Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема: розробка веб-застосунку для замовлення пошиття одягу (веб-Ательє)

Спеціальність: 121 Інженерія програмного забезпечення

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДП.ІПЗ-26.ПЗ
(позначення)

Рецензент
доц. Лазарович І.М.
(посада) (підпис) (дата) (розшифровка підпису)

Студент
ІПЗ-41 Уханський М.Д.
(шифр групи) (підпис) (дата) (розшифровка підпису)

Нормоконтролер
доц. Лазарович І.М.
(посада) (підпис) (дата) (розшифровка підпису)

Керівник дипломного проекту
зав.каф. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

зав.каф. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

2020

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики Кафедра інформаційних технологій
Спеціальність 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри Козленко М.І.
_____
„_____”_____ 20___р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Уханському Миколі Дмитровичу
(прізвище, ім’я, по батькові студента)

1. Тема проекту Розробка веб-застосунку для замовлення пошиття одягу (веб-Ательє)
затверджена розпорядженням по факультету математики та інформатики від „25” жовтня 2019 р. №7
2. Термін здачі студентом закінченого проекту 22 травня 2020 р.
3. Вихідні дані до дипломного проекту: перелік одягу, з яким працює ательє, вимоги до обчислення розміру одягу, технологія програмування серверної частини – Ruby on Rails, технології розробки клієнтської частини – HTML, CSS, Bootstrap 4, React JS.
4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати)
   1. Поняття та особливості ательє
   2. Проектування та моделювання веб-сайту для замовлення пошиття одягу
   3. Розробка дизайну та функціоналу веб-ательє
   4. Бізнес план веб-сайту ательє
5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень) 1. Титульний аркуш; 2. Мета веб-сайту для замовлення пошиття одягу та його особливості; 3. Актуальність розробки веб-сайту замовлення пошиття одягу; 4. Використання спеціальних рішень, для стабільної роботи веб-ательє; 5. Архітектура проекту ER-діаграма; 6. Діаграма архітектури авторизації користувача на веб-сайті для ательє; 7. Діаграма взаємодії з веб-аплікацією ательє замовника та адміністратора; 8. Економічна частина розробки; 9. Висновки
6. Дата видачі завдання 11.09.2019

Керівник _____ Козленко М.І.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Уханський М.Д.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
1. Обґрунтування актуальності, формулювання мети, завдання, предмету та об'єкту дослідження роботи	30.11.2019	Виконав
2. Опрацювання джерел з теми роботи	15.12.2019	Виконав
3. Поняття та особливості ательє	15.01.2019	Виконав
4. Проектування та моделювання веб-сайту для замовлення пошиття одягу	31.01.2020	Виконав
5. Розробка дизайну та функціоналу веб-ательє	20.02.2020	Виконав
6. Бізнес план веб-сайту ательє	20.03.2020	Виконав
7. Виправлення зауважень попередніх звітів. Підготовка вступу і висновків	20.04.2020	Виконав
8. Оформлення роботи згідно вимог	10.05.2020	Виконав

Студент _____ Уханський М.Д.
(підпис) (розшифровка підпису)

Керівник проекту _____ Козленко М.І.
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 82 сторінки (без додатків), 39 рисунків, 2 таблиці, 28 джерел, 2 додатки на 14 сторінках.

Ключові слова: АТЕЛЬЄ, RUBY, ФРЕЙМВОРК, RUBY ON RAILS (ROR), СУБД (БД), POSTGRESQL, ORM, MVC, AASM, API, GEM, ВАЛІДАЦІЯ, ООП, РЕНДЕРИНГ, CSS, HTML, BOOTSTRAP.

Об`єктом дослідження є застосунок для створення замовлень на пошиття одягу, його функціонал, можливості та особливості.

Мета роботи: спроектувати та розробити клієнтську та серверну частину сайту для ательє, який прийматиме замовлення онлайн, буде передбачена можливість обчислення розміру одягу, та комфортний інтерфейс.

Стислий опис тексту пояснювальної записки: у даному дипломному проекті описано розробку веб-сайту на мові програмування Ruby, та фреймворку Ruby on Rails. Для збереження даних було використано бази даних PostgreSQL, для створення калькулятору використовувались технології React JS, а для створення зовнішнього вигляду використовувались html, scss, bootstrap.

ABSTRACT

Explanatory note: 82 pages (without appendix), 39 figures, 2 tables, 28 references, 2 appendixes on 14 pages.

Key words: ATELIER, RUBY, FRAMEWORK, RUBY ON RAILS (RoR), DBMS (DB), POSTGRESQL, ORM, MVC, AASM, API, GEM, VALIDATION, OOP, RENDERING, CSS, HTML, BOOTSTRAP.

Object of study: is the application for creating tailoring orders, its functions, abilities and peculiarities.

Aim of the thesis: to design and create customer base and server application of the atelier's website that will take orders online. The possibility of calculating clothing size and comfortable interface will be provided.

Short description of the explanatory note's text: in the given thesis, the development of websites on the basis of the Ruby programming language and Ruby on Nails framework was described. Database PostgeSQL was used for the data storaging; ReactJS technologies were used for creating computation; html, scss bootstrap were used for creating the outer look.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

ООП (Об'єктно-орієнтоване програмування) – одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція.

AASM – бібліотека для додавання станів до класів Ruby.

API – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено - це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

CSS (Cascading Style Sheets) – це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду.

Gem (гем) – пакет (файл) з бібліотекою або додатком, що має стандартизований вигляд і розташований в сховище в мережі.

HTML (Hypertext Markup Language) – це мова тегів, якою пишуться гіпертекстові документи для мережі Інтернет Веб-браузери отримують HTML-документи з веб-сервера або з локальної пам'яті і передають документи в мультимедійні веб-сторінки.

MVC (Model, View, Controller) – передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) 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.

ORM – технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

PostgreSQL – об'єктно-реляційна система управління базами даних (СУБД), або просто бази даних (БД). Є альтернативою як комерційним

СУБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СУБД з відкритим кодом (MySQL, Firebird, SQLite).

Ruby on Rails (RoR) – об'єктно-орієнтований програмний каркас (фреймворк) для створення веб-застосунків, написаний на мові програмування Ruby.

ВСТУП

З давна одяг повинен був захищати тіло людини від несприятливих зовнішніх природних умов. З плином часу він змінювався, повинен був нести не тільки функцію захисту, але і естетично задовольняти погляди людини, що формувались в певному соціальному середовищі. Завдяки теперішнім технологіям та інтернет можливостям придбання одягу не складає жодних проблем. В теперішній час велика кількість підприємств використовують інтернет для продажу товарів, чи закупівлі сировини. Інтернет став невід'ємним атрибутом сучасних людей. Все більше установ використовують інформаційні системи для управління внутрішніми процесами. Адже ріст об'єму швейних виробів і покращення якості виробів знаходиться в прямій залежності від вдосконалення технологій виробництва та сучасного програмного забезпечення.

Тепер виникає потреба в осучасненні всіх процесів виробництва. І виключенням не є ательє. Для таких підприємств, в теперішний час, дуже потрібно інтегрувати свою діяльність разом з сучасними технологіями. Тому в цій дипломній роботі ставилась мета допомогти цим підприємствам.

Перед працівниками швейної промисловості стоїть відповідальне завдання, задовольнити потреби населення в швейних виробах. Розширити масштаби випуску продукції, а надалі і продажу, при цьому полегшити умови праці. Дуже вигідно та престижно для будь якого підприємства є наявність власного веб-сайту. Адже завдяки сайту, велика кількість потенційних клієнтів ознайомлюється з підприємством та видом їхньої діяльності. Це збільшує кількість покупців, та допомагає підприємству розвиватись та рости.

В даний час зважаючи на різні умови, такі як зайнятість населення, карантин, недостатність часу для походів в магазин, велика частина населення звертається до онлайн покупок. Це зберігає час, та дозволяє навіть в транспорті зробити замовлення наряду на святковий вечір, та не тратити на це зайвого часу.

Функції сайту які представлені в даному дипломному проекті, максимально спрощують замовлення, яке потрібно виконати. Клієнт з легкістю може вибрати

модель, колір, тип, матеріал, визначити свій розмір, прикріпляти фото бажаного виробу та залишати відгуки.

Тому, було вирішено дослідити можливості створення веб-застосунку для замовлення пошиття одягу, що дозволить виконувати замовлення з будь-якого пристрою, який має доступ до мережі інтернет.

1 ПОНЯТТЯ ТА ОСОБЛИВОСТІ АТЕЛЬЄ

1.1 Поняття ательє

В сучасному світі є велика кількість людей, які бажають самовиражатися, показувати свою індивідуальність. Переважно це одяг та аксесуари, що утворюють довершений образ. Смак в поєднанні одягу з аксесуарами утворює особливий стиль. Кожна людина формує свій стиль відносно темпераменту та характеру. Гарячі та запальні холерики вдягаються в крок з модою, полюбляють екстравагантний та неординарний одяг, яскраві кольори, та не бояться експериментів. Сангвініки – повна протилежність, скуті та спокійні, обирають тьмяні кольори та стандартні фасони в одязі.

Дивлячись на зовнішній вигляд людини, можемо оцінити її одяг та по ньому визначити, який в особи статус, та чим вона займається. Купуючи одяг в магазинах чи торгових центрах, ви не зможете підібрати наряд, який ідеально ляже на ту чи іншу будову тіла (хіба ви володар тієї самої фігури, що задана в ГОСТ). Завжди будуть невеличкі недоліки, які не дозволяють людині почувати себе на всі сто відсотків. Інколи подобається все, окрім комірця або занадто великого вирізу. В кожному купленому одязі завжди є хоча б одна деталь, якою ми незадоволені. Щоб клієнти могли бути повністю задоволені кожною покупкою, їм потрібно самим вирішити всі особливості того чи іншого одягу. Якщо людина знає, чого вона бажає, має уявлення свого найкращого елементу одягу і вміє все гарно роз’яснити майстру, то ідеальний результат гарантовано.

Велика кількість людей вдома мають швейні машинки, за допомогою яких вони можуть злегка змінити щось в одязі, приховати якісь недоліки, додати якусь особливість звичайній речі. Для цих людей шиття - це хобі, вони не мають достатньої кількості знарядь праці та навиків. На превеликий жаль, в 90 випадках зі 100 цього недостатньо. Потрібна рука справжнього майстра, який з

непримітного шматка тканини може створити надзвичайний витвір мистецтва, що вразить будь-кого.

Тоді у нас виникає ще одна невелика проблема. Де ж нам шукати такого майстра своєї справи? Якщо запитати своїх знайомих, то хтось порадить звернутись до ательє.

Ательє (з французької – це майстерня, цех, студія) – салон індивідуального пошиття одягу, взуття, головних уборів або майстерня маляра, скульптора чи фотографа. Слово походить з французької мови. Спершу його застосовували для означення столярні. Згодом усі майстерні стали називатись ательє [1].

1.2 Різновиди ательє

Поняття «ательє» різниться в залежності від сфери застосування цього слова. Розглянемо, що означає дане поняття в кожній з галузей, в яких його використовують.

1.2.1 Ательє в мистецтві

В мистецтві ательє складається з майстра-художника, як правило, професійного живописця, скульптора або, з середини XIX століття, художнього фотографа, який працює з невеликою кількістю студентів, щоб навчити їх художньому чи образотворчому мистецтву (рис 1.1) [2]. Це саме слово також набуло інших подібних значень, що вказує на місце роботи та навчання модельєра високої моди , стиліста та перукарів взагалі. Всі будівлі, в яких працювали люди пов’язані з мистецтвом, називались ательє тому, що самі майстри почали називати їх саме так.

Рисунок 1.1 – Французьке ательє живопису під назвою «Школа образотворчих мистецтв – майстерня живопису»

Це була стандартна професійна практика для європейських художників від Середньовіччя до XIX століття, яка також поширювалася по інших місцях світу. У середньовічній Європі такий спосіб роботи та форми художнього чи образотворчого мистецтва часто застосовувався місцевими правилами гільдії, Гільдією художників Святого Луки та ще багатьма гільдіями інших ремесел. Учні зазвичай починали молодими, працюючи над простими завданнями, а через кілька років ставали мандрівниками, перш ніж стати майстрами. Систему поступово замінювали, коли потужні гільдії занепадали. І академія стала прихильним методом навчання, хоча багато професійних художників продовжували користуватися студентами та асистентами. Деякі платили своєю працею, а деякі платили грошові внески за навчання.

Рисунок 1.2 - Ательє Роберта-Флері в Академії Джуліана для студентів жіночого мистецтва

Хоча методи різняться, більшість ательєрів живопису навчають учнів умінням та прийомами, що пов'язані зі створенням певної форми образотворчого мистецтва Вони традиційно включають заняття з малювання оголеного мистецтва, як зображено на рисунку 1.2.

В мистецтві є надзвичайно багато випадків, коли приміщення, в яких працюють люди, називались ательє, проте сьогодні їх переважно називають студіями. Яскравими прикладами є студія акторської майстерності, архітектури, живопису, гончарства (кераміки), скульптури, оригамі, деревообробки, скрапбукінгу, фотографії, графічного дизайну, кіномистецтва, анімації, промислового дизайну, радіо чи телевізійного мовлення, або створення музики.

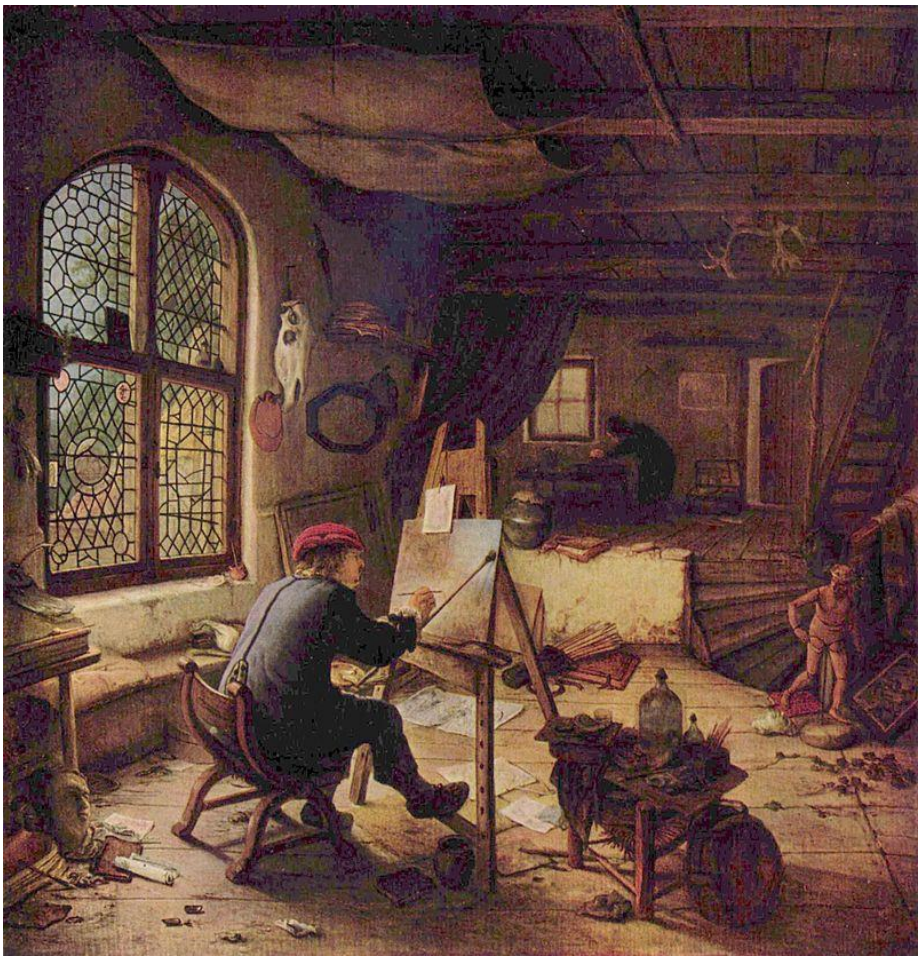

Рисунок 1.3 - Студія Adriaen van Ostade 1663.

Давайте розглянемо приклад навчальної студії (ательє). У навчальних студіях студенти розвивають навички, пов'язані з дизайном, починаючи від

архітектури до дизайну виробів. У конкретних навчальних ательє є аудиторії, де велика кількість студентів навчаються проектувати за допомогою інструктивної допомоги в коледжі. Вони відомі тим, що залишатись там можна до пізньої години, займаючись проектами та просто спілкуючись.

Студійне середовище характеризується двома типами в освіті:

– робоча область, де студенти, як правило, зорово зосереджені на роботі у відкритому середовищі. Цей час і простір виходять за рамки навчального часу і керівництву з викладачами недоступні. Це дозволяє студентам залучати один одного, допомагати та надихати один одного під час роботи;

– тип класу, який займає вищезазначений простір майстерні та відтворює його основний компонент відкритого робочого середовища. Він розмежовується на основі теми навчання, ізольованого простору, керівника чи інструктора для додаткової уваги або спрямованої критики.

1.2.2 Автомобільне ательє

Автомобільне ательє – приміщення в якому майстри налаштовують та покращують автомобілі (приміщення тюнінгу автомобілів). До основних завдань працівників таких ательє належать: підвищення потужності двигуна, поліпшення аеродинамічних якостей, зміна зовнішнього вигляду та інтер'єру, установка звукових, мультимедійних та охоронних систем.

Яскравим прикладом тюнінг-ательє є так зване «придворне ательє» Mercedes відоме як «Brabus» (рис. 1.4). «Брабусівскі Мерси» користуються заслуженою пошаною усюди, репутація ательє бездоганна. Майстри компанії «Brabus» мають можливість працювати над найновішими моделями Mercedes.

Повним спектром послуг можна вважати перелік, що включає: тюнінг двигуна, стайлінг (поліпшення зовнішнього вигляду завдяки новим кузовним елементам і спеціальному фарбуванню), обробка салону (перетяжка сидінь, заміна

приладів, установка пакетів, що підвищують комфорт), а також звукова інсталяція. Для того щоб надавати такий великий спектр послуг, крім мінімуму фахівців, потрібно ще й достатній простір - від 700 метрів квадратних.

Рисунок 1.4 - Зображення тюнінг-ательє компанії *«Brabus»*

А ще власники автомобілів полюбляють покращувати показники мотору та міняти оптику, щоб економити пальне навіть на звичайних лампах. І замість звичайних ламп розжарювання ставлять денні ходові вогні.

1.2.3 Столярне ательє

Схожість поняття «фабрика» та «ательє» полягає в тому, що виготовлення меблів об'єднується в одну лінію для отримання кінцевого продукту, але принципи роботи суттєво відрізняються. Починаючи розмову про фабрики, варто згадати мануфактури, адже саме з них почалося масове виробництво із застосуванням ручної праці найманих робітників. Перші мануфактури виникли в Італії в XIV в. Саме в мануфактурах вперше відбулося об'єднання ручної праці в єдиний процес з випуску продукції.

Слово «ательє» з'явилося в побуті в кінці XIX ст. Джерелом виступає латинське слово «assula» - майстерня. Під «ательє» мають на увазі будь-яке робоче приміщення, в якому йде творчий процес, який об'єднує художників, скульпторів, фотографів. Але часто термін використовують для позначення швейних або меблевих майстерень, адже в них створюються вироби за індивідуальними замовленнями, що вимагають творчого підходу.

Меблі під замовлення користуються великим попитом. Щоб замовити не просто якісні, але і виготовлені за індивідуальними розмірами предмети інтер'єру, варто неодмінно звертатися в ательє. Звідси головна відмінність меблевого ательє від фабрики - можливість замовити ліжко, диван або крісло за індивідуальними параметрами (рис. 1.5). Меблеві ательє мають у своєму розпорядженні всі можливості для виготовлення якісних меблів в будь-якій кількості під замовлення:

- власне виробництво з цехами для виготовлення меблів;
- можливість виготовлення меблів по фотографії за індивідуальними розмірами;
- підготовка ескізу або складання повноцінного проекту;
- виконання замовлень в будь-якому стилі: класика, авангард, модерн, хайтек, лофт.

Рисунок 1.5 - Меблі з масиву дерева в особливому оздобленні

Звідси потрібно зазначити, що звертаючись з індивідуальним замовленням в ательє, ви безумовно отримаєте унікальний, ретельно і акуратно виготовлений, відповідний саме вашому творчому задуму і індивідуальному смаку, а також розміру продукт. Але не варто очікувати, що бажаний виготовлений продукт буде коштувати дешевше, ніж аналог, що «клепають» з невідповідних матеріалів десь в Китаю. Існує велика кількість людей, які розуміють бажання зробити краще, розуміють, що ручна робота це те, що зроблено з душею, а гроші, якщо і мають якесь значення, то вельми опосередковане.

1.2.4 Ательє для пошиття одягу

Сьогодні немає проблем з покупкою готового одягу, але більшість людей все ж віддають перевагу індивідуальному пошиву, адже зшитий на замовлення одяг завжди сидить краще. А також ви сміливо можете бути впевнені, що він в одному екземплярі.

Сама ж послуга індивідуального пошиття, що надається будь-якою компанією, повинна відповідати всім критеріям якісного пошиття одягу на замовлення. Для створення індивідуальної викрійки, модельєри знімають близько 30 мірок і роблять фотографії замовника в чотирьох проекціях. Деякі кравці підганяють зняті мірки під викрійку, інші ж створюють індивідуальну.

