НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО”

Факультет Інформатики та обчислювальної техніки
(повне найменування інституту, факультету)

Обчислювальної техніки
(повна назва кафедри)

До захисту допущено
Завідувач кафедри

_____ _____
(підпис) (ініціали, прізвище)

“___”_____2019р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки **6.050103 « Програмна інженерія »** _____
(код і назва)

на тему: Система обробки замовлень в Інтернет магазині

Виконав (-ла): студент (-ка) __4__ курсу, групи __ІП-53__
(шифр групи)

Чекіна Данило Романович _ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник асист. Регіда П. Г. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант нормо контроль д.т.н, проф. Сімоненко В.П. _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМ. ІГОРЯ СІКОРСЬКОГО”

Факультет Інформатики та обчислювальної техніки
(повне найменування інституту, факультету

Обчислювальної техніки

Освітньо-кваліфікаційний рівень **бакалавр**

Напрям підготовки **6.050103 « Програмна інженерія »**

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ _____
(прізвище ініціали) (підпис)

“___”_____2019 р.

ЗАВДАННЯ

на бакалаврський дипломний проект студента

Чекіни Данила Романовича
(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) *Система обробки замовлень в Інтернет магазині*
керівник проекту (роботи) асист. Регіда П. Г. ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “23”04 2019 року №1180-С
1. Термін здачі студентом закінченого проекту (роботи) *20 червня 2019*р.
3. Вихідні дані до проекту (роботи) технічна документація. теоретичні та статистичні дані з обробки Інтернет замовлень
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Опис та аналіз предметної області,аналіз наявних аналогів, дослідження розробки систем управління, конструювання архітектури та розробка системи обробки замовлень в Інтернет магазині, тестування та впровадження системи
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень)
структурна схема системи, узагальнена схема процесу обробки замовлення, блок-схема алгоритму роботи програми.

6. Консультанта проекту (робота), з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В.П.		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту (роботи)	Строк виконання етапів проекту(роботи)	Примітки
1.	Затвердження теми роботи	10.12.2019-15.12.2019	
2.	Вивчення та аналіз завдання	15.12.2019-15.03.2019	
3.	Розробка архітектури та загальної структури системи	15.01 2019-25.01.2019	
4.	Розробка структур окремих підсистем	25.01.2019-5.02.2019	
5.	Програмна реалізація системи	5.02.2019-15.03.2019	
6.	Оформлення пояснювальної	15.03.2019-20.04.2019	
7.	Захист програмного продукту	25.04.2019	
8.	Передзахист	29.04.2019	
9.	Захист	20.05 2019	

Студент-дипломник _____

(підпис)

Керівник роботи _____

(підпис)

Анотація

В даній бакалаврській дипломній роботі реалізовано систему обробки інтернет замовлень, що допомагає витрачати менше часу та ресурсів через розподіл процесу обробки на різні етапи, за які відповідають назначені відповідальні користувачі даної системи.

Програма дозволяє контролювати усі процеси, що буде проходити замовлення на кожному етапі, можливість комунікувати та проводити роботу з клієнтами, та потенційними покупцями. Контролювати роботу працівників, що відповідають за підготовку та збір замовлення.

Програмне забезпечення створене на мові Java у середовищі розробки IntelliJ IDEA клієнтська частина реалізована на JS, HTML, CSS3.

Пояснювальна записка скадається з чотирьох розділів, містить 35 рисунків, 20 таблиць, 5 додатків, та 3 джерел. Загалом складається з 71 сторінок.

Аннотация

В данной бакалаврской дипломной работе реализована система обработки интернет заказов, которая помогает тратить меньше времени и ресурсов из-за распределения процесса обработки на разные этапы, за которые отвечают ответственные люди.

Программа позволяет контролировать все процессы, которые будет проходить заказ на каждом из этапов, возможность коммуницировать и проводить работу с клиентами или потенциальными покупателями. Контролировать работу сотрудников, которые назначены за подготовку и сборку компонентов.

Программное обеспечение создано на языке Java в среде разработке IntelliJ IDEA, клиентская часть реализована на JS, HTML, CSS3.

Пояснительная записка состоит из четырех разделов, вмещает 35 рисунков, 20 таблиц, 5 дополнений, та 3 источников. В целом состоит из 71 страничек.

Annotation

In this work for a Bachelor's Degree, a system for processing Internet orders is implemented. This system helps to spend less time and resources due to the distribution of the processing at different stages. This system has responsible persons for different stages.

The program allows you to control all the processes that the order will pass at each of the stages, the ability to communicate and work with clients or potential customers. Supervise the work of employees who are assigned for the preparation and assembly of components.

The software was created in the Java language in the IntelliJ IDEA development environment, the client part is implemented in JS, HTML, CSS3.

The explanatory note consists of four sections, contains 35 figures, 20 tables, 5 additions, and 3 sources. In general, consists of 71 pages.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1.	А4		Завдання на дипломний проект	1	
2.	А4	ИАЛЦ.467200.001 ВП	Відомість проекту	1	
3.	А4	ИАЛЦ.467200.002 ТЗ	Технічне завдання	4	
4.	А4	ИАЛЦ.467200.003 ПЗ	Пояснювальна записка	71	
5.	А4	ИАЛЦ.467200.004 Д1	Схема алгоритму обробки замовлення	1	
6.	А3	ИАЛЦ.467200.005 Д2	Схема взаємодії класів при обробці замовлення	1	
7.	А3	ИАЛЦ.467200.006 Д3	UML діаграма класів програми	1	
8.	А3	ИАЛЦ.467200.006 Д4	Схема бізнес процесів системи	1	
9.	А4	ИАЛЦ.467200.007 Б1	Лістинг програми	24	

Технічне завдання
до дипломного проекту

на тему: «Система обробки замовлень в Інтернет магазині»

Київ – 2019

ЗМІСТ

1. Найменування та область застосування

Найменування: «Система обробки замовлень в Інтернет магазині».
Область застосування: інтернет-сервіс «Обробка інтернет замовлень».

2. Підстави для розробки

Підставою для розробки системи обробки замовлень в Інтернет магазині є завдання на дипломне проектування, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (НТУУ «КПІ ім. І. Сікорського»)

3. Мета та призначення розробки

Метою даної розробки є побудова системи по обробці інтернет замовлень. Призначення даної розробки полягає в створенні програмного забезпечення яке надає можливість відстежувати стан замовлень, обробляти замовлення з високою швидкістю за рахунок розподілу на різні етапи розробки з призначеними на них відповідальних людей.

4. Джерела розробки

Джерелом розробки є науково-технічна література по інформаційним питанням та публікації в глобальній мережі Інтернет з даних питань.

5. Технічні вимоги

5.1. Вимоги до розроблюваного продукту

Вимоги до розробки:

- реалізація ролей в системі, редагування адміністратором доступу до певних сторінок сервісу для користувачі системи

– редагування інформації контактів, клієнтів
– редагування інформації клієнтів
– робота з замовленнями в різних етапах
– дошка замовлень, вибірка та фільтрація замовлень
– експорт наборів даних, таких як список клієнтів, статистика по замовленням

5.2. Вимоги до програмного забезпечення

Вимоги до програмного забезпечення:

– операційна система MS Windows, Linux, MacOS;
– веб-браузер Google Chrome, Opera, Safari, Firefox, MS Edge, IE 9+.

5.3. Вимоги до апаратного забезпечення

Вимоги до апаратного забезпечення:

- 32-розрядний (x86) або 64-розрядний (x64) процесор із тактовою частотою 1 ГГц або швидший;
- 1 гігабайт (ГБ) RAM;

6. Етапи розробки

Етап	Дата
1. Вивчення літератури за тематикою проекту	24.11.2018
2. Складання і узгодження технічного завдання	25.12.2018
3. Аналіз області застосування та особливостей системи	15.01.2019
4. Розробка загальної архітектури системи	15.03.2019
5. Реалізація програмного забезпечення	05.04.2019
6. Тестування програмного забезпечення	13.04.2019
7. Оформлення документації дипломної роботи	15.04.2019

Пояснювальна записка
до дипломного проекту

на тему: «Система обробки замовлень в Інтернет магазині»

Київ – 2019

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

BPMN – система умовних позначень (нотація) для моделювання бізнес-процесів.

CRM – система управлі́ння відно́синами з клієнтами.

API – программный интерфейс приложения.

БД – база даних.

UML – графічна мова для візуалізації.

ВСТУП

У сучасному світі Інтернет задіяний практично у всіх сферах діяльності людини. Мережа надає безліч можливостей, але необхідно вміти ними користуватися.

Інтернет-магазини все більше і більше завойовують популярність серед користувачів. Перевага перед звичайними магазинами полягає в тому, що онлайн магазини ведуть торгівлю товарами у мережі, що надає можливість не виходячи з дому отримати усі необхідні товари або послуги.

Усі покупки в інтернет-магазинах можна здійснювати з дому, з транспорту або ж на відпочинку. Потрібен лише доступ в інтернет і не існуватиме ніяких обмежень. Онлайн магазини стали невід'ємною частиною життя сучасної людини.

Саме через таку чималу популярність інтернет-магазинів і виникає необхідність спрощення та автоматизування процесу його роботи. Адже вже зараз кількість користувачів даного виду послуг чимала і продовжує стрімко зростати. Обробка такої великої кількості замовлень потребує багато сил, часу та людських ресурсів. Тому дуже гостро встає питання спрощення та автоматизації процесу обробки замовлень в інтернет-магазинах. Завданням даної роботи є розробка системи обробки замовлень інтернет-магазину, яка надасть можливість налагодити процес створення, підготовки та доставки товару онлайн магазину.

Результатом роботи є інтегрована система, яка може бути використана для будь-якого інтернет-магазину, де необхідно налагодити та спростити механізм підготовки товару для продажу, обробки замовлення та доставки до клієнта, для того щоб зробити цей процес більш прозорим, контрольованим та зменшити кількість витраченого часу та людських ресурсів.

РОЗДІЛ 1
АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Для того щоб розвивати інтернет-магазин, долучати нових клієнтів та збільшувати кількість проданих товарів необхідно налагодити усі процеси роботи з клієнтами та між працівниками сервісу. Цю проблему вирішують системи управління відносинами з клієнтами.

Управління відносинами з клієнтами (англ. CRM – Customer relationship management) — поняття, що охоплює концепції, котрі використовуються компаніями для управління взаємовідносинами зі споживачами, включаючи збір, зберігання й аналіз інформації про споживачів, постачальників, партнерів та інформації про взаємовідносини з ними.

Сучасна CRM направлена на вивчення ринку і конкретних потреб клієнтів. На основі цих знань розробляються нові товари або послуги і таким чином компанія досягає поставлених цілей і покращує свій фінансовий показник.

Існує три CRM-підходи, кожен з яких може бути реалізованим окремо від інших:

– Оперативний — автоматизація споживчих бізнес-процесів, що допомагає персоналу з роботи з клієнтами виконувати свої функції.

– Співробітницький — програма взаємодіє зі споживачами без участі персоналу з роботи з клієнтами.

– Аналітичний — аналіз інформації про споживачів із різноманітними цілями.

CRM-рішення для інтернет-магазину об’єднує та надає можливість керувати внутрішніми та зовнішніми процесами магазину:

– Прийняття та оформлення замовлення.

– Автоматична реєстрація нових клієнтів.

– Управління базою даних клієнтів.

– Інтеграція з системи бухгалтерського обліку, обліку товарів на складі, перевірки вартості, формування рахунку.

– Глибока аналітика показників роботи інтернет-магазину та його менеджерів.

– Управління підготовкою готових товарів.

– Управління службою доставки.

1.2 Змістовний опис та аналіз предметної області

Структура роботи інтернет-магазину є доволі простою і при певному підході не викликає ніяких труднощів.

Схема роботи наступна:

– Покупець потрапляє на сайт інтернет-магазину.

– Покупець обирає товар та оформлює замовлення, вказавши усю необхідну контактну інформацію та подробиці доставки.

– Інтернет-магазин підтверджує замовлення та відправляє його в службу підготовки товарів.

– Коли товар повністю готовий, кур’єрська служба або пошта доставляють товар.

– Покупець оплачує товар та послуги доставки.

Якщо покупець здійснює передплату, то у даній схемі можна поміняти місцями пункти доставки та оплати.

Принцип роботи інтернет-магазину зводиться до того, щоб покупець, потрапивши на цільову сторінку товару, якомога швидше прийняв рішення на користь покупки даного товару.

Схема придбання товару покупцем:

– Покупець потрапляє на сайт інтернет-магазину.

– Бачить цікавий для нього товар або товари.

– Додає їх до кошика.

– Покупець додатково щось обирає і заповнює кошик.

– Покупець оформлює замовлення.

– Співробітник інтернет-магазину зв’язується з клієнтом і підтверджує замовлення, потім передає його в службу обробки замовлень.

Наступний етап – обробка замовлення самим інтернет-магазином. Підтвердження замовлення магазином є важливим і обов’язковим этапом, так як покупець після оформлення замовлення очікує підтвердження, що товар є в наявності, а також остаточного узгодження дати та часу доставки.

Варіанти способів підтвердження замовлень інтернет-магазином:

– Телефонний дзвінок – найбільш ефективний спосіб спілкування з клієнтами. По телефону завжди можливо підтвердити замовлення, обрати день та час доставки. Також більшість покупців очікують телефонний дзвінок.

– E-mail повідомлення – відправка повідомлення на електронну пошту покупця з підтвердженням замовлення, списком обраних товарів та з іншими параметрами замовлення.

– Sms-повідомлення – можуть бути використані як окремий, так і додатковий крок підтвердження замовлення.

Далі виконується підготовка, або за потреби виготовлення замовлених товарів. На цьому етапі необхідно правильно організувати усі процеси та налагодити виробництво. За необхідності розбити усі задачі на підзадачі та призначити відповідальних осіб. Лише у випадку грамотної організації можливо буде уникнути чималої кількості проблем, помилок, перешкод та

досягти найбільшої можливої продуктивності та якості обслуговування клієнтів.

1.3 Аналіз успішних IT-проектів

На сьогодні, в інтернеті можливо знайти та використати величезну кількість різних CRM – систем, що мають подібний основний набір функцій та можливостей:

До основних функцій CRM – систем належать:

– Облік клієнтів – можливість зберігання усіх клієнтів(компаній, та контактних осіб) в єдину базу даних CRM – системи. Фільтрація, сортування списку в залежності від типу клієнта. Для кожного створюється своя картка з усіма контактними даними, та інформацією про взаємодію з цим клієнтом.

– Управління продажами – функція, що надає змогу зберігати в базу даних потенційні угоди та вести їх по воронці продаж, зберігая при цьому інформацію про кожний єтап. Кожна угода прив'язуеться до певного клієнта, та зберігається в його картці.

– Розмежування прав надає змогу керівництву контролювати доступ до контактів та їх замовлень. Також керівництво має можливість назначати відповідальних та створювати певні задачі для підлеглим.

– Аналітика. Функція аналітики являє собою набір звітів таких як: статистика продаж, воронка продаж, статистика по виконаним задачам.

За змістом і джерелами формування звітність буває:

– Статистична.

– Фінансова.

– Податкова.

– Спеціальна.

– Внутрішньогосподарська.

Такі системи можуть різнитися специфічним додатковим функціоналом для орієнтованих на це клієнтів, як:

– Автоматизація обробки замовлень. Усі замовлення потравляють в єдиний список після чого розподіляються між операторами.

– Внутрішній мессенджер, що надає можливість підтримувати контакт с клієнтами не покидаючи CRM.

– Обмін різноманітними файлами між клієнтом та менеджером, або між менеджерами.

– Інтеграція служб доставки.

1.3.1 «ReatailCRM»

RetailCRM – система, спеціально розроблена для інтернет торговлі. Дозволяє автоматизувати продаж інтернет замовлень, покращює обслуговування клієнтів, замовників. Надає можливість аналізувати дані по продажу товарів, для того щоб мати оцінку діяльності компанії. Має потужний інструмент маркетингу для максимально ефективного росту одиниць продажу. Сервіс має демо – версію, що надає можливість опрацьовувати до 300 заказів на місяць без обмежень функціоналу.

Дану систему можно використовувати для будь яких інтернет магазинів. Має зручний та легкий дизайн для використання, гарну технічну підтримку, та багато функцій.

Ключові особливості:

– Має єдине вікно для обробки заказів.

– Відкритий API, що дає можливість настроювати двосторонню інтеграцію зі службами доставки.

– Клієнтські дані збираються з різних систем та сайтів в єдиній картці.

– Зручне відображення аналітики в одному вікні.

– Обслуговування: технічна підтримка, адміністрування, консультування.

– Впровадження нового функціонала в існуючий.

1.3.2 «Бітрікс 24»

Бітрікс 24 – система, що підійде для будь-якого інтернет – магазину. Великий перелік з функціоналу для роботи компанії, можливість керувати як інтернет – магазином так і внутрішніми процесами компанії. Один з мінусів даного сервісу – повільна робота технічної підтримки. Має безкоштовний(для 12 співробітників) пакет та платні версії.

Ключові особливості:

– Всі замовлення та повідомлення, що будуть отриманні з сайту, пошти, або месенджерів, автоматично заносяться у систему, та потрапляють в роботу менеджерам.

– Система самостійно відправляє листи, sms, голосові повідомлення, показують рекламу, назначають різноманітні завдання для менеджерів.

– Доступний календар, планування, регістрація робочого часу працівників, телефонія, інтеграція з 1С, дошка канбан для постановки задач та контролю переміщення замовлень по ній.

– Хмарне сховище об'ємом до 100 ГБ.

– Можливість використання канбан(дошка замовлень).

– Можливість переглядати всю історію події з клієнтами, коментарі, що були прикріплені на різних єтапах угоди.

– Можливість спілкуватись(онлайн-чат) з різними представниками системи в обраній картці, при тому нові учасники чату не зможуть вносити зміни до угоди.

1.3.3 «AmoCRM»

AmoCRM – системне програмне заюезпечення для компанії, що надає можливість автоматизувати взаємодію з клієнтами. Використовуеється для того щоб збільшити рівень продажу, та для покращення обслуговування клієнтів шляхом зберігання детальної інформації про клієнтів, та історії взаємовідносин з ними.

Ключові особливості системи:

– Простота в управлінні та використанні, що надає можливість швидко навчати співробітників.

– Автоматична фіксація заявок по різним каналам, таким як телфон, пошта, веб-форми, соціальні мережі. Данний функціонал надає можливість не втратити потенційного клієнта.

– Інтеграція з телефонією та поштою.

– Можливість інтеграції сайту та інтеграція системі Google.Analytics.

– Можливість використовувати веб версію без додаткових інсталяцій.

1.4 Аналіз вимог до програмного забезпечення.

Для того щоб визначити вимоги до програмного забезпечення, потрібно визначити ролі користувачів та функцій, що вони виконуть у системі. Отже, типи користувачів у даному програмному забезпеченні:

- Адміністратор.
- Менеджер.
- Відповідальний за підготовку компонентів.
- Відповідальний за збір замовлення.

Адміністратор має можливість продивлятись список клієнтів, редагувати будь-яку інформацію пов’язану з цим клієнтом, також має доступ до списку замовлень клієнтів та історії замволеня. Може редагуати інформацію про замовлення, видаляти клієнта з бази даних сервісу. Адміністратор має доступ до списку контактів, може змінювати контактну інформацію клієнтів, може продивлятись історію побудови індивідуального замовленя. Має доступ до дошки замовлень, що використовується для контролю статусу замовлень у системі. Також адміністратор працює зі списком зареєстрованих користувачів у системі, що виконують работу на різних життєвих етапах замовлення, змінювати їх рівень доступу до даних, редагувати індивідуальну інформацію, створювати нових користувачів системи. Може призначати відповідального за підготовку компонентів та відповідального за збір замволення. Має права до сторінки «Конфігурації», на якій може редагувати конфігурації конструктора для одного з типу замовлень, та до сторінки «Компоненти» на якій можно змінювати інформацію про компоненти, що будуть використані у замовленні.

Менеджер має доступ до клієнтської та контактної сторінки, можливість змінювати/редагувати дані контакта/клієнта. Має доступ до дошки зі статусами замовлення. Може змінювати та назначати користувача відповідальним за підготовку компонентів та відповідального за збір замовлення.

Відповідальний за підготовку компонентів, має доступ до сторінки «Замовлення», на якій користувач має можливість перегляди список наявних замовлень у системі, що потребують підготовки компонентів для подальшої роботи з цим замовленням, до яких данний користувач був призначенний. Має можливість переглядати список компонентів кожного замовлення. Також відповідальний за підготовку компонентів може спостерігати за подальшим статусом замовлення, що було на його етапі.

Загалом система повинна мати такий функціонал:

– Аутентифікація користувача.
– Реєстрація нових користувачів.
– Реалізація ролей в системі, редагування адміністратором доступу до певних сторінок сервісу для користувачі системи.
– Редагування інформації контактів.
– Редагування інформації клієнтів.
– Дошка замовлень, вибірка та фільтрація замовлень.
– Реалізація різних життєвих етапів, що проходить замовлення. Взаємодія користувачів системи.
– Зберігання інформації про контакти, деталі замовлення, реалізація збереження фото звітів.
– Підтвердження готовності замовлення.

1.4.1 Розробка функціональних вимог

В системі передбачено наступні варіанти використання, що розлянуті в таблицях 1.1 – 1.16:

Таблиця 1.1 Варіант використання UC001

Назва	Авторизація користувача.
Опис	Користувач має можливість авторизуватись в системі.
Учасники	Не авторизований користувач.
Передумови	
Постумови	Користувач проходить авторизацію у застосунку.
Основний сценарій	Система демонструє вікно авторизації, яке містить адрес електронної пошти та пароль. Користувач заповнює поля вводу. Система демонструє кнопку “Увійти”. Користувач натискає кнопку “Увійти”. Система авторизує користувача.
Розширення сценаріїв	Система виявляє, що дані, внесені адміністратором, не є валідними. Система демонструє користувачеві повідомлення про відповідну помилку.

Таблиця 1.2 Варіант використання UC002

Назва	Реєстрація нового користувача у системі.
Опис	Адміністратор має можливість реєструвати нового користувача у системі, та призначати роль.
Учасники	Адміністратор.
Передумови	Користувач авторизований як адміністратор.
Постумови	В системі створюється новий користувач, якому призначається роль, дані зберігаються у БД.
Основний сценарій	Адміністратор заповнює поля вводу(інформацію про нового користувача). Адміністратор призначає роль для майбутнього користувача; Система проводить первірку введених данів. Система додає набір даних до БД. Система відправляє повідовмлення на пошту нового користувача, що містить пароль на почту майбутньому користувачу, та посилання на нову сторінку для підтвердження аккаунта. Користувач підтверджує регістрацію, після чого даний користувач має можливість аутентифікувати себе у системі.

Таблиця 1.3 Варіант використання UC003

Назва	Створення нового контакту у системі.
Опис	Адміністратор/менеджер має можливість створити новий контакт у системі, заповнити іннформацію та зберігти дані у БД.
Учасники	Адміністратор/менеджер.
Передумови	Користувач авторизований як адміністратор/менеджер.
Постумови	Новий контак створено та збережено у БД.
Основний сценарій	На сторінці «Контакти» адміністратор/менеджер натискає кнопку «Створити новий контакт». Система створює новий контакт, та зберігає дані у БД, та відображає новий рядок у таблиці «Контакти». Система відображає новий запис в таблиці контактів. Адміністратор/менеджер заповнює інформацію про контакт у новому записі таблиці. Оноволені дані про контакт автоматично зберігаються у БД.

Таблиця 1.4 Варіант використання UC004

Назва	Редагування конфігурації для одного з типу замволення.
Опис	Користувач авторизований як адміністратор має можливість редагувати конфігурації, що використовується для формування конструктора для одного з типу замволення.
Учасники	Адміністратор.

Продовження таблиці 1.4 Варіант використання UC004

Передумови	Користувач авторизований як адміністратор.
Постумови	Інформація про клієнта оновлена та збережена у БД.
Основний сценарій	На сторінці «Конфігурації» користувач подвійним натиском активує режим редагування в клітинці. Користувач редагує дані конфігурації. Оноволені дані зберігаються у БД.

Таблиця 1.5 Варіант використання UC005

Назва	Призначеня відповідальних за відповідні етапи обробки замовлення.
Опис	Адміністратор/менеджер має можливість призначати відповідальних за підготовку та збір замовлення на сторінці «Замволення», вкладка «Оплачені замовлення».
Учасники	Адміністратор/менеджер.
Передумови	Користувач авторизований як адміністратор/менеджер, оплачене замовлення, що збережено у БД.
Постумови	Відповідному замовленню призначений відповідальний за підготовку компонентів та відповідальний за збір замовленя. Замовлення автоматично змінило свій статус на «На підготовці» та видалилось зі сторінки «Замовлення». Дані збережені у БД.

Продовження таблиці 1.5 – Варіант використання UC005

Основний сценарій	Авторизований користувач обирає замовлення на сторінці «Замовлення», вкладка «Оплачені замовлення» та натискає кнопку «Призначити відповідальних». Система відображає модальне вікно; Авторизований користувач обирає відповідального за підготовку компонентів зі списку співробітників, що мають роль «Відповідльний за підготовку»; Дані про відповідального за підготовку збережені у БД. Авторизований користувач обирає відповідального за збір замовлення зі списку співробітників, що мають роль «Відповідальний за збір»; Дані про відповідального за збір замовлення збережені у БД. Замовлення видаляється с зданої сторінки та змінює свій статус на «На підготовці»

Таблиця 1.6 Варіант використання UC006

Назва	Редагування статусу роботи на різних єтапах обробки замовлення.
Опис	Авторизований користувач, що призначений відповідальни за підготовку компонентів замовлення/збір замовлення може редагувати свій статус роботи.
Учасники	Авторизований користувач.
Передумови	Авторизований користувач, що призначений відповідальним за підготовку компонентів/збір замовлення.
Постумови	Статус роботи на певному єтапі змінено, дані збережені у БД.

