МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____Аліна САВЧЕНКО

"____"_____2021 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"МАГІСТРА"

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ "ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Прикладний програмний інтерфейс системи замовлень для інтернет маркетплейсу"

Виконавець: Сульжик Роман Володимирович

Керівник: к.т.н., доцент Савченко Аліна Станіславівна

Нормоконтролер: _____ Ігор РАЙЧЕВ

Київ - 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра Комп'ютерних інформаційних технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології" .

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Аліна САВЧЕНКО
«____»_____2021р.

ЗАВДАННЯ

на виконання дипломної роботи студента

Сульжика Романа Володимировича
(прізвище, ім'я, по батькові)

1. Тема роботи: «Прикладний програмний інтерфейс системи замовлень для інтернет маркетплейсу» затверджена наказом ректора від 12.10.2021 за № 2228/ст.

2. Термін виконання роботи: з 12.10.2021 по 31.12.2021.

3. Вихідні дані до роботи: теоретичні відомості та основи програмування на мові PHP, проектування та розробка API на основі веб фреймворку.

4. Зміст пояснювальної записки: вступ, аналітичний огляд і постановка завдання, аналіз можливих методів та засобів для розробки пропонованого продукту, план, структура та розробка продукту, висновок..

5. Перелік обов'язкового ілюстративного матеріалу: слайди, презентація.

6. Календарний план-графік

№ п/п	Завдання	Термін виконання	Підпис керівника
1.	Отримання завдання на дипломну роботу та побудова плану-графіку виконання робіт.	12.10.2021 – 15.10.2021	
2.	Аналіз існуючих методів та інструментів для створення API.	16.10.2021 – 19.10.2021	
3.	Проаналізувати літературу та джерела за темою дипломного проекту.	20.10.2021 – 24.10.2021	
4.	Проектування та розробка логіки прикладного програмного інтерфейсу.	25.10.2021 – 31.10.2021	
5.	Написання Розділу 1 дипломної роботи.	01.11.2021 – 07.11.2021	
6.	Обрання всіх можливих інструментів та програм для виконання задач. Написання Розділу 2 дипломної роботи.	08.11.2021 – 17.11.2021	
7.	Написання Розділу 3 дипломної роботи. Створення програмної частини проекту. Завершення створення пояснювальної записки дипломної роботи.	18.11.2021 – 01.12.2021	
8.	Оформлення та друк пояснювальної записки.	02.12.2021 – 11.12.2021	
9.	Створення презентації, доповіді та підготовка до захисту дипломної роботи	12.12.2021 – 20.12.2021	

7. Дата видачі завдання: 12.10.2021р.

Керівник дипломної роботи Аліна САВЧЕНКО
(підпис керівника)

Завдання прийняв до виконання Роман СУЛЬЖИК
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Прикладний програмний інтерфейс системи замовлень для інтернет маркетплейсу” складається із вступу, трьох розділів, висновків до кожного розділу, загальних висновків, списку використаних джерел і містить 94 сторінки тексту, 60 рисунків та 1 таблицю. Список використаних джерел містить 15 найменувань.

Метою дипломної роботи є дослідження теоретичних засад та практичних способів реалізації REST API та розробка інтеграцій даних інтернет маркетплейсу із зовнішніми сервісами.

Предметом дослідження є кросплатформна система обміну даних про замовлення в інтернет маркетплейсі.

Об’єктом дослідження є процес обміну даних в рамках інтернет маркетплейсу із зовнішніми сервісами, бекенд та база даних сайту, його технології та інструменти.

Ключові слова: ВЕБ-САЙТ, БЕКЕНД, API, REST, RESTFULL, REDBEANPHP, MVC, PHP, HTTP, JSON, SQL, LOG, MARKETPLACE, CROSSPLATFORM.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Маркетплейс – тип веб-сайту електронної комерції, де інформація про товар чи послугу надається кількома третіми сторонами.

API – Application Programming Interface – набір чітко визначених методів для взаємодії різних компонентів.

Фронтенд – клієнтська сторона WEB-додатку.

Бекенд – серверна сторона WEB-додатку.

Лог – файл у якому накопичується службова та статична інформація про події.

БД – база даних.

REST – Representational State Transfer – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів.

ВСТУП

WEB-додаток – розподілений застосунок, в якому клієнтом виступає браузер, а сервером – веб сервер. Маркетплейс — тип веб-сайту електронної комерції, де інформація про товар чи послугу надається кількома третіми сторонами. Маркетплейси характеризуются тим, що розроблена спеціальна платформа для надання послуг електронної комерції, розроблено інструменти для замовлення продукції, відгуків, огляд товару чи послуг, але найхарактернішою рисою є наявність багатьох продавців на одному ресурсі. Маркетплейси не мають складських приміщень чи запасів товару, фактично, здійснюють інформаційні послуги зв'язка між покупцем та продавцем.

Найвідоміші приклади мультибрендових сайтів: Amazon і Alibaba Group. В Україні прикладом маркетплейсів є Prom.ua, Rozetka.ua, Bezet.com.ua.

Очевидно, що кожен з постачальників використовує свої власні системи обробки замовлень та врахування наявності товарів на складі. Серед поширених CRM систем можна виділити Bitrix24, AmoCRM, Zoho. Деякі бренди можуть використовувати власні розробки через зручність чи відсутність потрібного функціоналу.

Стандартний шлях збереження замовлення на сайті:

1) Збереження в базі даних;
2) В адміністративній панелі, та в панелі доступу менеджера можна отримати доступ до замовлень з БД;
3) Менеджер керує замовленням з доступної для нього панелі.

Однак, очевидно, що менеджер не може завжди бути онлайн, особливо якщо бренд працює одночасно з десятками маркетплейсів. В такому випадку, при отриманні нового замовлення – сервер надсилає дані на CRM систему бренду, до якого прийшло замовлення, і всі подальші дії з замовленням відбуваються на стороні бренду. Такий функціонал вирішує проблеми навантаження основного серверу, зручності обробки замовлень для кожного бренду, та забезпечення одного обробника подій для різних систем, що в свою чергу дозволяє легко редагувати всі дані одразу.

У зв’язку з відсутністю API на маркетплейсі Bezet та з постійним збільшенням кількості брендів актуальною є задача розробки REST API для обробки замовлень.

Для вирішення поставленої задачі необхідно:

- провести аналітичний огляд всіх можливих аналогів;
- проаналізувати найкращі існуючі методи для швидкодії;
- розробити функціонал сповіщень про замовлення;
- розробити систему авторизації для запитів;
- розробити запити на отримання даних;
- розробити запити на додавання даних;
- розробити запити на редагування даних;
- розробити лог помилкових запитів

РОЗДІЛ 1.
АНАЛІТИЧНИЙ ОГЛЯД І ПОСТАНОВКА ЗАДАЧІ

1.1. Поняття інтернет маркетплейсу

Інтернет маркетпейс – це веб-сайт або додаток, що дозволяє робити покупки в різних постачальників [1]. Власник маркетплейсу не володіє ніякими товарами фізично, його справа – представити товари чи послуги інших користувачів споживачу та сприяти здійсненню угоди. eBay – найкращий приклад інтернет маркетплейсу, вони продають все і кожному.

Рис. 1.1. Ebay – схема роботи

Оскільки користувачі отримують доступ до товарних запасів постачальників в електронному вигляді, і власник не зобов'язаний володіти ними, перш ніж пропонувати їх споживачам, усі товари, що продаються постачальниками, доступні споживачам, а інформація про продукти, які представлені споживачам, доступна в режимі реального часу. В таких додатках асортимент набагато ширший, ніж в будь якому стаціонарному магазині.

Споживачі не люблять користуватися програмами чи сайтами окремих продавців. Вони набагато частіше завантажують додаток, який пропонує асорти-

мент продуктів від багатьох брендів. Це одна з головних проблем, яку вирішує маркетплейс.

Такі сайти мають деякі мінуси. Оскільки товари пропонуються багатьма продавцями, інформація про них часто не актуальна, а швидкість доставки від продавців різна.

Однак маркетплейси – це не тільки покупки. Це також може бути сервіс шерінгу послуг. Так, наприклад, Тревіс Каланік і Геррет Кемп марно намагалися зловити таксі в Парижі, коли місто заметало снігом. Тоді їх осяяла ідея: як було б зручно натиснути кнопку на мобільному та побачити порожні машини поблизу. У березні 2009-го вони реалізували задум в Америці, в 2010 – вийшли на міжнародний рівень.

Uber став одним із популярних маркетплейсів на ринку послуг — він допомагає знаходити транспорт у різних містах світу (більше 450). Користувачі відстежують маршрут руху таксі та оплачують поїздки через мобільний додаток. За прикладом цієї компанії з'явилися й інші, де можна найняти дитину няню, знайти садівника для стрижки газону або ветеринара, який вилікує собаку.

Якщо ви не розумієте, чому маркетплейс завоювали інтернет, подумайте з позиції користувача, що простіше:

- Шукати речі онлайн або в пошуковій системі Ebay?
- Зберігати телефони безвідмовних служб таксі або скористатися програмою Uber?
- Продати старий ноутбук на ринку або розмістити оголошення на Olx?
- Зняти номер на офіційному сайті готелю чи вибрати квартиру на Airbnb?

Головний фактор, за яким ідентифікують маркетплейс – безліч вендорів на одному сайті.

Вендор (vendor, provider) – компанія-постачальник товарів або послуг, яка виробляє та реалізує продукцію.

За рахунок великої кількості вендорів маркетплейс підкорює користувачів різноманітністю речень. У клієнта завжди є можливість вибрати більше та краще. Зростаюча кількість зацікавлених покупців, у свою чергу, робить торговий майданчик привабливим для нових вендорів. Щоб ця схема працювала налагоджено, маркетплейс має виконати 3 функції:

1) познайомити споживачів із постачальниками;
2) спростити для них процес комунікації, обміну товарами, послугами та платежами;
3) забезпечити нормативно-правову базу, що ляже основою функціонування ринку.

Набираючи популярності, маркетплейси витісняють інтернет-магазини та централізують навколо себе торгівлю в інтернеті.

1.2. Поняття прикладного програмного інтерфейсу

Прикладний програмний інтерфейс (англ. Application Programming Interface, API) — набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення [2]. Спрощено — це набір чітко визначених методів для взаємодії різних компонентів. API надає програмісту засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

API пов'язує різні частини програмної платформи, щоб інформація, що передається, дійшла до місця призначення.

Ці сполучні вузли не тільки виконують роль внутрішніх каналів зв'язку, але і дозволяють зовнішнім інструментам отримувати доступ до цієї інформації.

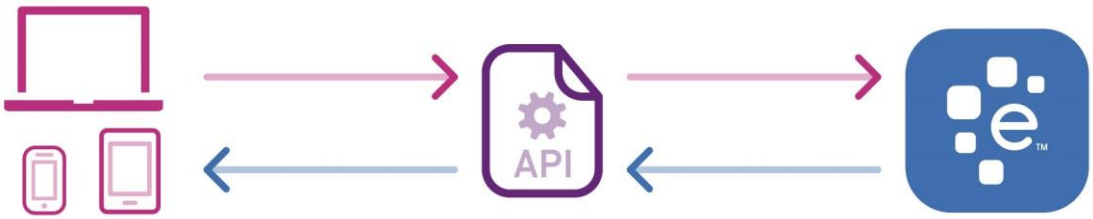


Рис. 1.2. Загальна схема роботи API

Веб-API – це інтерфейс прикладного програмування для веб-сервера або веб-браузера. Це концепція веб-розробки, зазвичай обмежена клієнтською стороною веб-програми (включаючи будь-які веб-фреймворки, що використовуються), і, як правило, не включає деталі реалізації веб-сервера чи браузера, якщо вони не є загальнодоступними для віддаленої веб-програми.

Інтерфейс – це зв’язок між додатком А та додатком Б. У ньому виникають процеси, які дозволяють програмам обмінюватись інформацією та виконувати функції, пов'язані з двома сторонами, приховуючи «внутрішню будову».

Такий підхід дозволяє налагоджувати взаємодію між нескінченними утилітами, не задумуючись про те, як вони побудовані, яка програмна логіка імітує і які форми обробляють передані дані. Інтерфейси підтримують роботу як для простих користувачів, так і для програмістів. Розробникам при цьому не потрібно вивчати коди інших програмістів, щоб підключити чужий продукт до свого, що значно економить час.

Основна мета API – інкапсуляція, коли приховується частина функцій для спрощення програми та зменшується виклик функцій програмного забезпечення, де один із розробників міг зробити помилку.

Багато компаній пропонують API як готовий продукт. Наприклад, Weather Underground продає доступ до свого API для отримання метеорологічних даних.

Сценарій використання API: на сайті невеликої компанії є форма для запису клієнтів на прийом. Компанія хоче вбудувати в нього Google Календар, щоб дати клієнтам можливість автоматично створювати подію та вносити деталі про майбутню зустріч.

Застосування API: ціль – сервер сайту повинен безпосередньо звертатися до сервера Google із запитом на створення події з зазначеними деталями, отримувати відповідь Google, обробляти її, і передавати відповідну інформацію в браузер, наприклад, повідомлення із запитом на підтвердження користувачеві.

В якості альтернативи браузер може зробити запит до API сервера Google, оминаючи сервер компанії.

Слово "application" (прикладний, програма) може застосовуватися в різних значеннях. У контексті API воно має на увазі:

- фрагмент програмного забезпечення з певною функцією,
- сервер повністю, додаток повністю або просто окрему частину програми.

Будь-який фрагмент ПЗ, який можна чітко виділити з оточення, може замінювати літеру «А» в англомовній абревіатурі, і теж може мати певний API. Наприклад, при введенні в код розробником сторонньої бібліотеки вона стає частиною всього додатка. Будучи самостійним фрагментом ПЗ, бібліотека матиме якийсь API, який дозволить їй взаємодіяти з іншим кодом програми.

В об'єктно-орієнтованому проектуванні код представлений як сукупності об'єктів. У додатку таких об'єктів, що взаємодіють між собою, можуть бути сотні. У кожного з них є свій API – набір публічних властивостей та методів для взаємодії з іншими об'єктами у додатку. Об'єкти можуть мати приватну, внутрішню логіку, яка прихована від оточення і не є API.

1.3. Види API та їх переваги

API — application programming interface (прикладний програмний інтерфейс), набір правил і механізмів, з допомогою яких один додаток взаємодіє з іншим. При цьому додаток який виступає на стороні клієнту не знає особливостей поведінки та складних бізнес процесів на основному додатку (сервері), та ігнорує їх. Він лише використовує команди для отримання чи створення чогось, без вдавання в деталі.

Плюсів використання API чимало:

- Найголовніший плюс роботи з API – це економія часу розробки власних сервісів. Програміст отримує готові рішення, і йому не потрібно витрачати час на написання коду для функціоналу, який вже давно реалізований.
- В API можуть враховуватися нюанси, які сторонній розробник може не врахувати або просто не знати,
- API дає додаткам певну системність і передбачуваність – одна й та сама функція за допомогою API може бути реалізована в різних додатках так, що буде зрозуміла та знайома всім користувачам.
- API дає стороннім розробникам доступ до закритих сервісів.

Але також є й мінуси:

- Якщо до основного сервісу вносяться зміни та доробки, в API вони можуть потрапити не відразу,
- Розробнику доступні готові рішення, як саме вони реалізовані і як виглядає вихідний код, він не знає,
- API призначений насамперед для загального використання, він може не підійти до створення якогось особливого функціоналу.

Публічні API випускаються такими компаніями, як Slack і Shopify, сподіваючись, що розробники їх використовуватимуть на своїх платформах. Компанії діляться набором вступних параметрів, які розробники використовують, щоб досягти якогось результату.

Публічне API можна використовувати без проблем – доступ до документації можна отримати без проблем.

Приватні API використовуються всередині компанії. Якщо компанія має багато програмних продуктів, приватне API використовується, щоб програми розмовляли між собою. Компоненти API можуть змінюватися за бажанням компанії, тоді як зміни в публічному API можуть викликати відчайдушні протести.

До WEB API відносяться [3]:

- XML-RPC та JSON-RPC
- SOAP
- REST

1.3.1. XML-RPC та JSON-RPC

XML-RPC (Extensible Markup Language Remote Procedure Call — XML-виклик віддалених процедур) — базується на XML стандарті (протоколі) виклику віддалених процедур, вирізняється простотою використання [4]. XML-RPC визначає набір стандартних типів даних та команд, які розробник може використовувати для доступу до функціональності іншої програми, що знаходиться на іншому комп'ютері в мережі. В даний момент не підтримується через перехід до новіших методів, як SOAP чи REST, однак використовується в завершених проектах.

