МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Веб-додаток: адміністрування ресторанів на основі javascript»

Завідувач

випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Шутилєва О.В.**

Студент групи ІН-62 **Сухоруков Я.Р.**

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Кафедра комп'ютерних наук

Затверджую _____
Зав. кафедрою Довбиш А.С.
“_____” _____2020 р.

Завдання
до випускної роботи

Студента четвертого курсу, ІН – 62 спеціальності “Інформатика” денної форми навчання Сухорукова Ярослава Романовича.

Тема: “**Веб-додаток: адміністрування ресторанів на основі javascript**”

Затверджена наказом по СумДУ
№_____ від _____2020 р.

Зміст пояснювальної записки: 1) аналіз проблеми та існуючих рішень; 2) постановка задачі; 3) пошук оптимальних методів вирішення проблеми; 4) проектування системи; 5) програмна реалізація.

Дата видачі завдання “_____” _____2020 р.
Керівник випускної роботи _____ Шутилєва О.В.
Завдання прийняв до виконання _____ Сухоруков Я.Р.

РЕФЕРАТ

Записка: 83 стор., 21 рис., 1 таблиця, 2 додатки, 21 джерело.

Об’єкт дослідження – процес проектування та розробки веб-додатку на основі javascript.

Мета роботи – реалізація веб-додатку для адміністрування закладів громадського харчування.

Результати – розроблено веб-додаток для адміністрування малих та середніх ресторанів. В роботі були виявлені недоліки вже існуючих рішень, та за допомогою сучасних технологій веб-розробки вони були вирішені.

ОДНОСТОРІНКОВИЙ ВЕБ-ДОДАТОК, АДМІНІСТРУВАННЯ РЕСТОРАНІВ, ПРИКЛАДНИЙ ПРОГРАМНИЙ ІНТЕРФЕЙС, ПАТЕРН MVC, АРХІТЕКТУРА КЛІЄНТ-СЕРВЕР

ЗМІСТ

ВСТУП

Робота присвячена проблемі адміністрування ресторанів, та її вирішення за допомогою веб-додатку. Не у кожного маленького чи середнього ресторану є можливість розробляти власні системи для збору статистики, збереження історії замовлень, аналітики та багато інших потрібних для розвитку речей. Для всього цього потрібно мати команду програмістів або ж замовляти таку систему у компаній які спеціалізуються на розробці веб-додатків, а це далеко не дешеве задоволення.

Роль CRM-систем в будь-якому бізнесі переоцінити складно, але особливого значення їм відводить керівництво закладів громадського харчування. Впровадження спеціального ПЗ здатне вирішити різні питання, які виникають у власників ресторанів, кафе, барів. Спеціальний софт дозволяє відстежувати переваги людей і подальше побудова цікавого для них пропозиції [1].

Система дозволяє оптимізувати і автоматизувати робочі процеси, вирішити проблеми з урахуванням залишків сировини і клієнтською базою. Це дозволяє збільшити прибуток будь-якого закладу до 30% за рахунок знання клієнтури і персоналізації. Головні завдання будь-якої системи - фінанси, аналітика, склад, онлайн-каса. Її використання зменшує навантаження на співробітників закладів громадського харчування, і дозволяє приділити особливу увагу залученню нових клієнтів і коректній роботі з завсідниками. В даний час багато компаній розробники пропонують бізнесменам продукцію такого роду [2]. Це говорить про те що попит на подібні рішення доволі високий і малоймовірно, що він знизиться найближчим часом.

У ході виконання випускної роботи буде створено систему з мінімальним набором найнеобхіднішого функціоналу, яких прокриє потреби маленьких та середніх закладів громадського харчування.

1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Історія CRM систем

Програмне забезпечення CRM програмне забезпечення для управління взаєминами з клієнтами – це термін, який відноситься до технологій, які компанії використовують для управління та аналізу взаємодій з клієнтами і даних, з метою поліпшення ділових відносин між бізнесом та користувачем, надання допомоги в утриманні клієнтів і стимулювання зростання продажів. Але програмне забезпечення CRM не завжди було надійним і автономним додатком, на яке сьогодні належить так багато компаній. За останні десятиліття він еволюціонував з безлічі інших бізнес-програм, починаючи від маркетингу баз даних та програмного забезпечення центрів обробки викликів до сучасного соціального та мобільного ПЗ зі значним рівнем автоматизації.

Аналітики CRM сходяться на думці, що еволюцію програмного забезпечення CRM можна розділити на 3 основних періоди або етапи. Вважається , що 4-й етап еволюції CRM вже почався.

1. 1980-ті роки або стадія походження.
2. 1990-ті роки або етап експансії.
3. 2000-ні роки або сучасний етап CRM.
4. Від 2010 і до нині: CRM програмне забезпечення інтегроване з хмарними сховищами [3].

Витоки сучасного програмного забезпечення CRM сягають середини 80-х років, коли прямий маркетинг перетворився в маркетинг баз даних. За допомогою статистичного моделювання зібрані дані були використані для налаштування комунікацій з потенційними клієнтами. Це період, коли компанії почали відправляти персоналізовані рекламні повідомлення та будувати більш великі кампанії, тому що контакти були більш ефективно організовані і доступні в базі даних.

Використання ПК стало більш поширеним явищем в офісному середовищі, і, крім того, архітектура клієнт/сервер була введена на робочому місці, що проклало шлях до вибухового зростання в розробці програмного забезпечення.

У другій половині 1980-х років було розроблено перше програмне забезпечення для управління контактами (CMS Software). Ці прості рішення CMS дозволили збирати та зберігати інформацію про клієнтів організованим чином, а також бути легко доступні, але не так багато ще можна було зробити з ними.

1.2 Переваги використання CRM для ресторану

По-перше, безпечно збирає дані про клієнтів [4]. Багато ресторанів не розуміють, як багато інформації вони мають у своєму розпорядженні про своїх клієнтів, тому що вона не організована належним чином.

Зазвичай у ресторані є система точок продажів, програма лояльності постачальник онлайн-замовлень, які самостійно збирають інформацію про клієнтів. Занадто часто вони знову і знову збирають одну і ту ж інформацію і не використовують її. Цей досвід шкодить ресторану і заважає спілкуватися з клієнтами цілеспрямовано і індивідуально.

Коли ресторанна CRM-система інтегрується у POS-систему, всі фрагментовані дані тепер акуратно зібрані в одному головному місці. Це дозволяє створити детальну клієнтську базу даних, яка включає в себе кілька точок даних, включаючи історію продажів, історію відвідувань і багато іншого.

Якщо адміністратор розміщуєте свої послуги онлайн-замовлення та доставки через систему POS, наприклад, про тих, хто використовував ці послуги, будуть зберігатися разом з інформацією, про гостей, які фізично прийшли в ресторан. Розуміння клієнтів є цінним, і важливо ставитися до них саме так.

По-друге, це приводить в дію програму лояльності ресторану. Програми лояльності стали більш популярними, ніж коли-небудь, тому що власники ресторанів, оператори та менеджери визнали важливість утримання клієнтів, а також залучення клієнтів.

Коли ресторанна CRM-система безпосередньо інтегрується з системою точок продажів в ресторані, програма лояльності/винагороди в ресторані стає легше просувати, контролювати і рости.

За допомогою CRM-системи є можливість визначити найкращих і лояльних гостей, а потім націлити їх на маркетингові кампанії, які одночасно показують вдячність і заохочують їх майбутнє заступництво. Це дозволить скористатися механізмом тиску на бали, феноменом, коли витрати учасника програми винагороди будуть зростати по мірі наближення до погашення винагороди.

Деякі інтегровані CRM-системи дозволяють клієнтам підписатися на програму лояльності ресторану кожен раз, коли вони дають платіж. Тепер кожен раз, коли учасник вашої програми лояльності платить за допомогою кредитної або дебетової картки, бали лояльності автоматично будуть пов'язані з їх історією контактів в інтегрованій CRM-системі ресторану.

По-третє, це дозволяє створювати персоналізовані рекламні кампанії. Потрібно інвестувати в ресторанну CRM-систему, щоб дізнатися більше про клієнтів і про те, як вони взаємодіють з бізнесом. Ця інформація покаже, як можна взаємодіяти з ними в майбутньому.

Щоб змусити все це працювати, знадобиться CRM-система, яка збирає інформацію про гостей і дозволяє легко розбивати й аналізувати дані. Це дозволить сегментувати дані, щоб показати, хто купує які товари. Можна використовувати ці ідеї для включення спеціальних маркетингових кампаній, щоб привабити гостей з значущими промо-акціями для них.

Наприклад, дані у CRM-системі можуть ідентифікувати всіх клієнтів, які замовили певну марку напою. Цей список можна використовувати для

залучення відвідувачів на рекламні заходи за участю розглянутого бренду або для заохочення відвідувань і переміщення надлишкового інвентарю.

1.3 Переваги використання CRM для клієнтів

Клієнти зазвичай обирають заклад який використовую CRM. Є багато способів, що такі системи покращують гостьової досвід:

- Лояльність клієнтів винагороджується за допомогою системи очок, яка надає безкоштовну страву або зі знижкою, коли вони досягають різних балів / меж витрат. Постійним гостям можуть бути запропоновані спеціальні знижки, перші можливості спробувати нові пункти меню, можливість дати зворотний зв'язок з ініціатив, які ви розглядаєте або реалізували, та інші пільги, які кажуть їм: "ми вас цінуємо".
- Інструменти CRM дозволяють налаштовувати пропозиції для осіб на основі демографічних характеристик, минулого перегляду та поведінки при купівлі, а також того, що інші воліють клієнти з аналогічними рисами.
- Також є можливість привітати гостей з Днем Народження, ювілеєм і запросити їх увійти зі спеціальною пропозицією, невеликий, але цінний жест.

Гостьовий трекінг, дозволяє доставити гостям меню з продуктів харчування, напоїв, нагород і вражень, тобто весь пакет, більш вузько адаптований до того, що вони хочуть. Журналіст Fox Business News Чарльз Пейн обговорював використання Amazon великих даних для доставки налаштованих CRM і маркетингу, коли він сказав: "у якийсь момент Amazon відправить вам додому речі, які ви не замовляли, але коли ви їх отримаєте, ви збережіть їх, тому що вони знали, що ви цього хочете" [5].

1.4 Огляд існуючих CRM

Copper, може похвалитися більш ніж 12 000 клієнтів, меню величезних клієнтів і чудовою інтеграцією G Suite. Таким чином, якщо потрібно

обслуговувати велике корпоративний захід або просту вечеря, Copper sharp CRM проведе через кожен крок цього процесу. Copper також має інтелектуальними інструментами відстеження продажів, щоб допомогти поліпшити меню. Наприклад, фірмова страва потребує ще декількох інградієнтів або в коктелі забагато алкоголю. Аналітика продажів говорить вам, що працює, і що ви повинні видалити негайно. Його актуальні звіти допоможуть вам зробити припущення з плануванням меню, а замість цього нехай смаки гостей спробують наступні рішення. Цінова політика: за 20 фунтів стерлінгів (на користувача в місяць) – базовий план. Для великих команд наступний крок – це £ 54,а найдорожчий пакет ціною в £93.

Перевагами даного ПЗ є:

- Безкоштовна пробна версія доступна
- Масштабований; росте разом з рестораном
- Гнучкі тарифні плани

Мінус тільки один – не призначений спеціально для ресторанної торгівлі.

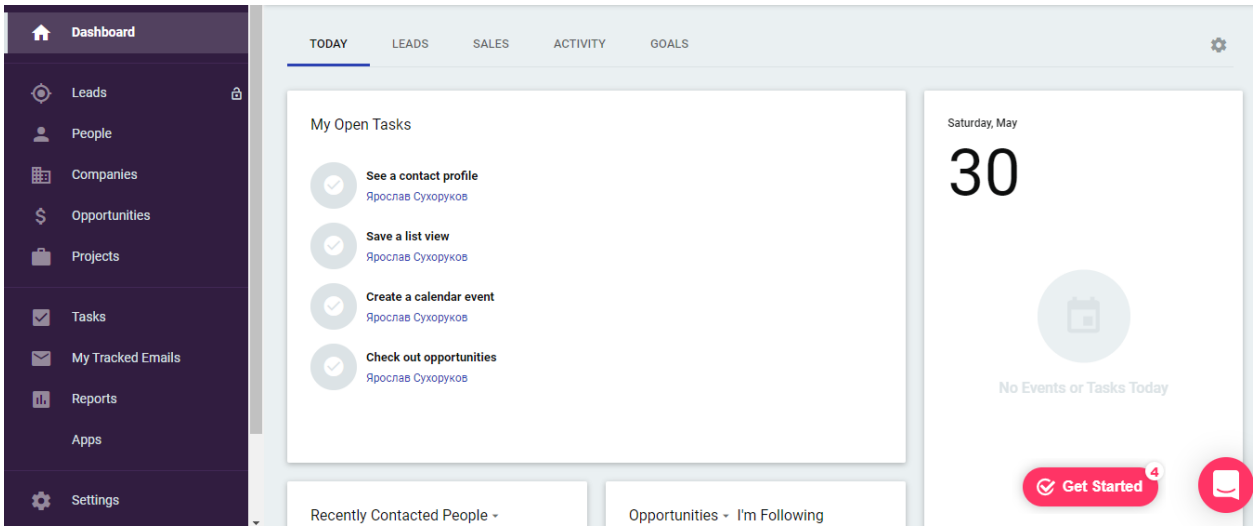

Рисунок 1.1 – Веб-інтерфейс Copper

Freshsales дозволяє взаємодіяти в реальному часі з відвідувачами сайту ресторану. Веб-форми і форми чату означають, що можна заохочувати людей забронювати онлайн або дізнаватися більше про ваш характер і меню. Також

є можливість отримати інформацію про кожного клієнта, тобто система завжди знає що і кому потрібно запропонувати. Freshsales також відмінно підходить, для мережі ресторанів в декількох місцях. Тому що він надає фунціонал для розділення персоналу на різні команди, які працюють з різних сайтів з однією і тією ж інформацією. Це гарантує, що всі співробітники знаходяться на одній сторінці, і означає, що надається послідовне обслуговування, весь час. Ціни: Freshsales CRM коштує £12 в місяць. Також є план до £20. Найдорожчі плани коштують по £40 і £65 відповідно, в місяць.

Переваги:

- Є безкоштовний пакет;
- Діапазон тарифних планів.

Серед недоліків, це інтерфейс не досить зручний.

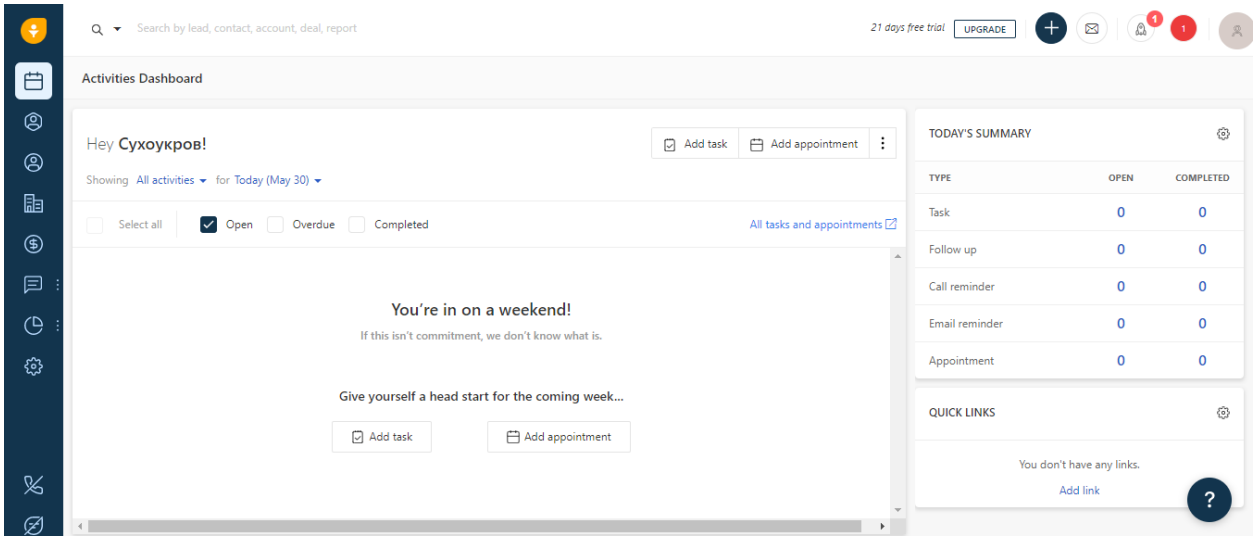

Рисунок 1.2 – Веб-інтерфейс Freshsales

CRM-інструмент TouchBistro поставляється у вигляді вбудованої функції своєї відомої системи сенсорного екрану POS. Він побудований виключно для унікальних потреб індустрії, а його програмне забезпечення CRM для ресторанів дозволяє створювати облікові записи для клієнтів з нотатками про їх дієти. Також є функціонал який дозволяє поповнити рахунок і оплатити страви та напої заздалегідь. Як результат, клієнти отримують

комфортне обслуговування. Ціни: щоб отримати доступ до CRM-функції TouchBistro, потрібно буде придбати всю систему POS, частиною якої вона є. Одна ліцензія – £ 49 в місяць, а додаткові ліцензії будуть трохи дешевше.