Продовження таблиці 1.6 - Варіант використання UC006

Основний сценарій	На сторінці «Підготовка компонентів»/«Збір замовлення» користувач натискає кнопку «Змінити статус роботи». Система відображає випадаючий список зі списком статусі, що користувач може обрати. Користувач обирає необхідний статус. Новий статус роботи зберігається у БД.

Таблиця 1.7 Варіант використання UC007

Назва	Редагування компонентів на сторінці «Компоненти»
Опис	Адміністратор/менеджер має можливість редагування компонентів на сторінці «Компоненти».
Учасники	Адміністратор/менеджер.
Передумови	Користувач авторизований як адміністратор/менеджер.
Постумови	Інформармація про компоненті оновленна та збережена у БД.
Основний сценарій	На сторінці «Компоненти» користувач змінює інформацію копоненту або матеріалу, що використовується для побудови конструктора одного з типів замовлення. Система зберігає дані у БД. Система відображає нові дані в записі таблиці.

Таблиця 1.8 Варіант використання UC008

Назва	Підтвердження замовлення.
Опис	Адміністратор/менеджер має можливість здійснити перевірку стану замовлення, та здійснити підтверження для його подальшого транспортування.
Учасники	Адміністратор/менеджер.
Передумови	Користувач авторизований як адміністратор/менеджер.
Постумови	Замовлення підтвердженно, статус замовлення змінено. Оновденні дані збережені у БД.
Основний сценарій	На сторінці «Замовлення», вкладка «Підтвердження», користувач переглядає фото-звіт про замовлення. Користувач натискає кнопку підтвердити. Замовлення змінює свій статус на «Підтверженний» Дані зберігаються у БД. Замовлення видаляється с даної вкладки.

Таблиця 1.9 Варіант використання UC009

Назва	Перегляд історії збірок замовлення клієнта
Опис	Адміністратор/менеджер має можливість здійснити перевірку історії збірок замовлення, що клієнт створював для себе.
Учасники	Адміністратор/менеджер.

Продовження таблиці 1.9 – Варіанти використання UC009

Передумови	Користувач авторизований як адміністратор/менеджер.
Постумови	Система відображає історію(список збірок) замовлення.
Основний сценарій	На сторінці «Контакти» авторизований користувач натискає на кнопку «Перегляд історії» в записі, що відповідає певному контакту. На сторінці з'являється нова таблиця з записами про збірки замовлень, що користувач будував для себе.

Таблиця 1.10 Варіант використання UC010

Назва	Редагування інформаціїї про контакт.
Опис	Адміністратор/менеджер має можливість здійснити редагування контактної інформації про контакт/клієнта.
Учасники	Адміністратор/менеджер.
Передумови	Користувач авторизований як адміністратор/менеджер.
Постумови	Контакта інформація клієнта змінена, дані збережені у БД. Система відображає оновлені дані.
Основний сценарій	На сторінці «Контакти» авторизований користувач обирає відповідний запис в таблиці з контактами, та подвійним кліком активує режим редагування. Користувач заносить нову контактну інформацію. Оновлені дані про контакт збережені у БД. Система відображає оновлені дані.

Т

Таблиця 1.11 Варіант використання UC011

Назва	Редагування інформаціїї про користувача системи.
Опис	Адміністратор має можливість здійснити редагування контактної інформації користувача.
Учасники	Адміністратор.
Передумови	Користувач авторизований як адміністратор.
Постумови	Інформація користувача системи оновлена, дані збережені у БД. Система відображає оноволені дані.
Основний сценарій	На сторінці «Користувачі системи» авторизований користувач обирає відповідний запис в таблиці, та подвійним кліком активує режим редагування. Користувач заносить нову інформацію про користувача. Оновлена інформація користувача системи збережена у БД. Система відображає оновлені дані.

Таблиця 1.12 Варіант використання UC012

Назва	Перегляд списку замовлень та їх відповідних копомнентів, що потребують підготовки компонентів.
Опис	Відповідальний за підготовку компонентів має модливість перегляду замовлень та їх компонентів, на які він був призначений.
Учасники	Відповідальний за підготовку.
Передумови	Користувач авторизований як «Відповідальний за підготовку».

Продовження таблиці 1.12 – Варіанти використання UC012

Постумови	Система відображає список замовлень та їх компоненти, на які був призначений даний користувач відповідальним.
Основний сценарій	На сторінці «Замовлення» авторизований користувач обирає відповідне замовлення, та натискає на кнопку «Компоненти». Система відображає модальне вікно, в якому міститься інформація про всі компоненти, що потрібні для підготовки даного замовлення.

Таблиця 1.13 Варіант використання UC013

Назва	Перегляд історії замовлень, що були на етапі підготовки для відповідального за підготовку.
Опис	Відповідальний за підготовку компонентів має можливість перегляду історії замовлення, на які він був призначений та закінчив етап підготовки.
Учасники	Відповідальний за підготовку компонентів.
Передумови	Користувач авторизований як «Відповідальний за підготовку».
Постумови	Система відображає список замовлень(історію) та їх компонентів, що знаходяться на інших етапах або завершені.
Основний сценарій	На сторінці «Історія», система відображає історію замволень, на які був призначений даний користувач, завершені або, що знаходяться на іншому етапі обробки.

Таблиця 1.14 Варіант використання UC014

Назва	Перегля замовлень та їх властивостей на етапі збірки замовлення.
Опис	Відповідальний за збірку замовлення, має можливість перегляду інформації про замовлення, на які він був призначенний відповідальним.
Учасники	Відповідальний за збірку.
Передумови	Користувач авторизований як «Відповідальний за збірку».
Постумови	Система відображає список замовлень, їх компонентів та деталей замволення. Також система відображає різний функціонал взаємодії з замовленням на даному етапі.
Основний сценарій	Система відображає сторінку «Замовлення в роботі». Авторизований користувач має змогу перегляду списку замовлень, на які він був призначенний відповідальним за збірку.

Таблиця 1.15 Варіант використання UC015

Назва	Додавання фото-звіту про зібране замовлення.
Опис	Відповідальний за збірку замовлення, має можливість додавати фото, в яких адміністратор буде мати можливість переглянути стан замволення, та підтвердити готовність.
Учасники	Відповідальний за збірку.

Продовження таблиці 1.15 – Варіанти використання UC015

Передумови	Користувач авторизований як «Відповідальний за збірку».
Постумови	Користувач здійснює посилання файлів(фото) на сервер. Сервер зберігає дані. Дані про файл зберігаються у БД.
Основний сценарій	Система відображає сторінку «Замовлення в роботі». Користувач натискає на кнопку «Додати фото» в відповідному записі до замовлення. Система відображає нове вікно, в якому користувач може обрати фото, та здійснити загрузку їх на сервер. Система зберігає фото на сервері. Система відображає збережені фото на сторінці. Система змінює статус замовлення на «Потребує підтвердження»

Таблиця 1.16 Варіант використання UC016

Назва	Відображення контактних данів про клієнта, для подальшого транспортування замовлення.
Опис	Відповідальний за збірку замовлення, має можливість перегляду даних клієнтів, замовлення яких пройшло перевірку на готовність, та має статус «Перевірено».
Учасники	Відповідальний за збірку.
Передумови	Користувач авторизований як «Відповідальний за збірку».
Постумови	Система відображає дані клієнта.

Продовження таблиці 1.16 – Варіанти використання UC016

Основний сценарій	Система відображає сторінку «Замовлення в роботі». Користувач натискає на кнопку «Отримати дані клієнта» в відповідному записі до замовлення. Система відображає нове вікно, в якому користувач може переглянути дані клієнта.

ВИСНОВКИ ДО РОЗДІЛУ 1

Налагодження процесів роботи з клієнтами та удосконалення процесу обробки замовлень є важливою частиною функціонування сучасних інтернет-магазинів. Сучасні тенденції розвитку даної галузі обумовлюють необхідність створення нових рішень, що здатні спростити та налагодити усі внутрішні процеси роботи інтернет-магазинів.

Існує чимало готових рішень даної проблеми, що мають ряд переваг та недоліків. Жодна з них не надає можливості налагодити процес поетапного виготовлення продукту для подальшого продажу, що необхідно для вирішення поставленої задачі.

Розробка вище описаної системи є актуальним рішення серед наявних аналогів. Готова система має підтримувати можливість функціонування дошки наявних замовлень, вибірку та фільтрацію замовлень, зберігання та адміністрування базою даних клієнтів інтернет-магазину, реєстрацію нових клієнтів та партнерську програму для постійних клієнтів.

РОЗДІЛ 2
МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Для створення якісного програмного забезпечення необхідно попереднє детальне моделювання процесів і архітектури програмного забезпечення.

Моделювання програмного забезпечення - це процес вирішення задач та планування для створення програмного рішення. Після того як мета і специфікація програми описані, розробник створить дизайн проекту, або найме дизайнера для розробки плану рішення. В дизайн включаються як описи низькорівневих компонентів, алгоритмів, так і огляд архітектури.

Для проектування бізнес-процесів було обрано методологію створення діаграм BPMN 2.0. Схему процесів діяльності наведено у додатку А.

Схема містить основні процеси системи: аутентифікацію, реєстрацію, ведення списку користувачів та покупців, веденні історії замовлень, робота з дошкою статусу замовлень.

Схема основної архітектури системи наведена на рис. 2.1.

Рис. 0.1

2.1.1 Аутентифікація та реєстрація користувача

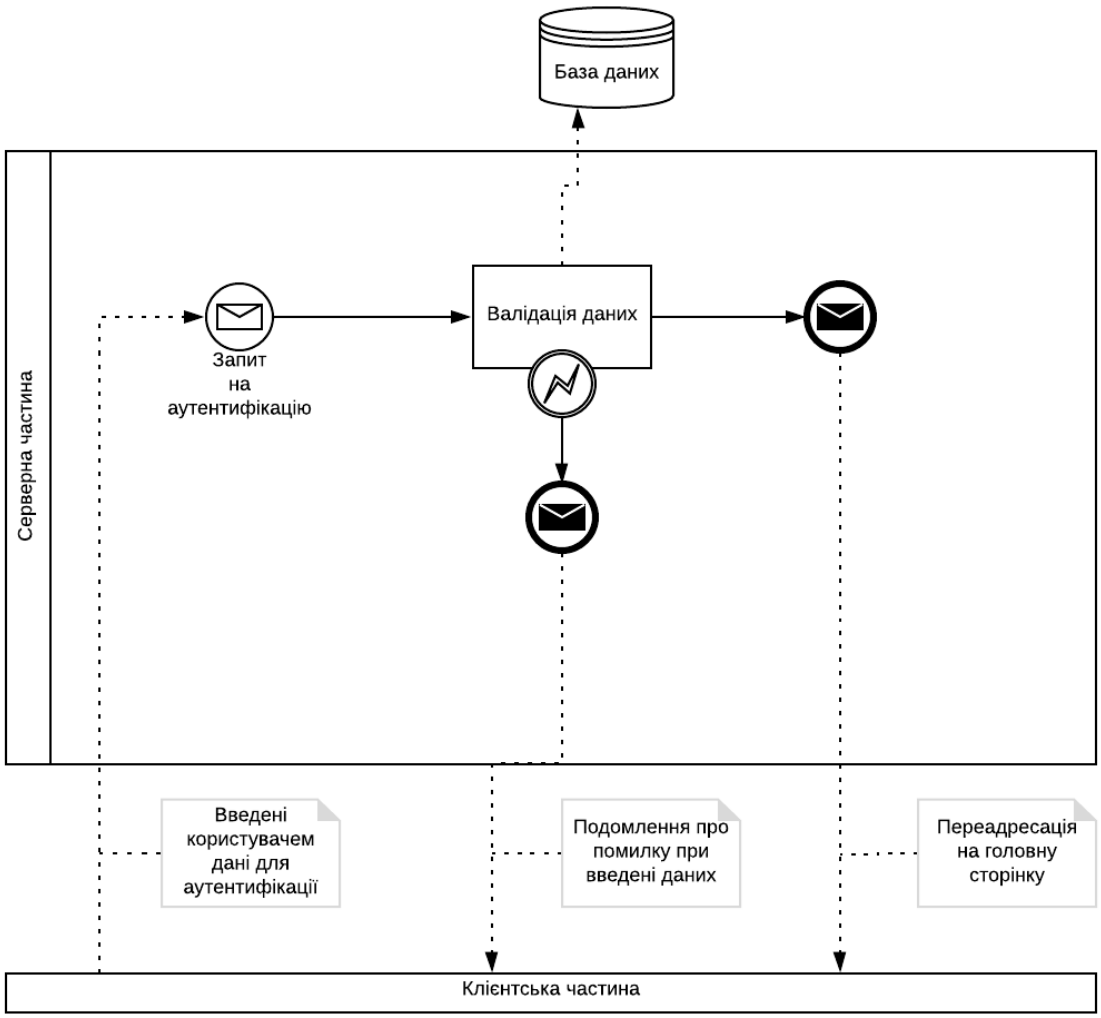


Рис. 0.2 Схема аутентифікація користувача

Послідовний опис процесу аутентифікації користувача зображеного на рис. 2.2:

– Система відображає сторінку, що має панель для аутентифікації користувача.

– Користувач вводить свої дані для аутентифікації.

– Система виконує пошук користувача у БД. Якщо користувача знайдено та його пароль співпадає з введеним паролем, користувач успішно аутентіфікований у систему;

– Система переадресовує користувача на головну сторінку.

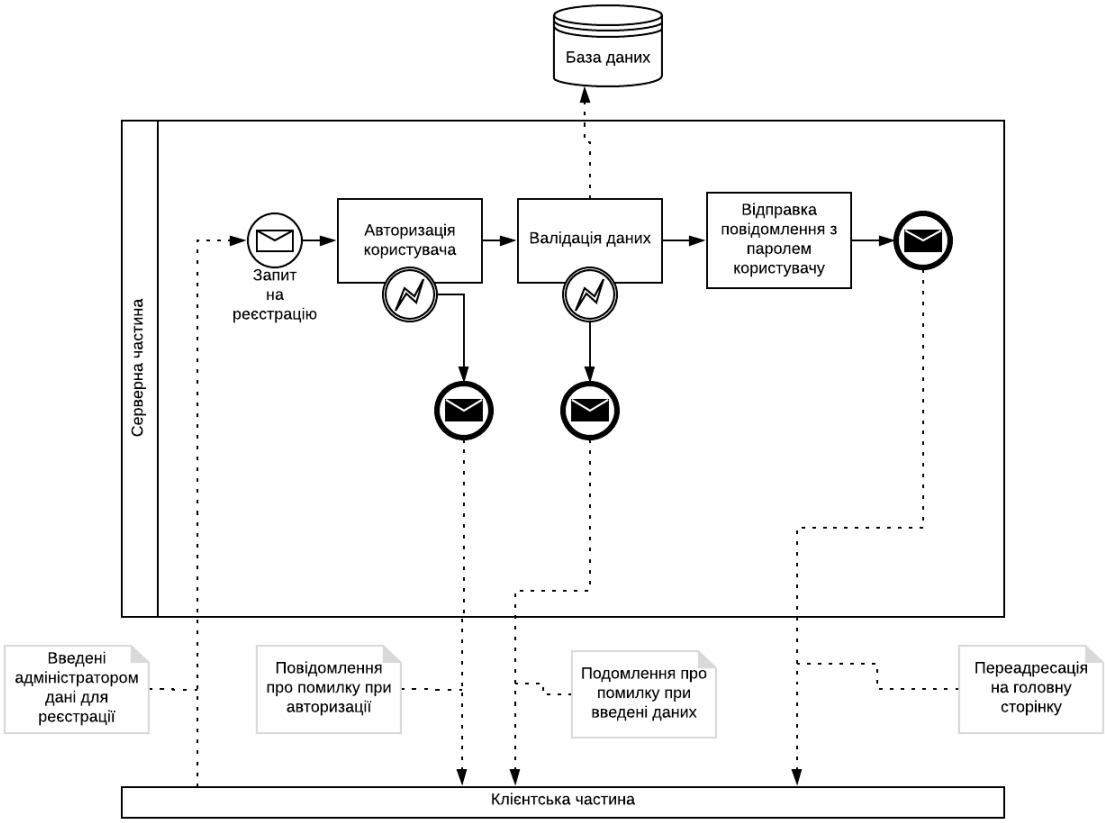


Рис. 0.3 Схема процесу реєстрації нового користувача

Послідовний опис процесу реєстрації нового користувача, зображеного на рис. 2.3:

– Система відображає сторінку, ща має панель для реєстрації нового користувача.

– Адміністратор вводить дані нового користувача.

– Сервер виконує процес авторизації адміністратора.

– Якщо при авторизації виникає помилка, сервер відправляє повідомлення про помилку авторизації.

– Система виконує пошук користувача з веденими даними у системі, та перевіряє нові введені дані на унікальність, після успішної перевірки реєструє нового користувача та зберігає дані у БД.

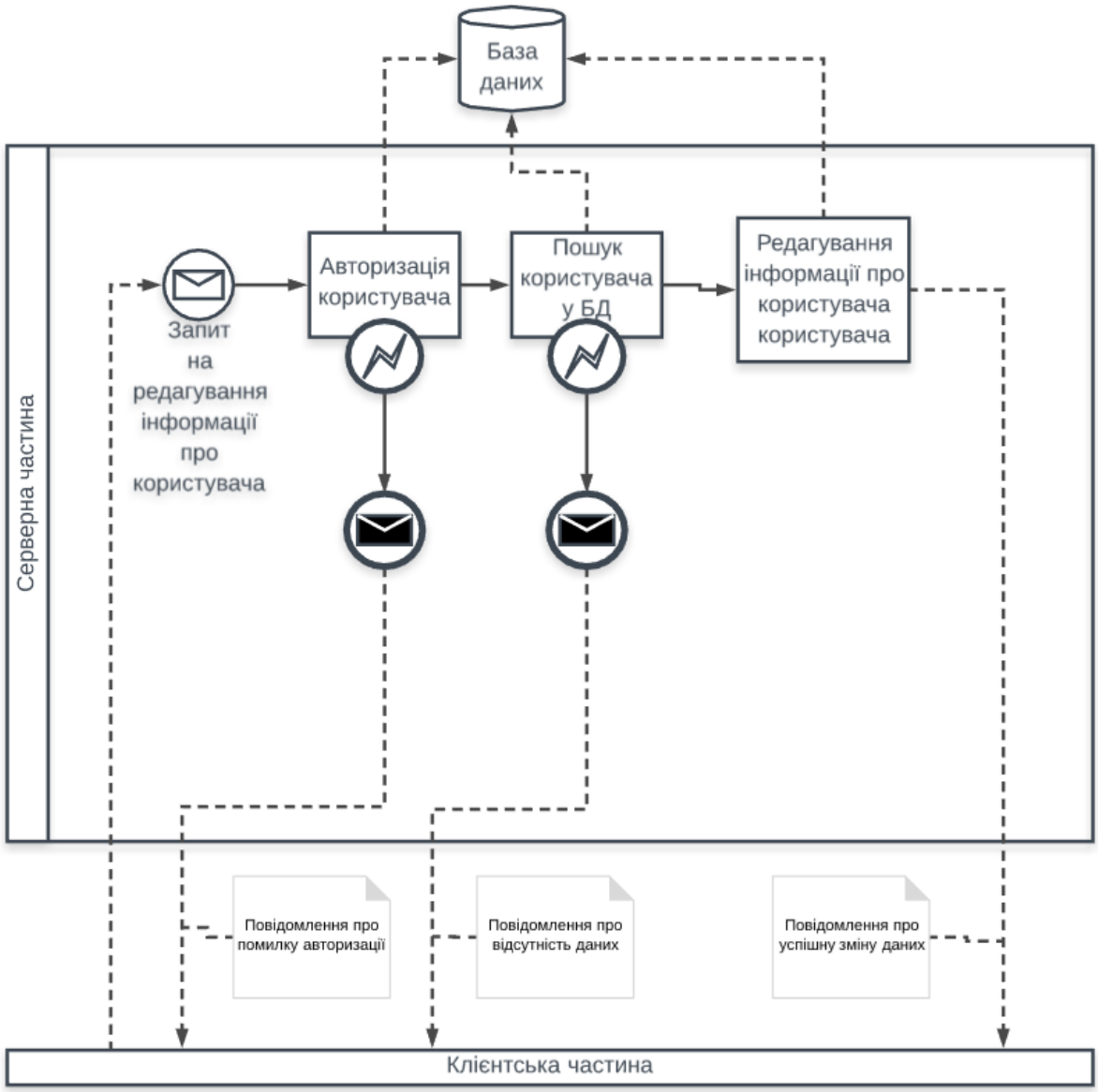


Рис. 0.4 Схема процесу редагування інформації користувача системи

Послідовний опис процесу редагування/зміни інформацію про користувача системи, зображеного на рис. 2.4:

– Клієнт надсилає запит на зміну інформації про користувача системи.

– Сервер проводить авторизацію користувача.

– Якщо при авторизації виникає помилка, сервер відправляє повідомлення про помилку авторизації.

– Сервер вибирає користувача по крітерію пошуку(в данному віпадку – id).

– Якщо при запиті до БД дані не знайдено, сервер відправляє повідомлення про помилку пошуку даних на клієнт.

– Сервер знаходить користувача

– Сервер змінює інформацію про користувача системи та зберігає оновлені дані у БД

2.1.2 Робота з замовленнями

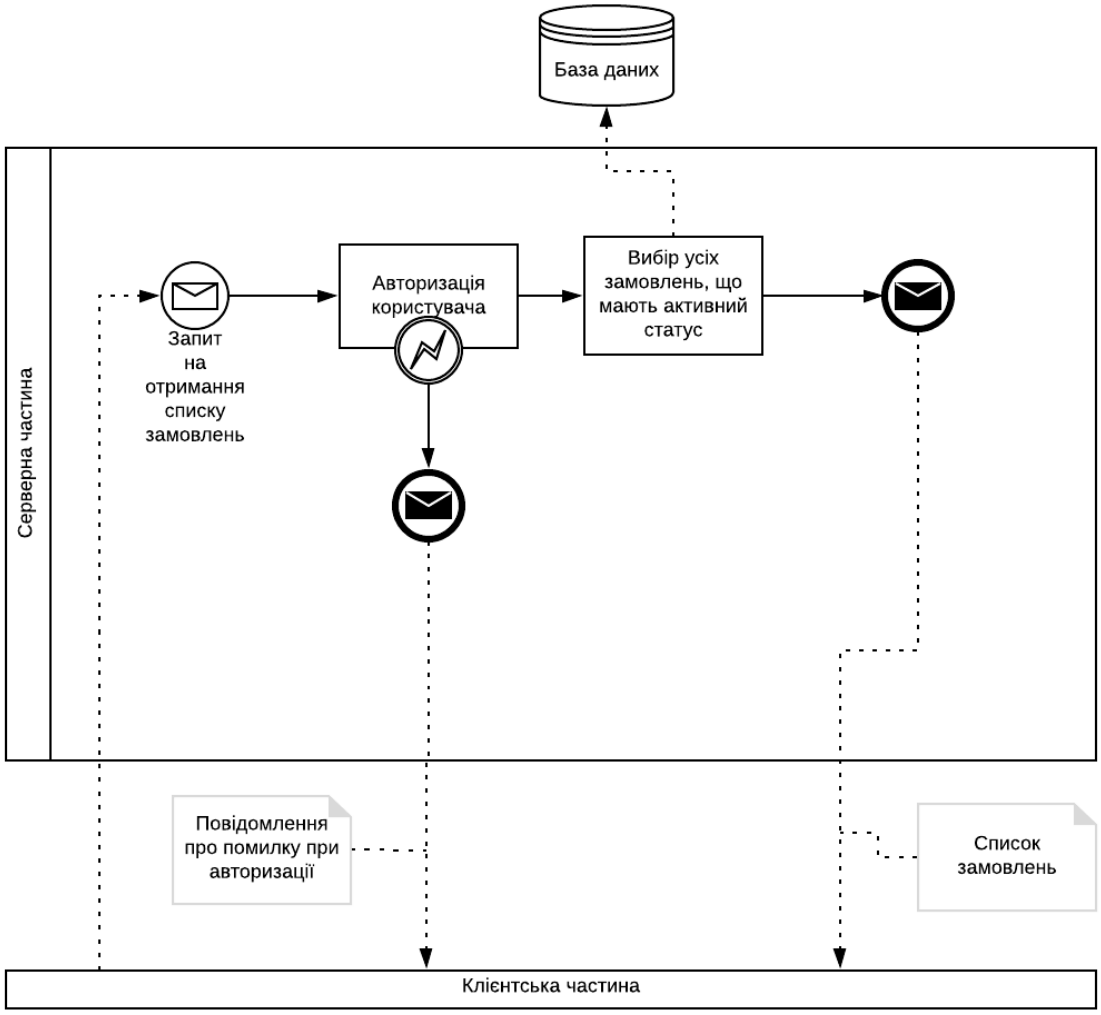


Рис. 0.5 Схема отримання списку активних замовлень

Послідовний опис процесів, зображених на рис. 2.5 для отримання списку активних замовлень:

– Клієнт надсилає запит на отримання списку активних замовлень.

– Сервер проводить авторизацію користувача.

– Якщо при авторизації виникає помилка, сервер відправляє повідомлення про помилку авторизації.

– Сервер вибирає активні замовлення у БД.

– Сервер надсилає список активних замовлень до клієнту.