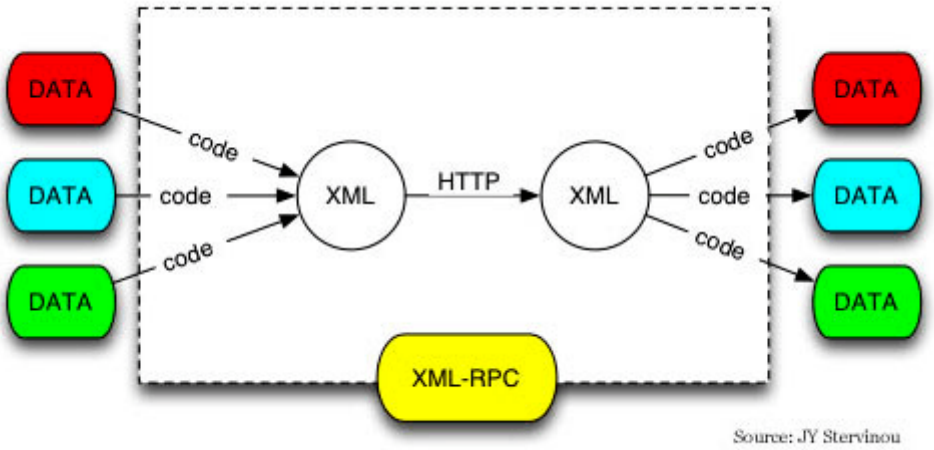


Рис. 1.3. Схема роботи XML-RPC API

JSON-RPC–це протокол виклику віддаленої процедури, кодований у JSON. Він подібний до протоколу XML-RPC. JSON-RPC дозволяє отримувати сповіщення (дані, що надсилаються на сервер, що не потребує відповіді), а також надсилати на сервер кілька запитів, на які можна відповідати асинхронно.

1.3.2. SOAP API. Переваги та недоліки

SOAP – протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, базується на форматі XML [5].

Протокол використовується для обміну повідомленнями в форматі XML, а не тільки для виклику процедур. SOAP є розширенням мови XML-RPC.

SOAP можна використовувати з будь-яким протоколом прикладного рівня: SMTP, FTP, HTTP та інші. Проте його взаємодія з кожним із цих протоколів має свої особливості, які потрібно відзначити окремо. Найчастіше SOAP використовується разом з HTTP.

Структура SOAP повідомлення має XML формат та наступний вигляд:

```
<?xml version='1.0' Encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
                env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2007-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
                env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
   <n:name>Fred Bloggs</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2007-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2007-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference></p:seatPreference>
   </p:return>
  </p:itinerary>
 </env:Body>
</env:Envelope>
```

Рис 1.4. Soap лист

Повідомлення SOAP структурується так:

SOAP-конверт

SOAP-заголовок

Елемент заголовку 1

Елемент заголовку 2

…

Елемент заголовку N

Тіло SOAP

Елемент тіла 1

Елемент тіла 2

…

Елемент тіла N

Переваги:

- Гнучкість;
- Мультипротокольність;
- Підтримка HTTP, SMTP.

Недоліки:

- Великий розмір даних
- Помилки кодування
- Різні формати передачі даних

1.3.3. REST API. Переваги та недоліки

REST – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP. В основі REST закладено

принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0 [6].

Компоненти REST системи спілкуються, передаючи один одному представлення ресурсу в форматі, що обирається з оновлюваного набору стандартних форматів даних. Формат обирається динамічно відповідно до бажань компонента-клієнта і можливостей сервера. Чи представлення має той самий формат, що й сам ресурс, чи є результатом якогось перетворення — це деталь реалізації, яка ховається за інтерфейсом.

Наглядну відмінність між SOAP та REST типами можна порівняти з листом та листівкою. SOAP – складна та важка процедура повного оформлення листа в конверт, підпис та підготовка матеріалу, в свою ж чергу REST – карточка з повністю готовими даними, де лише потрібно вказати отримувача.

Рис. 1.5. SOAP та REST порівняння з листом

Переваги:

- Простота;
- Використання готових стандартів;
- Кешування;
- Масштабування;
- Стандартні коди HTTP помилок.

Недоліки:

- Невеликий словник кодів відповідей;
- Нечіткість кодів;
- Складність налагодження.

Суть роботи алгоритму полягає у діях, залежно від типу запиту. Від роботи сервера залежить функціонал та можливості архітектури. Є 4 основних види щодо інформації:

- get – отримання, просто передача;
- delete – видалення, надалі вони не відображаються;
- post – оновлення або додавання;
- put – оновлення, регулярна операція.

Рис. 1.6. HTTP типи запитів

Запит зазвичай відправляється JSON масивом на вказаний конкретний URL. Там спрацьовує функція, а залежно від вже надісланих даних та поточного запиту починається певна дія. При цьому не має значення, з якого пристрою надіслана інформація – мобільний додаток, браузер комп'ютера, чи напряму з сервера клієнта.

1.4. Постановка задачі

В зв’язку з постійним розширенням проекту потреба в API інтеграції гостро зростає з кожним днем.

Порівняльний аналіз видів дозволив виявити API який раціонально використовувати в сучасних проектах. Враховуючи вищенаведене, актуальною є задача розробки прикладного програмного інтерфейсу системи замовлень для інтернет маркетплейсу BEZET.

Під час розробки API необхідно було створити запити для перегляду, додавання та редагування замовлень з сайту за допомогою REST API та запитів / відповідей в форматі JSON.

Для універсальності підключення до інших клієнтів було обрано саме REST API на HTTP протоколі передачі даних з стандартними кодами відповідей та простою документацією.

Таким чином для розв‘язання поставленої задачі розробки прикладного програмного інтерфейсу необхідно:

- провести аналітичний огляд всіх можливих варіантів;
- проаналізувати найкращі існуючі методи для швидкодії;
- розробити систему авторизації;
- розробити запити на отримання даних;
- розробити запити на додавання даних;
- розробити запити на редагування даних;
- розробити документацію до API.

Висновки до розділу 1

API – набір чітко визначених методів для взаємодії різних компонентів. Основна мета API – взаємодія між програмою А (в даному випадку маркетплейсом BEZET) та програмою Б (CRM системою бренду) за допомогою серверного протоколу HTTP.

За допомогою універсальних функції, та запитів можна оновлювати дані про замовлення, змінювати їх дані, статуси чи додавати нові замовлення на маркетплейс. Завдяки API користування платформою стає набагато зручнішим для користувача, адже весь інтерфейс він може вибрати сам під себе, та поєднати всі маркетплейси в одному місці.

Для зручності – всі запити сформовані в форматі JSON, що дозволяє легко їх обробляти, читати та редагувати. До того ж, такі запити займають менше місця, в порівнянні з XML.

Отже, актуальною є проблема розробки прикладного програмного інтерфейсу для обробки замовлень на інтернет маркетплейсі. В якості бази даних – виступає MYSQL база проекту з доповненням таблиць. Веб сервер – apache. Мова розробки – PHP.

РОЗДІЛ 2.
АНАЛІЗ МОЖЛИВИХ МЕТОДІВ ТА ЗАСОБІВ ДЛЯ РОЗРОБКИ ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ

2.1. Засоби розробки

Вся розробка включає в себе бекенд частину веб додатку та створення документації для розробників по взаємодії з API. Для розробки доцільно обирати стандартні засоби.

- SQL – мова для взаємодії з базами даних;
- PHP – скриптова мова програмування для генерації HTML сторінок на стороні веб-серверу;
- PHPStorm – середовище для розробників з зручними розширеннями;
- POSTMAN – ПЗ для тестування HTTP запитів;
- JSON – формат обміну даними.
- HTML – гіпертекстова мова розмітки;
- CSS – мова стилю сторінок;
- LAMP – Linux + apache + mysql + php – стандартний пакет ПЗ для роботи більшості веб додатків.

Рис. 2.1. LAMP

2.1.1. Мова програмування PHP

PHP – гіпертекстовий препроцесор (Hypertext Preprocessor) – скриптова мова програмування, яка була створена для формування HTML сторінок на стороні веб-сервера [7]. PHP – повноцінна мова з підтримкою ООП. Оскільки проект є монолітним та написаним на PHP саме ця мова обрана для розгортання API. Актуальною стабільною версією на момент розробки WEB-додатку є PHP v.8.0, однак розробка буде відбуватись на версії 7.3, оскільки це основна версія всього маркетплейсу. При цьому варто зберегти підтримку версій від 5.6 і вище, оскільки багато хостинг провайдерів досі в базовому тарифі пропонують версії 5.6-5.7.

PHP також зручна в використанні тому, що більшість готових коробочних рішень, CRM та CMS систем написані також на цій мові. Тому в випадку проблем з інтеграціями чи налаштуваннями один розробник може вирішувати різні цикли в задачах.

Проект використовує PHP v.7.3 та веб сервер Apache. Ця зв’язка популярна та активно використовується на багатьох середньо навантажених проектах.

Типовий PHP-сценарій – це набір виразів. Кожен вираз починається з нового рядка і закінчується крапкою з комою.

Вираз — це інструкція, яка наказує PHP-інтерпретатору виконати одну дію, наприклад скласти два числа або вивести на екран інформацію. Найпростіший сценарій, який виводитиме на екран один рядок: «Привіт, Світ!» виглядає наступним чином:

```
<?php echo("Привіт, світ");
```

Важливо відзначити, що будь-який PHP-сценарій обов'язково починається з такого рядка: <?php – так ми повідомляємо веб-серверу, що далі у файлі знаходиться код мовою PHP.

У прикладі сценарій складається з лише однієї інструкції: echo("Привіт, Світ!");

Тут ми просимо функцію echo вивести на екран наш текст – "Привіт, Світ!". Як відомо, PHP можна вбудувати у статичні HTML-файли та модифікувати їх за допомогою інструкцій. Ось як виглядатиме той самий приклад із виведенням тексту, але розташований усередині HTML:

```
<html>
  <head>
    <title>Наша перша сторінка php</title>
  </head>
  <body>
    <h1><?php print("Привіт, світ"); ?></h1>
  </body>
</html>
```

Важливо, що у цьому прикладі ми використали додатковий фрагмент – ?>. З його допомогою ми повідомляємо сервер, де закінчується наш PHP-сценарій. У випадку, якщо наш код останній у документі, і після нього нічого не відбувається, цей фрагмент не є обов'язковим.

2.1.2. Мова запитів SQL

База даних (БД) — впорядкований набір логічно взаємопов'язаних даних, що використовуються спільно, та зберігаються в одному місці. Якщо коротко, то найпростіша БД це звичайна таблиця з рядками та стовпцями у якій зберігається різного роду інформація (прикладом може слугувати таблиця в Excel). Так, часто, з БД нероздільно пов'язують Системи управління базами даних (СУБД), які надають функціонал для роботи з БД. Мова SQL якраз і є частиною СУБД, яка здійснює керування інформацією в БД.

SQL – декларативна мова програмування для взаємодії користувача з базами даних, що застосовується для формування запитів, оновлення, керування різноманітними БД [8]. Саме за допомогою запитів до баз даних, можна отримати дані про замовлення, товари, чи будь які інші об’єкти, які використовуються в додатку.

Основні запити в API будуть мати наступний вигляд:

- Отримання даних – SELECT ‘дані’ FROM ‘сутність’
- Вставка даних – INSERT INTO ‘сутність’ ‘дані’
- Оновлення даних – UPDATE ‘сутність’ SET ‘дані’ ‘умова’
- Видалення даних – UPDATE ‘сутність’ SET ‘дані’ ‘умова’

Видалення даних не відбувається через запит DELETE для безпеки проекту. Все видалення даних насправді не знищує запис, а лише переводить його в особливий статус, перегляд якого доступний виключно адміністратору проекту.

Проект BEZET має реляційну базу даних в якій 59 таблиць. Для швидкодії API та роботою з таблицями можна використовувати готовий драйвер PHP mysqli. Однак, оскільки в самому проекті вже використовується ORM RedBean, то доцільніше використовувати її, задля збереження єдиного стилю коду та записів на всьому проекті.

Основною командою з якою потрібно працювати при отриманні будь яких даних є SELECT.

SELECT (від англ. select - "вибрати") – оператор запиту (DML/DQL) у мові SQL, що повертає набір даних (вибірку) з бази даних.

Оператор повертає нуль або більше рядків. Список стовпців, що повертаються, задається в частині оператора, званої пропозицією SELECT. Оскільки SQL є декларативною мовою, запит SELECT визначає лише вимоги до набору даних, що повертається, але не є точною інструкцією з їх обчислення. СУБД транслює запит SELECT у внутрішній план виконання (query plan), який може відрізнятися навіть для синтаксично однакових запитів і від конкретної СУБД.

Оператор SELECT складається з кількох пропозицій (розділів):

- SELECT визначає список стовпців, що повертаються (як існуючих, так і обчислюваних), їх імена, обмеження на унікальність рядків у наборі, що повертається, обмеження на кількість рядків у повертається наборі;
- FROM визначає табличний вираз, який визначає базовий набір даних для застосування операцій, що визначаються в інших пропозиціях оператора;
- WHERE визначає обмеження на рядки табличного виразу з пропозиції FROM;
- GROUP BY поєднує ряди, що мають однакову властивість із застосуванням агрегатних функцій
- HAVING вибирає серед груп, визначених параметром GROUP BY
- ORDER BY задає критерії сортування рядків; відсортовані рядки передаються до точки виклику.

Оператор SELECT має наступну структуру:

```
SELECT
  [DISTINCT | DISTINCTROW | ALL]
  select_expression,...
FROM table_references
[WHERE where_definition]
[GROUP BY {unsigned_integer | col_name | formula}]
[HAVING where_definition]
[ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC], ...]
```

2.1.3. RedBeanPHP

RedBeanPHP – це потужний набір бібліотек та функцій, яке працює на основі ORM для PHP, яка спрощує роботу з базами даних в додатку. ORM або Object-relational mapping (Об'єктно-реляційне відображення) — це технологія

програмування, яка дозволяє перетворювати несумісні типи моделей в ООП, зокрема між сховищем даних та об'єктами програмування.

ORM використовується для спрощення процесу збереження об'єктів у реляційну базу даних та їх вилучення, при цьому ORM сама дбає про перетворення даних між двома несумісними станами.

При використанні RedBeanPHP (як і будь-який інший ORM) не завжди можна обмежитись простими методами пошуку (Finding). Часто існує потреба зробити складніший запит, який зробити простими методами вкрай проблематично. Для цього можна використовувати аналог прямого запиту в БД завдяки команді exec (execute).

2.2. JSON формат передачі даних

JSON (англ. JavaScript Object Notation, укр. запис об'єктів JavaScript) – це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується переважно для передачі структурованої інформації через мережу [8]. JSON широко використовується в API через низку причин, серед яких:

- простота читання;
- пряме кодування php масив – json – php масив;
- частота використання в API;
- менший розмір порівняно з XML;
- безпека в передачі даних;
- проста передача складних структур.

У JSON використовуються такі форми структур:

- Об'єкт – це послідовність пар назва/значення. Об'єкт починається з символу “{“ і закінчується символом “}”. Кожне значення слідує за : і пари назва/значення відділяються комами.

- Масив – це послідовність змінних. Масив починається символом “[“ і закінчується символом “]”. Значення відділяються комами.
- Змінна – може бути рядком в подвійних лапках, або числом, або логічними true чи false, або null, або об'єктом, або масивом. Ці структури можуть бути вкладені одна в одну.

JSON є рядком, формат якого дуже схожий на буквений формат об'єкта JavaScript. Ви можете включати ті самі базові типи даних всередині JSON, так само як і в стандартному об'єкті JavaScript – рядки, числа, масиви, логічні та інші об'єктні літерали. Це дозволяє побудувати ієрархію даних, наприклад, так:

```
{
  "Name": "Roman",
  "Surname": "Sulzhyk",
  "Date": "02.04.1999",
  "Country": "Ukraine",
  "Telephone": "+380633333333",
  "family": [
    {
      "Name": "Natali",
     "Surname": "Sulzhyk",
     "Date": "20.09.1973",
    },
    {
      "Name": "Volodymyr",
     "Surname": "Sulzhyk",
     "Date": "14.12.1974",
    },
    }
  ]
}
```

2.3. REST API

API – спосіб взаємодії та обміну даними між серверами. Більшість великих компаній розробляють цей інтерфейс для клієнтів, та для інтеграцій з зовнішніми сервісами.