Переваги:

- Безкоштовна пробна версія доступна
- Сильна підтримка клієнтів
- Схема винагород стимулює ваших найвідданіших клієнтів

Недолік: доступно тільки у складі POS-рішення.

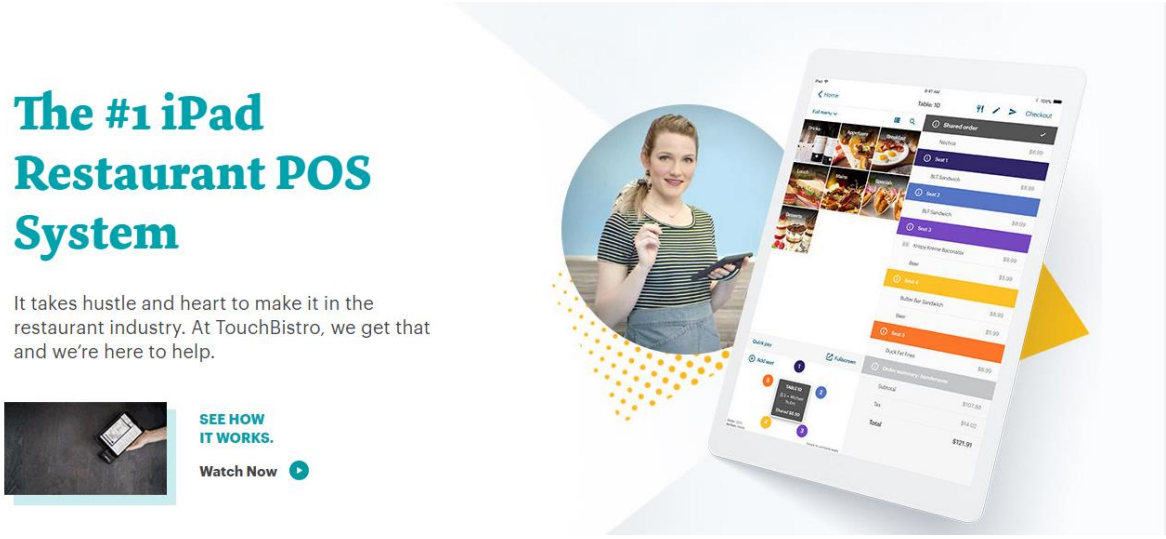

Рисунок 1.3 – Головна сторінка TouchBistro

Punchh – це перша в галузі мобільна CRM-платформа, дозволяє людям замовити онлайн і взаємодіяти з рестораном через опитування та огляди. Також є можливість адаптувати свою вже розумну систему лояльності для окремих клієнтів і винагороджувати їх подарунковими картками. Punchh CRM використовує сучасний, соціальний підхід, який обслуговує нове покоління відвідувачів ресторанів. Це рецепт, який додає до гостей не тільки відчувати себе більш цінним, але і витрачати більше грошей у ресторані. Переваги:

- Розроблена спеціально для ресторанів
- Найпростіша система для налаштування або навігації

Серед недоліків – ціноутворення не вистачає прозорості.

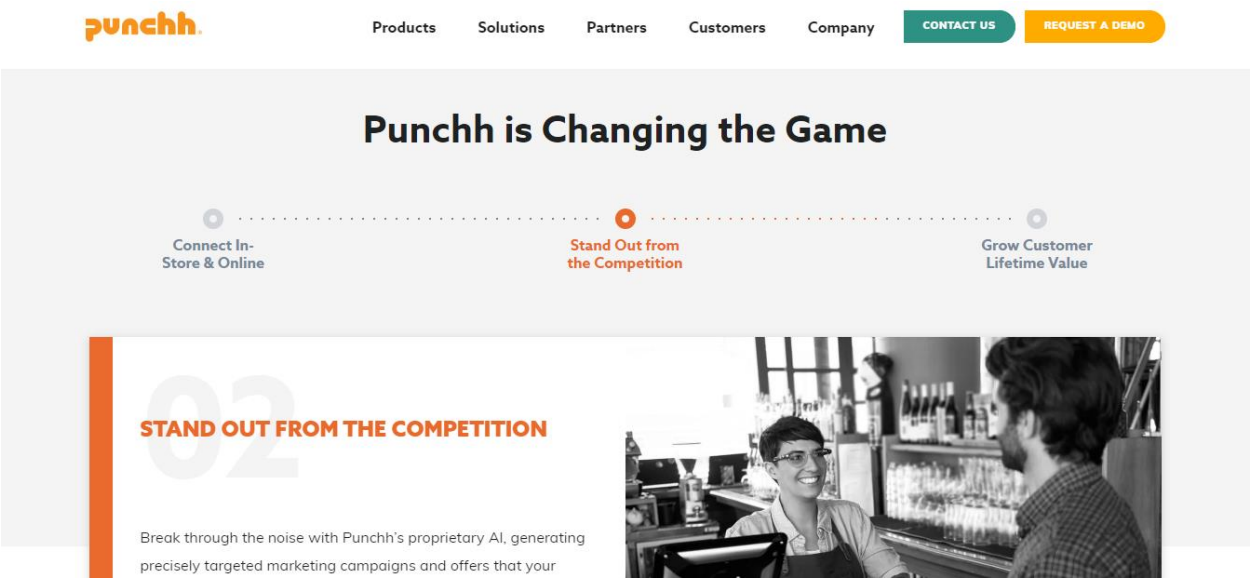

Рисунок 1.4 – Головна сторінка Punchh

Toast, програмне забезпечення CRM, яке є частиною системи POS для ресторанів. Як і TouchBistro, це CRM, який вбудований в ресторанний POS. Toast CRM відмінно підходить для поглибленої звітності про клієнтів: дізнатись, як часто клієнти приходять, що вони замовляють, і, що особливо важливо, скільки вони витрачають. Коли є ці ключові інгредієнти, CRM Toast може стати найрозумнішим маркетинговим інструментом ресторану. CRM також допомагає визначити, кого з постійних клієнтів не було протягом деякого часу. Потім можна запропонувати нові знижки на їх улюблену їжу і напої. Створювати рекламні акції, адаптувати їх до своєї аудиторії. Ціни на програмне забезпечення POS Toast починається з трохи більше £ 60 в місяць, з платою за установку близько £390.

Переваги:

- Підтримка клієнтів 24/7
- Діапазон інтеграції API для розширення функціональних можливостей

Недолік – ціна.

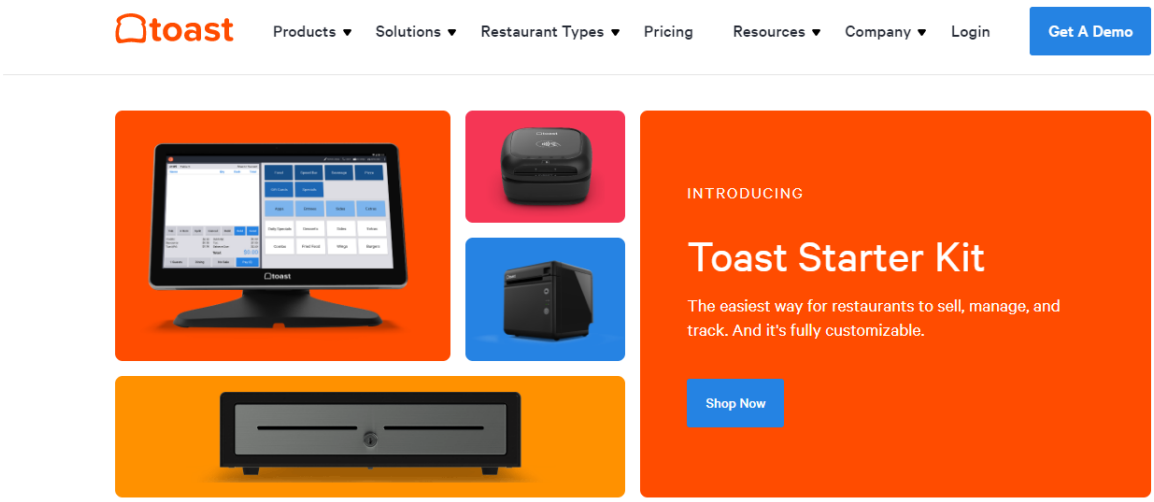

Рисунок 1.5 – Головна сторінка Toast

1.5 Аналіз та постановка задачі

Проаналізувавши вже існуючі аналоги, а також переваги для ресторанного бізнесу та його клієнтів, можна побачити, що існує попит на системи, які проводять цінову політику, котра дозволить кожному спробувати CRM та вирішити необхідність і корисність тих, чи інших функцій.

Тож потрібен фунціонал, який не буде перевантаженим, та занадто складним для освоєння, також не буде відлякувати клієнтів складним процесом замовлення та збором інформації про них:

- збір аналітики, який дозволить бачити на яку загальну суму було зроблено замовлень, та їх кількість;
- можливість порівняти виторг та кількість замовлень за останній час;
- перегляд історії замовлень;
- можливість зробити замовлення через веб-додаток;
- адміністратор повинен мати змогу змінити категорії в меню (додати нову або ж змінити існуючу);
- можливість змінювати пункт меню в категорії, видали його або додати новий.

2. МЕТОДИ РІШЕННЯ ПРОБЛЕМИ

2.1 Основні положення

Модель клієнт-сервер – це структура додатків, яка розподіляє роботу та навантаження між клієнтом та сервером. Часто клієнти та сервери знаходяться на різних комп’ютера, але є і виключення. Клієнти можуть ініціювати сеанси зв'язку із серверами, які лише відповідають на запити. Обидва елементи системи незалежні один від одного. Частіше всього сервер отримує запити від декількох клієнтів. Найпопулярнішим форматом для такого спілкування є json [6]. У такої структури є свої переваги:

- код не повторюється;
- обробка даних відбувається лише на сервері, то ж вимоги до комп’ютера клієнта зменшуються;
- данні більш захищені, так як зберігаються на серверах;
- система здатна краще масштабуватись, оскільки сервер не повинен зберігати стан клієнта між запитами. якщо не потрібно пам’ятати інформацію про стан клієнта, визволяєте ресурси сервера, щоб вони могли обслуговувати більше клієнтів одночасно;
- код на вимогу, це необов'язкове обмеження, і воно дозволяє клієнтам завантажувати програми для виконання на стороні клієнта. тобто ми завантажуємо код певної сторінки, тільки якщо клієнт переходить на неї. зараз це може здатися дуже очевидним, але в ранні віки інтернету це було розвиваючою концепцією і є корисною частиною архітектури інтернету.

MVC – це шаблон проектування, передбачає розподіл програми, на три окремих компоненти: Модель, Представлення і Контролер, це дозволяє доповнювати або ж модифікувати кожен елемент незалежно один від одного.

Моделі – це частини програми, що реалізують логіку домену даних програми. Об'єкти моделей отримують і зберігають в базі даних їх стан. Один

із прикладів, об’єкт клієнта може отримати інформацію з бази даних, працювати з нею, а потім перезаписати в таблицю відредаговану інформацію.

Вид – це компоненти, які відображають інтерфейс програми для користувача (UI). Зазвичай інтерфейс бере дані з моделі та дозволяє користувачеві бачити їх, та проводити будь-яку взаємодію(натискання на кнопку, перехід по посиланню та інше).

Контролери – це компоненти, які керують взаємодією користувачів, вони реагують на дії та в залежності від них змінюють дані моделі [7].

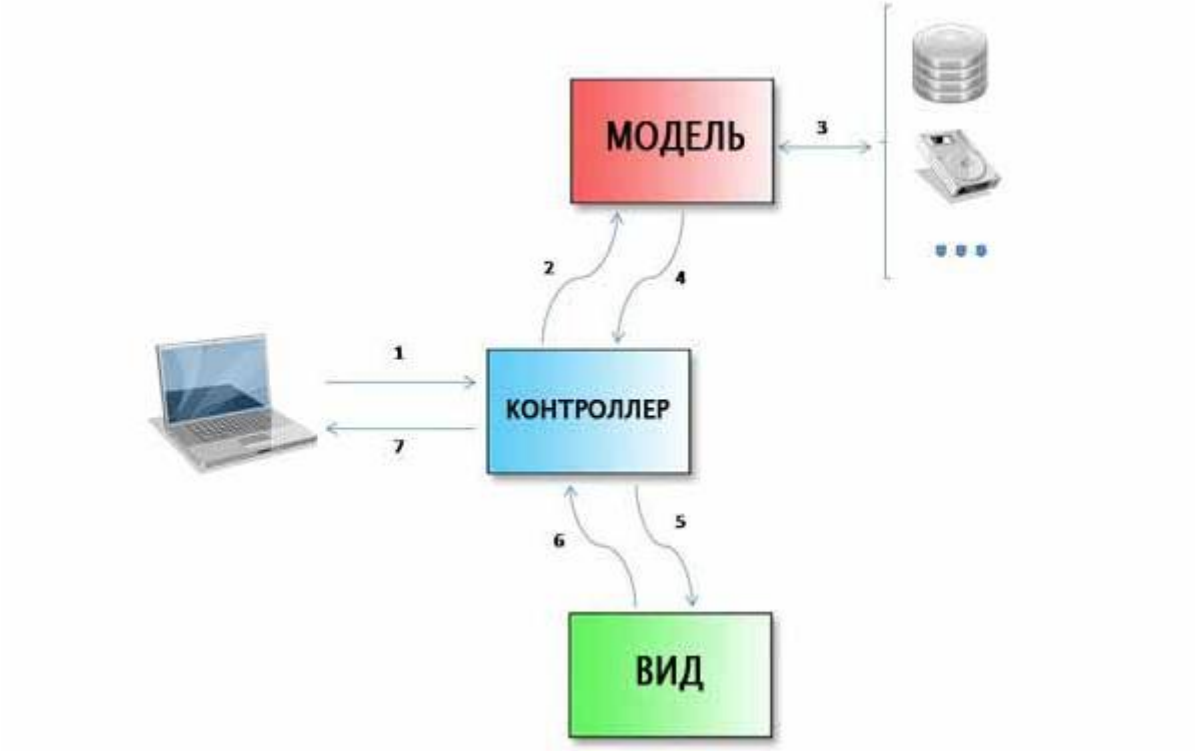

Рисунок 2.1 – Шаблон MVC

2.2 Інструменти розробки

Розробка буде клієнтської частини буде відбуватись на Angular. Це front-end фреймворк з відкритим кодом, який розробляється під керівництвом Angular Team у компанії Google, а також спільнотою приватних розробників та корпорацій. Angular – це AngularJS, який переосмислили та який був повністю переписаний тією ж командою розробників.

Даний фреймворк нині є одним із найпопулярніших на рівні з React та Vue. А враховуючи що за його розробкою стоїть IT-гіганг Google – можна з впевненістю говорити що це гарний інструмент для розробки веб-додатків.

Далі потрібно обрати CSS фреймворк, який буде задовольняти потреби в стилі та зручності його використання, MaterializeCSS – гарний вибір. Він заснований на принципах посібників Google Material Design, на відміну від його аналогу twitter bootstrap, який звісно є потужним інструментом, є більш сучасним та візуально приємнішим.

Для побудови графіків, буде використовуватись Chart.js – це бібліотека JavaScript з відкритим кодом для візуалізації даних. Створена лондонським веб-розробником Ніком Дауні в 2013 році, тепер він підтримується громадою і є другою за популярністю бібліотекою графіків JS на GitHub за кількістю зірок після D3.js, що вважається значно простішим у використанні, хоча і менш налаштованим, ніж останнє. Chart.js відображається в полотні HTML5 і широко висвітлюється як одна з найкращих бібліотек візуалізації даних. Вона доступна за ліцензією MIT.

Що до бекенду, то його розробка буде відбуватись на Node.js, що представляє середу виконання коду на JavaScript, яка побудована на основі Chrome V8, що дозволяє транслювати виклики на мові JavaScript в машинний код. Node.js перш за все призначений для створення серверних додатків. Хоча також можна згадати Electron, він дозволяє створювати десктопні додатки або ж навіть створення коду для мікроконтролерів. Але перш за все Node.js, це платформа для створення веб-додатків. Node.js є відкритим проектом, вихідні коди якого можна подивитися на github.com.

Фреймворк для бекенду, найпопулярніший та найпростіший в освоєнні – express. Це мінімалістичний і гнучкий веб-фреймворк написаний на JavaScript, що надає великий набір функцій для мобільних і веб-додатків. Маючи в своєму розпорядженні безліч службових методів HTTP і проміжних оброблювачів, дозволяє створити надійний API легко і досить швидко. Express

надає тонкий шар фундаментальних функцій веб-додатків, які не заважають працювати з давно знайомими і улюбленими вами функціями Node.js.

