміни паролю користувача

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проектування структури інформаційної системи

Розроблена інформаційна система повинна містити певний функціонал розділений на відповідні розділи. Це потрібно для того, щоб розподілити функціонал за певною ознакою: робота з корзиною, каталогом, налаштування користувача тощо. Карта сайту для зареєстрованого користувача представлена на рисунку 3.1.

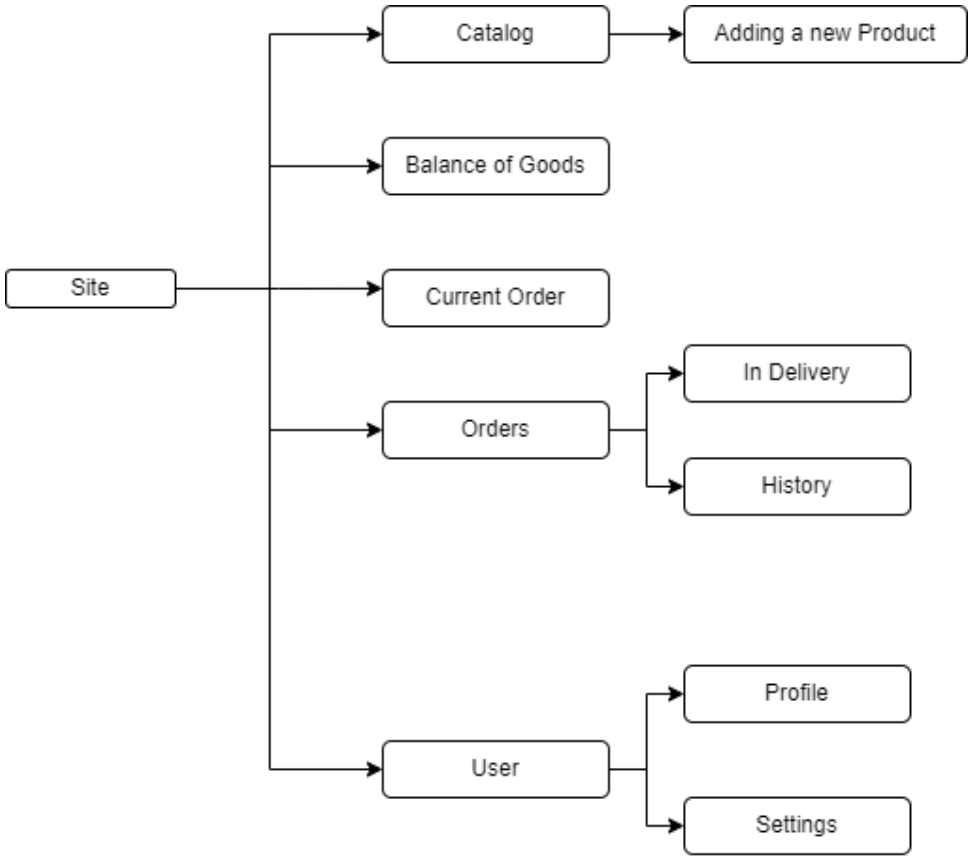

Рисунок 3.1 – Структура інформаційної системи

Незареєстрованому користувачу буде доступна лише головна сторінка з привітанням і форми реєстрації та авторизації. Для кращого розуміння варто коротко описати кожен із розділів веб-сайту:

– головна сторінка – привітання з найнеобхіднішою інформацією про інформаційну систему;

– «Catalog» - розділ з загальним каталогом та можливістю додавати нові за допомогою форми додавання.

– «Balance of Goods» - розділ з інформацією про наявні у користувача товари;

– «Current Order» - розділ з поточним замовленням користувача (кошик);

– «Orders» - розділ з інформацією про створені замовлення користувача: виконані та ті, що знаходяться в процесі доставки;

– «User» - розділ з інформацією про користувача та усім відповідним функцоналом (зміна паролю, адреси, номера телефону, імені тощо).

Після авторизації користувач матиме змогу створювати замовлення, додавати товари до каталогу, відстежувати статуси замовлень та керувати ними.

3.2 Проектування бази даних та інтеграція з фреймворком

Для збереження та зручного доступу до даних, з якими працюватиме система потрібно створити базу даних у якій зберігатимуться таблиці з потрібною інформацією. Концептуальна модель бази даних (ER–діаграма) представлена на рисунку 3.2.

База даних інформаційної системи на етап розробки тестової версії складається з 9 таблиць:

– Таблиця «Category» - таблиця, яка міститиме інформацію про категорії товарів представлених у каталозі.

– Таблиця «Stores» - таблиця, яка міститиме інформацію про таблиця, яка міститиме інформацію про інтернет-магазини, товари з яких, представлені у каталозі;

– Таблиця «Products» - таблиця, яка міститиме інформацію про товари, представлені у каталозі;

– Таблиця «Users» - таблиця, яка міститиме інформацію про користувачів системи;

– Таблиця «Orders» - таблиця, яка міститиме інформацію про замовлення користувачів та їх статуси;

– Таблиця «Order_items» - таблиця, яка міститиме інформацію про позиції в замовленні та інформацію про них (кількість, сумарна вартість, приналежніть певному замовленню);

– Таблиця «Pantries» - таблиця, яка міститиме інформацію про так звану «комору» користувача, тобто про те, що користувач має в наявності та використовує;

– Таблиця «Pantry_items» - таблиця, яка міститиме інформацію про позиції товарів у коморі користувача (кількість і т. д);

– Таблиця «Confirmation_token» - таблиця, яка міститиме інформацію про токени підтвердження електронної пошти користувача (дата та час «згорання», створення, підтвердження і т. д.)