REST – Representational State Transfer — підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів.

REST API використовує запити HTTP для виконання стандартних функцій баз даних, таких як створення, читання, оновлення чи видалення записів. Наприклад, REST API може використовувати запит GET для отримання запису, запит POST для створення, PUT – оновлення та DELETE для видалення. В запитах API підтримуються всі методи HTTP. Добре спроектований REST API можна порівняти з веб сайтом, який працює в веб-браузері з вбудованою підтримкою HTTP.

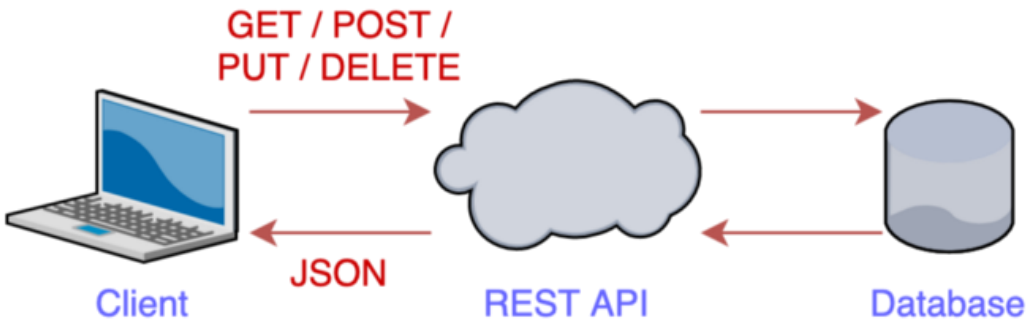


Рис. 2.2. Схема роботи REST API

2.3.1. RESTful та REST

Існує два поняття – REST та RESTful.

REST (REpresentational State Transfer) – це в основному архітектурний стиль забудови, який має деякі принципи [9]:

- Він має отримати доступ до всіх ресурсів із сервера, використовуючи лише URL
- Він не має вбудованого шифрування
- У ньому немає сесії
- Він використовує один і тільки один протокол – HTTP
- Для виконання операцій CRUD він повинен використовувати дієслова HTTP, такі як get, post, put і delete
- Він повинен повертати результат лише у вигляді JSON або XML.

Служби на основі REST дотримуються деяких з вищенаведених принципів, але не всіх. Застосунки RESTful означають, що вони відповідають усім вищезазначеним принципам. Це схоже на концепцію ООП:

- Об'єктно-орієнтовані мови підтримують усі концепції ООП, приклади: C++, C#
- Об’єктно-орієнтовані мови підтримують деякі функції ООП, наприклад: JavaScript, VB

2.4. Види HTTP методів на сервер

HTTP — протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від HyperText Transfer Protocol, протокол передачі гіпертекстових документів [10].

- GET запитує вміст вказаного ресурс. Запити з використанням цього методу можуть приймати параметри. Згідно зі стандартом HTTP, запити типу GET вважаються ідемпотентними — багаторазове повторення одного і того ж запиту GET повинне приводити до однакових результатів (за умови, що сам ресурс не змінився за час між запитами).
- HEAD запитує ресурс так само, як і метод GET, але без тіла відповіді.

- POST використовується для надсилання сутностей до певного ресурсу. Часто викликає зміну стану або якісь побічні ефекти на сервері. На відміну від методу GET – не вважається ідемпотентним, тобто багаторазове повторення одних і тих же запитів POST може повертати різні результати (наприклад, після кожного відправлення повідомлення з'являтиметься одна копія цього повідомлення).
- PUT замінює всі поточні уявлення ресурсу даними запиту.
- DELETE видаляє вказаний ресурс.
- CONNECT встановлює "тунель" до сервера, визначеного за ресурсом.
- OPTIONS повертає методи HTTP, які підтримуються сервером. Цей метод може служити для визначення можливостей вебсервера.
- TRACE виконує виклик тестового повідомлення, що повертається з ресурсу.
- PATCH використовується для часткової зміни ресурсу.

Офіційна документація HTTP протоколу має наступну класифікацію:

HTTP Method	RFC	Request Has Body	Response Has Body	Safe	Idempotent	Cacheable
GET	RFC 7231	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 7231	No	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	No	Yes	No	Yes	No
CONNECT	RFC 7231	Yes	Yes	No	No	No
OPTIONS	RFC 7231	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	No

Рис. 2.3. HTTP методи

Для забезпечення коректної роботи API та очевидну поведінку – потрібно дотримуватись варіантів відповідей та очікуваної поведінки відносно документації.

В стандартних API інтеграціях використовують чотири основних типи запитів:

- GET – отримання масиву даних
- POST – створення нового об’єкту
- PUT – оновлення даних об’єкту
- DELETE – видалення об’єкту

Процес створення, читання, оновлення та видалення інформації має назву CRUD (від англійських слів Create, Read, Update, Delete). CRUD широко використовується в усіх сучасних системах, оскільки є основоположним механізмом для роботи з таблицями в базах даних. Схематично аналогія між REST та CRUD виглядає наступним чином:

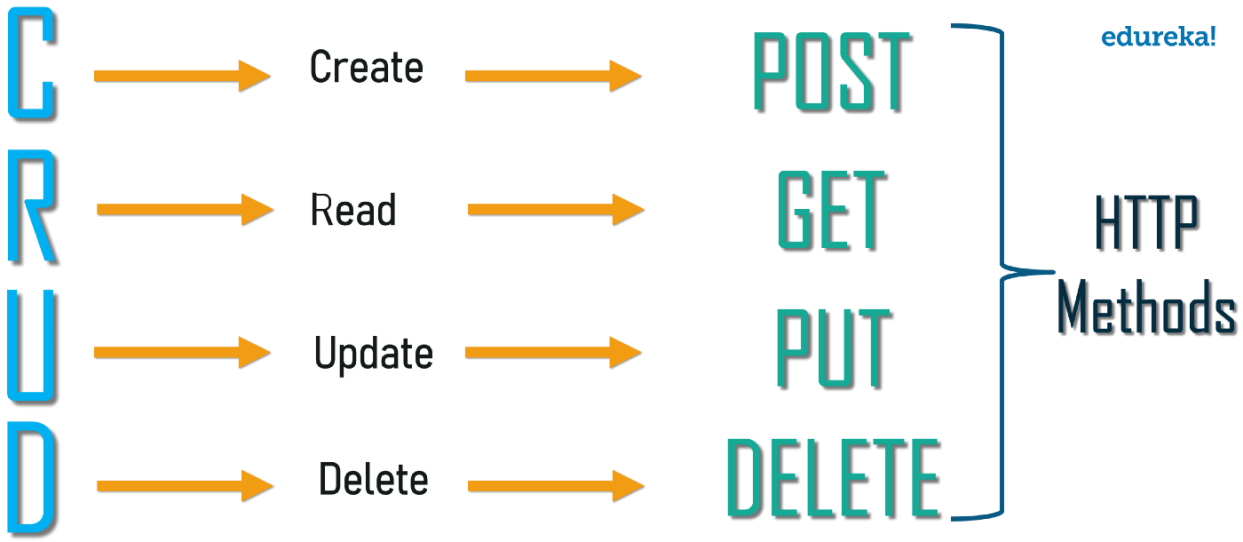


Рис. 2.4. CRUD – REST аналог

2.4.1. Заголовки в HTTP запиті

Заголовки HTTP дозволяють клієнту та серверу надсилати додаткову інформацію з HTTP запитом або відповіддю. HTTP-заголовок містить не чутливу до регістру назву, а потім після (:) безпосередньо значення.

HTTP-заголовки супроводжують обмін даними за протоколом HTTP. Вони можуть містити опис даних та інформацію, необхідну для взаємодії між клієнтом та сервером.

В API в заголовках надсилається ключ авторизації, для того, щоб впевнитись, що саме цей користувач надсилає дані. Для цього зазвичай використовується заголовок Authorization.

Для перевірки авторизації доцільно використовувати стандартний та безпечний метод з генеруванням ключа. На певну адресу для запиту надсилаються дані (логін та пароль або публічний та секретний ключ), по цим даним генерується тимчасовий bearer ключ, який в подальшому використовується в кожному запиті для авторизації.

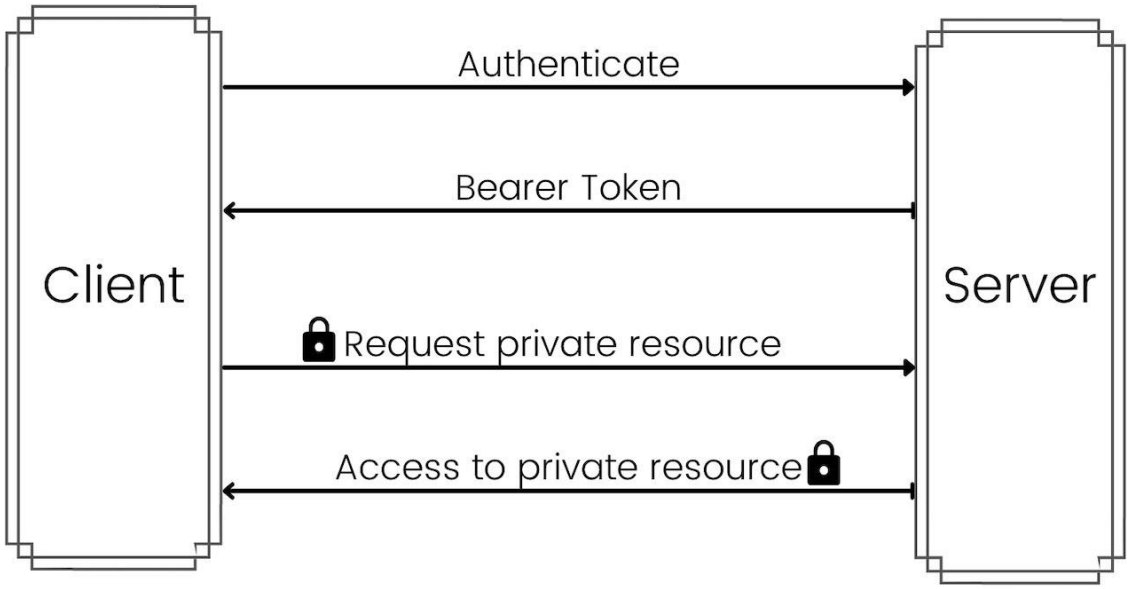


Рис. 2.5. Схема передачі даних в API

Також для API часто використовують заголовок з типом даних які повинен повернути сервер. Це робиться для додаткової перевірки та 100% збереження кодування при передаванні даних з одного пристрою на інший.

2.4.2. Відповіді HTTP сервера

Вся логіка роботи API запитів побудована на принципі запит-відповідь від сервера. При цьому, для відповідей використовуються стандартні коди та відповіді HTTP. Іноді, для більш детального пояснення, код помилки описується детальніше. Коди статусей відповідей представлені в наступним чином [11]:

- 1xx — інформаційний: запит прийнятий, продовжуй процес.
- 2xx — успіх: дія була успішно передана, зрозуміла, та прийнята.
- 3xx — перенаправлення: наступні дії мають бути успішно виконані для реалізації запиту.
- 4xx — помилка клієнта: запит містить синтаксичні помилки або не може бути виконаний.
- 5xx — помилка сервера: сервер не зміг виконати правильно сформований запит.

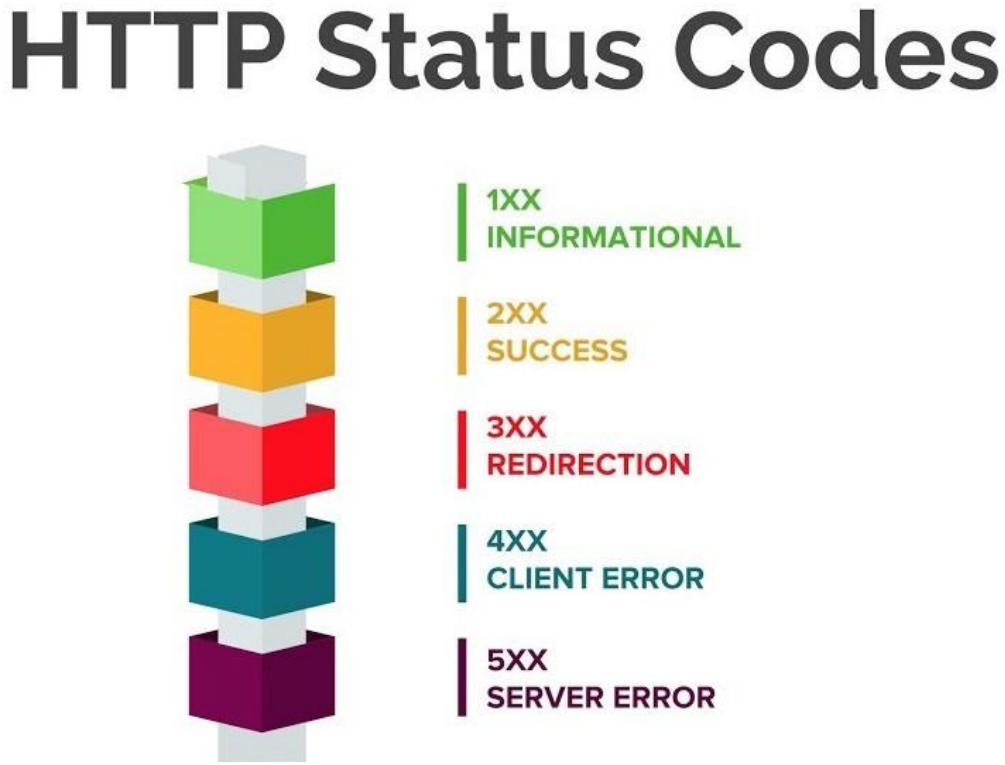


Рис. 2.6. HTTP відповіді серверу

2.5. POSTMAN

Основне призначення програми – створення колекцій із запитами до вашого API. Будь-який розробник або тестувальник, відкривши колекцію, зможе легко розібратися в роботі вашого сервісу. До того ж, Postman дозволяє проектувати дизайн API і створювати на його основі Mock-сервер [12]. Розробникам більше не потрібно витрачати час на створення "заглушок".

Реалізацію сервера та клієнта можна запустити одночасно. Тестувальники можуть писати тести та проводити автоматизоване тестування прямо з Postman. А інструменти для автоматичного документування з ваших колекцій заощадять час на ще одну "корисну фічу".

Головні поняття, якими оперує Postman це Collection (колекція) на верхньому рівні, та Request (запит) на нижньому. Вся робота починається з колекції та зводиться до опису вашого API за допомогою запитів. Схематично POSTMAN можна розділити на три складові – Collection, Folder, Request, які зображені на рис. 2.7.

