МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ОНЛАЙН ЗАМОВЛЕННЯ ПРОДУКЦІЇ»**

Виконав: студент 2 курсу, групи 8.1210-з

спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

Синицький О.О.
(ініціали та прізвище)

Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент
_____ Лісняк А.О.
(підпис)

« __ » _____ 2021 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Синицькому Олексію Олександровичу
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Розробка інформаційної системи для онлайн замовлення продукції

керівник роботи (проекту) к.ф.-м.н., завідувач кафедри Лісняк А.О.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 09 » червня 2021 року № 850-с

2. Строк подання студентом роботи 25.11.2021

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка інформаційної системи для онлайн замовлення продукції.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.		
2.	Збір вихідних даних.		
3.	Обробка методичних та теоретичних		
	джерел.		
4.	Розробка першого розділу.		
5.	Розробка другого розділу.		
6.	Оформлення та нормоконтроль		
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.		

Студент _____ _____
(підпис) (ініціали та прізвище)

Керівник роботи _____ _____
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ С.П. Швидка
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка інформаційної системи для онлайн замовлення продукції»: 51 с., 21 рис., 3 табл., 16 джерел, 3 додатки.

WEB-ДОДАТОК, РОЗРОБКА, АРХІТЕКТУРА, ОНЛАЙН ЗАМОВЛЕННЯ, ВИРОБНИЦТВО, ІНФОРМАЦІЙНА СИСТЕМА, ГНУЧКІСТЬ, МОБІЛЬНІСТЬ, МАСШТАБОВАНІСТЬ.

Об’єкт дослідження – розробка інформаційної системи для онлайн замовлення продукції.

Мета роботи: виконати аналіз предметної області, розробити та реалізувати інформаційну систему для онлайн замовлення продукції.

Гіпотеза дослідження: передбачаємо, що використана архітектура для створення Web-додатку дозволить інформаційній системі вирішити функціональні завдання та набути якості масштабованості, гнучкості та мобільності.

Методи дослідження: аналіз, класифікація, спостереження, тестування, прототипування, моделювання.

Наукова новизна полягає у розробці та впровадженні інформаційної системи для онлайн замовлення продукції яка буде простою у використанні для користувачів, гнучкою – здатною до використання у різних сферах виробництва продукції, мобільною – здатною взаємодіяти із різними середовищами операційних систем, масштабованою – здатною до збільшення користувачів.

Галузь використання: виробництво, розробка Web-додатків.

SUMMERY

Master's qualifying paper «Online product ordering system developing»: 51 pages, 21 figures, 3 tables, 16 references, 3 supplements.

WEB-APPLICATION, DEVELOPMENT, ARCHITECTURE, ONLINE ORDERING, PRODUCTION, INFORMATION SYSTEM, FLEXIBILITY, MOBILITY, SCALABILITY.

Object of the study is the development and launching of online product ordering system for production company.

Aim of the qualifying work is to substantiate theoretically, to develop and to deploy the online product ordering system for production company.

Methods of research – analytical, comparative analysis of software, prototype design.

To realize aim of the qualifying work we will do such tasks:

- to explore theoretically the main approaches to make specifications of development of the online product ordering system;
- to reveal the main stages of designing of the online product ordering system;
- to develop and deploy the online product ordering system based on Web-application for production company.

The qualifying work gives theoretical survey and developing of the online product ordering system by modern toolkits and based on the main conditions:

- to be flexible – to be able to satisfy the main needs of production company;
- to be mobile – to be able to be integrated in different operation systems environments;
- to have possibility for scalability – to be able to increase in quantity of users.

ЗМІСТ

ВСТУП

Актуальність дослідження. Сучасні тенденції швидкого розвитку інформаційних технологій дають людині можливість швидше приймати рішення та бути активним одночасно у різних сферах життя. Проте збільшення накопичення інформації у сервісах та додатках, велика вартість сервісів та їх підписок, і ускладнення систем здебільшого відлякують користувачів, що призводить до того, що сайти, які надають послуги не приносять очікуваних замовлень та прибутку.

Тому, щоб вирішити дану проблему потрібно розробити та впровадити інформаційну систему для онлайн замовлення продукції, яка буде простою у використанні для користувачів; безкоштовною, тобто не мати платної підписки; гнучкою – здатною до використання у різних сферах виробництва продукції.

Об’єкт дослідження – розробка інформаційної системи онлайн замовлення продукції виробничої компанії.

Предмет дослідження – особливості бізнес процесу онлайн замовлень продукції в виробничій компанії.

Мета: розробити та ввести в експлуатацію інформаційну систему для онлайн замовлення виробничої продукції.

Завдання дослідження:

1. Теоретично дослідити основні підходи проектування інформаційних систем.
2. Спроектувати інформаційну систему для онлайн замовлень.
3. Виконати вимоги до системи: має бути мобільною – здатною взаємодіяти із різними середовищами операційних систем, масштабованою – здатною до збільшення користувачів за допомоги компонентної архітектури для створення Web-додатку.

4. Реалізувати та впровадити інформаційну систему для онлайн замовлень на основі Web-додатку для виробничої компанії.

Методи дослідження: аналіз, класифікація, спостереження, тестування, прототипування, моделювання.

Інструменти які використані у розробці інформаційної системи: фреймворки Django, Django REST на мові програмування Python, фреймворки Angular8 на мові програмування TypeScript, сервіси Amazon Web Services для хостингу, скрипти автоматичного розгорнення BitBucket Pipelines, програмне забезпечення для автоматизації розгортання та керування програмами в середовищах з підтримкою контейнеризації Docker та інші.

Структура кваліфікаційної роботи магістра. Складається зі вступу, трьох розділів, висновків до розділів, загальних висновків, списку використаних джерел з 16 найменувань та 3 додатків на 15 сторінках. Загальний обсяг основного тексту 51 сторінок. Робота містить 3 таблиці та 21 рисунок.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Особливості проектування інформаційних систем онлайн замовлення виробничої продукції

Для розробки інформаційної системи для онлайн замовлення продукції у виробничій сфері будемо використовувати евристичний метод програмної інженерії, а саме об’єктно-орієнтованого підхід [6, с.24-26].

Переваги об’єктно орієнтованого підходу до розробки систем:

– системи більш відкриті до внесення змін та базуються на більш стійких формах, через це системи можуть розвиватися більш поступово, не ризикуючи бути повністю переробленою навіть у випадку значних змін;

– системи будуються компактного та меншого розміру шляхом використання загальних механізмів класів, що виражається у економічній вартості систем;

– об’єктна декомпозиція зменшує ризик створення складних систем, так як дає еволюційний шлях розвитку системи від простих класів до складних;

– об’єктна модель більш природна, оскільки орієнтована на людське сприйняття світу, а не на комп’ютерну реалізацію;

– об’єктна модель дає можливість використовувати усі можливості об’єктно-орієнтованих мов програмування.

Процес розробки програмного забезпечення пов’язаний із сутністю життєвого циклу програмного забезпечення, що являється одним із базових понять програмної інженерії.

Життєвий цикл програмного забезпечення – це період часу, який починається з моменту прийняття рішення про необхідність створювати програмне забезпечення і закінчується у момент повного його вилучення із експлуатації.

Модель життєвого циклу – це схема виконання робіт і задач у рамках процесів, що забезпечують розробку, експлуатацію і супровід програмного продукту. Ця схема відображає еволюцію програмного забезпечення, починаючи від формулювання вимог і закінчуючи припиненням користування нею [5, с.53-55].

Історично така схема робіт містить у собі:

– розробку вимог або технічного завдання;
– розробку ескізного або технічного проекту;
– програмування або робоче проектування;
– пробну експлуатацію;
– супровід і поліпшення;
– зняття з експлуатації.

Загалом, є три класичні та найбільш розповсюджені моделі життєвого циклу програмного забезпечення: каскадна, ітеративна та спіральна.

Кожна із моделей має свої переваги та недоліки та можуть модифікуватися залежно від ситуації того чи іншого проекту. Проте, кожна модель має наступні основні кроки, які показують спрощений процес розробки програмного забезпечення [6, с.36]:

– **аналізу** (процес осмислення предметної області);
– **проектування** (процес осмислення структури компонентів в області доступних рішень та специфікацій);
– **реалізації** (архітектурні компоненти проецируються на компілятори та програмні засоби).

Таким чином для ефективної розробки програмного забезпечення у більшості випадків використовуються об’єктно-орієнтовані методи розробки, враховуючи основні етапи життєвого циклу інформаційної системи: аналіз, проектування та реалізація. При цьому поєднуються переваги основних видів моделей життєвого циклу інформаційної системи: каскадна, ітеративна та спіральна.

1.2 Технічне завдання

1.2.1 Вимоги до інформаційної системи

Функціональне призначення системи – реалізувати та впровадити інформаційну систему для онлайн замовлень на основі Web-додатку для виробничої компанії.

Експлуатаційне призначення системи: система може експлуатуватися адміністратором і клієнтом системи на різних рівнях доступу.

В системі мають працювати 4 ролі: адміністратори, клієнти, оператори виробничого підприємства та менеджери.

Система має надавати користувачам клієнтам такі можливості:

– створення/редагування/перегляд статусу замовлення ;

– реєстрація та аутентифікація до системи.

Система має надавати адміністраторам такі можливості:

– створення/редагування/перегляд/видалення користувачів;

Система має надавати операторам такі можливості:

– Відхилення замовлення, редагування статусу замовлення;

Система має надавати менеджерам такі можливості:

– перегляд звітів про усі замовлення;

Система повинна відображати коректно інтерфейс користувача на будь-якому пристрої. А також відповідати області знань SWEBOK «Конструювання програмного забезпечення (Software Construction)» [3]. містить у собі такі розділи:

– зниження складності;

– попередження відхилень від стилю;

– структуризація перевірок;

– використання стандартів.

Зниження складності – це мінімізація, зменшення і локалізація складності конструювання [5, с.32]. Мінімізація складності – це обмеження на обробку

складних структур і великих обсягів інформації протягом тривалого періоду часу. Вона досягається, зокрема, використанням у процесі конструювання простих елементів, а також рекомендацій стандартів.

Система повинна працювати для наступних браузерів останніх версій: Mozilla Firefox, Google Chrome, Safari, Opera.

Вимоги до продуктивності:

– система повинна відображати будь-яку сторінку не довше, ніж за 3 секунди;

– система повинна обмінюватися повідомленням із сервером не більше ніж за 5 секунд.

Вимоги до безпеки:

– система не повинна дозволяти клієнтам, операторам та менеджерам фізичний доступ до інтерфейсу адміністратора.

– система не повинна надавати доступ неавторизованим користувачам доступ до даних системи.

1.2.2 Опис системи

Предметною областю є розробка системи створення онлайн замовлень. Дана система повинна надати користувачам можливість ведення нового замовлення продукції. Процес взаємодії з системою проходить наступним чином. Клієнт, повинен ввести адресу сайту в браузері та ввести дані у форму входу до системи. Якщо користувач вперше на сайті, він повинен спочатку пройти усі кроки реєстрації у системі.

Після входу до системи користувач клієнт має можливість взаємодіяти з такими розділами як: створити замовлення, переглянути статус замовлення.

Після входу до адміністратор має можливість взаємодіяти з такими розділами як адміністрування та надавати доступ такий як операторські права та менеджерські. Після цього користувач оператор може редагувати замовлення клієнтів: змінювати їх статус або відхиляти із вказанням причини.

А менеджер може переглядати звіти у інтерфейсі менеджера.

Процес внесення даних при створенні нового замовлення проходить наступним чином. Клієнт обирає розділ «Створити замовлення», система відображає інтерфейс взаємодії з даним розділом і відображає форму створення нового запису замовлення. Після введення даних клієнт натискає кнопку «Підтвердити замовлення», система зберігає введені дані, у розділі «Мої замовлення» з'являється новий запис про існуючи замовлення. Ці дані надалі будуть використовуватися для відображення статусу замовлення.

У системі будуть використовуватися такі статуси: «Новий», «У роботі»; «Виконаний»; «Відхилено».

Окрім основних функцій система надає адміністраторам можливість керувати обліковими записами користувачів та переглядати детальну статистику за обраними налаштуваннями.

Система має бути мобільною – здатною взаємодіяти із різними середовищами операційних систем, масштабованою – здатною до збільшення користувачів за допомоги компонентної архітектури для створення Web-додатку.

1.2.3 Вимоги до інтерфейсу користувача

У зв'язку з тим, що веб-додаток буде взаємодіяти з користувачем через браузер, то доцільно розробити правила дизайну інтерфейсу та зручність використання. Саме зручність використання – це якісна оцінка сайту на простоту та комфорт роботи з ним. Згідно праці Стіва Круга «Не примушуйте мене думати» [4], сайт повинен відповідати головному закону зручності використання (usability) - не примушувати думати користувача. Все важливе на сайті повинно бути доступно не більш ніж як за два кліки мишки. Мінімізація складної інформації на сайті, спрощення інтерфейсу та лаконічність у інтерфейсі.

При вході на сайт буде використаний наступний шаблон для розробки інтерфейсу (див. рис. 1.1).

Рисунок 1.1 – Шаблон першої сторінки інтерфейсу інформаційної системи

При роботі із створенням замовлень продукції буде використовуватися наступний шаблон для розробки інтерфейсу (див. рис. 1.2).