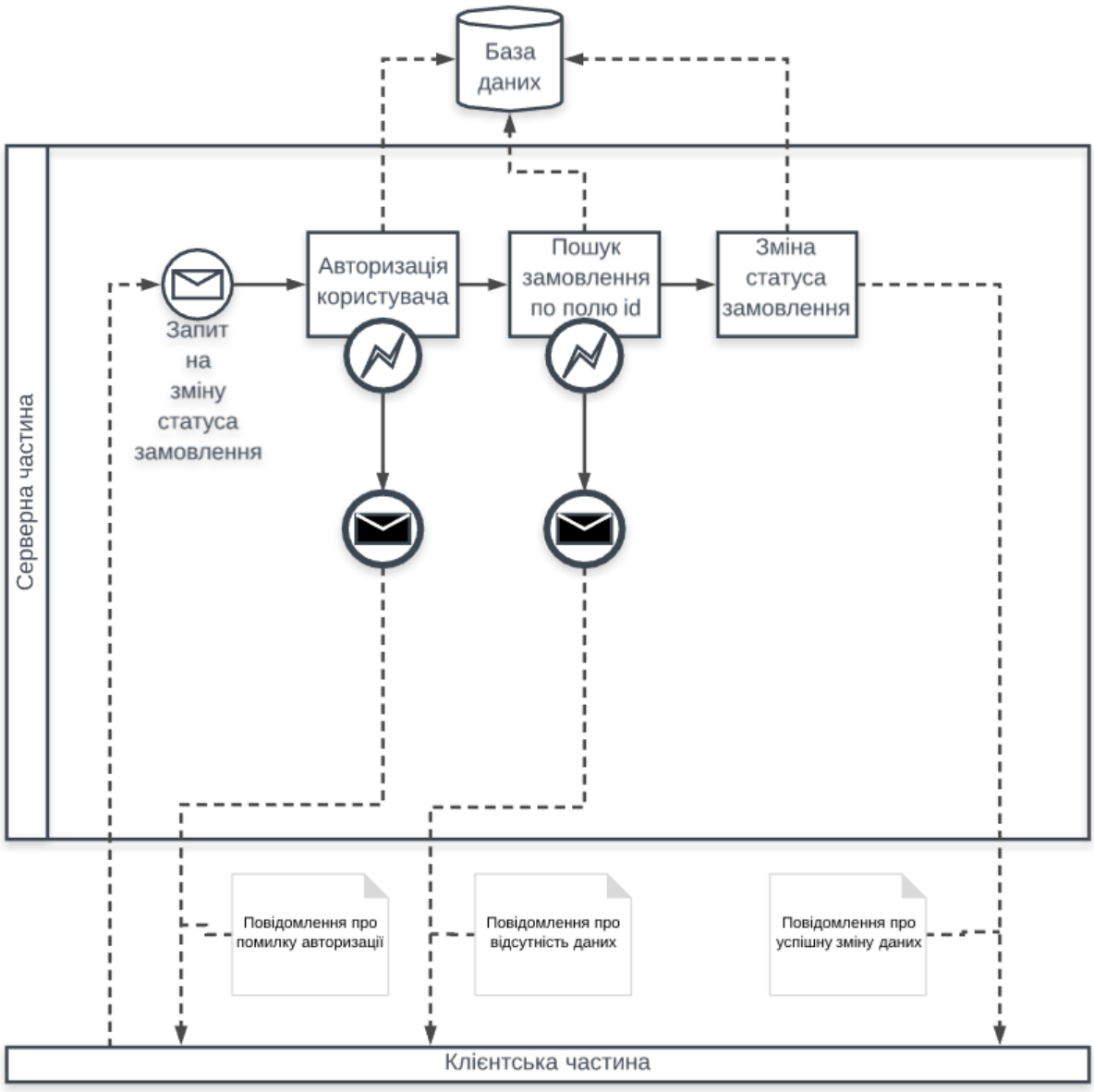


Рис. 0.6 Схема зміни статусу замовлення

Послідовний опис процесів, зображених на рис. 2.6 для зміни статуса замовлення:

– Клієнт надсилає запит на зміну статусу замовлення.

– Сервер проводить авторизацію користувача.

– Якщо при авторизації виникає помилка, сервер відправляє повідомлення про помилку авторизації.

– Сервер вибирає замовлення по крітерію пошуку(в данному віпадку – id).

– Якщо при запиті до БД дані не знайдено, сервер відправляє повідомлення про помилку пошуку даних на клієнт.

– Сервер знаходить замовлення

– Сервер змінює статус даного замовлення та зберігає оновлені дані у БД

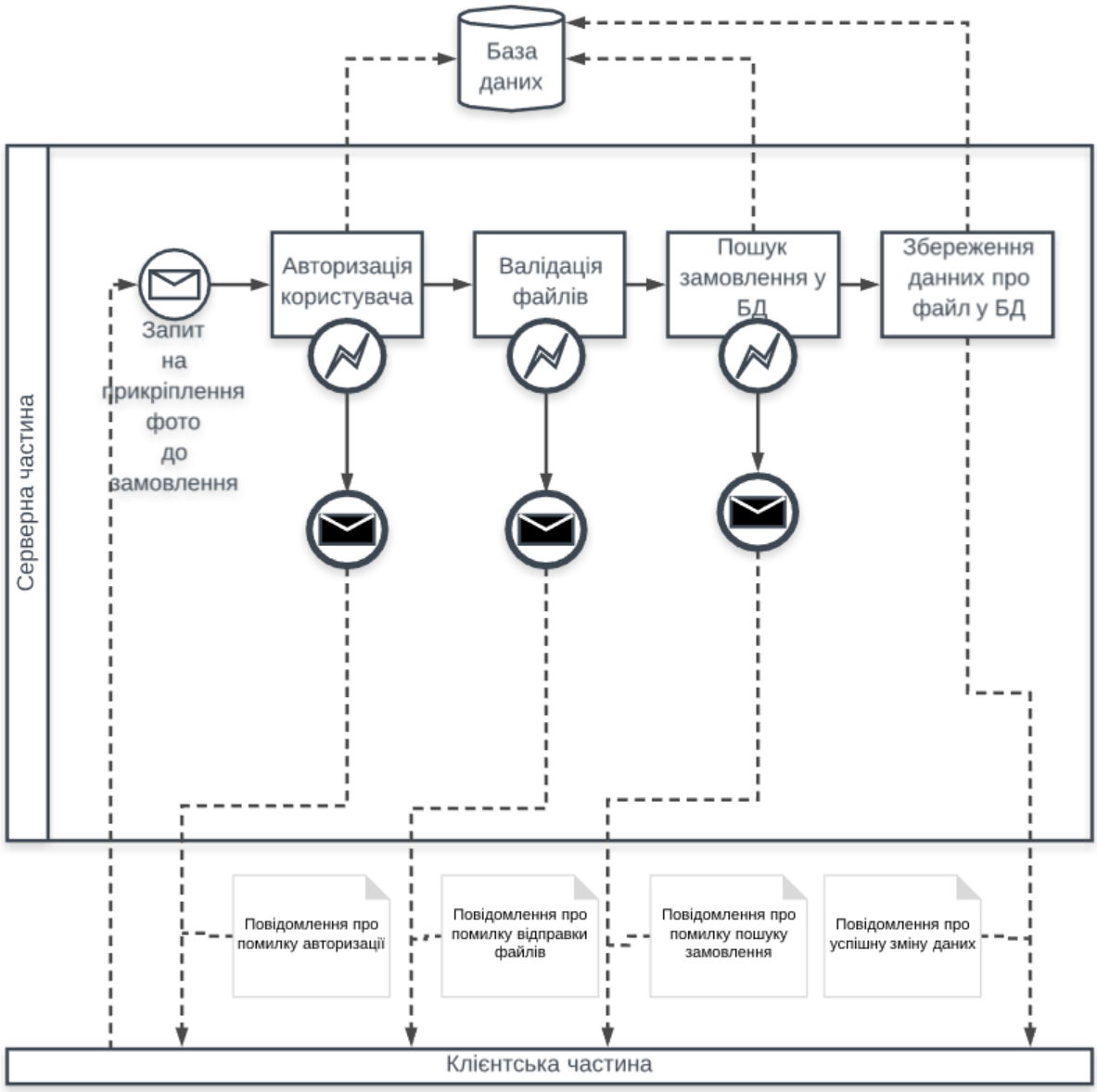


Рис. 0.7 Схема прикріплення файлів до замовлення

Послідовний опис процесів, зображених на рис. 2.7 для прикріплення файлів(фото) до замовлення:

– Клієнт надсилає запит на прикріплення файлів до замовлення.

– Сервер проводить авторизацію користувача.

– Якщо при авторизації виникає помилка, сервер відправляє повідомлення про помилку авторизації.

– Сервер валідує файли.

– Якщо при валідації файлів виникла помилка, сервер відправляє повідомлення про помилку валідації файлів.

– Сервер вибирає замовлення по крітерію пошуку(в данному віпадку – id).

– Якщо при запиті до БД дані не знайдено, сервер відправляє повідомлення про помилку пошуку даних на клієнт.

– Сервер знаходить замовлення

– Сервер зберігає інформацію про файли, прив’язує файли до замовлення та зберігає дані у БД.

2.1.3 Робота зі списком контактів та клієнтів

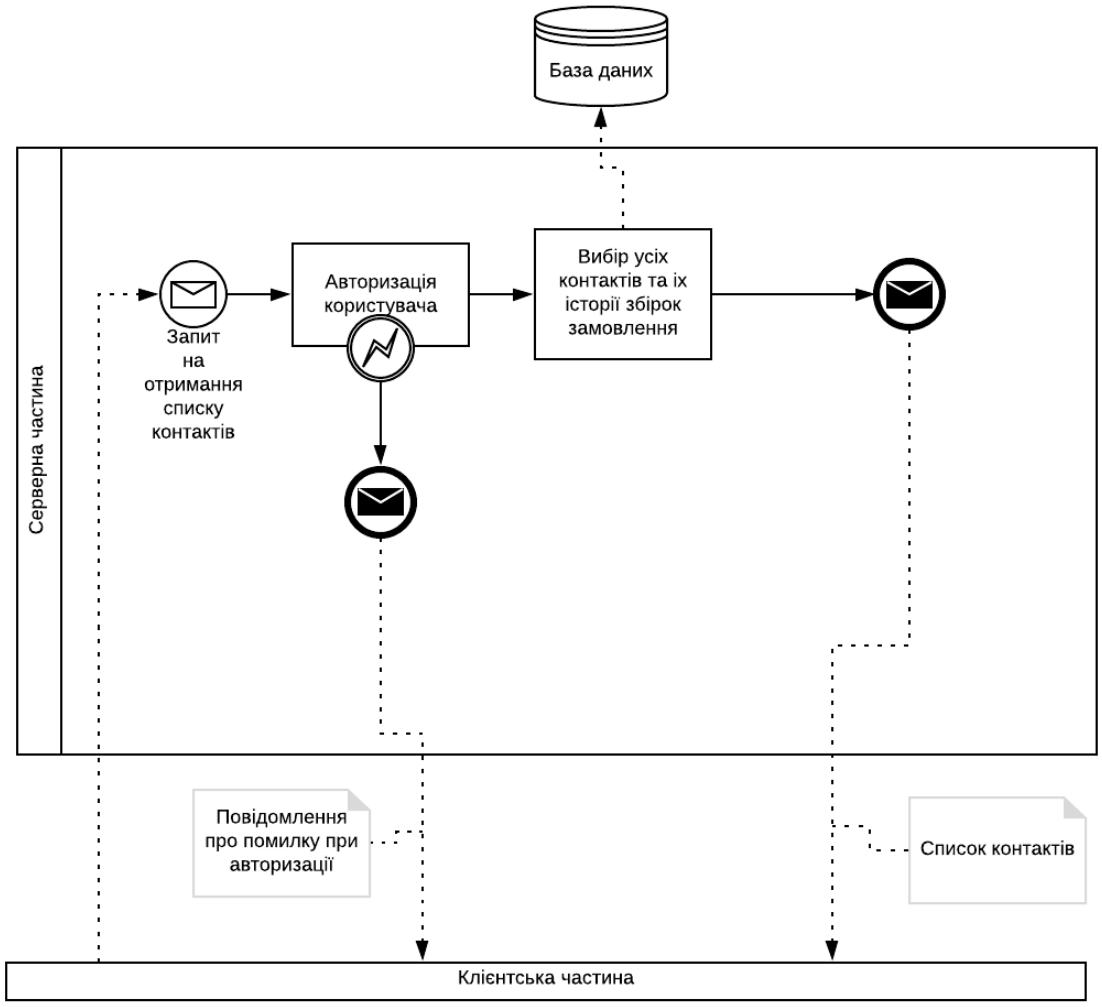


Рис. 0.8 Схема отримання списку контактів

Послідовний опис процесів, зображених на рис. 2.8 для отримання списку контактів:

– Клієнт надсилає запит на отримання списку контактів.

– Сервер проводить авторизацію користувача.

– Якщо при авторизації виникає помилка, сервер відпправляє повідомлення про помилку авторизації.

– Сервер вибирає список контактів та їх історію замовлень збірок замовленнь у БД.

– Сервер надсилає список контактів та іх історій збірок до клієнту.

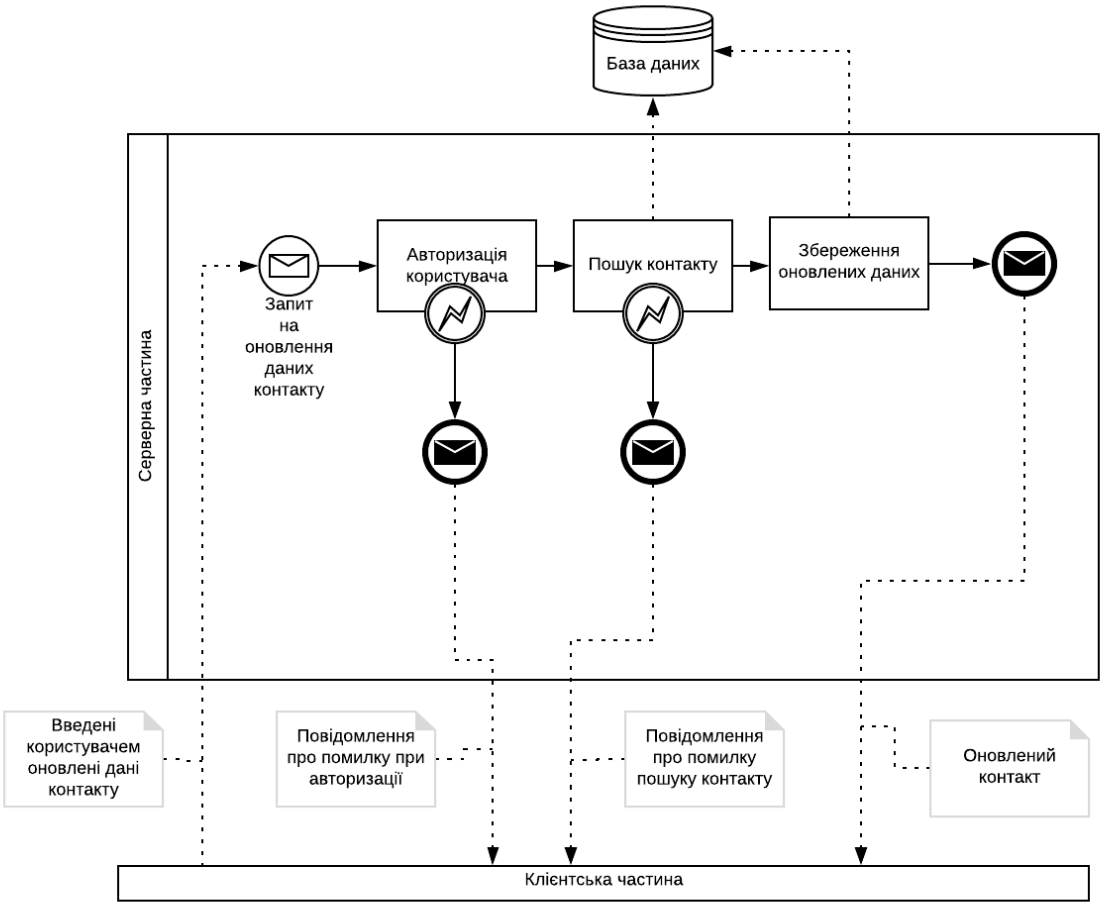


Рис. 0.9 Схема оновлення даних контакту

Послідовний опис процесів, зображених на рис. 2.9 для оновлення даних контакту:

– Клієнт надсилає запит на оновлення інформації контакту.

– Сервер проводить авторизацію користувача.

– Якщо при авторизації виникає помилка, сервер відпправляє повідомлення про помилку авторизації.

– Сервер виконує пошук контатку у БД.

– Якщо у БД даний контакт відсутній, сервер відправляє повідомлення про помилку пошуку.

– Інформація контакта оновлюється та зберігається у БД.

– Сервер надсилає оновлений контакт та його історію збірок замовлень до клієнту.

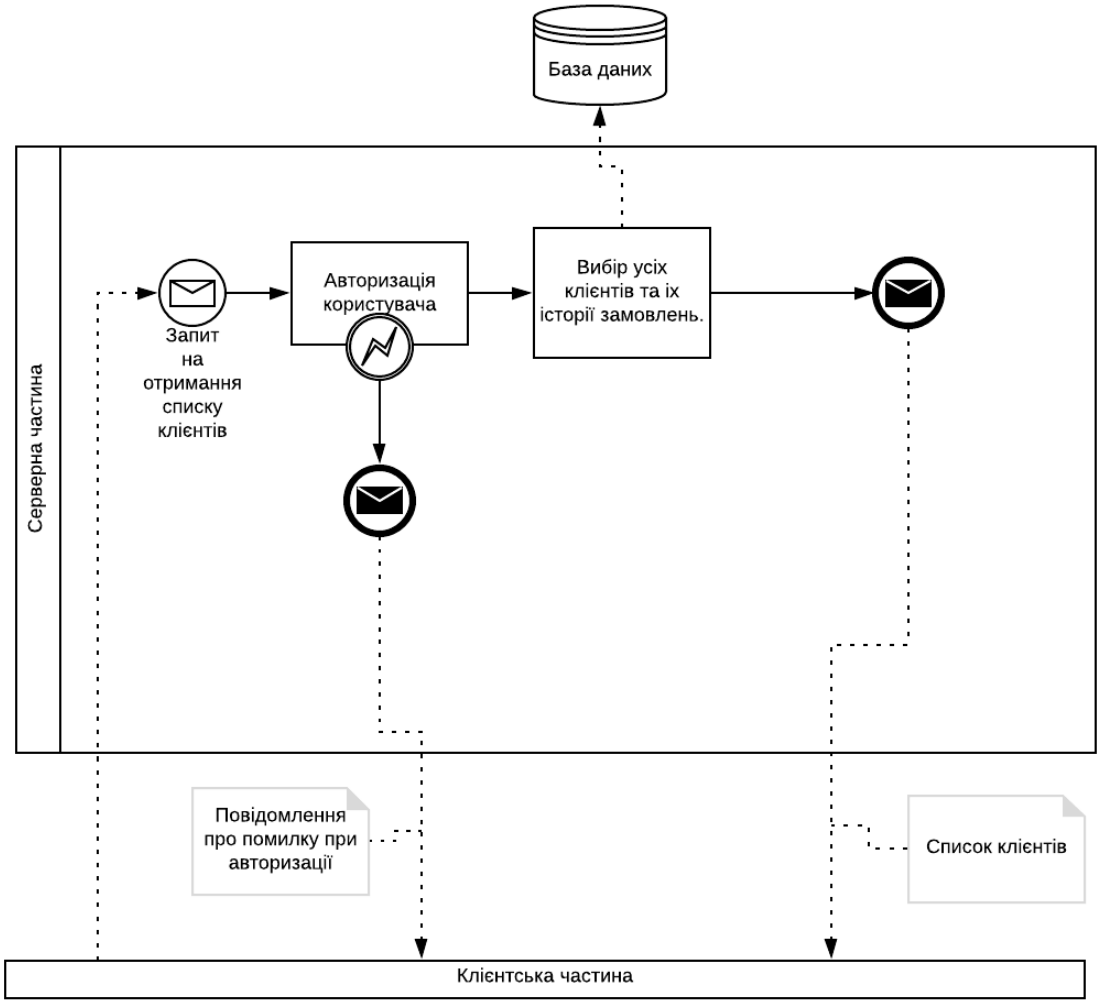


Рис. 0.10 Схема отримання списку клієнтів

Послідовний опис процесів, зображених на рис. 2.10 для отримання списку клієнтів:

– Клієнт надсилає запит на отримання списку клієнтів.

– Сервер проводить авторизацію користувача.

– Якщо при авторизації виникає помилка, сервер відправляє повідомлення про помилку авторизації.

– Сервер вибирає список клієнтів та їх історію замовлень у БД.

– Сервер надсилає список клієнтів та історій їх замовлень клієнту.

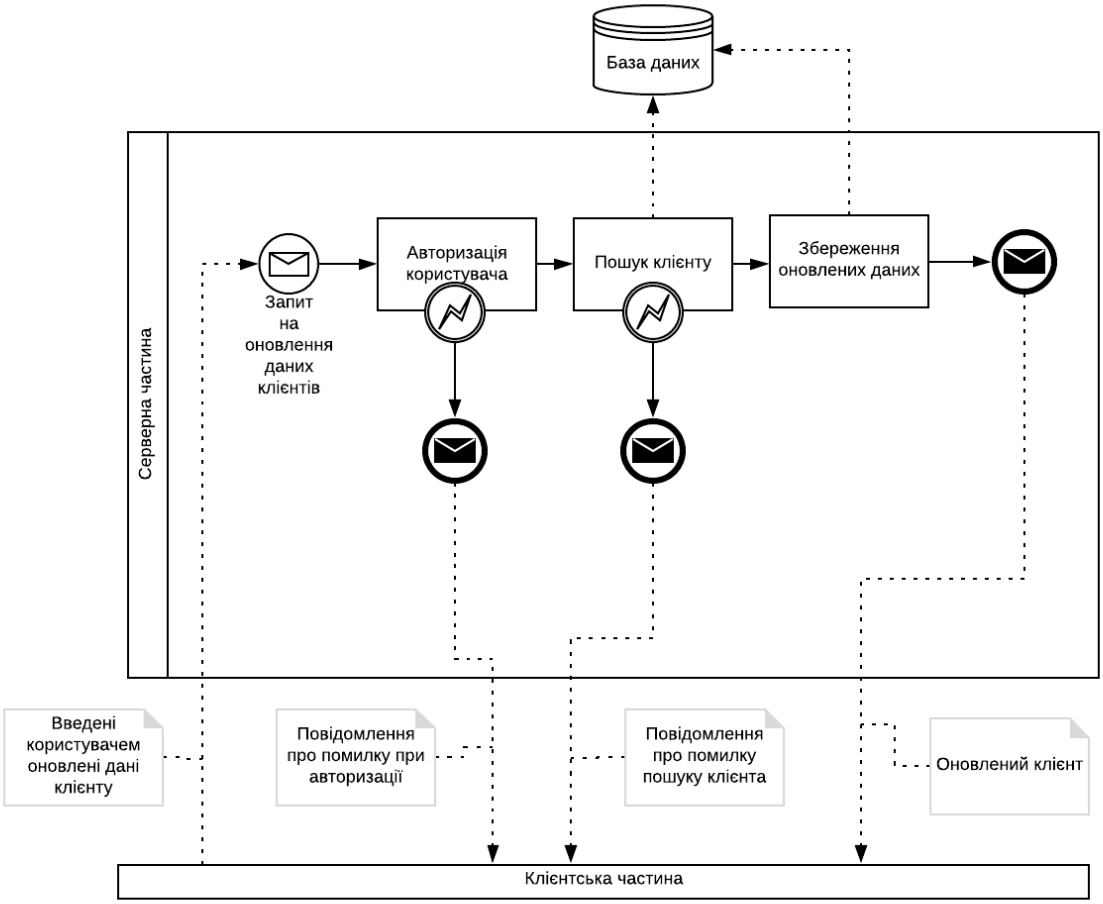


Рис. 0.11 Схема оновлення даних клієнта

Послідовний опис процесів, зображених на рис. 2.11 для оновлення даних клієнта.

– Клієнт надсилає запит на оновлення інформації клієнту.

– Сервер проводить авторизацію користувача.

– Якщо при авторизації виникає помилка, сервер відпправляє повідомлення про помилку авторизації.

– Сервер виконує пошук клієнту у БД.

– Якщо у БД даний клієнт відсутній, сервер відправляє повідомлення про помилку пошуку.

– Інформація клієнта оновлюється та зберігається у БД.

– Сервер надсилає оновлений клієнт та його історію замовлень до клієнту.