Індивідуальний пошив класичного одягу відображає ставлення людини, яка носить його, до самої себе та оточуючих. Магазини чоловічих костюмів, які продають готові вироби, можуть запропонувати клієнтові безліч моделей, але у всіх них не буде тієї особливості, яку можна віднайти тільки в ательє з пошиття одягу. Ваш костюм – це вже 50% загального враження про вас. Воно виникає миттєво, в перші секунди зустрічі, поки ви ще навіть не встигли відрекомендуватися. Одна справа одягнути хоч і брендовий, але пошитий за певними загальними стандартами піджак, без урахування особливостей вашої фігури, а інша – це особисто ваш костюм в усіх сенсах цього слова.

Костюм може без зайвих слів показати той стиль і смак, яким володіє його власник. Чоловік в хорошому, гарному одязі виглядає надійно, впевнено, він викликає повагу колег і справляє враження на жінок. Неважливо, на яку подію ви йдете: бізнес-зустріч, випускний, вечеря з коханою чи власне весілля – важливо завжди виглядати на висоті.

Магазини чоловічих костюмів мають готові зразки, які більшою чи меншою мірою вам підходять: по фігурі, смаку, матеріалу і т.д. Однак всі вони пошиті не для вас. Тільки кравець здатний врахувати тип фігури, приховати недоліки, підкреслити переваги, знайти правильний спосіб художньої обробки індивідуальних особливостей (рис. 1.6).

У пошитті чоловічого одягу на замовлення важлива кожна деталь: кількість внутрішніх і зовнішніх кишень, розмір, матеріал ґудзиків, фасон коміра, наявність і кількість шліц, складки на брюках, матеріал і колір підкладки, наявність спеціальної секретної внутрішньої кишені, кількість петель для ременя і багато іншого. Погодьтеся, що магазини чоловічих костюмів ніколи не запропонують повністю ідеального костюма.

Рисунок 1.6 - Процес шиття чоловічого класичного костюма

Існує велика кількість популярних та ефективних методів пошиття одягу. Однією з таких є технологія bespoke [3], передбачає створення індивідуального лекала. Спочатку знімаються десятки мірок. Крім того, для більшої точності потрібно зробити знімки клієнта в чотирьох проекціях. Це не тільки робочий процес, але і своєрідний чоловічий ритуал з особливою атмосферою. Клієнту необхідно почекати кілька тижнів після узгодження всіх деталей. Цей термін цілком виправданий, тому що пошиття чоловічого одягу на замовлення відбувається на 80% вручну, що виправдовується високою якістю. Це тривала робота майстрів з дотриманням всіх пропорцій і найдрібніших деталей.

Але не тільки чоловіки бажають виглядати чудово замовляючи одяг в ательє. Жінки так само, якщо і не більше, хочуть мати такі елементи одягу, які затьмарять інших. Послуга індивідуального пошиття жіночого одягу на замовлення підходить тим, хто стежить за модою і піклується про враження від себе. Дуже важливо, щоб речі не тільки відповідали актуальним тенденціям, але були б неповторні. Індивідуальне пошиття жіночого одягу на замовлення найкращим чином вирішує цю задачу: ви отримуєте речі, виконані в руслі наймодніших тенденцій, при цьому можете бути впевнені, що ні в кого не зустрінете точно таку ж одіж.

Послуги з пошиття жіночого одягу за індивідуальними мірками - ціле мистецтво. Потрібно враховувати не тільки особливості фігури замовника, а й те, як він рухається, сидить. Тільки виконання всіх цих вимог дозволить створити унікальний предмет гардеробу, що вигідно відрізняється від стандартизованого жіночого одягу, який висить на вішалках будь-якого з магазинів.

Висококласні майстри здатні реалізувати задум будь-якого рівня складності. Великий досвід індивідуального пошиття жіночого одягу на замовлення дозволяє професійно вловлювати суть різних побажань замовниці і виконувати речі навіть найскладніших фасонів. Якщо Ви хочете замовити плаття, вечірній ансамбль або інший вид одягу, але поки не визначилися в своєму виборі, майстри допоможуть в розробці індивідуального дизайну, дадуть консультацію щодо вибору відповідних Вам фасонів, запропонують варіанти тканин і обробки.

У ательє з пошиття жіночого одягу на замовлення Ви також можете зшити наряд по фото, яке ви, можливо, знайшли в глянцевому журналі або магазині.

Послуги з пошиття жіночого одягу на замовлення - це оптимальне рішення для всіх, хто володіє нестандартною фігурою. Готовий одяг зазвичай розрахований на стандартні параметри, так що його фасон і посадка по фігурі часто залишають бажати кращого. Одяг, пошитий на замовлення в ательє, буде сидіти на Вас ідеально, а його фасон буде розроблений так, щоб підкреслити переваги і приховати недоліки Вашої фігури.

Кожне пошиття одягу на замовлення передбачає побудову окремої конструкції і лекал, що забезпечує ідеальну посадку по фігурі. При пошитті жіночого одягу на замовлення використовується поєднання сучасних технологій і ручної роботи, що гарантує бездоганний зовнішній вигляд і якість (рис. 1.7).

Рисунок 1.7 - Процес пошиву жіночого одягу

Ательє пропонує дизайнерську розробку та індивідуальне пошиття на замовлення одягу для жінок різних видів: блузи, сукні, брюки, спідниці, костюми, вечірні та весільні сукні, пальто і будь-які вироби з трикотажу.

Але ательє займаються не тільки пошиттям нового одягу, а й виконують ряд таких робіт:

– ремонт одягу з будь-яких матеріалів і різної складності. До всіх речей необхідно ставитись особливо, дбайливо й уважно;

– кожен покупець класичних костюмних брюк чи спідниць хоча б раз у житті стикався з проблемою їх довжини. Тому ательє пропонують надзвичайно корисну послугу – вкорочення низу штанів, джинсів та спідниць;

– якщо вам важко розпрощатися з улюбленими дірявими джинсами, то в ательє легко можуть повернути їм життя. Для цього варто лише принести вашу річ до спеціалістів. Майстри у найкоротший термін усунуть усі недоліки на вашому одязі;

– сукня – улюблена річ в гардеробі кожної модниці. Враховуючи усі ваші побажання, в ательє зможуть оновити вашу сукню до непізнаваності: додати стрази, блискітки, декоративні тканини, змінити фасон відповідно до вашого смаку;

– встановлення кнопок і люверсів. Робота з фурнітурою дуже кропітка. За лічений час є можливість без проблем встановити кнопки та люверси на різноманітні вироби зі шкіри та тканини;

– ремонт трикотажних речей. Робота з цим матеріалом вимагає точності та акуратності майстра;

– пошиття постільної білизни. Можливий пошив білизни з будь-якого матеріалу на замовлення різного розміру: та білизна для новонароджених;

– пошиття уніформи.

В теперішний час велика кількість людей одягаються стильно. Тому, якщо у Вас є бажання мати одяг, який буде тільки у Вас, то вам дорога до будь-якого ательє для пошиття одягу.

1.3 Аналоги сайту для замовлення пошиття одягу (веб-ательє)

Найбільш відомим аналогом інтернет-ательє в світі є Frilly [4]. Frilly - це фактично перша модна онлайн-платформа, що забезпечує справжнє налаштування. Якщо плаття кидається в очі, ви можете вибрати колір, довжину подолу, декольте, комір, водолазку чи V – стиль, а також вид рукавів і тканину. Залежно від стилю, ви можете додати такі деталі, як гребінчастий подол або пояс, створивши дизайн, яким більше ніхто не володіє. Їх фірмове програмне забезпечення для 3D-рендерінгу, яке потребувало трьох років розробки, показує клієнтам кожен варіант, коли вони будують свій одяг. Настроювання показано на моделях, тож ви можете побачити, куди все потрапляє. На сайті ательє Frilly розробники вже мали велику базу замовлень і фотографій. Завдяки ній вони і розробили 3D-примірку одягу. Щоб інтерпретувати цю примірку в реальність, потрібно розділити один фрагмент одягу на невеликі частини. Пізніше можна змінювати тільки один фрагмент, не змінюючи інші. Але для того, щоб зробити такий проект знадобилась велика кількість розробників та 3 роки часу.

В Україні подібних аналогів немає. В більшості випадків це прості сайти-візитки для ательє. В таких веб-застосунках є тільки інформація про сайт та галерея виробів. Для замовлення пошиття є тільки електронна адреса або номер телефону. Рідше можна знайти ательє, розроблене на основі інтернет магазину. На таких сайтах користувач може зробити замовлення онлайн, але в такому випадку він матиме змогу вибрати щось з невеличкого каталогу. На превеликий жаль, в жодному з випадків немає змоги додати щось від себе. Виключно стандартні фасони та розміри. А якщо і є можливість придумати щось надзвичайне, то немає можливості описати всі бажання разом в своєму замовленні. Завжди доведеться уточнювати всі дрібниці по телефону чи в повідомленні.

На сайті, що описується в межах даної дипломної роботи є можливість обирати тип вибраного фрагменту. А ще є спеціальне поле, де користувачі можуть описати всі свої забаганки, або прикріпити фото аналогічного одягу.

1.4 Постановка задачі для веб-сайту ательє

Основним завданням цього дипломного проекту є створення веб-застосунку ательє для пошиття одягу. Цей проект особливо актуальний в теперішний час, коли по цілому світу розповсюджений вірус COVID-19. Якщо ви хочете замовити собі наряд, не виходячи з дому, то в цьому немає жодних проблем. Немає нічого простішого! У нашому креативному онлайн-ательє ви можете зробити це за кілька хвилин. Впишіть у відповідні поля форми свої розміри і прикріпіть фото моделі, яку хочете замовити. Далі просто відправте свою заявку, а ательє допоможе зробити вашу мрію реальною.

Для реєстрації та авторизації необхідні деякі персональні дані користувачів, що бажають користуватись цією аплікацією.

Щоб зробити замовлення в інтернет-ательє необхідні наступні вхідні дані: ім’я та прізвище замовника, електронна адреса і номер телефону (для уточнення чи сповіщень щодо замовлень) та дані, що стосуються самого замовлення. До даних, що стосуються самого замовлення належать: стать замовника, найменування товару (різновид одягу), колір, тип тканини, вид комірця, довжина рукава, довжина вибору, розмір замовника та мінімальна ціна.

Коли клієнт не знає свого розміру одягу, йому для зручності потрібен калькулятор, що допоможе визначити розмір в декілька кліків. Вхідними даними для калькулятора є дані вимірювань персони, а вихідними даними - визначений стандартний розмір в буквеній формі (XS, S, M, L, XL, XXL).

Для розробки інтернет-застосунка для ательє необхідно створити аплікацію з наступним списком функцій:

– розробити сховище даних для сайту, в якому зберігатимуться всі дані та замовлення користувача;

– створити реєстрацію та авторизацію користувачів, щоб постійні клієнти могли переглянути всі свої замовлення, а також потенційні клієнти могли створити свій кабінет на цьому сайті;

– зробити створення актуальних новин для сайту адміністраторами;

– створити комфортні сторінки для створення та редагування замовлень;

– додати можливості підтвердити або видалити замовлення;

– розробити калькулятор, що з введених даних користувача обчислить його розмір без зайвих кроків, дзвінків чи інструкцій;

– створити сповіщення на електронну пошту про підтвердження чи готовність замовленого продукту;

– також необхідно отримувати відгуки про персонал та про продукцію.

Перевагами даного веб-застосунку для пошиття одягу є:

– можливість використання з будь-якого пристрою, що має доступ до інтернету;

– постійне оновлення даних;

– надсилання сповіщень на електронну пошту про стан готовності замовлення;

– можливість автоматично визначити свій розмір одягу онлайн.

Недоліками даного застосунку є:

– відсутність 3D примірки онлайн;

– проблеми з інтегруванням калькулятора до сайту.

2 ПРОЕКТУВАННЯ ТА МОДЕЛЮВАННЯ ВЕБ-САЙТУ ДЛЯ ЗАМОВЛЕННЯ ПОШИТТЯ ОДЯГУ

2.1 Проектування бази даних

Одним з найперших етапів розробки аплікації є створення сховища для всіх даних [5]. Щоб програма працювала відмінно, необхідно дуже чітко обдумати всі кроки. Потрібно передбачити всі сутності, що будуть в застосунку, і ,звісно, всі властивості сутності. В більшості випадків одну сутність відображають як одну таблицю бази даних. Проте інколи розділяють на декілька таблиць (коли одна таблиця занадто велика). Що ж потрібно для веб-застосунку ательє?

Насамперед нам потрібно зберігати дані кожного користувача. Для цього необхідно створити таблицю user. Тепер потрібно вирішити, які властивості користувача необхідно зберігати. Звичайно, мусить бути ідентифікатор! Саме завдяки ідентифікатору буде змога швидко і зручно звертатись до того чи іншого користувача. Ідентифікатор має бути унікальним, тому що імена та інші дані про користувача можуть повторюватись. Наступним кроком будемо зберігати такі особисті дані кожного користувача: ім’я, прізвище, електронну адресу, стать, номер телефону, адресу проживання, розмір одягу та роль користувача. Цілком зрозуміло, що всі вищеописані пункти - це ті дані про користувача, які необхідні для створення замовлення або для підтримки зв’язку з користувачем. Для того, щоб розділити функціонал сайту на декілька частин, потрібний пункт «роль користувача» (admin). Цей пункт зберігатиме булеве значення (true або false). По замовчуванню користувач створюватиметься зі значенням «false» для поля admin. Значення «true» має адміністратор сайту.

Адміністратор – це та особа, що відповідає за створення та редагування новин сайту. Він має змогу переглянути всі замовлення на сайті та повинен змінювати статуси замовлень в залежності від їх готовності. Змінити свої особисті

дані користувач має змогу на сторінці свого профілю. Для перегляду загальної картини таблиці «користувач» дивіться на Рис. 2.1.

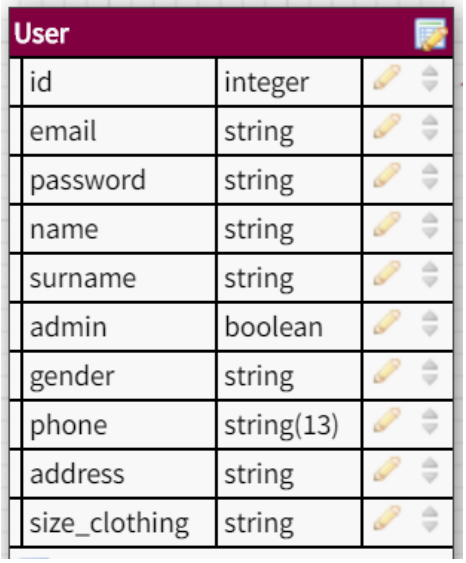

Рисунок 2.1 - Таблиця «Користувач» БД

Найпростішою сутністю в базі даних (БД) є пости (рис. 2.2). Пости відображаються на основній сторінці веб-застосунку. Їх можна використовувати для завантаження новин сайту, для відображення найкращих замовлень або важливої інформації для клієнтів. Створювати пости має змогу тільки користувач, який має права адміністратора. Таблиця «пост» містить наступні поля: заголовок, тіло та вкладений файл. Звісно, заголовок відображатиметься жирнішим та більшим шрифтом. Тіло поста міститиме звичайний текст. А вкладений файл зберігає шлях до зображення. Адміністратор може прикріпити фотокартку до поста, якщо вважає це потрібним.

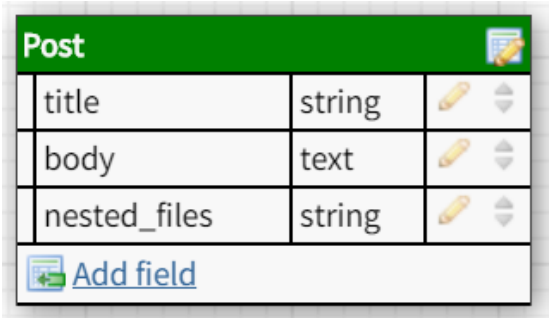

Рисунок 2.2 - Таблиця «Пост» БД

Ось і настав час описати основні таблиці для веб-ательє. Основний функціонал для цієї аплікації є створення, редагування та видалення замовлень на пошиття одягу. Для сутності «замовлення» в даному проекті було вирішено створити 2 таблиці. Одна так і називається: «Замовлення (order)», інша – «Деталі (detail)»(рис. 2.3 та рис. 2.4).

Таблиця «Замовлення» повинна містити дані про замовника. Тому необхідні наступні поля: ідентифікатор замовлення, ідентифікатор користувача, ім’я, прізвище, електронна адреса, телефон та адреса самого замовника. Також в замовленні потрібно вказати стать того, для кого буде створюватись одяг. Для зручності було додане поле, яке зберігає стани (state). Дане поле записуватиме поточний стан замовлення. Коли користувач тільки-но створив замовлення, автоматично стан було записано як «створено» (created). Для підтвердження потрібно натиснути всього на 1 кнопку в кошику замовлень користувача. Після підтвердження замовлення, на електронну адресу приходить сповіщення про те, що замовлення підтверджено і перебуває в стані обробки. Пізніше змінювати стани матиме змогу тільки адміністратор. Коли адміністрація ательє вибрала особу, яка буде шити одяг, то стан змінюється на «розпочато шиття» (теж приходить сповіщення). Зрозуміло, що коли замовлений продукт готовий, стан зміниться на «шиття завершене».

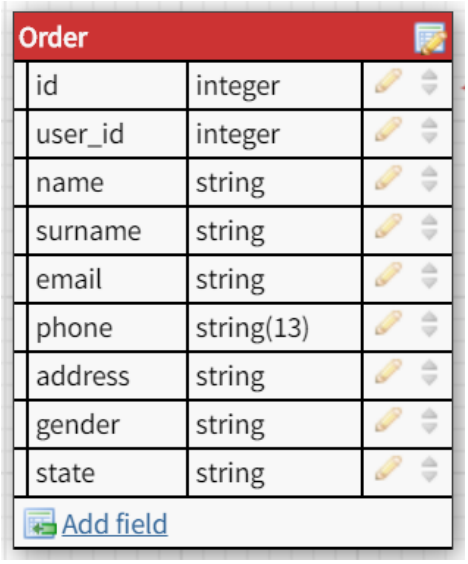

Рисунок 2.3 - Таблиця "Замовлення" БД

Також для нашого замовлення потрібно зберігати його окремі деталі. В табличці «Деталі» потрібно зберігати ідентифікатор замовлення (до якого замовлення прикріплені дані деталі), назву одягу (сукня, костюм, сорочка…), колір виробу, тип комірця (якщо потрібно), довжину рукава, довжину виробу, розмір у буквеній формі, тканину. Якщо користувач хоче щось особливе, він має змогу написати всі свої побажання в текстовому полі «wishes». В цьому полі можна записати всі розміри користувача для кращого пошиття, вказати особливості свого тіла. Також до деталей замовлення необхідно додати можливість прикріпляти фото бажаного виробу. В такому випадку модельєр наглядно буде бачити, як йому пошити замовлення. Поле, в якому зберігається шлях до зображення, називається «nested_files».

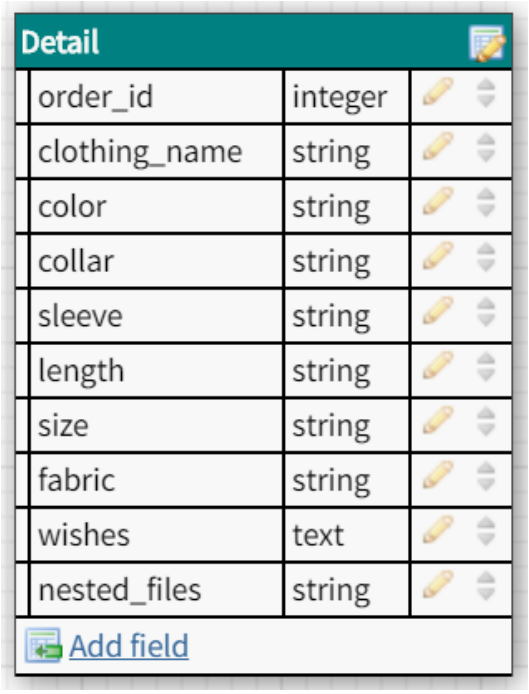

Рисунок 2.4 - Таблиця «Деталі замовлення» БД

Тепер потрібно створити таблицю, в якій будуть зберігатись усі відгуки користувачів про вироби та про роботу самого ательє (рис. 2.5). Для цього було створено однойменну сутність «Відгуки». Для того, щоб вказати хто є автором цього відгуку, було створено поле «author_id». Поле автора відгука містить ідентифікатор користувача, який його створив. Для зручності ще збережемо ім’я

та прізвище цього користувача. Тепер, щоб зберегти увесь текст відгуку, було створене поле під назвою «тіло відгуку (body)». Якщо замовник виявив брак у роботі, то для того, щоб продемонструвати це, було створене поле, яке зберігає адресу до прикріпленого зображення.