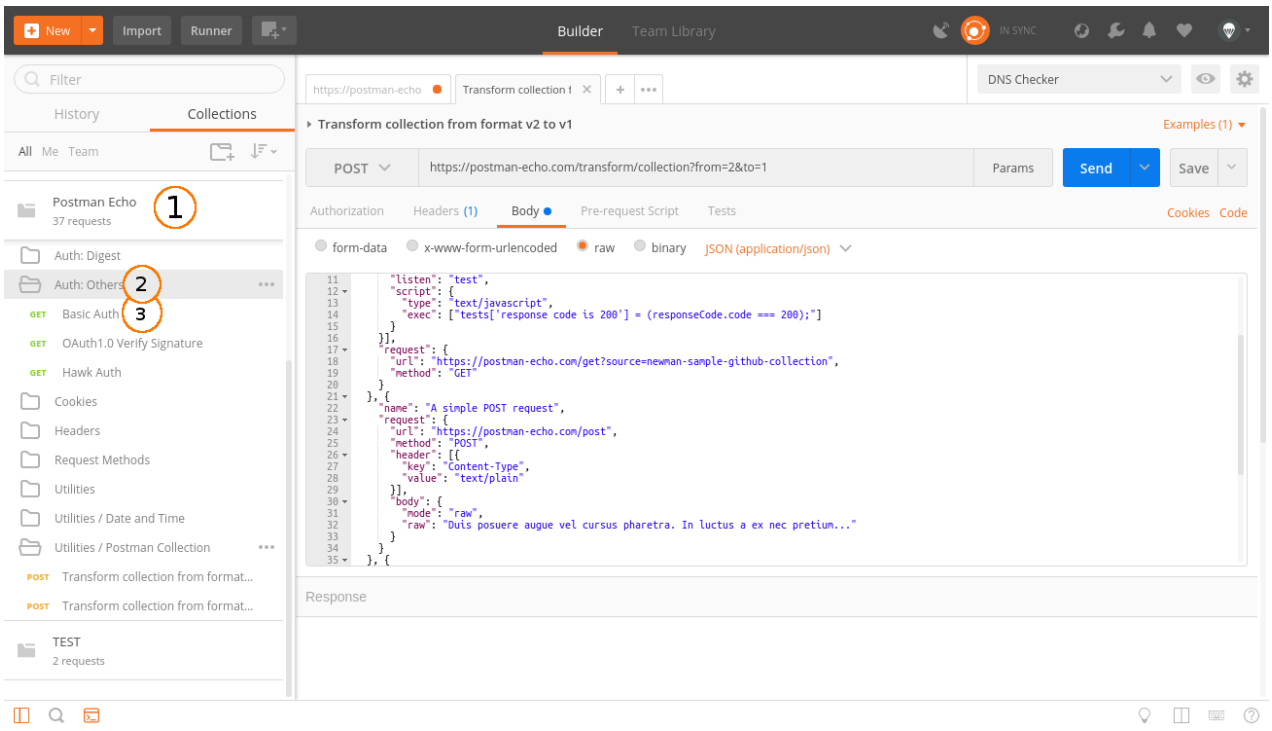


Рис. 2.7. POSTMAN ієрархія

2.5.1. POSTMAN Collection

Колекція – відправна точка для нового API. Можна розглядати колекцію як файл проекту. Колекція поєднує у собі всі пов'язані запити. Зазвичай API описується в одній колекції, але якщо ви бажаєте, то немає жодних обмежень зробити по-іншому. Колекція може мати свої скрипти та змінні.

2.5.2. POSTMAN Folder

Папка — використовується для об'єднання запитів до групи всередині колекції. Наприклад, можна створити папку для першої версії свого API – "v1", а всередині згрупувати запити за змістом виконуваних дій – "Order & Checkout", "User profile" тощо. Все обмежується лише фантазією та потребами. Папка, як і колекція, може мати свої скрипти, але не змінні.

2.5.3. POSTMAN Request

Запит – основна складова колекції, то заради чого все й починалося. Запит створюється у конструкторі. Конструктор запитів – це головний простір, з яким вам доведеться працювати. Postman вміє виконувати запити за допомогою всіх стандартних методів HTTP, всі параметри запиту під вашим контролем. Ви можете легко змінити або додати необхідні заголовки, cookie, і тіло запиту. Запит має свої скрипти. Вкладки "Pre-request Script" та "Tests" дозволяють додати скрипти перед виконанням запиту та після. Саме ці дві можливості роблять Postman потужним інструментом, що допомагає при розробці та тестуванні.

2.6. Документація API

Існує безліч варіантів представляти документацію в API. Найбільш поширеним способом є підключення сервісу API до онлайн тестувальної платформи SWAGGER [13], яка в свою чергу буде включати як теставання, так і онлайн документацію для всіх даних.

Інтерфейс Swagger надає Фреймворк, який читає специфікацію OpenAPI. і створює веб-сторінку з інтерактивною документацією.

Специфікація OpenAPI не залежить від мови. Також її можна поширювати на нові технології і не тільки HTTP протоколи.

З декларативною специфікацією ресурсу OpenAPI, клієнти можуть розуміти і використовувати сервіси без знання деталей реалізації сервера.

Swagger має свій синтаксис, код якого представлено наступним чином:

```
openapi: "3.0.0"
info:
  version: 1.0.0
  title: My API
servers:
  - url: http://myapi.mydomain/v1
paths:
  /data:
    get:
      summary: Get data
      responses:
        '201':
          description: Data response
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Data"
        default:
          description: unexpected error
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Error"
components:
  schemas:
    Error:
      required:
        - code
        - message
      properties:
        code:
          type: integer
          format: int32
        message:
          type: string
    Data:
      type: string
```

Однак документацію доцільно розробляти власну, статичну, на чистому HTML та CSS.

2.6.1. HTML

HTML (HyperText Markup Language — «мова гіпертекстової розмітки») — базовий будівельний блок Інтернету. Він визначає зміст та структуру веб-контенту [14]. Інші технології, крім HTML, зазвичай використовуються для опису зовнішнього вигляду/уявлення (CSS) або функціональності/поведінки (JavaScript) веб-сторінки.

HTML-елемент виділяється з іншого тексту в документі за допомогою "тегів", які складаються з імені елемента оточеного "<" та ">". Ім'я елемента всередині тега не чутливе до регістру. Тобто воно може бути написане у верхньому або нижньому регістрі, або змішане. Наприклад, тег <meta> може бути записаний як <Meta>, <META>, або будь-яким іншим способом.

HTML використовує розмітку ("markup") для відображення тексту, зображень та іншого контенту у веб-браузері. HTML-розмітка включає спеціальні "елементи", такі як <head>, <html>, <body>, <title>, <header>, <footer>, <section>, <p>, , < span>, <img>, <audio>, <canvas>, <nav>, <output>, <progress>, <video> та багато інших.

HTML впроваджує засоби для:

- створення структурованого документа шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше;
- отримання інформації зі Всесвітньої мережі через гіперпосилання;
- створення інтерактивних форм;
- включення зображень, звуку, відео та інших об'єктів до тексту.

2.6.2. CSS

Як і HTML, CSS насправді не може бути мовою програмування. CSS – це не мова розмітки – це мова таблиці стилів. Це означає, що він дозволяє

використовувати стилі вибірково до елементів у документах HTML. Наприклад, щоб вибрати всі елементи абзацу на сторінці HTML і змінити текст всередині них з чорного на червоний [15].

Вся структура називається набором правил (але часто для стислості "правило"). Відзначимо також імена окремих частин:

- Селектор (Selector). Ім'я HTML-елемента на початку набору правил. Він вибирає елемент(и) для застосування стилю. Для стилізації іншого елемента просто змініть селектор.
- Оголошення (Declaration). Єдине правило, наприклад, color: red; показує, які з властивостей елемента хочете стилізувати.
- Властивості (Properties). Способи, якими можна стилізувати певний HTML-елемент. У CSS ви вибираєте, які властивості ви хочете торкнутися у вашому правилі.
- Значення (Property value). Праворуч від властивості, після двокрапки, у нас є значення властивості, яке вибирає одну з безлічі можливих ознак для даної властивості.

При цьому існує певна структурна особливість для правил в CSS, а саме:

- Кожен набір правил (крім селектора) має бути обгорнутий у фігурні дужки ({}).
- У кожному оголошенні необхідно використовувати двокрапку (:), щоб відокремити властивість від його значень.
- У кожному наборі правил необхідно використовувати крапку з комою (;), щоб відокремити кожне оголошення від наступного.

Висновки до розділу 2

Оскільки задача розробки прикладного програмного інтерфейсу є актуальною, було прийнято рішення використовувати наступні технології:

1. Мова програмування PHP – основна мова для програмування бекенд частини інтерфейсу та всіх операцій на сервері.
2. Мова запитів SQL – мова для зв’язку з базою даних.

3. JSON – формат передачі даних між серверами.
4. HTTP запити GET, POST, PUT, DELETE – основні запити для синхронізації даних.
5. PHPStorm – основне середовище розробки яке включає в себе синтаксис всіх мов.
6. POSTMAN – програмне забезпечення для тестування запитів.
7. HTML + CSS – структуровані мови для створення документації.

При цьому, для вдалої та успішно розширюваної в майбутньому системи необхідно спочатку розробити архітектуру моделей, продумати логіку запитів – відповідей та затвердити можливі відповіді та помилки в результаті виконання запитів на сервер. Також важливим пунктом є використання мікросервісної архітектури, що дозволить обійти обмеження які висуває сам сервер, та застосовувати сучасні засоби, які не пов’язані напряму з сайтом.

Особливу увагу при розробці необхідно приділити методу та способу авторизації для збереження конфіденційності даних та, водночас, зручному підключенню без додаткових складностей.

Для створення документації – рентабельніше всього створити статичну сторінку з інформацією про всі можливі методи, запити, та особливості кожного з них. Для цього використати HTML, CSS, фреймворк BOOTSTRAP.

РОЗДІЛ 3.
РОЗРОБКА ПРИКЛАДНОГО ПРОГРАМНОГО ІНТЕРФЕЙСУ

3.1. Логіка роботи API

Загальна суть роботи API базується на отриманні відповіді, чи виконанні якихось дій при передачі даних за певним посиланням. Частіше всього це оновлення інформації, або ж читання вже існуючої. Для забезпечення безпеки та коректності запиту \ відповіді для всіх користувачів за тією ж адресою існує авторизація в системі. Якщо уявити шлях запиту до серверу, щоб отримати будь які дані, то він виглядатиме наступним чином.

1. Запит на отримання токену для авторизації
2. Відповідь серверу з авторизацією
3. Збереження токену в базі
4. Запит на отримання даних
5. Перевірка токену авторизації
6. Видача даних

Більш детально та наглядно цю схему можна зобразити наступним чином:

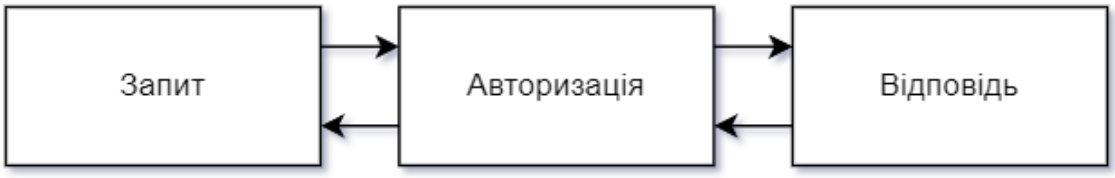


Рис.3.1. Діаграма по схемі роботи API

На перший погляд вся суть роботи є доволі простою, однак існує кілька глобальних проблем, основною з яких є безпека для передачі даних та можливість підбору токену для перехвату даних. Для цього кожен токен має бути захищеним та тимчасовим. Для збереження даних в світовій практиці прийнято рішення зберігати час роботи токену протягом деякого часу (від 5 хвилин до 7 діб), та дозволювати повторювати запит з існуючим токеном протягом цього часу, при цьому при кожному успішному запиті поновлювати таймер роботи токену. Такі дії переконують, що запит виконаний з того ж пристрою, та зменшує навантаження на систему.

3.2. Мікросервісна архітектура

Всі проекти можна розділити на два типи архітектури. Перший – монолітна, коли один додаток – це один сервер, та всі дії взаємопов'язані. Інший – це мікросервісна архітектура, яка означає що дії за окремий процес роботи з додатком виконує якийсь свій окреми сервіс який не обов'язково на тому ж сервері. Це вирішує проблему стійкості проекту при падінні будь якого з функціоналу (як наприклад пошук, база даних чи проблеми з сервером). Крім того, це дозволяє працювати одразу декільком командам над одним проектом, не створюючи конфліктних ситуацій в коді одними з іншими. В той же час, завдяки мікросервісній архітектурі на проекті одночасно може працювати декілька команд з різним стеком та на різних мовах програмування, оскільки по факту вони взаємодіють напряму з БД проекту, і повинні виконувати якісь функції, та повертати готовий резулаьтат. Для основного сервісу не важливо що і як відбувається всередині кожного процесу. На схемі, що на рис. 3.2. редставлено схематичне відображення відмінностей монолітної та мікросервісної архітектури.

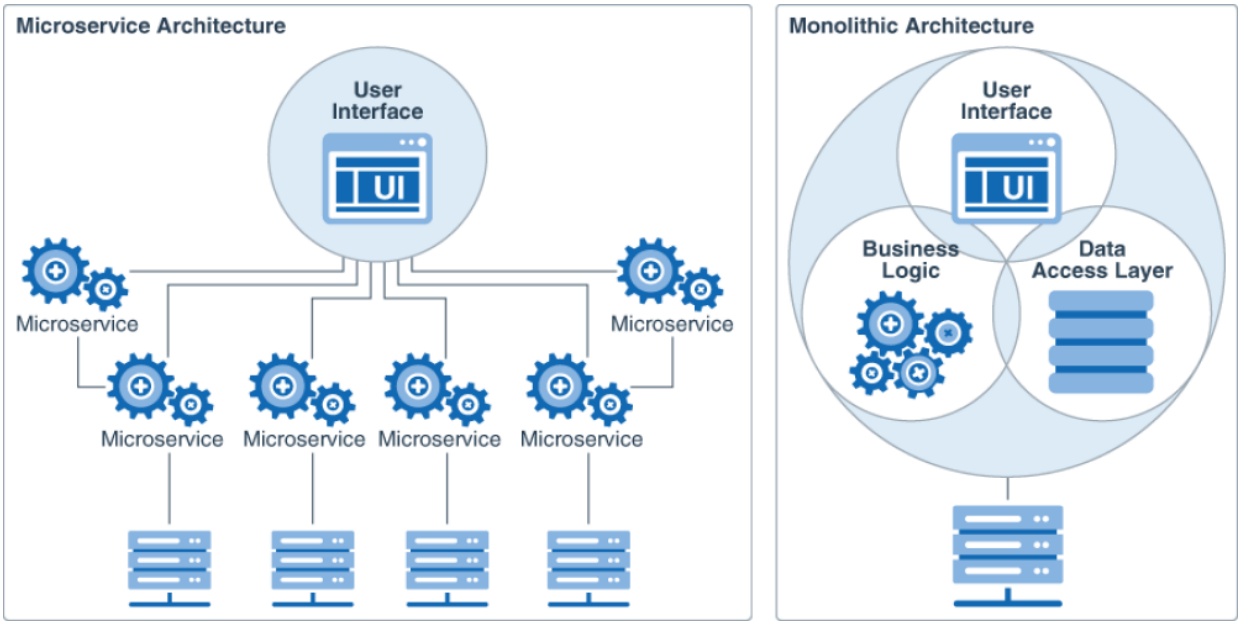


Рис. 3.2. Відмінність мікросервісної та монолітної архітектури

На проекті BEZET вже застосована мікросервісна архітектура, оскільки функціонал пошуку та адміністративної панелі винесено на окремі процеси та сервери. Для зручності подальної роботи, та незалежності API доцільно використовувати її на окремому сервері через ряд причин, серед яких:

1) Окремий лог запитів для безпеки
2) Зменшення навантаженості на основний веб сайт
3) Зручність в редагуванні незалежно від основного проекту
4) Незалежність від стилю коду основного сайту
5) Можливість використання нових технологій без рефактору основного коду

3.3. Клас API та базові функції

Весь проект реалізовано на MVC фреймворці, який був створений раніше. Загальна схема роботи MVC фреймворку представлена на рис. 3.3.

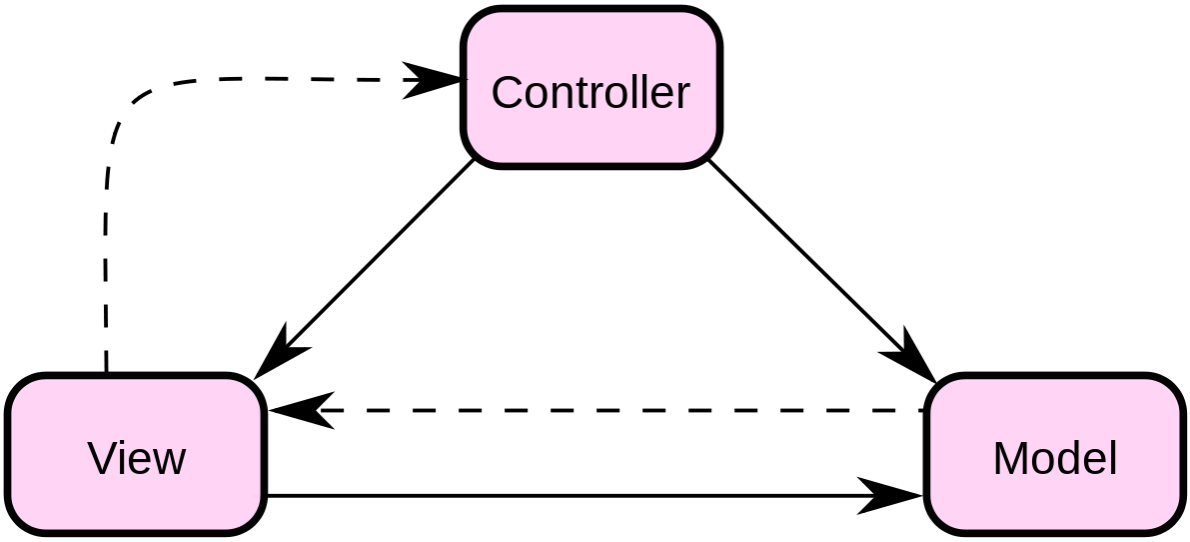


Рис. 3.3. MVC шаблон

Для перевірки арі запитів доцільно використовувати окремий підпроект який буде мікросервісом, або використовувати middleware, який буде перехвачувати запити ще до обробки основних контроллерів в додатку. Для розуміння який спосіб краще обрати – було створено таблицю переваг першого та другого методів, в якій по 10ти бальній шкалі визначено основні переваги в кожному з методів. В порівнянні – 0 – складно або не реалізуємо, а 10 – кращий варіант.