2.1.4 Робота зі списком компонентів та конфігурацій

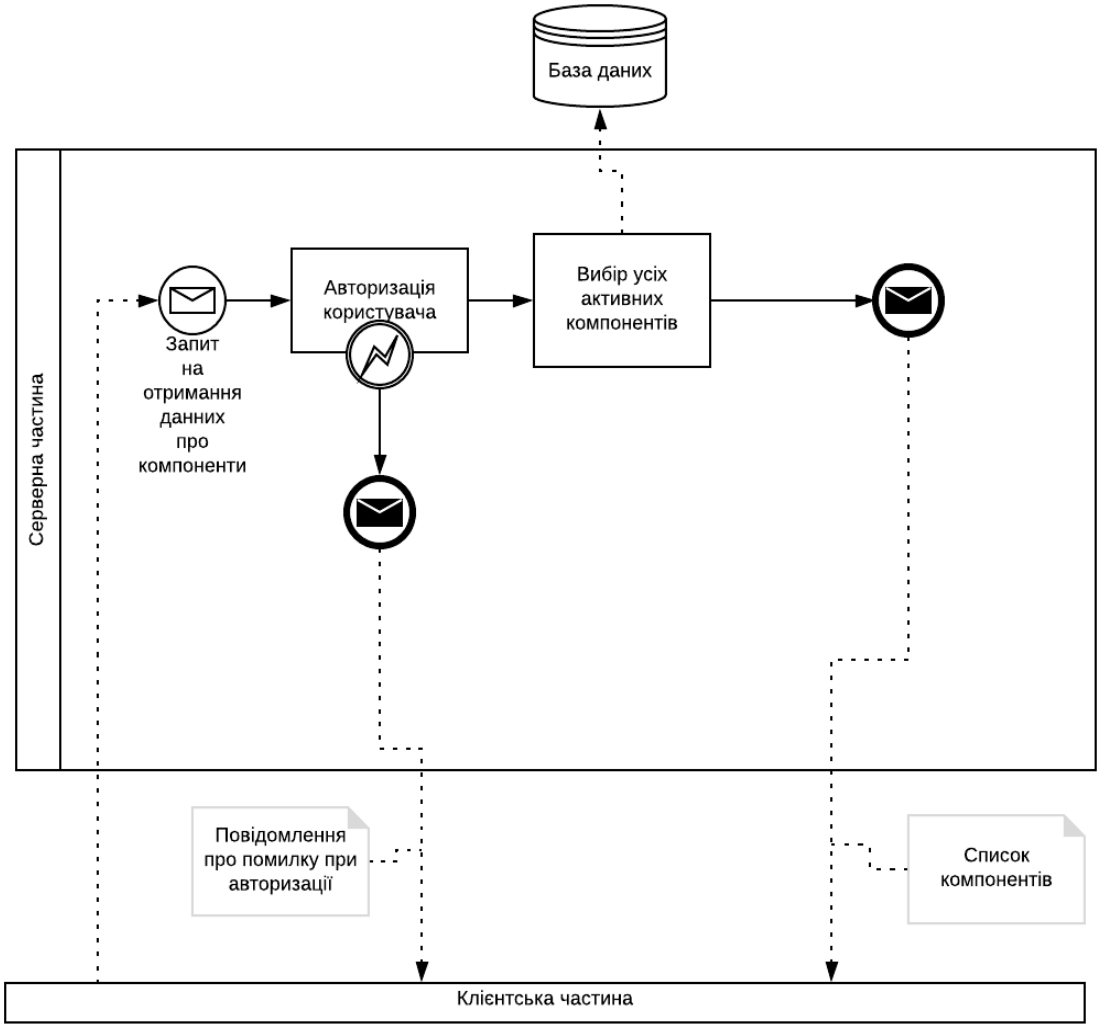


Рис. 0.12 Схема отримання списку компонентів

Послідовний опис процесів, зображених на рис. 2.12 для отримання списку компонентів:

– Клієнт надсилає запит на отримання списку компонентів.

– Сервер проводить авторизацію користувача.

– Якщо при авторизації виникає помилка, сервер відпправляє повідомлення про помилку авторизації.

– Сервер вибирає список компонентів у БД.

– Сервер надсилає список компонентів до клієнту.

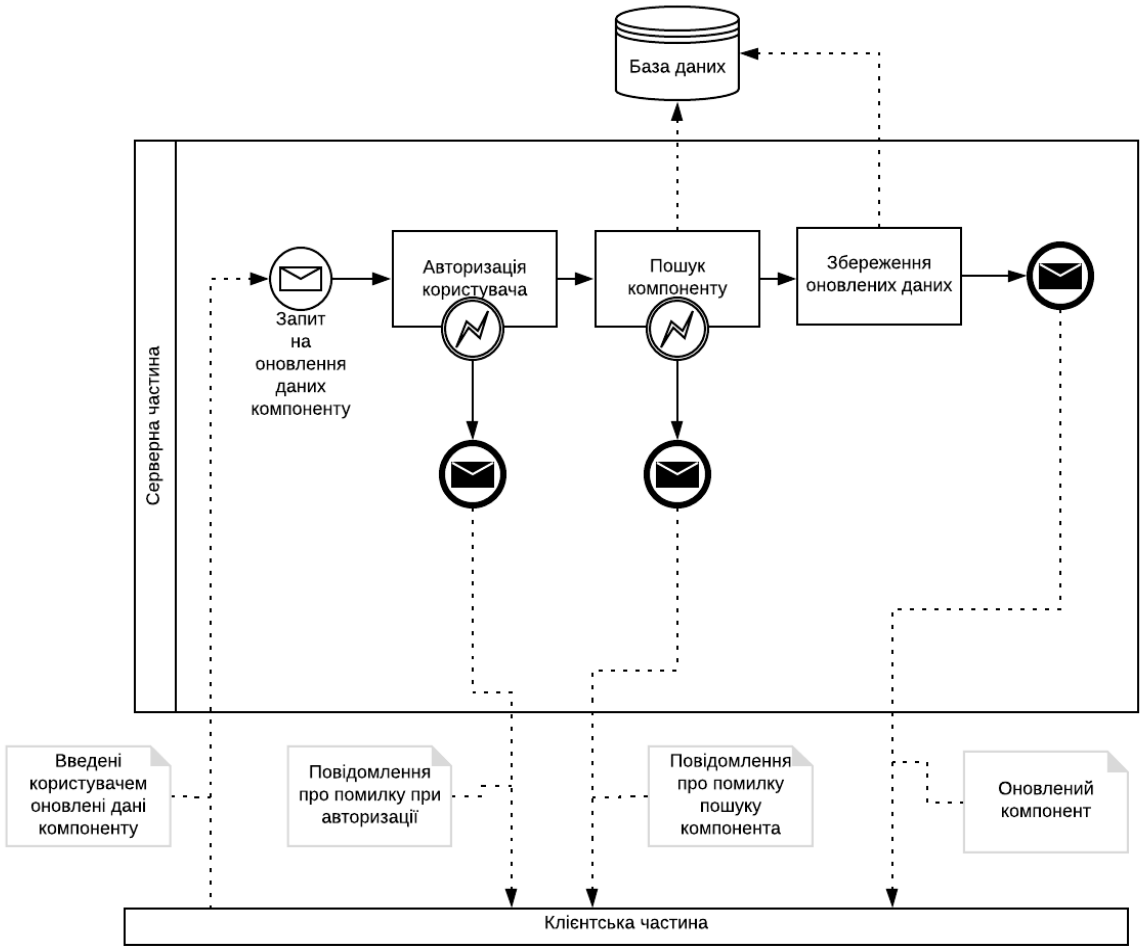


Рис. 0.4 Схема оновлення даних компоненту

Послідовний опис процесів, зображених на рис. 2.13 для оновлення даних компоненту:

– Клієнт надсилає запит на оновлення інформації компоненту.

– Сервер проводить авторизацію користувача.

– Якщо при авторизації виникає помилка, сервер відпправляє повідомлення про помилку авторизації.

– Сервер виконує пошук клієнту у БД.

– Якщо у БД даний компонент відсутній, сервер відправляє повідомлення про помилку пошуку.

– Інформація компонента оновлюється та зберігається у БД.

– Сервер надсилає оновлений компонент до клієнту.

2.2 Програмне забезпечення системи

Архітектура системи побудована на таких складових:
серверна частина – Spring Boot, JPA(Hibernate), Spring Data, Spring Security;
клієнтська частина – Html, Thymeleaf, CSS3, JavaScript, JQuery.

Hibernate — засіб відображення між об'єктами та реляційними структурами (object-relational mapping, ORM) для платформи Java. Hibernate є вільним програмним забезпеченням. Hibernate надає легкий для використання каркас (фреймворк) для відображення між об'єктно-орієнтованою моделлю даних і традиційною реляційною базою даних.
Метою Hibernate є звільнення розробника від значних типових завдань із програмування взаємодії з базою даних. Розробник може використовувати Hibernate як при розробці з нуля, так і для вже існуючої бази даних.

Thymeleaf — сучасний серверний механізм Java-шаблонів що здатний обробляти HTML, XML, JavaScript, CSS і навіть простий текст.

JavaScript — динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, що надає можливість на стороні клієнта (пристрої кінцевого користувача) взаємодіяти з користувачем, керувати браузером, асинхронно обмінюватися даними з сервером, змінювати структуру та зовнішній вигляд веб-сторінки.

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу.

JQuery – бібліотека, що використовується в мові JavaScript. Основний фокус бібліотеки – взаємодія Html з JavaScript. Закладена на рівні ядра, що дозволяє перейти до об’єктної моделі файлів та документів. Базовий функціонал JQuery може бути розширенний завдяки плагінам. Основні переваги в використанні цієї бібліотеки:

– Мала кількість коду, так як більшість JavaScript коду реалізовано у бібліотеці.
– Програмний код більш зрозуміліший, та читабельний.
– Зручна маніпуляція елементами.

Висновки до розділу 2

У даному розділі було проведено моделювання та проектування системи обробки замовлень інтернет-магазину. Було створено та описано усі необхідні схеми та табиці. Розглянуто програмне забезпечення та технології, що використовуться для створення системи.

РОЗДІЛ 3
КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Базуючись на отриманих вимогах для розробки системи обробки інтернет замовлень опишемо результат створення програмного забезпечення.

3.1 Конструювання програмного забезпечення

Для розробки серверної частини було обрано архітектурний шаблон MVC, що поділяє систему на три частини: модель даних, вигляд даних та керування. Застосовується для відокремлення даних (модель) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

У мові програмування Java концепція MVC підтримується на рівні стандартних класів-бібліотек. В результаті використання парадигми MVC програміст отримує в своє розпорядження могутню структуру об'єктів-компонентів, функції яких чітко розмежовані, що гарантує надійність і розширюваність системи, що розробляється.

3.2 Опис програмних модулів системи

Програмні модулі системи наведені у таблиці 3.1.

Таблиця 3.1 Програмні модулі системи

№ з/п	Позначення	Призначення
1	HomeController.java	Контролер домашньої сторінки системи
2	LoginController.java	Контролер який відповідає за аутентифікацію користувачів

Продовження таблиці 3.1 Програмні модулі системи

№ з/п	Позначення	Призначення
3	ContactController.java	Контролер який відповідає за контактну інформацію та історію активності у системі.
4	LedLampController.java	Контролер який відповідає за конструювання замовлення типу «LedLamp».
5	MaterialController.java	Контролер який відповідає відображення компонентів, реагування інформації про компоненти.
6	OrderController.java	Контролер який відповідає за відображення списків замовлення, відфільтрованих та відсортованих.
7	ResourceController	Контролер, що відповідає за збереження ресурсніх файлів, що використовує система.
8	ServiceController	Контролер який відповідає за відображення необхідних сторінок системи.
9	UserController	Контролер, що відповідає за підтвердження нових зареєстрованних користувачів в системі.
10	ComponentsApiController	API контроллер, що надає можливість отримувати інформацію про конкретні компоненти.

Продовження таблиці 3.1 Програмні модулі системи

№ з/п	Позначення	Призначення
11	ConfigurationsApiController	API контроллер який використовується для того, щоб отримувати дані про конфігурації в конструкторі збірки замовлення для клієнтів, редагування інформації конфігурації.
12	ContactsApiController	API контроллер який використовується для того, щоб отримувати дані про контакти зареєстровані в системі. Редагування, створення нових контактів.
13	ImagesApiController	API контроллер який використовується для зберігання файлі(фото) у системі, перегляду наявних у системі.
14	MaterialsApiController	API контроллер який використовується для того, щоб отримувати дані про компоненти в системі. Редагування існуючих компонентів, що використовуються у замовленнях.
15	OrdersApiController	API контроллер який використовується для того, щоб отримувати дані про замовлення в системі. Редагування існуючих замовлень, зміна статусів. Отримання інформації про деталі замовлень.

Продовження таблиці 3.1 Програмні модулі системи

№ з/п	Позначення	Призначення
16	SystemSettingsApiController	API контроллер, що відповідає за роутінг в системі
17	UsersApiController	API контроллер який використовується для того, щоб отримувати інформацію про користувачів системи, редагувати дані, створювати нових користувачів.
18	Assemblage.html	Сторінка замовлень, що призначені на відповідального за підготовку.
19	Assemble-history	Сторінка, ня якій можна переглядати історію замовлень, що були призначені на відповідального за підготовку.
20	Board.html	Сторінка, що відповідає за представлення усіх активних замовлень в системі, що відсортувовані за статусом.
21	Build-receive.html	Сторінка з замовленнями, які може переглядати відповідний за збір замовлення.
21	Build-in-work.html	Сторінка з замовленнями, які може переглядати відповідний за збір замовлення. На даній сторінці можна прикріпляти фото, переглядати інформацію про клієнта.
22	Components.html	Сторінка на якій відображаються компоненти, що використовуються для побудови замовлення.

Продовження таблиці 3.1 Програмні модулі системи

№ з/п	Позначення	Призначення
23	Configurations.html	Сторінка на якій відображаються конфігурації, що використовуються для побудови замовлення.
24	Contacts.html	Сторінка що використовується для відображення даних про контакти/клієнтів. Редагування та додавання клієнтів.
25	Login.html	Сторінка з формою логіну для аутентифікації в системі.
26	Orders.html	Сторінка на якіі можливий перегляд замовлень, що знаходятся в статусі «Оплачені», «На підтвердження».
27	Users.html	Сторінка що використовується для відображення користувачів системи. Створювання нових, редагування інформації.

Опис методів в освновних классах зображено на таблицях 3.2 – 3.4.

Таблиця 3.2 Методи класу OrderService

Назва метода	Аргументи	Значення, що повертається	Призначення методу
createOrder	orderRequest	-	Створення нового замовлення в системі.

Продовження таблиці 3.2 Методи класу OrderService

Назва метода	Аргументи	Значення, що повертається	Призначення методу
getOrderDataOrThrowException	Long id	OrderData orderData	Отримати замовлееня за його полем id
setResponsiblePersons	Long id, Long respAsId, Long respBuId	-	Призначення відповідального за підготовку компонентів, та відповідального за збір замовлення.
getOrdersPage	Pageable pageable	List <OrderData> orders	Отримати список замовлень відсортованих
getOrdersByStatus	String status	List <OrderData> orders	Отримати список замовлень відсортованих за статусом
getAllOrders	-	List <OrderData> orders	Отримати список усіх замовлень

Продовження таблиці 3.2 Методи класу OrderService

Назва метода	Аргументи	Значення, що повертається	Призначення методу
getAllOrdersByResponsibleId	Long respId	List <OrderData> orders	Отримати список усіх замовлень відфільтрованих за відподальним.
changeStatys	String status, Long id	OrderData orderData	Змінити статус замовлення.
changeSubStatus	String subStatus, Long id	OrderData orderData	Змінити підстатус замовлення.
getApproveOrders	-	List <AproveOrder> approveOrders	Отримати список замовлень з фото – звітом, що потребують підтвердження.

Таблиця 3.3 Методи класу ContactService

Назва метода	Аргументи	Значення, що повертається	Призначення методу
getContacts	Conditions: sort, page, limit	List<Contact> contacts	Отримати список контактів за набором умов фільтрації, або сортування
getContactById	Long id	List<Contact> contact	Отримання контакту за його полем Id
grtContactByPhone	String phone	Contact contact	Отримання контакту за його полем phone
updateContact	Contact contact, Long id	Contact contact	Оновлення контакту, що був знайдений у бд за його полем Id
deleteById	Long Id	-	Видалення контакту з БД, що був знайдений по полю Id
deleteByPhone	String phone	-	Видалення контакту з БД, що був знайдений по полю phone

Таблиця 3.4 Методи класу UserService

Назва метода	Аргументи	Значення, що повертається	Призначення методу
login	String login, String password	UserDetails userDetails	Аутентифікація користувача
save	UserCreate user	-	Збереження користувача у БД
getUserById	Long id	UserResponse user	Отримати інформацію про користувача
updateUser	Long id, UserCreate user	UserResponse user	Редагування інформації користувача
activateUser	String activationCode	-	Активування користувача в системі.
getUsersByRole	String role	List <User>	Отримати список користувачів системи за роллю.

Висновки до розділу 3

У даному розділі було описано основні модулі програмного забезпечення, що використовуються для системи обробки даних. Архітектуру системи було розроблено на базі архітектурного шаблону MVC.

РОЗДІЛ 4
ВПРОВАДЖЕННЯ ТА ОГЛЯД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Щоб розгорнути серверну частину програмного забезпечення, необхідна наявність JDK 8+, що можуть бути встановлені з офіційного сайту для необхідної ОС.

4.2 Огляд інтерфейсу системи

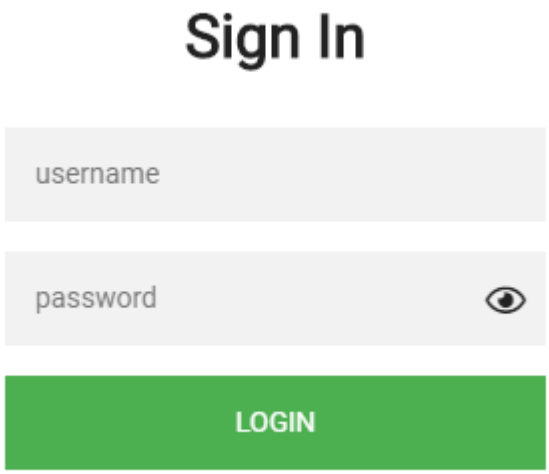


Рис. 4.1 Інтерфейс компонента аутентифікації

На почтакову єкрані відображається вікно, що зображено на рис. 4.1 аутентифікації користувача. Користувач має можливість ввести дані, що будуть відправлені до сервера, та в випадку успішної валідації аунтентифікуватись в систему. Якщо, дані не пройшли валідацію на серверній частині, система відобразить повідомлення про помилку «Не праввильно введенний логін або пароль», що зображено на рис. 4.2.

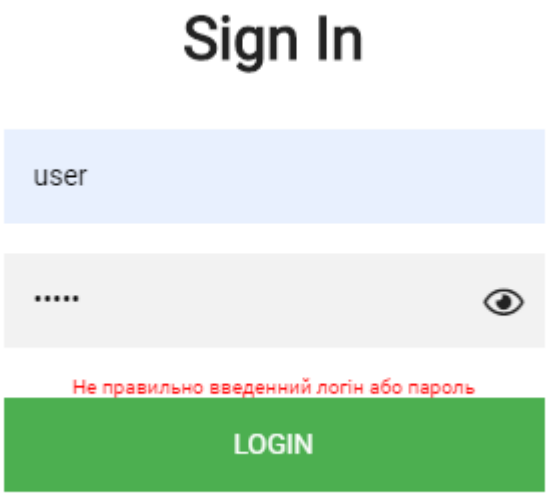


Рис. 4.2 Компонент аутентифікаціх, що видає помилку

У разі успішної аутентифікації користувач перенаправляється на головну сторінку системи, що показана на рис. 4.3. На даній сторінці зображено усі активні замовлення та етапи на яких вони зараз знаходятся, та їх статуси роботи. Також можно побачити призначених відповідальних користувачів на деяких етапах роботи, таких як підготовка компонентів замовлення та збір замовлення.

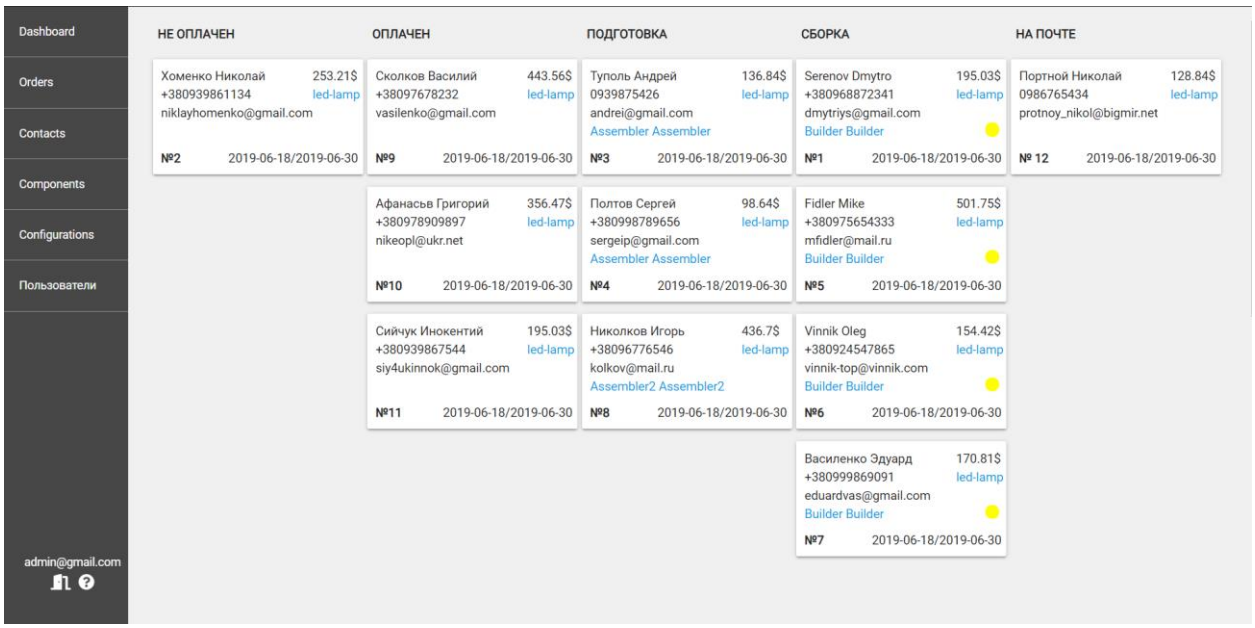


Рис. 4.1 Основна сторінка сервісу «Дошка замовлень»

На рис. 4.4 зображене головне меню системи, що може бачити Користувач системи з роллю «Адміністратор».

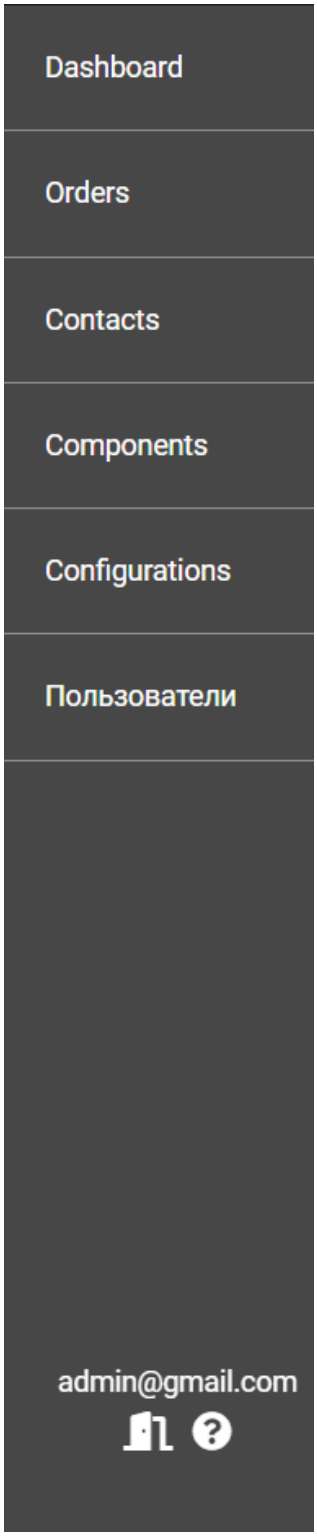
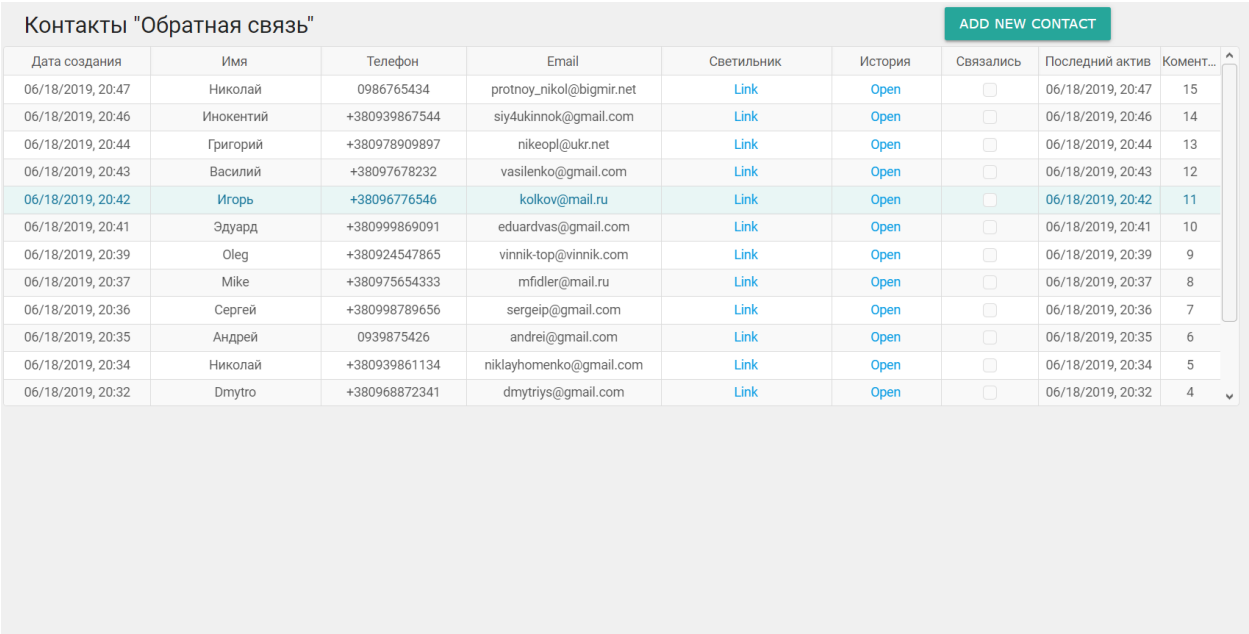


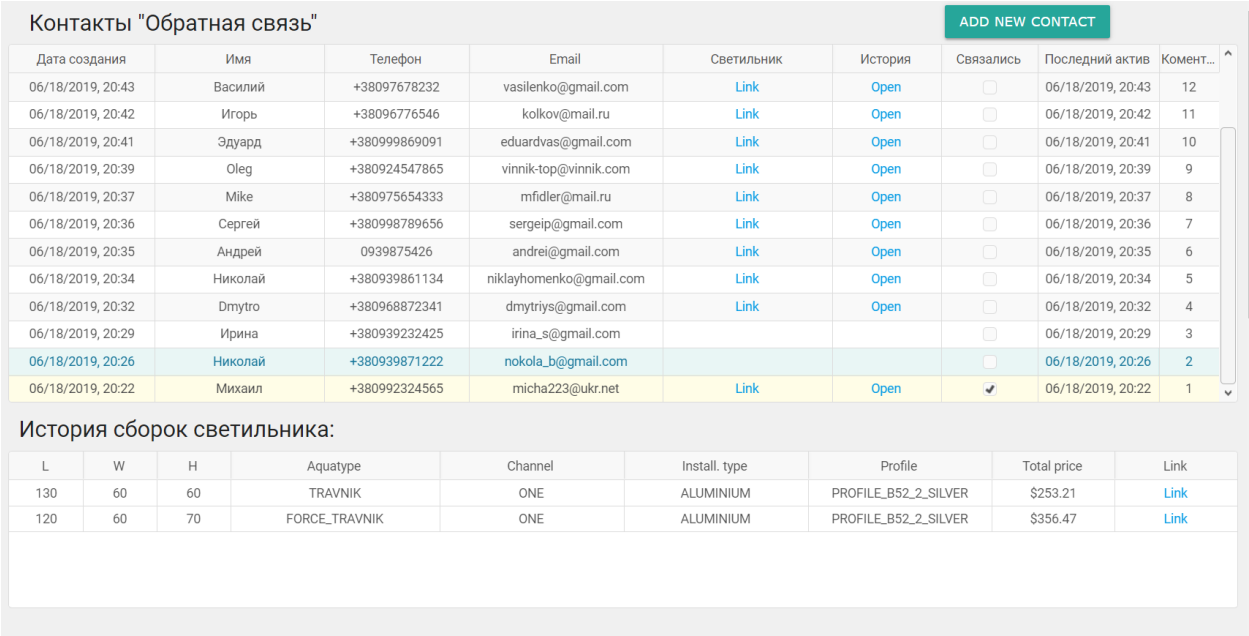
Рис. 4.2 Головне меню системи

На рис. 4.5. зображена сторінка, що відображає список усіх контактів та історію їх збірок замовлення, що зображено на рис 4.6.