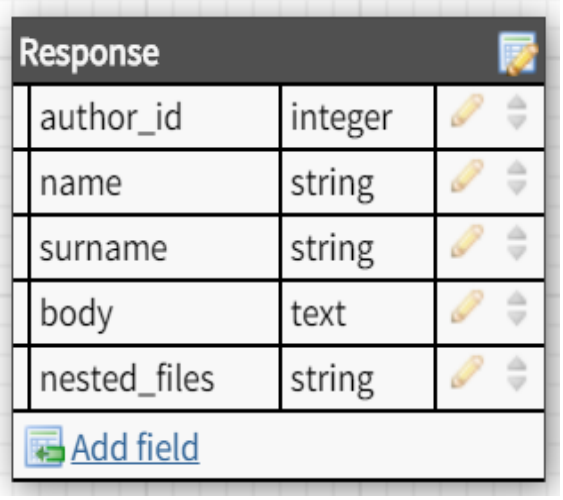

Рисунок 2.5 - Таблиця «Відгук» БД

Для взаємодії між таблицями, їх потрібно з'єднати зв'язками. Наприклад, остання таблиця «Відгук» повинна бути з'єднана з таблицею «Користувач». Це дає можливість переглянути всі дані користувача і дізнатись причину, чому був написаний той чи інший відгук. Ці дві таблиці з'єднуються зовнішнім ключем (Foreign key) завдяки полям ідентифікаторам. В таблиці «Відгук» є поле «author_id», що сполучене зовнішнім ключем з таблицею «Користувач» полем «id». Аналогічно з користувачем з'єднана інша таблиця - «Замовлення». Відмінним є те, що в цій таблиці поле має назву «user_id». В даному проекті сутність «Замовлення» складається з 2 частин. Тому ці частини теж потрібно сполучити разом. Для цього в табличці «Деталі» створене поле «order_id». Це поле сполучене зовнішнім ключем з полем id таблички «Замовлення». В аплікації для веб-ательє немає потреби сполучати пости сайту з будь-якою іншою сутністю. Адже за допомогою цих постів, було вирішено тільки інформувати користувачів про загальні новини сайту, а для цього брати дані з інших сутностей немає сенсу.

Після того, як ми обдумали всі сутності та зв’язки, які будуть будуватись між ними, ми маємо змогу розробити ER-Діаграму [6]. ER-діаграма – це така діаграма, що описує модель «сутність-зв'язок». Існує велика кількість інструментів представлення даних, але, на мою думку, модель «сутність-зв'язок» є найпростішою та найбільш зрозумілою, як на рисунку 2.6.

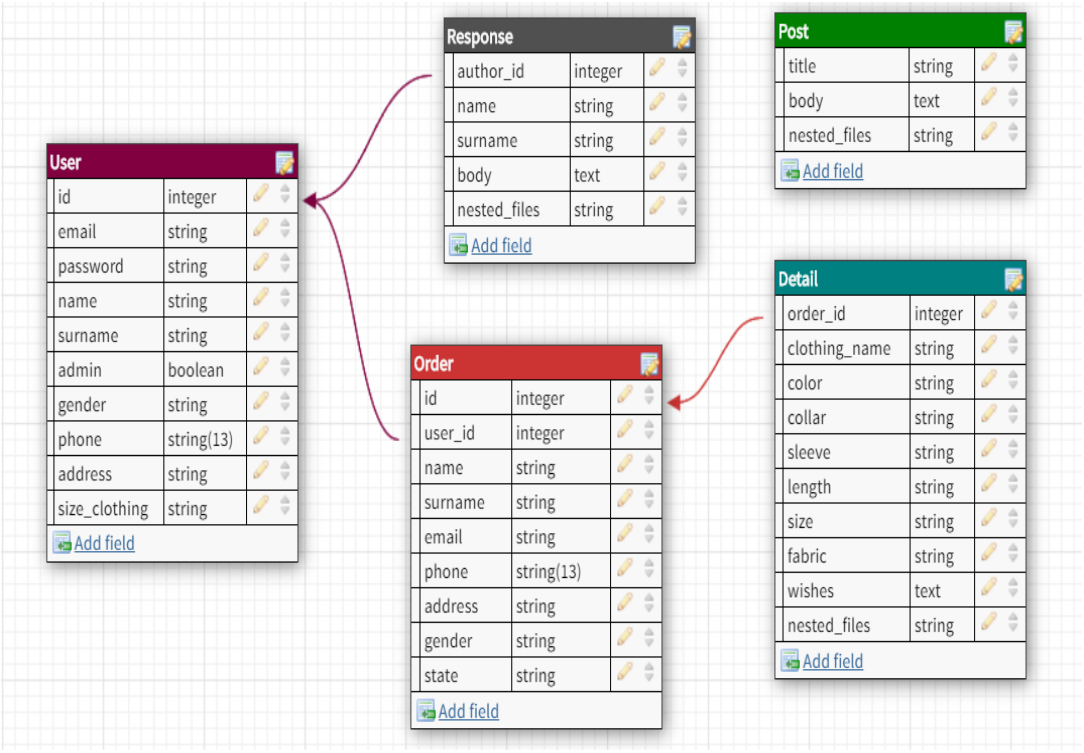

Рисунок 2.6 - ER діаграма сховища даних

2.2 Структура аплікації для замовлення пошиття одягу

Веб-застосунок для ательє розроблений на мові програмування Ruby та фреймворку Ruby on Rails. Тому при створенні цього проекту автоматично було встановлено певні папки та файли в спеціальній послідовності. Якщо створити якийсь функціонал, але не завантажити його в місце призначення, то проект не працюватиме. Потрібно вивчити структуру застосунку і аж тоді приступати до виконання якоїсь роботи. В такому випадку розглянемо структуру цієї аплікації.

За винятком незначних змін між випусками нових версій фреймворку, кожен проект Rails має однакову структуру з однаковими угодами про імена. Ця послідовність дає величезну перевагу. Ви можете переміщатися між проектами, написаними на фреймворку Rails, без повторного вивчення організації проекту.

Щоб зрозуміти структуру каталогів [7], давайте використовувати розроблений додаток для ательє (рис. 2.7).

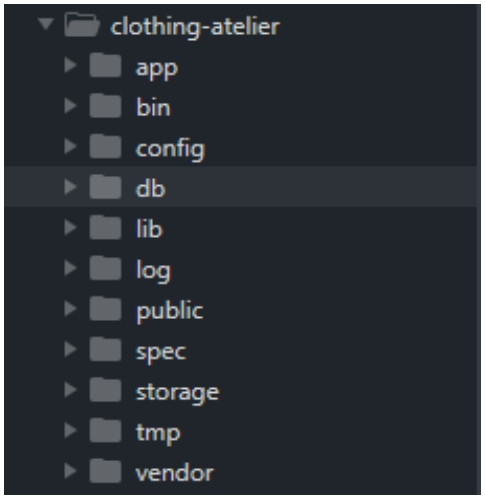

Рисунок 2.7 - Структура аплікації для ательє

App – містить всі компоненти програми. Програма розроблена з використанням MVC моделі. Отже, в підкаталогах знаходяться всі частини MVC моделі та інші файли аплікації.

Bin – вміщує в собі Rails скрипти, які запускають дану програму. Також каталог bin може включати скрипти для налаштування, оновлення та розгортання.

Config – має невеликий обсяг коду конфігурації, який знадобиться цьому додатку, включаючи конфігурацію бази даних (в файлі database.yml), структуру середовища Rails (environment.rb) і маршрутизацію веб-запитів (routes.rb). Ви також можете налаштувати поведінку трьох середовищ Rails для тестування, розробки і розгортання за допомогою файлів, що знаходяться в каталозі середовищ.

DB – Rails-додаток для ательє має об'єкти моделі, які звертаються до таблиць реляційної бази даних. Для того, щоб керувати реляційною базою даних за допомогою скриптів, ми створюємо і розміщуємо в цьому каталозі ці скрипти.

Lib – тут ви розмістите бібліотеки, якщо вони явно не належать іншим місцям (наприклад, бібліотекам постачальників).

Log – тут зберігаються журнали помилок. Rails створює скрипти, що допомагають керувати різними журналами помилок. Ви знайдете окремі журнали для сервера (server.log) і кожної середи Rails (development.log, test.log і production.log).

Public – подібно загальнодоступному каталогу для веб-сервера, в цьому каталозі є незмінні веб-файли, такі як файли графіки (public / images), таблиці стилів (public / stylesheets) і файли HTML (громадськості).

Spec – не є стандартною папкою проекту. Стандартна папка має назву test. Але в даному проекті використовується спеціальна бібліотека для різних видів тестування, тому її названо таким чином. В цьому каталозі розміщаються тести для всіх моделей, контролерів та інше.

Tmp – використовується цей каталог для зберігання всіх тимчасових файлів (кеш, pid).

Vendor – бібліотеки, які надаються сторонніми постачальниками (наприклад, бібліотеки безпеки або утиліти баз даних за межами основного дистрибутива Rails).

Окрім всіх цих каталогів, Rails застосунок містить декілька файлів. Основними є 2 наступні: README, Gemfile та Rakefile.

Gemfile – це файл для збереження всіх гемів (бібліотек чи просто готових рішень) та їх версій. Bundler – це менеджер для управління залежностями гемів в ruby додатках. Ця утиліта дозволяє легко встановлювати необхідні геми для застосування, при цьому зовсім не залежати від встановлених в системі. Bundler включили в Rails 3.0 за замовчуванням і тепер, саме він використовується для управління залежностями гемів в даній версії фреймворка. Цю утиліту можна використовувати для будь-якого ruby фреймворка.

Rakefile – цей файл допомагає при складанні, упаковці та тестуванні коду Rails, створенні, видаленні бази даних, чи навіть виведення всіх URL шляхів застосунку. Утилітою rake поставляється разом з установкою Ruby.

README – цей файл містить основну інформацію про аплікацію і опис структури каталогів, описаної вище.

Тепер ми розглянули основну структуру даного проекту. Але це тільки основні файли, що встановлюються системою. Для розуміння самого проекту розглянемо усі компоненти програми що знаходяться в каталозі App. Даний каталог вміщає в собі наступне:

– app / controllers – в цьому підкаталозі Rails шукає класи контролерів. Контролер обробляє всі веб-запити від користувача;

– app / models – підкаталог models містить класи, які моделюють і упаковують дані, що зберігаються в базі даних застосування для ательє. У більшості середовищ ця частина програми може стати досить брудною, багатослівною і схильною до помилок. Rails робить його неймовірно простим;

– app / view – підкаталог views містить шаблони відображення для заповнення даними з нашого застосування, перетворення в HTML і повернення в браузер користувача;

– app / view / layouts – містить файли шаблонів для макетів, які будуть використовуватися з відображеннями зовнішнього вигляду. Це моделює загальний метод заголовка / нижнього колонтитула для подання стилів та самого відображення сторінок;

– app / helpers – підкаталог helpers містить всі допоміжні класи, використовувані для підтримки класів моделі, уявлення і контролера. Це допомагає зберегти розмір моделі, уявлення і код контролера невеликим, сфокусованим і незасміченим;

– app / assets – підкаталог створений для збереження таблиць стилів та зображень (фотографій) проекту. Раніше тут створювали javascript файли, але від Rails 6.0 версії javascript перенесли в інше місце;

– app / javascript – каталог в якому зберігаються VueJS, ReactJS, Angular чи JavaScript файли.

На цьому етапі ми ознайомились з структурою проекту і можемо переходити до діаграми архітектури авторизації користувача, яка зображена на рисунку 2.8. Для авторизації було використано гем «devise».

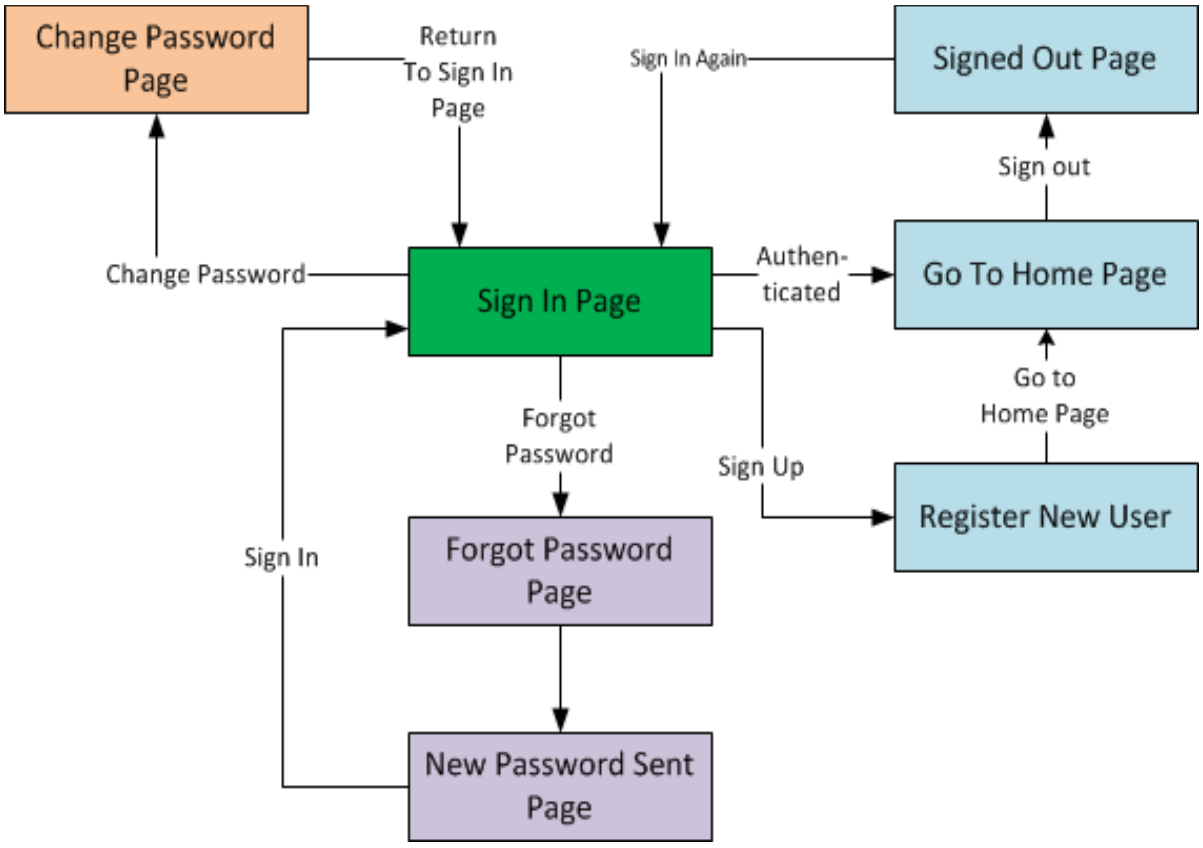

Рисунок 2.8 - Діаграма архітектури авторизації користувача на веб-сайті для ательє

На рисунку 2.8 чудово демонстровано всі веб-сторінки бібліотеки «devise». Основною сутністю на цій діаграмі є сторінка авторизації. Якщо користувач користується посиланням для зміни паролю, його відправляє на сторінку призначену для цього. Після збереження даних, користувач знову переміщається на сторінку авторизації. У випадку, коли користувач не може згадати свій пароль, він скористається посиланням «forgot password». На тій веб-сторінці потрібно заповнити деякі дані свого профілю, якщо введені дані існують, тоді користувач

переміститься на сторінку створення нового паролю, а вже після створення, знову на сторінку авторизації. Якщо користувач не має свого облікового запису, тоді він може скористатись сторінкою реєстрації і після заповнення якої він опиниться на основній сторінці веб-сайту. У випадку, коли користувач вже має створений акаунт, ввівши електронну адресу і пароль, він теж опиниться на основній сторінці застосунку-ательє. Перебуваючи на основній сторінці, є можливість вийти зі свого облікового запису, натиснувши на кнопку «Log out». Після цієї дії користувач знову опиниться на сторінці авторизації.

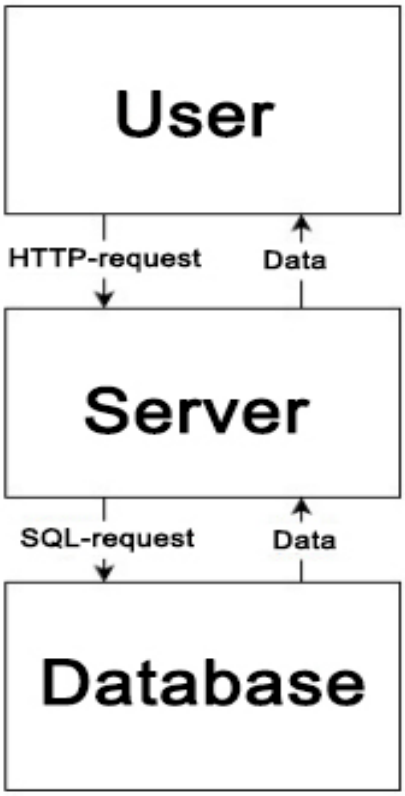

Рисунок 2.9 - Архітектура веб-сайту ательє

Основною сутністю архітектури веб-сайту для замовлення пошиття одягу є сервер, адже він виконує майже всю роботу. Його робота полягає в прийманні запитів зі сторони користувача, та у відповідь на ці запити надсилати потрібні дані зі сховища даних. У сервері розміщена логіка доступу до бази даних. Саме сервер унеможливлює отримання особистих даних користувачів для інших осіб.

Клієнт не повинен на пряму взаємодіяти зі сховищем даних. Клієнтська частина повинна містити тільки бізнес-логіку, та зберігати стани веб-сайту. На стороні клієнта можлива перевірка правильності вводу даних, або деякі прості операції.

Останньою сутністю архітектури веб-застосунку ательє являється бази даних. Саме тут необхідно створити всі умови для збереження у відповідних полях усі дані. В цій сутності використовують ключі та зв'язки для структури та цілісності даних.

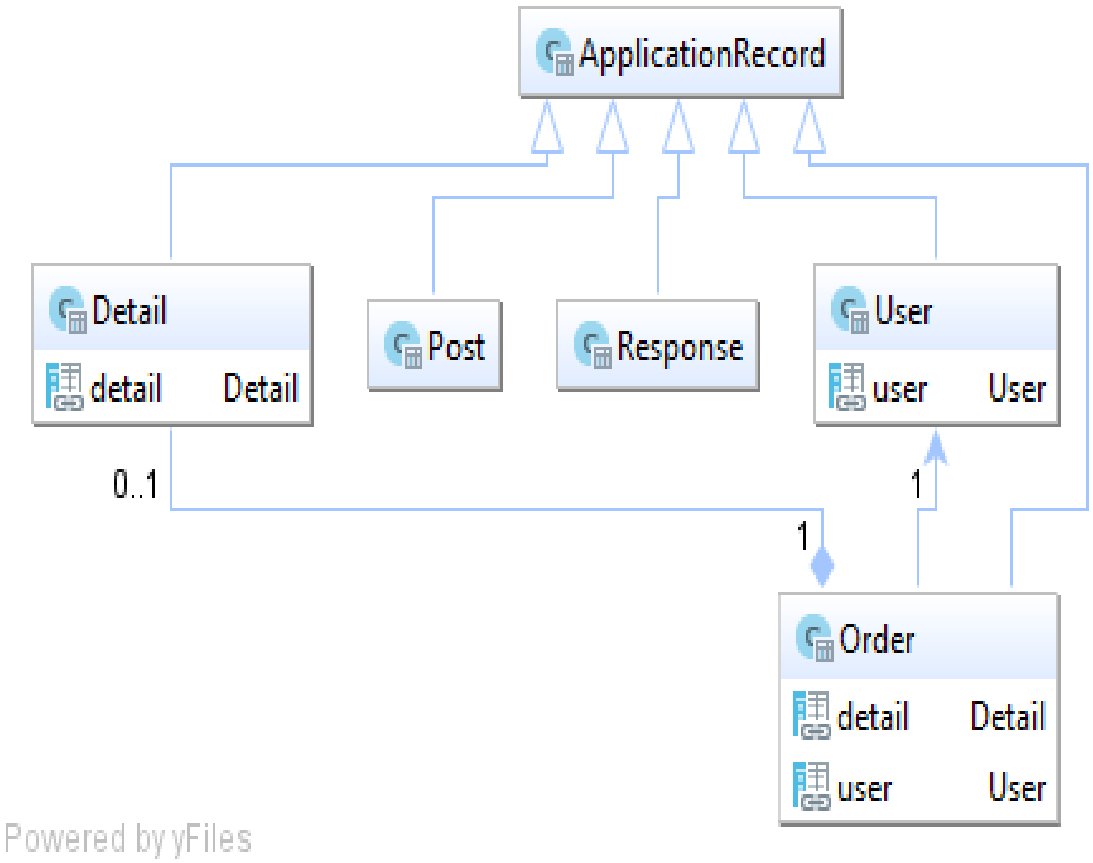

Рисунок 2.10 - Діаграма залежності моделей веб-сайту ательє

На діаграмі залежності моделей застосунку продемонстровано, що всі моделі наслідуються від основного класу «AplicationRecord» (рис. 2.10). Також показано зв'язки замовлення з користувачами та деталями замовлення.