База даних, MongoDB – це міжплатформна програма, орієнтована на документи. MongoDB, класифікована як програма баз даних NoSQL, використовує документи, подібні JSON, зі схемою. MongoDB розроблений компанією MongoDB Inc. та ліцензується під публічною ліцензією на стороні сервера (SSPL). MongooseJS – це ODM, який спрощує використання MongoDB шляхом перекладу документів в базі даних MongoDB об'єкти в програмі. Крім MongooseJS є кілька інших ODM.

Три основних переваги використання мангуст проти рідної MongoDB є:

- MongooseJS надає шар абстракції поверх MongoDB, який виключає необхідність використання іменованих колекцій.
- Моделі виконують основну частину роботи по встановленню значень за замовчуванням для властивостей документа і перевірці даних.
- Функції можуть бути добавлені до моделей в MongooseJS. Це дозволяє внести нову функціональність.
- Запити використовують ланцюжок функцій, що призводить до того, що код є більш гнучким і його легше читати, отже, він його простіше підтримувати та вдосконалювати.

Кінцевим результатом цього є спрощення доступу до бази даних з прикладних програм. Головним недоліком Mongoose є те, що абстракція відбувається за рахунок продуктивності порівняно з рідним MongoDB.

Mongoose використовує схеми для моделювання даних, які додаток хоче зберігати і маніпулювати в MongoDb. Це включає в себе такі функції, як приведення типів, перевірка, побудова запитів і багато іншого. Схема описує атрибути властивостей (полів), якими буде управляти додаток. Ці атрибути включають в себе такі речі, як:

- Тип даних (наприклад, рядок, число і т. д.).
- Це є обов'язковим або необов'язковим полем.

- Чи є це значення унікальним, що означає, що база даних може містити лише один документ з цим значенням в даній властивості.

Модель складається з схеми і визначає документ, з якими буде працювати програма. Більш точно, модель-це клас, який визначає документ з властивостями і поведінкою, оголошеними в схемі. Всі операції бази даних, що виконуються над документом з використанням mongoose, повинні посилатися на модель.

Перше відмінність між Mongoose і нативним додатком MongoDB полягає в тому, що в каталозі повинен бути створений модуль, що містить схему і модель. Рекомендується, щоб цей файл мав те ж ім'я, що і модель. Перший символ цього файлу знаходиться у верхньому регістрі, так як модель-це клас, побудований з схеми. Як і будь-який клас, це перший символ, тому він повинен бути великою літерою.

Moment.js – це бібліотека JavaScript для управління датами і часом, без інших залежностей; це потужний інструмент для аналізу, перевірки і відображення дат. Коли дати повинні відображатися в локалізованому форматі, наданому місцем розташування користувача і правильним форматом, він підтримує інтернаціоналізацію і часовий пояс, що є безсумнівною перевагою. Вона має певні переваги над своїми аналогами:

- Він працює як в браузері, так і node.js environments. При використанні бібліотеки, яка управляє часом, важливо, щоб працювати можна було в незалежності від середи розробки.

- Підтримує i18n (інтернаціоналізацію) і l10n (локалізацію), що дає можливість швидко налаштувати час для будь якої мови та часового поясу. Це означає, що не потрібно створювати додаткових файлів(наприклад, створення перекладу) для додаткових переказів для мов, які вже реалізовані в наших програмах.

- Набір доступних опцій дивний: він надає майже все, що може використовувати розробник frontend в додатку. Іноді здається що там є все.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проектування бекенду

Для спрощення розробки, та тестування API, потрібно спочатку зробити перелік всіх посилань, та їх короткий опис.

Таблиця 3.1 – Короткий опис посилань на API

Метод	Посилання	Що робить
POST	/api/auth/login	Авторизувати користувача
POST	/api/auth/register	Зареєструвати користувача
GET	/api/analytics/overview	Отримати аналіз за вчора
GET	/api/analytics/analytics	Отримати повну аналітику
GET	/api/category	Отримати всі категорії
GET	/api/category/:id	Отримати категорію за її id
DELETE	/api/category/:id	Видалити категорію за її id
POST	/api/category	Додати категорію
PATCH	/api/category/:id	Редагувати категорію
GET	/api/order	Отримати всі замовлення
POST	/api/order	Додати замовлення
GET	/api/position/:categoryId	Отримати всі товари за id категорії
POST	/api/position	Додати пункт меню
PATCH	/api/position /:id	Редагувати пункт меню
DELETE	/api/position /:id	Видалити пункт меню

3.2 Створення HTML/CSS розмітки

На рисунку 3.1 зображено сторінку логіну, на ній користувач може авторизуватись в системі, якщо логін і пароль були введені коректно, якщо ж ні, то про це повідомить система і не пропустить його далі

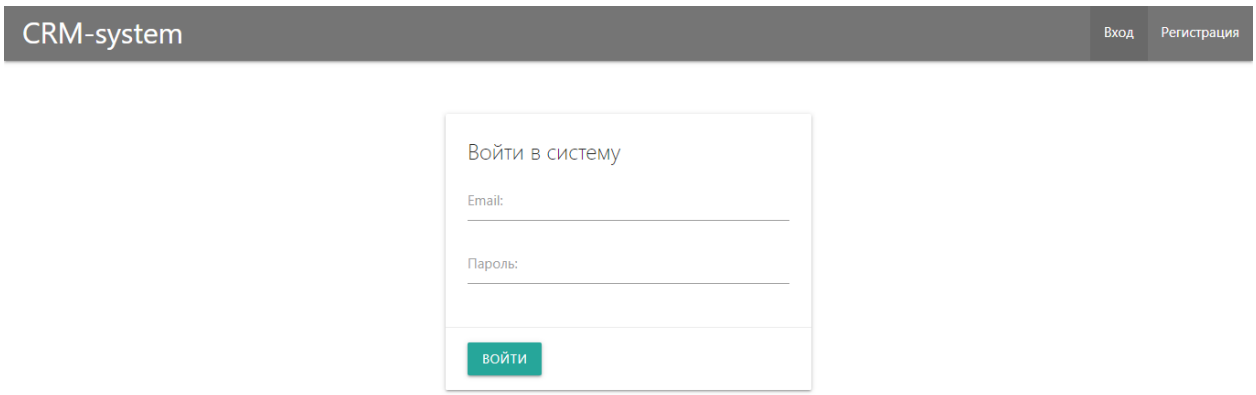

Рисунок 3.1 – Сторінка входу в систему

На рисунку 3.2 зображено сторінку реєстрації, на ній користувач може зареєструватись в системі, якщо такої поштової скриньки ще немає в системі, та введений пароль більше 6 символів:

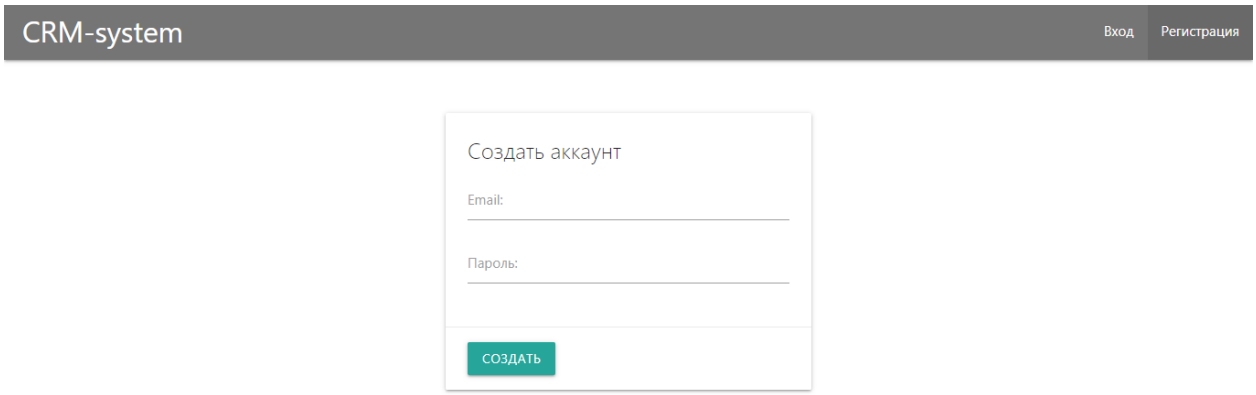

Рисунок 3.2 – Сторінка реєстрації

На рисунку 3.3 зображено сторінку огляду, на ній зображено порівняння вчорашнього виторгу та кількості замовлень з середнім значенням:

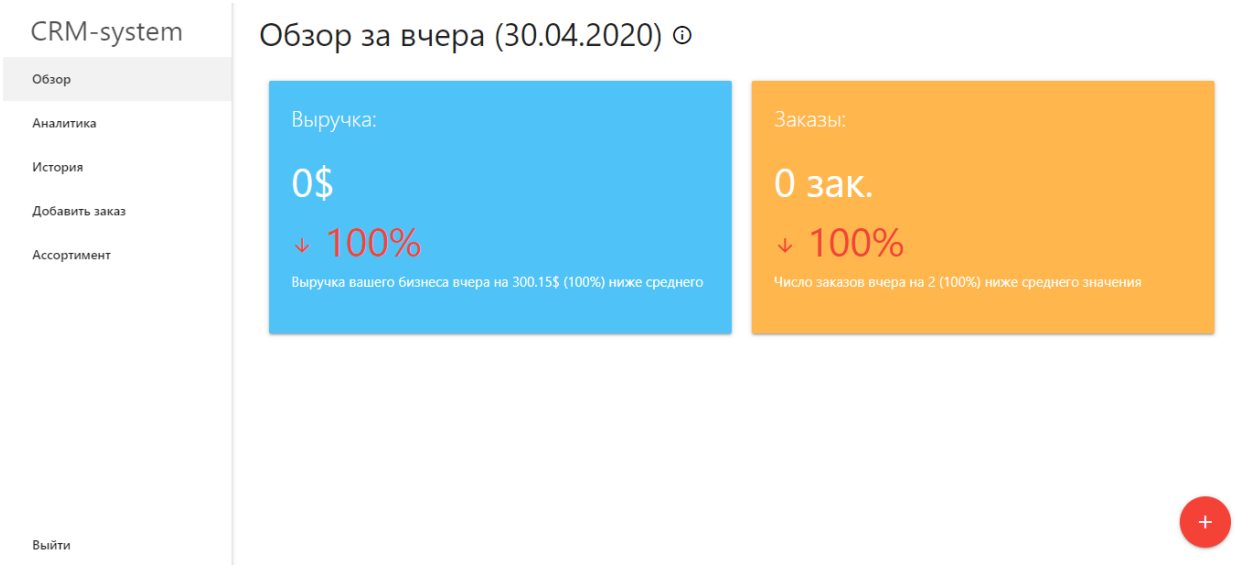

Рисунок 3.3 – Сторінка огляду

На рисунку 3.4 зображено пояснення до сторінки огляду для нових користувачів:

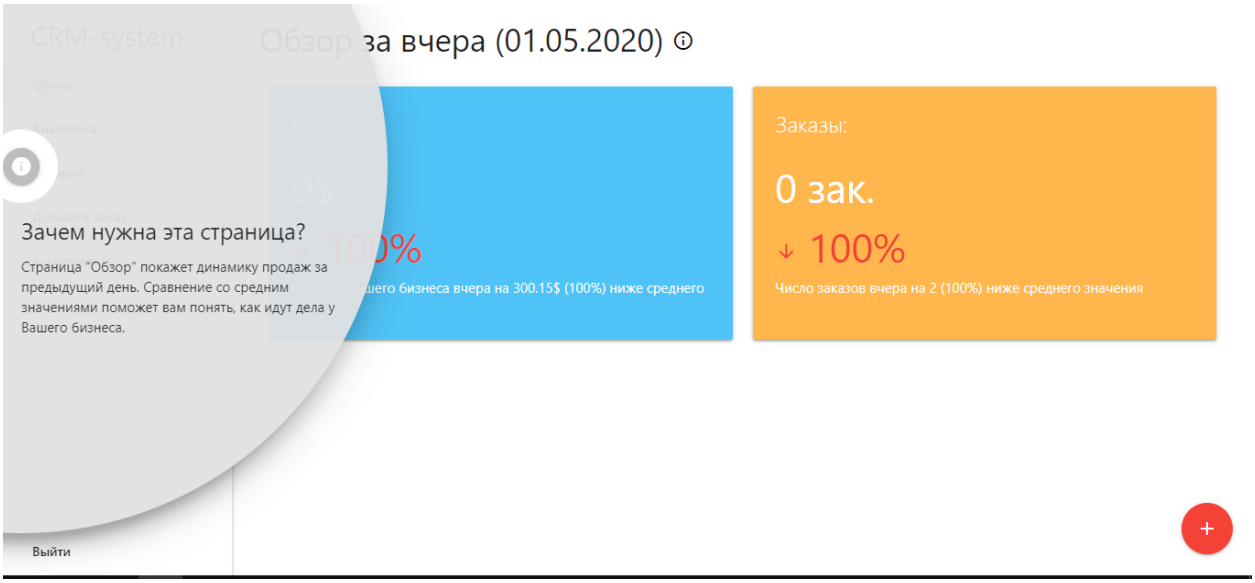

Рисунок 3.4 – Сторінка огляду з поясненням

На рисунку 3.5 зображено сторінку аналітики, за допомогою неї можна подивитись графіки зміни виторгу та кількості замовлень:

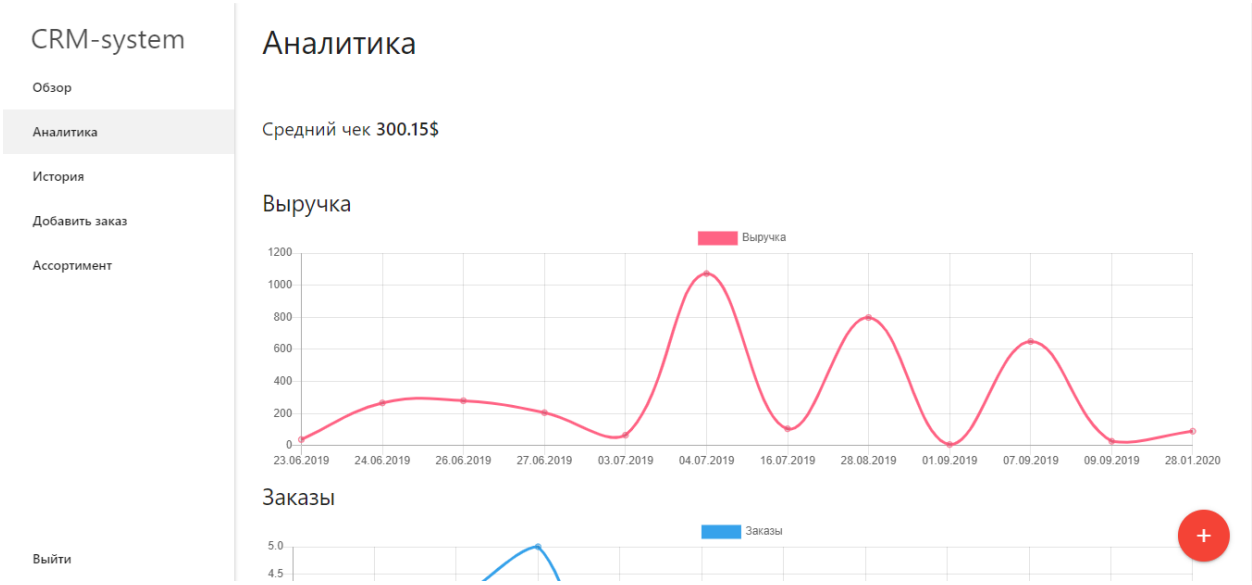

Рисунок 3.5 – Сторінка аналітики

На рисунку 3.6 зображено сторінку історії, вона допомагає переглянути конкретне замовлення, або ж знайти замовлення за вказаними фільтрами:

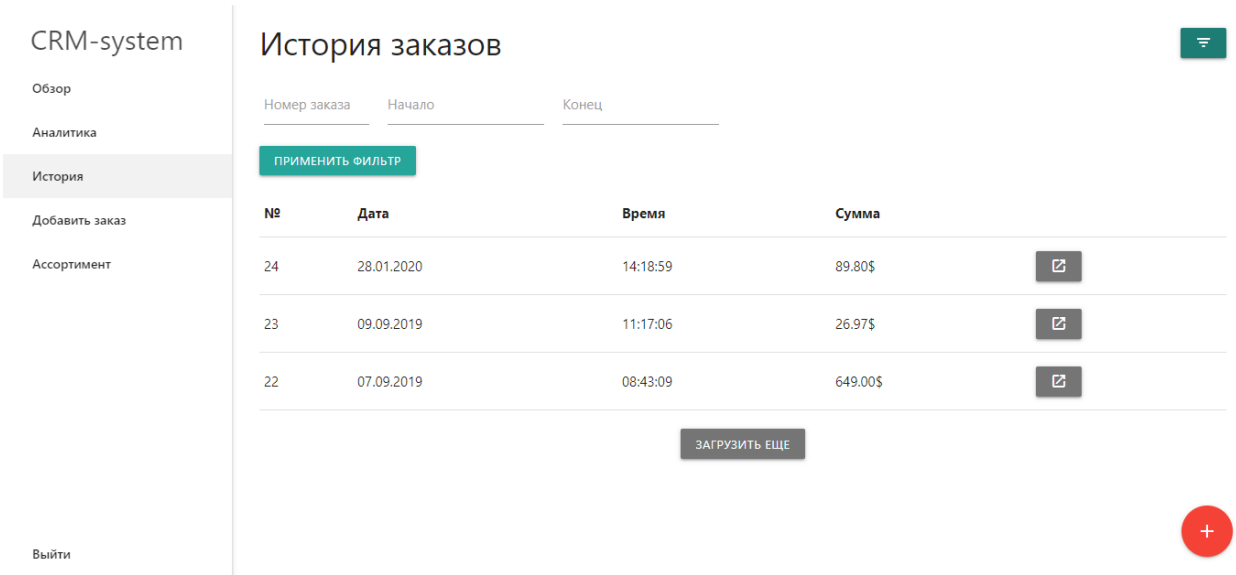

Рисунок 3.6 – Сторінка історії замовлень

На рисунку 3.7 зображено виконане замовлення, яке можна переглянути з сторінки історії:

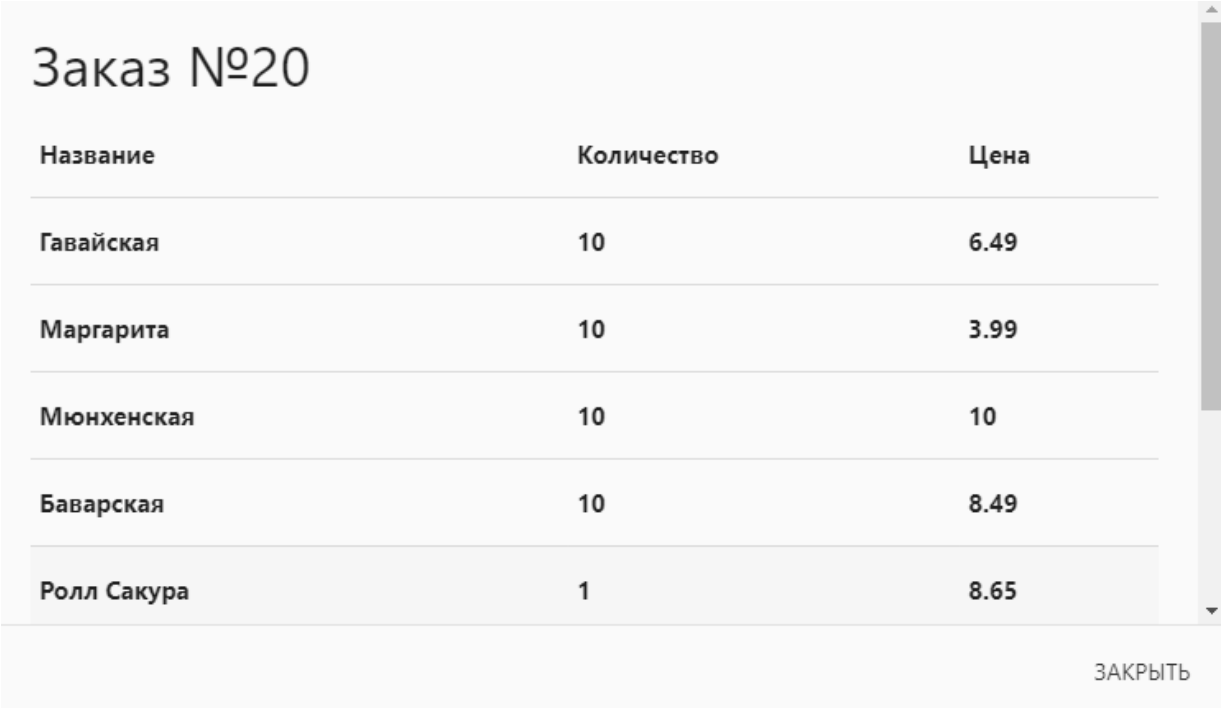

Рисунок 3.7 – Сторінка історії замовлень

На рисунку 3.8 зображено сторінку створення замовлення, за допомогою неї можна вибрати категорію потрібного продукту:

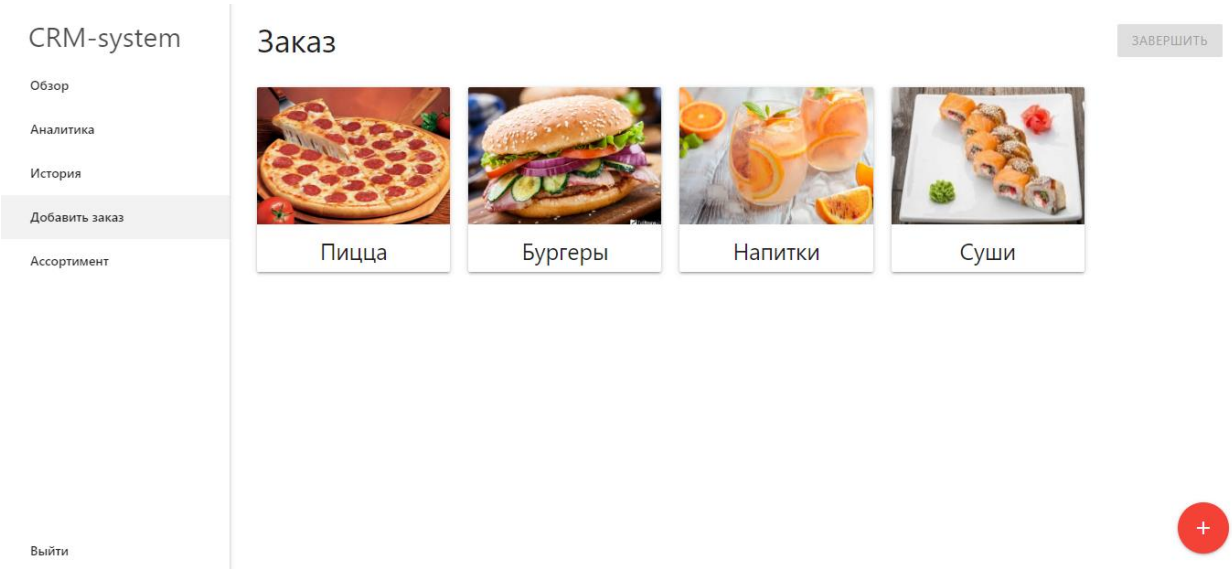

Рисунок 3.8 – Сторінка вибору категорії при замовленні

На рисунку 3.9 зображено сторінку створення замовлення, на ній можна добавити певну позицію в замовлення:

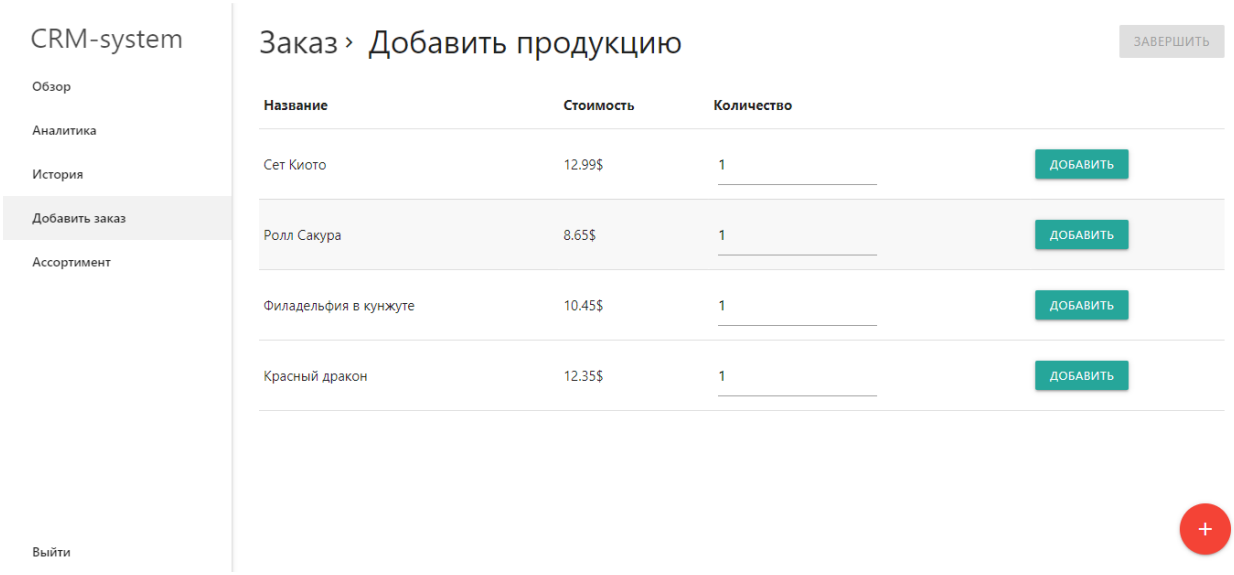

Рисунок 3.9 – Сторінка замовлення

На рисунку 3.10 зображено сторінку асортименту, вона дозволяє вибрати певну категорію для редагування або ж перейти до створення нової.

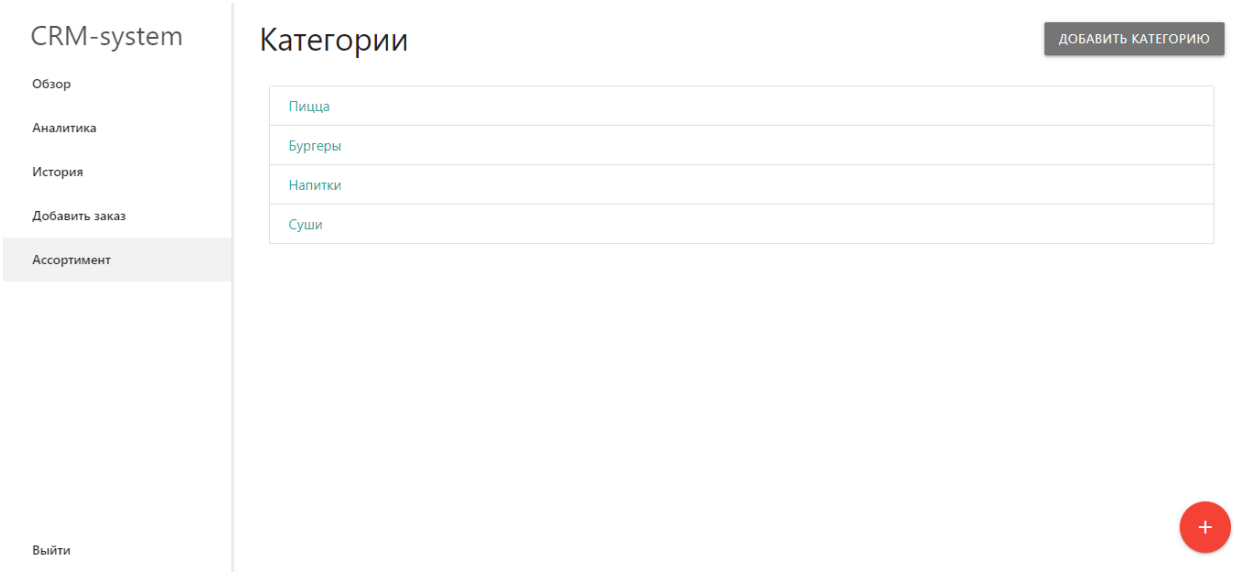

Рисунок 3.10 – Сторінка редагування асортименту

На рисунку 3.11 зображено сторінку створення нової категорії, потрібно ввести назву, та за бажанням завантажити фото, воно повинне бути формату jpeg або png, та не більше 5 мб.

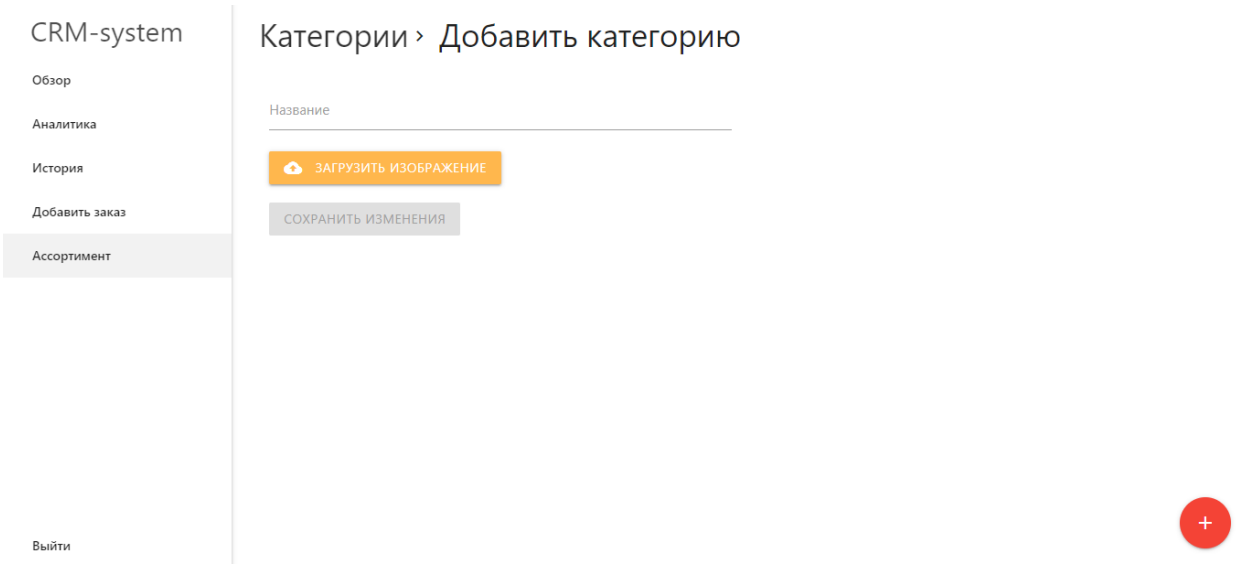

Рисунок 3.11 – Сторінка створення категорії

На рисунку 3.12 зображено сторінку редагування категорії, вона дозволяє саме редагувати назву, та змінити фото, або ж видалити категорію. Також можна перейти до створення товару, редагування його або ж видалення, якщо він більше не потрібен.

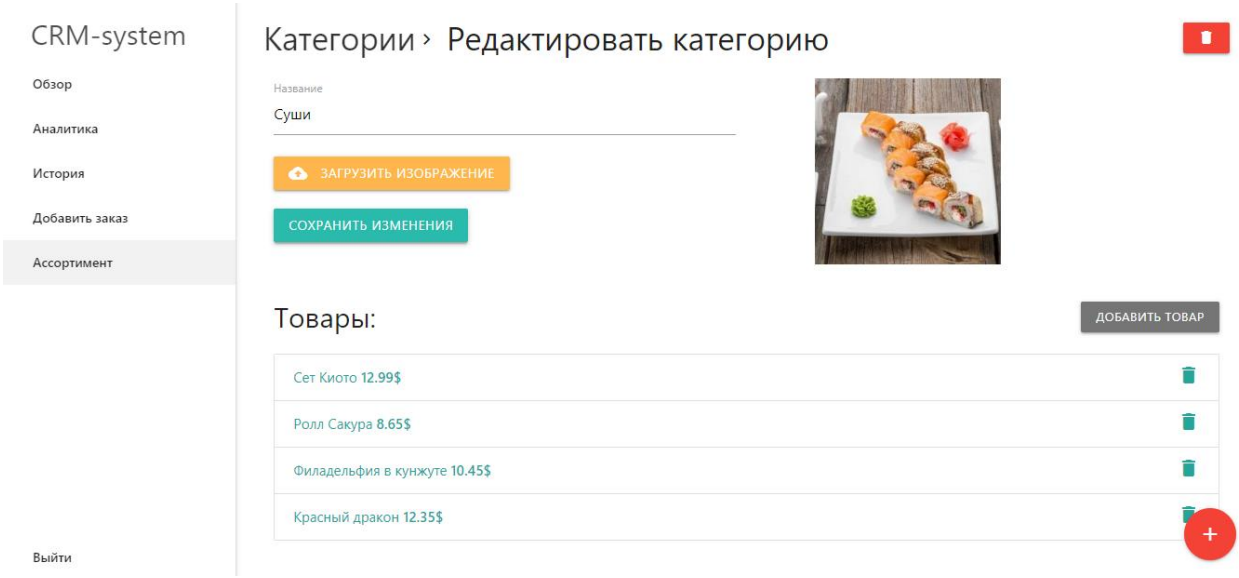

Рисунок 3.12 – Сторінка редагування категорії

На рисунку 3.13 зображено поп-ап створення товару, для цього потрібно ввести назву, та ціну, вона повинна бути більшою ніж 0,01.