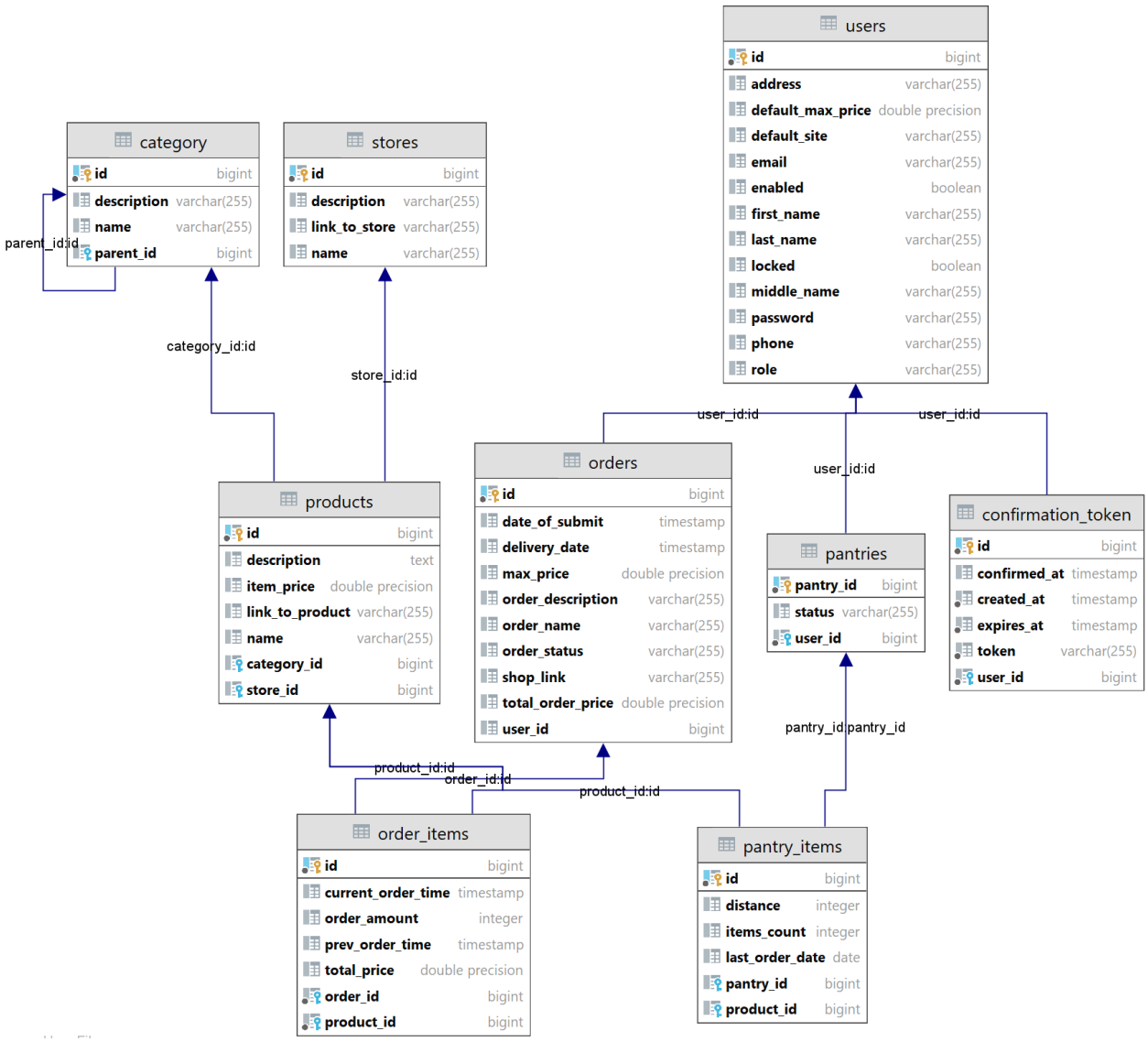

Рисунок 3.2 – Концептуальна модель бази даних

Варто зазначити, що ролі користувачів будуть зберігатись безпосередньо в таблиці «Users» (рисунок 3.3). Цей факт порушує Другу нормальну форму, але в даному контексті розділення на декілька таблиць не є необхідним, адже система буде мати фіксовану кількість ролей і ця інформація (роль) має безпосереднє відношення до сутності «Користувач».

locked	middle_name	password	phone	role
false	Viktorovych	$2a$10$1gbOblzweGrrrJWuIlIq6evJJ22Mh5GgaQsy9ui3kXObfJA…	+380666575465	USER
false		$2a$10$7Bxf1YVSHgVCFPNQsr/lOezynYfDfzTcC5PhHcej8w5SJmu…	+380666693635	USER
false		$2a$10$feWTDLWNjGFzpvXEbnsQOusSYjBW/yDjI/NpPcFOgOa5hhU…	+380698563214	USER

Рисунок 3.3 – Розміщення ролі користувача в таблиці «Users»

Також для забезпечення унікальності рядків таблиць було вирішено використати «sequences» (послідовності) – це механізм у деяких базах даних, який дозволяє автоматично генерувати послідовності чисел та використувати числа з послідовності у якості ідентифікатора рядка певної таблиці (рисунок 3.3).

```
create sequence confirmation_token_sequence;
create sequence user_id_sequence;
create sequence stores_id_sequence;
```

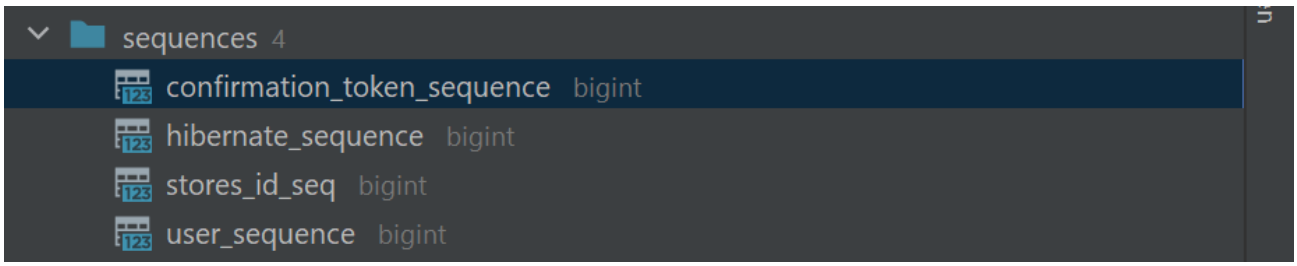

Рисунок 3.3 – «Sequences» бази даних

Як було зазначено раніше, робота з базою даних в даному модулі реалізована через Java Persistence API. Важливо додати, що класи. Які відображають сутності бази даних розміщено в окремому модулі (рисунок 3.4).