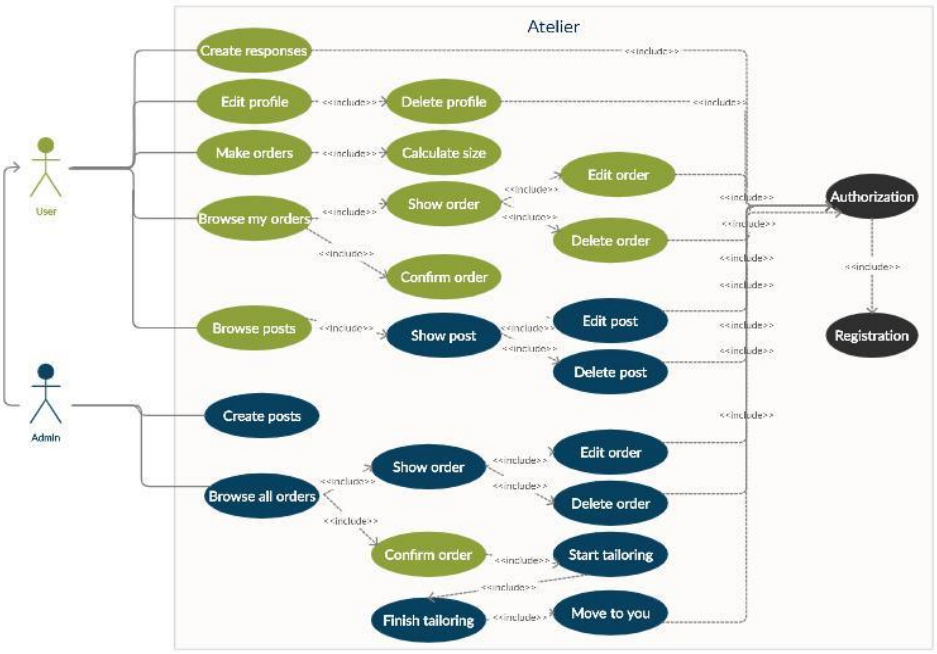

Рисунок 2.11 - Діаграма взаємодії з веб-аплікацією ательє замовника та адміністратора

Діаграма взаємодії була створена для відображення можливих функцій користувача, та адміністратора ательє (рисунок 2.11). З самого початку кидається в очі те, що для будь-якої дії на цьому веб-сайті необхідно авторизуватись або зареєструватись. І це не залежить від того, чи користувач є адміністратором, чи звичайним користувачем (замовником). Для зручності всі функції притаманні для звичайного користувача зображено зеленим кольором, а для адміністратора ательє – синім.

Користувач має можливість створювати відгуки про роботу ательє, веб-сайту, персоналу та інше. Можливість редагування своїх особистих даних теж входить до списку можливостей замовника. Редагування своїх даних включає видалення облікового запису користувача. Створення замовлення включає в себе обчислення розміру одягу для замовника. Перегляд всіх своїх замовлень містить посилання для перегляду окремого замовлення, та посилання для зміни стану замовлення на «підтверджено». А сторінка перегляду окремого замовлення містить сторінку його редагування або видалення. А ще, на основній сторінці веб-сайту замовник може переглядати всі пости, новини, та інформацію про ательє.

Для того щоб не описувати геть усі можливості адміністратора, зазначимо, що адміністратор має всі ті ж самі можливості що і замовник, та декілька недоступних для простого користувача. До недоступних відносяться: створення нових постів, перегляд усіх замовлень всіх користувачів. Звісно, створення постів, включає їхнє редагування та видалення. Важливою справою адміністратора є зміна станів замовлення будь-якого замовлення. Після того як замовник змінив стан замовлення на «підтверджено», адміністратор повинен якось інформувати замовників про прогрес. Тому для автоматизації цих справ, після зміни кожного з станів, буде слатись сповіщення на електронну адресу замовника.

Завдяки діаграмі взаємодії адміністратора і замовника з застосунком для замовлення пошиття одягу, необов'язково описувати всі ці інструкції в документації проекту.

2.3 Використані технології та мови програмування для веб-ательє

2.3.1 Мова програмування Ruby

Веб застосунок для ательє було вирішено розробляти на мові програмування Ruby. Ruby - одна з найбільш простих для вивчення мов. Існує багато навчальних ресурсів, а вся потрібна інформація міститься в open source. Пошук вирішення проблем не вимагає додаткових витрат з вашого боку, тільки якщо знадобиться якесь дуже специфічне рішення, але це навряд чи трапиться на початковому етапі.

Ruby – інтерпретована мова програмування високого рівня. Володіє незалежною від операційної системи реалізацією багатопоточності, чіткою динамічною типізацією, «збирачем сміття» і багатьма іншими можливостями [8]. Ця мова підтримує багато різних парадигм програмування, перш за все об'єктно-орієнтований підхід. Ruby був задуманий в 1993 році (24 лютого) японцем Юкіхіро Мацумото, який прагнув створити мову, яка поєднує всі якості інших

мов, що сприяють полегшенню праці програміста. Ruby увібрала в себе найкращі риси Perl, Java, Python, Smalltalk, Eiffel, Ada і Lisp. Ruby поєднує в собі Perl-подібний синтаксис із елементами мови Smalltalk.

В Ruby все – об'єкт. Для кожної частинки інформації або коду можуть бути визначені власні властивості і дії. В об'єктно-орієнтованому програмуванні властивості називаються змінними об'єкта, а дії – методами. Найчастіше об'єктно-орієнтований підхід Ruby може бути продемонстрований парою рядків коду, в яких проводиться дія над числом. У багатьох мовах, числа та інші примітивні типи даних не є об'єктами. Ruby під впливом мови Smalltalk дозволяє задати методи і змінні об'єкта всім типам даних. Це спрощує використання Ruby, так як правила застосовні до об'єктів - застосовані до всього Ruby.

Ruby дуже гнучка мова, так як вона дозволяє його користувачам вільно міняти її частини. Основні частини Ruby можуть бути видалені або перевизначені за бажанням. А існуючі частини можна змінювати. Ruby намагається ні в чому не обмежувати користувача. Наприклад, додавання виконується операцією плюс (+). Якщо ви хочете використовувати більш читане слово plus – ви можете додати такий метод прямо у внутрішній клас Numeric.

Блоки в Ruby теж є відмінним джерелом гнучкості. Розробник може додати замикання до будь-якого методу, описуючи, як цей метод повинен діяти. Замикання – це блок, що є однією з найбільш популярних конструкцій для тих, хто прийшов у світ Ruby зі світу імперативних мов програмування, таких як PHP або Visual Basic. Створення блоків було натхненне функціональними мовами програмування.

На відміну від багатьох об'єктно-орієнтованих мов програмування, Ruby навмисно надає лише одиночне наслідування. Але Ruby також надає концепцію модулів. Модулі – це колекції методів. Класи можуть вільно вміщувати в собі модуль і отримувати всі його методи. В основному, програмісти що працюють з мовою програмування Ruby знаходять це більш прозорим, ніж множинне спадкування, яке може бути досить складним і мати будь-які обмеження.

Оскільки в Ruby пунктуація зустрічається досить рідко і, зазвичай, в якості ключових слів використовується англійська мова, деякі знаки пунктуації використовуються для прикраси Ruby. Ruby не потребує оголошення змінних. У ньому використовуються прості угоди по іменування, щоб розділити. області видимості змінних, наприклад:

- var - може бути локальною змінною;
- @var - змінна об'єкта;
- $var - глобальна змінна.

Дана символіка підвищує читабельність, дозволяючи програмісту легко ідентифікувати роль кожної змінної. Це також дозволяє не використовувати обтяжливе self для кожного об'єкта.

Ruby сповнений іншими особливостями і конструкціями, і ось деякі з них:

- в Ruby є конструкції для обробки винятків, як в Java або Python, які дозволяють простіше працювати з помилками;
- в Ruby представлений справжній збирач сміття типу mark-and-sweep (позначити і відчистити). Не потрібно вручну відстежувати кількість посилань в сторонніх бібліотеках;
- писати розширення на C в Ruby простіше, ніж в Perl або Python за допомогою дуже елегантного API для виклику Ruby з C. Воно включає в себе виклики для вбудовування Ruby в програмне забезпечення, щоб використовувати його як скриптову мову. Також доступний інтерфейс SWIG;
- ruby може довантажувати сторонні бібліотеки динамічно, якщо дозволяє операційна система;
- в Ruby реалізовані незалежні від операційної системи потоки. На будь-яких платформах, де ви запускаєте Ruby, ви також можете використовувати багатопоточність, не залежно від того, чи підтримує дана система потоки чи ні. Ви можете скористатися наявними можливостями багатопоточності навіть в MS-DOS;

– ruby відрізняється високою переносимістю: ця мова була розроблена здебільшого на GNU / Linux, але працює на багатьох типах UNIX, macOS, Windows, DOS, BeOS, OS / 2, і так далі;

Ruby, як мова програмування має декілька різних реалізацій. Вони бувають дуже корисні в різних ситуаціях, надають велику інтеграцію з іншими мовами або оточеннями.

Список інших реалізацій мови Ruby:

– jRuby – це Ruby, реалізований на JVM (Java Virtual Machine), використовує оптимізований JIT-компілятор, збирач сміття, нативні потоки, інструментальну екосистему і величезну кількість бібліотек JVM [9];

– rubinius – це «Ruby написаний на Ruby». Реалізовано на основі LLVM - віртуальній машині, на ній створено інші відомі мови [10];

– ironRuby – це реалізація «інтегрована з .NET Framework» [11];

– cardinal - це «компілятор Ruby для віртуальної машини Parrot» (Perl);

– mruby - це легка реалізація Ruby, яка може бути підключена і вбудована в додаток. Очолює розробку mruby творець мови Ruby, Юкіхіро Мацумото (Yukihiro Matsumoto) [12];

– magLev - це «швидка, стабільна реалізація Ruby з інтегрованим довготривалим зберіганням об'єктів і розподіленим відкритим кешем" [13].

2.3.2 Фреймворк Ruby on Rails

Ruby on Rails (RoR) – це повноцінний, багаторівневий фреймворк, який створений для побудови веб-додатків, що використовують бази даних [14]. RoR заснований на архітектурі Модель-Представлення-Контролер (MVC).

Однією з основних переваг програмування на RoR є швидкість розробки. Практика показує, що швидкість розробки проектів на RoR збільшується на 30-40 відсотків в порівнянні з будь-якою іншою мовою програмування або

фреймворком. В першу чергу приріст швидкості розробки визначається великим набором готових до роботи штатних інструментів RoR, колосальним набором готових рішень в співтоваристві мови Ruby. Також багато розробників відмічають і простоту програмування на ньому. Однією з найважливіших частин культури RoR є соціальність. Вирішив проблему, – допоможи вирішити її іншим. Реалізував свій модуль, - поділися з спільнотою. Таким чином, на даний момент у відкритому доступі зберігаються тисячі готових рішень тих чи інших завдань. Системи автентифікації, авторизації, коментування, системи платежів, поштові розсилки та багато іншого (все те, що зазвичай розробляється «з нуля») впроваджуються реалізовані кимось іншим, протестовані і рекомендовані численним співтовариством.

Ruby on Rails – це фреймворк. Найчастіше фреймворк не дозволяє вам самодіяльність. Звичайно ж, в Ruby on Rails можна «винайти свій велосипед» і програмувати в будь-яких напрямках, не спираючись на стандарти. Але майже у всіх випадках цього робити не потрібно. Стандарти розміщення файлів в проекті, стандарти написання коду в проекті, загальні правила програмування в Ruby on Rails сильно структурують будь-який проект. За рахунок цього проект стає читабельним. Входження в проект новачків відбувається дуже швидко. Досвід показує, що будь-який новачок в проекті в перший же день роботи робить свої перші корисні правки. За рахунок цього не вважається великою проблемою, якщо розробку проекту спочатку вела одна команда програмістів, а підтримку проекту або доопрацювання – зовсім інша. Проект на RoR в загальному зрозумілий для будь-якого розробника [15].

При розробці будь-якого великого проекту постає питання. Як і хто буде тестувати проект? Часто у замовника немає коштів і бажання створювати цілі відділи тестування, до цього ж хочеться автоматизувати процес тестування. На відміну від інших фреймворків, в складі RoR існує багато рішень автоматизованого тестування. В інших мовах програмування та фреймворках штатних засобів тестування немає. Звичайно, є сторонні розробки, що дозволяють організувати автоматичне тестування проекту на інших мовах програмування, як

PHP, але вони не ставляться "з коробки" і про їх використання програмісти частіше не замислюються. У проекті на Ruby on Rails, в ідеалі, код проекту не пишеться до тих пір, поки під цей код не написані тести. RoR ідеологія передбачає початкове використання методів BDD або TDD.

Кешування проектів – один з найважливіших етапів розробки великого web-проекту. У PHP є різноманітні варіанти кешування даних проекту. Ці варіанти та інструменти прикручуються, прилаштовуються, прикріплюються збоку. До цього часу в співтоваристві PHP немає єдиної думки: що краще використовувати, як краще кешувати дані, та якими інструментами користуватися. Ruby on Rails в своїй базовій комплектації має штатні засоби кешування даних. При створенні самого проекту надаються інструменти, що дозволяють реалізувати кешування даних на проекті. Ви можете кешувати цілі сторінки або ж, навіть блоки коду. Можете кешувати результати ваших запитів і ActiveRecord-моделі. Кешувати можна як за допомогою memcached або redis, так і іншими засобами. Для реалізації кешування на Ruby on Rails проект вам в 95 відсотках випадків не буде потрібно нічого крім уже готових і штатних рішень.

Дуже часто зустрічається ситуація, коли хтось зробив проект, а потім розуміє, що для продовження розвитку проекту необхідна англійська версія. Розробники багатьох фреймворків при цьому починають заводити розмови про те, що це не було передбачено заздалегідь, що це довго і вкрай складно. Рельси в базовій комплектації мають інструменти локалізації. Ви можете передбачити необхідність підтримки різних мов на сайті як спочатку, так і в подальшому. RoR вміє роздавати різні шаблони для різних мов, містить конфігураційні файли з перекладами термінів та інші штатні інструменти для локалізації проекту [16].

Найчастіше в багатьох проектах ми можемо бачити картину, коли адреса певної сторінки величезна і незрозуміла. В Ruby on Rails є стандартна можливість гнучко налаштовувати ваш роутинг, вид адреси, назви основних розділів. Є можливість швидко змінити адреси в одному місці без необхідності її зміни в усьому проекті. У спільноті RoR-розробників активно використовуються ідеологія REST. Адреси сторінок в проектах на Ruby on Rails завжди зрозумілі,

красиві, прості. У Ruby on Rails прекрасно реалізований інструментарій, що дозволяє затверджувати вхідні дані. Ваші користувачі заповнюють форми, де потрібно перевіряти правильність введення адреси електронної пошти, наявність пароля або необхідну мінімальну довжину логіна.

Повсякденна проблема багатьох проектів – неможливість розуміти засоби та інструменти контролю структури бази даних. Зміни в структуру бази даних часто вносяться вручну і прямо в базу. Через це в проекті з'являються численні незрозумілі поля і таблиці, про які ніхто нічого не пам'ятає. В Ruby on Rails існують штатні інструменти роботи з базами даних – «міграції». Структура бази даних зберігається в коді програми і конфігурується з проекту. Ваша структура буде завжди в репозиторії, будь-яка зміна структури буде задокументованою і прив'язаною до певного комміту в репозиторію.

Фреймворк Ruby on Rails заточений під безпеку проекту. При використанні інструментів RoR виключені SQL ін'єкції і XSS атаки. Всі вхідні параметри екрануються за замовчуванням. Виведені змінні в шаблонах також екрануються, тільки якщо ви не вказали зворотної опції. У розробника немає шансів допустити помилки безпеки (не без винятків, звісно).

2.3.3 Шаблон проектування MVC

Шаблон проектування MVC передбачає поділ даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти: модель, представлення і контролер – таким чином, що модифікація кожного компонента може здійснюватися незалежно, як продемонстровано на рисунку 2.12. Важливе зауваження: концепція MVC не прив'язана до якоїсь конкретної мови програмування, вона також не прив'язана і до використовуваної парадигми програмування. Тобто, ви цілком можете проектувати свій додаток по MVC, при цьому не застосовуючи ООП. Мета цього шаблону – це гнучкий дизайн програмного забезпечення, що повинен полегшувати всі зміни чи розширення

програм, а також давати можливість повторного використання окремих компонентів програми [17]. Окрім того, використання даного шаблону у великих системах сприяє впорядкованості їхньої структури, а також робить їх більш зрозумілими за рахунок зменшення складності.

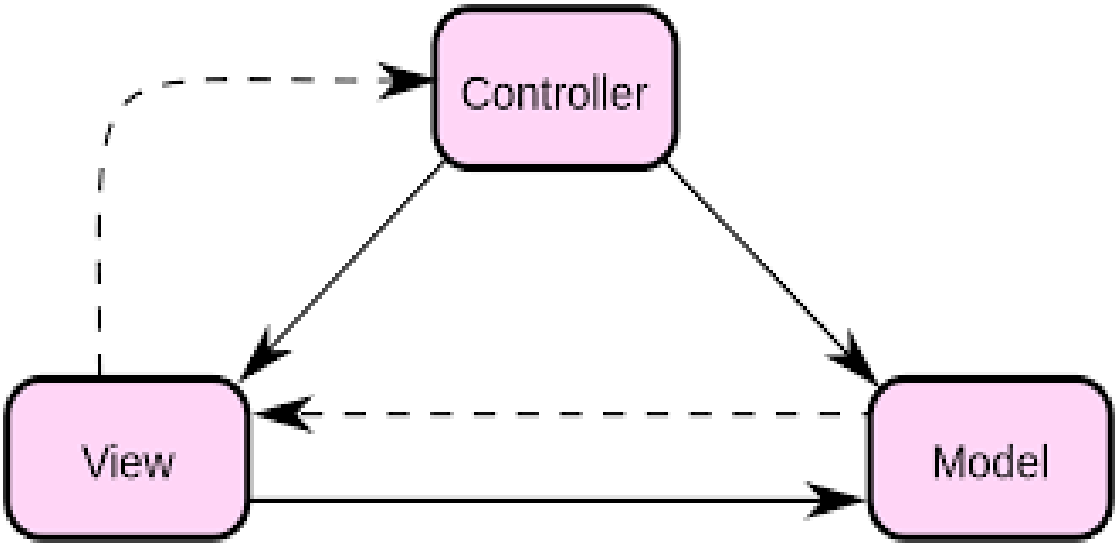

Рисунок 2.12 – Зображення архітектурного шаблону MVC

У цьому шаблоні модель (Model) виконує роль зберігання даних та їх структуру. Вигляд (View) відповідальний за інтерфейс вашої програми, тобто представлення цих даних користувачеві. Контролер (Controller) керує компонентами, він отримує сигнали у вигляді відповіді на дії користувача (зміна положення курсора миші, ввід даних в текстове поле, натискання кнопки) і передає дані у модель. Контролер отримує вхідні дані й перетворює їх на команди для моделі або вигляду.

Модель є центральним компонентом шаблону MVC і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Вона стосується прямого керування даними, логікою та правилами застосунку. Модель вміщує в собі ядро даних та основний функціонал їхньої обробки і не залежить від процесу вводу чи виводу даних.

Вигляд являє собою представлення інформації на стороні користувача, одержуване на виході, наприклад графік чи діаграму. Одночасно в одному проекті

можуть співіснувати кілька виглядів (представлень) однієї і тієї ж інформації, наприклад гістограма для керівництва компанії й таблиці для інших працівників. А ще вигляд може мати декілька взаємопов'язаних областей, наприклад різні таблиці і поля форм, в яких відображаються дані.

До функцій контролера належить відстеження визначених подій, що виникають в результаті дій користувача. Контролер дозволяє структурувати код шляхом групування пов'язаних дій в окремий клас. Наприклад, у типовому MVC-проекті може бути користувацький контролер, що містить групу методів, пов'язаних з управлінням обліковим записом користувача, таких як реєстрація, авторизація, редагування профілю та зміна пароля.

Зареєстровані події транслюються в різні запити, які спрямовуються об'єктам або компонентам моделі, що є відповідальними за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати відображення інформації для різних компонентів. Таким чином, якщо користувач через контролер занесе зміни до моделі даних, то інформація, подана одним або декількома візуальними компонентами, буде автоматично відкоригована відповідно до змін, що відбулися.

2.3.4 Бази даних PostgreSQL

PostgreSQL (Постгрес) – це не просто реляційна, а об'єктно-реляційна СУБД [18]. PostgreSQL базується на мові SQL і підтримує велику кількість різноманітних можливостей. Це дає йому деякі переваги над іншими SQL базами даних з відкритим вихідним кодом.

Фундаментальна характеристика об'єктно-реляційної бази даних PostgreSQL – це підтримка об'єктів та їхньої поведінки, включаючи типи даних, функції, домени, операції і також індекси. Це робить Постгрес неймовірно гнучким і надійним [19]. А ще потрібно зазначити, що він вміє створювати, зберігати та видавати складні структури даних.

Існує дуже великий список типів даних, що підтримує Постгрес. Крім числових, з плаваючою точкою, текстових, булевих і інших очікуваних типів даних, PostgreSQL може похвалитися підтримкою uuid, грошового, геометричного, бінарного типів, мережевих адрес, бітових рядків, текстового пошуку, xml, json, масивів, композитних типів і діапазонів, а також деяких внутрішніх типів для ідентифікації об'єктів.

PostgreSQL забезпечує зберігання різних типів мережевих адрес. Тип даних CIDR (безкласова маршрутизація інтернет домену, Classless Internet Domain Routing), та угодою для мережевих адрес IPv4 і IPv6. Ось кілька прикладів:

- 192.168.100.128/25;
- 10.1.2.3/32;
- 2001: 4f8: 3: ba: 2e0: 81ff: fe22: d1f1 / 128;
- :: FFFF: 1.2.3.0/128.

Також для зберігання мережевих адрес доступний тип даних INET, який використовується для IPv4 і IPv6 хостів, де підмережі є необов'язковими. Тип даних MACADDR може використовуватися для зберігання MAC-адрес для ідентифікації обладнання, таких як 08-00-2b-01-02-03.

У MySQL і MariaDB теж є INET функції для конвертації мережевих адрес, але вони не надають типи даних для їх внутрішнього зберігання. У Firebird теж немає типів для зберігання мережевих адрес.