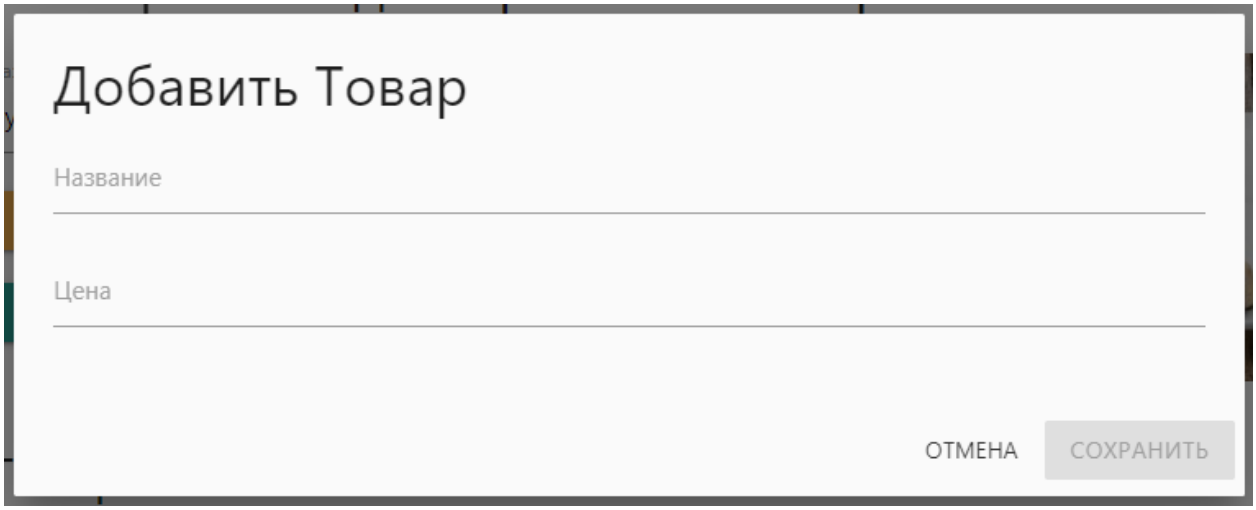

Рисунок 3.13 – Сторінка створення товару

3.3 Розробка API

Back-end складається з наступних елементів:

- файлів конфігурації для задання ключів доступу;
- контролерів для виконання роботи над даними;
- файлів для виконання проміжних дій (middleware);
- моделей даних для mongoDB;
- описання маршрутів (routing);
- файлу app.js, для підключення бібліотек в проект;
- файлу index.js, головний файл для запуску API.

Файлову структуру можна побачити на рисунках 3.14-15:

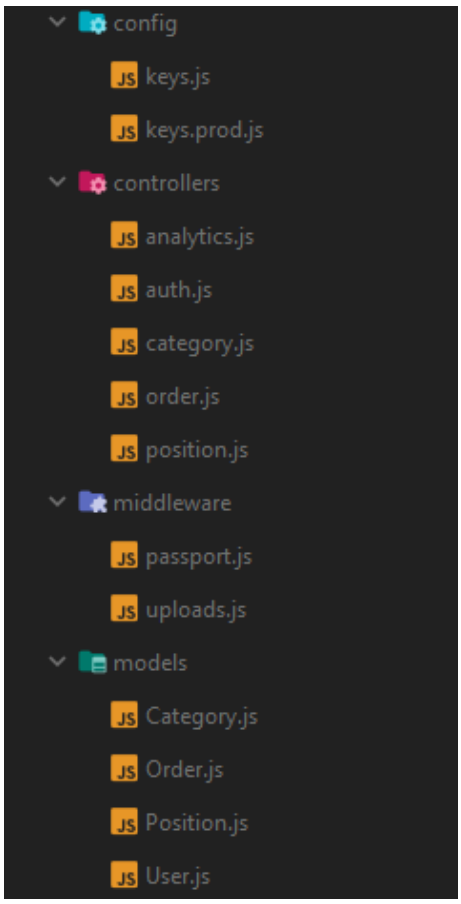

Рисунок 3.14 – Структура API частина 1

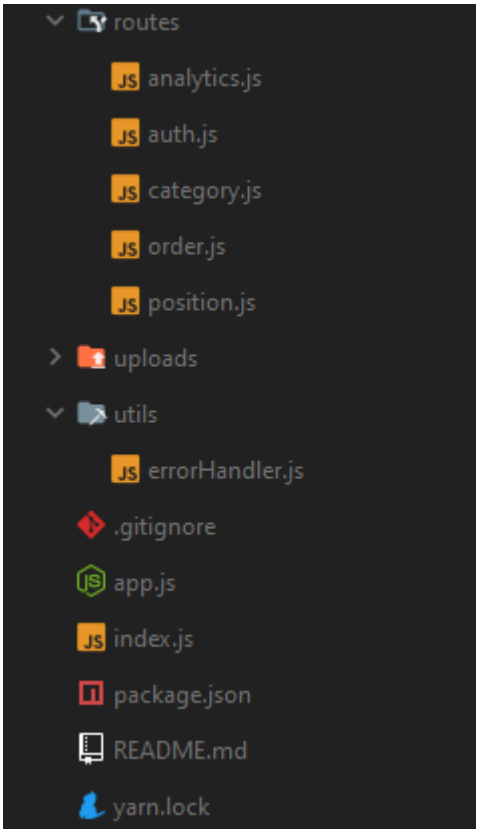

Рисунок 3.15 – Структура API частина 2

При розробці бекенду слід не забувати про безпеку даних, тому паролі слід зберігати в хешованому вигляді. Для цього використаємо бібліотеку bcrypt. Ця бібліотека має свої особливості, наприклад солоне хешування-генерація випадкових байтів (сіль) і об'єднання їх з паролем перед хешуванням, створює унікальні хеші за паролем кожного користувача. Якщо два користувачі мають один і той же пароль, вони не будуть мати один і той же хеш пароля. Також використовує алгоритми хешування, які є односторонніми функціями . Вони перетворюють будь-яку кількість даних “відбиток пальця " фіксованої довжини, який не можна повернути назад. У них також є властивість, що якщо вхідні дані змінюються навіть на крихітний біт, результуючий хеш повністю відрізняється (див. приклад вище). Це відмінно підходить для захисту паролів, тому що потрібно зберігати паролі у формі, яка захищає їх, навіть якщо сам файл паролів скомпрометований, але в той же час потрібно мати можливість перевірити, що пароль користувача є правильним.

Робочий процес реєстрації і перевірки достовірності облікових записів в системі облікових записів на основі хеш виглядає наступним чином:

- користувач створює обліковий запис.
- пароль хешується і зберігається в базі даних. Ні в якому разі простий текст (незашифрований пароль ніколи не записується на жорсткий диск).
- коли користувач намагається увійти в систему, хеш пароля, який він ввів, перевіряється проти хеша їх справжнього пароля (отриманого з бази даних).
- якщо хеші збігаються, користувачеві надається доступ. Якщо ні, користувачеві кажуть, що вони ввели неприпустимі дані для входу.

Bcrypt дозволяє вибирати вартість saltRounds, що дає контроль над складністю обробки даних. Чим більше це число, тим більше часу потрібно машині для обчислення хеша, пов'язаного з паролем. Дуже важливо при виборі цього значення, щоб вибрати число досить високе, щоб той, хто намагається підібрати пароль користувача за допомогою підстановки, витратив стільки часу, щоб згенерувати всі можливі хеші паролів, що результат не компенсує

його. І з іншого боку, він повинен бути достатньо малим, щоб не випробовувати терпіння користувача при реєстрації та вході в систему (це терпіння зазвичай невелике). За замовчуванням значення saltRounds = 10.

Коли пароль введений правильно, потрібно налагодити зв'язок між клієнтом та сервером. В даному випадку використаємо jwt. JWT-це стандарт, який визначає компактний і автономний спосіб безпечної передачі інформації між клієнтом і сервером в якості об'єкта JSON. Завдяки компактному розміру токени легко переносяться через URL-адресу, параметр POST або всередині заголовка HTTP. Крім того, оскільки вони автономні, то містять всю необхідну інформацію про користувача. Інформація в JWT може бути довірена, тому що вона має цифровий підпис з використанням пари секретних або відкритих/закритих ключів.

JWT в основному використовуються для аутентифікації. Після входу користувача, додаток створить JWT і відправить його назад користувачеві. Наступні запити користувача будуть включати в себе JWT. Маркер вказує серверу, до яких маршрутів, служб і ресурсів дозволено доступ. JWTs можна використовувати в якості механізму аутентифікації, який не вимагає наявності бази даних. Сервер не може використовувати базу даних, так як сховище даних у JWT, відправлений клієнту, безпечно. Може скластись враження, що використання JWTs для токенів сеансу завжди ідеальний варіант:

- є можливість зберігати будь-які відомості про користувача на клієнті;
- сервер може довіряти клієнтові, тому що JWT підписаний, і немає необхідності викликати базу даних для отримання інформації, яка вже зберігається в JWT
- не потрібно координувати сеанси в централізованій базі даних, коли з’являється доступ до можливої проблеми горизонтального масштабування.

Зрештою, якщо вже є база даних для додатку, можна просто використовувати таблицю сеансів і регулярні сеанси, що надаються обраною серверною платформою. Використання JWTs пов'язане з певними витратами: вони відправляються на сервер для кожного запиту, і це завжди витрачає

ресурси порівняно з сеансами на стороні сервера. Крім того, хоча ризики для безпеки мінімізовані, відправляючи JWTs з допомогою HTTPS, завжди є можливість, що його перехоплять і дані користувача розшифровуються. Але в нашому випадку, використання токенів гарна ідея, адже наші користувачі, це менеджери та офіціанти, вони використовують приватні точки доступу до мережі.

3.4 Написання Front-end

Клієнтська частина складається з наступних елементів:

- Головного модуля app.module.ts
- Модуля навігації app-routing.module.ts
- Головного компонента app.component.ts
- Модуля авторизації login.module.ts
- Модуля самого додатку site.module.ts
- Компонентів кожної із сторінок:
    - analytics-page.component.ts
    - categories-page.component.ts
    - history-page.component.ts
    - order-page.component.ts
    - overview-page.component.ts
    - site-layout.component.ts

- Shared модуля – тут зберігаються всі компоненти які можна використати повторно, експортуючи їх з нього і імпортуючи його в ті модулі проекту, де ці компоненти можуть знадобитися. У такий модуль можна помістити компоненти кнопки, списку, якого-небудь стилізованого блоку тексту і т.д, а також різні директиви і пайпи.

o Core модуля – він містить код, який буде використовуватися для ініціалізації вашого додатка та завантаження деяких основних функціональних можливостей.

- Сервісів для роботи з API:
    - analtics.service.ts – запити для сторінки аналітики
    - categories.service.ts – запити для категорії
    - orders.service.ts – запити для сторінки історії
    - position.service.ts – запити для сторінки замовлень
    - auth.service.ts – запити для сторінки авторизації
- auth.guard.ts – дозволяє обмежити навігацію по визначених маршрутах. Наприклад, якщо для доступу до певного ресурсу потрібна аутентифікація або якісь інші умови, в залежності від яких ми можемо надати користувачеві доступ, а можемо і не надати.
- material.service.ts – через нього проходить взаємодія додатку з бібліотекою materialize css
- token.interceptor.ts – перехоплювач, перебуває між додаткам та бекендом. Коли додаток робить запит, то його переробляють перед відправкою на сервер (додають заголовки до запиту). Застосовується для аутентифікації.
- interfaces.ts – тут знаходяться всі інтерфейси які використовує додаток.

ВИСНОВКИ

Під час розробки проекту для випускної роботи, на javascript, за допомогою сучасних фреймворків та бібліотек, було написано систему яка вирішує більшість повсякденних проблем пов'язаних з аналітикою та адмініструванням ресторанів:

- Збір аналітики, який дозволить бачити на яку загальну суму було зроблено замовлень, та їх кількість.
- Можливість порівняти виторг та кількість замовлень за останній час.
- Перегляд історії замовлень.
- Можливість зробити замовлення через веб-додаток.
- Користувач може редагувати чи додавати нові категорії.
- Змінювати пункти меню в категорії, видаляти їх або редагувати.

СПИСОК ЛІТЕРАТУРИ

1. Огляд crm-систем для ресторанного бізнесу – Режим доступу: https://crm-systems.info/crm-dlya-kafe/#__CRM
2. Кращі CRM системи для ресторану, кав'ярні і кафе – Режим доступу: https://neurocrm.ru/blog/2019/03/14/crm-sistemy-dlja-restorana/
3. History and Evolution of CRM Software – Режим доступу: https://viennaadvantage.com/blog/technologies/history-of-crm-software/
4. Why Your Restaurant Needs an Integrated Guest CRM System – Режим доступу: https://www.ieateryapp.com/blog/why-your-restaurant-needs-an-integrated-guest-crm-system.html
5. Does It Make Sense to Use CRM for Restaurants? – Режим доступу: https://www.therail.media/stories/2017/7/31/does-it-make-sense-to-use-crm-for-restaurants
6. Клієнт-серверна архітектура та ролі серверів. – Режим доступу: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229
7. Model-View-Controller (MVC) – Режим доступу: https://medium.com/datadriveninvestor/model-view-controller-mvc-75bcb0103d66
8. Фримен Адам. Angular для профессионалов, 2017 – 800 ст.
9. Кайл Симпсон. You Don't Know JS : Scope & Closures. 2014 – 98 ст.
10. Кайл Бэнкер: MongoDB в действии. 2017 – 394 ст.
11. Итан Браун. Веб-разработка с применением Node и Express. Полноценное использование стека JavaScript. 2016 - 336 ст.
12. Simon Holmes. Mongoose for Application Development. 2013 – 233 ст.
13. Simon Holmes. Getting MEAN with Mongo, Express, Angular, and Node. 2015 – 504 ст.
14. Mario Casciaro. Node.js Design Patterns. 2014 – 520 ст.

15. David Herman. Effective JavaScript: 68 Specific Ways to Harness the Power of JavaScript. 2012 – 240 ст.
16. Jeremy Wilken. Angular in Action. 2018 – 320 ст.
17. Ari Lerner, Felipe Coury, and Nate Murray. Ng-Book 2: The Complete Book on Angular 2. 2016 – 609 ст.
18. Boris Cherny. Programming TypeScript: Making Your JavaScript Applications Scale. 2019 – 394 ст.
19. Dan Vanderkam. Effective TypeScript: 62 Specific Ways to Improve Your TypeScript. 2019 – 266 ст.
20. Jon Duckett. HTML & CSS: Design and Build Web Sites. 2011 – 514 ст.
21. Jennifer Niederst Robbins. Learning Web Design. 2012 – 808 ст.