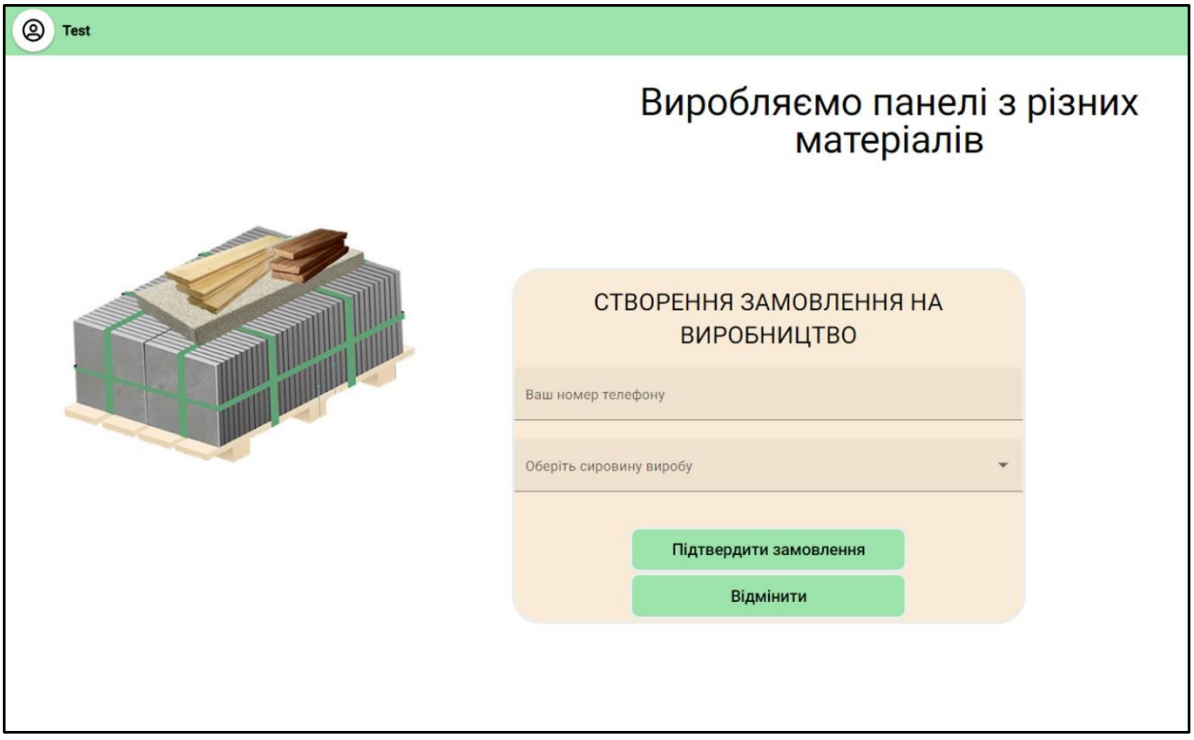

Рисунок 1.2 – Шаблон сторінки інтерфейсу роботи із замовленням

При роботі із створенням замовлень продукції буде використовуватися наступний шаблон для розробки інтерфейсу (див. рис. 1.3).

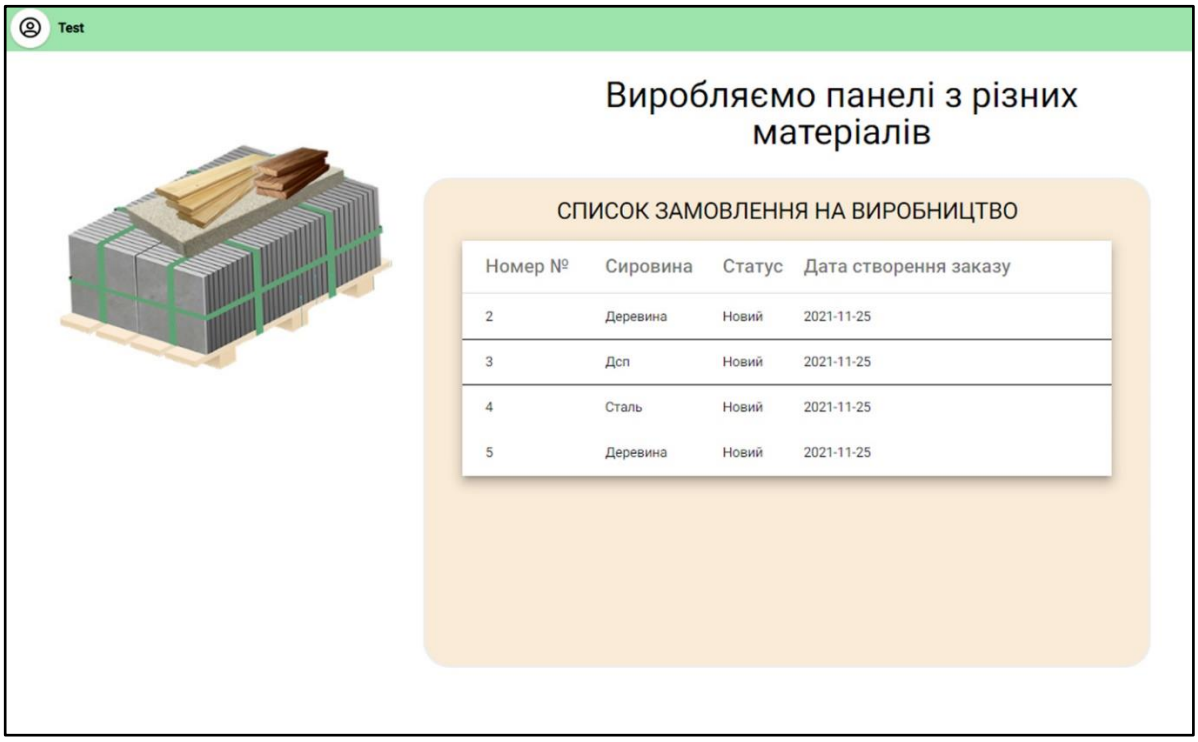

Рисунок 1.3 – Шаблон сторінки інтерфейсу список замовлень

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ТА ОГЛЯД ІНСТРУМЕНТІВ РОЗРОБКИ

2.1 Проектування інформаційної системи для онлайн замовлення виробничої продукції

Усі методи проектування інформаційних систем засновані на ідеї створення моделей систем, які можна відобразити графічно та використовувати данні моделі як специфікацію інформаційної системи та як її структуру.

Вважаємо, що більш ефективно використовувати модель складної системи для проектування програмного забезпечення для онлайн замовлення виробничої продукції. Важливим принципом проектування складних систем є принцип багатомодельності. Він формулюється наступним чином – не треба обмежуватися створенням тільки однієї моделі, треба використовувати совокупність декількох моделей, майже незалежних один від одного. На рисунку 2.1 представлена схема складної моделі [6, с.54-56].

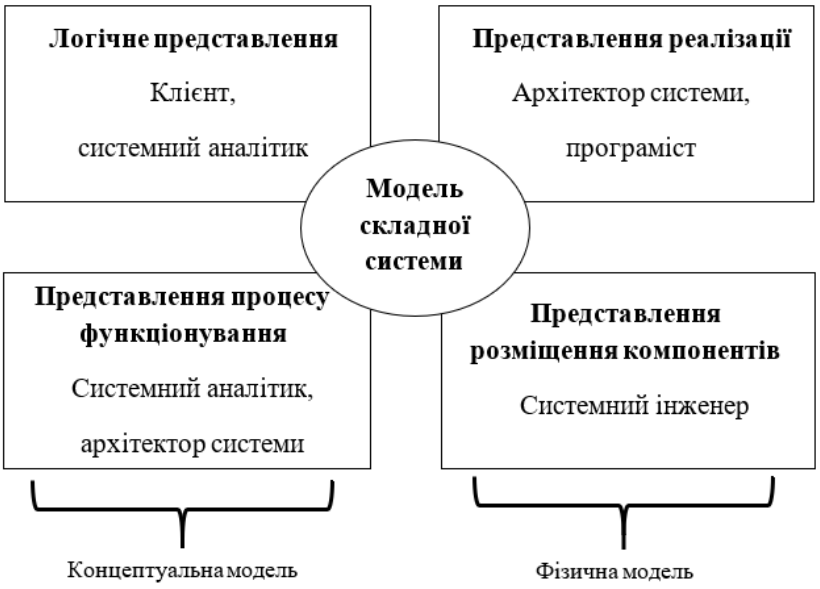

Рисунок 2.1 – Модель складної системи для проектування програмного забезпечення

Таким чином, при проектуванні інформаційної системи для виробничої продукції представимо цільову інформаційну систему у вигляді таких моделей:

- **логічну** (дану модель осмислюють такі ролі як клієнт та системний аналітик, що найпростішим шляхом відображає та прописує основні потреби та функції інформаційної системи);
- **функціонування** (дану модель осмислюють такі ролі як архітектор та системний аналітик, розглядаються основні функції системи, розробляються специфікації процесів);
- **компонентна** (дану модель осмислюють такі ролі як системний інженер, розглядається архітектура, безпека та економічна собівартість архітектури);
- **реалізаційна** (дану модель осмислюють такі ролі як програміст та архітектор системи, розглядається архітектура та інструменти, які компілюють специфікації спроектованої інформаційної системи у програмний код).

Інформаційної системи для виробничої продукції буде розроблятися у вигляді веб-додатку. Веб-додаток – це клієнт-серверна програма, в якій клієнт взаємодіє з веб-сервером за допомогою браузера або іншої програми клієнта [1]. Логіка веб-додатка розподілена між сервером та клієнтом, зберігання даних здійснюється переважно на сервері, обмін інформацією відбувається по мережі. На рисунку 2.2 представлена схема роботи веб-додатку.

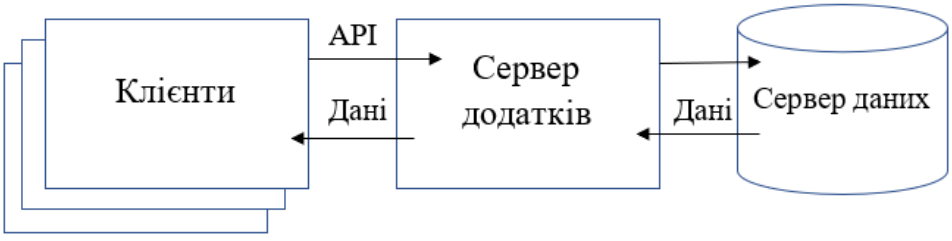

Рисунок 2.2 – Схема клієнт-серверної програми

Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-додатки є міжплатформними службами.

Відобразимо основні цілі та функціональні можливості інформаційної системі для онлайн замовлень графічно відповідно до моделі складної системи. Для створенная специфікацій та діаграм використаємо хмарний додаток https://lucid.app/.

Використаємо діаграму варіантів (Use case diagram) для відображення основних варіантів використання інформаційної системи для онлайн замовлень виробничої продукції.

Додаємо опис до варіантів використання (див. рис. 2.3):

1. Варіант «Створити замовлення» дає можливість замовнику ввести новий заказ у систему.

2. Варіант «Перевірити статус замовлення» дає можливість замовнику перевірити статус замовлення – «Новий», «У роботі», «Виконаний», «Відхилено».

3. Варіант «Відхилити замовлення» дає можливість оператору відхилити замовлення та вказати причину відхилення.

4. Варіант «Змінити статус замовлення» дає можливість оператору ввести новий статус замовлення і таким чином дати інформацію клієнту про стан його замовлення.

5. Варіант «Сформувати список замовлень» (виконаних, відкритих, у роботі та відхилених) дає можливість менеджеру перевірити звіти у системі.

6. Відношення варіанту «Відхилити замовлення» розширяє варіант «Перевірити статус замовлення» бо змінює його інформативність, відображає у новому полі причину відхилення.

7. Відношення варіанту «Змінити статус замовлення» розширяє варіант «Перевірити статус замовлення» бо змінює його інформативність, відображає у якому статусі знаходиться замовлення з додатковою інформацією о планових термінах виконання замовлення, о технічних характеристиках проекту.

Рисунок 2.3 – Діаграма варіантів

Таке графічне відображення у простій формі показує замовнику інформаційної системи основні планові цілі розробки системи, узгодити та сформувати в усіх учасниках розробки інформаційної системи одне загальне бачення функціоналу.

Побудована діаграма послідовності для варіанту використання «Створити замовлення» (див. рис. 2.4). Діаграма відповідає успішному варіанту ходу подій. Кожен варіант використання (Use case) може бути відображений у такій схемі діаграми послідовності.

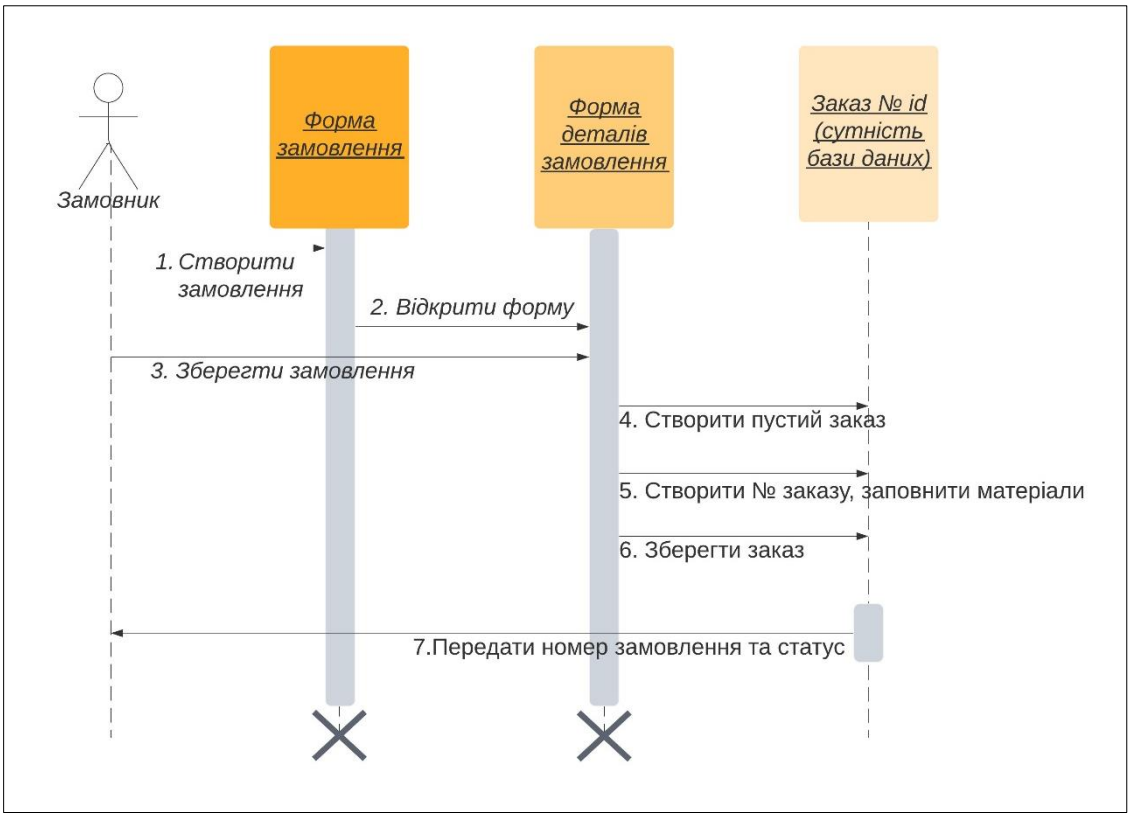

Рисунок 2.4 – Діаграма послідовності для варіанту використання «Створити замовлення»

Можемо бачити з діаграми взаємодію форм та сутності Заказу. Після завершення сьомої операції «Передати номер замовлення та статус» форми вибору замовлення та деталей замовлення зникають. Діаграма показує етапи збереження даних на формі і в сутності Заказу, тобто дані, які ввів Замовник з форми передаються у сутність Заказ, потім данні на формі знищуються.