Оскільки Постгрес – це об'єктно-реляційна база даних, масиви значень теж можуть зберігатися для більшості існуючих типів даних. Зробити це можна шляхом додавання квадратних дужок до специфікації типу даних для стовпця або за допомогою виразу ARRAY. Розмір масиву може бути заданий, але це не є обов'язково. MySQL, MariaDB, і Firebird так не вміють. Щоб зберігати такі масиви значень в традиційних реляційних базах даних, доведеться використовувати обхідний шлях і створювати окрему таблицю з рядками для кожного із значень масиву.

Підтримка JSON в PostgreSQL дозволяє користувачам перейти до зберігання schema-less даних в SQL сховищі даних. Це може бути корисно, якщо структура даних вимагає певної гнучкості. Наприклад, коли в процесі розробки структура все ще змінюється, або невідомо які поля буде містити об'єкт даних. Тип даних JSON забезпечує перевірку коректності JSON, який дозволяє використовувати спеціалізовані JSON оператори і функції, вбудовані в Постгрес для виконання запитів і маніпулювання даними. Також доступний тип JSONB – це двійковий різновид формату JSON, у якому пробіли видаляються, сортування об'єктів не зберігається, натомість вони зберігаються найбільш оптимальним чином, і зберігається тільки останнє значення для ключів-дублікатів. JSONB зазвичай є кращим форматом, оскільки вимагає менше місця для об'єктів, може бути проіндексованим і обробляється швидше, так як не вимагає повторного синтаксичного аналізу.

В MySQL 5.7.8 і MariaDB 10.0.1 була додана підтримка вбудованих об'єктів JSON. Але, хоча існує безліч функцій і операторів для JSON, які тепер доступні в цих базах даних, вони не індексуються так, як JSONB в PostgreSQL. Firebird поки що не підтримує об'єкти JSON тільки у вигляді тексту.

Якщо раптом так трапиться, що великого списку типів даних Постгреса програмістам виявиться недостатньо, то вони можуть використовувати команду CREATE TYPE, щоб створити нові типи даних. Оскільки інші бази даних не є об'єктно-реляційними, то вони і не пропонують таку потужну функціональність.

Постгрес прагне відповідати стандарту ANSI-SQL: 2008, відповідає вимогам ACID (атомарність, узгодженість, ізольованість і надійність) і відомий своєю посилальною і транзакційною цілісністю. Первинні ключі, що обмежують і каскадні зовнішні ключі, унікальні обмеження, обмеження NOT NULL, перевірочні обмеження та інші функції забезпечення цілісності даних дають впевненість, що тільки коректні дані будуть збережені.

Поточні обмеження перераховані нижче:

Таблиця 2.1 – Обмеження БД PostgreSQL

Максимальний розмір бази даних	необмежений
Максимальний розмір таблиці	32 ТБ
Найбільшою довжиною строки	1,6 ТБ
Максимальний розмір поля	1 GB
Максимальна кількість рядків в таблиці	необмежений
Максимальна кількість стовпців в таблиці	250-1600 залежно від типу
Максимальна кількість індексів в таблиці	необмежений

Сховище Постгрес створене з використанням об'єктно-реляційної моделі, воно підтримує складні структури і широкий спектр вбудованих типів даних. Постгрес забезпечує розширену ємність даних і заслужив довіру дбайливим ставленням до цілісності даних. Можливо, вам не знадобляться всі ті просунуті функції зберігання даних, описані вище, але, оскільки потреби можуть швидко зрости, є безсумнівна перевага в тому, щоб мати все це під рукою.

У розробці простих сайтів PostgreSQL використовується не так часто, ніж MySQL чи MariaDB, але все ж ця пара з помітним відривом випереджає за частотою використання інші системи управління базами даних . При цьому в розробці складних сайтів і веб-додатків PostgreSQL випереджає опонентів. Більшість фреймворків (наприклад, Ruby on Rails, Yii , Symfony, Django) підтримують використання PostgreSQL в розробці [19].

3 РОЗРОБКА ДИЗАЙНУ ТА ФУНКЦІОНАЛУ ВЕБ-АТЕЛЬЄ

3.1 Розроблення функціоналу веб-застосунку для ательє

В будь-якій програмі повинен бути якийсь мінімальний набір функцій чи методів, завдяки яким ця програма буде виконувати їх і приносити задоволення своїм замовникам та клієнтам. Майже для усіх веб-сайтів, насамперед, потрібно розробити можливість реєстрації та авторизації користувачів. Devise один з кращих гемів для автентифікації в rails-додатках.

Devise – це ruby-гем, що надає можливості для автентифікації в rails-додатках [20]. Devise працює в зв'язці з гемом Warden, який в свою чергу надає сам механізм для аутентифікації в rack-базованих ruby-додатках [21]. Основні особливості Devise описані нижче:

- заснований на Rack;
- є закінченим MVC-рішенням, заснованим на Rails;
- дозволяє вхід в систему по декількох моделях одночасно;
- заснований на модульній основі: використовує тільки те, що дійсно необхідно для проекту.

Для шифрування використовується гем «bcrypt-ruby». Він надає просту обгортку для роботи з паролями. В основі лежить криптографічна хеш-функція *bcrypt()*. А гем «orm_adapter» – надає єдину точку входу для використання основних функцій Ruby ORM. Також разом з «devise» установлюється гем «thread_safe» (надає потоко-безпечні колекції і утиліти для Ruby), та «railties» (внутрішні компоненти Rails, такі як завантажувачі додатки, плагіни, генератори і rake-завдання). Для завантажування всіх пакунків цього гему потрібно тільки прописати його назву в файлі «Gemfile» (файл описано в пункті 2.2 Структура проекту), та в терміналі запустити команду: *bundle install.*

В даний момент, гем був викачаний, але ще ніяк не взаємодіє з нашим веб-сайтом. Для виконання установки Devise потрібно виконати в терміналі

наступну команду: *rails generate devise:install*. Після виконання цієї команди, у вашому проекті відбулося автоматичне налаштування гему «devise». Наступним кроком була створена модель «Користувач (User)». За замовчуванням у вас буде підключено 5 модулів. Але ви можете відредагувати цей список:

– database Authenticatable: надає можливість входу в систему на основі зашифрованого і зберігається в базі даних пароля. Вхід може бути виконаний за допомогою відправки POST-запиту або за допомогою HTTP Basic Authentication;

– omniauthable: підтримка Omniauth (авторизація через соціальні мережі);

– confirmable: дозволяє відправляти лист з інструкціями для підтвердження акаунта, створеного під час реєстрації;

– recoverable: дозволяє відновлювати забутий пароль. Відправляє інструкції по відновленню на пошту;

– registerable: управляє реєстрацією користувачів, дозволяє редагувати і видаляти акаунти;

– rememberable: дозволяє запам'ятовувати користувачів на основі cookies. Управляє створенням і видаленням токенів;

– trackable: веде статистику кількості входів, враховує час і IT-адреси;

– timeoutable: відповідає за тривалість сесії активності користувача в системі;

– validatable: надає інструменти валідації e-mail і пароля. Модуль може бути легко налаштований, ви можете визначити власні валідатори;

– lockable: блокує акаунт після зазначеного в налаштуваннях кількість невдалих спроб авторизації. Акаунт може бути розблоковано за допомогою email або через певний період часу.

В нашій моделі активними є 5 наступних модулів: *database_authenticatable, registerable, recoverable, rememberable, validatable.* Також ми описали зв'язок до

замовлень: *has_many :orders*. Цей зв'язок означає, що один користувач може мати багато замовлень.

Останнім кроком опису моделі «користувач» є опис валідації необхідних форм вводу даних. Оскільки таблиця «користувач» містить велику кількість даних, потрібно вибрати ті дані, які обов'язкові до заповнення. Такими полями є: «електронна адреса», «ім'я», «прізвище», «пароль», «стать», «номер телефону» та «адреса». Поле «ідентифікатор» створюється унікальним автоматично, а «адміністратор» зазвичай приймає значення «false» для кожного користувача. Розробник присвоює «true» декільком користувачам, які працюють в ательє і повинні реагувати на замовлення. Для перевірки поля «електронна адреса» створено регулярний вираз, що перевіряє правильність вводу. Перевірка поля «пароль» здійснюється в 2 етапи. Першим етапом є перевірка кількості символів (зазначений діапазон від 6 до 40 символів). Під час другого етапу необхідно повторити цей пароль. Поле «номер телефону» містить тільки числа і кількість від 10 до 15 символів. У всіх інших полях перевіряється тільки кількість символів.

У створенні контролера для користувача потреби не було, тому що гем «Devise» має набір стандартних функцій.

Для того, щоб все працювало належним чином, залишилось тільки ініціалізувати шляхи та виконати перевірку авторизації перед будь-якою дією. Цей рядок коду (*before_action :authenticate_user!*) знаходиться в основному контролері аплікації, саме він виконує перевірку авторизації користувача, перед будь-якою дією. Щоб описати всі шляхи для користувача, достатньо додати *devise_for :users* до файлу «routes.rb». А вже після комбінації усіх цих кроків, користувачу відкривається наступний список можливостей на сторінці ательє:

- реєстрація нового користувача;
- авторизація;
- редагування та перегляд особистих даних;
- можливість змінити пароль;

– перегляд всіх веб-сторінок (доступних в залежності від того, чи поточний користувач має права адміністратора чи ні);

– створення, редагування, видалення замовлень та відгуків;

– можливість видалення свого акаунту.

Коли користувач здійснив реєстрацію чи авторизацію на сайті для ательє, його автоматично відправляє на головну сторінку веб-аплікації. Ця сторінка містить всі пости (новини або важливу інформацію для клієнтів).

Для реалізації створимо модель «пост». Оскільки ця модель не сполучена жодним зв’язком до інших сутностей, виконаємо тільки валідації для форм: «заголовок» та «тіло».

Створення усіх посилань (відображення, створення, видалення та редагування постів) в файлі «routes.rb» робить рядок: *resources :posts*. Але для того, щоб відобразити всі пости на головній сторінці потрібно додати наступне: *root 'posts#index', as: 'home'*.

Наступним кроком буде створення контролера для постів (рис. 3.1). Тепер потрібно зрозуміти яким чином відбувається відображення, створення, редагування та видалення постів.

```
def index
  @post = Post.all
  respond_to do |format|
    format.html { Post.all }
    format.json { render json: Post.all }
  end
end
```

Рисунок 3.1 - Контролер «posts» виведення усіх постів

Метод «індекс» в даному прикладі через Rails ORM звертається до бази даних, робить запит на усі пости та записує їх у змінну «post». Пізніше відправляє ці всі дані на html сторінку index в каталозі «posts» і також формує дані в json формат. В багатьох проектах зовнішній вигляд розробляють на інших мовах

програмування, і тому для відправлення чи отримання даних використовують json формат. А ще цей формат дуже часто використовують, коли через API дані з основного сайту беруть для мобільних додатків та телеграм-ботів.

Перш ніж розглянути такі методи як «edit», «show», «new», «create» і «update», потрібно описати приватний метод «post_params» (приватний тому, що використовується тільки в межах цього класу і ніде інде) і «find_post» зображено на рис. 3.2.

```
private

def post_params
  params.require(:post).permit(:title, :body, :nested_files)
end

def find_post
  @post = Post.find(params[:id])
end
```

Рисунок 3.2 – Контролер «posts», методи «post_params» та «find_post»

Функція «*post_params*» викликає параметри необхідні для поста, тобто для його створення і редагування. І як ми бачимо на рис. 3.2, дозволяється параметр «заголовок», «тіло» і «вкладені файли». Саме цей метод викликатимуть інші функції, для яких необхідні ці параметри.

Назва метода «find_post» говорить сама за себе. В цьому методі виконується пошук певного поста за його ідентифікатором. Даний метод був винесений в окремий для того, щоб не повторювати однаковий код всюди, а тільки викликати в потрібний момент. Сам виклик відбувається завдяки цьому рядку: *before_action :find_post, only: %i[show edit update destroy]*. У ньому зазначається, що перед дією «show», «edit», «update» чи «destroy» відбувається виклик метода «find_post».

```ruby
def new
  @post = Post.new
end

def create
  @post = Post.new(post_params)

  respond_to do |format|
    if @post.save
      format.html { redirect_to @post, notice: 'Order was successfully created.' }
      format.json { render :show, status: :created, location: @post }
    else
      format.html { render :new }
    end
  end
end
```

Рисунок 3.3 - Контролер «posts», методи «new» та «create»

Метод «new» (рис.3.3) рендерить сторінку для створення нового поста. А метод «create» викликається коли будь-який адміністратор підтверджує створення нового посту, натиснувши на кнопку. Цей метод записує усі введені дані з форм в змінну «post» (2 рядок цього методу) і намагається її зберегти до бази даних завдяки інтеграції ORM у Ruby on Rails. І далі стоїть умова: якщо пост збережено, то користувач переноситься на сторінку новоствореного поста та виводиться повідомлення (також ми не забули про json). А якщо умова не виконалась, то користувач знову переходить на сторінку створення нового поста.

```ruby
def show
  respond_to do |format|
    format.html { @post }
    format.json { render json: @post }
  end
end

def edit; end

def update
  respond_to do |format|
    if @post.update(post_params)
      format.html { redirect_to @post, notice: 'Post was successfully updated.' }
      format.json { render :show, status: :ok, location: @post }
    else
      format.html { render :edit }
    end
  end
end

def destroy
  @post.destroy
  respond_to do |format|
    format.html { redirect_to posts_path, notice: 'Post was successfully destroyed.' }
    format.json { head :no_content }
  end
end
```

Рисунок 3.4 - Контролер «posts», методи «show», «edit», «update» та «destroy»

Метод «show», що на рисунку 3.4, дуже схожий на метод «index». Відмінністю між ними є те, що в першому виводиться тільки один пост з конкретним ідентифікатором, а в другому – всі.

А от метод «edit» створений тільки для виклику при рендерингу сторінки для редагування якогось поста.

Настала черга для функції «update». Перед викликом цієї функції відбувається пошук того поста який потребує змін. Потім на сторінці редагування поста (описана вище), у створених формах відбуваються зміни, які вводить користувач. Після завершення правок користувач натискає на кнопку для збереження змін і всі правки методом patch відправляються до контролера. Якщо виконується умова, що пост з певним ідентифікатором оновлено, тоді веб-сайт відправляє користувача на сторінку відображення того поста (при цьому виводить сповіщення про успішне збереження змін). А у випадку, коли умова не виконалась, тоді сайт знову відправляє користувача на сторінку редагування.

У випадку, якщо той чи інший пост уже не потрібний, прийдеться його видаляти. Для видалення в контролері аналогічно було створено метод «destroy». Він працює наступним чином: з веб-аплікації приходить ідентифікатор того поста який потрібно видалити, тоді в базі даних шукають потрібний пост і застосовують для нього метод «destroy». Після цього виконується відправлення користувача до переліку всіх постів (на головну сторінку веб-сайту).

Всі інші контролери працюють за тим самим принципом, що описаний вище. Тому в подальшому немає потреби описувати все те саме, тільки з іншими назвами. І для інших контролерів будуть описані тільки ті речі, яких не існує в контролері «post», або вони кардинально різняться.

Оскільки контролер «responses» містить функціонал, як і контролер «posts», приступимо до контролера «pages». Він є надзвичайно простий, адже містить тільки однойменну сторінку з методом контролера «about». Цей метод рендерить файл з назвою «about.html.erb». В ньому виводяться дані про ательє, його місцезнаходження і викликаються методи контролера «responses» (відгуки).

У моделі відгуків відбувається тільки валідація деяких полів. В ній перевіряється наявність пустих полів, мінімальна та максимальна кількість символів, і вказано що можливі вкладені файли.

Тепер розглянемо дві моделі однієї сутності – це «order» та «detail». Модель «detail» теж не є складною. Вона містить зв'язок *belongs_to :order* до моделі «order» (замовлення). Цей зв'язок сполучає одне замовлення з одними деталями (позначається 1:1, one-to-one). А валідація в цьому файлі перевіряє наявність обов'язкових полів та кількість символів.

А от модель, яка називається «order», є набагато цікавішою. Вона має два зв'язки з іншими сутностями. Наприклад першим є зв'язок *belongs_to :user*. Ми знаємо що це зв'язок один до одного (одне замовлення відповідає одному користувачу). Але користувач має зв'язок багато до одного (багато замовлень може мати один користувач). Другим є зв'язок *has_one :detail, dependent: :destroy*. Це означає що одне замовлення має тільки одну деталь. Що ж означає значення після коми? Це написано для того, щоб коли ми видаляємо своє замовлення, видалялись і деталі замовлення. Це потрібно, щоб база даних не засмічувалась непотрібними даними. Також ми бачимо рядок, який вказує що деталі є вкладеними в замовлення (*accepts_nested_attributes_for :detail*).

На рисунку 3.5 ми бачимо, що до нашої моделі ми підключили бібліотеку під назвою «AASM». AASM починалась як плагін (act as state machine), але пізніше вона розширилася до бібліотеки. Бібліотека AASM більше не націлена лише на моделі ActiveRecord [22]. В даний час вона пропонує адаптери для багатьох ORM, але її можна використовувати для будь-якого класу Ruby, незалежно від того, який батьківський клас вона має.

Для того щоб використовувати AASM бібліотеку в проекті, ми зазначили що вона змінює стани в колонці бази даних під назвою: «state». Після цього необхідно ініціалізувати всі стани, які є потрібними на веб-сайті. Замовник вирішив, що п'яти наступних станів буде цілком достатньо:

– ініціалізовано (initialized);

– підтверджено (confirmed);
– розпочате шиття (tailoringStarted);
– шиття завершене (tailoringFinished);
– прямує до Вас (moveToYou).

```
include AASM

aasm column: 'state' do
  state :initialized, initial: true
  state :confirmed
  state :tailoringStarted
  state :tailoringFinished
  state :moveToYou

  event :confirm, after: :order_email do
    transitions from: :initialized, to: :confirmed
  end

  event :tailoring_start, after: :order_email do
    transitions from: :confirmed, to: :tailoringStarted
  end

  event :tailoring_finish, after: :order_email do
    transitions from: :tailoringStarted, to: :tailoringFinished
  end

  event :move_to_you, after: :order_email do
    transitions from: :tailoringFinished, to: :moveToYou
  end
end

private

def order_email
  UserMailer.order_email(self).deliver_now
end
```

Рисунок 3.5 - Модель "Замовлення" та (AASM)

Перший стан встановлюється автоматично при створенні будь-якого замовлення. Другий – коли користувач натиснув кнопку «Підтвердити» в своїй корзині (будь який користувач, окрім адміністратора бачить виключно свої замовлення, тому і підтвердити має змогу тільки своє замовлення). Всі інші стани має змогу змінювати тільки адміністратор сайту. Третій стан адміністратор може встановити в тому випадку, коли є назначений модельєр для поточного виробу і він приступив до роботи. Стан теж змінюється за допомогою кнопки (виконано

таким чином для простоти та зручності). Четвертий стан установлюється, коли у ательє вже є готовий замовлений виріб. П’ятий стан не є обов’язковим тому, що замовник може сам забрати одяг з ательє. Але якщо він не має змоги, тоді працівники відправлять його Новою поштою, змінять стан на останній і при отриманні замовлення користувач може видалити замовлення. Коли замовлення зникне, працівники будуть знати що замовлення прибуло до місця призначення.

Після опису ініціалізації всіх станів, нижче є описані події переходу від одного стану до іншого. Також зазначений порядок переходу (можливість переходу від першого і одразу до третього стану не зазначена в подіях, тому це не є можливим). В подіях також виконується код, що має викликати метод «order_mail» після зміни стану. Цей метод повинен відправляти повідомлення на електронну пошту замовника. Повідомлення після зміни кожного стану сповіщають про стадію готовності замовленого виробу. Для кожного стану заготовлене окреме повідомлення.

Залишилось описати контролер для замовлень. Цей контролер дуже схожий на усі інші. Різниться тільки трьома наступними пунктами: параметри замовлення містять вкладені деталі для них, метод «індекс» передбачає окремо вивід для адміністратора та для звичайного користувача і методи для зміни станів замовлення.

```
def order_params
  params.require(:order).permit(:name, :surname, :email, :phone, :address,
                                :gender, :state,
                                detail_attributes: %i[order_id clothing_name
                                                      color collar sleeve
                                                      length size fabric
                                                      wishes nested_files])
end
```

Рисунок 3.6 - Контролер "orders", параметри замовлення

На рисунку 3.6 зображений метод, що викликає необхідні параметри для замовлення. Від метода «post_params» він різниться тим, що містить в собі вкладений масив даних (detail_attributes) серед усіх параметрів. А в тому масиві знаходяться всі деталі замовлення.

```ruby
def index
  @orders = Order.all

  if current_user.admin == true
    respond_to do |format|
      format.html { @orders }
      format.json { render json: @orders, include: [:detail] }
    end
  else
    respond_to do |format|
      format.html { current_user.orders }
      format.json { render json: current_user.orders, include: [:detail] }
    end
  end
end
```

Рисунок 3.7 - Контролер "orders", метод "index"

Метод «index» (рис. 3.7) повинен перевіряти чи поточний користувач є адміністратором. Якщо умова справдилась, то виводяться всі замовлення будь-якого користувача. Якщо умова не була виконана, то це означає, що авторизований звичайний користувач. І в такому випадку виводяться всі замовлення поточного користувача. І в першому, і в другому випадках також включаються всі деталі та всі дані формуються в json формат.

```ruby
def confirm
  @order = Order.find(params[:id])
  @order.confirm!
  render 'show'
end

def tailoring_start
  @order = Order.find(params[:id])
  @order.tailoring_start!
  render 'show'
end

def tailoring_finish
  @order = Order.find(params[:id])
  @order.tailoring_finish!
  render 'show'
end

def move_to_you
  @order = Order.find(params[:id])
  @order.move_to_you!
  render 'show'
end
```

Рисунок 3.8 - Контролер «orders», методи для зміни станів

Останнім, третім пунктом, є методи для зміни станів AASM (рис. 3.8). Будь-який з цих методів викликається при натисканні відповідної кнопки на сайті. При створенні будь-якого замовлення, автоматично в БД ставиться статус «створено (created)». Коли користувач впевнився у всіх даних, які він заповнив, він натискає на кнопку для підтвердження замовлення. І тоді викликається метод «confirm», отримавши ідентифікатор потрібного замовлення, метод перезаписує старий стан на новий. А *tailoring_start!* – це подія, яка описана в моделі, і саме вона вказує, який стан потрібно змінити на потрібний нам.