Дата создания	Имя	Телефон	Email	Светильник	История	Связались	Последний актив	Коммент...
06/18/2019, 20:47	Николай	0986765434	protnoy_nikol@bigmir.net	Link	Open		06/18/2019, 20:47	15
06/18/2019, 20:46	Инокентий	+380939867544	siy4ukinnok@gmail.com	Link	Open		06/18/2019, 20:46	14
06/18/2019, 20:44	Григорий	+380978909897	nikeopl@ukr.net	Link	Open		06/18/2019, 20:44	13
06/18/2019, 20:43	Василий	+38097678232	vasilenko@gmail.com	Link	Open		06/18/2019, 20:43	12
06/18/2019, 20:42	Игорь	+38096776546	kolkov@mail.ru	Link	Open		06/18/2019, 20:42	11
06/18/2019, 20:41	Эдуард	+380999869091	eduardvas@gmail.com	Link	Open		06/18/2019, 20:41	10
06/18/2019, 20:39	Oleg	+380924547865	vinnik-top@vinnik.com	Link	Open		06/18/2019, 20:39	9
06/18/2019, 20:37	Mike	+380975654333	mfidler@mail.ru	Link	Open		06/18/2019, 20:37	8
06/18/2019, 20:36	Сергей	+380998789656	sergeip@gmail.com	Link	Open		06/18/2019, 20:36	7
06/18/2019, 20:35	Андрей	0939875426	andrei@gmail.com	Link	Open		06/18/2019, 20:35	6
06/18/2019, 20:34	Николай	+380939861134	niklayhomenko@gmail.com	Link	Open		06/18/2019, 20:34	5
06/18/2019, 20:32	Dmytro	+380968872341	dmytriys@gmail.com	Link	Open		06/18/2019, 20:32	4

Рис. 4.3 Сторінка «Контакти»



Дата создания	Имя	Телефон	Email	Светильник	История	Связались	Последний актив	Коммент...
06/18/2019, 20:43	Василий	+38097678232	vasilenko@gmail.com	Link	Open		06/18/2019, 20:43	12
06/18/2019, 20:42	Игорь	+38096776546	kolkov@mail.ru	Link	Open		06/18/2019, 20:42	11
06/18/2019, 20:41	Эдуард	+380999869091	eduardvas@gmail.com	Link	Open		06/18/2019, 20:41	10
06/18/2019, 20:39	Oleg	+380924547865	vinnik-top@vinnik.com	Link	Open		06/18/2019, 20:39	9
06/18/2019, 20:37	Mike	+380975654333	mfidler@mail.ru	Link	Open		06/18/2019, 20:37	8
06/18/2019, 20:36	Сергей	+380998789656	sergeip@gmail.com	Link	Open		06/18/2019, 20:36	7
06/18/2019, 20:35	Андрей	0939875426	andrei@gmail.com	Link	Open		06/18/2019, 20:35	6
06/18/2019, 20:34	Николай	+380939861134	niklayhomenko@gmail.com	Link	Open		06/18/2019, 20:34	5
06/18/2019, 20:32	Dmytro	+380968872341	dmytriys@gmail.com	Link	Open		06/18/2019, 20:32	4
06/18/2019, 20:29	Ирина	+380939232425	irina_s@gmail.com				06/18/2019, 20:29	3
06/18/2019, 20:26	Николай	+380939871222	nokola_b@gmail.com				06/18/2019, 20:26	2
06/18/2019, 20:22	Михаил	+380992324565	micha223@ukr.net	Link	Open	✓	06/18/2019, 20:22	1

L	W	H	Aquatype	Channel	Install. type	Profile	Total price	Link
130	60	60	TRAVNIK	ONE	ALUMINIUM	PROFILE_B52_2_SILVER	$253.21	Link
120	60	70	FORCE_TRAVNIK	ONE	ALUMINIUM	PROFILE_B52_2_SILVER	$356.47	Link

Рис. 4.4 Історія активностей контакта

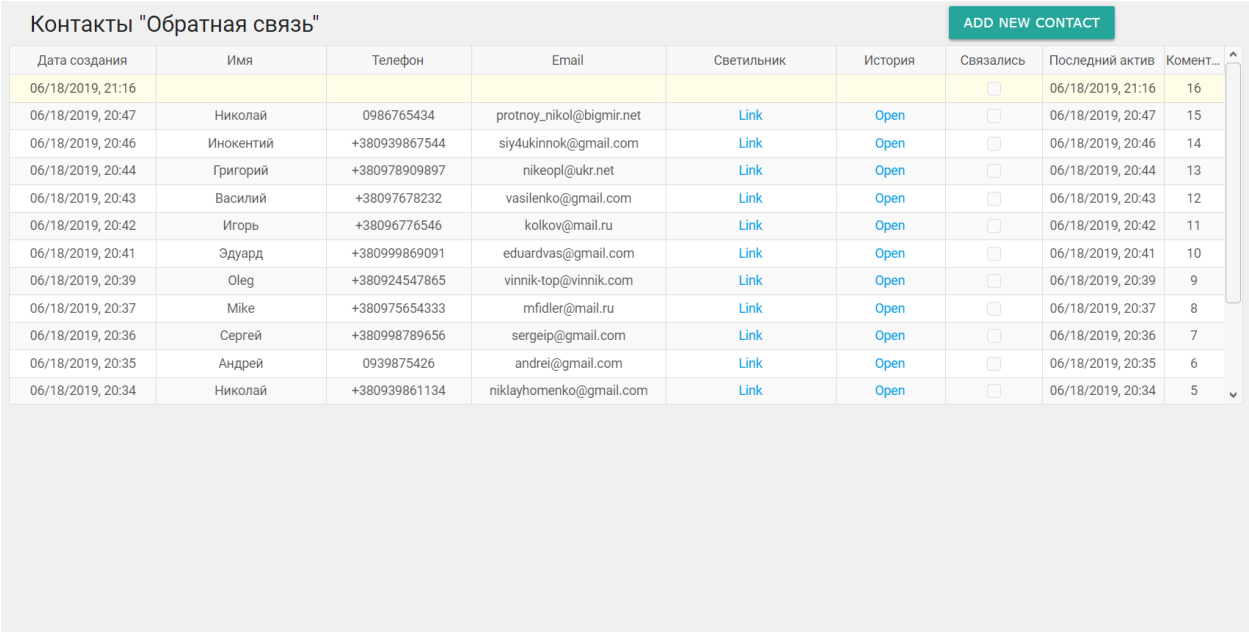


Рис. 4.5 Створення нового контакту в системі

Натиснувши кнопку «Додати новий контакт», автоматично створиться новий ряд даних в таблиці «Контакти». Для того щоб почати редагувати, або додавати інформацію про контакт потрібно натиснути подвійним кліком на необхідну клітинку в записі, щоб активувати режим редагування, зображений на рис. 4.8.

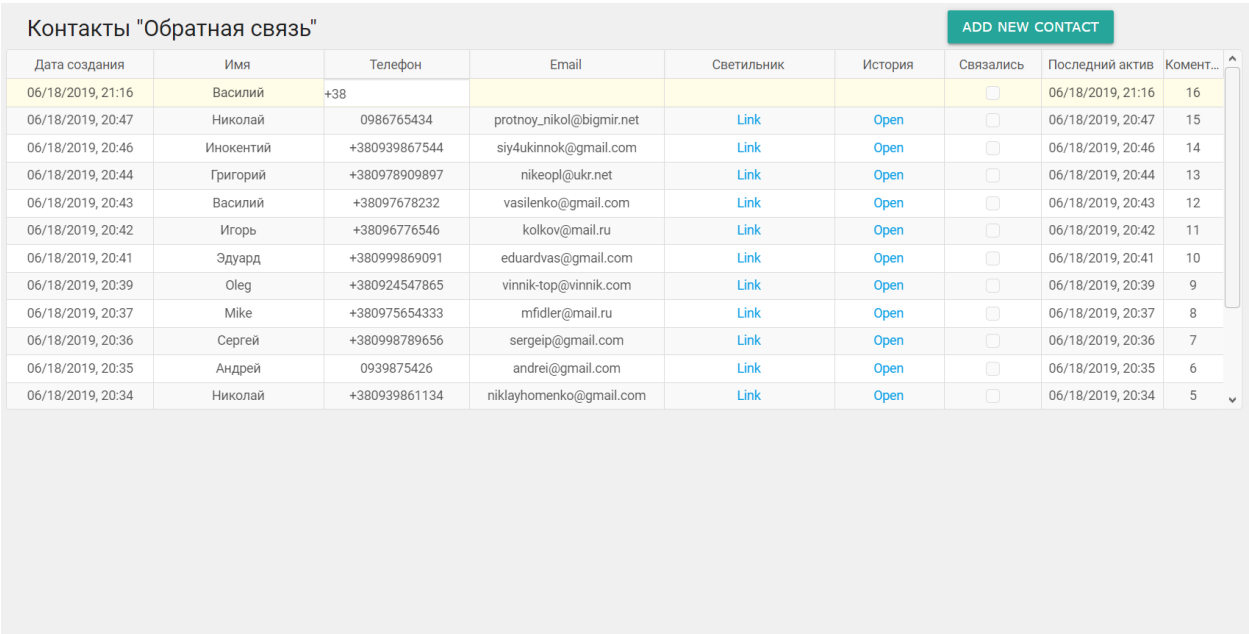


Рис. 4.8 Режим редагування таблиці «Контакти»

На сторінці «Клієнти», що зображено на рис. 4.9 можна побачити список усіх клієнті системи, та їх історію замовлень. Для того щоб почати редагувати, або додавати інформацію про контакт потрібно натиснути подвійним кліком на необхідну клітинку, щоб активувати режим редагування.

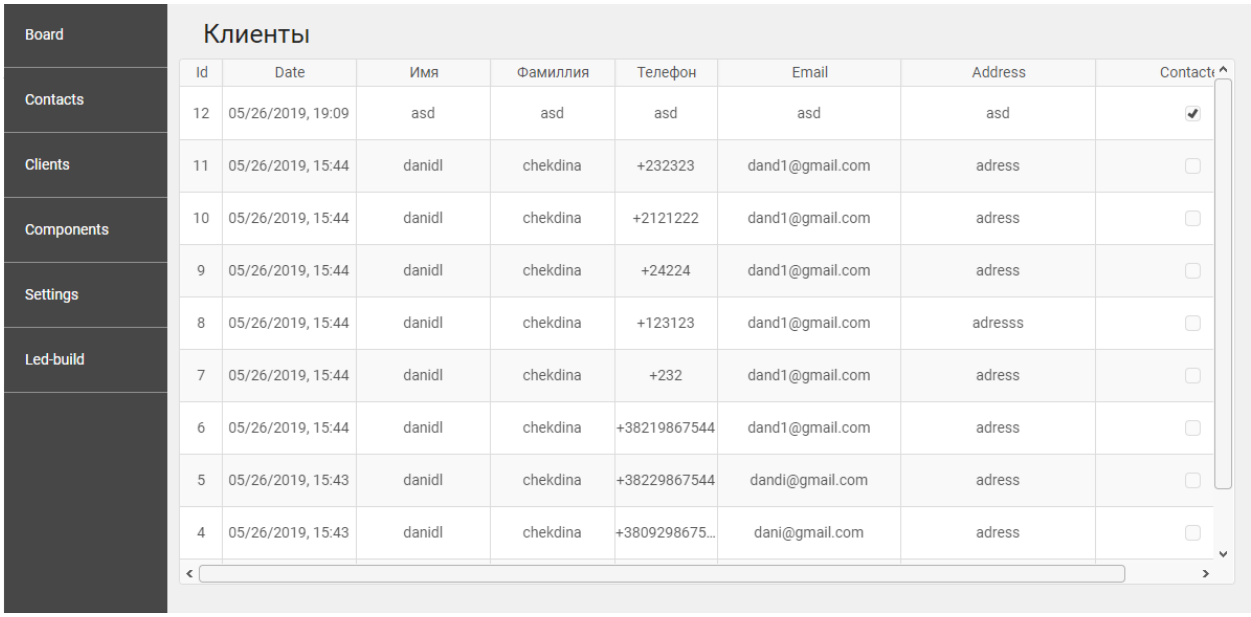


Рис. 4.6 Сторінка «Клієнти»

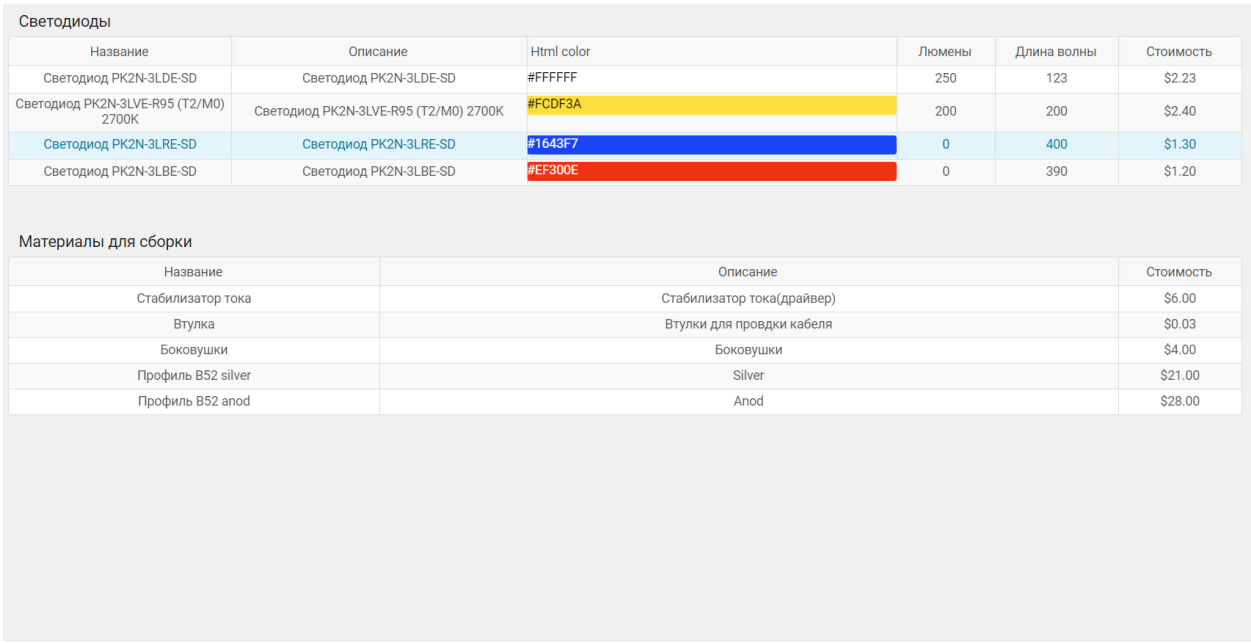


Рис. 4.7 Сторінка «Компоненти»

На сторінці «Компоненти» рис. 4.10 можно побачити список компонентів, що використовуються для збірки замовлення. Для того щоб почати редагувати, або додавати інформацію про компонент потрібно натиснути подвійним кліком на необхідну клітинку, щоб активувати режим редагування.

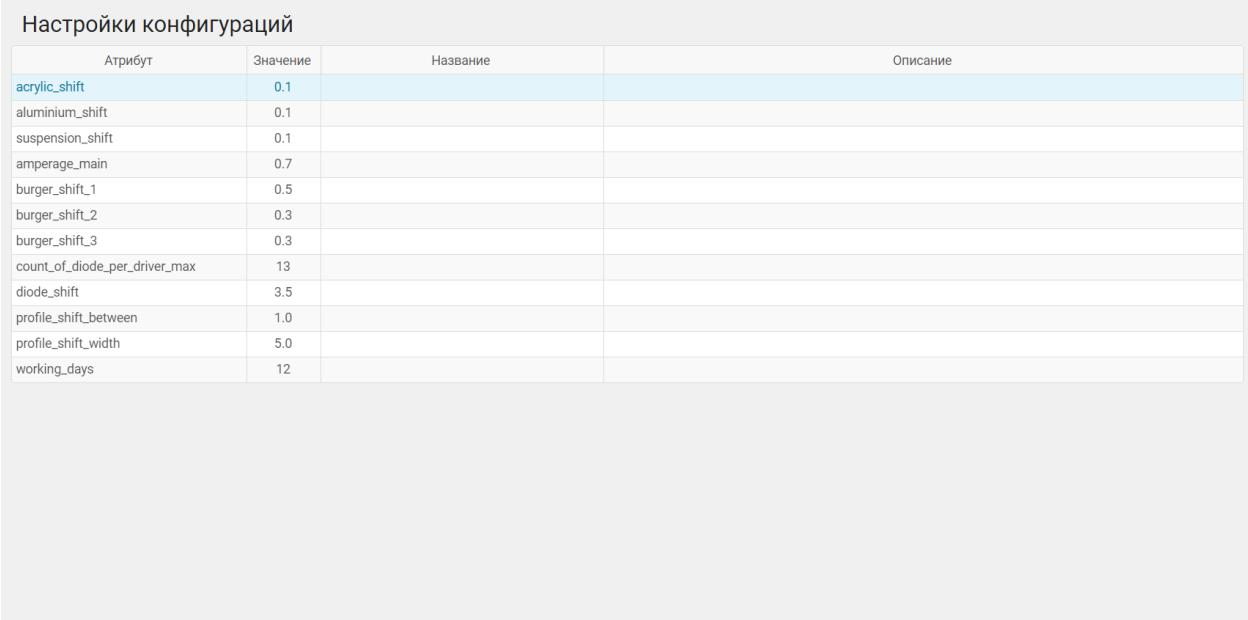


Рис. 4.11 Сторінка «Конфігурації»

На сторінці «Конфігурації» рис. 4.11 зображено список конфігурацій, що використовуються для збірки замовлення. Для того щоб почати редагувати, або додавати інформацію про конфігурацію потрібно натиснути подвійним кліком на необхідну клітинку записа, щоб активувати режим редагування.

На сторінці «Користувачі системи» рис. 4.12 зображено список користувачів, що зареєстровані у даній системі, та детальна інформація про кожного з них.

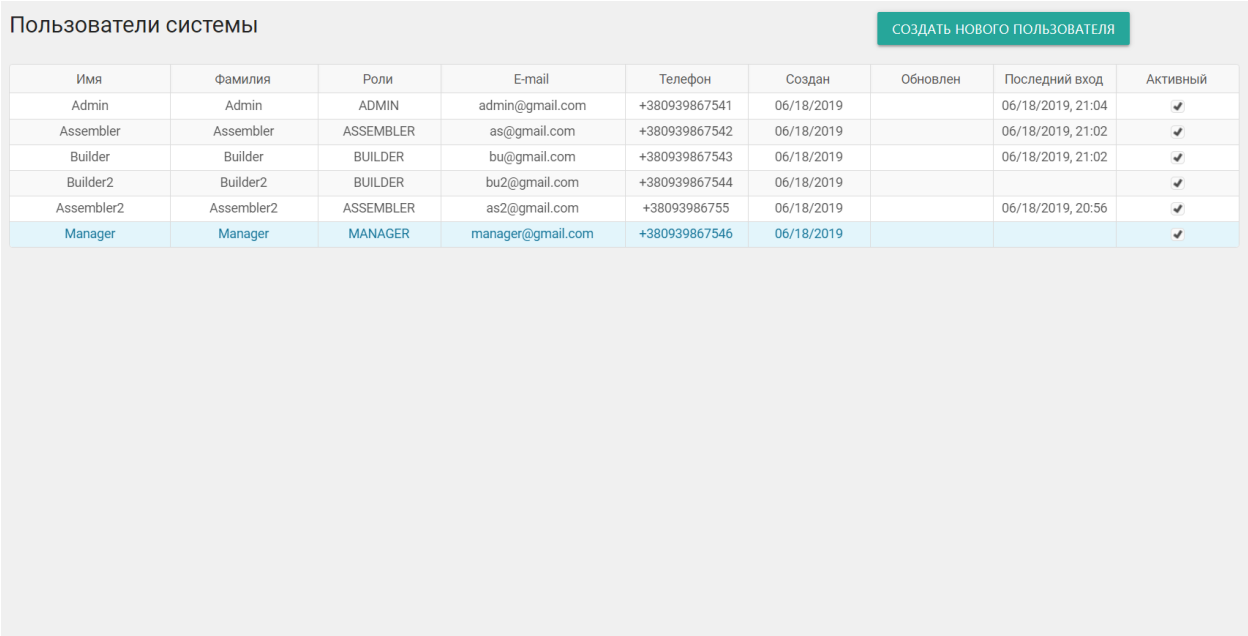


Рис. 4.12 Сторінка «Користувачі системи»

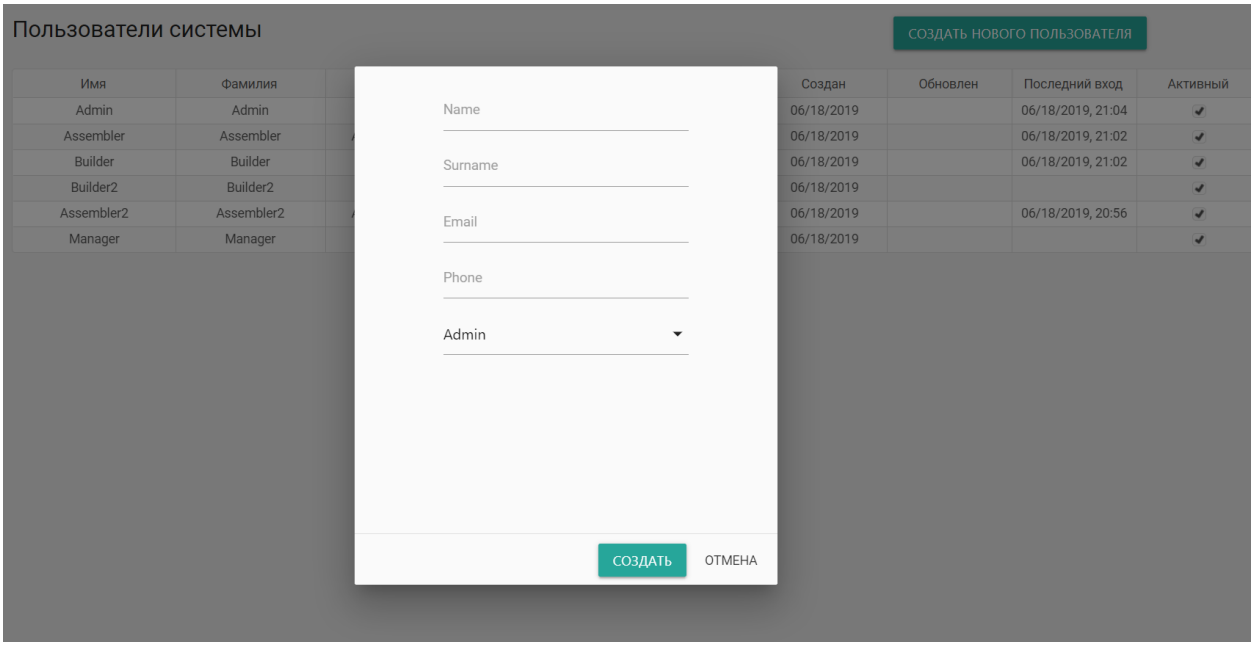


Рис. 4.13 Сторінка «Користувачі системи», реєстрація нового користувача

Щоб зареєструвати нового користувача до системи, потрібно натиснути на кнопку «Створити нового користувача», після чого система відобразить модальне вікно, що зображено на рис. 4.13 та заповнити усі необхідні поля, після чого натиснути кнопку «Зберегти».