Додамо на діаграму послідовності назви операцій та класів, які будуть створюватися при послідовності подій. Діаграма показує операції пов’язані із сутностями класів, які мають власні методи (див. рис. 2.5). Так, метод to_component("order") належить кореневій компоненті app, метод to_component("ordercreation") належить компоненті order, метод createOrder() належить дочірній компоненті ordercreation, функція createOrderServer() належить спільному сервісу orderService, orderCreateview() визивається на стороні сервера бази даних, OrderSerializer() аналізує права доступу та серіалізує відповідь відповідно до інтерфейсу класу Order передаючи response функції

createOrderServer. Вертикальні товсті стовпці у діаграмі відображають строк життя компонент та класів у процесі варіанту використання «Створити замовлення».

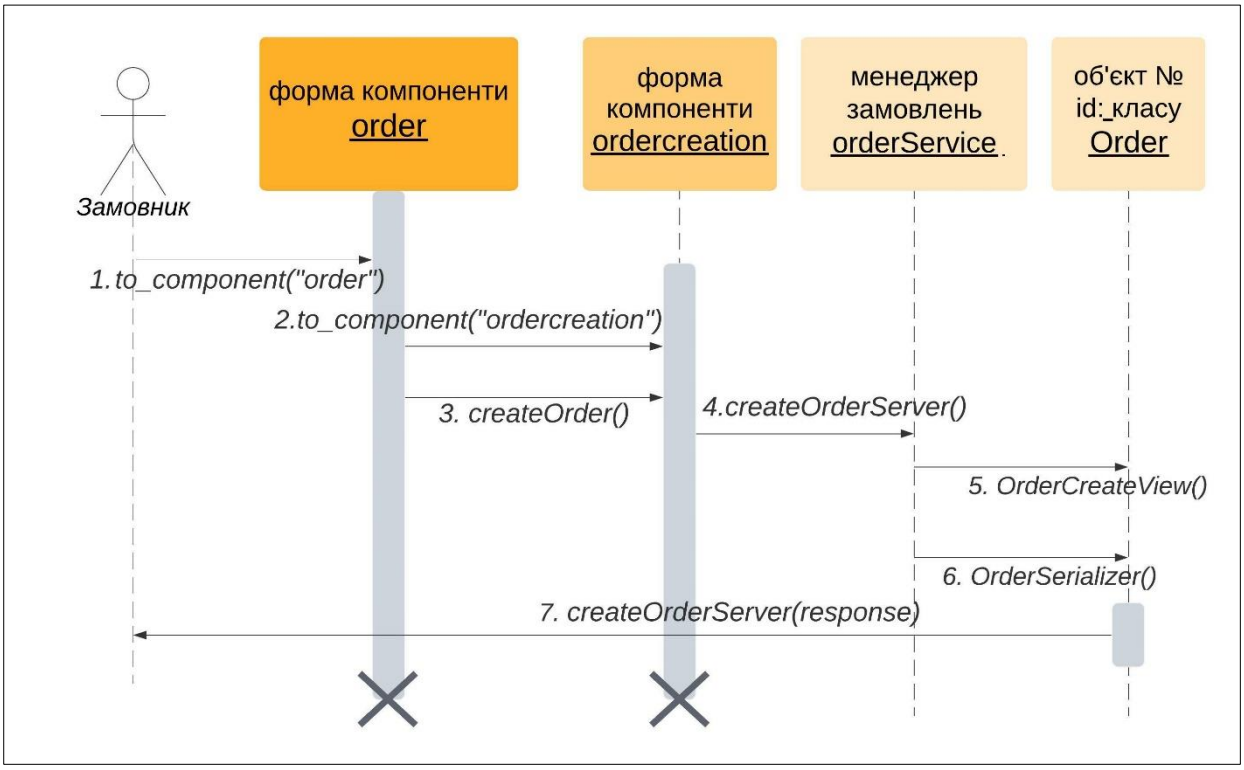

Рисунок 2.5 – Діаграма послідовності із класами та методами

Діаграма структури класів сутностей із їхніми атрибутами для варіанту використання «Створити замовлення» наведена у рисунку 2.6.

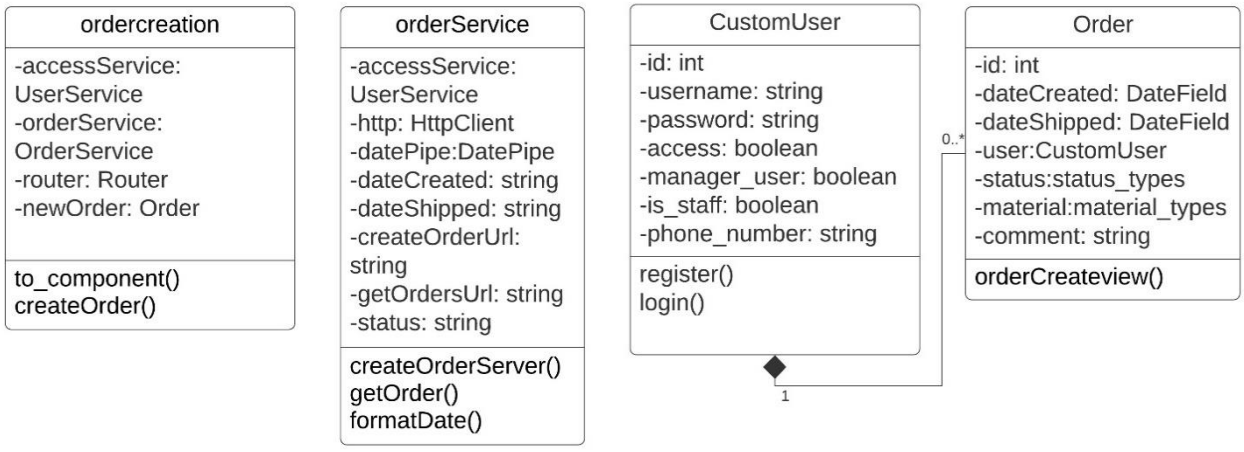

Рисунок 2.6 – Діаграма структури класів

Класи ordercreation та orderService будуть створюватися на стороні Front-end, авторизація буде проходити на стороні Back-end. Сутність Order та CustomUser буде створюватися на стороні Back-end та зберігатися у базі даних Back-end.

Атрибути класів характеризовані їх типом, що створює специфікацію для учасників розробки інформаційної системи – програмістів.

Залежність класів показана лінією відношенням Клієнт – Заказ. Відношення 1 до множини, тобто, один Клієнт може мати декілька Заказів, проте один Заказ має завжди одного Клієнта.

Клас Order має атрибут «status», який має декілька значень – станів замовлення клієнта. У системі будуть використовуватися такі статуси:

– «Новий»;
– «У роботі»;
– «Виконаний»;
– «Відхилено».

Доцільно побудувати діаграму станів для класу Order. Це допоможе прояснити основні послідовності дій для кожного із станів – статусів (див. рис. 2.7).

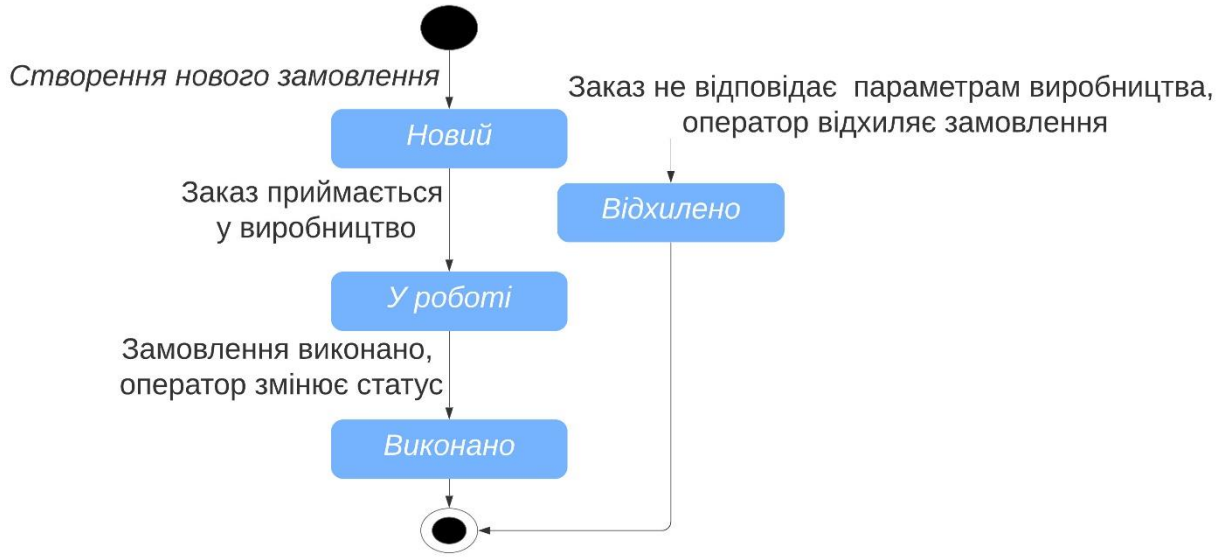

Рисунок 2.7 – Модель складної

User-case «Створити замовлення» завершується статусом замовлення «Новий», при цьому будуть використані декілька компонентів системи:

1. Сутності представлення форм та внесення інформації для клієнта розташовані на компонентах на стороні Front-end;
2. Сутності компонентів на стороні Front-end, які передають інформацію у потрібному форматі до компонентів на стороні Back-end;
3. Сутності компонентів обробки введеної інформації, створення класів на стороні Back-end;
4. Сутності компонентів, які зберігають створені класи у базі даних на стороні Back-end;
5. Сутності компонентів на стороні Back-end, які передають інформацію про створений клас Заказ на сторону Front-end.

Коли Оператор приймає заказ у виробництво, він також має заходити через інтерфейс Front-end, а зміни статусу повинні обробитися та зберегтися на стороні Back-end. Таким чином, User-case «Змінити статус замовлення» завершується статусом «У роботі» або «Виконаний», при цьому будуть використані декілька компонентів системи:

1. Сутності представлення форм та внесення інформації для оператора розташовані на компонентах на стороні Front-end;
2. Сутності компонентів на стороні Front-end, які передають інформацію у потрібному форматі до компонентів на стороні Back-end;
3. Сутності компонентів обробки введеної інформації, створення класів на стороні Back-end;
4. Сутності компонентів, які зберігають створені класи у базі даних на стороні Back-end;
5. Сутності компонентів на стороні Back-end, які передають інформацію про зміну значення атрибуту Status класу Заказ на сторону Front-end.

Аналогічно сформованим діаграмам для варіанту використання (use case) «Створити замовлення» можна сформувати діаграми для інших варіантів використання (use cases): варіанту «Перевірити статус замовлення», варіанту

«Змінити статус замовлення», варіанту «Відхилити замовлення», варіанту «Сформувати список замовлень».

2.2 Архітектура додатку

Для представлення фізичної моделі розміщення компонентів представимо специфікацію інформаційної системи для системних інженерів. Данна специфікація представлена у вигляді діаграми розміщення основних компонентів архітектури інформаційної системи. На діаграмі представлені фізичне розташування компонентів, обрані технології та їх взаємодія.

Таке представлення дає змогу системному інженеру розрахувати необхідну потужність ресурсів та економічну вартість використання даних інструментів

На даному етапі, системний інженер, також, може проаналізувати безпечність обраних сервісів, не оптимальне використання ресурсів.

Також оцінюються вірогідність досягнення цілей, щоб інформаційна система була мобільною – здатною взаємодіяти із різними середовищами операційних систем, масштабованою – здатною до збільшення користувачів.

Архітектура інформаційної системи складається за таких компонентів (див. рис. 2.8):

– Back-end компоненти, яка буде оброблювати запити від Front-end компоненти, взаємодіяти з базою даних; дана компонента буде написана за допомогою фреймворку Django REST;

– Front-end компоненти, яка обробляє запити користувачів системи та взаємодіє із Back-end компоненти через API інтерфейс; дана компонента буде написана за допомогою фреймворку Angular8;

– сервіси AWS для публікації та хостингу web-додатку;

– сервіси Bitbucket у поєднанні з програмою Docker для автоматичного розгорнення змін додатку на серверах AWS.