3.2 Розроблення зовнішнього вигляду для веб-сайту

Для будь-якого веб-застосунку важливим атрибутом є приємний для людського ока вигляд і, звісно, зручний інтерфейс. Як у Ruby on Rails відображаються html файли? Для того, щоб у цьому фреймворку відображати зовнішній вигляд, використовується «препроцесор» erb. Його додають до кінця будь-якої назви html файлу і таким чином цей препроцесор повідомляє Rails, що потрібно відображати (рендерити) потрібну html сторінку [23].

З розділу про структуру проекту в дипломній роботі ми знаємо, що всі файли для зовнішнього вигляду знаходяться в каталозі «views». Але в цьому каталозі також знаходяться інші каталоги. Основний файл називається «application.html.erb», в каталозі «layouts». В ньому описана структура html і всередині нього викликаються усі інші файли зовнішнього вигляду. Завдяки RVM структурі, каталоги, які вміщають html, мають ту саму назву з моделями та контролерами (orders, pages, posts, responses). Розглянемо один для прикладу. В каталозі «ордер» містяться 4 фали html. Назва кожного з них відповідає назві метода в однойменному контроллері («edit.html.erb», «index.html.erb», «new.html.erb», «show.html.erb»). В такому випадку все структуровано і цілком зрозуміло для розробника.

Для того, щоб не загромаджувати файли, розробники Rails передбачили можливість розбивати один файл на кілька частин (парціалів). Такий файл дуже легко відрізнити від основних. Він містить нижнє підкреслення на початку назви. Іноді парціали зберігаються в тому ж каталозі, в якому його викликають. Але інколи треба винести загальні парціали. Тоді їх розміщують в каталозі «shared».

Для кращого розуміння, пояснення розробки зовнішнього вигляду будуть супроводжуватись зображеннями різних сторінок або їхніх значущих частин.

Для взаємодії з веб-сайтом, потрібно мати свій акаунт. Адже з самого початку, користувач стикається зі сторінкою авторизації або реєстрації. Це цілком зрозуміло, адже перед тим, як виконати замовлення потрібно мати доступ до посилання.

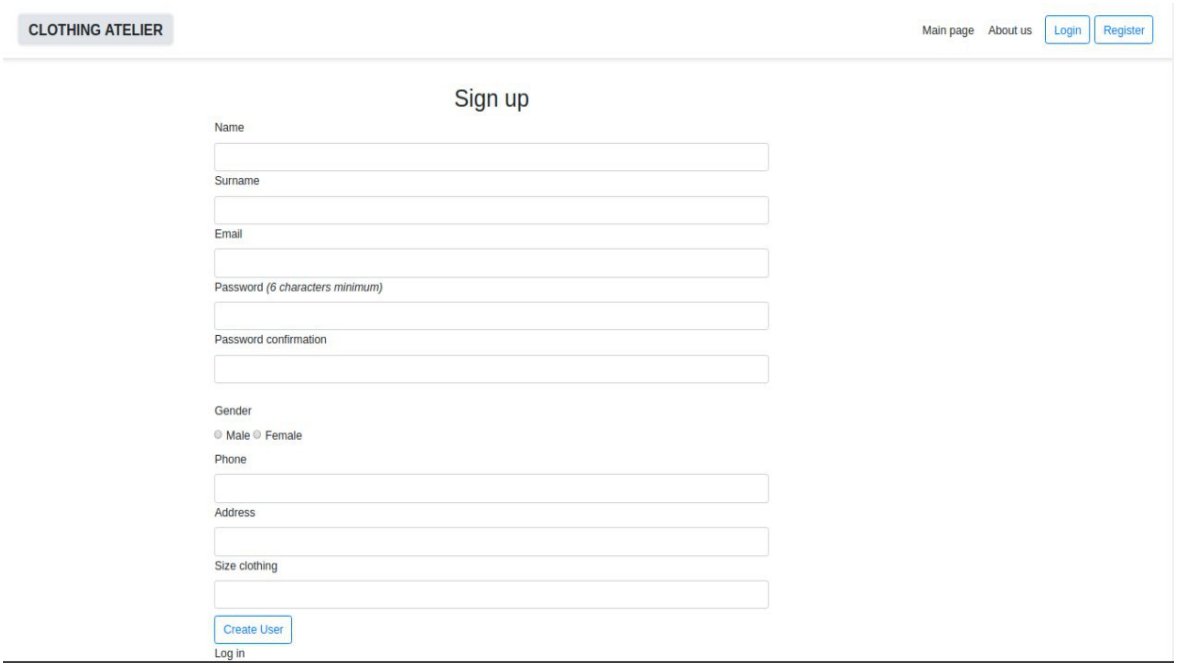

Рисунок 3.9 - Сторінка реєстрації користувача

На рисунку 3.9 продемонстровано сторінку реєстрації користувача. Файл цієї сторінки знаходиться за адресою «views/devise/registrations/new.html.erb». Цей файл було інстальовано автоматично, але він редагувався для змін стилів, та додавались необхідні форми вводу. У всіх сторінках використовується технологія bootstrap [24]. Це безкоштовний набір інструментів для створення веб-сайтів. Він містить готові шаблони CSS і html для форм вводу, кнопок, навігації, тексту та

багато іншого. Bootstrap спрощує створення динамічних веб-застосунків. Саме завдяки цьому, зовнішній вигляд цього проекту виконаний в мінімалістичному стилі, як зображено на рисунку 3.10.

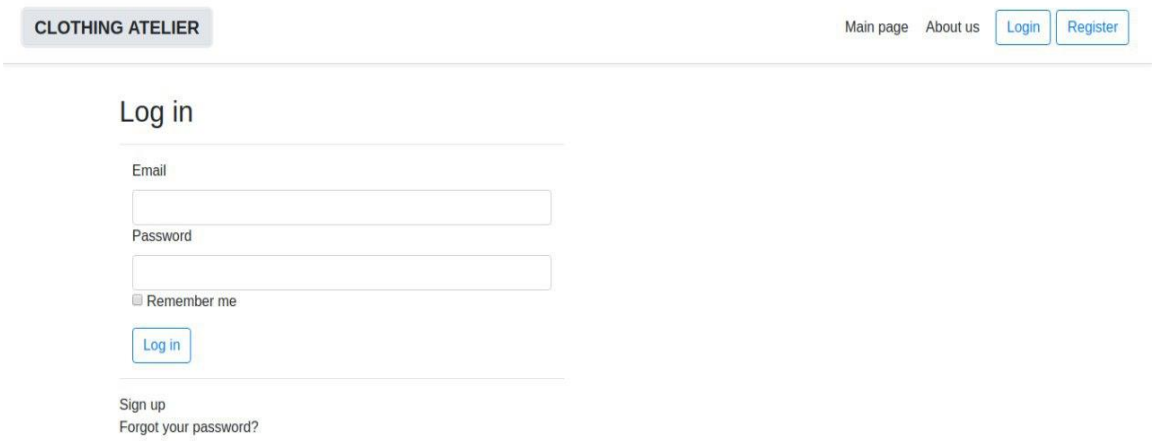

Рисунок 3.10 - Сторінка авторизації

У створенні форм для авторизації, реєстрації та сторінки «profile» використовується стандартна функція *form_for*. На інших сторінках використовувалась інша технологія. Під час розробки веб-сайту, було вирішено не змінювати стандартні функції для більш стабільної роботи застосунку. На сторінках авторизації та реєстрації, дані з форм вводу відправляються методом POST.

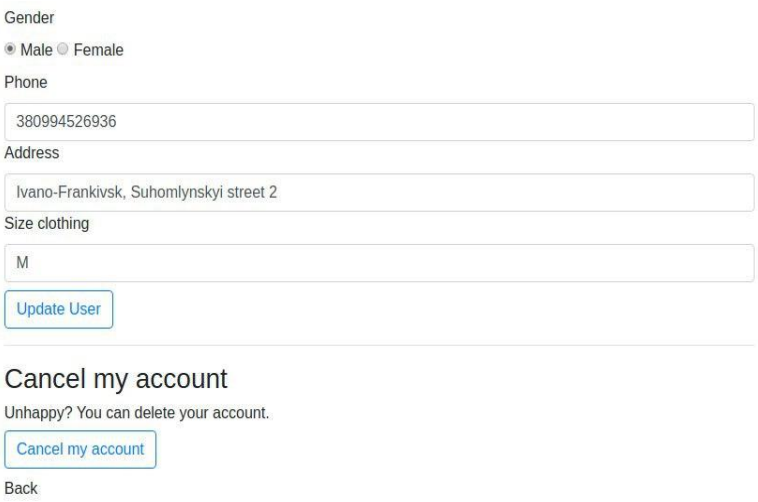

Рисунок 3.11 - Частина сторінки "profile"

Сторінка для редагування особистих даних ззовні майже не відрізняється від сторінки реєстрації. Різниться тільки елементами внизу: кнопка для видалення акаунту та посилання для повернення назад (рис. 3.11). Зі сторони коду, ця сторінка має ті самі форми, тільки дані з них відправляються методом PUT. Завдяки цьому методу, необов'язково описувати всі дані у форму заново. Достатньо лише виправити помилку і підтвердити відредаговані дані. І в такому випадку не потрібно перезаписувати усі дані одного користувача на стороні бази даних.

Рисунок 3.12 – Верхня панель навігації користувача та адміністратора

Верхня панель навігації містить назву сайту, набір посилань на різноманітні сторінки та кнопку для виходу зі свого облікового запису. Назва сайту поміщена в кнопку, тому при натисканні на неї випливає або приховується ліва панель. Контент навігації містить перевірку, чи поточний користувач має права адміністратора. Якщо умова виправдалась, виводяться приховані для звичайного користувача посилання як на рисунку 3.12.

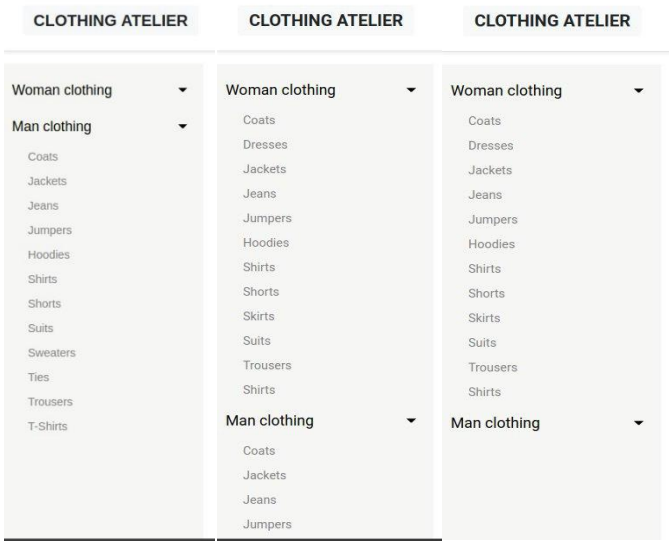

Рисунок 3.13 – Види розгортання лівої панелі

На рисунку 3.13 відображено приклади виконання випадних списків. В лівому меню знаходяться два випадні списки: чоловічий одяг та жіночий одяг. Випадний список містить перелік одягу, який пропонується модельєрами для пошиття в цьому ательє. При натисканні на будь-який елемент з цього списку, користувач переміщається на сторінку створення замовлення (рисунок 3.14).

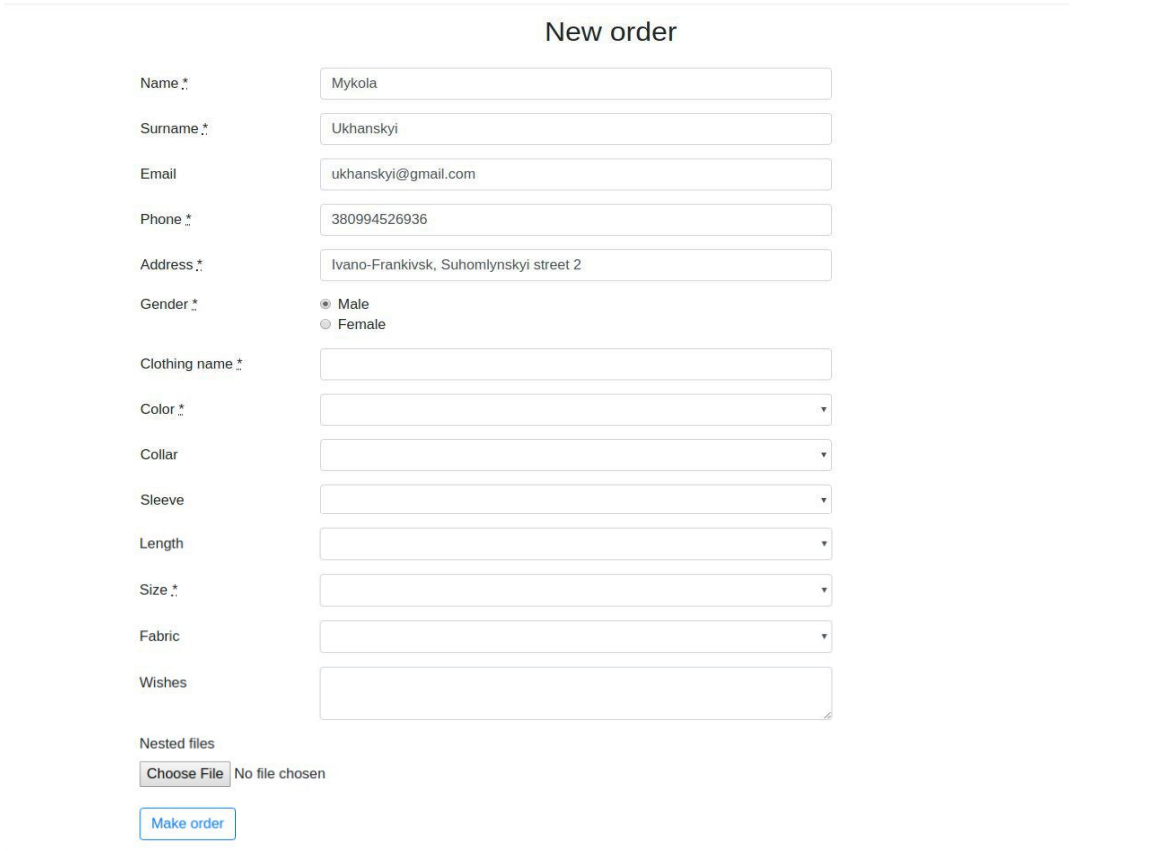

Рисунок 3.14 - Зображення створення замовлення

Випадні списки при натисканні рендеряться без перезавантаження цілої сторінки. А коли натиснути на назву сайту, то ліва панель (меню) цілком зникає або з’являється. Коли користувач тільки розпочинає створювати замовлення, декілька форм вже заповнені. Автоматично заповнені дані поточного користувача беруться з бази даних, які при реєстрації заповнив замовник. У додаткова функція, виконана для того, щоб користувачі не витрачали час на заповнення цих самих форм. А, якщо замовник робить замовлення не для себе, тоді він змінить тільки стать і продовжить заповнювати особливості виробу для тієї персони. Особливості товару містять необов’язкові поля, тому що для одного товару необхідно вводити всі дані, а для іншого – ні. Також створено поле, яке вміщає

великий обсяг тексту. В цьому полі замовник вводить всі побажання для виробу, або ще якусь важливу інформацію. І ще, приємним бонусом є можливість прикріпляти фотографію бажаного виробу. Коли модельєр буде бачити фото, йому легше зрозуміти всі деталі. Веб-сторінка замовлень використовує для форм технологію S*imple form* [25]. Основна мета її – не чіпати свій спосіб визначення макету для форм. І саме ця технологія дуже гарно працює з усіма Bootstrap стилями.

Цікавою особливістю цієї сторінки є можливість підрахувати власний розмір одягу в буквеній формі онлайн. Для цього потрібно виміряти обхват вказаної частини тіла та ввести половину того значення у форму (рис. 3.15).

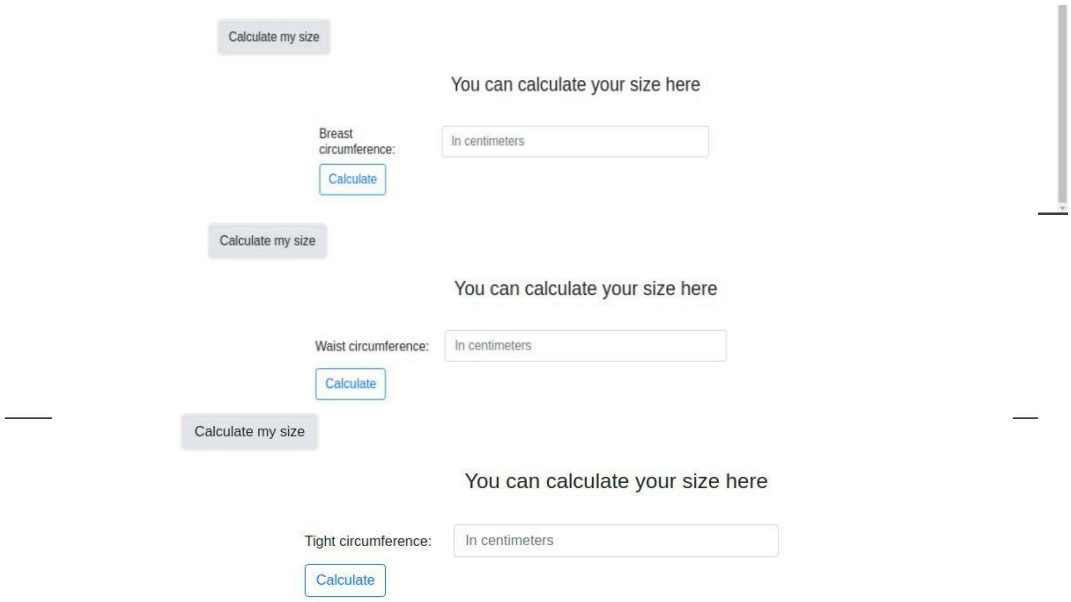

Рисунок 3.15 - Калькулятор визначення розміру одягу

Цей калькулятор за замовчуванням приймає три набори вхідних даних. Вироби поділені на 3 категорії, для обчислення розміру яких потрібно знати обхват грудей, стегон і талії. Тому для цього всі назви одягу відносяться до однієї з трьох категорій. До першої категорії (об’єм грудей) належать: футболки, пальта, куртки, худі та інші. Обхват талії та стегон враховується для штанів, суконь, шортів… У зв’язку з особливостями тіла чоловіків та жінок, прийнято рахувати розмір штанів у чоловіків за пів-обхватом талії (тому, що в чоловіків переважно

більший живіт), а у жінок – за пів-обхватом бедер. Тому, в залежності від статі, випадного списку і елементу одягу виводиться одна з трьох форм. Коли користувач заповнив форму даними та натиснув кнопку «Підрахувати розмір», поле «розмір» автоматично було заповнено результатом калькулятора.

Мінусом даного калькулятора є те, що для роботи з ним потрібно усю Front-end частину проекту переписати на React [26]. В момент написання дипломного проекту, автор ще не був знайомий з технологією React. І калькулятор розробляв паралельно навчаючись [27].