Таблиця 3.1.

Порівняння монолітної та мікросервісної архітектури

	Монолітна архітектура	Мікросервісна архітектура
Простота реалізації	10	5
Кросплатформенність	5	10
Відмовостійкість	3	10
Зручність розширення	5	9
Самостійність	0	10
Розподілене навантаження	0	10

В результаті порівняння – очевидно, що краще використовувати зовнішній сервер для API інтеграцій. Для цього було прийнято рішення створити api піддомен, де розвернути свій підпроект. При цьому, для розробки було прийнято рішення зберегти вже готовий фреймворк, і лише змінити логіку перегляду запитів. Якщо зазвичай шлях запиту перевірявся лише за посиланням, то в випадку API запитів основними критеріями є шлях та метод HTTP запиту. Саме в цьому буде основна відмінність в MVC фреймворку.

3.4. Отримання заголовків та стандартна архітектура

Для архітектури проекту було прийнято рішення створити два основних типи класів. Перший – це запит на авторизацію, який працює за POST запитом та отримує новий ключ для авторизації. Другий – це запит з будь яким заголовком та певним тілом чи без нього для отримання тих чи інших даних. Для цього в класі Router за допомогою базових функцій php отримується заголовок, який прийшов по цій адресі, та відповідно до адреси викликається відповідний клас та метод в ньому. Для такої обробки використовується наступна конструкція:

```
<?php
$method = $_SERVER['REQUEST_METHOD'];
$request = explode("/", substr(@$_SERVER['PATH_INFO'], 1));

switch ($method) {
  case 'PUT':
    do_something_with_put($request);
    break;
  case 'POST':
    do_something_with_post($request);
    break;
  case 'GET':
    do_something_with_get($request);
    break;
  default:
    handle_error($request);
    break;
      }
```

В першому рядку за допомогою суперглобальної змінної SERVER ми отримуємо метод який прийшов на запит. Далі завдяки стандартній конструкції switch визначаємо в який контроллер та який метод буде направлено даний запит.

3.5. Ключі авторизації

Оскільки основний проект вже існує, то для авторизації можна передбачити декілька шляхів.

1. Авторизація напряму за логіном та паролем.
2. Авторизація завдяки токену, який згенерований на основі логіна та паролю.
3. Авторизація завдяки токену, який згенерований на основі ключів.
4. Власний метод.

Перший метод вважають небезпечним та застарілим. Методи з авторизацією по логіну \ паролю та ключами є схожими, просто для них використовуються різні ключі. Єдиною перевагою використання ключів є відокремлення доступу до всього кабінету, а окремо від API, для залучання зовнішніх розробників. Цей метод широко використовується в сучасних банківських системах та еквайрінгах, тому прийнято рішення використовувати саме його.

Отож, перше що потрібно – це генерація публічних та приватних ключів для кожного користувача. Для того щоб їх згенерувати – для початку потрібне місце де їх зберігати. Для цього в базі даних додано дві колонки, public_key та secret_key. Це створено для додаткової безпеки.

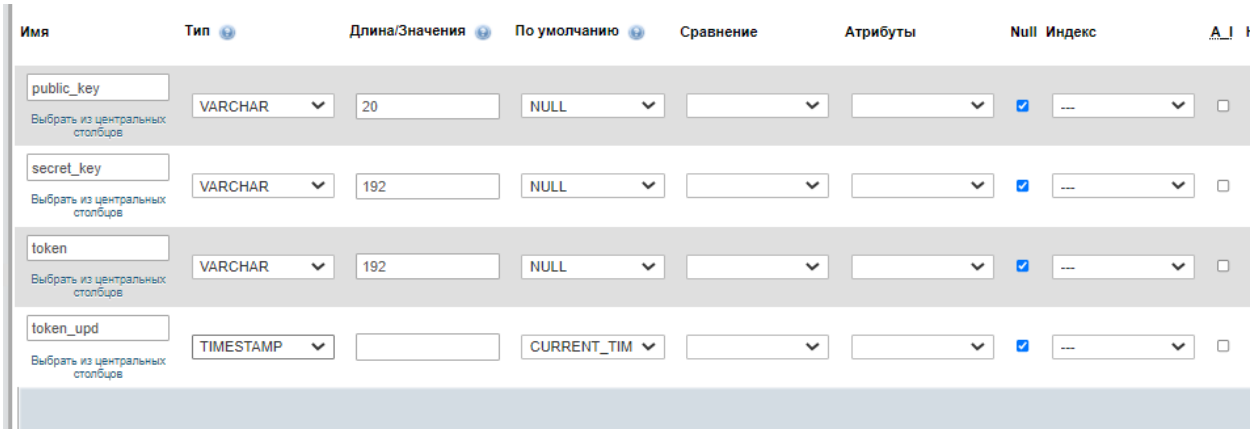


Рис. 3.4. Додавання стовпців з токеном

Після цього додано такі колонки як token, token_upd_at, які за замовченням дорівнюють NULL. Після цього в основній адміністративній панелі сайту створено кнопку «Генерация API», яка в свою чергу добавляє в базу даних два випадково згенерованих ключа. Ці ж ключі будуть використані для створення тимчасового токена.

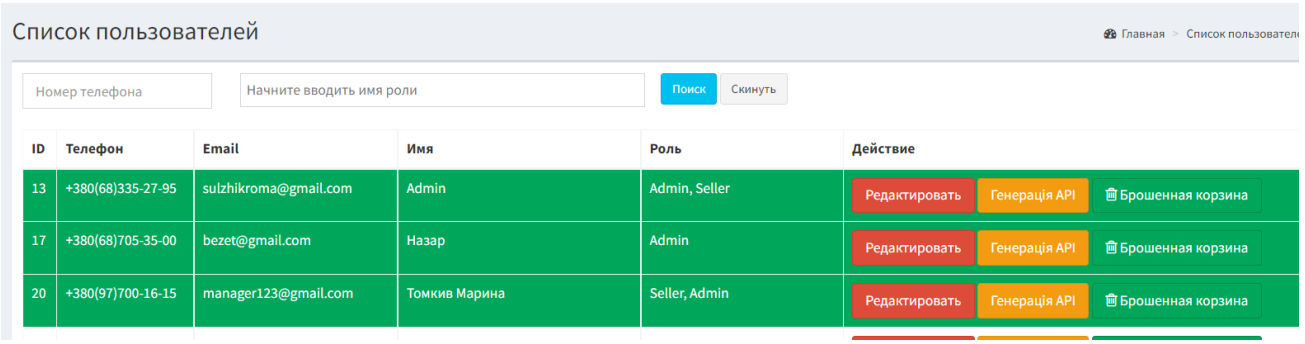


Рис. 3.5. Генерація ключів

Для самого створення ключів викликається існуюча функція generaterandomstrign для генерації двух рядків з довжиною в 10 та в 80 символів. Такий запис забезпечує унікальність, захищеність та випадковість створеного ключа. Код для генерації двух змінних представлено наступним чином:

```php
public function createtokenAction(){
    $user_id = $this->getRequestId();
    $pk = $this->generateRandomString();
    $sk = $this->generateRandomString( length: 80);
    \R::exec( sql: 'UPDATE user SET public_key=?, secret_key=? WHERE id=?',[$pk,$sk,$user_id]);
    $_SESSION['success'] = 'Ключи успешно сгенерированы'.
    redirect();
}

private function generateRandomString($length = 10) {
    $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
    $charactersLength = strlen($characters);
    $randomString = '';
    for ($i = 0; $i < $length; $i++) {
        $randomString .= $characters[rand(0, $charactersLength - 1)];
    }
    return $randomString;
}
```

Рис. 3.6. Генерація публічного та приватного ключа

Вся подальша робота з API інтерфейсом буде відбуватись на окремому домені, на окремому сервері, однак на тій же БД, оскільки дані беруться з основного проекту.

Запит на авторизацію повинен бути надісланий POST методом, оскільки він генерує або оновлює токен, та не є ідемпотентним. В відповідь на запит клієнт отримає або успішно згенерований токен, або відповідь з помилкою та описом. Отримати ключі для роботи API клієнт може в своєму робочому кабінеті.

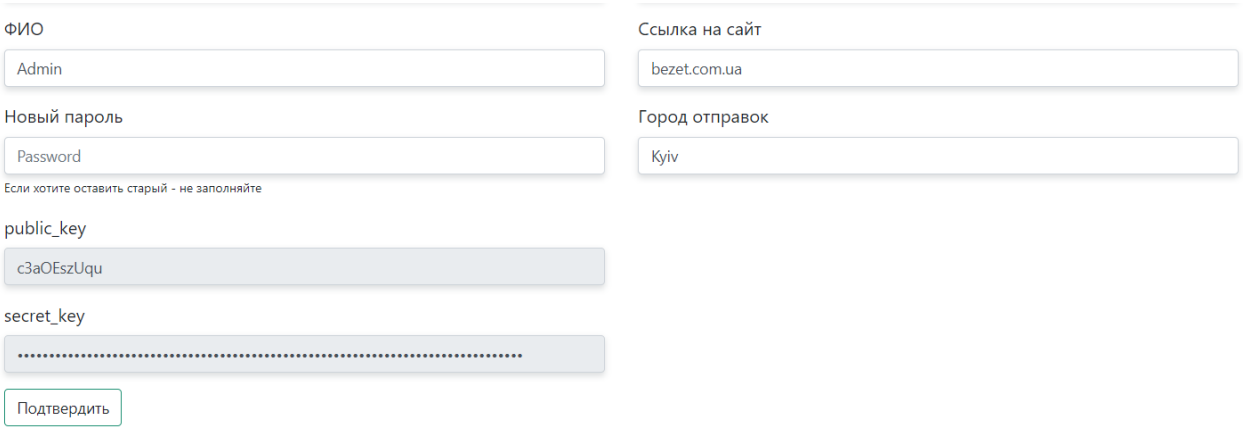


Рис. 3.7. Кабінет менеджера з ключами

3.6. Моделі для запитів

В готовій базі даних існує безліч системних чи адміністративних полів, які не потрібно віддавати клієнту при запиті. Для забезпечення корректної роботи всіх запитів та видачі тільки потрібної інформацій (чи валідації даних при створенні) можна винести всі потрібні запити в конструктори моделей, та додатково перевіряти кожен з запитів.

Оскільки першочерговою задачею є відображення замовлень та їх подальша обробка – очевидно, що перша модель буде “order”. В моделі присутні декілька методів, які відповідають за валідацію даних. Однак основне – це дані полів які потрібно отримувати з бази даних серверу для відповіді в API.

Всі функції в моделі повинні бути публічними, для можливості перевиклику їх в інших (майбутніх) моделях.

Для побудови всіх даних по замовленню потрібно згрупувати всі поля, які існують в кожному замовленні. Однак, замовлення поділяється на три різних таблиці і паралельно з цим кількість зв'язків може поширюватись з часом. Тому для кожної з підзадач створено свій запит. Фінальна структура замовлення та схематичне зображення відповіді преставлено на схемі ниже:

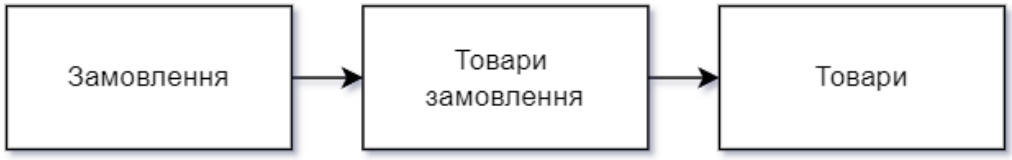


Рис.3.8. Схематичне зображення моделі замовлення

3.7. Створення мікросервісу та порожнього проекту

В зв'язку з тим, що маркетплейс – це високонавантажений проект та враховуючи порівняння, які вказані в таблиці 3.1. весь функціонал API винесено

на окремий підсервер на власний хостинг аккаунт. Для зв'язку з сервером додано доменне ім'я api.bezet.com.ua.

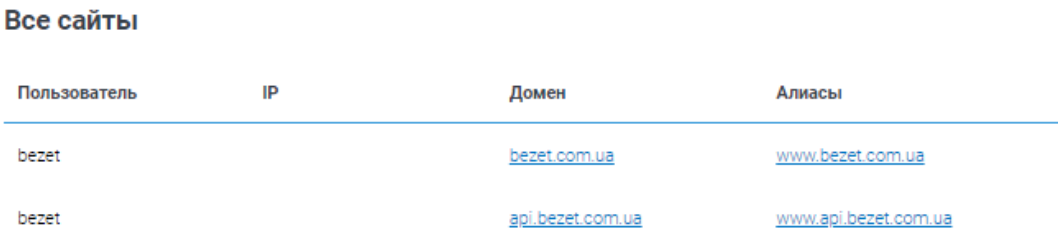


Рис. 3.9. Створення піддомену для api

Для початку роботи потрібно врахувати стандартні проблеми та засоби захисту до яких можна віднести наступні пункти:

Загальна логіка роботи прикладного інтерфейсу включає в себе ще й документацію для розробників, яка буде доступна за адресою пустого запиту, тобто GET запит на доменне ім'я повинне буде переадресовувати на документацію з всіма даними.

Після створення порожнього проекту та стандартного “HW!” для перевірки працездатності серверу через FTP клієнт на сервер занесено MVC фреймворк, який в свою чергу за GET запитом на корінний домен відображає інформацію «HW MVC!» що зображено на рис. 3.10., що означає що проект успішно запущено та розгорнуто.