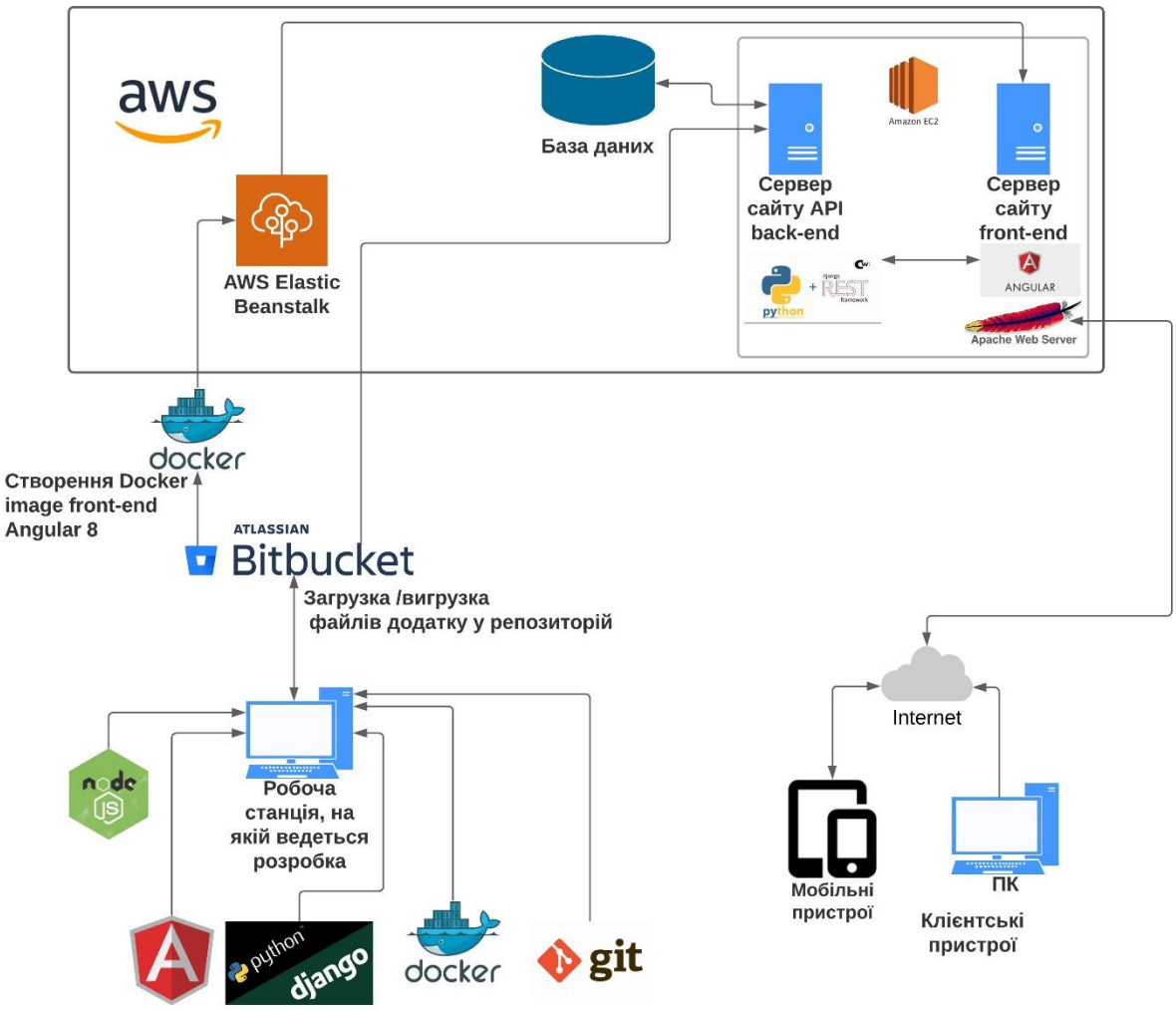

Рисунок 2.8 – Діаграма фізичної моделі розміщення компонентів

Django – це високорівневий веб-фреймворк для мови програмування Python, що дозволяє швидко розробляти динамічні веб-програми та включають пакети аутентифікації. Даний фреймворк вільний та використовує шаблон проектування MVC. Також, на відміну від інших фреймворків, Django обробники URL конфігуруються явно, за допомогою регулярних виразів. Для роботи з базою даних Django використовує власний ORM (Object-Relational Mapping), у якому модель даних описується класами Python, і генерується схема бази даних [2, 10].

Django REST Framework (DRF) – це бібліотека, яка працює зі стандартними моделями Django для створення гнучкого та потужного API для проекту [13, 14].

Основна архітектура API DRF складається з 3-х шарів: серіалізатора, виду (ViewSet) та маршрутизатора.

Серіалізатор перетворює інформацію, що зберігається в базі даних та визначену за допомогою моделей Django, у формат, який легко та ефективно передається через API.

Вигляд (ViewSet) визначає функції (читання, створення, оновлення, видалення), які будуть доступні через API.

Маршрутизатор визначає URL-адреси, які надаватимуть доступ до кожного виду (ViewSet).

Angular 13 відкрита та вільна платформа для розробки клієнтських web-додатків, написана мовою TypeScript, що розробляється командою з компанії Google. Заснований на компонентах фреймворк для створення веб-додатків, що масштабуються, має набір добре інтегрованих бібліотек, що охоплюють широкий спектр функцій: маршрутизація, управління формами, клієнт-серверна взаємодія, набір інструментів розробника, які допоможуть вам розробляти, збирати, тестувати та оновлювати ваш код. Angular, перш за все, націлений на розробку SPA-рішень (Single Page Application), тобто односторінкових додатків [13].

TypeScript [16] виглядає як звичайний код JavaScript з додаванням типізованих даних. Зручність типів полягає в тому, що це запобігає випадковому або неправильному використанню даних. Це дає слабко типізованій мові суворіший підхід до кодування. TypeScript вказує на помилки компіляції під час компіляції (що відбувається під час розробки). Тому, помилки легше протестувати до розгортання робочої версії.

Angular [13] використовує безліч концепцій, запозичених з таких технологій, як Java/.net, тому досвідченим back-end розробникам Angular здасться знайомим і не представлятиме особливої проблеми. Архітектура

Angular складається з: Modules, Dependency Injection, Components, Performance techniques, HTTP Services.

Хмарні сервіси AWS (Amazon Web Services) – комерційна публічна хмара, що підтримується і розвивається компанією Amazon з 2006 року. Надає передплатникам послуг як за інфраструктурною моделлю (віртуальні сервери, ресурси зберігання), так і платформного рівня (хмарні бази даних, хмарне програмне забезпечення, хмарні безсерверні обчислення, засоби розробки) [8].

Bitbucket – веб-сервіс для хостингу проектів та їхньої спільної розробки, заснований на системах контролю версій Git. За призначенням і основним пропонованим функціям аналогічний GitHub, від якого відрізняється з одного боку меншою базою користувачів, а з іншого, має переваги в плані розміщення непублічних репозиторіїв – можливістю їхнього безкоштовного хостингу з обмеженням на розмір команди [12].

Bitbucket Pipelines – це комплексний сервіс CI/CD (continuous integration / continuous deployment), вбудований у Bitbucket. З його допомогою можна виконувати автоматичне збирання проекту, тестування і навіть розгортання коду, використовуючи конфігураційний файл з репозиторію [12].

Docker – програмне забезпечення для автоматизації розгортання та керування програмами в середовищах з підтримкою контейнеризації, контейнеризатор додатків. Дозволяє «упакувати» додаток з усім його оточенням та залежностями у контейнер, який може бути розгорнутий на будь-якій Linux-системі з підтримкою cgroups у ядрі (контрольна група процесів в Linux-системі), а також надає набір команд для керування цими контейнерами [11].

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ОНЛАЙН ЗАМОВЛЕННЯ ПРОДУКЦІЇ

3.1 Розгортання проекту Django

На локальній машині був встановлений Python 3.7.4. Також, встановили віртуальне середовище для того, щоб встановлювати додаткові пакети фреймворків потрібних версій у відповідне середовище, яке прив'язано до проекту командами «pip install virtualenv» та «D:\Desktop\prod\api> virtualenv .»

У папці проекта створилася папка для встановлення пакетів. Активували віртуальне середовище командою: «D:\Desktop\prod\api> scripts\activate»

Встановили у наше віртуальне середовище останню версію фреймворку Django-3.2.9: «pip install django».

Створено проект **apiprod** командою: «django-admin startproject apiprod .» у папці src. Ініціалізуємо bitbacket репозитарій командами:

(api) PS D:\Desktop\prod\api\src> git init .

(api) PS D:\Desktop\prod\api\src> git remote add origin https://alexey_sinitskiy@bitbucket.org/alexey_sinitskiy/api.git

Стандартний функціонал Django включає інструменти адміністрування сайту з відповідним розподілом прав доступу. Проте, для програмування ролей потрібно перевизначити стандартний клас User.

Для цього створено новий додаток myuser Django у рамках проекту **apiprod** командою: "python manage.py startapp myuser". Наслідуємо базовий клас користувача.

Django має такі базові класи користувачів: AbstractUser та AbstractBaseUser. Якщо потрібно просто додати існуючому користувачеві додаткові поля, то треба використовувати AbstractUser, тому що це простіше змінювати. Якщо необхідно змінити процес аутентифікації, то треба використовувати AbstractBaseUser та перевизначати усі поля та їх типи [2].

Додали до нового класу CustomUser нове поле – manager_user вид користувача. Вид користувача буде мати тип Boolean. Стандартні реквізити моделі User вже мають поля: IsStuff, IsAmin. Таким чином, ми зможемо розділити логіку програми на 4 ролі:

– клієнт (звичайний user);
– оператор (IsStaff = True);
– адміністратор (IsAdmin = True);
– менеджер (manager_user = True).

Реалізуємо вище сказане у файлі api\src\myuser\models.py:

```
from django.db import models
from django.contrib.auth.models import AbstractUser
class CustomUser(AbstractUser):
    manager_user = models.BooleanField(default = False)
```

У файл settings.py додамо інформацію про створений додаток у структуру INSTALLED_APPS, а також параметр моделі користувача за замовченням: AUTH_USER_MODEL = 'myuser.CustomUser'.

Також, додамо поле у панель адмімністратора нове поле для класу CustomUser у файлі api\src\myuser\admin.py:

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import CustomUser
from django.contrib.auth.forms import UserChangeForm
class MyUserChangeForm(UserChangeForm):
    class Meta(UserChangeForm.Meta):
        model = CustomUser
class MyUserAdmin(UserAdmin):
    form = MyUserChangeForm
    fieldsets = UserAdmin.fieldsets + (
            (None, {'fields': ('manager_user', 'phone_number', )}),  )
admin.site.register(CustomUser, MyUserAdmin)
```

Слід зауважити, що створену модель користувача слід поєднувати при реєстрації саме з адмінським класом UserAdmin з бібліотеки django.contrib.auth.admin, інакше, модель не матиме усіх необхідних методів з «коробки» Django.

Для відображення змін моделей у базі даних Django, потрібно запустити команди: «python manage.py makemigrations» та «python manage.py migrate». Після цього створяться таблиці та сама база даних засобами Django ORM (Object Relational Mapping) : api\src\db.sqlite3. Це дозволяє нам взаємодіяти з базою даних, використовуючи Python код, а не SQL. Для тестової розробки проекту зручно використовувати базу даних, що створюється командою «migrate», проте, фреймфорк дає можливість підключатися до сторонніх баз даних, конфігуруючи структуру DATABASES у фалі settings.py проекту.

Запустили локально сервер Django для перевірки роботи сайту та змін у сутності користувача: «python manage.py runserver» за локальною адресою http://127.0.0.1:8000/. Для того щоб зайти у адмінську панель запущеного додатку Django за адресою http://127.0.0.1:8000/admin потрібно створити користувача з правами адміністратора. Виконали команду: python manage.py createsuperuser, створили користувача admin та зайшли у панель (див. рис. 3.1).

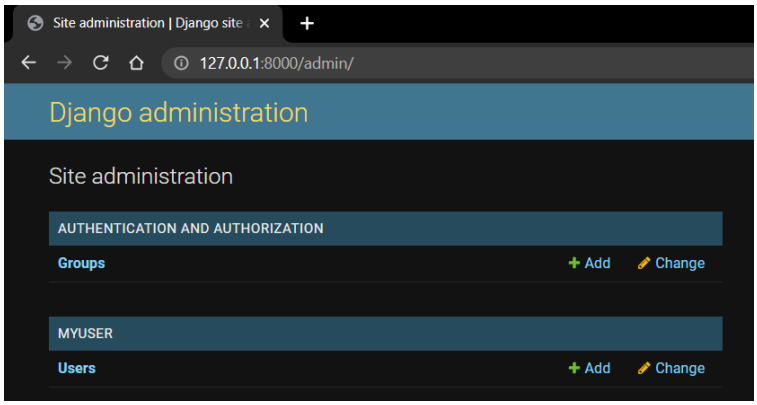

Рисунок 3.1 – Панель адміністратора Django

Стандартний клас User був перевизначений та нове поле manager_user було добавлене в інтерфейс та у модель користувача. При створені користувача admin

вже був використаний клас CustomUser, в якому є потрібні реквізити для розподілу ролей (див. рис. 3.2)

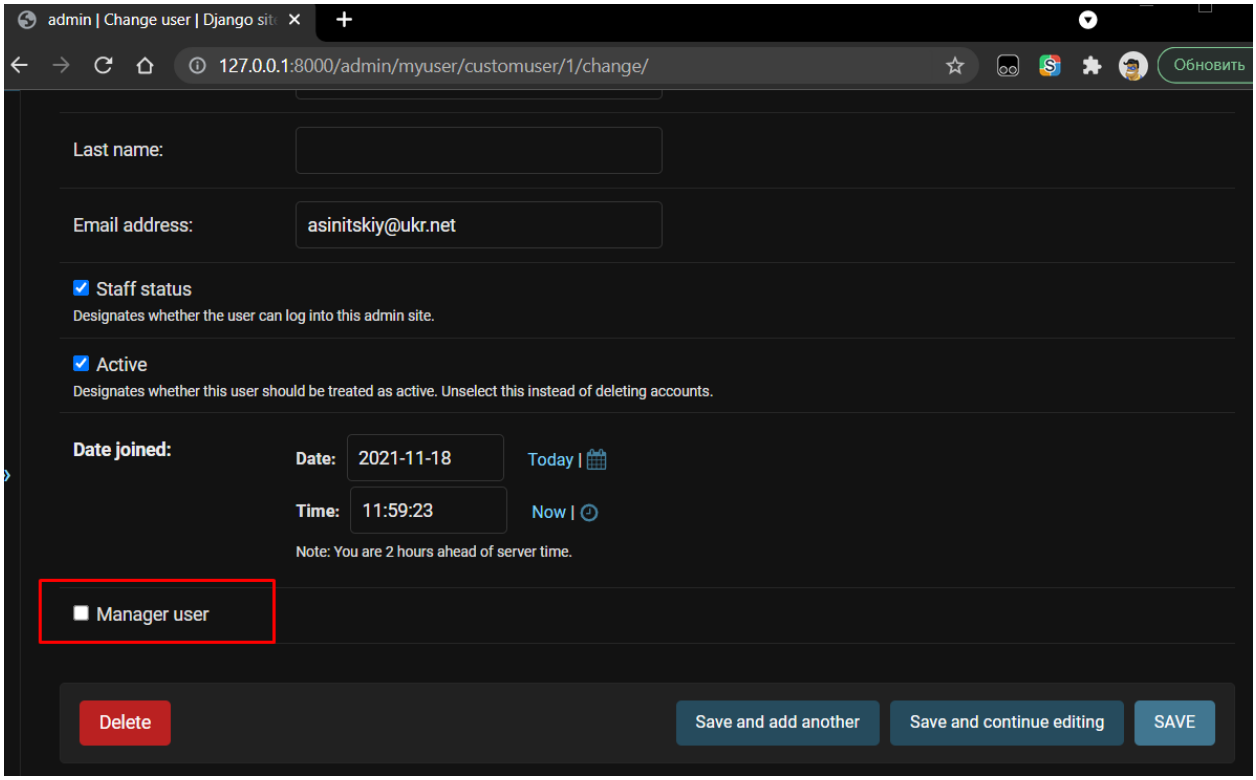

Рисунок 3.2 – Клас CustomUser унаслідував усі поля AbstractUser, додав своє поле manager_user та став основною моделлю користувача

Для реалізації функціоналу сутності замовлення створили новий додаток Order у проекті та клас Order відповідно у файлі api\src\order\models.py:

```
from django.db import models
from django.contrib.auth import get_user_model
from django.conf import settings
User = settings.AUTH_USER_MODEL
class Order(models.Model):
    status_types = (
        (1, 'Новий'),
        (2, 'У роботі'),
        (3, 'Виконаний'),
        (4, 'Відхилено')        )
```

```
material_types = (
    (1, 'Деревина'),
    (2, 'Кераміка'),
    (3, 'Камінь'),
    (4, 'Дсп'),
    (5, 'Сталь'),
    (6, 'Скло'),        )
```

status = models.IntegerField(verbose_name = 'Status_types', choices = status_types)

material = models.IntegerField(verbose_name = 'Material_types', choices = material_types)

user = models.ForeignKey(User, verbose_name = 'User', on_delete=models.CASCADE)

dateCreated = models.DateField()

dateShipped = models.DateField(blank=True, null=True)

comment = models.CharField(verbose_name = 'Comment', max_length = 100, blank=True)

phone_number = models.CharField(verbose_name = 'Phone number', max_length = 15, blank=True)

Клас Order має усі необхідні поля для створення замовлення та запису даних полів у базу даних. Були створені фіксовані списки статусів та матеріалів, які будуть використовуватися у інтерфейсі клієнта та оператора. Клас користувача отримується внутрішньою функцією фреймворку get_user_model() бібліотеки auth. У редакторі коду Visual studio code бачимо наступну структуру проекту (див. рис. 3.3).

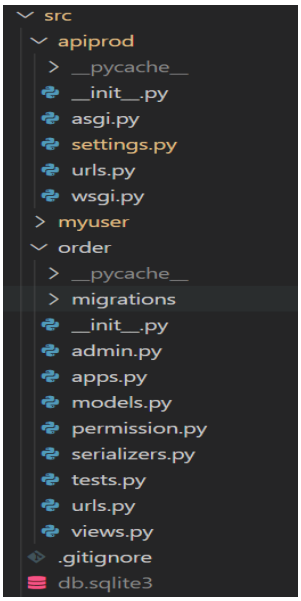

Рисунок 3.3 – Структура проекту apiprod

У файл settings.py додали інформацію про створений додаток Order у структуру INSTALLED_APPS.

Після оновлення міграцій моделі додатку Order у базу даних та реєстрації у файлі api\src\order\admin.py, запускаємо сервер та бачимо можливість роботи з замовлення через адмінську панель Django (див. рис. 3.4).

Замовлення дозволяє ввести дату створення, дату виконання, обрати користувача, обрати тип матеріалу та статус замовлення. Ці всі поля доступні через адмінські права доступу, проте звичайні користувачі не зможуть зайти до панелі та редагувати ці поля, та і це не планувалося технічним завданням. Користувачі типу Клієнт, Оператор та Менеджер будуть «розмовляти» із компонентною back-end через клієнтську програму за допомогою API інтерфейсу.

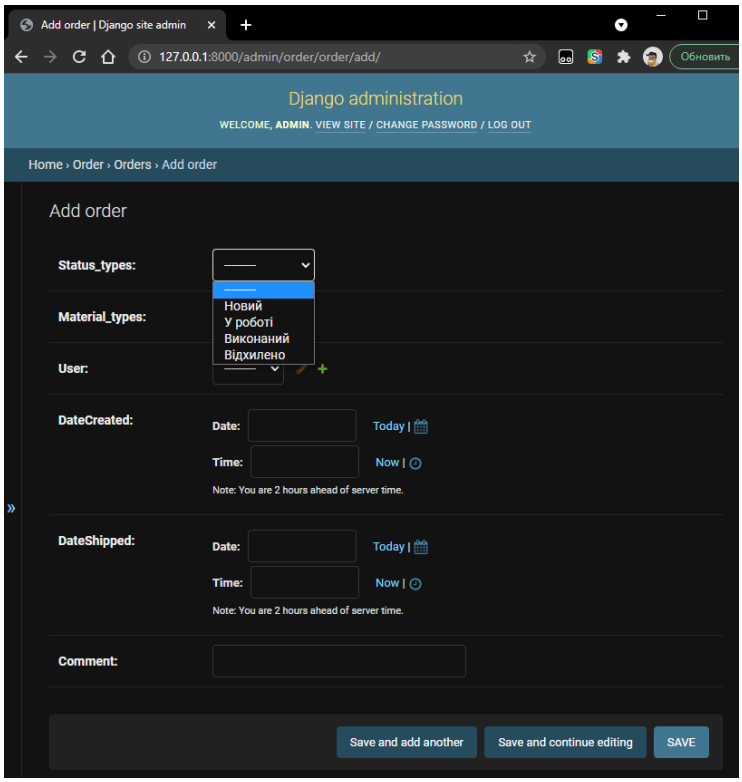

Рисунок 3.4 – Створення замовлення Order через панель адміністратора

3.2 Створення API

Для розробки API нам знадобляться спеціальні інструменти, які називаються серіалізатори [13], які перетворюють інформацію у формат, який легко та ефективно передається через API. Дані серіалізатори знаходяться у пакеті Django Rest framework (DRF). Встановимо даний пакет останньої версії 3.12.4 командою: pip install djangorestframework; та розробимо необхідні процедури та функції. У файл settings.py додамо інформацію про встановлений пакет rest_framework у структуру INSTALLED_APPS.

У файлі api\src\order\serializers.py внесли в нього дані про класи які треба серіалізувати. Клас OrderSerializer унаслідує всі методи та поля стандартного класу серіалізації пакету DRF:

```
from .models import Order
from rest_framework import serializers
class OrderSerializer(serializers.ModelSerializer):
```

```
    user = serializers.HiddenField(default=serializers.CurrentUserDefault())
    class Meta:
        model = Order
        fields = '__all__'
```

Поле User при серіалізації буде скрито та підставлятися за замовченням значення поточного аутентифікованого користувача. Конструкція fields = '__all__' означає, що буде серіалізовано усі поля класу Order.

Для опису логіки серіалізації за принципами REST потрібно розробити файл-вигляд Django (ViewSet) api\src\order\views.py:

```
from django.shortcuts import render
from .models import Order
from rest_framework import generics
from .serializers import OrderSerializer
from .permission import IsOwner
from rest_framework.permissions import IsAdminUser, IsAuthenticated
class OrderList(generics.ListAPIView):
    serializer_class = OrderSerializer
    permission_classes = (IsAuthenticated, IsOwner)
    def get_queryset(self):
        user = self.request.user
        return Order.objects.filter(user=user)
```

та додатковий файл доступу api\src\order\permission.py:

```
from rest_framework import permissions
class IsOwner(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        return obj.user == request.user
```

Клас OrderList унаслідується від generics класу пакету DRF, в якому вже прописаний метод GET для отримання списку. Доступ до всіх своїх замовлень

може отримати через інтерфейс API тільки автор замовлення, що перевіряється через умову permission_classes = (IsAuthenticated, IsOwner).

Для формування посилань на функцію представлення будуємо у файлах url.py додатку та проекту. Відповідно код, який зв’язує функцію представлення OrderList. as_view() із url, написаний у файлі api\src\order\urls.py:

```
from django.urls import path
from .views import OrderList
urlpatterns = [
    path('order/', OrderList.as_view()),]
```

Усі файли усіх додатків проекту об’єднються у файлі url.py проекту api\src\apiprod\urls.py:

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('order.urls')),
    path('api-auth/', include(('rest_framework.urls', 'rest_framework'))),]
```

У коді вище, був підключений url.py додатку Order до головного файлу url-ів проекту apiprod.

Перевіримо доступ до api, яке повинно віддавати увесь список за адресом http://127.0.0.1:8000/api/v1/order (рис. 3.6). Програма не дає отримати дані через те, що користувач, який їх запитує є не авторизований:

```
{    "detail": "Authentication credentials were not provided."    }.
```

Програмою PostMan протестували доступ до даних, вказавши в авторизації Auth Base логін та пароль користувача Test, який є автором третього замовлення та отримали у відповідь дані у json форматі (див. рис. 3.5).

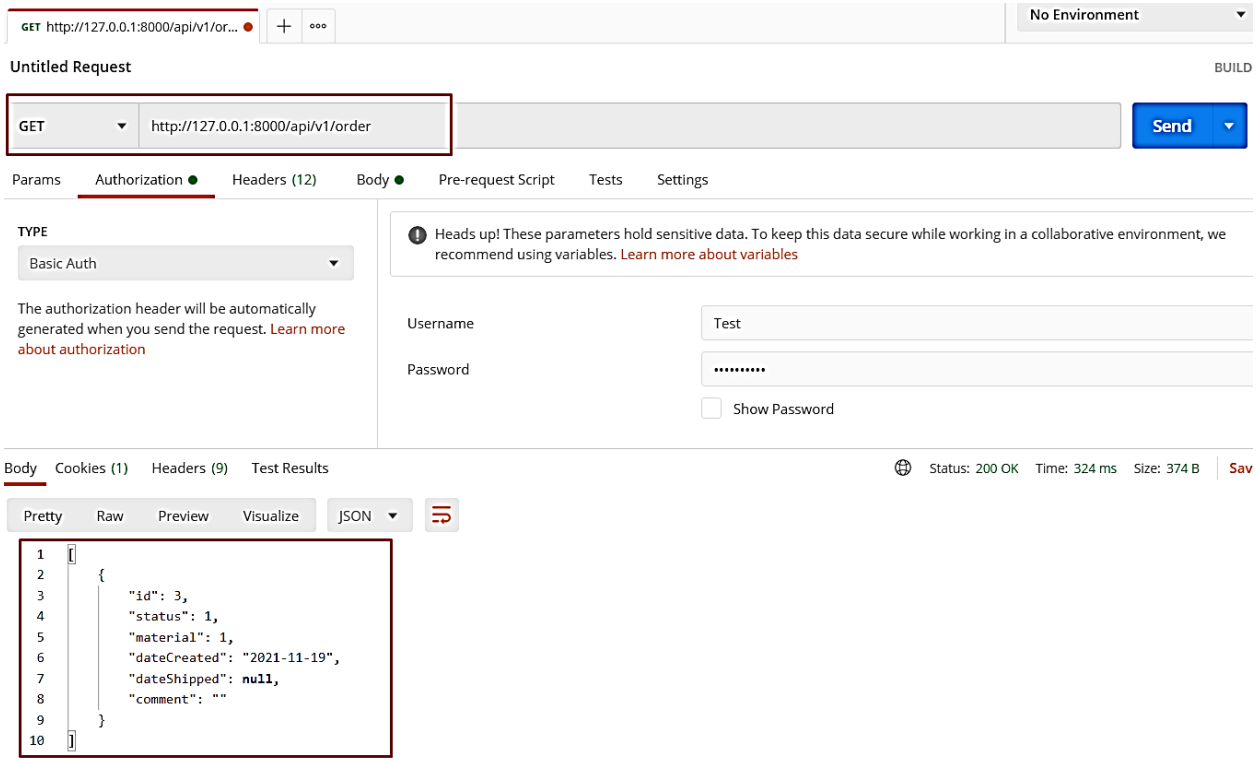

Рисунок 3.5 – Тестування API у програмі PostMan

Аналогічно створили вигляди (ViewSet) для POST та UPDATE запитів, які створюють та редагують замовлення:

```
class OrderCreateView(generics.CreateAPIView):
    serializer_class =  OrderSerializer
    permission_classes = ( IsAuthenticated, )
class OrderUpdateView(generics.RetrieveUpdateDestroyAPIView):
    serializer_class = OrderSerializer
    queryset = Order.objects.all()
    permission_classes = (IsAdminUser, )
```

Дані класи виглядів використовують той же серіалізатор, що був створений раніше OrderSerializer. Зв’язали вигляди із url посиланнями у фалі tm_project\apitm\urls.py:

```
 from django.urls import path
from .views import OrderList, OrderCreateView, OrderUpdateView
urlpatterns = [
    path('order/', OrderList.as_view()),
```

path('ordercreate/', OrderCreateView.as_view()),

path('orderupdate/<int:pk>/', OrderUpdateView.as_view()),]

Програмою PostMan протестували створення замовлення користувачем Test, відправивши на адресу http://127.0.0.1:8000/api/v1/ordercreate/ наступне повідомлення:

{'status': '1',

'material': '2',

'dateCreated': '2021-11-01',

'phone_number': '063121212'}

та отримали у відповідь дані у json форматі, про створений об’єкт з id=4 класу Order :

{ "id": 4,

"status": 1,

"material": 2,

"dateCreated": "2021-11-01",

"dateShipped": null,

"comment": "",

"phone_number": "063121212" }

Для редагування окремого замовлення використовується атрибут <int:pk>, в якого передається id замовлення.

Аналогічно, були створені API для реєстрації та авторизації користувача. Усі API представлені у таблиці 3.1. Повні коди файлів представлені у додатку А.