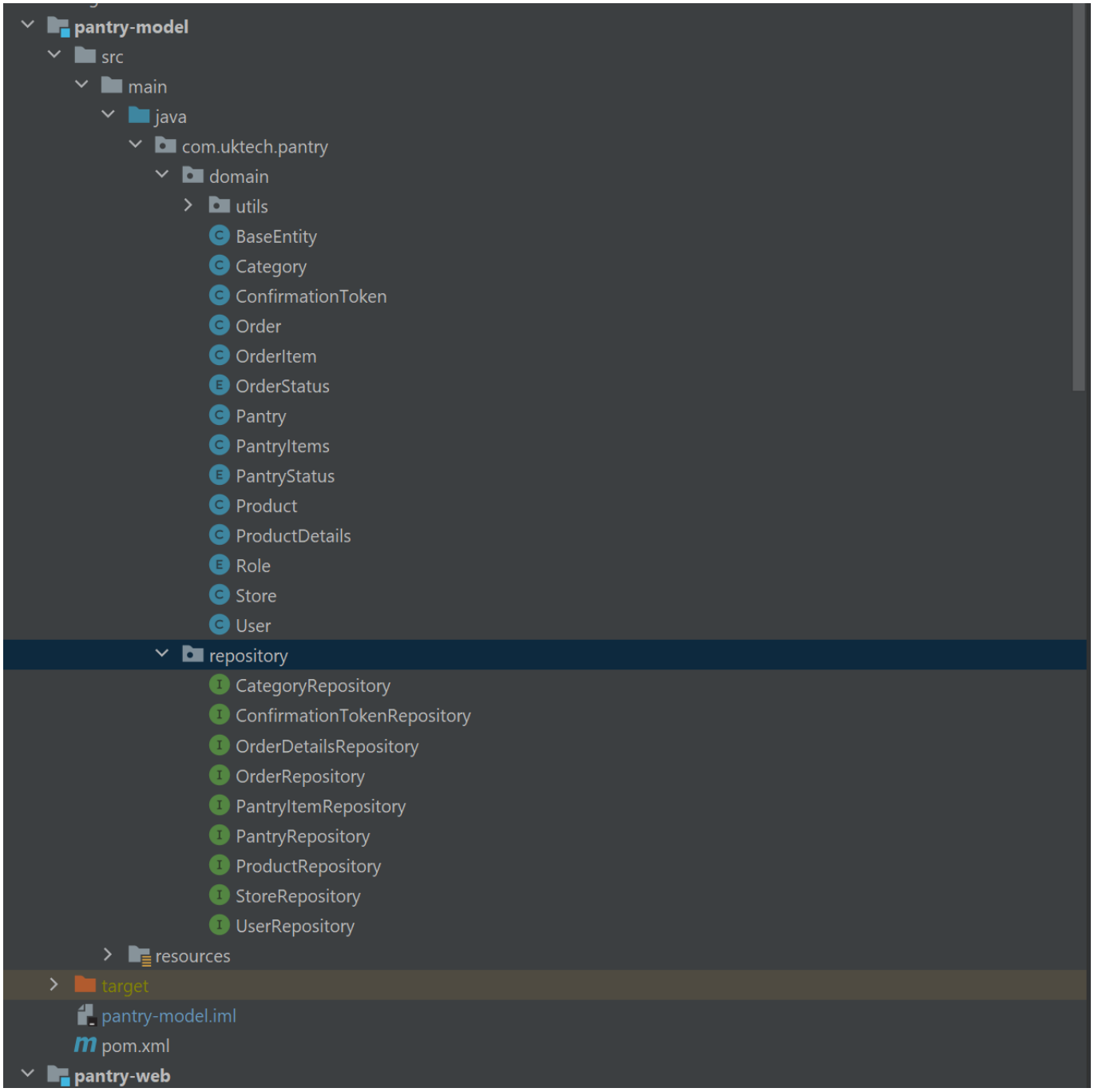

Рисунок 3.4 – Модуль «pantry-model»

Взаємодія веб-додатку з базою даних включає в себе три важливі складові:

- модель – представлення сутності БД у вигляді POJO класів;
- репозиторій (Repository) – інтерфейс з основними методами взаємодії з сутністю;
- сервіс (Service) – клас, який обробляє отриману з репозиторію інформацію для використання у бізнес-логіці.

Для наглядності варто розглянути кожну з наведених вище складових. Для демонстрації можливостей варто взяти складові, які відносяться до різних сутностей.

Модель сутності Product складається з полів та необхідних методів для роботи з об’єктом. Код даної сутності представлено на рисунку 3.5. У даному прикладі немає геттерів та сеттерів, адже вони створюються автоматично за допомогою залежності Lombok, яка надає безліч корисних можливостей для розробки додатків мовою Java.

```java
@Data
@Entity
@Table(name = "products")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @EqualsAndHashCode.Exclude
    private Long id;
    private String name;

    @Column(columnDefinition="TEXT")
    private String description;

    @ManyToOne
    @JoinColumn(name = "category_id")
    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    private Category category;

    private Double itemPrice;

    @ManyToOne
    @JoinColumn(name = "store_id")
    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    private Store store;

    private URL linkToProduct;


    public Product (String name, String description, Category category, Double itemPrice, Store store, URL linkToProduct)
        this.id = id;
        this.name = name;
        this.description = description;
        this.category = category;
        this.itemPrice = itemPrice;
        this.store = store;
        this.linkToProduct = linkToProduct;
    }

    public Product() {}
```

Рисунок 3.5 – Код сутності «Product»

Можна помітити, що для даного класу використовується декілька анотацій:

– `@Data` – анотація від Lombok, яка надає класу можливості методів `@ToString`, `@EqualsAndHashCode`, `@Getter`, `@Setter` та `@RequiredArgsConstructor`;

– `@Entity` – вказує, що даний бін (клас) є сутністю;

– `@Table(name = "products")` – вказує на ім'я таблиці, яка буде відповідати даній сутності.

Окрім анотацій проставлених над класом сутності для роботи з JPA використовуються інші анотації призначені для полів класу (таблиці):

– `@Id` – вказує на те, що дане поле буде використовуватися як ключ таблиці;

– `@GeneratedValue(strategy = GenerationType.AUTO)` - обирає стратегію призначення унікального значення (може бути SEQUENCE і т. д.);

– `@EqualsAndHashCode.Exclude` - вказує Lombok'у на те, що дане поле потрібно вилучити з методу `equals` та `hashCode`;

– `@Column(columnDefinition="TEXT")` – явно встановлює тип для поля таблиці;

– `@ManyToOne` – вказує тип залежності між сутностями «багато-до-одного»;

– `@JoinColumn` – дозволяє явно вказати назву поля для зв'язку;

– `@ToStringExclude` - вказує Lombok'у на те, що дане поле потрібно вилучити з методу `toString`;

– `@Transient` – не використовується у наведеному прикладі на рисунку 3.5, але варто розуміти, що ця анотація дозволяє не створювати поле помічене нею у таблиці бази даних.

Наступною складовою є Repository, приклад для сутності OrderDetails зображено на рисунку 3.6.

```java
public interface OrderDetailsRepository extends JpaRepository<OrderItem, Long> {

    @Query(value = "select oi.order_amount from order_items oi where oi.id = ?1", nativeQuery = true)
    Integer findOrderAmountById(Long id);

    @Query(value = "select oi.total_price from order_items oi where oi.id = ?1", nativeQuery = true)
    Double findTotalPriceById(Long id);

    OrderItem findOrderItemById(long id);

    Collection<OrderItem> findOrderItemByOrderId(Long id);

}
```

Рисунок 3.6 – Приклад репозиторію

Даний інтерфейс лише надає сигнатури методів з анотаціями, які за допомогою фреймворку Hibernate перетворяться на справжні методи під час виконання програми, але важливо пам’ятати, що кожен метод для вибірки має починатися словом «find» після чого потрібно вказати критерії (назви полів) для пошуку так, як це визначено у моделі. Також даний інтерфейс наслідує JpaRepository і в дженерику вказано клас моделі, з яким працюватиме репозиторій і тип головного ключа. Якщо можливостей фреймворку недостатньою для створення коректної вибірки, то в анотації `@Query` можна явно вказати параметризований запит і додати властивість `“nativeQuery=true”` для того, щоб Hibernate розумів запити написані на нативною мовою SQL.

Наступною складовою є Service, приклад зображено на рисунку 3.7.

Рисунок 3.7 – Приклад сервіс

Саме тут веб-додаток використовує отримані репозиторієм дані з бази даних і проводить необхідні маніпуляції з ними для використання, наприклад, у контролері. Кожен сервіс помічається анотацією `@Service` і в даному випадку має два поля помічені анотацією `@Autowired`, яка дозволяє не створювати бін вручну, а отримати з ApplicationContext (особливість фреймворку Spring). Також даний клас має методи для обробки даних і виконання, якихось дії щодо них або ж отримання цих даних іншому місці.

3.3 Реалізація ключових частин функціоналу

В даному розділі варто розглянути декілька ключових частин функціоналу – як вони працюють в коді та що, бачить користувач в UI, а саме:

- додавання нового товару до каталога;
- додавання товару до поточного замовлення та ппідтвердження замовлення (Submit Order button).

Після реєстрації в веб-додатку потрібно додати нові товари і для цього користувач переходить до розділу Catalog, після цього натискаю кнопку Add New Product to Catalog (рис. 3.8).