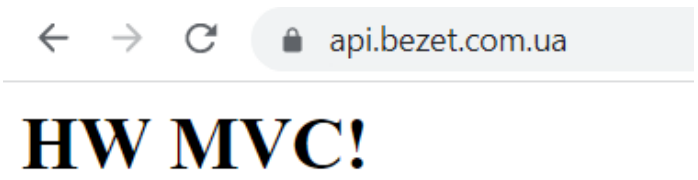


Рис. 3.10. Демонстрація працездатності серверу

Після того як проект успішно запущений та розгорнутий на сервері – для зручності подальної роботи з кодом – прийнято рішення створити локальну копію проекту за допомогою IDE PHPStorm, та працювати одразу на основній версії API сайту напряму з локального пристрою. В IDE це має наступний вигляд:

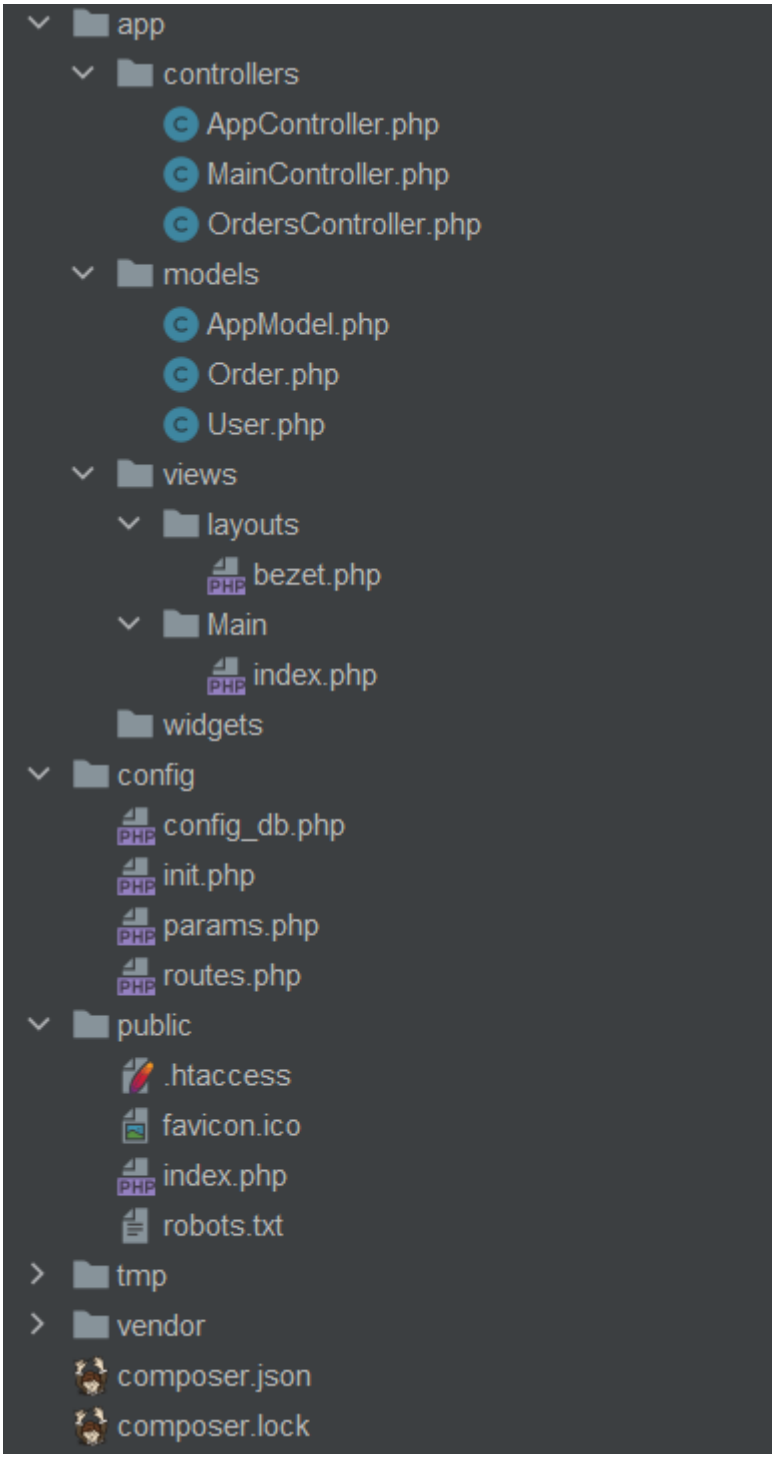


Рис. 3.11. Структура API BEZET в IDE PHPStorm

Оскільки сам фреймворк взятий з основного сайту BEZET, то його структура повністю аналогічна. Почистивши від зайвих контроллерів, моделей та видів структура проекту має простий та очевидний вигляд, в якій по назвах файлів очевидно за який пункт відповідає який з файлів.

3.8. Клас ROUTER

Маючи чітке розуміння логіки обробки запитів прийнято рішення переписати клас ROUTER, який відповідає за виклик тої чи іншої функції, в залежності від HTTP header method. Для цього створено функцію, яка зображена на рис. 3.12.

```
if(class_exists($controller)){
    $controllerObject = new $controller(self::$route);
    $action = 'Action'.self::upperCamelCase($_SERVER['REQUEST_METHOD']);
```

Рис. 3.12. Зміна методу Action

Функція змінює метод Action який викликається в контролері на метод, який буде працювати за типом HTTP запиту. Таким чином, обробка запиту за посиланням “/controller” буде відбуватись наступним чином:

1. Перевірка існування контроллера
2. Перевірка існування функції ActionHTTPMETHOD, де замість HTTPMETHOD буде підставлено значення методу (GET \ POST \ PUT \ DELETE).
3. Виклик відповідної функції та її обробка.

3.9. Клас запитів

Перед початком розробки класу створено схему, яка по факту відображає в nocode стилі всю логіку та архітектурну складову типового класу запитів. Схема створена на основі загальносвітової практики використання api.

1. GET – /orders – відображення всіх замовлень (з врахуванням умов пагінації)
2. GET – /orders/{id} – відображення замовлення з №{id}
3. POST – /orders – вставка нового замовлення
4. PUT – /orders/{id} – оновлення даних про замовлення з №{id}
5. DELETE – /orders/{id} – видалення замовлення з №{id}

Маючи всі базові методи та дані створено функції, які зображено на рис. 3.10:

1. ActionGET – включає в себе отримання всіх замовлень чи одного.
2. ActionPOST– додавання нового замовлення
3. ActionPUT– оновлення існуючого замовлення
4. ActionDELETE– псевдовидалення існуючого замовлення

```php
<?php
    namespace app\controllers;
    use bezet\Cache;
    class OrdersController extends AppController{

        public function ActionGET(){
            echo('get');
            die();
        }

        public function ActionPOST (){
            echo('insert');
            die();
        }

        public function ActionPUT(){
            echo('update');
            die();
        }

        public function ActionDELETE(){
            echo('delete');
            die();
        }

    }
?>
```

Рис. 3.13. Оновлений OrdersController на основі HTTP методів

3.9.1. Запит авторизації

Для роботи будь якого з запитів повинна спрацьовувати авторизація. Авторизація включає в себе лише перевірку токену для користувача. Для генерації токену використовується POST запит на посилання auth. Це означає, що буде викликаний контроллер AuthController з методом ActionPOST, який повинен приймати в json форматі лише два значення – public key та secret key. В випадку їх відповідності – незалежно від існування токену повинен бути згенерований новий.

Базова функція контроллеру включає в себе отримання всіх даних які прийшли в тілі POST. JSON декодування даних в масив. Звірку даних в БД про існування таких ключів. В випадку успішного існування – генерація токену та відповідь, в іншому ж випадку – виведення 405 помилки, що означає невірну авторизацію. В коді це виглядає наступним чином:

```
public function ActionPOST (){
    $data = json_decode(file_get_contents( filename: 'php://input'), associative: true);
    if(count($data)==2){
        $pk = htmlspecialchars($data['pk']);
        $sk = htmlspecialchars($data['sk']);
        $user = \R::findOne( type: 'user', sql: 'public_key=? AND secret_key=?',[$pk,$sk]);
        if($user){
            $token = $this->generateRandomString( length: 150);
            $user->token = $token;
            $user->token_upd = date( format: 'Y-m-d H:i:s');
            \R::store($user);
            echo($token);
            http_response_code( response_code: 200);
            die();
        } else {
            echo ('User not found');
            http_response_code( response_code: '405');
            die();
        }
    } else {
        echo('You need to send 2 fields. PK and SK');
        http_response_code( response_code: '405');
        die();
    }
    die();
}
```

Рис. 3.14. AuthController – генерація токену

Для забезпечення безпеки та очевидності запитів, в усіх запитах крім POST виведено повідомлення про помилку, та потрібний метод авторизацій.

```
    }

    public function ActionPUT(){
        echo('Only POST method allowed');
        http_response_code( response_code: '400');
        die();
    }

    public function ActionDELETE(){
        echo('Only POST method allowed');
        http_response_code( response_code: '400');
        die();
    }
```

Рис. 3.15. Помилка при не дозволеному методі

Для перевірки роботи генерації коду можна використати застосунок postman, який дозволяє надсилати запити будь якого типу з свого зручного інтерфейсу. Для прикладу, даний запит виглядатиме наступним чином:

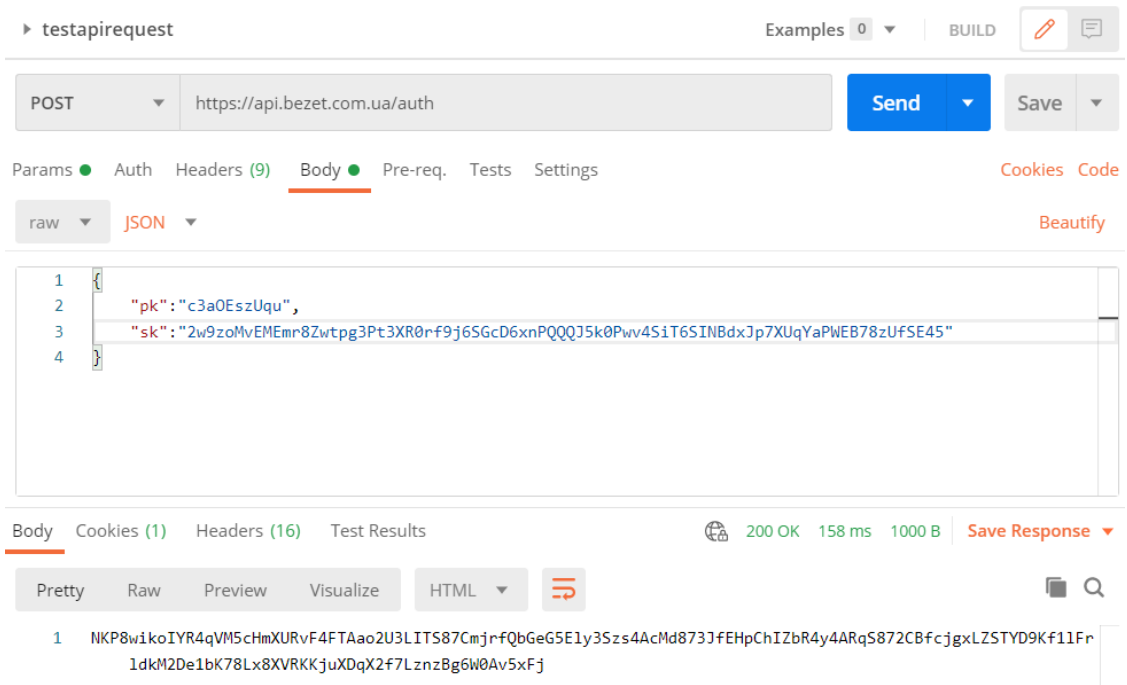


Рис. 3.16. Postman запит авторизації

Як видно з скриншоту, при передачі даних публічного та приватного ключа згенерувався новий токен, який повернувся в відповідь. Цей токен і буде в подальшому використовуватись при всіх запитах. В випадку, якщо метод відправки не POST, буде повідомлення, яке відображено на рис. 3.17.

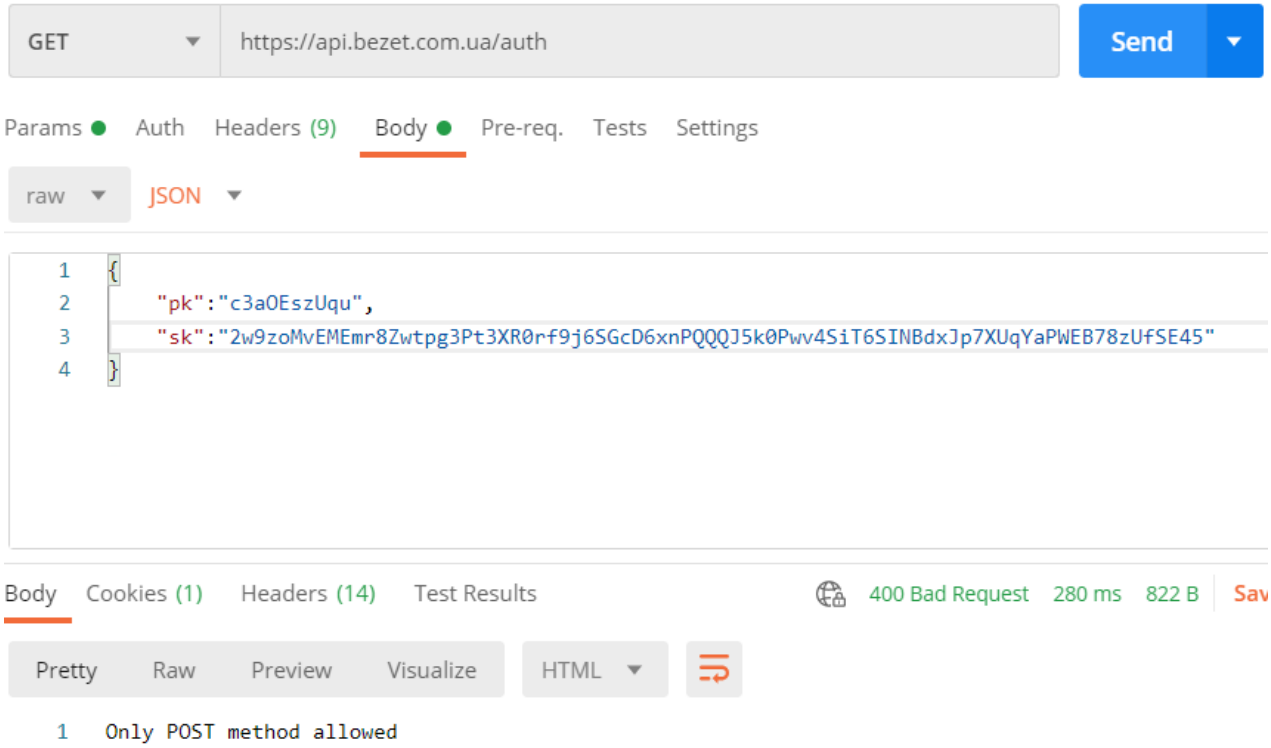


Рис. 3.17. Заборона методу GET

Кожен з наступних методів повинен перевіряти існування токену. Для цього в моделі USER можна створити метод, який виконує перевірку даних, та повертає користувача який активний під токен який в запиті. Завдяки тому, що цей код буде в моделі, його можна викликати з будь якого методу в будь який момент, та він повинен бути першим. Дана функція буде або повертати об'єкт з користувачем, або повністю завершувати виконання коду. В коді це виглядає як на рис. 3.18.

```php
public function getUserByToken(){
    $user = '';
    if(isset($_SERVER['HTTP_TOKEN'])){
        $user = \R::findOne( type: 'user', sql: 'token=?',[$_SERVER['HTTP_TOKEN']]);
        if(!empty($user)&&$user≠null) {
            return $user;
        } else {
            echo('Token not exist');
            http_response_code( response_code: 405);
            die();
        }
    } else {
        echo('None header token');
        http_response_code( response_code: 405);
        die();
    }
}
```

Рис. 3.18. Функція перевірки токену

Для перевірки можна виконати тестовий GET запит на отримання замовлень, в самій же функції виведення замовлень потрібно зробити перевірку на існування токену, та вивести будь яке поле (наприклад ім'я) в відповідь. Відповідь сервера зображена на рис. 3.19.

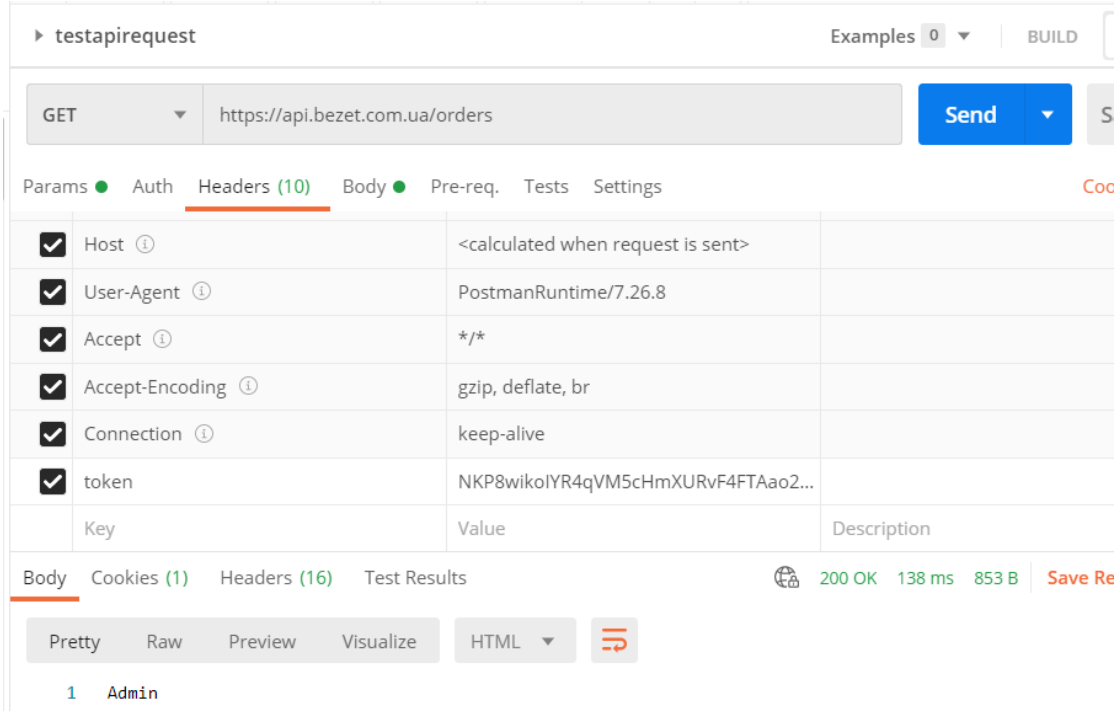


Рис. 3.19. Перевірка запиту з токеном та отримання користувача

3.9.2. Запит отримання даних

Задачу з отримання даних можна умовно розділи на три частини.