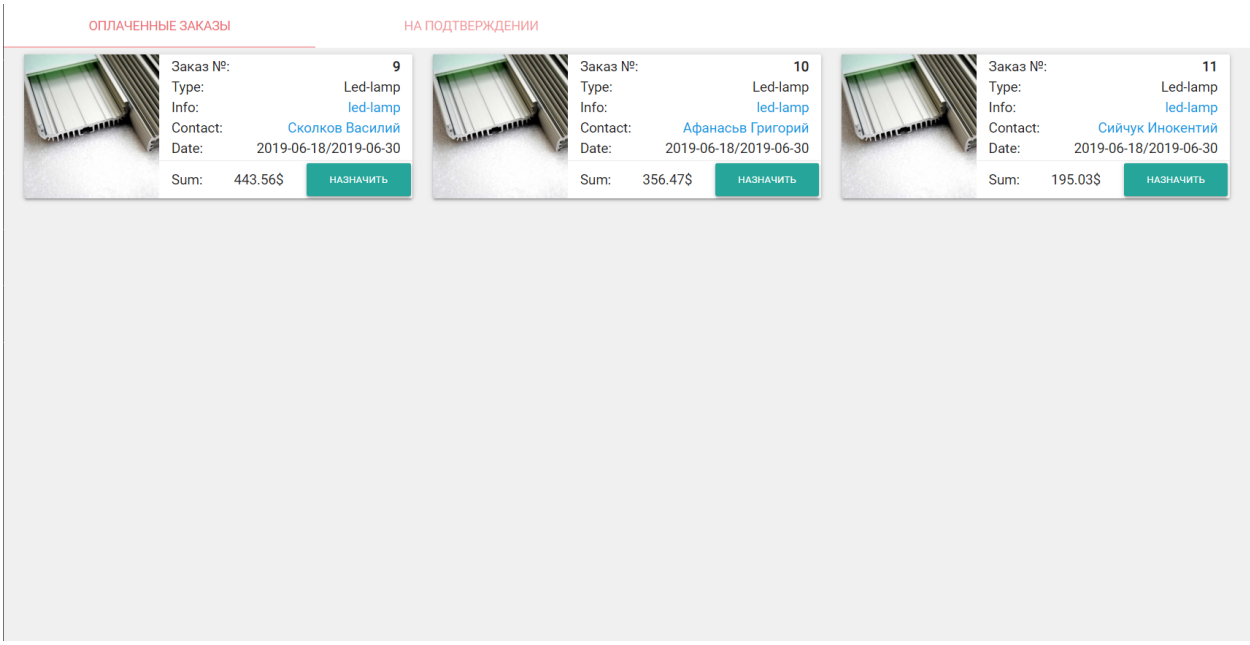


Рис. 4.14 Сторінка «Замовлення».

На сторінці «Замовлення», що зображено на рис. 4.14 можна побачити замовлення, що знаходяться в статусі «Оплачені», та готові для подальших етапів. Щоб призначити відподальних за підготовку компонентів для замовлення, та його збірку, потрібно відкрити модальне вікно, натиснувши на кнопку «Призначити», та обрати відповідних персон(див. рис. 4.15). необхідні поля, після чого натиснути кнопку «Зберегти».

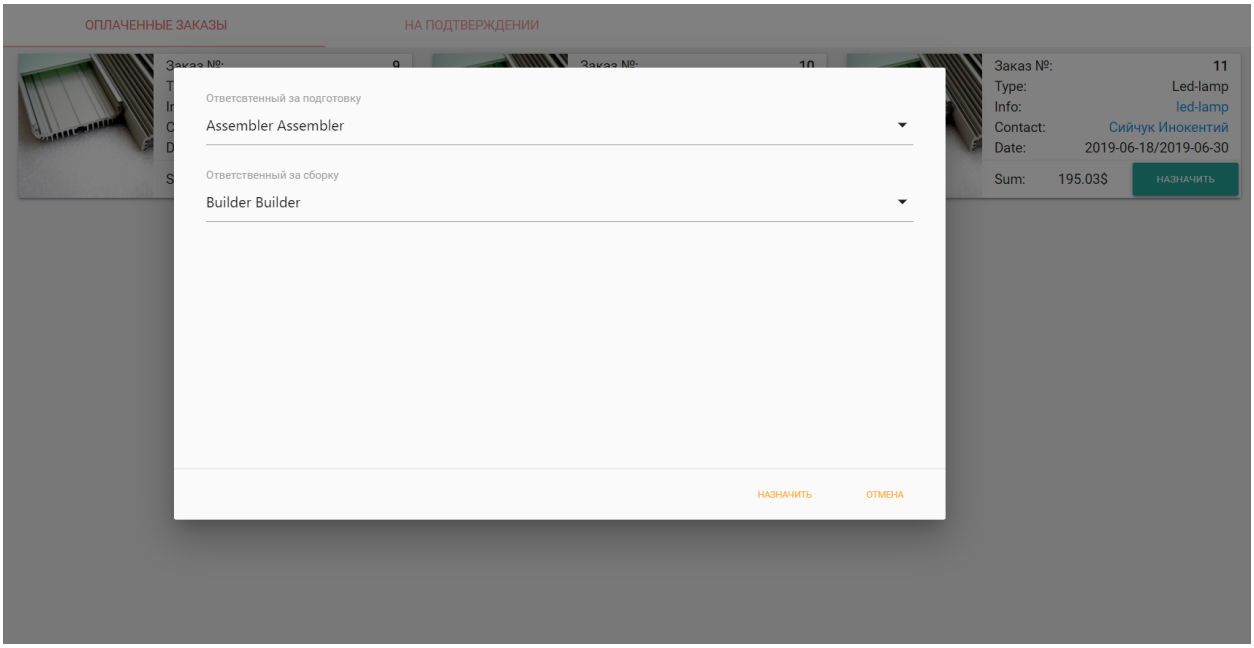


Рис. 4.15 Сторінка «Замовлення», призначення відповідальних персон.

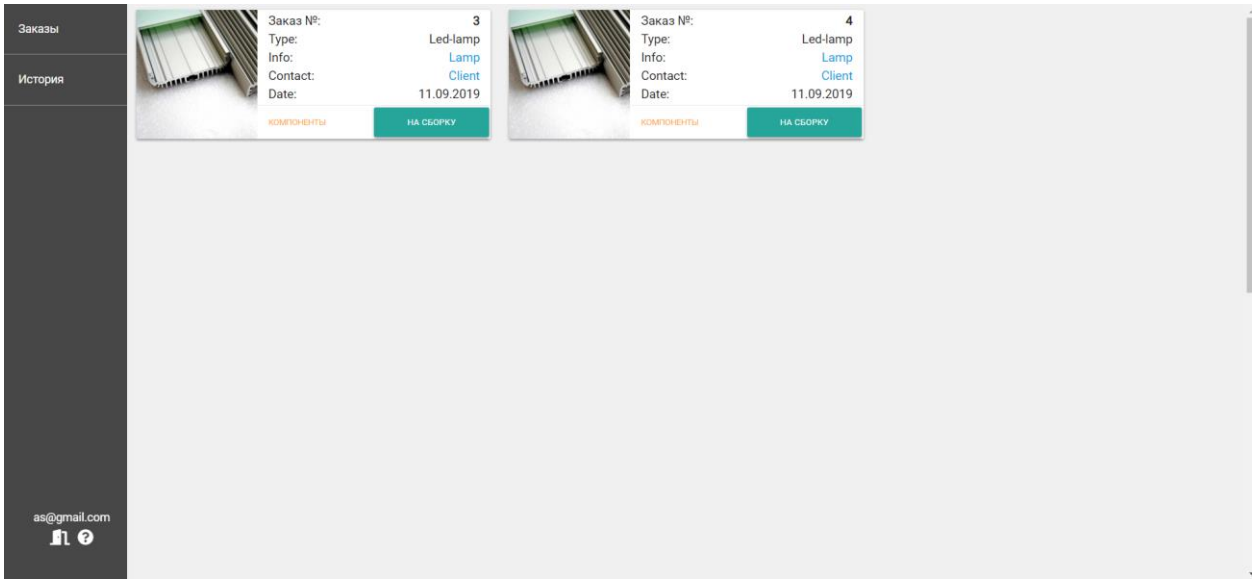


Рис. 4.16 Сторінка «Замовлення»

На рис. 4.16 зображена сторінка «Замовлення», яку можуть бачити користувачі системи, що мають роль «Відповідальний за підготовку». Користувач може відкрити нове модальне вікно натиснувши на кнопку «Компоненти», що зображено на рис. 4.17.

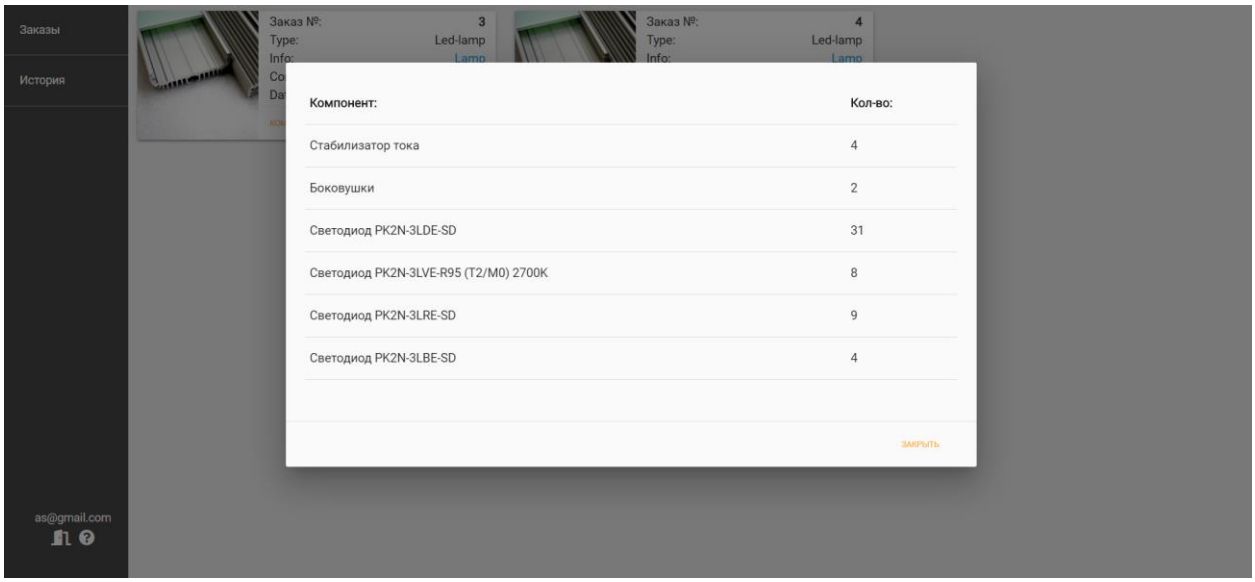


Рис. 4.17 Сторінка «Замовлення», список компонентів замовлення

На сторінці «Історія замовлень», що зображено на рис. 4.18 користувач може побачити на якому етапі знаходиться замовлення за яке він був відповідальним, та також переглянути список компонентів.

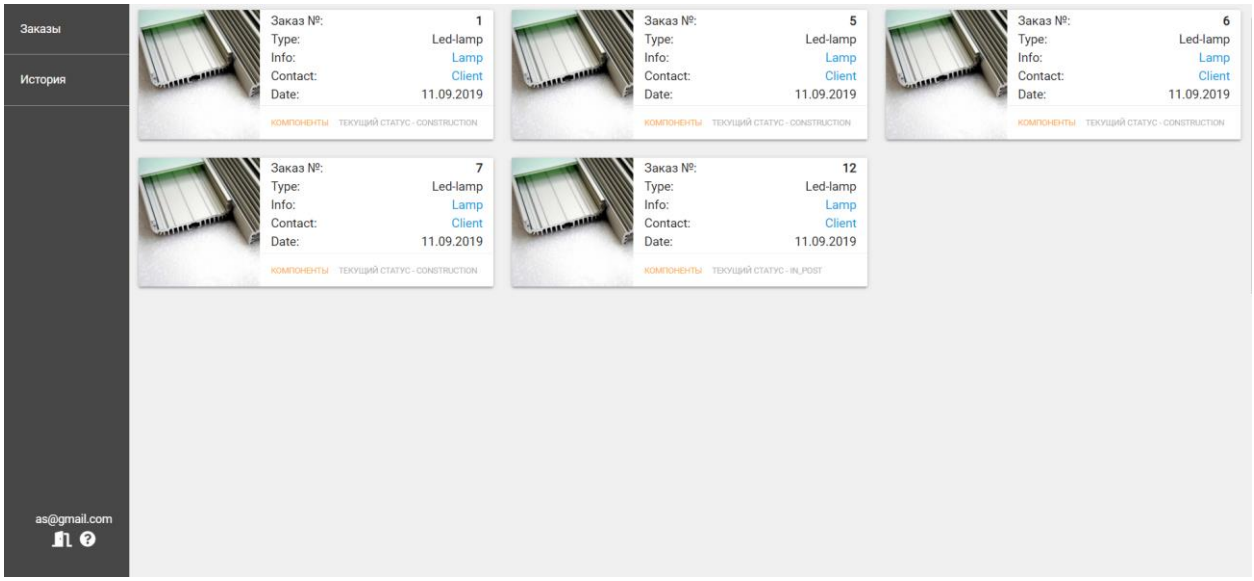


Рис. 4.18 Сторінка «Історія замовлень»

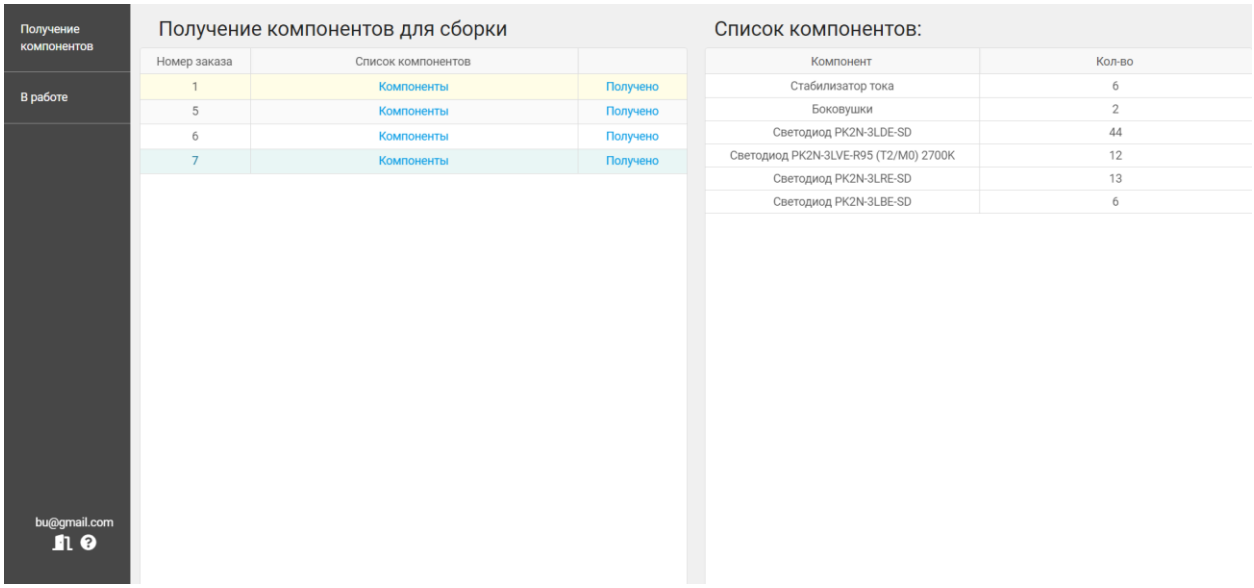


Рис. 4.19 Сторінка «Отримання компонентів»

На рис. 4.19 можна побачити сторінку «Отримання компонентів», на якій описані замовлення, які були призначені на відповідального за збірку до якої права має користувач с зроллю «Відповідальний за збірку». На даній сторінці користувач може переглянути список компонентів, та прийняти їх в роботу.

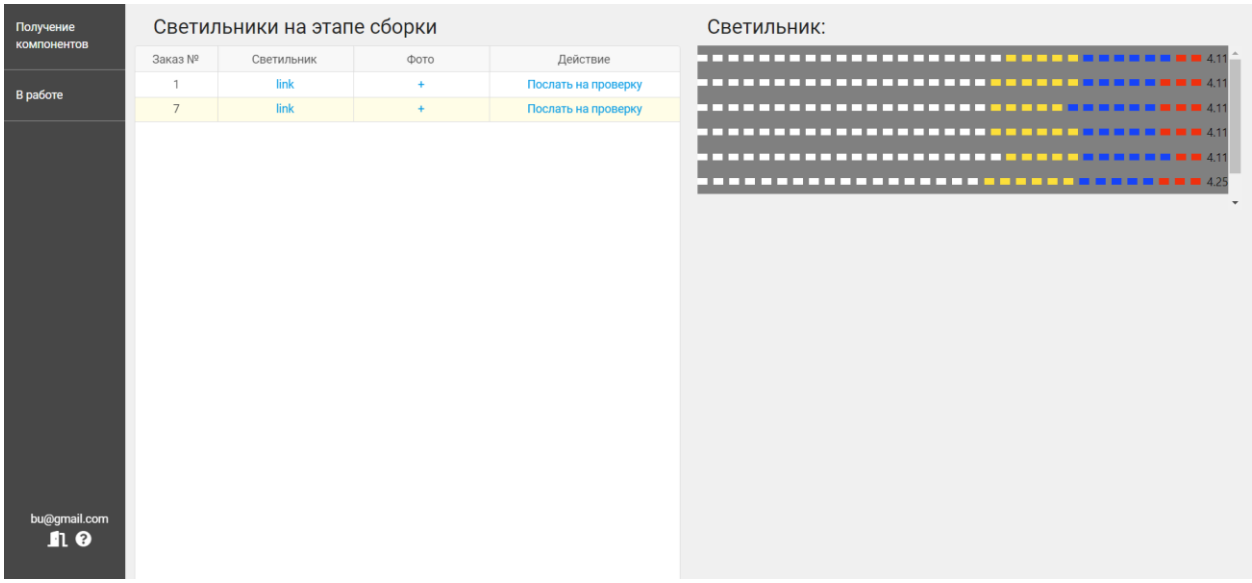


Рис. 4.20 Сторінка «Отримання компонентів», перегляд замовлення

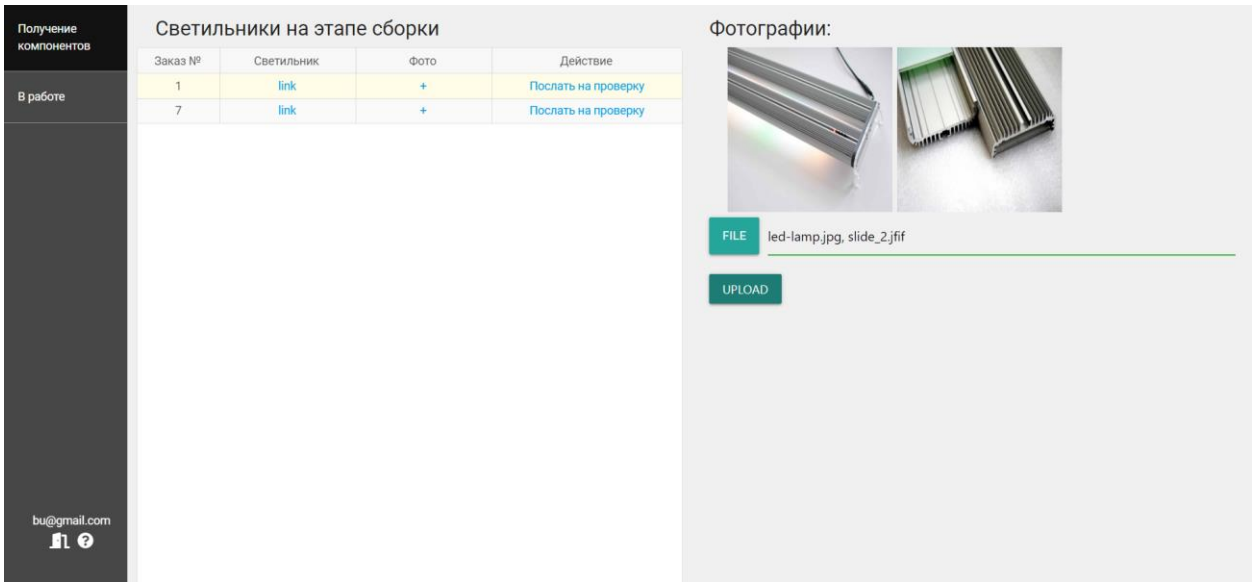


Рис. 4.21 Сторінка «Отримання компонентів», прикріплення фото

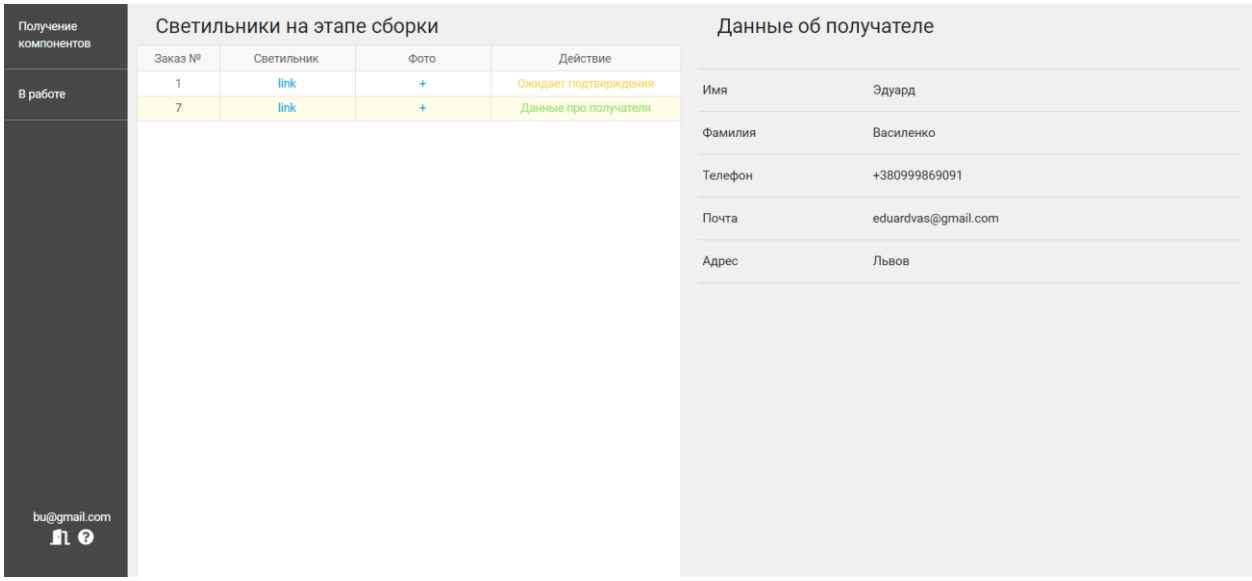


Рис. 4.22 Сторінка «Отримання компонентів», інформація про клієнта

На рис. 4.20 зображена сторінка «Замовлення в роботі», на якій можна відкрити деталі замовлення рис. 4.21. Також можна прикріпити фотографії замовлення для подальшого підтвердження адміністратором, що зображено на рис. 4.22, та отримати інформацію про клієнта, після того як замовлення було підтверджене.

Висновки до розділу 4

У даному розділі продемонстровано процес розгортання системи та проведено огляд інтерфейсу користувача. Описано інструкцію користування системою.

ВИСНОВКИ

У ході дипломного проекту було спроектовано і створено систему обробки замолень інтернет-магазину, що може бути використана для налагодження та спрощення механізму підготовки товару для продажу, обробки замовлення та доставки до клієнта. Було проаналізовано та виконано усі поставлені задачі.

Також було створено усю необхідну документацію, наведено керівництво користувача по роботі з системою.

Система має зручний та інтуїтивний інтерфейс, присутня взаємодія з базою даних.

Результати дослідження показали що робота має практичне застосування і є актуальним рішенням серед аналогів.