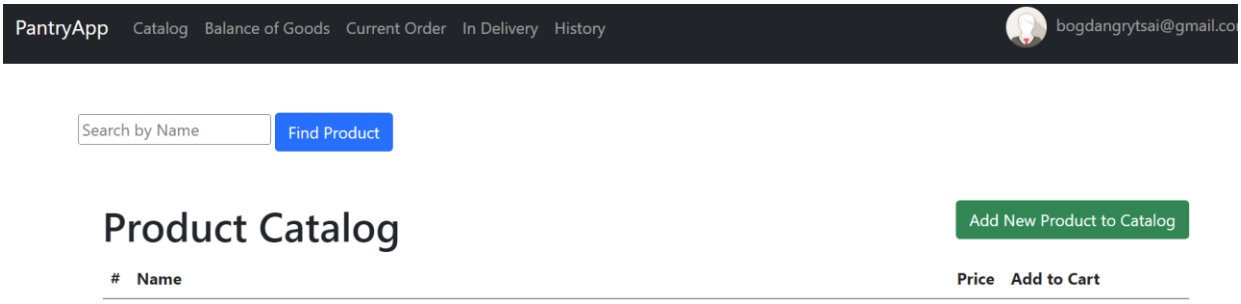

Рисунок 3.8 – Каталог товарів з кнопкою додавання

Після натискання на кнопку в коді відбувається виклик методу контролера з методом запиту GET (рис 3.9).

```
@GetMapping("/catalog/add-product")
public String addNewProduct(Model model) {
    List<Category> allCategories = (List<Category>) categoryService.findAll();
    List<Store> allStores = (List<Store>) storeService.findAll();
    model.addAttribute( attributeName: "categories", allCategories);
    model.addAttribute( attributeName: "stores", allStores);
    return "catalog/add_product_to_catalog";
}
```

Рисунок 3.9 –Метод контролера, що повертає сторінку з формою

У даному методі відбувається формування моделі для передачі на форму, а саме додавання категорій товарів та списку інтернет-магазинів, де його можна придбати, для форми створення нового товару. Після цього метод повертає шлях до файлу з формою для створення продукту (рис. 3.10).

```
<form method="post">
    <input type="text" name="name" placeholder="Name: " class="form-control" required maxlength="255"><br>
    <select name="category" class="form-control" required>
        <option disabled selected>Choose Category</option>
        <#list categories as category>
        <option value="${category.id}">${category.name}</option>
        </#list>>
    </select><br>

    <input type="text" name="description" placeholder="Description: " class="form-control"><br>
    <input type="number" step="0.01" name="itemPrice" placeholder="Item Price: " class="form-control" required><br>
    <select name="store" class="form-control" required><br>
        <!-- get from databases -->
        <option disabled selected>Choose Store</option>
        <#list stores as store>
            <option value="${store.id}">${store.name}</option>
        </#list>
    </select><br>
    <input type="text" name="link" placeholder="Link to Product in Shop: " class="form-control" required maxlength="255"><br>
    <input type="hidden" name="_csrf" value="${_csrf.token}" />
    <button type="submit" class="btn btn-success">Create</button>
</form>
```

Рисунок 3.10 – Форма додавання в файлі Freemaker

У цьому файлі отримуються дані моделі, а саме категорії та інтернет-магазини і переліковуються всередині випадаючих форм всередині блоку `<#list>`. В той же, час користувач бачить форму зображену на рисунку 3.11.

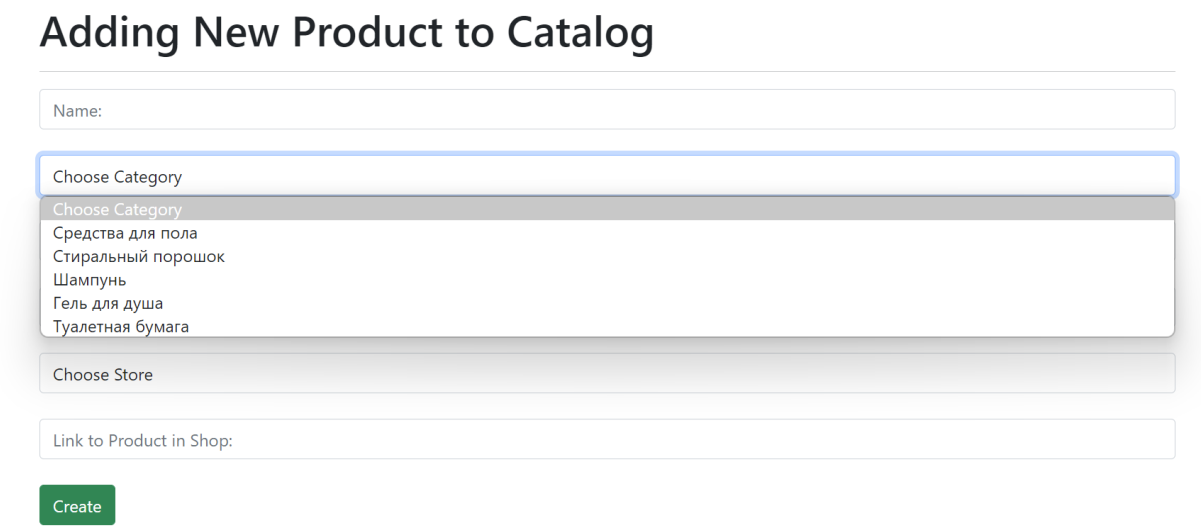

Рисунок 3.11 – Форма додавання нового товару

Після заповнення необхідних полів та натискання кнопки Create програма переходить до обробки запиту методом POST, всередині якого отримує дані отримані від користувача, на основі яких сторює новий об’єкт типу Product та зберігає відомості про нього до бази даних (рис. 3.12).

```java
@PostMapping("/catalog/add-product")
public String addNewProduct(@RequestParam String name,
                            @RequestParam Long category,
                            @RequestParam String description,
                            @RequestParam float itemPrice,
                            @RequestParam Long store,
                            @RequestParam String link,
                            Model model) throws MalformedURLException {
    Category categoryObj = categoryService.findById(category);
    Store storeObj = storeService.findById(store);
    Product newProduct = new Product(name, description, categoryObj, Double.valueOf(itemPrice), storeObj, new URL(link));
    productService.saveProduct(newProduct);
    return "redirect:/catalog";
}
```

Рисунок 3.12 – Обробка та зберігання введених користувачем даних

Система повертає користувача до каталогу, де знаходиться створений товар (рис 3.13).

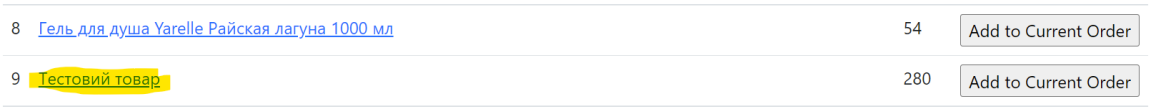

Рисунок 3.13 – Каталог з новоствореним товаром

Після цього користувач може натиснути один або декілька разів (в залежності від того, скільки одиниць товару потрібно додати до поточного замовлення) кнопку Add to Current Order. В свою чергу система виконає POST метод контролера, у якому знайде продукт, поточне замовлення, та всередині сервісу `OrderService` створить одиницю замовлення, яка міститиме даний продукт (рис. 3.14).

```java
@PostMapping("/catalog-add")
public String addProductToOrder(@RequestParam Long id,
                                @AuthenticationPrincipal User user,
                                Model model) {
    Product product = productService.findById(id);
    List<Product> products = new ArrayList<>();
    products.add(product);
    Order currentOrder = orderService.findActiveOrderOrCreateDefault(OrderStatus.ACTIVE, user.getId()).stream().findFirst().get();

    orderService.addProductToOrder(currentOrder, products, user, amount: 1, product.getItemPrice());
    return "redirect:/catalog";
}
```

Рисунок 3.14 – Код методу додавання товару до замовлення

Тепер товар додано до замовлення і користувач може натиснути кнопку Submit Order (рис. 3.15).