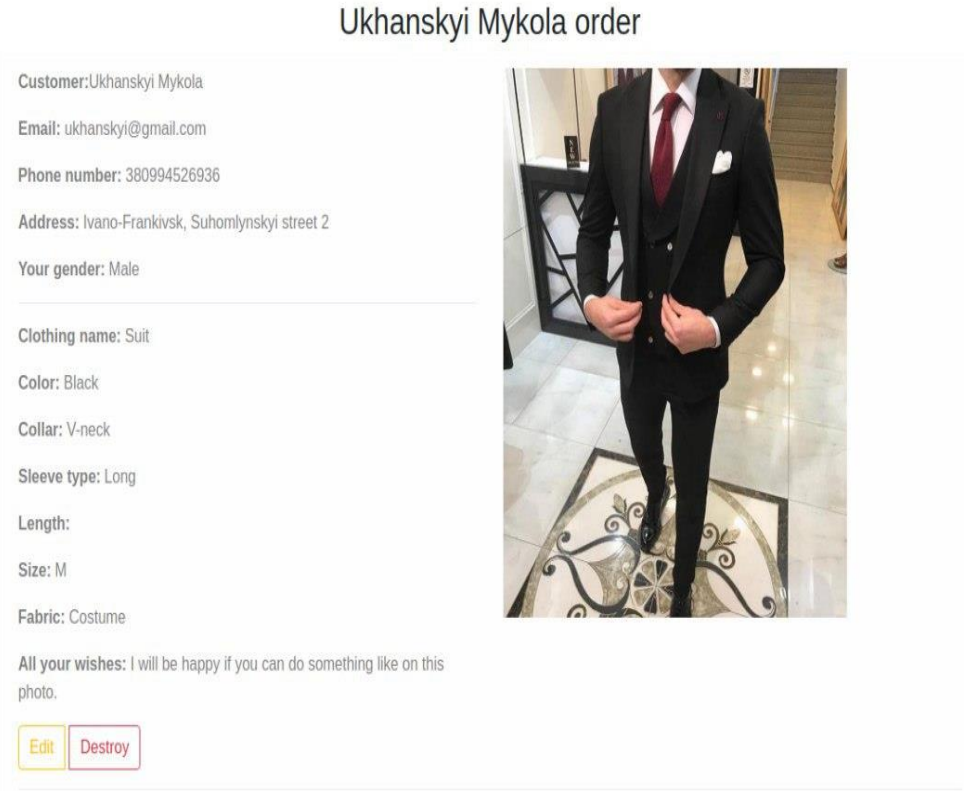

Рисунок 3.16 - Відображення замовлення

Коли користувач заповнив всі форми та створив замовлення він може перевірити коректність усіх введених даних (рис. 3.16). Зображення, яке прикріпив користувач розміщується правіше усіх деталей. Внизу створені дві кнопки: перша відправляє користувача на сторінку редагування, наступна - видаляє замовлення.

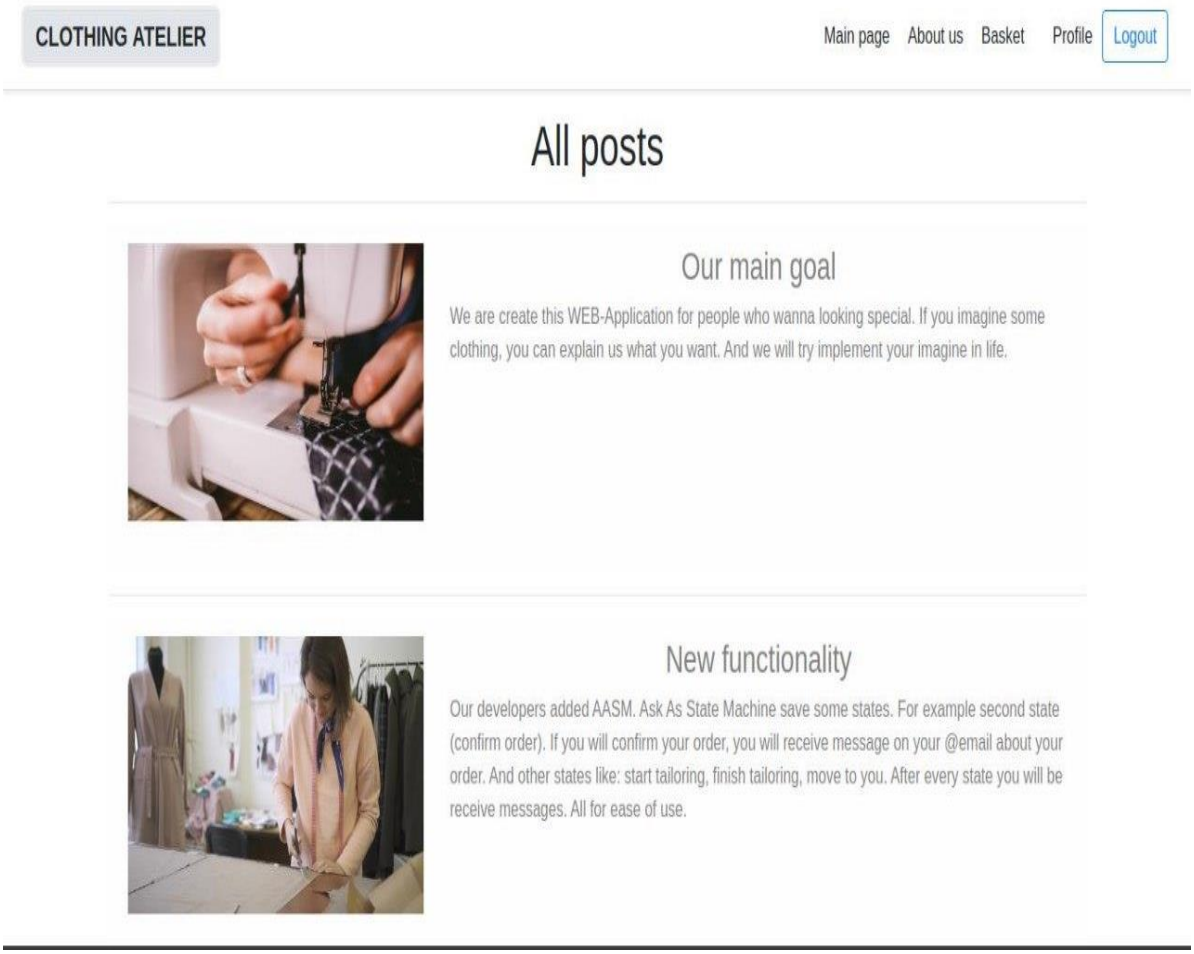

Рисунок 3.17 - Головна сторінка ательє

На рисунку 3.17 відображена основна сторінка веб-сайту. Коли авторизований адміністратор, то під кожним постом з’являється кнопка для перегляду. І вже тоді, після натискання на неї, адміністратор може редагувати або видалити пост. А на зображенні вище, показано основну сторінку на яку зайшов користувач без правів адміністратора, тому ми не бачимо кнопок для редагування та видалення постів.

Сторінка «Про ательє» створена для реклами самого сайту та підприємства. На ній вказана адреса підприємства та важлива інформація. А також продемонстровано на Google Maps картах місцезнаходження підприємства. Трохи нижче створена форма для створення нових відгуків та перелік усіх відгуків про ательє, чи працюючий персонал, або про пошите замовлення (дивіться на рисунку 3.18).

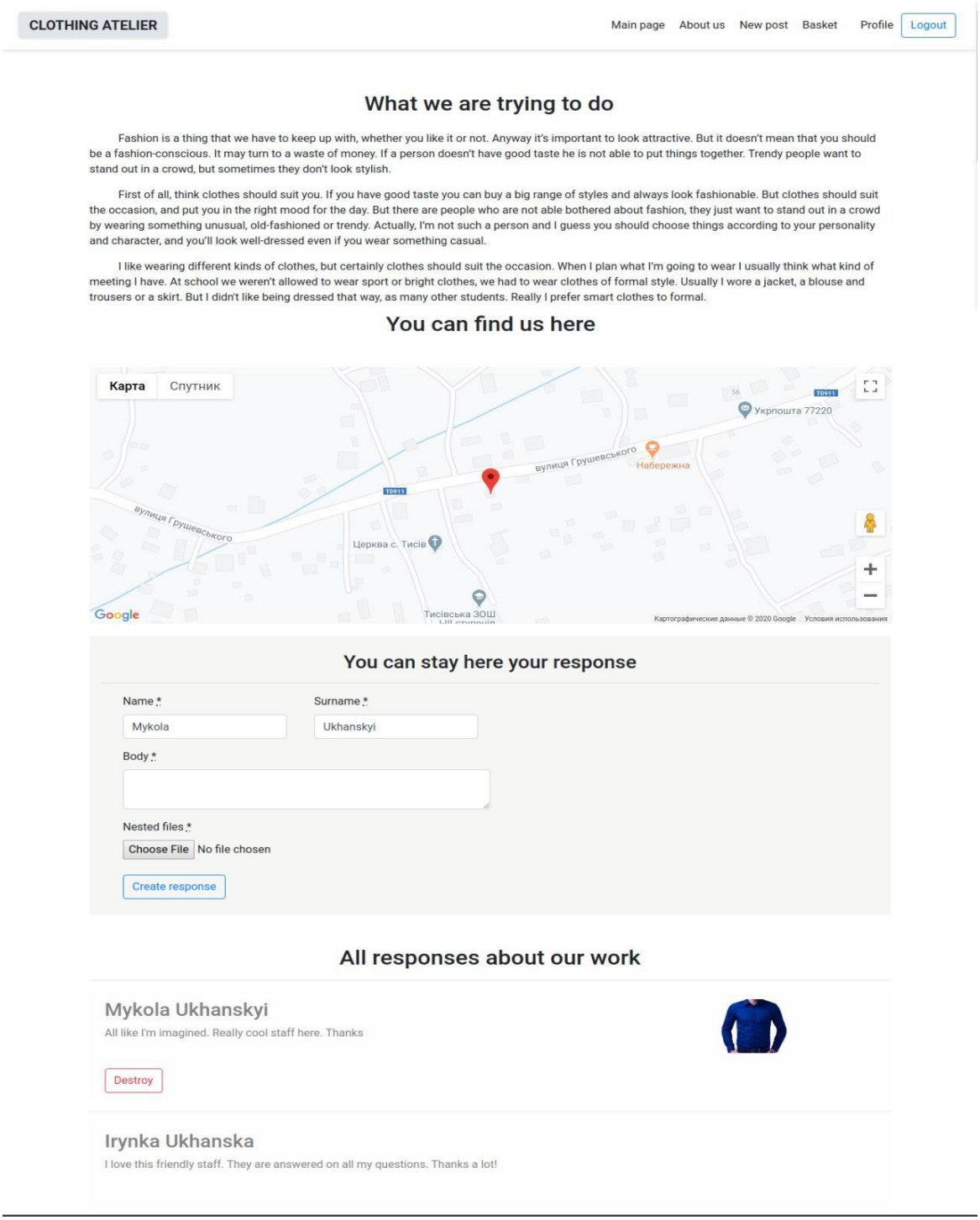

Рисунок 3.18 - Веб-сторінка "Про нас"

Останнім посиланням, про яке ми забули згадати, це є корзина. В ній зберігаються всі замовлення користувача. Там користувач може підтвердити своє замовлення. Потрібно зазначити, що для адміністратора виводяться повністю всі замовлення, а простий користувач має можливість переглядати тільки свої замовлення.

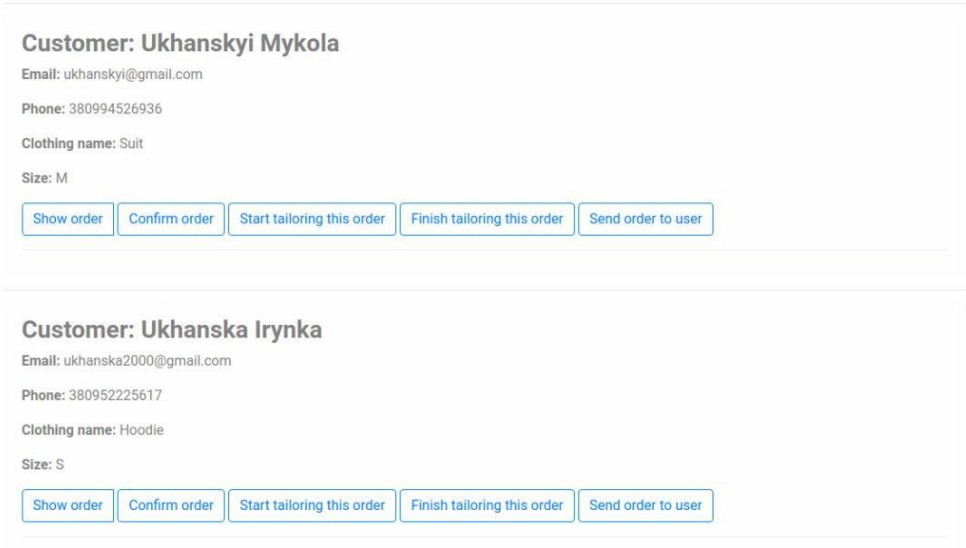

Рисунок 3.19 - Корзина адміністратора

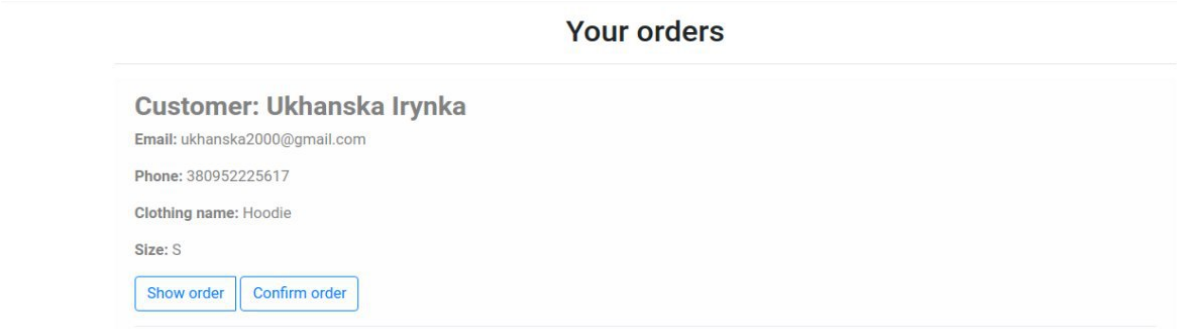

Рисунок 3.20 - Корзина користувача

Як ми бачимо на рисунках 3.19 та 3.20 можливість змінювати стани замовлення виконана дуже просто та зручно за допомогою звичайних кнопок. І тепер ми вже ознайомились з розробкою зовнішнього вигляду для ательє. Для перегляду фрагментів коду аплікації дивіться Додаток А, а при потребі переглянути увесь код – дивіться Додаток Б.

4 БІЗНЕС ПЛАН ВЕБ-САЙТУ АТЕЛЬЄ

4.1 Резюме

Веб-сайт для замовлення пошиття одягу буде приймати замовлення користувачів та допоможе структурувати всі замовлення підприємств ательє. Також веб-застосунок буде інформувати користувачів про їхні замовлення. Замовник матиме можливість переглядати важливі події на сайті або цікаву йому інформацію про підприємство.

Цільовою аудиторією цього веб-застосунку є люди від 16 до 60 років, які мають пристрої з доступом до мережі Інтернет та вміють ними користуватись.

Для створення веб-аплікації для ательє необхідна сума в розмірі від 20 000 до 25 000 грн. для того, щоб орендувати забезпечення для розміщення і розгортання на ньому веб-застосунку.

Кошти буде оплачувати власник (керівник) приватного підприємства ательє.

Для розробки необхідно залучити 2 працівників.

Очікуваний мінімальний місячний дохід ательє становить 7 000 грн.

Чистий прибуток в рік становитиме від 84 000 грн.

4.2 Маркетинг

Види послуг

Веб-сайт допомагає клієнтам, не виходячи з дому, робити замовлення на пошиття чи ремонт одягу в ательє. У зв’язку з епідемією, тепер більшість підприємств стараються працювати онлайн.

Щоб не проводити примірки, в проекті була розроблена можливість підрахувати потрібні розміри для одягу онлайн.

Проект не працює як інтернет магазин, він створений для отримання та оброблення замовлення на пошиття або ремонту.

Веб-ательє може використовуватись для мережі багатьох ательє підприємств.

Його унікальність полягає у тому, що у веб-аплікації є можливість швидко обчислити розмір замовника під час створення самого замовлення. Більшість сайтів ательє українських підприємств не мають такої можливості. Переважно це прості сайти-візитки, що містять тільки коротку інформацію про місцезнаходження підприємства і про вид їхньої діяльності. Також існують веб-сайти, що працюють як інтернет магазини, але на такому сайті можна придбати вже готовий пошитий одяг, а не замовити індивідуальне пошиття.

Для використання веб-застосунку для замовлення пошиття одягу необхідно тільки перейти за посиланням і створити свій обліковий запис.

До недоліків цього проекту можна віднести обов'язковість підключення до мережі Інтернет та відсутність 3D-примірки онлайн. Адже в закордонному аналогу існує така розробка, проте вона дуже дорога і потребує багато часу для її розробки.

Дослідження ринку

Для пошуку інформації про ательє використовувались інформаційні джерела мережі Інтернет (статистика про кількість запитів, факти про ательє та інше), а також компетентна особа в швейній сфері.

Оцінка попиту

Дослідження показали, що дана розробка є актуальною для багатьох підприємств у різних областях країни. В пошуковій мережі Google виконують близько 120 000 запитів про ательє щомісяця, а пошиття одягу – 50 000 запитів.

Найбільша кількість запитів ательє в Рівненській, Львівській, Київській, Івано-Франківській, Хмельницькій та Тернопільській областях.

Якщо майбутні клієнти не мають змоги прийти по замовлення, їм можуть надіслати його новою поштою, тому користувачі не прив’язані до місцезнаходження самого ательє.

Конкуренти:

– за даним напрямком в Україні працює велика кількість ательє, більшість з яких знаходиться у великих містах: Київ, Львів…

– конкуренти використовують радіо, телекомунікації, інтернет-ресурси для просування своїх послуг;

– бізнес в конкурентів зростає стрімкими темпами, в них вже є велика кількість постійних клієнтів;

– в конкурентів занадто дорогі ціни для ремонту, а особливо для пошиття нового одягу;

– перевагами цього проекту над конкурентами являється більший функціонал та простота використання.

Оптимістичний сценарій розвитку:

– через цікавість багато людей забажають скористатись послугами нового веб-ательє;

– помірний ціновий діапазон допоможе підтримувати кількість замовників;

– задоволені клієнти будуть рекомендувати послуги ательє своїм друзям та знайомим;

– розширення підприємства та створення нових філій і робочих місць.

Песимістичний сценарій розвитку:

– можливо, протягом великого часу не буде припливу клієнту;

– збільшення цін на вироби;

– збільшення вартості послуг для хостингу чи оренди програмного забезпечення;

– реклама виявиться недостатньо ефективною;

– недостатньо хороша якість пошиття замовлення;

– недостатньо зручне розташування приміщення ательє.

Реалістичний сценарій розвитку:

– з плином невеликого проміжку часу назбирається достатня кількість постійних клієнтів та зникне недовіра до «нового» ательє;

– працівники відповідально шитимуть одяг і завдяки цьому не страждатиме репутація ательє;

– підтримання веб-сайту на перших сходинках пошукових систем охоплюватиме більше нової аудиторії;

– реклама в соціальних мережах залучатиме нову клієнтуру;

– ціни відповідатимуть якості виконання замовлення.

Маркетингова підтримка

На початку на рекламу буде виділено 3 000 гривень. Підтримка декількох видів реклами буде продовжуватися надалі.

Покращення роботи сайту та додавання додаткових корисних функцій до нього.

Якщо керівництво ательє сприятиме додатковому розширенню веб-застосунку, то завдяки розробці 3D-примірки буде перевага над усіма конкурентами.

Встановлення рівня цін

Ціна виробів встановлюватиметься на обговоренні з керівництвом ательє. Користування веб-застосунком буде безкоштовним. Отримання прибутку з сайту можливе з допомогою інтеграції невеликої реклами.

План збуту

Застосунок просуватиметься власними силами, без допомоги третіх сторін.

Замовлення користувачі матимуть змогу забирати самі. Також можлива відправка Новою поштою для замовників з інших міст, районів, областей.

План комплексу просування веб-сайту

Веб-сайт просуватиметься за допомогою реклами в соцмережах (Instagram, Facebook, YouTube).

В рекламних оголошеннях буде робитись наголос на зручності, відповідальності робочого персоналу, швидкості та якості виконання замовлень.

Також відмічатиметься розумний діапазон цін.

З самого початку реклама буде фінансуватись великими вкладами, а пізніше, коли буде велика кількість постійних замовників, виділятиметься невеликий відсоток від виручки.

4.3 Обґрунтування необхідних фінансових вкладень

Опис виробничих потужностей

Веб-застосунок буде розташований на одному із найпопулярніших платформ для хостингу (Amazon Web Services, Google Cloud, Azure). І швидше за все буде використовуватись найдешевший регіон для знаходження центру даних (серверу).

Режим роботи веб-сайту – цілодобовий (за виключенням технічних проблем на сервері).

Потрібно також передбачити співпрацю з хостингами, що пропонують невелику кількість можливостей за меншу ціну. Якщо вони погодяться зробити знижку, то зможемо розмістити їхню рекламу на веб-сайті для замовлення пошиття одягу.

Витрати на послуги

Таблиця 4.1 – Вартість на послуги утримання веб-сайту

Оренда сховищ даних:	≈ 700 грн
Оренда IP-адресу	≈ 100 – 150 грн
Оренда хостингу	≈ 1 500 – 2 000 грн
Сума:	**≈ 2 300 – 2 850 грн**

Підбір персоналу та оплата праці

Після розміщення веб-сайту на хостинг, не потрібно залучати програмістів. Коли набереться певна кількість користувачів, потрібно все ж таки найняти розробника для підтримки веб-застосунку.

Для економії коштів можна залучити студента, компетентного в підтримці сайтів. Пізніше, коли ательє приноситиме більший прибуток, можна залучити більшу кількість розробників для покращення сайту:

– прийняття на роботу програміста відбуватиметься після співбесіди;

– для працівників встановлено 8-годинний робочий день;

– для підтримки сайту можна найняти людину з гнучким робочим графіком;

– оплата праці буде залежати від виробітку;

– якщо робота в ательє буде швидко просуватись, стануть можливі премії кращим працівникам.

4.4 Нормативно-правові нюанси

Організаційно-правова форма бізнес-проекту

Проект буде належати приватному підприємству, тому що в Україні в даний момент не існує державних ательє.

Організаційний план

Ведення фінансів буде виконуватись директором ательє. Можливо, буде необхідна підтримка юриста, банку і страхової компанії.