1. Перевірка авторизації та отримання користувача.
2. Отримання всіх замовлень по користувачу.
3. Отримання даних про товари в замовленнях.

Перший пункт реалізується за допомогою вже готового методу перевірки токену getUserByToken. Метод повертає об'єкт користувача, який в подальшому і буде використовуватись.

Другий пункт можна виконати за допомогою SQL запиту, однак, необхідно вибрати які саме дані потрібно отримувати з бази даних. Проаналізувавши таблицю потрібні дані виглядають наступним чином:

- id – номер замовлення
- user_tel – номер телефону замовника
- user_name – ім'я замовника
- status – статус замовлення відносно системи
- date – дата та час замовлення
- note – коментар до замовлення
- city – місто для відправки
- warehouse – відділення для відправки
- track – номер накладної в системі
- typedel – вид поштової служби
- call – чи потрібно перетелефонувати клієнту
- confirmed – статус підтвердженого замовлення від менеджера

При цьому необхідно врахувати, що замовлень може бути багато, тому для початку потрібно вивести останні 100 записів по фільтрації. В результаті – функція отримання даних матиме вигляд, який зображено на рис. 3.20.

Рис. 3.20. Запит отримання замовлень

Очевидно, що відповідь яка прийде клієнту буде недостатньої для повного оформлення замовлення, адже в ній немає інформації про товари та загальної суми. Для того щоб поєднати дві таблиці та отримати коректні дані – можна використати фукнцію INNER JOIN, яка поєднає таблиці order та order_product, та надасть потрібний результат. Однак в цього методу є мінус – як дублювання більшості даних (загальних даних про замовлення) в кожному результаті. Для уникнення такого функціоналу – можна до кожного замовлення в циклі отримувати всі товари. Однак таке рішення теж є поганим, по причині високого навантаження на сервер.

Рішення цієї проблеми можна вирішити наступним чином. На стороні серверу можна отримати список ID замовлень, для яких потрібно завантажити товари. Після чого, зробити один запит, в якому вказати всі ці ID, який поверне масив об'єктів. На стороні серверу поєднати замовлення та товари які з ними пов'язані. Таким чином буде виконано лише два запита до бази даних, що значно спростить навантаження на mysql сервіс та прискорить роботу програмного інтерфейсу. Програмна черга функції матиме наступний вигляд: отримання замовлень – отримання списку ID замовлень – отримання товарів до замовлень зі списку – поєднання замовлень та товарів – виведення json відповіді. На мові програмування php уривок цієї функції представлени на рис. 3.21.

```php
$idsoforders = [];
foreach($orders as $order){
    $idsoforders[] = $order['id'];
}
$idsofordersstr = implode( separator: ',',$idsoforders);
$order_products = \R::getAll( sql: 'SELECT
`order_id`,`product_id`,`qty`,`title`,`size`,`price`
FROM `order_product`
WHERE order_id IN ('.$idsofordersstr.')');

foreach($orders as $k=>$order)
    foreach($order_products as &$op){
        if($op['order_id']==$order['id']){
            $orders[$k]['products'][] = $op;
        }
    }
echo(json_encode($orders));
```

Рис. 3.21. Поєднання таблиць за допомогою двух запитів

3.9.3. Запит додавання даних

Для створення замовлень та додавання їх в базу даних спочатку створено модель Order, з масивами атрибутів, дані які необхідно отримувати та вставляти. В коді це представлено наступним чином:

```php
class Order extends Appmodel{

    public $attributes = [
        'user_tel' => '',
        'user_name' => '',
        'status' => '',
        'date' => '',
        'note' => 'user',
        'city' => '',
        'warehouse' => '',
        'track' => '',
        'typedel' => '',
        'call' => '',
        'confirmed' => '',
    ];

    public $attributes_product = [
        'order_id'=>'',
        'product_id'=>'',
        'qty'=>'',
        'title'=>'',
        'size'=>'',
        'price'=>'',
    ];
}
```

Рис. 3.22. Модель Order

Завдяки цій моделі можна створювати екземпляр класу замовлення та напряму працювати з ним. Для збереження даних потрібно проводити наступну чергу функцій:

1. Перевірка токену авторизаці
2. Збереження всіх потрібних даних в відповідні змінні
3. Валідація даних для збереження
4. Збереження замовлення
5. Збереження товарів до замовлення
6. Виведення номеру замовлення в відповідь на запит

При збереженні даних в відповідь користувач отримає номер створеного замовлення, який є унікальним. Також після збереження замовлення в базі даних – будуть запущені додаткові сервіси (як смс сповіщення чи списання балансу комісії з користувача), однак ці функції вже реалізовані на основному веб додатку, отже створювати їх заново не потрібно. За допомогою веб сервісів та хуків з callback фукнціями дані будуть надіслані, а функції автоматично виконані в фоновому режимі.

Перший пункт валідації вже використовувався в запиті отримання даних і виглядає аналогічно. Отримання користувача за даними токену, або виведення помилки.

Другий пункт можна розбити на етапи, а саме: отримання всіх даних, збереження даних в змінній замовлення, збереження даних в змінній товару. Для збереження даних – спочатку їх потрібно розділити. Зроблено це завдяки вкладеності масиву та документації до запиту на додавання даних. Запит повинен включати всі поля з сутності замовлення, а потім масив товарів, який спочатку винесено в окрему змінну та видалено з основних даних. Це зроблено для зручної роботи з кожною сутністю окремо. В коді це представлено як на рис. 3.23.

```php
if(!isset($data['products'])){
    echo('none products');
    http_response_code( response_code: 400);
    die();
} else {
    $orderdataproducts = $data['products'];
    unset($data['products']);
}
```

Рис. 3.23. Розділення товарів та даних про замовлення

Наступним кроком є завантаження даних в об'єкт замовлення. Для цього можна використати вже готову функцію з основної моделі, яка має назву load, та записує всі дані з масиву який приходить в існуючий об'єкт за умови існування поля. В коді це виглядає наступним чином:

```php
public function load($data){
    foreach($this->attributes as $name => $value){
        if(isset($data[$name])){
            $this->attributes[$name] = $data[$name];
        }
    }
}
```

Рис. 3.24. Завантаження даних в об'єкт моделі

Для логічного завершення збереження замовлення, потрібно провести валідацію даних. Для цього можна використати функцію валідації, яка вже є в MVC фреймвору, та отримувати помилки за допомогою іншої існуючої функції. Однак, для валідації спершу створино правила, які представлені на рис. 3.25.

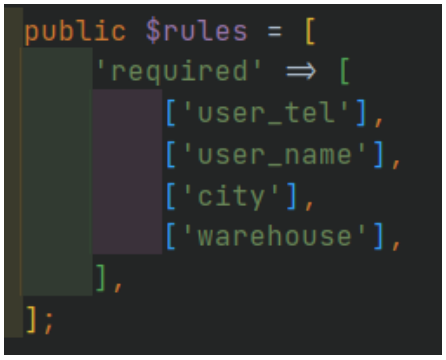


Рис. 3.25. Правила для полів замовлення

Вся перевірка правил обмежена лише існуванням обов'язковості поля, оскільки всі подальні перевірки вже вбудовано в ORM систему readbean і будуть проведені автоматично (як відсутність букв в номері телефону). Сама функція валідації даних з об'єкту замовлення представлена на рис. 3.26.

Рис. 3.26. Валідація даних замовлення

Після пройдення всіх перевірок необхідно зберегти дані в базі даних та повернути ID для подальшої обробки та збереження даних про товари в замовленні. Для цього використана вже готова функція save, яка виконує в собі три підфункції, відкриває таблицю яка передана, заповнює дані в ній, зберігає та повертає повний об'єкт який збережено. Функція має наступний вигляд:

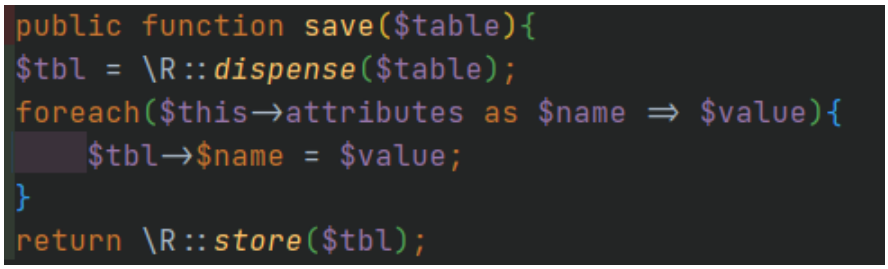


Рис. 3.27. Збереження даних в таблиці

Інформація про збереження даних про товари створена аналогічно до замовлення. Однак, потрібно додати до масиву даних значення з ID замовлення яке отримано в попередньому кроці. Результатуючий код преставлено на рис. 3.28.

```php
public function ActionPOST (){
    $user = new User();
    $userdata = $user->getUserByToken();
    $data = json_decode(file_get_contents( filename: 'php://input'),
    $orderdata = new Order();
    $orderdatapr = new Order();

    if(!isset($data['products'])){
        echo('none products');
        http_response_code( response_code: 400);
        die();
    } else {
        $orderdataproducts = $data['products'];
        unset($data['products']);
    }

    $orderdata->load($data);

    if(!$orderdata->validate($data)){
        $orderdata->getErrors();
        http_response_code( response_code: 400);
        die();
    }

    $orderdata->attributes['date'] = date( format: 'Y-m-d H:i:s');
    $orderdata->attributes['brand'] = $userdata['brand'];
    $order = $orderdata->save( table: 'order');

    $orderdatapr->load($orderdataproducts);

    $orderdatapr->findIdProductsBySku();

    if(!$orderdatapr->validate($orderdataproducts)){
        $orderdatapr->getErrors();
        http_response_code( response_code: 400);
        die();
    }

    $order = $orderdata->save( table: 'order');

    echo($order);
    die();
```

Рис. 3.28. Збереження даних товарів до замовлення

Для перевірки правильності роботи API можна знову зкористатись POSTMANом. Отримавши токен на попередніх кроках можна створити замовлення дотримуючись правил, які згодом будуть представлені в офіційній документації до BEZET API v.1.0. До цих правил належать деякі обов'язкові поля, типи полів та значення які можуть передаватись в них (якщо тип поля select). Знаючи всі базові правила – можна створити замовлення. Всі дані повинні надсилатись в вигляді JSON об'єкту. Тіло запиту виглядає наступним чином:


```
{"user_tel":"380631234567","user_name":"\u0421\u0443\u043b\u044c\u0436\u0438\u043a
\u0420\u043e\u043c\u0430\u043d","city":"\u041a\u0438\u0457\u0432","wareho
use":"336","products":[{"sku":"0257","title":"\u0428\u0410\u041f\u041a\u0410 BEZET BLACK","price":199},{"sku":"0651","title":"\u0411\u0430\u0444 BEZET \u0447\u043e\u0440\u043d\u0438\u0439","price":240}]}
```


При цьому кодування запиту змінюється на серверне, тому замість кирилиці в запиті ми бачимо закодований JSON URL текст. Запит з таким тілом повинен створити замовлення з двома товарами: “Шапка BEZET black” та “Баф BEZET чорний”. Для зв'язку товарі – пошук відбувається за полем артикул (sku) напряму в базі маркетплейсу при збережнні даних, а потім поєднуться по id товару. Виконавши такий запит через POSTMAN в відповідь отримуємо номер замовлення 848787 (див. рис. 3.29), що вказує, що всі дані успішно пройдено, замовлення створено та відображається в адміністративній панелі сайту. Для того щоб переконатись в тому, що всі дані збережено так як цього хотілось можна перейти в особистий кабінет менеджеру, та переглянути замовлення звідти. Скриншот з кабінету з поточним замовленням представлений на рис. 3.30. Переконавшись, що дані співпадають можна переходити до наступного кроку на редагування даних.

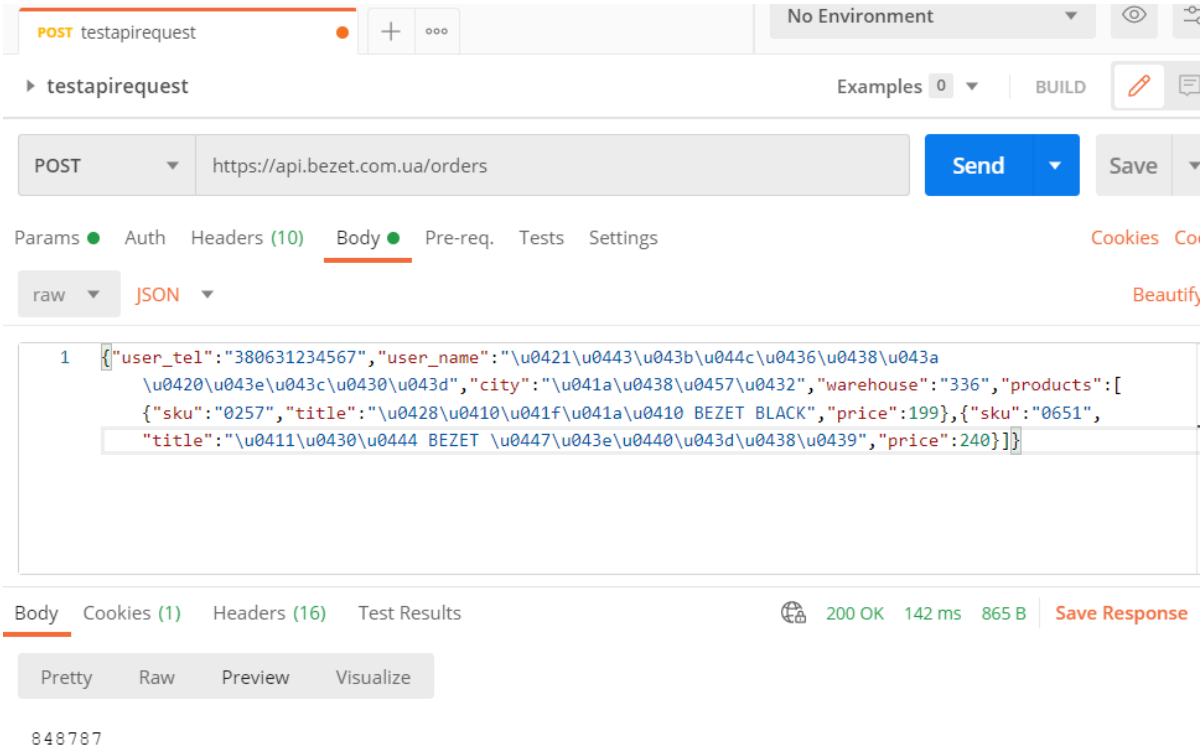
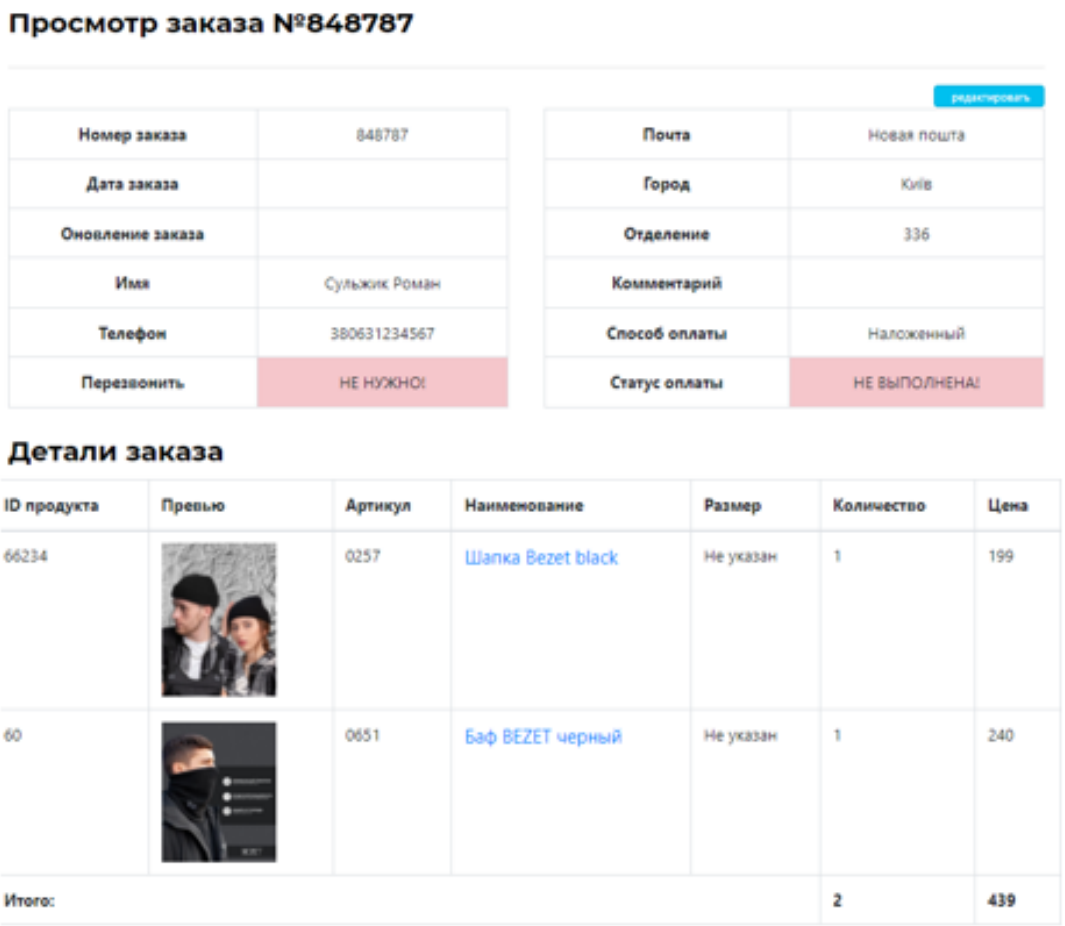