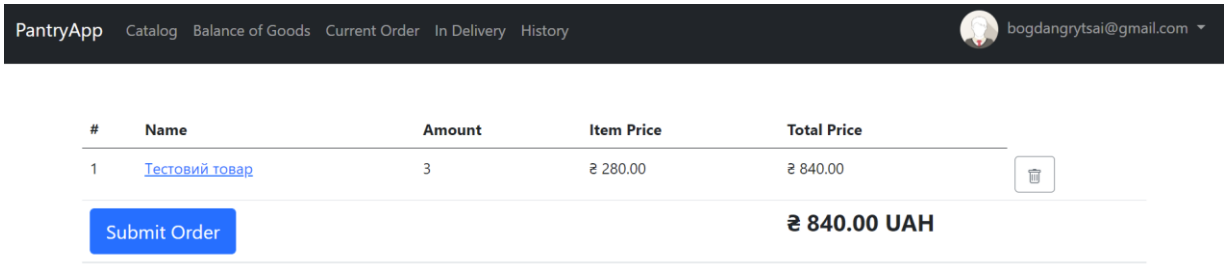

Рисунок 3.15 – Поточне замовлення з новоствореним товаром

Після підтвердження замовлення система переходить до відповідного методу контролера OrderController, в якому перевіряє наявність активного замовлення, викликає метод сервісу, в якому формує замовлення (змінює статус і т. д), відправляє листа з підтвердженням на пошту користувача та створює нове пусте замовлення для користувача (рис. 3.16 – 3.19).

Для відправлення листів створено окремий обліковий запис Google і надано доступ веб-додатку до нього.

```
@PostMapping("/submitorder")
public String submitOrder(@AuthenticationPrincipal User user, Model model) {
    Order activeOrder = orderService.findActiveOrderOrCreateDefault(OrderStatus.ACTIVE, user.getId()).stream().findFirst().get();
    if (activeOrder != null) {
        orderService.submitOrder(activeOrder, user);
    }

    return "redirect:/indelivery";
}
```

Рисунок 3.16 – Метод контролеру підтвердження замовлення

```
public void submitOrder(Order activeOrder, User currentUser) {
        activeOrder.setOrderStatus(OrderStatus.IN_DELIVERY);
        activeOrder.setDateOfSubmit(LocalDateTime.now());

        generateAndSendNoteToUser(activeOrder, currentUser);

        orderRepository.save(activeOrder);

        Map<String, Object> model = new HashMap();
        model.put("name", currentUser.getFirstName());
        model.put("title", "Order " + activeOrder.getOrderName() + " has submitted!");
        model.put("order", activeOrder);

        emailService.sendEmail(currentUser.getEmail(), subject: "[PantryApp] Order " + activeOrder.getOrderName() + " has submitted!", SUBMITTED

        createDefaultOrderForUser(currentUser);
}
```

Рисунок 3.17 – Метод сервісу підтвердження замовлення.

```java
public void sendEmail(String toAddress, String subject, String templateName, Map<String, Object> model) {

    try {
        MimeMessage mimeMessage = emailSender.createMimeMessage();
        MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, encoding: "utf-8");
        helper.setTo(toAddress);
        helper.setSubject(subject);
        helper.setFrom(FROM_EMAIL);
        helper.setText(geContentFromTemplate(model, templatePath: EMAIL_TEMPLATE_FOLDER_PATH + templateName), html: true);
        emailSender.send(mimeMessage);
    } catch (MessagingException e) {
        log.error(FAILED_TO_SEND_EMAIL_MSG, e);
        throw new IllegalStateException(FAILED_TO_SEND_EMAIL_MSG);
    }
}

public String geContentFromTemplate(Map<String, Object > model, String templatePath) {
    StringBuffer content = new StringBuffer();

    try {
        content.append(FreeMarkerTemplateUtils.processTemplateIntoString(freeMakerConfiguration.getTemplate(templatePath), model));
    } catch (Exception e) {
        e.printStackTrace();
    }
    return content.toString();
}
```

Рисунок 3.18 – Код відправки листа на пошту користувача

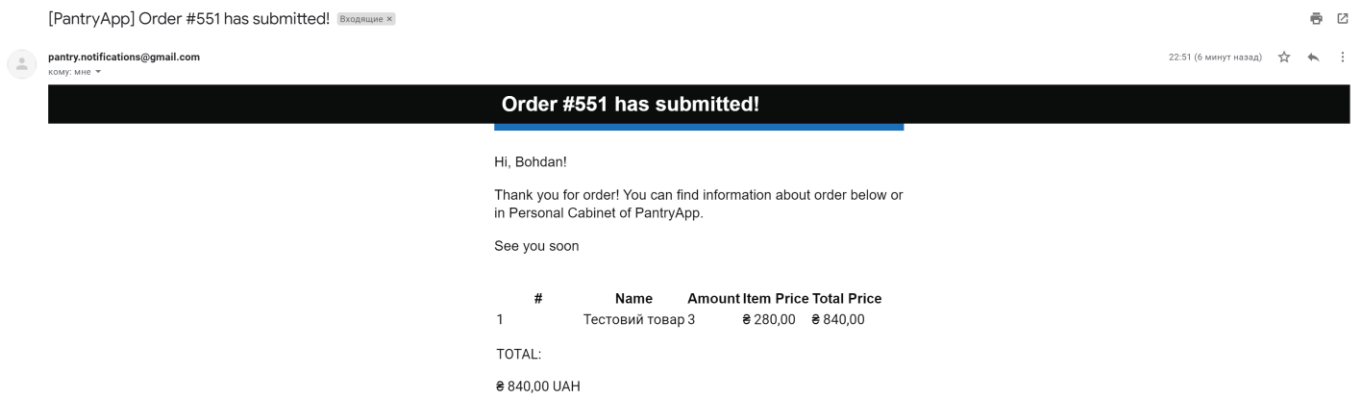

Рисунок 3.19 – Зміст листа отриманого користувачем

Після цих дій користувач може знайти своє замовлення в розділі In Delivery та натиснути кнопку Delivered для переміщення замовлення до розділу History, а товарів до розділу Balance of Goods.

3.4 Безпека та сек’юрність інформаційної системи

Кожна сучасна інформаційна система має володіти засобами для забезпечення захисту від несанкціонованого доступу, авторизації та аутентифікації. Для цього в інформаційній системі використовується фреймворк Spring Security. Для кращого розуміння варто переглянути конфігураційний файл, який зображено на рисунку 3.20.