ДОДАТОК А

Back-end, лістинг коду

Файл controllers/analytics.js

```
const moment = require('moment')
const Order = require('../models/Order')
const errorHandler = require('../utils/errorHandler')

module.exports.overview = async function (req, res) {
  try {
    const allOrders = await Order.find({ user: req.user.id }).sort({ date: 1 })
    const ordersMap = getOrdersMap(allOrders)
    const yesterdayOrders = ordersMap[moment().add(-1,
'd').format(('DD.MM.YYYY'))] || []

    const totalOrdersNumber = allOrders.length

    const yesterdayOrdersNumber = yesterdayOrders.length

    const daysNumber = Object.keys(ordersMap).length

    const ordersPerDay = (totalOrdersNumber / daysNumber).toFixed(0)

    const ordersPercent = (((yesterdayOrdersNumber / ordersPerDay) - 1) *
100).toFixed(2)

    const totalGain = calculatePrice(allOrders)

    const gainPerDay = totalGain / daysNumber

    const yesterdayGain = calculatePrice(yesterdayOrders).toFixed(2)

    const gainPercent = (((yesterdayGain / gainPerDay) - 1) * 100).toFixed(2)

    const compareGain = (yesterdayGain - gainPerDay).toFixed(2)

    const compareNumber = (yesterdayOrdersNumber - ordersPerDay).toFixed(2)

    res.status(200).json({
      gain: {
        persent: Math.abs(+gainPercent),
        compare: Math.abs(+compareGain),
        yesterday: +yesterdayGain,
        isHigher: +gainPercent > 0
      },
      orders: {
        persent: Math.abs(+ordersPercent),
```

```
      compare: Math.abs(+compareNumber),
      yesterday: +yesterdayOrdersNumber,
      isHigher: +ordersPercent > 0
     }
    })
  } catch (e) {
   errorHandler(res, e)
  }
}

module.exports.analytics = async function (req, res) {
  try {
   const allOrders = await Order.find({ user: req.user.id }).sort({ date: 1 })
   const ordersMap = getOrdersMap(allOrders)

   const average = +(calculatePrice(allOrders) /
Object.keys(ordersMap).length).toFixed(2)

   const chart = Object.keys(ordersMap).map(label => {
    // label = '26.06.2019'
    const gain = +calculatePrice(ordersMap[label]).toFixed(2)
    const order = ordersMap[label].length
    return { label, gain, order }
   })
   res.status(200).json({average, chart})

  } catch (e) {
   errorHandler(res, e)
  }
}

function getOrdersMap(orders = []) {
  const dayOrders = {}
  orders.forEach(order => {
   const date = moment(order.date).format('DD.MM.YYYY')
   if (date === moment().format('DD.MM.YYYY')) {
    return
   }
   if (!dayOrders[date]) {
    dayOrders[date] = []
   }
   dayOrders[date].push(order)
  })

  return dayOrders
}
function calculatePrice(orders = []) {
  return orders.reduce((total, order) => {
   const orderPrice = order.list.reduce((orderTotal, item) => {
```

```
    return orderTotal += item.cost * item.quantity
   }, 0)
   return total += orderPrice
  }, 0)
}
```

Файл controllers/auth.js

```
const bcrypt = require('bcryptjs')
const jwt = require('jsonwebtoken')
const User = require('../models/User')
const keys = require('../config/keys')
const errorHandler = require('../utils/errorHandler')

module.exports.login = async function (req, res) {


 const candidate = await User.findOne({ email: req.body.email })

 if (candidate) {
  //сравнения паролей
  const passwordResult = bcrypt.compareSync(req.body.password,
candidate.password)
  if (passwordResult) {
   // пароли совпали, генерация токена

   const token = jwt.sign({
    email: candidate.email,
    userId: candidate._id,
   }, keys.jwt, { expiresIn: 604800 })

   res.status(200).json({
    token: `Bearer ${token}`
   })
  } else {
   //Пароли не совпали
   res.status(401).json({
    message: 'Пароли не совпали('
   })
  }
 } else {
  res.status(404).json({
   message: 'Пользователь с таким email не найден'
  })
 }
}

module.exports.register = async function (req, res) {
```

```
  const candidate = await User.findOne({ email: req.body.email })

  if (candidate) {
    res.status(409).json({
      message: 'Такой email уже есть'
    })
  } else {
    const salt = bcrypt.genSaltSync(10)
    const password = req.body.password
    const user = new User({
      email: req.body.email,
      password: bcrypt.hashSync(password, salt)
    })

    try {
      await user.save()
      res.status(201).json(user)
    } catch (e) {
      errorHandler(res, e)
    }

  }
}
```

Файл controllers/category.js

```
const Category = require('../models/Category')
const errorHandler = require('../utils/errorHandler')
const Position = require('../models/Position')
const aws = require('aws-sdk')
const keys = require('../config/keys')

const s3 = new aws.S3({
  secretAccessKey: keys.IAM_USER_SECRET,
  accessKeyId: keys.IAM_USER_KEY,
  region: 'eu-north-1'
})


module.exports.getAll = async function (req, res) {
  try {
    const categories = await Category.find({ user: req.user.id })
    res.status(200).json(categories)
  } catch (e) {
    errorHandler(res, e)
  }
}
```

```
module.exports.getById = async function (req, res) {
  try {
    const category = await Category.findById(req.params.id)
    res.status(200).json(category)
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.remove = async function (req, res) {
  try {
    const removedCategory = await Category.findOneAndDelete({ _id: req.params.id
})
    await Position.remove({ category: req.params.id })

    if (removedCategory.imageSrc) {
      const params = {
        Bucket: keys.BUCKET_NAME,
        Key: removedCategory.key,
      };
      await s3.deleteObject(params, (err, data)=> console.log(err, data))
    }

    res.status(200).json({
      message: 'Категория и товары в ней удалены'
    })
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.create = async function (req, res) {

  const category = new Category({
    name: req.body.name,
    user: req.user.id,
    key: req.file ? req.file.key : '',
    imageSrc: req.file ? req.file.location : ''
  })
  try {
    category.save()
    res.status(201).json(category)
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.update = async function (req, res) {
```

```
  let updated = {
    name: req.body.name,
  }
  if (req.file) {
    updated.key = req.file.key
    updated.imageSrc = req.file.location
  }
  try {

    const category = await Category.findOneAndUpdate(
      { _id: req.params.id },
      { $set: updated }
    )
    const params = {
      Bucket: keys.BUCKET_NAME,
      Key: category.key,
    };
    await s3.deleteObject(params, (err, data)=> console.log(err, data))

    res.status(200).json({message: 'Обновлено'})
  } catch (e) {
    errorHandler(res, e)
  }
}
```

Файл controllers/order.js

```
const Order = require('../models/Order')
const errorHandler = require('../utils/errorHandler')

module.exports.getAll = async function (req, res) {
  let query = {
    user: req.user.id
  }

  //дата старта
  if (req.query.start) {
    query.date = {
      //больше или равно
      $gte: req.query.start
    }
  }

  //дата конца
  if (req.query.end) {
    if (!query.date) {
      query.date = {}
    }
```

```
    //меньше или равно
    query.date['$lte'] = req.query.end
  }

  if(req.query.order){
    query.order = +req.query.order
  }

  try {
    const orders = await Order.find(query)
      .sort({ date: -1 })
      .skip(+req.query.offset)
      .limit(+req.query.limit)

    res.status(200).json(orders)
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.create = async function (req, res) {
  try {
    const lastOrder = await Order.findOne({ user: req.user.id })
      .sort({ date: -1 })

    const maxOrder = lastOrder ? lastOrder.order : 0

    const order = await new Order({
      list: req.body.list,
      user: req.user.id,
      order: maxOrder + 1
    }).save()
    res.status(201).json(order)
  } catch (e) {
    errorHandler(res, e)
  }
}
```

Файл controllers/position.js

```
const Position = require('../models/Position')
const errorHandler = require('../utils/errorHandler')

module.exports.getByCategoryId = async function (req, res) {
  try {
    const positions = await Position.find({
      category: req.params.categoryId,
      user: req.user.id
```

```
    })
    res.status(200).json(positions)
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.create = async function (req, res) {
  try {
    const position = await new Position({
      name: req.body.name,
      cost: req.body.cost,
      category: req.body.category,
      user: req.user.id
    }).save()
    res.status(201).json(position)
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.remove = async function (req, res) {
  try {
    await Position.remove({ _id: req.params.id })
    res.status(200).json({
      message: 'Позиция удалена'
    })
  } catch (e) {
    errorHandler(res, e)
  }
}

module.exports.update = async function (req, res) {
  try {
    const position = await Position.findOneAndUpdate(
      { _id: req.params.id },
      { $set: req.body },
      { new: true }
    )
    res.status(200).json(position)
  } catch (e) {
    errorHandler(res, e)
  }
}
```

Файл middleware/passport.js

```
const JwtStrategy = require('passport-jwt').Strategy
const ExtractJwt = require('passport-jwt').ExtractJwt
```

```
const User = require('../models/User')
const keys = require('../config/keys')


const opts = {
 jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
 secretOrKey: keys.jwt
}

module.exports = passport => {
 passport.use(new JwtStrategy(opts, async (jwt_payload, done) => {
  try {
   const user = await User.findById(jwt_payload.userId).select('email id')
   if (user) {
    return done(null, user)
   } else {
    return done(null, false)
   }
  } catch (e) {
   console.log(e);
  }
 }));
}
```

Файл middleware/uploads.js

```
const aws = require('aws-sdk')
const multer = require('multer')
const multerS3 = require('multer-s3')
const keys = require('../config/keys')


const s3 = new aws.S3({
 secretAccessKey: keys.IAM_USER_SECRET,
 accessKeyId: keys.IAM_USER_KEY,
 region: 'eu-north-1'
})

const upload = multer({
 storage: multerS3({
  s3: s3,
  bucket: keys.BUCKET_NAME,
  acl: 'public-read',
  fileFilter,
  metadata: function (req, file, cb) {
   cb(null, { fieldName: file.fieldname });
  },
  key: function (req, file, cb) {
   cb(null, `${Date.now()}-${file.originalname}`)
```

```
    }
  })
})
function fileFilter(req, file, cb) {
  if (file.size >= 5242880) {
    cb(null, false)
  }
  if (file.mimetype === 'image/jpeg' || file.mimetype === 'image/png') {
    cb(null, true)
  }
  cb(new Error('Тип файла долже быть jpeg или png, а размер меньше 5 мб'))

}
module.exports = upload
```

Файл models/Category.js

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const categotySchema = new Schema({
  name: {
    type: String,
    required: true
  },
  key: {
    type: String
  },
  imageSrc: {
    type: String,
    default: ''
  },
  user: {
    ref: 'users',
    type: Schema.Types.ObjectId
  }

})

module.exports = mongoose.model('categories', categotySchema)
```

Файл models/Order.js

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const ordersSchema = new Schema({
  date: {
```

```
    type: Date,
    default: Date.now
  },
  order: {
    type: Number,
    required: true
  },
  list: [{
    name: {
      type: String,
    },
    quantity: {
      type: Number
    },
    cost: {
      type: Number
    }
  }],
  user: {
    ref: 'users',
    type: Schema.Types.ObjectId
  }
})

module.exports = mongoose.model('orders', ordersSchema)
```

Файл models/Position.js

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const positionSchema = new Schema({
  name: {
    type: String,
    required: true
  },
  cost: {
    type: Number,
    required: true
  },
  category: {
    ref: 'categories',
    type: Schema.Types.ObjectId
  },
  user: {
    ref: 'users',
    type: Schema.Types.ObjectId
  }
```

```
})

module.exports = mongoose.model('positions', positionSchema)
```

Файл models/User.js

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const userSchema = new Schema({
  email: {
    type: String,
    required: true,
    unique: true
  },
  password: {
    type: String,
    required: true
  }
})

module.exports = mongoose.model('users', userSchema)
```

Файл routes/analytics.js

```
const express = require('express')
const passport = require('passport')
const router = express.Router()
const controller = require('../controllers/analytics')

router.get('/overview',  passport.authenticate('jwt', { session: false }),
controller.overview)

router.get('/analytics', passport.authenticate('jwt', { session: false }),
controller.analytics)

module.exports = router
```

Файл routes/auth.js

```
const express = require('express')
const router = express.Router()
const controller = require('../controllers/auth')

router.post('/login', controller.login)
```

```
router.post('/register', controller.register)

module.exports = router
```

Файл routes/category.js

```
const express = require('express')
const router = express.Router()
const passport = require('passport')
const upload = require('../middleware/uploads')
const controller = require('../controllers/category')

router.get('/', passport.authenticate('jwt', { session: false }), controller.getAll)
router.get('/:id', passport.authenticate('jwt', { session: false }), controller.getById)
router.delete('/:id', passport.authenticate('jwt', { session: false }),
controller.remove)
router.post('/', passport.authenticate('jwt', { session: false }),
upload.single('image'), controller.create)
router.patch('/:id', passport.authenticate('jwt', { session: false }),
upload.single('image'), controller.update)

module.exports = router
```

Файл routes/order.js

```
const express = require('express')
const router = express.Router()
const passport = require('passport')
const controller = require('../controllers/order')

router.get('/', passport.authenticate('jwt', { session: false }), controller.getAll)
router.post('/', passport.authenticate('jwt', { session: false }), controller.create)

module.exports = router
```

Файл routes/position.js

```
const express = require('express')
const router = express.Router()
const passport = require('passport')
const controller = require('../controllers/position')

router.get('/:categoryId', passport.authenticate('jwt', { session: false }),
controller.getByCategoryId)
router.post('/', passport.authenticate('jwt', { session: false }), controller.create)
```

```
router.patch('/:id', passport.authenticate('jwt', { session: false }),
controller.update)
router.delete('/:id', passport.authenticate('jwt', { session: false }),
controller.remove)

module.exports = router
```

Файл utils/errorHandler.js

```
module.exports = (res, error) =>{
  res.status(500).json({
    success: false,
    message: error.message? error.message : error
  })
}
```

Файл app.js

```
const express = require('express')
const app = express()
const mongoose = require('mongoose')
const passport = require('passport')
const bodyParcer = require('body-parser')
const path = require('path')
const authRoutes = require('./routes/auth')
const analyticsRoutes = require('./routes/analytics')
const categoryRoutes = require('./routes/category')
const orderRoutes = require('./routes/order')
const positionRoutes = require('./routes/position')
const keys = require('./config/keys')
const mongoURL = 'mongodb://localhost:27017/fullstackCRM'


mongoose.connect(keys.mongoURI)
  .then(() => console.log('mongoDB connected'))
  .catch(err => console.log(err))


app.use(passport.initialize())
require('./middleware/passport')(passport)
app.use(require('morgan')('dev'))
app.use('/uploads', express.static('uploads'))
app.use(bodyParcer.urlencoded(({ extended: true })));
app.use(bodyParcer.json())
app.use(require('cors')())
```

```
app.use('/api/auth', authRoutes)
app.use('/api/analytics', analyticsRoutes)
app.use('/api/category', categoryRoutes)
app.use('/api/order', orderRoutes)
app.use('/api/position', positionRoutes)

if (process.env.NODE_ENV === 'production') {
  app.use(express.static('./client/dist/client'))

  app.get('*', (req, res) => {
    res.sendFile(path.resolve(
      __dirname, 'client', 'dist', 'client', 'index.html'
    ))
  })
}


module.exports = app
```

Файл index.js

```
const app = require('./app')
const port = process.env.PORT || 3001
app.listen(port, () => console.log(`Node go, port ${port}`))
```

ДОДАТОК Б

Front-end, лістинг коду

Файл analtics.service.ts

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { OverviewPage, AnalyticsPage } from 'src/app/shared/interfaces';

@Injectable()
export class AnalyticsService {

 constructor(
  private http: HttpClient
 ) { }
 getOverview(): Observable<OverviewPage> {
  return this.http.get<OverviewPage>('/api/analytics/overview')
 }

 getAnalytics(): Observable<AnalyticsPage>{
  return this.http.get<AnalyticsPage>('/api/analytics/analytics')
 }
}
```

Файл categories.service.ts

```
import { Category, Message } from '../../shared/interfaces';
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';


@Injectable()
export class CategoriesService {

 constructor(private http: HttpClient){
 }

 fetch(): Observable<Category[]>{
  return this.http.get<Category[]>('/api/category')
 }

 getCategoryById(id: string): Observable<Category>{
  return this.http.get<Category>(`/api/category/${id}`)
 }
```

```
  create(name: string, image?: File): Observable<Category>{
    const fd = new FormData()
    if(image){
      fd.append('image', image, image.name)
    }
    fd.append('name', name)

    return this.http.post<Category>('/api/category', fd)
  }

  update(id: string, name: string, image?: File): Observable<Message>{
    const fd = new FormData()
    if(image){
      fd.append('image', image, image.name)
    }
    fd.append('name', name)

    return this.http.patch<Message>(`/api/category/${id}`, fd)
  }

  delete(id: string): Observable<Message>{
    return this.http.delete<Message>(`/api/category/${id}`)
  }
}
```

Файл orders.service.ts

```
import { Order } from './../../shared/interfaces';
import { HttpClient, HttpParams } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';

@Injectable()
export class OrdersService {

  constructor(private http: HttpClient) { }

  create(order: Order): Observable<Order> {
    return this.http.post<Order>('/api/order', order)
  }
  fetch(params: any = {}): Observable<Order[]> {
    return this.http.get<Order[]>('/api/order',{
      params: new HttpParams({
        fromObject: params
      })
    })
  }
```

```
}
```

Файл position.service.ts

```
import { Position, Message } from '../../shared/interfaces';
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';

@Injectable()
export class PositionsService {

 constructor(
  private http: HttpClient
 ){}

 fetch(categoryId: string): Observable<Position[]>{
  return this.http.get<Position[]>(`/api/position/${categoryId}`)
 }
 create(position: Position): Observable<Position>{
  return this.http.post<Position>('/api/position', position)
 }
 update(position: Position): Observable<Position>{
  return this.http.patch<Position>(`/api/position/${position._id}`, position)
 }
 delete(positionId: string):Observable<Message>{
  return this.http.delete<Message>(`/api/position/${positionId}`)
 }
}
```

Файл core.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { TokenInterceptor } from '../shared/Classes/token.interceptor';

@NgModule({
 declarations: [],
 imports: [
  CommonModule
 ],
 providers: [
  {
   provide: HTTP_INTERCEPTORS,
   multi: true,
```

```
    useClass: TokenInterceptor
   }
  ]
})
export class CoreModule { }
```