Таблиця 3.1 – Створені API

API	**Опис**
http://127.0.0.1:8000/api/v1/order/	видає усі замовлення авторизованого користувача
http://127.0.0.1:8000/api/v1/ordercreate/	створює нове замовлення
http://127.0.0.1:8000/api/v1/orderupdate/<int:pk>/	редагує замовлення відповідного id
http://127.0.0.1:8000/api/v1/account/register/	створює нового користувача

Продовження таблиці 3.1 – Створені API

http://127.0.0.1:8000/api/v1/account/login/	надає username та password у форматі base64

Таким чином, була розроблена компонента back-end за допомогою Django Rest фреймворку. Слід зауважити, що посилання http://127.0.0.1:8000 у API зміняться при публікації сайту на хостингу. Для забезпечення біль захищеної системи потрібно використовувати TokenAuthentication [10, с.264] методи замість BasicAuthentication, використовуючи додаткові пакети, які генерують токени JSON Web Token (JWT) та слідкують за їхнім часом життя, наприклад пакет Django Djoser [17].

3.3 Розгортання проекту Angular8

Для реалізації front-end компоненти слід встановити платформу Nodejs версії 16.13.0 та залежності перелічені у файлі package.json (додаток Б.) командою: npm install [9].

У каталозі проекту були створені 3 кореневі компоненти та 4 дочірні компоненти для розподілу функціоналу (див. табл. 3.2).

Таблиця 3.2 – Створені компоненти

Конструктор створення	**Ціль**
ng new prodapp	створює проект prodapp із корневим компонентом APP, який запускається першим при зборці додатку, керує наступним виводом компонентів
ng generate component user	керує виводом авторизації та реєстрації, відповідно компонентами login та register
ng generate component order	керує виводом компонентів ordercreation та ordertable

Продовження таблиці 3.2 – Створені компоненти

ng generate component user/register	керує виводом реєстраціїї користувача та функціоналом реєстрації
ng generate component user/login	керує виводом авторизації користувача та функціоналом авторизації
ng generate component order/ordercreation	впроваджує створення нового замовлення
ng generate component order/ordertable	виводить усі замовлення авторизованого користувача

У компонентах прописані функції та процедури, які оброблюють дані введені користувачем у формах та передають інформацію до спільних модулів – сервісів.

Для реалізації компонентів ordercreation, ordertable та order використовується спільний модуль src\app\order\order.service.ts, в якому реалізовані методи createOrderServer, formatDate та getOrder:

```
 public createOrderServer(body) {
   let httpOptions = {
     observe: 'response' as 'response',
     headers: new HttpHeaders({
       'Content-Type': 'application/json',
       'Authorization': 'Basic ' + this.accessService.access.password }),  }
     return this.http.post(this.createOrderUrl, body, httpOptions) ;    }
  public formatDate(d:Date) {
   return this.datePipe.transform(d,"yyyy-MM-dd")
  }
  public getOrder() {
   let httpOptions = {observe: 'body' as 'body',
                      headers: new HttpHeaders({'Content-Type':  'application/json',
                      'Authorization': 'Basic '+ this.accessService.access.password }),
                    }
        return this.http.get(this.getOrdersUrl, httpOptions );  }}
```

Функція createOrderServer формує POST запит до серверу API back-end компоненти та вертає відповідь від серверу.

Змінна this.getOrdersUrl була визначена при ініціалізації сервісу :

import { environment } from 'src/environments/environment';

private createOrderUrl = environment.createOrderUrl;.

Сховище environment зберігає змінні, які використовуються під час розробки проекту та під час зборки проекту. Відповідно, змінна має таке значення – createOrderUrl : 'http://127.0.0.1:8000/api/v1/ordercreate/'.

Заголовки headers формуються відповідно до API авторизації через метод Base Auth. Для цього використовується інший спільний сервіс accessService, який має клас access та імпортований у сервіс OrderService. Екземпляр класу access створюється при ініціалізації сервісу accessService, коли користувач успішно авторизувався на сайті та зберігається під час усієї сесії користувача, тобто сервіс є синглтоном. Коди сервісу accessService показані у додатку В.

Значення access.password зберігається у наступному форматі: btoa(userAccess.username + ':'+ userAccess.password). Функція btoa() кодує ім’я користувача та пароль у формат Base64.

Виклик функції createOrderServer виконується із файлу компоненти ordercreation.component.ts наступною процедурою :

```
createOrder() {
   let body: any;
   body = { 'dateCreated': this.orderService.formatDate(new Date()),
    'status': '1',
    'material': this.newOrder.material,
    'phone_number': this.newOrder.phone_number }
      this.orderService.createOrderServer(body)
    .subscribe((response: HttpResponse<Order>) => {
      this.success = true;
    } ,
     err => {this.errors = err['error'];  console.log(err);  }    ) }
```

Данна процедура формує повідомлення body до серверу, передає функції createOrderServer та підписується на відповідь від неї за допомогою методу subscribe(). Данна підписка використовується через функціональність фреймворку Angular виконувати асинхронні запити. При успішній відповіді у компоненті змінюється значення змінної success = true. Ця змінна прив’язана у html сторінки відповідного компоненту ordercreation.component.html за допомогою умовної прив’язки *ngIf="this.success" і змінює відображення блоків html:

```
<div class="card-container2" *ngIf="this.success" >
  <h1> ЗАМОВЛЕННЯ УСПІШНО ПЕРЕДАНО В ОБРОБКУ </h1>
 ……..
  </div>
</div>
```

Виклик процедури createOrder() відбувається із події кнопки ordercreation.component.html:

```
<div class="card" (click)="createOrder()">
        Підтвердити замовлення
  </div>
```

Користувач, зайшовши до сайту потрапляє на html сторінку кореневого компоненту та бачить простий інтерфейс. Протестувати додаток можна, запустивши локальний сервер командою: ng serve --open.

3.4 Реалізація інтерфейсу користувача

При вході на сторінку, користувач є неавторизованим, тому при натисканні на кнопку «Створити замовлення на виробництво», виводиться компонента user, яка має посилання на компоненти register та login (див. рис.3.6).

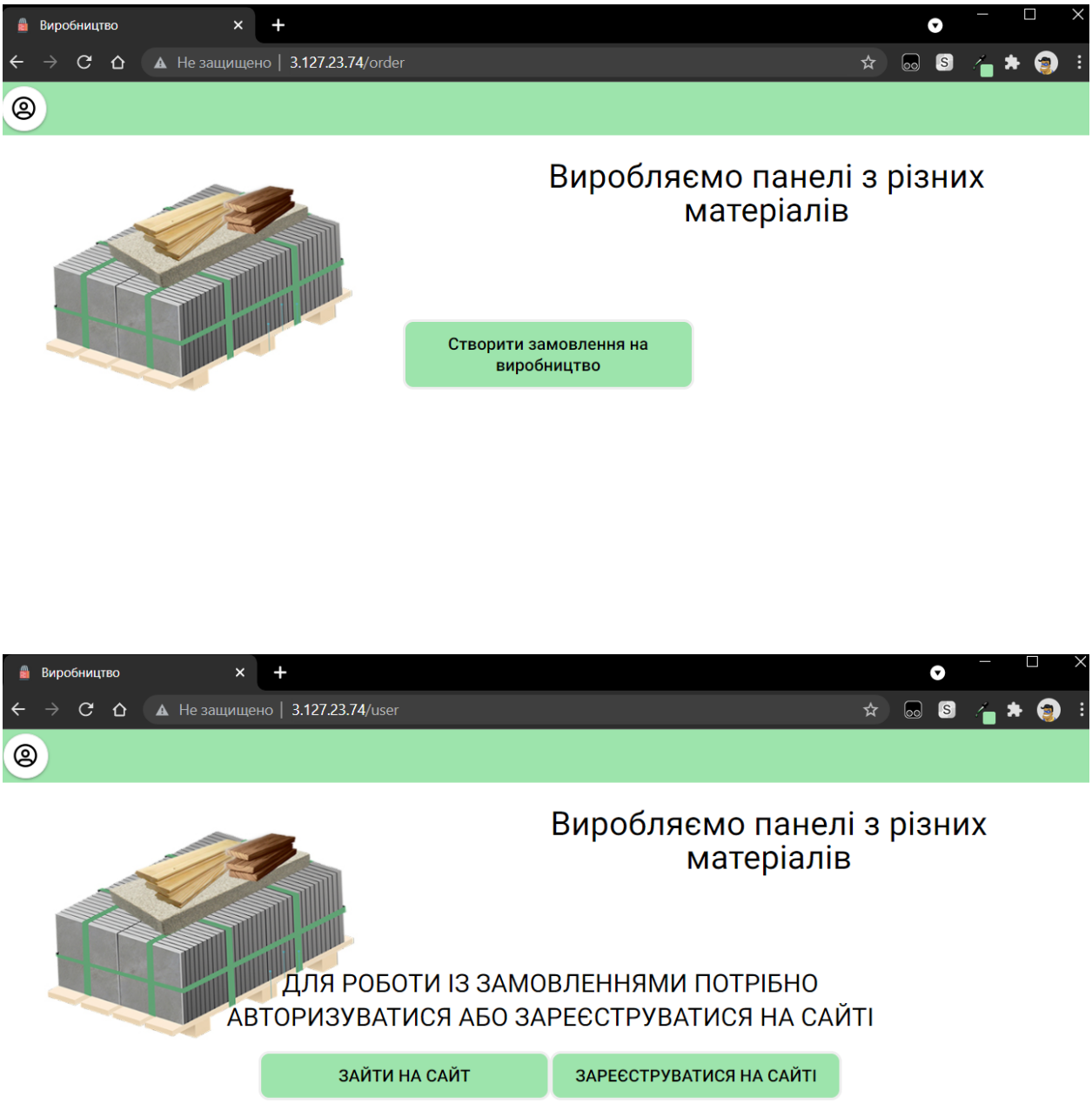

Рисунок 3.6 – HTML відображення компонентів APP та user

Пройшовши реєстрацію на сайті користувач потрапляє до компоненту order, якій відображає посилання на дочірні компоненти ordercreation, ordertable, відповідно на створення нового замовлення та перегляд здійснених замовлень (див. рис 3.7).

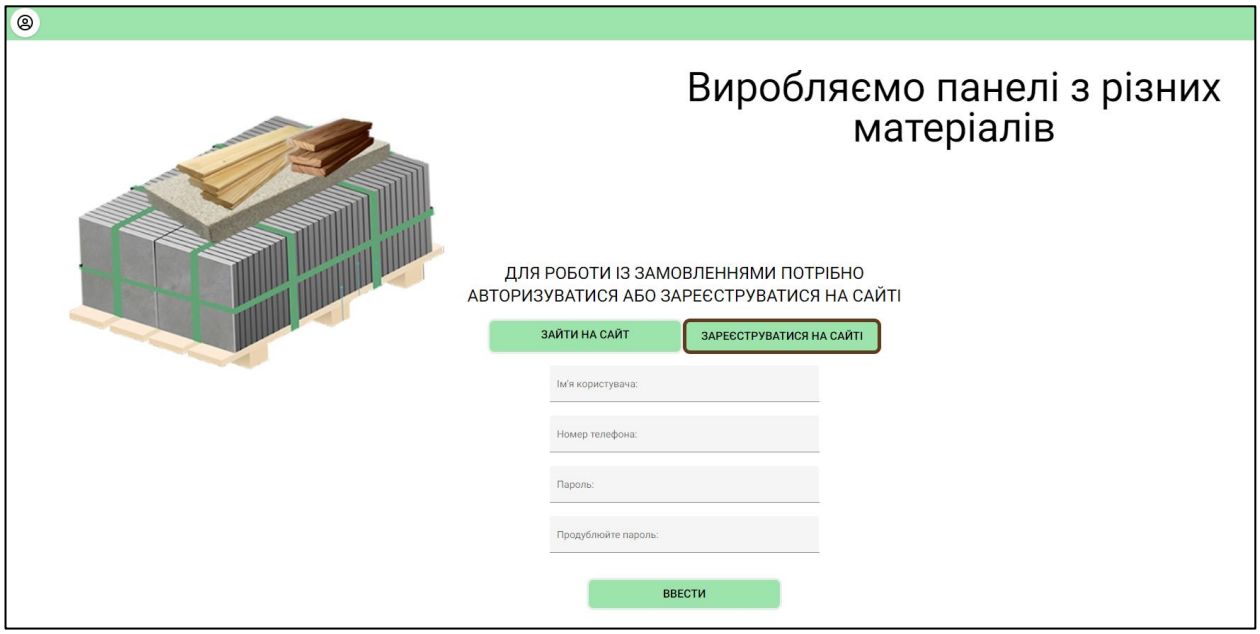

Рисунок 3.7 – HTML відображення компонентів APP та register

Коли користувач натискає на кнопку «Створити замовлення на виробництво», то потрапляє на відображення компоненти ordercreation, де заповнює вид сировини та свій телефон. Проте, телефон заповнюється автоматично при ініціалізації компоненти ordercreation та береться із спільного сервісу accessService після авторизації (див. рис. 3.8).

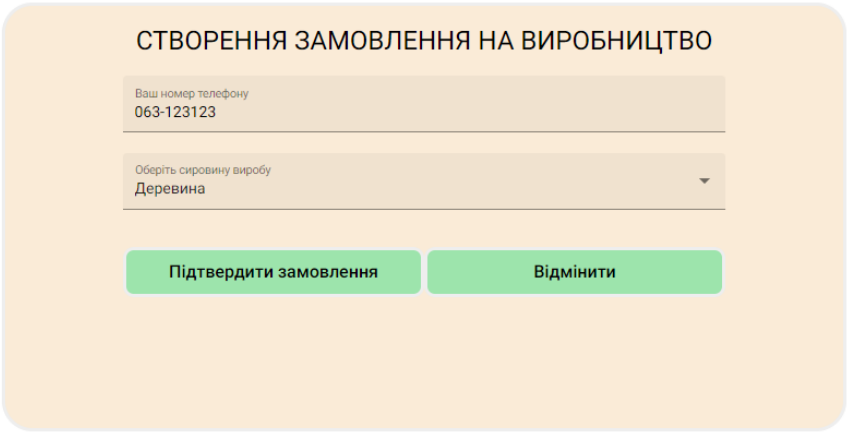

Рисунок 3.8 – HTML відображення компоненти ordercreation

Після створення замовлення, користувач повертається до компоненти order (див. рис. 3.9), якій виконується умова *ngIf="this.success" та змінюється відображення.