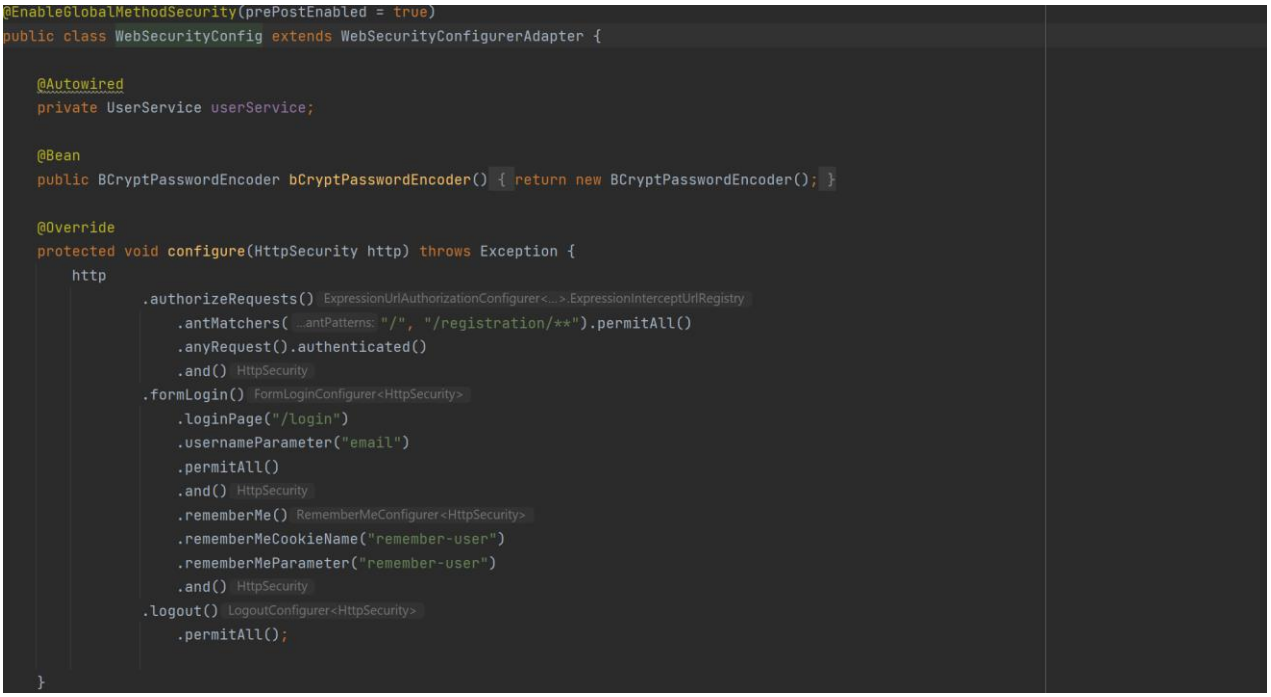

Рисунок 3.20 – Конфігураційний клас Spring Security

Даний файл встановлює певні правила для запитів, які надходять до бекенд частини веб-додатку. Деякі із комбінацій методів налаштування варто розглянути окремо:

– `.antMatchers("/", "/registration/**").permitAll()` - надає доступ до головної сторінки та сторінки реєстрації нового користувача абсолютно всім відвідувачам в тому числі неавторизованим;

– `.formLogin()` – здійснює налаштування форми авторизації;

– `.loginPage("/login").usernameParameter("email").permitAll()` – вказує мапінг для сторінки авторизації, параметр, який буде сприйматися системою як ім’я користувача, а також дозволяє всім відвідувачам потрапляти на сторінку;

– `.rememberMe().rememberMeCookieName("remember-user")` `.rememberMeParameter("remember-user")` – налаштування механізму запам’ятовування користувача при авторизації (вказується ім’я параметру Cookie, який буде збережено браузером);

– `.logout()` `.permitAll()` – дозволяє всім користувачам мати змогу розлогінитись (вийти з системи).

Також на рисунку 3.21 налаштовується доступ до папок з ресурсами: стилі, скрипти, шрифти та щображення є доступними для всіх користувачів.

```
@Override
public void configure(WebSecurity web) {
    web.ignoring().antMatchers(
            //static
            ...antPatterns: "/styles/**",
            "/js/**",
            "/fonts/**",
            "/images/**"
    );
}
```

Рисунок 3.21 – Конфігурація доступу до ресурсів

ВИСНОВКИ

У наш час повсякденне життя поступово оцифровується. Всі організації та підприємства переходять до безпаперових способів обробки інформації. Комерційна діяльність не є винятком, що підкреслює актуальність обраної теми роботи – створення веб-орієнтованої інформаційної системи «Pantry Helper».

В рамках виконаної роботи було:

- досліджено предметну галузь;
- проведено аналіз аналогу, виявлено недоліки та переваги, а також визначено актуальність даної роботи;
- проведено підготовку середовища для планування роботи та контролем за завданнями;
- здійснено постановку завдання для реалізації інформаційної системи;
- визначено середовище та засоби для розробки веб-орієнтованого додатку;
- створено прототип інформаційної системи;
- реалізовано основний функціонал: створення облікових записів з можливістю верифікації по електронній пошті, додавання продуктів до каталогу, створення замовлень, відправка повідомлень на пошту про підтвердження замовлення, механізм зміни статусу замовлень, механізм контролю за наявністю товарів у «коморі» користувача, можливість змінювати налаштування користувача;
- реалізовано механізми контролю доступу.

Отже, наразі веб-додаток працює в тестовому (демо) режимі з підтримкою всіх основних функціональних можливостей. Користувачі можуть створити нові товари на основі існуючих в сучасних інтернет-магазинах та сформувати своє замовлення. Також веб-додаток буде підтримуватися надалі, покращуватися і збільшуватиметься кількість наявного в ньому функціоналу

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ ТА ДЖЕРЕЛ

1. Веб-ресурс Cognitivelot – Інформаційні системи. URL: http://cognitivelot.ru/about/database/raznoe/informacionnye-sistemy//
2. Веб-ресурс Sekretariat. –Чим веб-орієнтовані системи відрізняються від інших. URL: https://www.sekretariat.ru/question/211658-qqqa-16-m4-chem-veb-orientirovannye-sistemy-otlichayutsya-ot-drugih-sed
3. Сайт фреймворку Spring URL: https://spring.io/
4. Офіційна документація фреймворку Bootstrap. URL: https://getbootstrap.com/docs/4.5/getting-started/introduction/
5. Stack Overflow. URL: https://ru.stackoverflow.com/
6. Офіційна документація фреймворку Spring. URL: https://spring.io/
7. Учебник по Bootstrap. URL: https://itchief.ru/bootstrap/
8. Дакетт, Джон Основи веб-програмування з використанням HTML, XHTML і CSS / Джон Дакетт. - М .: Ексмо, 2019. – 77 с.
9. Craig Walls, Spring in Action 5, Fifth edition : Manning Publications Co, 2019 – 498 с.
10. Дакетт, Д. HTML і CSS. Розробка і дизайн веб-сайтів / Д. Дакетт. - М .: Ексмо, 2018. - 208 с.
11. Гамма , Р. Хелм , Р. Джонсон , Д. Вліссідес, Патерни об’єктно орієнтованого проектування, : ООО «Издательство Питер», 2020. – 448 с.