4.5 Складання фінансового бюджету

Джерела фінансування

Джерелом фінансування являються кошти керівництва ательє. Або ж знайдеться спонсор, який в майбутньому, за умови успішної роботи ательє, зможе отримувати відсоток від прибутку.

Кошторис витрат

Для створення веб-аплікації для ательє необхідно: 2-3 розробника, біля 3 місяців розробки та сума в розмірі від 55 000 до 65 000 грн.

Витрати на підтримку сайту становлять від 2 300.

Очікуваний мінімальний місячний дохід ательє становить 10 000 грн.

Чистий прибуток в рік становитиме від 120 000 грн.

Термін окупності: 7 місяців.

4.6 Оцінка можливих ризиків

Перелік можливих ризиків для ательє:

- стихійне лихо;
- різке падіння попиту;
- стрибки рівня інфляції;
- нестабільна робота сайту;
- збільшення ціни на оренду хостингу та апаратного забезпечення.

ВИСНОВКИ

У цьому дипломному проекті було досліджено та розроблено веб-сайт для замовлення пошиття одягу. За допомогою цього застосунку було вирішено проблеми тих, хто хочуть гарно та особливо одіватись, тих хто має нестандартну фігуру (особливості в будові тіла), а також підприємств, що можуть запропонувати послуги художнього моделювання й шиття. Завдяки таким сайтам люди не витрачатимуть свій час на пошуки ательє у своєму населеному пункті (чи в сусідніх). Їм взагалі не потрібно буде кудись їхати. Вони з легкістю зможуть замовити «саме такий» наряд, який їм до вподоби, зайшовши на сайт, та заповнивши невелику анкету. Ці дані допоможуть в створенні одягу, який ідеально підходитиме даній фігурі.

Було розроблено бізнес план проекту, де обчислені витрати на розробку веб-сервісу та вартість його утримання. Розраховано приблизний час за який проект має окупитись, та за цими обчисленнями відомо, що даний веб-сервіс є конкурентоспроможним на ІТ-ринку.

Під час розробки веб-сайту були освоєні методики для створення сайтів використовуючи мову програмування Ruby і веб-фреймворк Ruby on Rails. Також ознайомлення з деякими бібліотеками такими як: devise, aasm, carrierwave, та інші.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

REFERENCES

1. Інформація про ательє. URL: https://uk.wikipedia.org/wiki/Ательє (дата звернення: 15.11.2019);
2. Ательє в мистецтві. URL: https://en.wikipedia.org/wiki/Atelier (дата звернення: 05.12.2019);
3. Bestpoke пошив. URL: https://www.yuriyurik.ru/blog/about_bespoke (дата звернення: 05.12.2019);
4. Frilly. URL: https://www.frilly.com/ (дата звернення: 05.12.2019);
5. П. Федорук і М. Дутчак, “Побудова бази знань адаптивних систем дистанційного навчання на основі фреймової та продукційної моделей представлення знань,” Управляючі системи і машини (УСіМ), №5, с.35-42, 2012;
6. ER-діаграма. URL: https://uk.wikipedia.org/wiki/Модель_«сутність_—_зв%27язок» (дата звернення: 15.12.2019);
7. Структура rails веб-сайту: http://rusrails.ru/getting-started-with-rails (дата звернення: 25.12.2019);
8. Мова програмування Ruby. URL: https://medium.com/evrone-ru/почему-ruby-ruby-on-rails-5d08e2ce8d49, http://shpora.me/test/Ruby-567 (дата звернення: 17.01.2020);
9. JRuby. URL: https://uk.wikipedia.org/wiki/JRuby (дата звернення: 25.01.2019);
10. Rubinius. URL: https://ru.wikipedia.org/wiki/Rubinius (дата звернення: 25.01.2019);
11. IronRuby. URL: https://uk.wikipedia.org/wiki/IronRuby (дата звернення: 25.01.2019);
12. MRuby. URL: https://en.wikipedia.org/wiki/Mruby (дата звернення: 25.01.2019);
13. MagLev. URL: https://en.wikipedia.org/wiki/MagLev_(software) (дата звернення: 25.01.2019);

14. Фреймворк Ruby on Rails. URL: http://rusrails.ru/getting-started-with-rails (дата звернення: 22.02.2020);
15. David A., BlackJoseph Leo III, The Well-Grounded Rubyist Third Edition. USA : Manning Publications Co., 2019. 584 с.;
16. Sam Ruby, David Bryant Copeland, Dave Thomas Agile Web Development with Rails 6. Raleigh : the pragmatic bookshelf, 2020. 494с.;
17. Архітектура MVC. URL: https://habr.com/ru/post/181772/ https://uk.m.wikipedia.org/wiki/Модель-вид-контролер (дата звернення: 17.03.2020);
18. Сховища даних PostgreSQL. URL: https://habr.com/ru/post/282764/ (дата звернення: 27.03.2020);
19. Джуба С., Волков А., Изучаем PostgreSQL 10. Москва : ДМК Пресс, 2019. 402 с.;
20. Gem Devise. URL: https://habr.com/ru/post/208056/ (дата звернення: 03.04.2020);
21. Hafiz Barie Lubis, Nia Mutiara, Giovanni Sakti Learning Devise for Rails. Birmingham : Packt Publishing, 2013. 104с.;
22. Gem AASM. URL: https://github.com/aasm/aasm (дата звернення: 13.04.2020);
23. Фрэйн Б. HTML5 и CSS3 Разработка сайтов для любых браузеров и устройств 2-е изд. Санкт-Петербург : Питер, 2017. 272с.;
24. Сильвио Морето Bootstrap в примерах (Пер. с англ. Рагимов Р. Н.). Москва : ДМК Пресс, 2017. 314 с.;
25. Simple Form. URL: https://github.com/heartcombo/simple_form/wiki (дата звернення: 23.04.2020);
26. Robin Wieruch The Road to learn React. USA : CreateSpace, 2018. 198 с.;
27. Алекс Бэнкс, Ева Порселло, React и Redux Функциональная веб-разработка. Санкт-Петербург : Питер, 2018. 336 с.;
28. СІТ 2:2018. Стандарт кафедри інформаційних технологій. Дипломний проект. Вимоги до змісту та оформлення [Чинний від 2018-09-03]. Вид. офіц. Івано-Франківськ, 2018. 43 с. (дата звернення: 20.05.2020).

ДОДАТОК А

Лістинг програми (модель order.rb)

```
# frozen_string_literal: true

class Order < ApplicationRecord
  belongs_to :user
  has_one :detail, dependent: :destroy

  validates :name, presence: true, length: { within: 3..25 }
  validates :surname, presence: true, length: { within: 3..50 }
  validates :email, format: { with: URI::MailTo::EMAIL_REGEXP }
  validates :phone, presence: { message: 'Write your number in
                                          correct form!' },
                    numericality: true,
                    length: { within: 10...15 }
  validates :gender, presence: true
  validates :address, presence: true, length: { maximum: 100 }

  accepts_nested_attributes_for :detail

  include AASM

  aasm column: 'state' do
    state :initialized, initial: true
    state :confirmed
    state :tailoringStarted
    state :tailoringFinished
    state :moveToYou

    event :confirm, after: :order_email do
      transitions from: :initialized, to: :confirmed
    end

    event :tailoring_start, after: :order_email do
      transitions from: :confirmed, to: :tailoringStarted
    end

    event :tailoring_finish, after: :order_email do
      transitions from: :tailoringStarted, to: :tailoringFinished
    end

    event :move_to_you, after: :order_email do
      transitions from: :tailoringFinished, to: :moveToYou
    end
  end
```

```
  private

  def order_email
    UserMailer.order_email(self).deliver_now
  end
end
```

Лістинг програми (контролер orders_controller.rb)

```
# frozen_string_literal: true

class OrdersController < ApplicationController

  before_action :find_order, only: %i[show edit update destroy]

  def index
    @orders = Order.all

    if current_user.admin == true
      respond_to do |format|
        format.html { @orders }
        format.json { render json: @orders, include: [:detail] }
      end
    else
      respond_to do |format|
        format.html { current_user.orders }
        format.json { render json: current_user.orders,
                      include: [:detail] }
      end
    end
  end

  def new
    @order = Order.new
  end

  def confirm
    @order = Order.find(params[:id])
    @order.confirm!
    render 'show'
  end

  def tailoring_start
    @order = Order.find(params[:id])
    @order.tailoring_start!
    render 'show'
  end
```

```
def tailoring_finish
  @order = Order.find(params[:id])
  @order.tailoring_finish!
  render 'show'
end

def move_to_you
  @order = Order.find(params[:id])
  @order.move_to_you!
  render 'show'
end

def show
  respond_to do |format|
    format.html { @order }
    format.json { render json: @order, include: 'detail' }
  end
end

def edit; end

def update
  respond_to do |format|
    if @order.update(order_params)
      format.html { redirect_to @order,
                    notice: 'Order was successfully updated.' }
      format.json { render :show, status: :ok,
                    location: @order}
    else
      format.html { render :edit }
    end
  end
end

def destroy
  @order.destroy
  respond_to do |format|
    format.html { redirect_to orders_path,
                  notice: 'Order was successfully destroyed.' }
    format.json { head :no_content }
  end
end

def create
  # render plain: params[:post].inspect

  @order = Order.new(order_params)
  @order.user_id = current_user.id
```

```
    respond_to do |format|
      if @order.save
        format.html { redirect_to @order,
                      notice: 'Order was successfully created.' }
        format.json { render :show, status: :created,
                      location: @order }
      else
        format.html { render :new }
      end
    end
  end

  private

  def order_params
    params.require(:order).permit(:name, :surname, :email, :phone,
                                  :address, :gender, :state,
                                  detail_attributes: %i[order_id
                                                        clothing_name
                                                                color
                                                               collar
                                                               sleeve
                                                               length
                                                                 size
                                                               fabric
                                                               wishes
                                                      nested_files])
  end

  def find_order
    @order = Order.find(params[:id])
  end
end
```

Лістинг програми (router.rb)

```
# frozen_string_literal: true

Rails.application.routes.draw do
  devise_for :users
  # For details on the DSL available within this file, see
  # https://guides.rubyonrails.org/routing.html
  root 'posts#index', as: 'home'
  get 'about' => 'pages#about', as: 'about' do

    resources :responses, only: %i[create destroy]

  end
```

```
  get 'users/sign_up' => 'users#sign_up', as: 'sign_up'

  resources :responses
  resources :posts
  resources :orders do
    member do
      post :confirm
    end
  end
end
```

Лістинг програми (відображення «show.html.erb»)

```
<div class="alert alert-light">

  <h3><b><%= @order.surname %> <%= @order.name %></b></h3>
  <p></p>
  <p><b>Email:</b> <%= @order.email %></p>
  <p><b>Phone number:</b> <%= @order.phone %></p>
  <p><b>Address:</b> <%= @order.address %></p>
  <p><b>Your gender:</b> <%= @order.gender.capitalize %></p>
  <hr>
  <p><b>Clothing name:</b> <%= @order.detail.clothing_name %></p>
  <p><b>Color:</b> <%= @order.detail.color %></p>
  <p><b>Collar:</b> <%= @order.detail.collar %></p>
  <p><b>Sleeve type:</b> <%= @order.detail.sleeve %></p>
  <p><b><%= @order.detail.clothing_name %>:</b>
        <%= @order.detail.length %>
  </p>
  <p><b>Size:</b> <%= @order.detail.size %></p>
  <p><b>Price:</b> <%= @order.detail.price %> UAH</p>

  <div class="btn-group">
    <%= link_to "Edit", edit_order_path(@order),
                        class: 'btn btn-outline-warning' %>
    <%= link_to "Destroy", order_path(@order), method: :delete,
                           data: { confirm: "Are you sure?" },
                           class: 'btn btn-outline-danger ml-1' %>
  </div>

  <hr>
  <%= link_to "Back", :back %>

</div>
```

Лістинг програми (відображення «new.html.erb»)

```
<div class="content w-75 p-3">
  <% @order.build_detail %>

  <%= simple_form_for @order,
                      wrapper: :horizontal_form,
                      wrapper_mappings: {
                          boolean:       :horizontal_boolean,
                          check_boxes:   :horizontal_collection,
                          radio_buttons: :horizontal_collection,
                      } do |f| %>

    <%= f.error_notification %>

    <div class="container">
      <%= f.input :name, input_html: { value: current_user.name },
                         class: 'form-control'%>
    </div>

    <div class="container">
      <%= f.input :surname,
          input_html: { value: current_user.surname },
          class: 'form-control'%>
    </div>

    <div class="container">
      <%= f.input :email,
          input_html: { value: current_user.email },
          class: 'form-control'%>
    </div>

    <div class="container">
      <%= f.input :phone,
          input_html: { value: current_user.phone },
          prompt: "Enter phone number", class: 'form-control' %>
    </div>

    <div class="container">
      <%= f.input :address,
          input_html: { value: current_user.address },
          class: 'form-control'%>
    </div>

    <div class="container">
      <%= f.input :gender, as: :radio_buttons,
          collection: [%w(male Male), %w(female Female)],
          checked: current_user.gender, label_method: :second,
          value_method: :first, class: 'form-control' %>
    </div>
```

```
    <!-- from another table using nested attribute -->

    <%= f.simple_fields_for :detail do |detail| %>
      <div class="container">
        <%= detail.input :clothing_name, class: 'form-control' %>
      </div>

      <div class="container">
        <%= detail.input :color,
            collection: %w(Black White Red Purple Blue Green Pink
                           Orange Yellow Brown Grey),
            class: 'form-control' %>
      </div>

      <div class="container">
        <%= detail.input :collar,
            collection: %w(Basic Polo V-neck),
            class: 'dropdown-item outline-secondary' %>
      </div>

      <div class="container">
        <%= detail.input :sleeve,
            collection: %w(Short Middle Long) %>
      </div>

      <div class="container">
        <%= detail.input :length,
            collection: %w(Mini Above_knee Knee_length Cocktail
                           Midi Maxi Floor_length) %>
      </div>

      <div class="container">
        <%= detail.input :size, collection: %w(S M L XL XXL) %>
      </div>

      <div class="container">
        <%= detail.input :price %>
      </div>

    <% end %>

    <!-- end form with nested attribute -->

    <div class="container">
      <%= f.submit("Make order",
                   {class: 'btn btn-outline-primary mt-2'})%>
    </div>
  <% end %>
</div>
```

Лістинг програми (відображення «index.html.erb»)

```
<!--
  <%= react_component 'Orders', current_user: current_user %>
-->

<h2> Your orders </h2>

<% current_user.orders.each do |order| %>
  <hr>

  <div class="alert alert-light">
    <h3><b><%= order.surname %> <%= order.name %></b></h3>
    <p><b>Clothing name:</b> <%= order.detail.clothing_name%></
    <p><b>Size: </b><%= order.detail.size%></p>

    <div class="btn-group">
      <%= link_to "Show order", order_path(order),
          class: 'btn btn-outline-primary'  %>
      <%= button_to "Confirm order", confirm_order_path(order),
          class: 'btn btn-outline-primary ml-1' %>
    </div>

    <hr>
  </div>

<% end %>
```

Лістинг програми (відображення «edit.html.erb»)

```
<%= react_component("Edit", current_user: current_user) %>

<div class="content w-75 p-3">
  <% @order.build_detail %>

  <%= simple_form_for @order,
                      wrapper: :horizontal_form,
                      wrapper_mappings: {
                          boolean:       :horizontal_boolean,
                          check_boxes:   :horizontal_collection,
                          radio_buttons: :horizontal_collection,
                      } do |f| %>

    <%= f.error_notification %>
```

```
<div class="container">
  <%= f.input :name, class: 'form-control'%>
</div>
<div class="container">
  <%= f.input :surname, class: 'form-control'%>
</div>

<div class="container">
  <%= f.input :email,
      input_html: { autocomplete: 'email' },
      class: 'form-control'%>
</div>

<div class="container">
  <%= f.input :phone,
      prompt: "Enter phone number",
      class: 'form-control' %>
</div>

<div class="container">
  <%= f.input :address, class: 'form-control'%>
</div>

<div class="container">
  <%= f.input :gender, as: :radio_buttons,
      collection: [%w(true male), %w(false female)],
      label_method: :second, value_method: :first,
      class: 'form-control' %>
</div>

<!-- from another table using nested attribute -->

<%= f.simple_fields_for :detail do |detail| %>
  <div class="container">
    <%= detail.input :clothing_name, class: 'form-control' %>
  </div>

  <div class="container">
    <%= detail.input :color,
        collection: %w(Red Black White Yellow Purple Blue
                       Orange Green Brown Grey),
        class: 'form-control' %>
  </div>

  <div class="container">
    <%= detail.input :collar,
        collection: %w(Basic Polo V-neck),
        class: 'dropdown-item outline-secondary' %>
  </div>
```

```
      <div class="container">
        <%= detail.input :sleeve,
            collection: %w(Short Middle Long) %>
      </div>
      <div class="container">
        <%= detail.input :length,
            collection: %w(Mini Above_knee Knee_length Cocktail
                           Midi Maxi Floor_length) %>
      </div>

      <div class="container">
        <%= detail.input :size, collection: %w(S M L XL XXL) %>
      </div>

      <div class="container">
        <%= detail.input :price %>
      </div>

    <% end %>

    <div class="container">
      <%= f.submit("Make order",
          {class: 'btn btn-outline-primary mt-2'})%>
    </div>

  <% end %>
</div>
```

Лістинг програми (Калькулятор «javascript/types.js»)

```
/**
 * SIZES
 */
export const XS = 'XS';
export const S = 'S';
export const M = 'M';
export const L = 'L';
export const XL = 'XL';
export const XXL = 'XXL';

/**
 * GENDERS
 */
export const WOMAN = 'WOMAN';
export const MAN = 'MAN';
```

```
/**
 * CLOTHES
 */
export const COAT = 'Coat';
export const DRESS = 'Dress';
export const JACKET = 'Jacket';
export const JEANS = 'Jeans';
export const JUMPER = 'Jumper';
export const HOODIE = 'Hoodie';
export const SHIRT = 'Shirt';
export const SHORTS = 'Shorts';
export const SKIRT = 'Skirt';
export const SUIT = 'Suit';
export const SWEATER = 'Sweater';
export const TROUSERS = 'Trousers';
export const TSHIRT = 'T-Shirt';
```

Лістинг програми (Калькулятор «javascript/constans.js»)

```
import { XS, S, M, L, XL, XXL } from './types';

export const standardSizes = {
  manClothes: [41, 44, 47, 50, 53, 56],
  womanClothes: [42, 44, 46, 48, 50, 52],
  letters: [XS, S, M, L, XL, XXL],
};
```

Лістинг програми (Калькулятор «javascript/helpers/calculateClothesSize.js»)

```
import { MAN, WOMAN } from '../types';
import { standardSizes } from '../constants';
export default function (gender, actualValue) {
  if (gender !== MAN && gender !== WOMAN) return null;
  if (Number.isNaN(actualValue) || actualValue < 0) return null;
  if (gender === MAN) {
    for (let i = 0; i < standardSizes.manClothes.length; i++) {
      if (standardSizes.manClothes[i] - actualValue >= 0) return
      standardSizes.letters[i];
    }
  } else {
    for (let i = 0; i < standardSizes.womanClothes.length; i++) {
      for (let i = 0; i < standardSizes.womanClothes.length; i++)
      {
```

```
        if (standardSizes.womanClothes[i] - actualValue >= 0)
        return standardSizes.letters[i];
      }
    }
  }

  return null;
}
```

Лістинг програми (Калькулятор «javascript/ClothingSizeCalculatorWaist/index.js»)

```
import React, { Component } from 'react';
import Collapsible from '../../Collapsible';
import calculateClothesSize from
       '../../../helpers/calculateClothesSize';
import './styles.css';

class ClothingSizeCalculatorWaist extends Component {
  constructor(props) {
    super(props);
    this.state = {
      waistValue: '',
    };
  }

  handleWaistValueChanged = (e) => {
    this.setState({ waistValue: e.target.value });
  }

  handleCalculateSubmitted = () => {
    const { gender, onClothingSizeChanged } = this.props;
    const { waistValue } = this.state;
    const clothinSize = calculateClothesSize(gender, waistValue);
    if (clothinSize) onClothingSizeChanged({
       target: { value: clothinSize }
    });
  }

  render() {
    const { waistValue } = this.state;

    return (
      <div>
        <br />
        <Collapsible button={
          <div>
            <button className="menu btn btn-light">
```

Продовження Додатку А

```
                Calculate my size
              </button>
            </div>
          }>
            <div className="container col-sm-9">
              <br />
              <h4 className="row justify-content-center">
                  You can calculate your size here
              </h4>
              <br/>
              <div>
                <div className=" row group mt-1">
                  <h6 className="my-auto col-sm-3">
                      Waist circumference:
                  </h6>

                  <input
                    type="number"
                    name="waist"
                    placeholder='In centimeters'
                    className="form-control col-md-6"
                    value={waistValue}
                    onChange={this.handleWaistValueChanged}
                  />
                </div>

                <div>
                  <button className="btn btn-outline-primary mt-2"
                          onClick={this.handleCalculateSubmitted}
                  >
                    Calculate
                  </button>
                </div>

              </div>

              <br/>
            </div>

          </Collapsible>

        </div>
      );
    }
}

export default ClothingSizeCalculatorWaist;
```

ДОДАТОК Б

Код сайту доступний для перегляду за посиланням на GitHub: https://github.com/Ukhanskyi/clothing-atelier