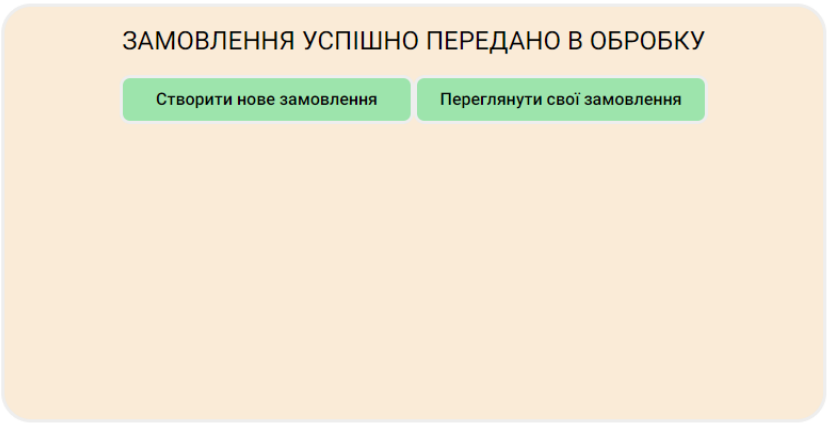

Рисунок 3.9 – HTML відображення компоненти order

Натиснувши кнопку «Переглянути свої замовлення» користувач потрапляє на відображення компоненти ordertable. Ця компонента також використовую спільний сервіс авторизації accessService та сервіс роботи із замовлення за допомогою OrderService. У сервісу OrderService, для отримання списку замовлень авторизованого користувача, використовується функція getOrder(). Виклик даної функції виконується при ініціалізаціїї компоненти ordertable, тобто, коли користувач натискає «Переглянути свої замовлення»:

```
ngOnInit( ): void {
    this.orderService.getOrder().subscribe(res=>{
    this.dataSource = res;
    this.dataSource.forEach(x =>
     { if (x.status ==  '1')  {x.status = 'Новий'}
       if (x.status ==  '2')  {x.status = 'У роботі'}
       if (x.status ==  '3')  {x.status = 'Виконаний'}
       if (x.status ==  '4')  {x.status = 'Відхилено'}
       if (x.material ==  '1')  {x.material = 'Деревина'}
       if (x.material ==  '2')  {x.material = 'Кераміка'}
```

```
if (x.material ==  '3')  {x.material = 'Камінь'}
if (x.material ==  '4')  {x.material = 'Дсп'}
if (x.material ==  '5')  {x.material = 'Сталь'}
if (x.material ==  '6')  {x.material = 'Скло'}      }      );   }) }
```

Після того, як функція orderService.getOrder() дала відповідь від back-end серверу, змінна res оброблюється відповідно до статусу та матеріалу, щоб відобразити користувачу текстову відповідність змінних, формується масив та передається сторінці html. Вивід усіх замовлень користувача Test показано на рисунку 3.10.

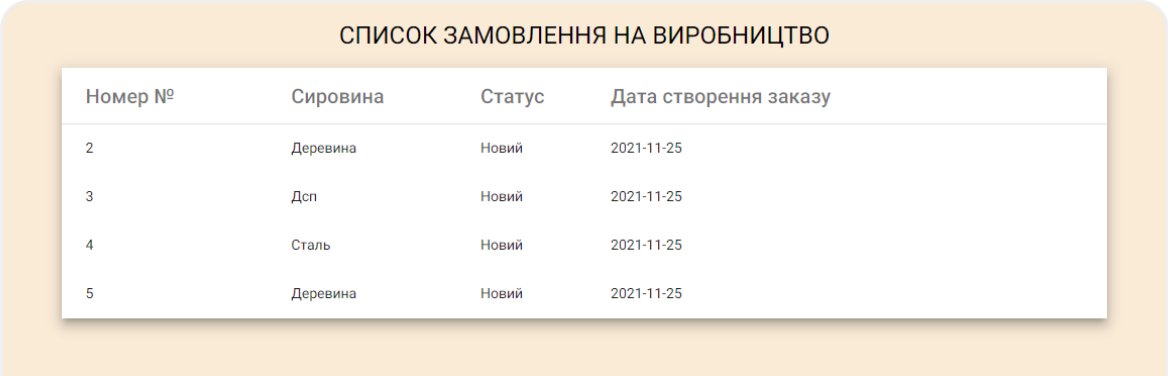

Рисунок 3.10 – HTML відображення компоненти ordertable

Переадресація сторінок [9] реалізована через бібліотеку @angular/router. Коли користувач заходить на сайт, то ініціалізується коренева компонента app.component.ts, яка постійно зберігається у пам’яті. Ця компонента керує наступним виводом компонентів у відповідній частині кореневої html сторінки. При цьому, коренева компонента не оновлюється, а оновлюється тільки частина html сторінки, яка позначається: <router-outlet></router-outlet>.

Для конфігурації маршрутів використовується константа у модулі кореневого компонента app.module.ts – Routes з типом імпортованого класу Routes із бібліотек Angualr:

```
import { Routes, RouterModule} from '@angular/router';
const appRoutes: Routes = [
  { path: '', redirectTo: 'order', pathMatch: 'full' },
  { path: 'user',  component: UserComponent,
```

```
        children: [ { path: 'login', component: LoginComponent},
                    { path: 'register', component: RegisterComponent}, ] },
      { path: 'order', component: OrderComponent},
      { path: 'ordercreation', component: OrdercreationComponent},
      { path: 'ordertable', component: OrdertableComponent}, ];
```

Також, клас Router можна використовувати у всіх компонентах, імпортувавши його, ініціалізувавши та визвавши у створених процедурах, наприклад, у компоненті order процедура public to_component():

```
import { Router } from '@angular/router';
@Component({
  selector: 'app-order',
  templateUrl: './order.component.html',
  styleUrls: ['./order.component.css'] })
export class OrderComponent implements OnInit {
  constructor(public accessService: UserService ,   private router: Router) { }
  ngOnInit() { }
  public to_component(path: string){
    this.router.navigate([path])}}
```

Методе this.router.navigate(path) переадресує відповідно до строкового значення параметру path до компоненти, зазначеної у маршрутах константи Routes кореневого модулю app.module.ts.

Таким чином, була розроблена компонента front-end за допомогою фреймворку Angular 13.

3.5 Розгортання Web-додатку

Для публікації Web-додатку були використані хмарні сервіси AWS [8]. Фактично, було розгорнуто дві інстанції серверів для хостингу двох сайтів: front-end компоненти та back-end компоненти.

Для розгортання back-end компоненти на Python фреймворку Django були вчинені наступні кроки:

1. Встановлена інстанція EC2 Ubuntu Server 20.04 LTS - (ami-0a49b025fffbbdac6) та сконфігуровані права доступу;
2. Через протокол SHH встановлені сервер Apache2, віртуальне середовище та необхідні модулі командою sudo apt-get install python3-pip apache2 libapache2-mod-wsgi-py3, sudo pip3 install virtualenv;
3. Через протокол SHH встановлені залежності Django у віртуальне середовище: djangorestframework==3.2.9, pip install django-cors-headers;
4. У каталог проекту був клонований проект back-end компоненти командою: git clone https://alexey_sinitskiy@bitbucket.org/alexey_si niskiy/api.git ;
5. Через Linux редактор vi були настроєні конфігураційні фали settings.py та apache2/sites-available/000-default.conf (додаток Б);
6. Перезапущено сервіс apache2.
7. Сайт back-end компоненти з функціоналом API доступний за посиланням http://3.66.174.12/.

Таким чином, усі API приймають вигляд у таблиці 3.2.

Таблиця 3.2 – API сайту http://3.66.174.12/

API	**Опис**
http://3.66.174.12/api/v1/order/	видає усі замовлення авторизованого користувача
http://3.66.174.12/api/v1/ordercreate/	створює нове замовлення
http://3.66.174.12/api/v1/orderupdate/<int:pk>/	редагує замовлення відповідного id
http://3.66.174.12//api/v1/account/register/	створює нового користувача
http://3.66.174.12/api/v1/account/login/	надає username та password у форматі base64

Для розгортання front-end компоненти на мові Type-script фреймворку Angular 13.0.0 були вчинені наступні кроки:

1. Настроєний конфігураційний файл Docker [11] (додаток Б);

2. Настроєний файл для запуску автоматичної зборки та розгортання проекту [12] bitbucket-pipelines.yml (додаток Б);

3. Під час команди git push -u origin master проект оновлюється на репозиторії https://alexey_sinitskiy@bitbucket.org/alexey_sinitskiy/prodapp_repo .git та створюється за допомогою Pipeline Bitbucket зборка проекту, упаковується у Docker image та передається сервісу AWS; всі етапі компіляції настроєні у bitbucket-pipelines.yml (додаток Б).

4. Сайт front-end компоненти з браузерним користувацьким інтерфейсом доступний за посиланням http://3.127.23.74/

Таким чином, були опубліковані два сайти.

Доступ до інформаційної системи для онлайн замовлення продукції здійснюється через посилання http://3.127.23.74/.

Доступ до back-end компоненти звичайним клієнтам надається тільки через інтерфейс API; доступ до admin панелі Django надається користувачам з правами SuperUser.

ВИСНОВКИ

У результаті теоретичної частини було виявлено, що для розробки інформаційної системи онлайн замовлення виробничої продукції підходять об’єктно-орієнтовані методи розробки, враховуючи основні етапи життєвого циклу інформаційної системи: аналіз, проектування та реалізація.

Для проектування виробничої системи були розроблені специфікації згідно моделі складної системи, що дозволить представити цільову інформаційну систему у вигляді таких моделей як: логічна, функціональна, компонентна та реалізаційна, побудовані відповідні діаграми.

Розробили технічне завдання для розробки інформаційної системи, в якому описали функціональні вимоги, вимоги до безпеки, вимоги до продуктивності, мобільності та масштабованості.

Розробили політику проектування інтерфейсу через максимізацію зручності використання веб-додатку та мінімізацію складної інформації та об’єктів на сторінках сайту, що будуть відображатися для користувача.

Згідно технічного завдання реалізували інформаційну систему у вигляді клієнт-серверної компонентної архітектури. Такий підхід відповідає вимогам до масштабованості системи.

Данна архітектура включає два сайти front-end компоненти та back-end компоненти, розроблені на фреймворках Django Rest мови програмування Python та Angular 13 на мові програмування Typescript.

Для виконання вимог безпеки розробили інструменти адміністрування сайтів з відповідним розподілом прав доступу.

Реалізували варіанти використання системи для ролі Клієнт.

Сайт back-end компоненти, з доступом звичайним користувачам тільки через API інтерфейс, доступний за посиланням http://3.66.174.12/ та здійснюється через клієнта front-end.

Доступ до інформаційної системи для онлайн замовлення продукції здійснюється через посилання front-end компоненти http://3.127.23.74/.

Реалізований зв’язок між сайтами за допомогою інтерфейсу API, відповідає вимогам здатності до мобільності тобто, здатності взаємодіяти із клієнтами різних середовищ.

Підчас публікації системи були використані сервіси AWS, BitBucket, скрипти автоматичного розгорнення та керування програмами в середовищах з підтримкою контейнеризації Docker.

Успішно протестували взаємозв’язок двох компонентів архітектури, які працюють як одна інтегрована інформаційна система для онлайн замовлення продукції та відповідають вимогам технічного завдання.

Данна система може слугувати прототипом, шаблоном для створення більш складних функціональних систем, використовуючи принципи – зниження складності через компонентну архітектуру – розбивку складних сутностей чи процесів на прості компоненти.

ПЕРЕЛІК ПОСИЛАНЬ

1. Веб застосунок. Вікіпедія. URL: http://uk.wikipedia.org/wiki/Вебзастосунок/ (дата звернення: 01.11.2021).

2. Документація Django 3.2. з перекладом. URL: https://django.fun/docs/django/ru/3.2/
(дата звернення: 01.11.2021).

3. Міжнародний стандарт програмної інженерії SWEBOK. URL: https://www.computer.org/education/bodies-of-knowledge/software-engineering/v3 (дата звернення: 01.11.2021).

4. Круг С. Не примушуйте мене думати. Москва : Ексмо, 2017. 256 с.

5. Лавріщева К.М. Програмна інженерія: підручник. Київ : КПІ, 2008. 319 с.

6. Черткова А.Е. Програмна інженерія. Візуальне моделювання програмних систем. Москва : Юрайт, 2019. 147 с.

7. Фрімен А. Angular для професіоналів. Санкт-Петербург : Пітер, 2018. 799 с.

8. AWS documentation. Official site. URL: https://docs.aws.amazon.com/ (дата звернення: 01.11.2021).

9. Angular 13 documentation. Official site. URL: https://angular.io/start (дата звернення: 01.11.2021).

10. McGaw J. Beginning Django E-Commerce. New York : Apress, 2009. 384 p.

11. Build your Node image. Official site. URL: https://docs.docker.com/language/nodejs/build-images/(дата звернення: 01.11.2021).

12. Configure bitbucket-pipelines. Official site. URL: https://support.atlassian.com/bitbucket-cloud/docs/configure-bitbucket-pipelinesyml/ (дата звернення: 01.11.2021).

13. Django rest framework russian documentation URL:https://ilyachch.gitbook.io/django-rest-framework-russian-documentation /overview/quick-start/serialization (дата звернення: 01.11.2021).

14. Django rest framework. Official site. URL: https://www.django-rest-framework.org/(дата звернення: 01.11.2021).

15. Саммерфілд М. Python на практиці. Москва : ДМК Прес, 2014. 338с.

16. Typescript documentation. Official site. URL: https://www.typescriptlang.org/docs/ (дата звернення: 01.11.2021).

17. Welcome to djoser’s documentation. Official site. URL: https://djoser.readthedocs.io/en/latest/ (дата звернення: 01.11.2021).