ДОДАТКИ

Додаток А

Модель «User»:

```
@Getter
@Setter
@EqualsAndHashCode
@NoArgsConstructor
@Entity
@Table(name = "users")
public class User implements UserDetails {
    @SequenceGenerator(name = "user_sequence", sequenceName = "user_sequence", allocationSize = 1)
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "user_sequence")
    private Long id;
    private String password;
    @Enumerated(EnumType.STRING)
    private Role role;
    private Boolean locked = false;
    private Boolean enabled = false;
    // User custom settings
    private String firstName;
    private String lastName;
    private String middleName;
    private String address = "No address";
    private String email;
    private String defaultSite = "No site";
    private String phone;
    private Double defaultMaxPrice = 500D;
    @Transient
    private String passwordConfirm;
    @OneToMany(targetEntity = Pantry.class, mappedBy = "user")
    private Set<Pantry> pantries = new HashSet<>();
```

```
    public User(
            String firstName,
            String lastName,
            String middleName,
            String password,
            Role role,
            String address,
            String email,
            String defaultSite,
            String phone,
            String passwordConfirm,
            Set<Pantry> pantries
    ) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.middleName = middleName;
        this.password = password;
        this.role = role;
        this.address = address;
        this.email = email;
        this.defaultSite = defaultSite;
        this.phone = phone;
        this.passwordConfirm = passwordConfirm;
        this.pantries = pantries;
    }
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities()
{
        SimpleGrantedAuthority       authority       =       new
SimpleGrantedAuthority(role.name());
        return Collections.singletonList(authority);
    }
    @Override
    public boolean isAccountNonExpired() {
```

```
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return !locked;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return enabled;
    }


    public boolean isAdmin() {
        return "ADMIN".equals(role.name());
    }

    public String getFullName() {
        String fullName = getFirstName() + getLastName();
        if (middleName != null) {
            fullName+= ", " + middleName;
        }
        return fullName;
    }
}
```

Модель «Order»:

```
@Entity(name = "Order")
@Table(name = "orders")
@Data
public class Order {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String orderName;
    private String orderDescription;
    private Long userId; //(fk)
    private String shopLink;

    @OneToMany(mappedBy = "order", cascade = CascadeType.ALL)
    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    private Set<OrderItem> orderDetails  = new HashSet<>();
    private Double totalOrderPrice;
    private LocalDateTime dateOfSubmit;
    private LocalDateTime deliveryDate;
    private Double maxPrice;

    @Enumerated(EnumType.STRING)
    private OrderStatus orderStatus;

    public Order() {
    }

    public Order(String orderName, String orderDescription, Long
userId, OrderStatus orderStatus, Double maxPrice) {
        this.orderName = orderName;
        this.orderDescription = orderDescription;
        this.userId = userId;
```

```
        this.orderStatus = orderStatus;
        this.maxPrice = maxPrice;
    }

    public Order(Long userId, String shopLink, OrderStatus
orderStatus) {
        this.userId = userId;
        this.shopLink = shopLink;
       this.orderStatus = orderStatus;
   }

   public String getDateOfSubmitInSimpleFormat() {
       DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
        return dateOfSubmit.format(formatter);
   }

    public String getDeliveryDateInSimpleFormat() {
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
        return deliveryDate.format(formatter);
    }

}
```

Модель «ConfirmationToken»:

```
@Getter
@Setter
@NoArgsConstructor
@Entity
public class ConfirmationToken {
    @SequenceGenerator(name = "confirmation_token_sequence",
sequenceName = "confirmation_token_sequence", allocationSize = 1)
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator
= "confirmation_token_sequence")
```

```
    private Long id;

    @Column(nullable = false)
    private String token;

    @Column(nullable = false)
    private LocalDateTime createdAt;
    @Column(nullable = false)
    private LocalDateTime expiresAt;
    private LocalDateTime confirmedAt;

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

    public ConfirmationToken(String token, LocalDateTime
createdAt, LocalDateTime expiredAt, User user) {
        this.token = token;
        this.createdAt = createdAt;
        this.expiresAt = expiredAt;
        this.user = user;
    }
}
```

Модель «Product»:

```
@Data
@Entity
@Table(name = "products")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @EqualsAndHashCode.Exclude
    private Long id;
    private String name;
```

```
    @Column(columnDefinition="TEXT")
    private String description;

    @ManyToOne
    @JoinColumn(name = "category_id")
    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    private Category category;

    private Double itemPrice;

    @ManyToOne
    @JoinColumn(name = "store_id")
    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    private Store store;

    private URL linkToProduct;


    public Product (String name, String description, Category
category, Double itemPrice, Store store, URL linkToProduct) {
        this.id = id;
        this.name = name;
        this.description = description;
        this.category = category;
        this.itemPrice = itemPrice;
        this.store = store;
        this.linkToProduct = linkToProduct;
    }
    public Product() {}
}
```

Додаток Б

Репозиторій «UserRepository»:

```
@Repository
public interface UserRepository extends JpaRepository<User, Long>
{
    Optional<User> findByEmail(String email);

    @Transactional
    @Modifying
    @Query("UPDATE User a " +
            "SET a.enabled = TRUE WHERE a.email = ?1")
    int enableAppUser(String email);
}
```

Репозиторій «OrderRepository»:

```
public interface OrderRepository extends JpaRepository<Order, Long>
{
    Order findOrderById(Long orderId);
    List<Order>        findOrderByOrderStatusAndUserId(OrderStatus
orderStatus,
long userId);
}
```

Репозиторій «ConfirmationTokenRepository»:

```
@Repository
public     interface     ConfirmationTokenRepository     extends
JpaRepository<ConfirmationToken, Long> {
    Optional<ConfirmationToken> findByToken(String token);
    @Transactional
    @Modifying
    @Query("UPDATE ConfirmationToken c " +
            "SET c.confirmedAt = ?2 " +
            "WHERE c.token = ?1")
    int updateConfirmedAt(String token,
                          LocalDateTime confirmedAt);
}
```

Додаток В

Сервіс «UserService»:

```
@Service
public class UserService implements UserDetailsService {
    private static final String USER_NOT_FOUND_MSG = "User %s is not found!";
    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private ConfirmationTokenService confirmationTokenService;
    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        return userRepository.findByEmail(email).orElseThrow(() -> new UsernameNotFoundException(String.format(USER_NOT_FOUND_MSG, email)));
    }

    public String signUpUser(User user) {
        boolean userExists = userRepository
                .findByEmail(user.getEmail())
                .isPresent();

        if (userExists) {
            throw new IllegalStateException("Email already exists");
        }
        String encodedPassword = bCryptPasswordEncoder.encode(user.getPassword());
        user.setPassword(encodedPassword);

        userRepository.save(user);
```

```
        String token = UUID.randomUUID().toString();

        ConfirmationToken confirmationToken = new
ConfirmationToken(
                token,
                LocalDateTime.now(),
                LocalDateTime.now().plusMinutes(15),
                user
        );
confirmationTokenService.saveConfimationToken(confirmationToken);
        return token;
    }
    public List<User> findAll() {
        return userRepository.findAll();
    }

    public User findUserById(Long userId) {
        Optional<User> userFromDb =
userRepository.findById(userId);
        return userFromDb.orElse(new User());
    }

    public void saveUser(User user) {
     userRepository.save(user);
    }

    public boolean deleteUser(Long userId) {
        if (userRepository.findById(userId).isPresent()) {
            userRepository.deleteById(userId);
            return true;
        }
        return false;
    }
```

```
    public void updateProfile(User user, String password) {
        if(!StringUtils.isEmpty(password)) {
            user.setPassword(password);
            userRepository.save(user);
        }
    }

    public int enableAppUser(String email) {
        return userRepository.enableAppUser(email);
    }
}
```