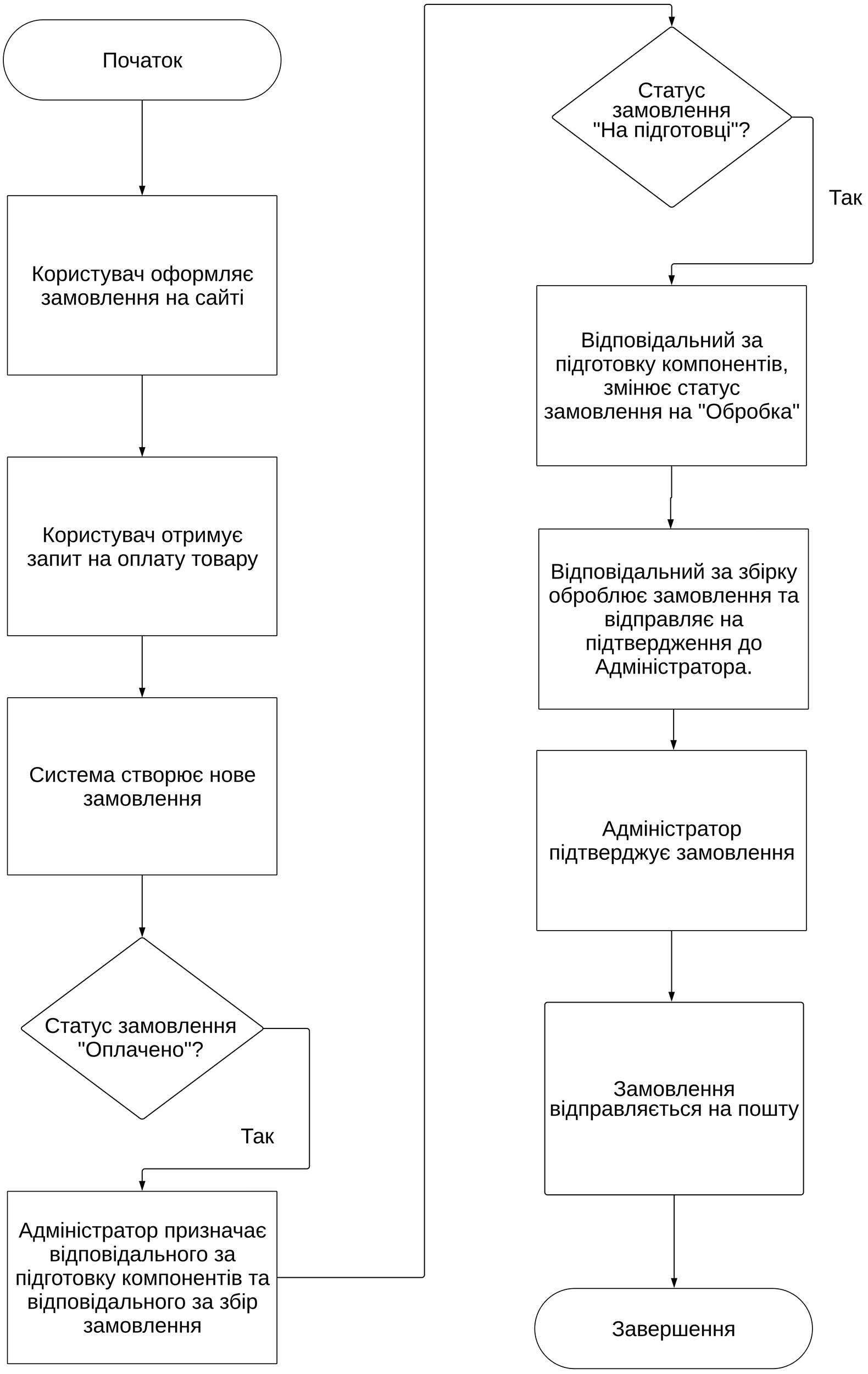
ПЕРЕЛІК ПОСИЛАНЬ

1. MVC [Електронний ресурс]: (Стаття) – Електрон. дан. (1 файл). – 2018. – Режим доступу: http://design-pattern.ru/patterns/mvc.html – Назва з екрана.
2. JavaScript [Електронний ресурс]: (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://uk.wikipedia.org/wiki/JavaScript– Назва з екрана.
3. Bcrypt [Електронний ресурс]: (Стаття) / Wikipedia. – Електрон. дан. (1 файл). – 2018. – Режим доступу: https://uk.wikipedia.org/wiki/Bcrypt. – Назва з екрана.

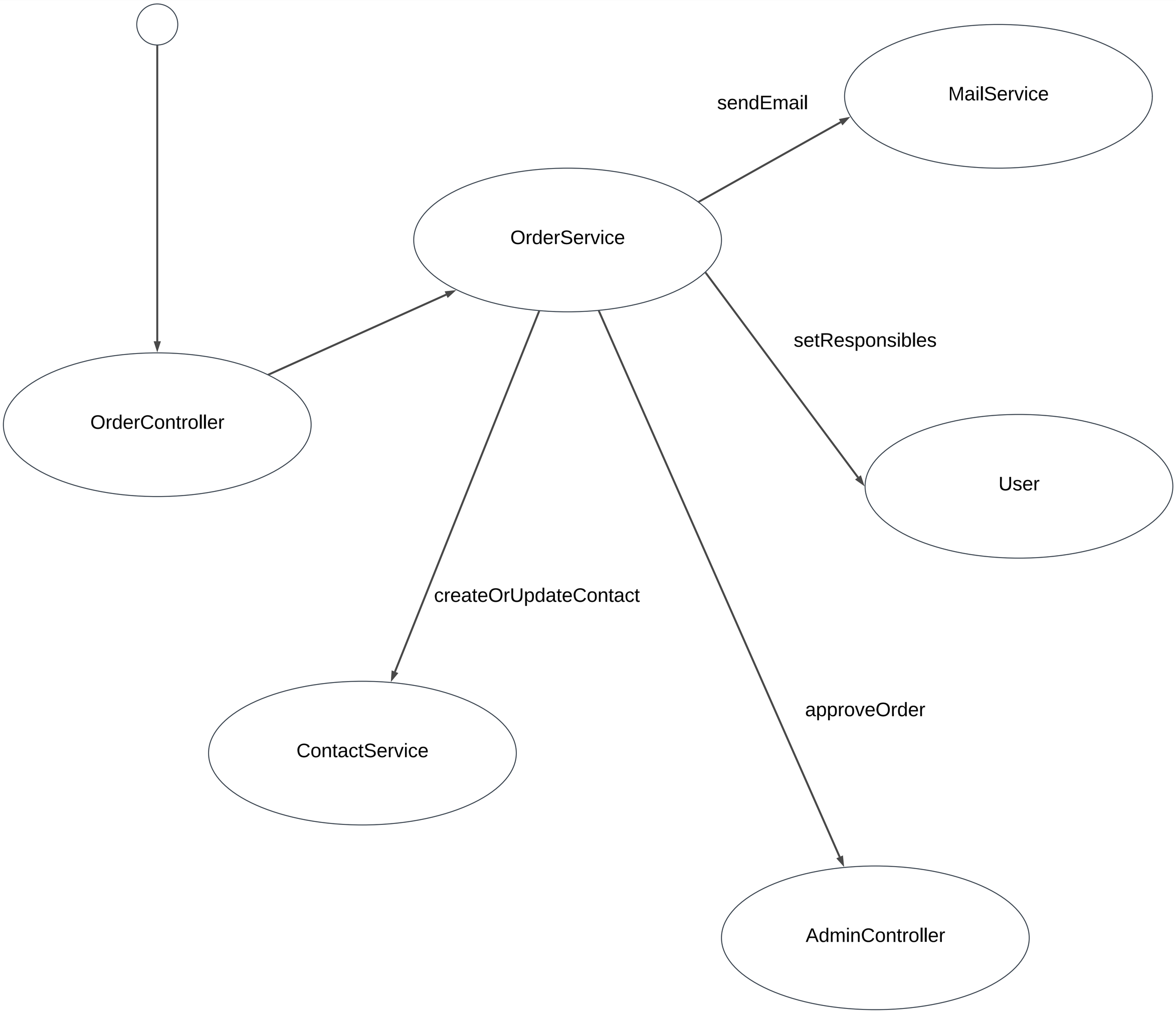
Графічні матеріали
до дипломного проекту

на тему: «Система обробки замовлень в Інтернет магазині»

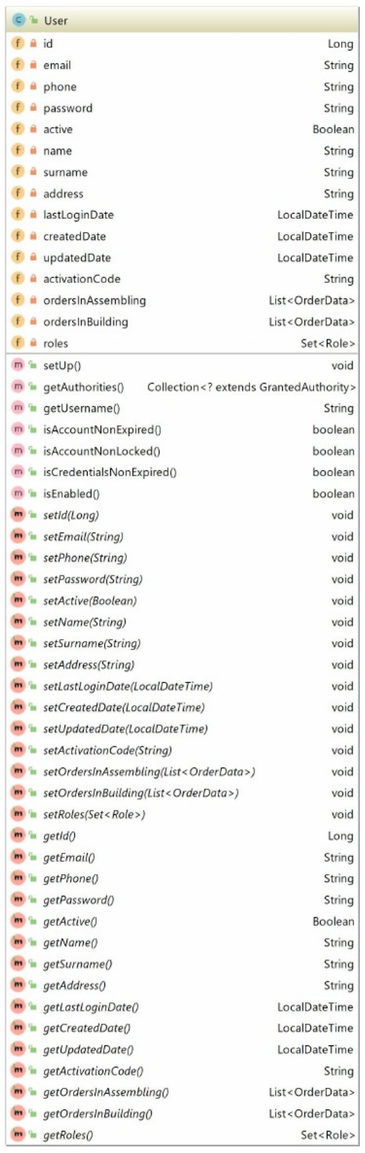
Київ – 2019



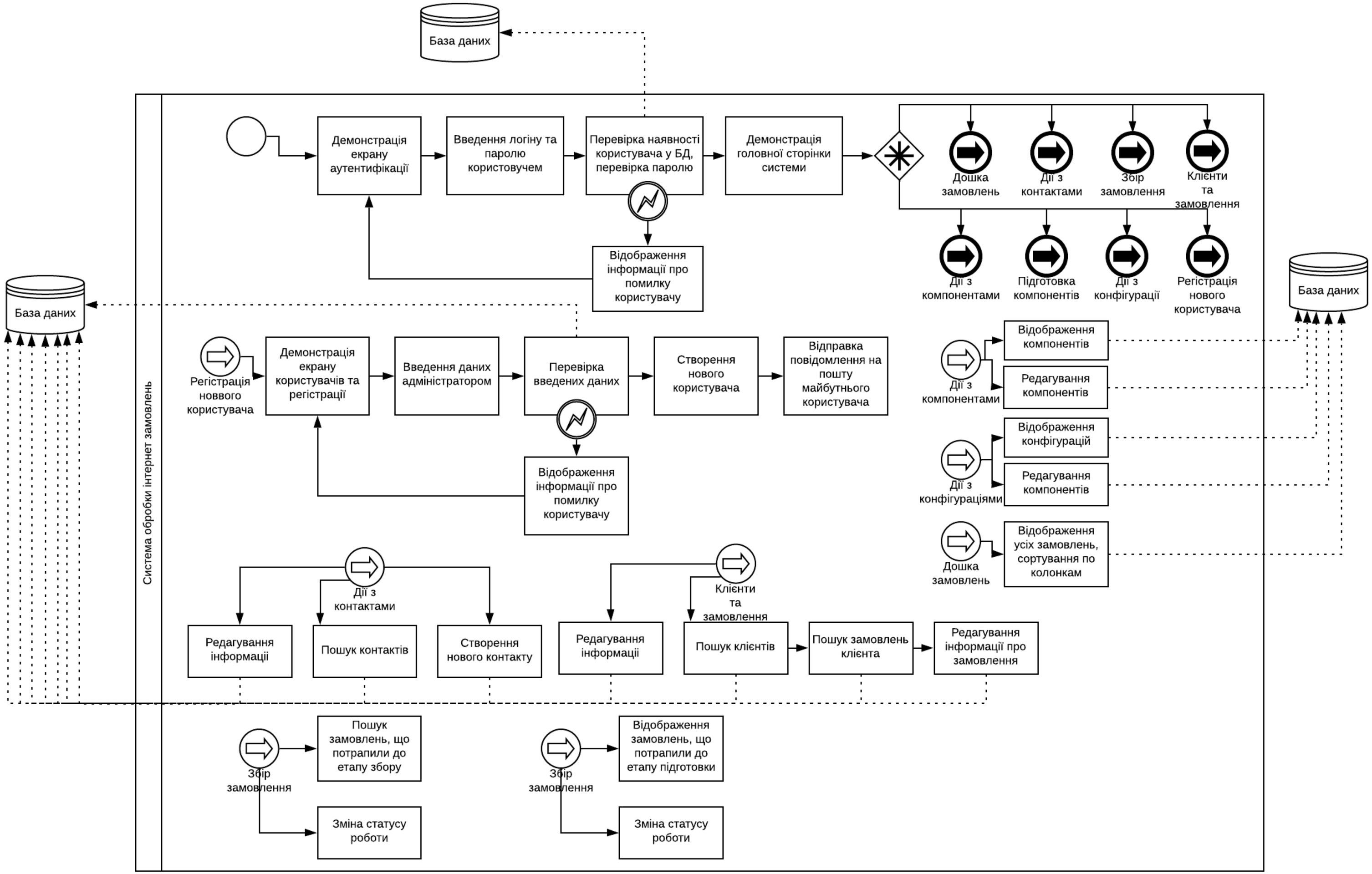
					ІАЛЦ.467200.004 Д1	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Чекіна Д.Р.			Система обробки замовлень Інтернет магазину. Схема алгоритму обробки замовлення.	Літ. / Арк. 1 / Аркушів 1
Перевір.		Регіда П. Г.				
						НТУУ „КПІ", ФІОТ, ІП-53
Н. Контр.		Сімоненко В.П.				
Затверд.		Стіренко С.Г.				



					ІАЛЦ.467200.005 Д2	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Чекіна Д.Р.			Система обробки замовлень Інтернет магазину. Схема взаємодії класів системи.	Літ. / Арк. 1 / Аркушів 1
Перевір.		Регіда П. Г.				
						НТУУ „КПІ", ФІОТ, ІП-53
Н. Контр.		Сімоненко В.П.				
Затверд.		Стіренко С.Г.				



					ІАЛЦ.467200.006 ДЗ	
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Чекіна Д.Р.			Система обробки замовлень Інтернет магазину. UML діаграма класів програми.	Літ. / Арк. 1 / Аркушів 1
Перевір.		Регіда П. Г.				
Н. Контр.		Сімоненко В.П.				НТУУ „КПІ", ФІОТ, ІП-53
Затверд.		Стіренко С.Г.				



					ІАЛЦ.461000.007 Д4
Змн.	Арк.	№ докум.	Підпис	Дата	
Розроб	.	Чекіна Д. Р.			Система обробки заказів Інтернет магазину. Схема бізнец процесів системи
Перевір.		Регіда П. Г.			
Н. Контр.		Сімоненко В.П.			
Затверд.		Стіренко С.Г.			

Літ.	Арк.	Аркушів
	1	1

НТУУ „КПІ", ФІОТ, ІП-53

ДОДАТОК А

Система формування звітів

Текст програми

ІАЛЦ.467200.007 А1

Листів 24

Київ – 2019

OrderCnotroller

```
package com.aquaconstructor.demo.endpoints.controllers;

import com.aquaconstructor.demo.endpoints.requests.LedLampOrderCreateRequest;
import com.aquaconstructor.demo.models.entity.OrderData;
import com.aquaconstructor.demo.models.entity.User;
import com.aquaconstructor.demo.models.enums.Role;
import com.aquaconstructor.demo.services.ImageService;
import com.aquaconstructor.demo.services.LedLampService;
import com.aquaconstructor.demo.services.OrderService;
import com.aquaconstructor.demo.services.UserService;
import lombok.AllArgsConstructor;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import javax.validation.Valid;

@Controller
@AllArgsConstructor
@RequestMapping("/order")
public class OrderController {

    private OrderService orderService;
    private LedLampService ledLampService;
    private UserService userService;
    private ImageService imageService;

    @GetMapping("/{ledLampId}")
    public ModelAndView getOrderPage(@PathVariable Long ledLampId) {
        ModelAndView maw = new ModelAndView("order");
        maw.addObject("ledLamp", ledLampService.getLedLampResponseById(ledLampId));
        LedLampOrderCreateRequest ledLampOrderCreateRequest = new LedLampOrderCreateRequest();
        ledLampOrderCreateRequest.setLedLampId(ledLampId);
        maw.addObject("orderRequest", ledLampOrderCreateRequest);
        return maw;
    }

    @PostMapping("/create")
    public ModelAndView createOrder(@Valid LedLampOrderCreateRequest request) {
        orderService.createOrder(request);
        return new ModelAndView("success");
    }

    @GetMapping
    public ModelAndView getOrders() {
        ModelAndView maw = new ModelAndView("orders");
        maw.addObject("payedOrders", orderService.getAllOrdersByStatus(OrderData.Status.PAYED));
        maw.addObject("assemblers", userService.getUsersByRole(Role.ASSEMBLER));
        maw.addObject("builders", userService.getUsersByRole(Role.BUILDER));
        maw.addObject("approveOrders", orderService.getApproveOrders());
//        maw.addObject("payedOrders", orderService.getAllOrdersByStatus(OrderData.Status.PAYED));
        return maw;
    }

    @GetMapping("/assemble")
    public ModelAndView getAssemblePage(@AuthenticationPrincipal User user) {
```

```
        ModelAndView maw = new ModelAndView("assemble");
        maw.addObject("activeOrders", orderService.getActiveOrdersForAssembler(user));
        return maw;
    }

    @GetMapping("/assemble/history")
    public ModelAndView getAssembleHistoryPage(@AuthenticationPrincipal User user) {
        ModelAndView maw = new ModelAndView("assemble-history");
        maw.addObject("notActiveOrders", orderService.getNotActiveOrdersForAssembler(user));
        return maw;
    }


    @GetMapping("/build-receive")
    public ModelAndView getBuildPage() {
        return new ModelAndView("build-receive");
    }

    @GetMapping("/build-in-work")
    public ModelAndView getBuildInWorkPage() {
        return new ModelAndView("build-in-work");
    }

    @GetMapping("/approve")
    public ModelAndView getBuildApprovePage() {
        return new ModelAndView("build-approve");
    }
}
```

ContactController

```
package com.aquaconstructor.demo.endpoints.controllers;

import com.aquaconstructor.demo.endpoints.requests.ContactCreateRequest;
import com.aquaconstructor.demo.services.ContactService;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.servlet.http.HttpSession;
import javax.validation.Valid;
import java.util.List;

@Controller
@AllArgsConstructor
@RequestMapping("/contact")
public class ContactController {

    private ContactService contactService;

    @PostMapping("/create")
    public String createContactFromLedLamp(@Valid ContactCreateRequest request,
                                           HttpSession httpSession) {
        request.setLedLampIds((List<Long>) httpSession.getAttribute("ledLampIds"));
        contactService.create(request);
        httpSession.removeAttribute("ledLampIds");
        return "redirect:../led-lamp";
```

```
    }
}
```

MaterialController

```
package com.aquaconstructor.demo.endpoints.controllers;

import com.aquaconstructor.demo.endpoints.requests.MaterialUpdateRequest;
import com.aquaconstructor.demo.models.entity.Material;
import com.aquaconstructor.demo.services.MaterialService;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller
@AllArgsConstructor
@RequestMapping("/material")
public class MaterialController {

    private MaterialService materialService;

    @GetMapping("/all")
    public @ResponseBody List<Material> getAllMaterials() {
        return materialService.getAllMaterials();
    }

    @PutMapping("/{id}")
    public @ResponseBody Material updateMaterial(@PathVariable Long id, @RequestBody
MaterialUpdateRequest request) {
        return materialService.updateMaterial(id, request);
    }
}
```

ResourceController

```
package com.aquaconstructor.demo.endpoints.controllers;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import java.io.*;

@Controller
public class ResourceController {

    @Value("classpath:app/json/particles.json")
    private Resource resourceFile;

    @GetMapping("/particles")
    public @ResponseBody String getParticles() throws IOException {
        File file = resourceFile.getFile();
        InputStream is = new FileInputStream(file);
        BufferedReader buf = new BufferedReader(new InputStreamReader(is));

        StringBuilder sb = new StringBuilder();
        buf.lines().forEach(sb::append);
        return sb.toString();
```

```
    }
}
```

ServiceController

```
package com.aquaconstructor.demo.endpoints.controllers;

import lombok.AllArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
@AllArgsConstructor
public class ServiceController {

    @GetMapping("/service")
    public ModelAndView getServicePage() {
        return new ModelAndView("service");
    }

    @GetMapping("/configurations")
    public ModelAndView getSettingsPage() {
        return new ModelAndView("configurations");
    }

    @GetMapping("/components")
    public ModelAndView getComponentsPage() {
        return new ModelAndView("components");
    }

    @GetMapping("/contacts")
    public ModelAndView getContactsPage() {
        return new ModelAndView("contacts");
    }

    @GetMapping("/board")
    private ModelAndView getBoard() {
        return new ModelAndView("board");
    }

    @GetMapping("/users")
    private ModelAndView getUsersPage() {
        return new ModelAndView("users");
    }


}
```

UserController

```
package com.aquaconstructor.demo.endpoints.controllers;

import com.aquaconstructor.demo.services.UserService;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
```

```
@AllArgsConstructor
@RequestMapping("/user")
public class UserController {

    private UserService userService;

    @GetMapping("/activation/{code}")
    public ModelAndView activateUser(@PathVariable String code) {
        userService.activate(code);
        return new ModelAndView("/login");
    }
}
```

ComponentApiController

```
package com.aquaconstructor.demo.endpoints.api;

import com.aquaconstructor.demo.converterservice.ConversionService;
import com.aquaconstructor.demo.endpoints.responses.ComponentResponse;
import com.aquaconstructor.demo.services.LedLampService;
import lombok.AllArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/v1/components")
@AllArgsConstructor
public class ComponentsApiController {

    private LedLampService ledLampService;
    private ConversionService conversionService;

    @GetMapping("/led-lamp/{ledLampId}")
    public ResponseEntity getComponentsByLedLampId(@PathVariable Long ledLampId) {
        return ResponseEntity.ok(conversionService.convertToList(
                ledLampService.getLedLampById(ledLampId).getComponents(), ComponentResponse.class));
    }
}
```

ConfigurationsApiController

```
package com.aquaconstructor.demo.endpoints.api;

import com.aquaconstructor.demo.endpoints.requests.ConfigurationUpdateRequest;
import com.aquaconstructor.demo.models.entity.Configuration;
import com.aquaconstructor.demo.services.ConfigurationService;
import lombok.AllArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/v1/configurations")
@AllArgsConstructor
public class ConfigurationsApiController {

    private ConfigurationService configurationService;

    @GetMapping
    public ResponseEntity getAllConfigs() {
```

```
        return ResponseEntity.ok(configurationService.getAllConfigs());
    }

    @PutMapping("/{id}")
    public ResponseEntity updateConfig(@PathVariable Long id,
                                       @RequestBody ConfigurationUpdateRequest request) {
        return ResponseEntity.ok(configurationService.updateConfig(id, request));
    }
}
```

ContactsApiController

```
package com.aquaconstructor.demo.endpoints.api;

import com.aquaconstructor.demo.endpoints.requests.ContactUpdateRequest;
import com.aquaconstructor.demo.models.entity.Contact;
import com.aquaconstructor.demo.services.ContactService;
import lombok.AllArgsConstructor;
import org.springframework.data.domain.Pageable;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/v1/contacts")
@AllArgsConstructor
public class ContactsApiController {

    private ContactService contactService;

    @PostMapping
    public ResponseEntity createContact(Contact request) {
        return ResponseEntity.ok(contactService.create(request));
    }

    @GetMapping
    public ResponseEntity getAllContact(Pageable pageable) {
        return ResponseEntity.ok(contactService.getContacts(pageable));
    }

    @PutMapping("/{id}")
    public ResponseEntity updateContact(@PathVariable Long id,
                                        @RequestBody ContactUpdateRequest request) {
        return ResponseEntity.ok(contactService.updateContact(id, request));
    }
}
```

ImagesApiController

```
package com.aquaconstructor.demo.endpoints.api;

import com.aquaconstructor.demo.services.ImageService;
import lombok.AllArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;

@RestController
@RequestMapping("/api/v1/images")
@AllArgsConstructor
public class ImagesApiController {
```

```
    private ImageService imageService;

    @PostMapping("/order/{orderId}")
    public ResponseEntity uploadImages(@PathVariable Long orderId,
                                       @RequestParam("files") MultipartFile[] files) throws IOException {
        return ResponseEntity.ok(imageService.uploadImages(orderId, files));
    }

    @GetMapping("/order/{orderId}")
    public ResponseEntity downloadImages(@PathVariable Long orderId) {
        return ResponseEntity.ok(imageService.download(orderId));
    }
}
```

MaterialApiController

```
package com.aquaconstructor.demo.endpoints.api;

import com.aquaconstructor.demo.endpoints.requests.MaterialUpdateRequest;
import com.aquaconstructor.demo.services.MaterialService;
import lombok.AllArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/v1/materials")
@AllArgsConstructor
public class MaterialsApiController {

    private MaterialService materialService;

    @GetMapping
    public ResponseEntity getAllMaterials() {
        return ResponseEntity.ok(materialService.getAllMaterials());
    }

    @PutMapping("/{id}")
    public ResponseEntity updateMaterial(@PathVariable Long id,
                                         @RequestBody MaterialUpdateRequest request) {
        return ResponseEntity.ok(materialService.updateMaterial(id, request));
    }
}
```

OrdersApiController

```
package com.aquaconstructor.demo.endpoints.api;

import com.aquaconstructor.demo.models.entity.User;
import com.aquaconstructor.demo.services.OrderService;
import lombok.AllArgsConstructor;
import org.springframework.data.domain.Pageable;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/v1/orders")
@AllArgsConstructor
public class OrdersApiController {

    private OrderService orderService;
```

```
    @GetMapping
    public ResponseEntity getOrders(Pageable pageable) {
        return ResponseEntity.ok(orderService.getOrders(pageable));
    }

    @GetMapping("/status/{status}")
    public ResponseEntity getOrders(@PathVariable String status) {
        return ResponseEntity.ok(orderService.getOrdersByStatus(status));
    }

    @PostMapping("/{id}/responsible")
    public ResponseEntity setResponsiblePersons(@PathVariable Long id,
                                                @RequestParam("assemblerId") Long assemblerId,
                                                @RequestParam("builderId") Long builderId) {
        orderService.setResponsiblePersons(id, assemblerId, builderId);
        return ResponseEntity.ok(null);
    }

    @PutMapping("/{id}/status/{status}")
    public ResponseEntity changeStatus(@PathVariable(name = "id") Long id, @PathVariable(name =
"status") String status) {
        orderService.changeStatus(id, status);
        return ResponseEntity.ok(null);
    }

    @GetMapping("/build/sub-status/{subStatus}")
    public ResponseEntity getOrdersForBuild(@AuthenticationPrincipal User user,
                                            @PathVariable String subStatus) {
        return ResponseEntity.ok(orderService.getOrdersForBuild(user, subStatus));
    }

    @GetMapping("/build/sub-status-inv/{subStatus}")
    public ResponseEntity getOrdersInvForBuild(@AuthenticationPrincipal User user,
                                               @PathVariable String subStatus) {
        return ResponseEntity.ok(orderService.getOrdersInvForBuild(user, subStatus));
    }

    @GetMapping("/{id}/build/contact-info")
    public ResponseEntity getOrderContactInfo(@PathVariable Long id) {
        return ResponseEntity.ok(orderService.getOrderContactInfo(id));
    }

    @PutMapping("/{id}/build/sub-status/{subStatus}")
    public ResponseEntity changeSubStatus(@PathVariable(name = "id") Long id,
                                          @PathVariable(name = "subStatus") String subStatus) {
        orderService.changeSubStatus(id, subStatus);
        return ResponseEntity.ok(null);
    }
}
```