Файл login-page.component.ts

```
import { Component, OnInit, OnDestroy } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms';
import { Router, ActivatedRoute, Params } from '@angular/router';
import { Subscription } from 'rxjs';
import { AuthService } from '../../shared/services/auth.service';
import { MaterialService } from 'src/app/shared/Classes/material.service';

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

 aSub: Subscription = null

 form: FormGroup

 constructor(
  private auth: AuthService,
  private router: Router,
  private route: ActivatedRoute
 ) {
 }

 ngOnInit() {
  this.form = new FormGroup({
   email: new FormControl(null, [Validators.required, Validators.email]),
   password: new FormControl(null, [Validators.required,
Validators.minLength(6)])
  })

  this.route.queryParamMap.subscribe(({ params }: Params) => {

   if (params['registered']) {
    //message Вы зарегестрированы
    MaterialService.toast('Теперь вы зарегестрированы, войдите')
   }
   if (params['accessDenied']) {
    //message Сначала войдите
```

```
      MaterialService.toast('Сначала войдите')
    }
    if (params['sessionFailed']) {
      MaterialService.toast('Войдите заново')
    }
  })
 }

 ngOnDestroy() {
  if (this.aSub) {
   this.aSub.unsubscribe()
  }
 }

 onSubmit() {
  this.form.disable()

  this.aSub = this.auth.login(this.form.value).subscribe(
   res => {
    this.router.navigate(['/overview'])
   },
   e => {
    //Вывод ошибки
    MaterialService.toast(e.error.message)
    this.form.enable()
   }
  )
 }

}
```

Файл register.component.ts

```
import { MaterialService } from './../../shared/Classes/material.service';
import { Subscription } from 'rxjs';
import { AuthService } from '../../shared/services/auth.service';
import { Router } from '@angular/router';
import { FormGroup, FormControl, Validators } from '@angular/forms';
import { Component, OnInit, OnDestroy } from '@angular/core';

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

 form: FormGroup
```

```
  aSub: Subscription

  constructor(
    private auth: AuthService,
    private router: Router,
  ) { }

  ngOnInit() {
    this.form = new FormGroup({
      email: new FormControl(null, [Validators.required, Validators.email]),
      password: new FormControl(null, [Validators.required,
Validators.minLength(6)])
    })
  }
  ngOnDestroy(){
    if (this.aSub) {
      this.aSub.unsubscribe()
    }
  }
  onSubmit() {
    this.form.disable()

    this.aSub = this.auth.register(this.form.value).subscribe(
      () => {
        this.router.navigate(['/login'], {
          queryParams: {
            registered: true
          }
        })
      },
      e => {
        //Вывод ошибки
        MaterialService.toast(e.error.message)
        this.form.enable()

      }
    )
  }

}
```

Файл login.module.ts

```
import { SharedModule } from './../shared/shared.module';
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { LoginPageComponent } from './login-page/login-page.component';
```

```
import { AuthLayoutComponent } from './auth-layout/auth-layout.component';
import { RegisterComponent } from './register/register.component';
import { AuthService } from '../shared/services/auth.service';

@NgModule({
  declarations: [
    LoginPageComponent,
    AuthLayoutComponent,
    RegisterComponent
  ],
  imports: [
    CommonModule,
    SharedModule
  ],
  providers: [AuthService]
})
export class LoginModule { }
```

Файл auth.guard.ts

```
import { Injectable } from '@angular/core';
import { CanActivate, CanActivateChild, ActivatedRouteSnapshot,
RouterStateSnapshot, Router } from '@angular/router';
import { Observable, of } from 'rxjs';
import { AuthService } from '../services/auth.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate, CanActivateChild {

  constructor(
    private auth: AuthService,
    private router: Router
  ) { }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean> {
    if (this.auth.isLogined()) {
      return of(true)
    } else {
      this.router.navigate(['/login'], {
        queryParams: {
          accessDenied: true
        }
      })
      return of(false)
    }
```

```
  }
  canActivateChild(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean> {
    return this.canActivate(route, state)
  }
}
```

Файл material.service.ts

```
import { ElementRef } from '@angular/core';

declare var M

export interface MaterialInstance {
  open?(): void,
  close?(): void,
  destroy?(): void,
}

export interface MaterialDatepicker extends MaterialInstance{
  date?: Date
}

export class MaterialService {

  static toast(message: string) {
    M.toast({ html: message })
  }

  static initFloatingButton(elem: ElementRef) {
    M.FloatingActionButton.init(elem.nativeElement);
  }

  static updateInputs() {
    M.updateTextFields()
  }
  static initModal(ref: ElementRef): MaterialInstance {
    return M.Modal.init(ref.nativeElement)
  }

  static initTooltip(ref: ElementRef): MaterialInstance {
    return M.Tooltip.init(ref.nativeElement)
  }
  static initDatepicker(ref: ElementRef, onClose: () => void): MaterialDatepicker {
    return M.Datepicker.init(ref.nativeElement, {
      format: 'dd.mm.yyyy',
      showClearBtn: true,
      onClose
```

```
    })
  }
  static initTaptarget(ref: ElementRef): MaterialInstance {
    return M.TapTarget.init(ref.nativeElement)
  }
}
```

Файл token.interceptor.ts

```
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent, HttpErrorResponse
} from '@angular/common/http';
import { AuthService } from '../services/auth.service';
import { Observable, throwError } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { Router } from '@angular/router';

@Injectable()
export class TokenInterceptor implements HttpInterceptor {

  constructor(
    private auth: AuthService,
    private router: Router
    ) { }

  intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
    if (this.auth.isLogined()) {
      req = req.clone({
        setHeaders: {
          Authorization: this.auth.getToken()
        }
      })
    }
    return next.handle(req).pipe(
      catchError((err: HttpErrorResponse) => this.handleAuthError(err))
    )
  }


  private handleAuthError(error: HttpErrorResponse): Observable<any> {
    if (error.status === 401) {
      this.router.navigate(['/login'])
    }
    return throwError(error)
  }
}
```

Файл auth.service.ts

```
import { Injectable } from '@angular/core';
import { User } from '../interfaces';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { tap } from 'rxjs/operators';

@Injectable()
export class AuthService {

  private token: string = null

  constructor(private http: HttpClient) {
  }

  login(user: User): Observable<{ token: string }> {
    return this.http.post<{ token: string }>('/api/auth/login', user).pipe(
      tap(({ token }) => {
        localStorage.setItem('auth-token', token)
        this.setToken(token)
      })
    )

  }

  register(user: User): Observable<User> {
    return this.http.post<User>('/api/auth/register', user)
  }

  setToken(token: string) {
    this.token = token
  }

  getToken(): string {
    return this.token
  }

  isLogined(): boolean {
    return !!this.token
  }

  logout() {
    this.setToken(null)
    localStorage.clear()
  }

}
```

Файл interfaces.ts

```
export interface User {
  email: string
  password: string
}
export interface Category {
  name: string
  imageSrc?: string
  user?: string
  _id?: string
}
export interface Message {
  message: string
}
export interface Position {
  name: string
  cost: number
  category: string
  user?: string
  _id?: string
  quanyity?: number
}
export interface Order {
  list: OrderPosition[]
  date?: Date
  order?: number
  user?: string
  _id?: string
}

export interface OrderPosition {
  name: string
  cost: number
  quantity: number
  _id?: string
}
export interface Filter {
  start?: Date
  end?: Date
  order?: Number
}
export interface OverviewPage {
  gain: OverviewPageItem
  orders: OverviewPageItem
}

export interface OverviewPageItem {
  percent: number
```

```
  compare: number
  yesterday: number
  isHegher: boolean
}

export interface AnalyticsPage {
  average: number,
  chart: AnalyticsChartItem[]
}
export interface AnalyticsChartItem {
  gain: number
  order: number
  label: string
}
export const BACK_END = 'https://peaceful-beach-47546.herokuapp.com'
```

Файл shared.module.ts

```
import { LoaderComponent } from './components/loader/loader.component';
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HttpClientModule } from '@angular/common/http';
import { ReactiveFormsModule, FormsModule } from '@angular/forms';
import { AppRoutingModule } from './../app-routing.module';


@NgModule({
  declarations: [
    LoaderComponent
  ],
  imports: [
    CommonModule,
    AppRoutingModule,
    FormsModule,
    ReactiveFormsModule,
    HttpClientModule
  ],
  exports: [
    AppRoutingModule,
    FormsModule,
    ReactiveFormsModule,
    HttpClientModule,
    LoaderComponent
  ]
})
export class SharedModule { }
```

Файл analytics-page.component.ts

```
import { Component, OnInit, ViewChild, ElementRef, AfterViewInit, OnDestroy }
from '@angular/core';
import { untilDestroyed } from 'ngx-take-until-destroy';
import { AnalyticsService } from './../../core/services/analtics.service';
import { AnalyticsPage } from 'src/app/shared/interfaces';
import { Chart } from 'chart.js'
@Component({
 selector: 'app-analytics-page',
 templateUrl: './analytics-page.component.html',
 styleUrls: ['./analytics-page.component.scss']
})
export class AnalyticsPageComponent implements AfterViewInit, OnDestroy {

 @ViewChild('gain') gainRef: ElementRef
 @ViewChild('order') orderRef: ElementRef

 average: number
 pending = true

 constructor(private service: AnalyticsService) { }

 ngAfterViewInit() {
  let gainConfig: any = {
   label: 'Выручка',
   color: 'rgb(255,99,132)'
  }
  let orderConfig: any = {
   label: 'Заказы',
   color: 'rgb(54,162,235)'
  }
  this.service.getAnalytics().pipe(untilDestroyed(this)).subscribe((data:
AnalyticsPage) => {
    gainConfig.labels = data.chart.map(item => item.label)
    gainConfig.data = data.chart.map(item => item.gain)

    orderConfig.labels = data.chart.map(item => item.label)
    orderConfig.data = data.chart.map(item => item.order)

    this.average = data.average

    let gainContext = this.gainRef.nativeElement.getContext('2d')
    let orderContext = this.orderRef.nativeElement.getContext('2d')

    gainContext.canvas.height = '300px'
    orderContext.canvas.height = '300px'

    new Chart(gainContext, chreateChartConfig(gainConfig))
    new Chart(orderContext, chreateChartConfig(orderConfig))
```

```
      this.pending = false
    })
  }

  ngOnDestroy() {
    // Нужен для untilDestroyed
  }
}
function chreateChartConfig({labels, data, label, color}) {
  return {
    type: 'line',
    options: {
      responsive: true
    },
    data: {
      labels,
      datasets: [
        {
          label, data,
          borderColor: color,
          steppedLine: false,
          fill: false
        }
      ]
    }
  }
}
```

Файл categories-form.component.ts

```
import { Category, Message } from './../../../shared/interfaces';
import { MaterialService } from './../../../shared/Classes/material.service';
import { CategoriesService } from '../../../core/services/categories.service';
import { FormGroup, FormControl, Validators } from '@angular/forms';
import { ActivatedRoute, Params, Router } from '@angular/router';
import { Component, OnInit, ViewChild, ElementRef, OnDestroy } from
'@angular/core';
import { switchMap, takeUntil } from 'rxjs/operators';
import { of, Observable, Subject } from 'rxjs';


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

```
private ngUnsubscribe = new Subject();

@ViewChild('input') inputRef: ElementRef
image: File
isNew = true
imagePreview: string | ArrayBuffer = ''
form: FormGroup
category: Category
constructor(
  private route: ActivatedRoute,
  private catServ: CategoriesService,
  private router: Router
) { }

ngOnInit() {
  this.form = new FormGroup({
    name: new FormControl(null, Validators.required)
  })
  this.form.disable()
  this.route.params.pipe(
    switchMap((params: Params) => {
      if (params['id']) {
        this.isNew = false;
        return this.catServ.getCategoryById(params['id'])
      }
      return of(null)
    }),
    takeUntil(this.ngUnsubscribe)
  ).subscribe((category: Category) => {
    if (category) {
      this.form.patchValue({
        name: category.name,
      })
      this.category = category
      this.imagePreview = category.imageSrc
      MaterialService.updateInputs()
    }
    this.form.enable()
  },
    err => MaterialService.toast(err.error.message)
  )

}

ngOnDestroy() {
  this.ngUnsubscribe.next();
  this.ngUnsubscribe.complete();
}
```

```
  triggerClick() {
    this.inputRef.nativeElement.click()
  }

  onFileUpload(event: any) {

    this.image = event.target.files[0]

    const reader = new FileReader()

    reader.onload = () => {
      this.imagePreview = reader.result
    }

    reader.readAsDataURL(this.image)

  }

  onSubmit() {
    this.form.disable()
    if (this.isNew) {
      this.catServ.create(this.form.value.name, this.image).pipe
        (takeUntil(this.ngUnsubscribe))
        .subscribe(category => {
          this.category = category
          MaterialService.toast('Сохранено')
        }, err => {

          MaterialService.toast(err.error.message)
        }, () => this.form.enable())
    } else {

      this.catServ.update(this.category._id, this.form.value.name, this.image).pipe
      (takeUntil(this.ngUnsubscribe))
      .subscribe(message => {
        MaterialService.toast('Сохранено')
      }, err => {

        MaterialService.toast(err.error.message)
      }, () => this.form.enable())
    }

  }

  deleteCategory() {
    const desidion = window.confirm(`Вы точно хотите удалить категорию
${this.category.name}`)
    if (desidion) {
      this.catServ.delete(this.category._id).pipe(
```

```
      takeUntil(this.ngUnsubscribe)
    )
      .subscribe(
        res => MaterialService.toast(res.message),
        err => MaterialService.toast(err.error.message),
        () => this.router.navigate(['/categories'])
      )
  }


 }


}
```

Файл positions-form.component.ts

```
import { Position } from './../../../../shared/interfaces';
import { FormGroup, FormControl, Validators } from '@angular/forms';
import { MaterialService, MaterialInstance } from
'src/app/shared/Classes/material.service';
import { Component, OnInit, Input, ViewChild, ElementRef, AfterViewInit,
OnDestroy } from '@angular/core';
import { PositionsService } from '../../../../core/services/position.service';
import { Subject } from 'rxjs';
import { takeUntil } from 'rxjs/operators';

@Component({
 selector: 'app-positions-form',
 templateUrl: './positions-form.component.html',
 styleUrls: ['./positions-form.component.scss']
})
export class PositionsFormComponent implements OnInit, AfterViewInit, OnDestroy
{
 @Input('category') categoryId: string
 @ViewChild('modal') modalRef: ElementRef

 private ngUnsubscribe = new Subject();



 loading = false
 positions: Position[] = []
 modal: MaterialInstance
 form: FormGroup
 positionId = null
```

```
constructor(
  private posServ: PositionsService
) { }

ngOnInit() {
  this.form = new FormGroup({
    name: new FormControl(null, Validators.required),
    cost: new FormControl(null, [Validators.required, Validators.min(0.01)])
  })

  this.loading = true
  this.posServ.fetch(this.categoryId).pipe(
    takeUntil(this.ngUnsubscribe)
  ).subscribe(positions => {
    this.positions = positions
    this.loading = false
  })
}
ngAfterViewInit() {
  this.modal = MaterialService.initModal(this.modalRef)
}
ngOnDestroy() {
  this.modal.destroy()
  this.ngUnsubscribe.next();
  this.ngUnsubscribe.complete();
}
onSelectPosition(position: Position) {
  this.positionId = position._id

  this.form.patchValue({
    name: position.name,
    cost: position.cost
  })
  this.modal.open()
  MaterialService.updateInputs()
}
onAddPosition() {
  this.positionId = null
  this.form.reset()
  this.modal.open()
}
onCancel() {
  this.modal.close()
}
onSubmit() {
  this.form.disable()

  let position: Position = {
    name: this.form.value.name,
```

```
    cost: this.form.value.cost,
    category: this.categoryId,
  }

  const complete = () => {
    this.form.enable()
    this.modal.close()
    this.form.reset({ name: null, cost: null })
  }

  if (this.positionId) {
    position._id = this.positionId
    this.posServ.update(position).pipe(
      takeUntil(this.ngUnsubscribe)
    ).subscribe(
      pos => {
        const index = this.positions.findIndex(p => p._id == pos._id)
        MaterialService.toast('Товар изменен')
        this.positions[index] = pos
      }, err => {

        MaterialService.toast(err.error.message)
      },
      complete
    )
  } else {

    this.posServ.create(position).pipe(
      takeUntil(this.ngUnsubscribe)
    ).subscribe(
      pos => {
        MaterialService.toast('Товар создан')
        this.positions.push(pos)
      }, err => {

        MaterialService.toast(err.error.message)
      },
      complete
    )
  }


}
onDeletePosition(e: Event, position: Position) {
  e.stopPropagation()
  const desidion = window.confirm(`Вы точно хотите удалить ${position.name}`)
  if (desidion) {
    this.posServ.delete(position._id).pipe(
      takeUntil(this.ngUnsubscribe)
```

```
    ).subscribe(res => {
      const index = this.positions.findIndex(p => p._id == position._id)
      this.positions.splice(index, 1)
      MaterialService.toast(res.message)
    }, err => MaterialService.toast(err.error.message))
  }
 }