Сервіс «DeliveryService»:

```
@Service
public class DeliveryService {
    @Autowired
    private OrderService orderService;
    @Autowired
    private PantryItemRepository pantryItemRepository;
    public void deliveredProductsSaveInPantry(@AuthenticationPrincipal User user, Pantry pCurrentPantry, Order orderInDelivery) {
        Set<PantryItems> existingPantryItems = pCurrentPantry.getProductDetails();
        for (OrderItem orderItem : orderInDelivery.getOrderDetails()) {
            boolean isPantryItemAmountUpdated = false;

            for (PantryItems existingPantryItem : existingPantryItems) {
                if (orderItem.getProduct().getId().equals(existingPantryItem.getProduct().getId())) {
                    Integer currentItemAmount = existingPantryItem.getItemsCount();
```

```
existingPantryItem.setItemsCount(currentItemAmount +
orderItem.getOrderAmount());

existingPantryItem.setLastOrderDate(LocalDate.now());

pantryItemRepository.save(existingPantryItem);
                    isPantryItemAmountUpdated = true;
                }
            }

            if (!isPantryItemAmountUpdated) {
                Product newPantryProduct = orderItem.getProduct();
                PantryItems pantryItem = new PantryItems(1,
pCurrentPantry, newPantryProduct, LocalDate.now());

pantryItem.setItemsCount(orderItem.getOrderAmount());
                pantryItemRepository.save(pantryItem);}}

    public void moveOrderInDeliveryToHistory(Order
orderInDelivery) {

orderInDelivery.setOrderDescription(orderInDelivery.getOrderName(
) + " delivered " + LocalDateTime.now());
        orderInDelivery.setDeliveryDate(LocalDateTime.now());
        moveOrderToHistory();
        orderService.changeStatusAndSave(orderInDelivery,
OrderStatus.COMPLETED);
    }

    public void moveOrderToHistory() {

    }
}
```

Сервіс «StoreService»:

```
@Service
public class StoreService {
    @Autowired
    private StoreRepository storeRepository;

    public Iterable<Store> findAll() {
        return storeRepository.findAll();
    }

    public Store findById(Long id) {
        return storeRepository.findById(id).get();
    }
}
```

Сервіс «CategoryService»:

```
@Service
public class CategoryService {

    @Autowired
    private CategoryRepository categoryRepository;

    public Iterable<Category> findAll() {
        return categoryRepository.findAll();
    }

    public Category findById(Long id) {
        return  categoryRepository.getById(id);
    }
}
```

Додаток Г

Контролер «UserController»:

```
@Controller
@RequestMapping("/user")
public class UserController {
    @Autowired
    private UserService userService;
    @Autowired
    private PantryService pantryService;
    @GetMapping
    @PreAuthorize("hasAuthority('ADMIN')")
    public String userList(Model model) {
        model.addAttribute("users" , userService.findAll());
        return "userList";
    }
    @PostMapping
    @PreAuthorize("hasAuthority('ADMIN')")
    public String saveUser(@RequestParam String role,
                           @RequestParam("userid") User user,
                           @RequestParam   Map<String,   String>
form) {
        return "redirect:/user";}

    @GetMapping("{user}")
    @PreAuthorize("hasAuthority('ADMIN')")
    public String userEdit(@PathVariable User user,
                           Model model ) {
        model.addAttribute("user", user);
        model.addAttribute("roles",
Collections.singleton(user.getRole()));
        return "userEdit";
    }

    @GetMapping("/profile")
```

```
    public String getProfile(Model model, @AuthenticationPrincipal
User user) {
        model.addAttribute("username", user.getUsername());
        return "profile";
    }

    @PostMapping("/profile")
    public String updateProfile(@AuthenticationPrincipal User
user,
                                String password) {
        userService.updateProfile(user, password);
        return "redirect:/user/profile";
    }
}
```

Контролер «ProductCatalogController»:

```
@Controller
@Slf4j
public class ProductCatalogController {
    @Autowired
    private CategoryService categoryService;
    @Autowired
    private StoreService storeService;
    @Autowired
    private ProductService productService;
    @Autowired
    private OrderService orderService;
    @GetMapping("/catalog")
    public String currentOrderPage(Model model) {
            List<Product>    allProducts    =    (List<Product>)
productService.findAll();
            model.addAttribute("products", allProducts);
        return "catalog/product_catalog";}
    @PostMapping("/catalog-add")
    public String addProductToOrder(@RequestParam Long id,
```

```
                                    @AuthenticationPrincipal User
user, Model model) {
        Product product = productService.findById(id);
        List<Product> products = new ArrayList<>();
        products.add(product);
        Order                   currentOrder                   =
orderService.findActiveOrderOrCreateDefault(OrderStatus.ACTIVE,
user.getId()).stream().findFirst().get();
        orderService.addProductToOrder(currentOrder,    products,
user, 1, product.getItemPrice());
        return "redirect:/catalog"; }
    @GetMapping("/catalog/add-product")
    public String addNewProduct(Model model) {
        List<Category>    allCategories    =    (List<Category>)
categoryService.findAll();
        List<Store>       allStores       =       (List<Store>)
storeService.findAll();
        model.addAttribute("categories", allCategories);
        model.addAttribute("stores", allStores);
        return "catalog/add_product_to_catalog"; }
    @PostMapping("/catalog/add-product")
    public String addNewProduct(@RequestParam String name,
                                @RequestParam Long category,
                                @RequestParam String description,
                                @RequestParam float itemPrice,
                                @RequestParam Long store,
                                @RequestParam String link,
                                Model        model)        throws
MalformedURLException {
        Category categoryObj = categoryService.findById(category);
        Store storeObj = storeService.findById(store);
        Product  newProduct  =  new  Product(name,  description,
categoryObj, Double.valueOf(itemPrice), storeObj, new URL(link));
        productService.saveProduct(newProduct);
        return "redirect:/catalog";}}
```

Додаток Д

Сервіс «EmailService»:

```
@Service
public class EmailService implements EmailSender {
    private static final String FAILED_TO_SEND_EMAIL_MSG = "Failed to send email!";
    private static final String FROM_EMAIL = "pantry.notifications@gmail.com";
    private static final String EMAIL_TEMPLATE_FOLDER_PATH = "email_templates/";
    private final static Logger log = LoggerFactory.getLogger(EmailService.class);
    @Autowired
    private JavaMailSender emailSender;
    @Autowired
    Configuration freeMakerConfiguration;
    @Override
    @Async
    public void sendEmail(String toAddress, String subject, String templateName, Map<String, Object> model)  {
        try {
            MimeMessage mimeMessage = emailSender.createMimeMessage();
            MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, "utf-8");
            helper.setTo(toAddress);
            helper.setSubject(subject);
            helper.setFrom(FROM_EMAIL);
            helper.setText(geContentFromTemplate(model, EMAIL_TEMPLATE_FOLDER_PATH + templateName), true);
            emailSender.send(mimeMessage);
        } catch (MessagingException e) {
            log.error(FAILED_TO_SEND_EMAIL_MSG, e);
```

```
            throw new
IllegalStateException(FAILED_TO_SEND_EMAIL_MSG);
        }
    }
    public String geContentFromTemplate(Map<String, Object > model,
String templatePath) {
        StringBuffer content = new StringBuffer();
        try {
content.append(FreeMarkerTemplateUtils.processTemplateIntoString(
freeMakerConfiguration.getTemplate(templatePath), model));
        } catch (Exception e) {
            e.printStackTrace();
        }
        return content.toString();
    }
}
```

Інтерфейс «EmailSender»:

```
package com.uktech.kladovka.service.mail;

import java.util.Map;

public interface EmailSender {
    void sendEmail(String toAddress, String subject, String
templateName, Map<String, Object> model);
}
```