Додаток А

Коди API back-end компоненти

Коди для реалізації API back-end компоненти за допомогою Django Rest фреймворку:

src\order\models.py

```
User = settings.AUTH_USER_MODEL

class Order(models.Model):
    status_types = (
        (1, 'Новий'),
        (2, 'У роботі'),
        (3, 'Виконаний'),
        (4, 'Відхилено')
        )
    material_types = (
        (1, 'Деревина'),
        (2, 'Кераміка'),
        (3, 'Камінь'),
        (4, 'Дсп'),
        (5, 'Сталь'),
        (6, 'Скло'),
        )
    status = models.IntegerField(verbose_name = 'Status_types', choices = status_types)
    material = models.IntegerField(verbose_name = 'Material_types', choices = material_types)
    user = models.ForeignKey(User, verbose_name = 'User', on_delete=models.CASCADE)
    dateCreated = models.DateField()
    dateShipped = models.DateField(blank=True, null=True)
    comment = models.CharField(verbose_name = 'Comment', max_length = 100, blank=True)
    phone_number = models.CharField(verbose_name = 'Phone number', max_length = 15, blank=True)
```

src\order\serializers.py

```
from .models import Order
from django.conf import settings
from rest_framework import serializers
from django.contrib.auth import get_user_model

User = settings.AUTH_USER_MODEL

class OrderSerializer(serializers.ModelSerializer):
    user = serializers.HiddenField(default=serializers.CurrentUserDefault())
    class Meta:
        model = Order
        fields = '__all__'

class UserSerializer(serializers.ModelSerializer):
    class Meta:
       model = get_user_model()
       fields = ('id', 'password',  'username', 'is_staff', 'manager_user', 'phone_number')
       #fields = '__all__'

    def create(self, validated_data):
        password = validated_data.pop('password', '')
        user = self.Meta.model(**validated_data)
        user.set_password(password)
        user.save()
        return user
```

src\order\views.py

```
from django.shortcuts import render
from .models import Order
from rest_framework import generics
from .serializers import OrderSerializer, UserSerializer
from .permission import IsOwner, IsManager
from rest_framework.permissions import IsAdminUser, IsAuthenticated, AllowAny
from django.conf import settings
from rest_framework.viewsets import ModelViewSet
from django.contrib.auth import get_user_model

User = get_user_model()

#User = settings.AUTH_USER_MODEL

class OrderList(generics.ListAPIView):
```

```
    serializer_class = OrderSerializer
    permission_classes = ( IsOwner,)

    def get_queryset(self):
        user = self.request.user
        return Order.objects.filter(user=user)

class OrderCreateView(generics.CreateAPIView):
    serializer_class =  OrderSerializer
    permission_classes = ( IsAuthenticated, )

class OrderUpdateView(generics.RetrieveUpdateDestroyAPIView):
    serializer_class = OrderSerializer
    queryset = Order.objects.all()
    permission_classes = (IsAdminUser, )

class UserCreate(generics.CreateAPIView):

    serializer_class = UserSerializer
    permission_classes = (AllowAny, )

class UserLogin(ModelViewSet):

    serializer_class = UserSerializer
    #queryset = User.objects.all()
    def get_queryset(self):
        userpass = self.request.user.password
        usern = self.request.user.username
        return User.objects.filter(password=userpass, username=usern)
```

src\order\permission.py

```
from rest_framework import permissions
class IsOwner(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        return obj.user == request.user
class IsManager(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        return request.user.manager_user
class Login(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        return obj == request.user
```

src\order\urls.py

```
from django.urls import path
        from .views import OrderList, OrderCreateView, OrderUpdateView,
UserCreate, UserLogin

        urlpatterns = [
            path('order/', OrderList.as_view()),
            path('ordercreate/', OrderCreateView.as_view()),
            path('account/register/', UserCreate.as_view()),
            path('account/login/', UserLogin.as_view({'get': 'list'})),
            path('orderupdate/<int:pk>/', OrderUpdateView.as_view()),
        ]
```

src\myuser\models.py

```
from django.db import models

from django.contrib.auth.models import AbstractUser

class CustomUser(AbstractUser):

    manager_user = models.BooleanField(default = False)

    phone_number = models.CharField(max_length = 15)
```

src\apiprod\urls.py

```
from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('api/v1/', include('order.urls')),

    ]
```

Додаток Б

Коди розгортання front-end компоненти

Коди залежностей front-end компоненти для розгортання проекту, конфігурування Apache2 та зборки:

package.json

```
{
 "name": "prodapp",
 "version": "0.0.0",
 "scripts": {
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build",
  "watch": "ng build --watch --configuration development",
  "test": "ng test"
 },
 "private": true,
 "dependencies": {
  "@angular/animations": "~13.0.0",
  "@angular/cdk": "^13.0.2",
  "@angular/common": "~13.0.0",
  "@angular/compiler": "~13.0.0",
  "@angular/core": "~13.0.0",
  "@angular/forms": "~13.0.0",
  "@angular/material": "^13.0.2",
  "@angular/platform-browser": "~13.0.0",
  "@angular/platform-browser-dynamic": "~13.0.0",
  "@angular/router": "~13.0.0",
  "rxjs": "~7.4.0",
  "tslib": "^2.3.1",
  "zone.js": "~0.11.4"
 },
 "devDependencies": {
  "@angular-devkit/build-angular": "~13.0.3",
  "@angular/cli": "~13.0.3",
  "@angular/compiler-cli": "~13.0.0",
  "@types/jasmine": "~3.10.0",
  "@types/node": "^12.11.1",
  "jasmine-core": "~3.10.0",
```

```
"karma": "~6.3.0",
"karma-chrome-launcher": "~3.1.0",
"karma-coverage": "~2.0.3",
"karma-jasmine": "~4.0.0",
"karma-jasmine-html-reporter": "~1.7.0",
"typescript": "~4.4.3"  }}
```

apache2/sites-available/000-default.conf

```
<VirtualHost *:80>
ServerAdmin webmaster@example.com
DocumentRoot /home/ubuntu/django/api/apiprod
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined
Alias /static /home/ubuntu/django/api/static
<Directory /home/ubuntu/django/api/static>
Require all granted
</Directory>
<Directory /home/ubuntu/django/api/apiprod>
<Files wsgi.py>
Require all granted
</Files>
</Directory>
WSGIDaemonProcess apiprod python-path=/home/ubuntu/django/api python-home=/home/ubuntu/django/myprojectenv
WSGIProcessGroup apiprod
WSGIPassAuthorization On
WSGIScriptAlias / /home/ubuntu/django/api/apiprod/wsgi.py
</VirtualHost>
```

settings.py

```
BASE_DIR = Path(__file__).resolve().parent.parent

DEBUG = True

ALLOWED_HOSTS = ['*' ]


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'order',
    'myuser',
    'rest_framework',
    'corsheaders',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    #'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'corsheaders.middleware.CorsMiddleware',
]

CORS_ORIGIN_ALLOW_ALL = True

ROOT_URLCONF = 'apiprod.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
```

```
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'apiprod.wsgi.application'

STATIC_ROOT = os.path.join(BASE_DIR, "static/")
# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
AUTH_USER_MODEL = 'myuser.CustomUser'
# Password validation
# https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {       'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',    },
    {       'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',    },
    {       'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',    },
    {       'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',    },
]
REST_FRAMEWORK = {
    'DEFAULT_RENDERER_CLASSES': [
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.BasicAuthentication',
        #'rest_framework.authentication.SessionAuthentication',
```

```
    ],
}


# Internationalization
# https://docs.djangoproject.com/en/3.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = '/static/'

# Default primary key field type
# https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
~
```

Dockerfile

```
FROM ubuntu:16.04
RUN apt-get -y update
RUN apt-get -y install apache2
copy .  /var/www/html/
CMD ["/usr/sbin/apache2ctl", "-D","FOREGROUND"]
EXPOSE 80
```

bitbucket-pipelines.yml

```
image: node:10.16.3

pipelines:
  branches:
    master:
      - step:
          name: Installation
```

```
    caches:
      - node
    script:
      - npm install
    artifacts:
      - node_modules/** # Save modules for next steps
- step:
    name: Lint
    script:
      - npm run lint
- step:
    name: Build
    script:
      - npm run build:production
      - echo "нужно проверить папку dist"
    artifacts:
      - dist/** # Save build for next steps
- step:
    name: Zip
    script:
      - cp Dockerfile dist/prodapp/
      - apt-get update
      - apt-get install zip
      - cd dist/prodapp/
      - zip -r application.zip *
    artifacts:
      - dist/prodapp/application.zip
- step:
    name: "Deploy to AWS Docker"
    deployment: production
    script:
      - pipe: atlassian/aws-elasticbeanstalk-deploy:0.5.0
        variables:
          AWS_ACCESS_KEY_ID: $AWS_ACCESS_KEY_ID
          AWS_SECRET_ACCESS_KEY: $AWS_SECRET_ACCESS_KEY
          AWS_DEFAULT_REGION: "us-east-2"
          APPLICATION_NAME: "prod_angular_docker"
          ENVIRONMENT_NAME: 'Prodangulardocker-env'
          ZIP_FILE: "dist/prodapp/application.zip"
          S3_BUCKET: "prodangulardocker-elasticbeanstalk-deployment"
```

Додаток В

Коди авторизації та реєстрації у front-end компоненті

Коди авторизації та реєстрації у front-end компоненті за допомогою фреймворку Angular 13**:**

src\app\user\user.service.ts

```
import { Injectable } from '@angular/core';
import { Access } from './access';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root'
})
export class UserService {
  public access: Access = new Access;


  constructor(private http: HttpClient) { }

  private getloginUrl = environment.getlogin;
  private regUserUrl = environment.regUserUrl;

  public getAccess(userAccess: any) {
    let base64 = btoa(userAccess.username + ':'+ userAccess.password)
    let httpOptions = {observe: 'body' as 'body',
                       headers: new HttpHeaders({'Content-Type':  'application/json',
                       'Authorization': 'Basic '+ base64 }),
                                                              }
    this.access.password = base64;
     return this.http.get(this.getloginUrl, httpOptions );   }

   public registrate(userAccess){
    return  this.http.post(this.regUserUrl, {'phone_number':
userAccess.phone_number,
                              'username': userAccess.username,
                              'password': userAccess.password  });
                               }}
```

src\app\user\user.component.ts

```
import { Component, OnInit } from '@angular/core';
import { UserService } from './user.service';
import { Router } from '@angular/router';

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

  constructor(public accessService: UserService,  private router: Router) { }

  ngOnInit() {
  }

  to_order(){
    this.router.navigate([''])

  }
```

src\app\user\register\register.component.ts

```
import { Component, OnInit } from '@angular/core';
import { UserService } from '../user.service';
import { Access } from '../access';
import { Router } from '@angular/router';

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent implements OnInit {
  public userAccess: any;
  public errors: any = [];
  public errorpass: string;

  constructor(public accessService: UserService, private router: Router) { }

  ngOnInit() {
```

```
   this.userAccess = {
     username: '',
     password: '',
     password2: '',
     phone_number: ''
     };
  }

  register() {
    this.errorpass = '';
      if (this.userAccess.password !== this.userAccess.password2
          ||  this.userAccess.password==''){
           this.errorpass = 'Пароль введен невірно';
         }
      else {
        this.accessService.registrate(this.userAccess)
        .subscribe(data => {
          console.log(data);
        this.accessService.access.password = btoa(this.userAccess.username + ':'+
this.userAccess.password);
        this.accessService.access.id = data['id'];
        this.accessService.access.username = data['username'];
        this.accessService.access.is_staff = data['is_staff'];
        this.accessService.access.manager_user = data['manager_user'];
        this.accessService.access.phone_number = data['phone_number'];
        this.accessService.access.access = true;
        this.errors = '';
        this.router.navigate(['user'])

        },
          err => {
            this.errors = err['error'];
            //console.log(this.errors);
            if (typeof this.errors.username !== 'undefined' ){this.errorpass =
this.errors.username } ;
            if (typeof this.errors.password !== 'undefined') {this.errorpass =
this.errors.password } ;
            if (typeof this.errors.email !== 'undefined') {this.errorpass =
this.errors.email } ;
            }    );
        }  }}
```

src\app\user\login\login.component.ts

```
import { Component, OnInit } from '@angular/core';
```

```
import { UserService } from '../user.service';
import { Access } from '../access';
import { Router } from '@angular/router';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  public userAccess: any;
  public errors: any = [];
  public errorsusername: any = [];

  constructor(public accessService: UserService, private router: Router) { }

  ngOnInit() {
    this.userAccess = {
      username: '',
      password: '',
      password2: ''
      };
  }

  login() {
    this.accessService.getAccess({'username': this.userAccess.username, 'password':
this.userAccess.password})
      .subscribe(data => {
        this.accessService.access.id = data[0]['id'];
        this.accessService.access.username = data[0]['username'];
        this.accessService.access.is_staff = data[0]['is_staff'];
        this.accessService.access.manager_user = data[0]['manager_user'];
        this.accessService.access.phone_number = data[0]['phone_number'];
        this.accessService.access.access = true;
        this.errors = '';
        this.router.navigate([''])
        //console.log(data);
      },
        err => {
          this.errors = err['error'];
          this.accessService.access = new Access;
          console.log(this.errors);

        }    );    }}
```

src\app\user\user.component.html

```
<div *ngIf="!this.accessService.access.access">
<h1> ДЛЯ РОБОТИ ІЗ ЗАМОВЛЕННЯМИ ПОТРІБНО <br>
АВТОРИЗУВАТИСЯ АБО ЗАРЕЄСТРУВАТИСЯ НА САЙТІ </h1>
 <div class="card-container"  >
  <div class="card" [routerLink]="['login']" routerLinkActive="active"
[routerLinkActiveOptions]="{exact:true}">
        ЗАЙТИ НА САЙТ
  </div>
  <div class="card" [routerLink]="['register']" routerLinkActive="active">
        ЗАРЕЄСТРУВАТИСЯ НА САЙТІ
  </div>
</div>

<router-outlet></router-outlet>

</div>


<div *ngIf="this.accessService.access.access">
<div class="card-container"  >
     <h1>Привіт, {{ this.accessService.access.username }}<br><br>Ви успішно
авторизовані на сайті</h1>
</div>

  <div class="card-container"  >
  <div class="card" (click)='to_order()'>
        Перейти до замовлень продукції
  </div>
  </div>

</div>
```