}
```

Файл history-page.component.ts

```
import { Component, OnInit, ViewChild, ElementRef, OnDestroy, AfterViewInit }
from '@angular/core';
import { Subscription } from 'rxjs';
import { MaterialService, MaterialInstance } from
'src/app/shared/Classes/material.service';
import { OrdersService } from 'src/app/core/services/orders.service';
import { Order, Filter } from './../../shared/interfaces';

const STEP = 3

@Component({
 selector: 'app-history-page',
 templateUrl: './history-page.component.html',
 styleUrls: ['./history-page.component.scss']
})
export class HistoryPageComponent implements OnInit, OnDestroy, AfterViewInit {

 @ViewChild('tooltip') tooltipRef: ElementRef

 filter: Filter = {}
 tooltip: MaterialInstance
 offset = 0
 limit = STEP
 oSub: Subscription
 orders: Order[] = []

 isFilter = false
 reloading = false
 loading = false
 noMoreOrders = false

 constructor(
  private orderServ: OrdersService
 ) { }

 ngOnInit() {
```

```
    this.reloading = true
    this.fetch()
  }
  ngAfterViewInit() {
    this.tooltip = MaterialService.initTooltip(this.tooltipRef)
  }
  ngOnDestroy() {
    this.tooltip.destroy()
    if (this.oSub)
      this.oSub.unsubscribe()
  }

  fetch() {
    const params =  Object.assign({}, this.filter, {
      offset: this.offset,
      limit: this.limit
    })

    this.oSub = this.orderServ.fetch(params).subscribe(orders => {
      this.loading = false
      this.reloading = false
      this.noMoreOrders = orders.length < STEP
      this.orders.push(...orders)
    })
  }
  loadMore() {
    this.loading = true
    this.offset += STEP
    this.fetch()
  }
  apllyFilter(filter: Filter) {
    this.orders = []
    this.offset = 0
    this.filter = filter
    this.reloading = true
    this.fetch()
  }

  isFiltered(): boolean{
    return Object.keys(this.filter).length != 0
  }
}
```

Файл categories-page.component.ts

```
import { Observable } from 'rxjs';
import { Category } from './../../shared/interfaces';
import { CategoriesService } from '../../core/services/categories.service';
```

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

@Component({
  selector: 'app-categories-page',
  templateUrl: './categories-page.component.html',
  styleUrls: ['./categories-page.component.scss']
})
export class CategoriesPageComponent {

  categories$: Observable <Category[]>

  constructor(private catServ: CategoriesService) { }

  ngOnInit() {
    this.categories$ = this.catServ.fetch()
  }

}
```

Файл history-list.component.ts

```
import { Order } from './../../../shared/interfaces';
import { Component, OnInit, Input, ViewChild, ElementRef, OnDestroy,
AfterViewInit } from '@angular/core';
import { MaterialInstance, MaterialService } from
'src/app/shared/Classes/material.service';

@Component({
  selector: 'app-history-list',
  templateUrl: './history-list.component.html',
  styleUrls: ['./history-list.component.scss']
})
export class HistoryListComponent implements  OnDestroy, AfterViewInit {

  @Input('orders') orders: Order[]
  @ViewChild('modal') modalRef: ElementRef

  modal: MaterialInstance
  selectedOrder: Order

  constructor() { }

  ngAfterViewInit(){
    this.modal = MaterialService.initModal(this.modalRef)
  }
  ngOnDestroy() {
    this.modal.destroy()
  }
```

```
  computePrice(order: Order): Number {
    return order.list.reduce((total, item) => {
      return total += item.quantity * item.cost
    }, 0)
  }

  selectOrder(order: Order){
    this.selectedOrder = order
    this.modal.open()
  }
  closeModal(){
    this.modal.close()
  }
}
```

Файл history-filter.component.ts

```
import { MaterialService, MaterialDatepicker } from
'src/app/shared/Classes/material.service';
import { Component, OnInit, EventEmitter, Output, OnDestroy, AfterViewInit,
ViewChild, ElementRef } from '@angular/core';
import { Filter } from 'src/app/shared/interfaces';

@Component({
  selector: 'app-history-filter',
  templateUrl: './history-filter.component.html',
  styleUrls: ['./history-filter.component.scss']
})
export class HistoryFilterComponent implements OnDestroy, AfterViewInit {

  @Output() onFilter = new EventEmitter<Filter>();
  @ViewChild('start') startRef: ElementRef
  @ViewChild('end') endRef: ElementRef

  order: Number
  start: MaterialDatepicker
  end: MaterialDatepicker
  isValid = true

  constructor() { }


  ngAfterViewInit() {
    this.start = MaterialService.initDatepicker(this.startRef, this.validate.bind(this))
    this.end = MaterialService.initDatepicker(this.endRef, this.validate.bind(this))
  }
  ngOnDestroy() {
```

```
    this.start.destroy()
    this.end.destroy()
  }

  validate() {
    if (!this.start.date || !this.end.date) {
      this.isValid = true
      return
    }
    this.isValid = this.start.date < this.end.date
  }
  submitFilter() {

    let filter: Filter = {}

    if (this.order) {
      filter.order = this.order
    }
    if (this.start.date) {
      filter.start = this.start.date
    }
    if (this.end.date) {
      filter.end = this.end.date
    }

    this.onFilter.emit(filter)
  }

}
```

Файл order-category.component.ts

```
import { Category } from '../../../shared/interfaces';
import { CategoriesService } from '../../../core/services/categories.service';
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';

@Component({
  selector: 'app-order-category',
  templateUrl: './order-category.component.html',
  styleUrls: ['./order-category.component.scss']
})
export class OrderCategoryComponent implements OnInit {

  categories$: Observable<Category[]>

  constructor(private catServ: CategoriesService) { }
```

```
  ngOnInit() {
    this.categories$ = this.catServ.fetch()
  }

}
```

Файл order-positions.component.ts

```
import { MaterialService } from './../../../shared/Classes/material.service';
import { switchMap, map } from 'rxjs/operators';
import { Position } from './../../../shared/interfaces';
import { PositionsService } from './../../../core/services/position.service';
import { ActivatedRoute, Params } from '@angular/router';
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { OrderService } from '../order.service';

@Component({
  selector: 'app-order-positions',
  templateUrl: './order-positions.component.html',
  styleUrls: ['./order-positions.component.scss']
})
export class OrderPositionsComponent implements OnInit {

  positions$: Observable<Position[]>

  constructor(
    private route: ActivatedRoute,
    private posServ: PositionsService,
    private orderServ: OrderService
  ) { }

  ngOnInit() {
    this.positions$ = this.route.params.pipe(
      switchMap((params: Params) => this.posServ.fetch(params['id'])),
      map( (positions: Position[]) => {
        return positions.map( position =>{
          position.quanyity = 1
          return position
        })
      })
    )

  }
  addToOrder(position: Position){
    MaterialService.toast(`Добавлено x${position.quanyity}`)
    this.orderServ.add(position)
  }
```

```
}
```

Файл order.service.ts

```
import { Position, OrderPosition } from './../../shared/interfaces';
import { Injectable } from '@angular/core';

@Injectable()
export class OrderService {

  list: OrderPosition[] = []
  price = 0


  add(position: Position) {
    const orderPosition: OrderPosition = Object.assign({}, {
      name: position.name,
      cost: position.cost,
      quantity: position.quanyity,
      _id: position._id
    })

    const candidate = this.list.find(p => p._id == position._id)
    if (candidate){
      candidate.quantity += position.quanyity
    } else{
      this.list.push(orderPosition)
    }
    this.computePrice()
  }

  delete(orderPosition: OrderPosition) {
    const inx = this.list.findIndex( p => p._id == orderPosition._id)
    this.list.splice(inx, 1)
    this.computePrice()
  }

  clear() {
    this.list = []
    this.price = 0
  }


  private computePrice(){
    this.price = this.list.reduce( (total, item) => {
      return total += item.quantity * item.cost
    }, 0)
```

```
  }
}
```

Файл order-page.component.ts

```
import { takeUntil } from 'rxjs/operators';
import { Subject } from 'rxjs';
import { Component, OnInit, OnDestroy, ViewChild, ElementRef, AfterViewInit }
from '@angular/core';
import { Router, NavigationEnd } from '@angular/router';
import { OrderService } from './order.service';
import { OrdersService } from 'src/app/core/services/orders.service';
import { OrderPosition, Order } from './../../shared/interfaces';
import { MaterialInstance } from 'src/app/shared/Classes/material.service';
import { MaterialService } from './../../shared/Classes/material.service';

@Component({
  selector: 'app-order-page',
  templateUrl: './order-page.component.html',
  styleUrls: ['./order-page.component.scss']
})
export class OrderPageComponent implements OnInit, AfterViewInit, OnDestroy {
  @ViewChild('modal') modalRef: ElementRef

  isRoot: boolean = true
  modal: MaterialInstance
  pending = false
  private ngUnsubscribe = new Subject();

  constructor(
    private router: Router,
    private orderServ: OrderService,
    private ordersBackServices: OrdersService
  ) { }

  ngOnInit() {
    this.isRoot = this.router.url == '/order'
    this.router.events.pipe(takeUntil(this.ngUnsubscribe))
      .subscribe(event => {
        if (event instanceof NavigationEnd) {
          this.isRoot = this.router.url == '/order'
        }

      })
  }
  ngAfterViewInit() {
    this.modal = MaterialService.initModal(this.modalRef)
  }
```

```
  ngOnDestroy() {
    this.ngUnsubscribe.next();
    this.ngUnsubscribe.complete();
    this.modal.destroy()
  }

  open() {
    this.modal.open()
  }
  cencel() {
    this.modal.close()
  }
  submit() {
    this.pending = true
    const order: Order = {
      list: this.orderServ.list.map(item => {
        delete item._id
        return item
      })
    }

    this.ordersBackServices.create(order)
      .pipe(takeUntil(this.ngUnsubscribe))
      .subscribe(
        newOrder => {
          MaterialService.toast(`Заказ №${newOrder.order} был добавлен`)
          this.orderServ.clear()
        },
        err => MaterialService.toast(err.error.message),
        () => {
          this.modal.close()
          this.pending = false
        }
      )
  }
  deletePosition(orderPosition: OrderPosition) {
    this.orderServ.delete(orderPosition)
  }
}
```

Файл overview-page.component.ts

```
import { MaterialService } from 'src/app/shared/Classes/material.service';
import { MaterialInstance } from './../../shared/Classes/material.service';
import { Component, OnInit, ViewChild, ElementRef, OnDestroy, AfterViewInit }
from '@angular/core';
import { AnalyticsService } from 'src/app/core/services/analtics.service';
import { Observable } from 'rxjs';
```

```
import { OverviewPage } from 'src/app/shared/interfaces';

@Component({
 selector: 'app-overview-page',
 templateUrl: './overview-page.component.html',
 styleUrls: ['./overview-page.component.scss']
})
export class OverviewPageComponent implements OnInit, AfterViewInit,
OnDestroy {

 @ViewChild('tapTarget') tapTargetRef: ElementRef

 data$: Observable<OverviewPage>
 tapTarget: MaterialInstance
 yesterday = new Date()

 constructor(private service: AnalyticsService) { }

 ngOnInit() {
  this.yesterday.setDate(this.yesterday.getDate() - 1)
  this.data$ = this.service.getOverview()
 }
 ngAfterViewInit(){
  this.tapTarget = MaterialService.initTaptarget(this.tapTargetRef)
 }

 ngOnDestroy(){
  this.tapTarget.destroy()
 }
 openInfo(){
  this.tapTarget.open()
 }

}
```

Файл site-layout.component.ts

```
import { MaterialService } from 'src/app/shared/Classes/material.service';
import { AuthService } from './../../shared/services/auth.service';
import { Component, OnInit, AfterViewInit, ViewChild, ElementRef } from
'@angular/core';
import { Router } from '@angular/router';

@Component({
 selector: 'app-site-layout',
 templateUrl: './site-layout.component.html',
 styleUrls: ['./site-layout.component.scss']
})
```

```
export class SiteLayoutComponent implements AfterViewInit {
 @ViewChild('floatingButton') floatingRef: ElementRef

 links = [
  { url: '/overview', name: 'Обзор' },
  { url: '/analytics', name: 'Аналитика' },
  { url: '/history', name: 'История' },
  { url: '/order', name: 'Добавить заказ' },
  { url: '/categories', name: 'Ассортимент' }
 ]


 constructor(
  private auth: AuthService,
  private router: Router
 ) { }

 ngAfterViewInit(){
  MaterialService.initFloatingButton(this.floatingRef)
 }

 logout(){
  this.auth.logout()
  this.router.navigate(['/login'])
 }

}
```

Файл site.module.ts

```
import { AnalyticsService } from 'src/app/core/services/analtics.service';
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { SiteLayoutComponent } from './site-layout/site-layout.component';
import { OverviewPageComponent } from './overview-page/overview-
page.component';
import { AnalyticsPageComponent } from './analytics-page/analytics-
page.component';
import { HistoryPageComponent } from './history-page/history-page.component';
import { OrderPageComponent } from './order-page/order-page.component';
import { CategoriesPageComponent } from './categories-page/categories-
page.component';
import { CategoriesService } from '../core/services/categories.service';
import { CategoriesFormComponent } from './categories-page/categories-
form/categories-form.component';
import { PositionsFormComponent } from './categories-page/categories-
form/positions-form/positions-form.component';
import { PositionsService } from '../core/services/position.service';
```

```
import { OrderPositionsComponent } from './order-page/order-positions/order-positions.component';
import { OrderCategoryComponent } from './order-page/order-category/order-category.component';
import { SharedModule } from './../shared/shared.module';
import { OrderService } from './order-page/order.service';
import { OrdersService } from '../core/services/orders.service';
import { HistoryListComponent } from './history-page/history-list/history-list.component';
import { HistoryFilterComponent } from './history-page/history-filter/history-filter.component';

@NgModule({
 declarations: [
  SiteLayoutComponent,
  OverviewPageComponent,
  AnalyticsPageComponent,
  HistoryPageComponent,
  OrderPageComponent,
  CategoriesPageComponent,
  CategoriesFormComponent,
  PositionsFormComponent,
  OrderCategoryComponent,
  OrderPositionsComponent,
  HistoryListComponent,
  HistoryFilterComponent
 ],
 imports: [
  CommonModule,
  SharedModule
 ],
 providers: [
  CategoriesService,
  PositionsService,
  OrderService,
  OrdersService,
  AnalyticsService
 ]
})
export class SiteModule { }
```

Файл app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { SiteModule } from './site/site.module';
import { LoginModule } from './login/login.module';
```

```
import { CoreModule } from './core/core.module';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';



@NgModule({
 declarations: [
  AppComponent
 ],
 imports: [
  BrowserModule,
  AppRoutingModule,
  CoreModule,
  LoginModule,
  SiteModule
 ],
 providers: [],
 bootstrap: [AppComponent]
})
export class AppModule { }
```

Файл app-routing.module.ts

```
import { OrderPositionsComponent } from './site/order-page/order-
positions/order-positions.component';
import { CategoriesFormComponent } from './site/categories-page/categories-
form/categories-form.component';
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AuthGuard } from './shared/Classes/auth.guard';
import { AuthLayoutComponent } from './login/auth-layout/auth-
layout.component';
import { LoginPageComponent } from './login/login-page/login-page.component';
import { SiteLayoutComponent } from './site/site-layout/site-layout.component';
import { RegisterComponent } from './login/register/register.component';
import { OverviewPageComponent } from './site/overview-page/overview-
page.component';
import { CategoriesPageComponent } from './site/categories-page/categories-
page.component';
import { OrderPageComponent } from './site/order-page/order-page.component';
import { HistoryPageComponent } from './site/history-page/history-
page.component';
import { AnalyticsPageComponent } from './site/analytics-page/analytics-
page.component';
import { OrderCategoryComponent } from './site/order-page/order-category/order-
category.component';
```

```
const routes: Routes = [
 {
  path: '', component: AuthLayoutComponent, children: [
   { path: '', redirectTo: 'login', pathMatch: 'full' },
   { path: 'login', component: LoginPageComponent },
   { path: 'registration', component: RegisterComponent },
  ]
 },
 {
  path: '', component: SiteLayoutComponent, canActivate: [AuthGuard], children:
[
   { path: 'overview', component: OverviewPageComponent  },
   { path: 'analytics', component: AnalyticsPageComponent  },
   { path: 'history', component: HistoryPageComponent  },
   { path: 'order', component: OrderPageComponent, children: [
    { path: '', component: OrderCategoryComponent  },
    { path: ':id', component: OrderPositionsComponent  },
   ] },
   { path: 'categories', component: CategoriesPageComponent  },
   { path: 'categories/new', component: CategoriesFormComponent  },
   { path: 'categories/:id', component: CategoriesFormComponent  },
  ]
 },


];

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