SystemSettingsApiController

```
package com.aquaconstructor.demo.endpoints.api;

import com.aquaconstructor.demo.models.entity.User;
import com.aquaconstructor.demo.models.enums.Role;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import java.util.Set;

@RestController
@RequestMapping("/system-settings")
public class SystemSettingsApiController {

    @GetMapping("/route")
    public ResponseEntity getMainPage(@AuthenticationPrincipal User user) {
        Set<Role> roles = user.getRoles();
        if (roles.contains(Role.ADMIN)) {
            return ResponseEntity.ok("/board");
        } else if (roles.contains(Role.ASSEMBLER)) {
            return ResponseEntity.ok("/order/assemble");
        } else if (roles.contains(Role.BUILDER)) {
            return ResponseEntity.ok("/order/build-in-work");
        } else return null;
    }
}
```

UsersApiController

```
package com.aquaconstructor.demo.endpoints.api;

import com.aquaconstructor.demo.endpoints.requests.UserCreateRequest;
import com.aquaconstructor.demo.endpoints.requests.UserUpdateRequest;
import com.aquaconstructor.demo.services.UserService;
import lombok.AllArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;

@RestController
@RequestMapping("/api/v1/users")
@AllArgsConstructor
public class UsersApiController {

    private UserService userService;

    @GetMapping
    public ResponseEntity getAllUsers() {
        return ResponseEntity.ok(userService.getAllUsers());
    }

    @GetMapping("/{id}")
    public ResponseEntity getUserById(@PathVariable Long id) {
        return ResponseEntity.ok(userService.getUser(id));
    }

    @PutMapping("/{id}")
    public ResponseEntity updateUser(@PathVariable Long id,
                                     @RequestBody UserUpdateRequest userUpdateRequest) {
        return ResponseEntity.ok(userService.updateUser(id, userUpdateRequest));
    }

    @PostMapping
    public ResponseEntity createUser(@Valid @RequestBody UserCreateRequest request) {
        return ResponseEntity.ok(userService.createUser(request));
    }
}
```

Component

```
package com.aquaconstructor.demo.models.entity;

import com.aquaconstructor.demo.util.NumberFormatUtil;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;

@Entity
@Getter
@Setter
@NoArgsConstructor
public class Component {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private Double price;
    private Integer amount;
    private Double totalPrice;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "led_lamp_id")
    private LedLamp ledLamp;

    public Component(Material material, int amount, LedLamp ledLamp) {
        this.name = material.getName();
        this.price = NumberFormatUtil.positiveFormatDouble(material.getPrice());
        this.amount = amount;
        this.totalPrice = NumberFormatUtil.positiveFormatDouble(this.price * this.amount);
        this.ledLamp = ledLamp;
    }
}
```

Configuration

```
package com.aquaconstructor.demo.models.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Getter
@Setter
public class Configuration {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String atr;
    private String name;
```

```
    private String value;
    private String description;
}
```

Contact

```
package com.aquaconstructor.demo.models.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.time.LocalDateTime;
import java.util.List;

@Entity
@Getter
@Setter
public class Contact {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String phone;
    private String email;
    private String name;
    private String surname;
    private String address;

    private Boolean isContacted;
    private LocalDateTime createdDate;
    private LocalDateTime updatedDate;

    @OneToMany(mappedBy = "contact")
    private List<LedLamp> ledLamps;

    @OneToMany(mappedBy = "contact")
    private List<OrderData> orders;

    @PrePersist
    public void setUp() {
        this.createdDate = LocalDateTime.now();
        this.updatedDate = LocalDateTime.now();
    }
}
```

Image

```
package com.aquaconstructor.demo.models.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.time.LocalDateTime;

@Getter
@Setter
@Entity
public class Image {

    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String url;
    private LocalDateTime createdDate;

    @ManyToOne
    @JoinColumn(name = "order_id", referencedColumnName = "id")
    private OrderData orderData;

    public void setUp() {
        this.createdDate = LocalDateTime.now();
    }
}
```

Material

```
package com.aquaconstructor.demo.models.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Getter
@Setter
public class Material {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String atr;
    private String name;
    private Double price;
    private String description;
}
```

OrderData

```
package com.aquaconstructor.demo.models.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.time.LocalDateTime;
import java.util.List;

@Getter
@Setter
@Entity
public class OrderData {

    public enum Status {
        PRE_PAY,
        PAYED,
        ASSEMBLAGE,
        CONSTRUCTION,
```

```
        IN_POST,
        DONE
    }

    public enum SubStatus {
        ON_RECEIVE,
        IN_PROGRESS,
        ON_APPROVE,
        APPROVED
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private LocalDateTime createdDate;
    private LocalDateTime endDate;
    private Boolean isActive;

    @ManyToOne
    @JoinColumn(name = "contact_id")
    private Contact contact;

    @ManyToOne
    @JoinColumn(name = "responsible_assembler_id")
    private User responsibleAssembler;

    @ManyToOne
    @JoinColumn(name = "responsible_builder_id")
    private User responsibleBuilder;

    private String postStatus;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "led_lamp_id", referencedColumnName = "id")
    private LedLamp ledLamp;

    @Enumerated(EnumType.STRING)
    private Status status;

    @Enumerated(EnumType.STRING)
    private SubStatus subStatus;

    @OneToMany(mappedBy = "orderData")
    private List<Image> images;
}
```

User

```
package com.aquaconstructor.demo.models.entity;

import com.aquaconstructor.demo.models.enums.Role;
import lombok.Getter;
import lombok.Setter;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import javax.persistence.*;
import java.time.LocalDateTime;
import java.util.Collection;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
```

```java
@Entity
@Getter
@Setter
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String email;
    private String phone;
    private String password;
    private Boolean active;

    private String name;
    private String surname;
    private String address;

    private LocalDateTime lastLoginDate;
    private LocalDateTime createdDate;
    private LocalDateTime updatedDate;
    private String activationCode;

    @OneToMany(mappedBy = "responsibleAssembler")
    private List<OrderData> ordersInAssembling;

    @OneToMany(mappedBy = "responsibleBuilder")
    private List<OrderData> ordersInBuilding;

    @ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)
    @CollectionTable(name = "role", joinColumns = @JoinColumn(name = "user_id"))
    @Enumerated(EnumType.STRING)
    @Column(name = "role")
    private Set<Role> roles = new HashSet<>();

    @PrePersist
    public void setUp() {
        this.createdDate = LocalDateTime.now();
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return roles;
    }

    @Override
    public String getUsername() {
        return email;
    }

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

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

    @Override
    public boolean isCredentialsNonExpired() {
```

```
        return true;
    }

    @Override
    public boolean isEnabled() {
        return getActive();
    }
}
```

ContactService

```
package com.aquaconstructor.demo.services;

import com.aquaconstructor.demo.converterservice.ConversionService;
import com.aquaconstructor.demo.endpoints.requests.ContactCreateRequest;
import com.aquaconstructor.demo.endpoints.requests.ContactCreateRequestForLedLampOrder;
import com.aquaconstructor.demo.endpoints.requests.ContactUpdateRequest;
import com.aquaconstructor.demo.endpoints.responses.ContactResponse;
import com.aquaconstructor.demo.models.entity.Contact;
import com.aquaconstructor.demo.models.entity.LedLamp;
import com.aquaconstructor.demo.repositories.ContactRepository;
import lombok.AllArgsConstructor;
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Service
@AllArgsConstructor
public class ContactService {

    private ConversionService conversionService;
    private ContactRepository contactRepository;
    private LedLampService ledLampService;

    public void create(ContactCreateRequest request) {
        Contact contact = getContactByPhone(request.getPhone());
        if (contact == null) {
            contact = conversionService.convert(request, Contact.class);
        } else {
            contact.setIsContacted(false);
            contact.setCreatedDate(LocalDateTime.now());
        }

        Contact savedContact = contactRepository.saveAndFlush(contact);

        ArrayList<LedLamp> ledLamps = new
ArrayList<>(ledLampService.getLedLampsById(request.getLedLampIds()));
        ledLamps.forEach(ledLamp -> ledLamp.setContact(savedContact));
        ledLampService.saveAll(ledLamps);
    }

    public Contact createOrUpdate(ContactCreateRequestForLedLampOrder request) {
        Contact contact = getContactByPhone(request.getPhone());
        if (contact == null) {
            contact = conversionService.convert(request, Contact.class);
```

```
        } else {
            fillAdditionalInfo(contact, request);
            contact.setUpdatedDate(LocalDateTime.now());
            contact.setIsContacted(false);

        }
        return contactRepository.saveAndFlush(contact);
    }

    public Contact getContactByPhone(String phone) {
        return contactRepository.findByPhone(phone);
    }

    public Contact create(Contact contact) {
        return contactRepository.saveAndFlush(contact);
    }

    public Contact getContactById(Long id) {
        return getContactByIdOrThrowException(id);
    }

    private Contact getContactByIdOrThrowException(Long id) {
        return contactRepository.findById(id).orElseThrow(NullPointerException::new);
    }

    public List<ContactResponse> getContacts(Pageable pageable) {
        return conversionService.convertToList(getContactsPage(PageRequest.of(0, 100, Sort.Direction.DESC,
"id")), ContactResponse.class);
    }

    public List<Contact> getContactsPage(Pageable pageable) {
        return contactRepository.findAll(pageable).getContent();
    }

    public List<Contact> getAllContact(Pageable pageable) {
        return contactRepository.findAll();
    }

    public ContactResponse updateContact(Long id, ContactUpdateRequest request) {
        Contact contact = getContactById(id);
        Optional.ofNullable(request.getName()).ifPresent(contact::setName);
        Optional.ofNullable(request.getPhone()).ifPresent(contact::setPhone);
        Optional.ofNullable(request.getEmail()).ifPresent(contact::setEmail);
        Optional.ofNullable(request.getIsContacted()).ifPresent(contact::setIsContacted);
        return conversionService.convert(contactRepository.saveAndFlush(contact), ContactResponse.class);
    }

    private void fillAdditionalInfo(Contact contact, ContactCreateRequestForLedLampOrder request) {
        Optional.ofNullable(request.getAddress()).ifPresent(contact::setAddress);
        Optional.ofNullable(request.getEmail()).ifPresent(contact::setEmail);
        Optional.ofNullable(request.getSurname()).ifPresent(contact::setSurname);
    }
}
```

EmailService

```
package com.aquaconstructor.demo.services;

import lombok.AllArgsConstructor;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;
```

```
@Service
@AllArgsConstructor
public class EmailService {

    private JavaMailSender javaMailSender;

    public void send(String to, String subject, String message) {
        SimpleMailMessage simpleMessage = new SimpleMailMessage();
        simpleMessage.setTo(to);
        simpleMessage.setSubject(subject);
        simpleMessage.setText(message);
        javaMailSender.send(simpleMessage);
    }
}
```

ImageService

```
package com.aquaconstructor.demo.services;

import com.aquaconstructor.demo.endpoints.responses.ImageResponse;
import com.aquaconstructor.demo.models.entity.Image;
import com.aquaconstructor.demo.models.entity.OrderData;
import com.aquaconstructor.demo.repositories.ImageRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

@Service
@AllArgsConstructor
public class ImageService {

    private static final String IMAGE_FOLDER = "/chekina/studying/aqua-
constructor/src/main/resources/app/images/";

    private OrderService orderService;
    private ImageRepository imageRepository;

    public List<ImageResponse> uploadImages(Long orderId, MultipartFile[] files) throws IOException {
        OrderData order = orderService.getOrderDataOrThrowException(orderId);

        List<Image> images = new ArrayList<>();
        for (MultipartFile file : files) {
            if (file != null && file.getSize() > 0) {
                byte[] bytes = file.getBytes();
                LocalDateTime now = LocalDateTime.now();
                String imagePath = IMAGE_FOLDER + file.getOriginalFilename();
                Path path = Paths.get(imagePath);
                Files.write(path, bytes);
                Image image = new Image();
                image.setName(file.getName() + LocalDateTime.now());
                image.setOrderData(order);
                image.setUrl("/images/" + file.getOriginalFilename());
```

```
                images.add(image);
            }
        }

        imageRepository.saveAll(images);
        return download(orderId);
    }

    public List<ImageResponse> download(Long orderId) {
        OrderData order = orderService.getOrderDataOrThrowException(orderId);
        List<Image> images = order.getImages();
        images.sort((x1, x2) -> (int) (x2.getId() - x1.getId()));
        List<ImageResponse> response = new ArrayList<>();
        for (Image image : images) {
            ImageResponse imageResponse = new ImageResponse();
            imageResponse.setUrl(image.getUrl());
            response.add(imageResponse);
        }
        return response;
    }
}
```

MaterialService

```
package com.aquaconstructor.demo.services;

import com.aquaconstructor.demo.endpoints.requests.MaterialUpdateRequest;
import com.aquaconstructor.demo.models.entity.Material;
import com.aquaconstructor.demo.repositories.MaterialRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
@AllArgsConstructor
public class MaterialService {

    private MaterialRepository materialRepository;

    public Material getMaterialByIdOrThrowException(Long id) {
        return materialRepository.findById(id).orElseThrow(NullPointerException::new);
    }

    public Map<String, Material> getMaterialMap() {
        return getAllMaterials().stream().collect(Collectors.toMap(Material::getAtr, c -> c));
    }

    public List<Material> getAllMaterials() {
        return materialRepository.findAll();
    }

    public Material updateMaterial(Long id, MaterialUpdateRequest request) {
        Material material = getMaterialByIdOrThrowException(id);
        Optional.ofNullable(request.getName()).ifPresent(material::setName);
        Optional.ofNullable(request.getPrice()).ifPresent(material::setPrice);
        Optional.ofNullable(request.getDescription()).ifPresent(material::setDescription);
        return materialRepository.saveAndFlush(material);
```

```
    }
}
```

OrderService

```
package com.aquaconstructor.demo.services;

import com.aquaconstructor.demo.converterservice.ConversionService;
import com.aquaconstructor.demo.endpoints.dtos.ApproveOrder;
import com.aquaconstructor.demo.endpoints.requests.LedLampOrderCreateRequest;
import com.aquaconstructor.demo.endpoints.responses.ContactResponse;
import com.aquaconstructor.demo.endpoints.responses.ImageResponse;
import com.aquaconstructor.demo.endpoints.responses.OrderDataResponse;
import com.aquaconstructor.demo.endpoints.responses.OrderDataResponseForAssembler;
import com.aquaconstructor.demo.models.entity.*;
import com.aquaconstructor.demo.repositories.OrderRepository;
import lombok.AllArgsConstructor;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import static com.aquaconstructor.demo.util.Constants.WORKING_DAYS;

@Service
@AllArgsConstructor
public class OrderService {

    private OrderRepository orderRepository;
    private ConversionService conversionService;
    private LedLampService ledLampService;
    private ConfigurationService configurationService;
    private ContactService contactService;
    private UserService userService;

    public void createOrder(LedLampOrderCreateRequest request) {
        LedLamp ledLamp = ledLampService.getLedLampById(request.getLedLampId());

        Contact contact = contactService.createOrUpdate(request.getContact());

        Map<String, Configuration> configMap = configurationService.getConfigMap();

        OrderData orderData = new OrderData();
        orderData.setIsActive(true);
        orderData.setLedLamp(ledLamp);
        orderData.setContact(contact);
        orderData.setCreatedDate(LocalDateTime.now());

orderData.setEndDate(LocalDateTime.now().plusDays(Long.parseLong(configMap.get(WORKING_DAYS).g
etValue())));
        orderData.setStatus(OrderData.Status.PRE_PAY);
        orderRepository.saveAndFlush(orderData);

        ledLamp.setContact(contact);
        ledLampService.save(ledLamp);
    }

    public OrderData getOrderDataOrThrowException(Long id) {
```

```
        return orderRepository.findById(id).orElseThrow(NullPointerException::new);
    }

    public void setResponsiblePersons(Long id, Long assemblerId, Long builderId) {
        OrderData order = getOrderDataOrThrowException(id);
        order.setResponsibleAssembler(userService.getUserById(assemblerId));
        order.setResponsibleBuilder(userService.getUserById(builderId));
        order.setStatus(OrderData.Status.ASSEMBLAGE);
        orderRepository.saveAndFlush(order);
    }

    public List<OrderDataResponse> getOrders(Pageable pageable) {
        List<OrderData> content = getOrdersPage(pageable);
        return conversionService.convertToList(content, OrderDataResponse.class);
    }

    private List<OrderData> getOrdersPage(Pageable pageable) {
        return orderRepository.findAll(pageable).getContent();
    }

    public List<OrderDataResponse> getOrdersByStatus(String status) {
        List<OrderData> allOrders = getAllOrders();
        List<OrderData> response = allOrders.stream().filter(orderData ->
orderData.getStatus().name().equalsIgnoreCase(status)).collect(Collectors.toList());
        return conversionService.convertToList(response, OrderDataResponse.class);
    }

    public List<OrderData> getAllOrders() {
        return orderRepository.findAll();
    }

    public List<OrderData> getAllOrdersByResponsibleId(Long userId) {
        return orderRepository.findAllByResponsibleAssemblerId(userId);
    }

    public List<OrderData> getAllOrdersByBuilderId(Long userId) {
        return orderRepository.findAllByResponsibleBuilderId(userId);
    }

    public List<OrderDataResponseForAssembler> getActiveOrdersForAssembler(User user) {
        List<OrderData> allOrders = getAllOrdersByResponsibleId(user.getId());
        List<OrderDataResponseForAssembler> assemblageData = new ArrayList<>();
        for (OrderData order : allOrders) {
            if (order.getIsActive() != null && order.getIsActive() &&
order.getStatus().equals(OrderData.Status.ASSEMBLAGE)) {
                assemblageData.add(conversionService.convert(order, OrderDataResponseForAssembler.class));
            }
        }
       return assemblageData;
    }

    public List<OrderDataResponseForAssembler> getNotActiveOrdersForAssembler(User user) {
        List<OrderData> allOrders = getAllOrdersByResponsibleId(user.getId());
        List<OrderDataResponseForAssembler> assemblageData = new ArrayList<>();
        for (OrderData order : allOrders) {
            if (order.getIsActive() != null && order.getIsActive() &&
!order.getStatus().equals(OrderData.Status.ASSEMBLAGE) ) {
                assemblageData.add(conversionService.convert(order, OrderDataResponseForAssembler.class));
            }
        }
        return assemblageData;
    }
```

```
    public void changeStatus(Long id, String status) {
        OrderData order = getOrderDataOrThrowException(id);
        if (status.equalsIgnoreCase(OrderData.Status.CONSTRUCTION.name())) {
            order.setSubStatus(OrderData.SubStatus.ON_RECEIVE);
        }
        order.setStatus(OrderData.Status.valueOf(status));
        orderRepository.saveAndFlush(order);
    }

    public List<OrderDataResponse> getOrdersForBuild(User user, String subStatus) {
        List<OrderData> orders = getAllOrdersByBuilderId(user.getId());
        List<OrderDataResponse> response = new ArrayList<>();
        for (OrderData order : orders) {
            if (order.getIsActive() != null && order.getIsActive()
                    && order.getStatus().equals(OrderData.Status.CONSTRUCTION) &&
order.getSubStatus().equals(OrderData.SubStatus.valueOf(subStatus))) {
                response.add(conversionService.convert(order, OrderDataResponse.class));
            }
        }
        return response;
    }

    public void changeSubStatus(Long id, String subStatus) {
        OrderData order = getOrderDataOrThrowException(id);
        order.setSubStatus(OrderData.SubStatus.valueOf(subStatus));
        orderRepository.saveAndFlush(order);
    }

    public List<OrderDataResponse> getOrdersInvForBuild(User user, String subStatus) {
        List<OrderData> orders = getAllOrdersByBuilderId(user.getId());
        List<OrderDataResponse> response = new ArrayList<>();
        for (OrderData order : orders) {
            if (order.getIsActive() != null && order.getIsActive()
                    && order.getStatus().equals(OrderData.Status.CONSTRUCTION) &&
!order.getSubStatus().equals(OrderData.SubStatus.valueOf(subStatus))) {
                response.add(conversionService.convert(order, OrderDataResponse.class));
            }
        }
        return response;
    }

    public ContactResponse getOrderContactInfo(Long id) {
        OrderData order = getOrderDataOrThrowException(id);
        return conversionService.convert(order.getContact(), ContactResponse.class);
    }

    public List<OrderDataResponse> getAllOrdersByStatus(OrderData.Status status) {
        List<OrderData> allOrders = getAllOrders();
        List<OrderData> response = allOrders.stream().filter(orderData ->
orderData.getStatus().equals(status)).collect(Collectors.toList());
        return conversionService.convertToList(response, OrderDataResponse.class);
    }

    public List<ApproveOrder> getApproveOrders() {
        List<ApproveOrder> response = new ArrayList<>();
        List<OrderData> allOrders = getAllOrders();
        for (OrderData orderData: allOrders) {
            if (orderData.getStatus().equals(OrderData.Status.CONSTRUCTION) &&
orderData.getSubStatus().equals(OrderData.SubStatus.ON_APPROVE)) {
                ApproveOrder approveOrder = new ApproveOrder();
                approveOrder.setId(orderData.getId());
```

```
            List<Image> images = orderData.getImages();
            List<ImageResponse> imageResponses = new ArrayList<>();
            if (images != null && images.size() != 0) {
                int i = 0;
                for (Image image : images) {
                    ImageResponse imageResponse = new ImageResponse();
                    imageResponse.setUrl(image.getUrl());
                    imageResponses.add(imageResponse);
                    if (++i == 3) {
                        break;
                    }
                }
                approveOrder.setImages(imageResponses);
            } else {
                approveOrder.setImages(new ArrayList<>());
            }
            response.add(approveOrder);
        }
    }
    return response;
  }
}
```

UserService

```
package com.aquaconstructor.demo.services;

import com.aquaconstructor.demo.converterservice.ConversionService;
import com.aquaconstructor.demo.endpoints.requests.UserCreateRequest;
import com.aquaconstructor.demo.endpoints.requests.UserUpdateRequest;
import com.aquaconstructor.demo.endpoints.responses.UserResponse;
import com.aquaconstructor.demo.models.entity.User;
import com.aquaconstructor.demo.models.enums.Role;
import com.aquaconstructor.demo.repositories.UserRepository;
import lombok.AllArgsConstructor;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;
import java.util.UUID;
import java.util.stream.Collectors;

@Service
@AllArgsConstructor
public class UserService implements UserDetailsService {

    private UserRepository userRepository;
    private ConversionService conversionService;
    private EmailService emailService;

    @Override
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        return userRepository.findUserByEmail(email);
    }

    public void save(User user) {
```

```
        userRepository.saveAndFlush(user);
    }

    public List<UserResponse> getAllUsers() {
        return conversionService.convertToList(getAll(), UserResponse.class);
    }

    private List<User> getAll() {
        return userRepository.findAll();
    }

    public UserResponse getUser(Long id) {
        return conversionService.convert(getUserByIdOrThrowException(id), UserResponse.class);
    }

    public User getUserById(Long id) {
        return getUserByIdOrThrowException(id);
    }

    private User getUserByIdOrThrowException(Long id) {
        return userRepository.findById(id).orElseThrow(NullPointerException::new);
    }

    public UserResponse updateUser(Long id, UserUpdateRequest request) {
        User user = getUserByIdOrThrowException(id);
        Optional.ofNullable(request.getName()).ifPresent(user::setName);
        Optional.ofNullable(request.getSurname()).ifPresent(user::setSurname);
        Optional.ofNullable(request.getPhone()).ifPresent(user::setPhone);
        Optional.ofNullable(request.getAddress()).ifPresent(user::setAddress);
        Optional.ofNullable(request.getEmail()).ifPresent(user::setEmail);
        user.setUpdatedDate(LocalDateTime.now());
        return conversionService.convert(userRepository.saveAndFlush(user), UserResponse.class);
    }

    public UserResponse createUser(UserCreateRequest request) {
        User user = conversionService.convert(request, User.class);
        String activationCode = UUID.randomUUID().toString();
        user.setActivationCode(activationCode);
        user.setPassword(new BCryptPasswordEncoder().encode("123456"));
        User userFromDb = userRepository.saveAndFlush(user);
//        emailService.send(user.getEmail(), "Authentication", "Please use this link to activate your account:" +
//                " http://localhost:8080/user/activation/" + activationCode);
        return conversionService.convert(userFromDb, UserResponse.class);
    }

    public void activate(String code) {
        User user = userRepository.findUserByActivationCode(code);
        user.setActive(true);
        userRepository.saveAndFlush(user);
    }

    public List<UserResponse> getUsersByRole(Role role) {
        List<User> allUsers = getAll();
        return conversionService.convertToList(
                allUsers.stream().filter(user -> user.getRoles().contains(role)).collect(Collectors.toList()),
UserResponse.class);
    }
}
```