Рис. 3.29. POSTMAN запит на створення замовлення.



Просмотр заказа №848787

Номер заказа	848787	**Почта**	Новая пошта
Дата заказа		**Город**	Київ
Оновление заказа		**Отделение**	336
Имя	Сульжик Роман	**Комментарий**	
Телефон	380631234567	**Способ оплаты**	Наложенный
Перезвонить	НЕ НУЖНО!	**Статус оплаты**	НЕ ВЫПОЛНЕНА!

Детали заказа

ID продукта	Превью	Артикул	Наименование	Размер	Количество	Цена
66234		0257	Шапка Bezet black	Не указан	1	199
60		0651	Баф BEZET черный	Не указан	1	240
Итого:					**2**	**439**

Рис. 3.30. Замовлення з API в кабінеті маркетплейсу

3.9.4. Запит редагування даних

Редагування даних частково схоже з створенням початкового замовлення, однак існують певні відмінності, як в логіці, так і в коді. В логіці – даних яких не існує в запиті не повинні бути очищені та встановлені на стандартні. В запит додано поле з існуванням замовлення та перевірка даних на існування поточного замовлення та відповідність замовлення до даного аккаунту. Після оновлення даних повинне відбуватись перезапис всіх товарів які були в замовленні. Тобто, функція не матиме формату додавання чи віднімання наявності в товарі, а лише встановлення вже готової кількості.

Отже, перша функція з отриманням токену, перевіркою його, отриманням користувача та виведення помилки аналогічна як в попередньому способі. Далі потрібно розділити дані на замовлення то товари, що теж аналогічно методу додавання даних. Після розподілення потрібно перевірити існування поточного замовлення та відповідність номеру цього замовлення до користувача який надсилає дані. Наступним кроком є отримання даних про замовлення та завантаження прийдених даних в ньому. При цьому не передані дані не будуть перезаписані. Після цього відбувається видалення всіх товарів які прив'язані до товару, та по аналогії з п. 3.9.3. додаються товари до замовлення, ніби їх там і не існувало.

Для передачі даних використовується майже аналогічний PUT запит як зі створенням, однак в ньому додається поле id для розуміння яке замовлення буде редагуватись. Тіло запиту матиме наступний вигляд:


```
{"id":848787,"user_tel":"380631234567","user_name":"\u0421\u0443\u043b\u044c\u0436\u0438\u043a
\u0420\u043e\u043c\u0430\u043d","city":"\u041a\u0438\u0457\u0432","warehouse":"336","products":[{"sku":"0257","title":"\u0428\u0410\u041f\u041a\u0410 BEZET BLACK","price":199}]}
```

Для зміни типу запитів в POSTMAN можна використовувати готовий функціонал з dropdown меню запитів. Вибрати готовий можна в вікні, яке зображено на рис 3.31.

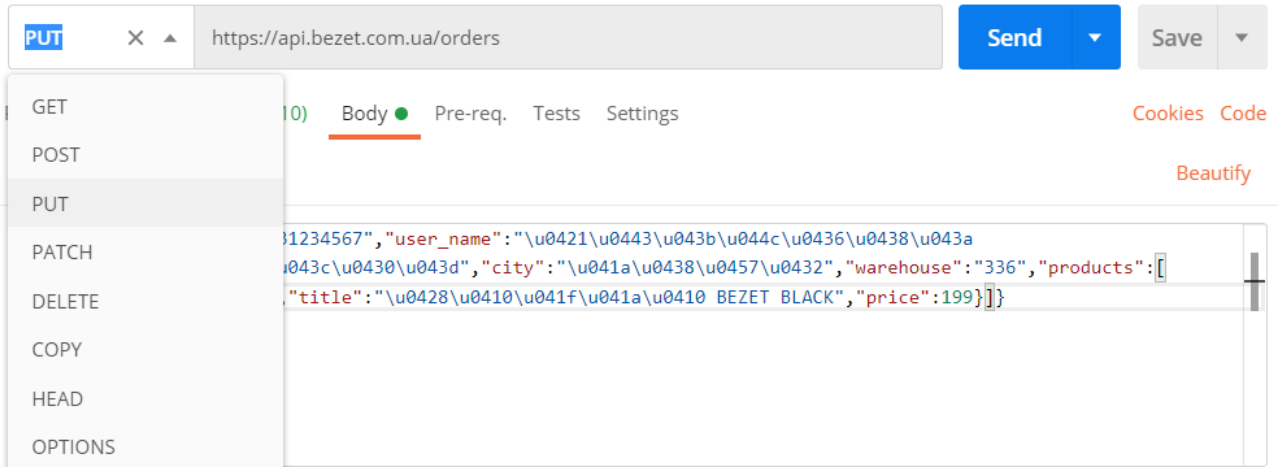
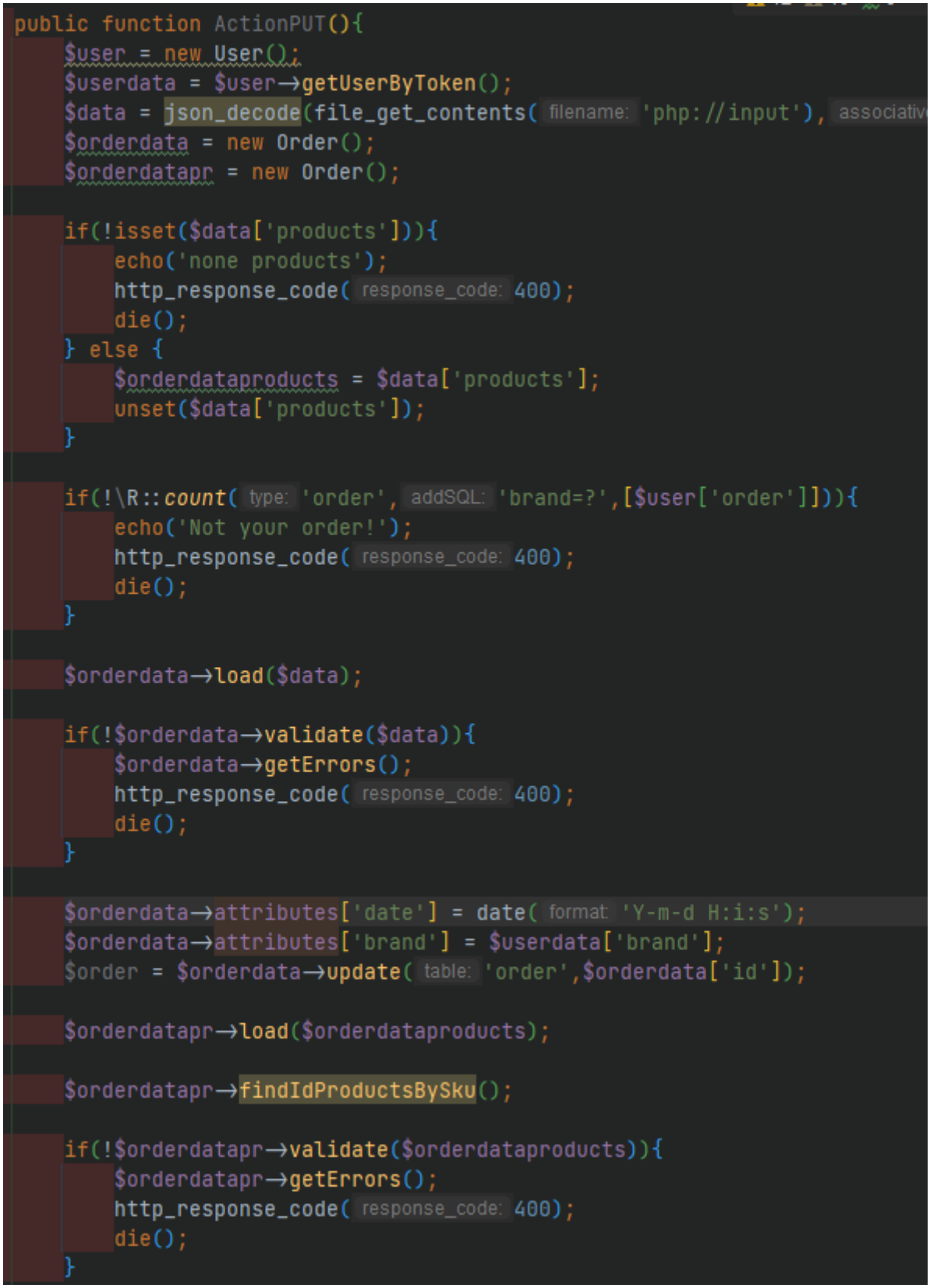


Рис. 3.31. POSTMAN зміна HTTP method

Всі подальші дії з замовленням будуть оброблятись на бекенді основного проекту, для цього при оновленні даних в БД викликається функція (on_update), яка в свою чергу викликає чергу з функцій для оновлення балансу, списання комісії, сповіщення користувача про зміну даних на сайті.

В запиті для перевірки існує тільки один товар, щоб переконатись в коректності роботи видалення товарів. Весь код обробки оновлення представлений на рис. 3.32, а результат оновлення в маркетплейсі – на рис. 3.33.



```
public function ActionPUT(){
    $user = new User();
    $userdata = $user->getUserByToken();
    $data = json_decode(file_get_contents( filename: 'php://input'), associativ
    $orderdata = new Order();
    $orderdatapr = new Order();

    if(!isset($data['products'])){
        echo('none products');
        http_response_code( response_code: 400);
        die();
    } else {
        $orderdataproducts = $data['products'];
        unset($data['products']);
    }

    if(!\R::count( type: 'order', addSQL: 'brand=?',[$user['order']])){
        echo('Not your order!');
        http_response_code( response_code: 400);
        die();
    }

    $orderdata->load($data);

    if(!$orderdata->validate($data)){
        $orderdata->getErrors();
        http_response_code( response_code: 400);
        die();
    }

    $orderdata->attributes['date'] = date( format: 'Y-m-d H:i:s');
    $orderdata->attributes['brand'] = $userdata['brand'];
    $order = $orderdata->update( table: 'order',$orderdata['id']);

    $orderdatapr->load($orderdataproducts);

    $orderdatapr->findIdProductsBySku();

    if(!$orderdatapr->validate($orderdataproducts)){
        $orderdatapr->getErrors();
        http_response_code( response_code: 400);
        die();
    }
```

Рис. 3.32. Оновлення замовлення

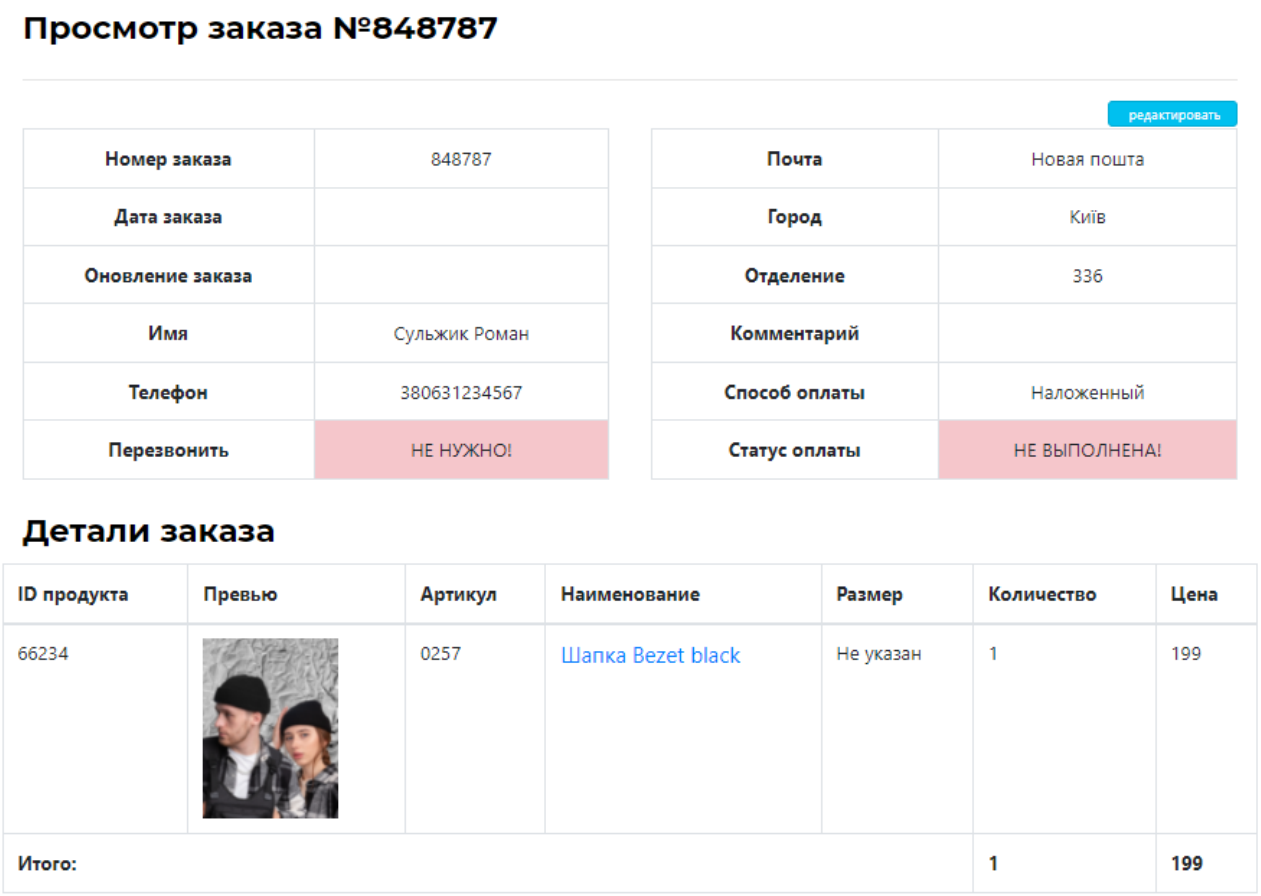


Рис. 3.33. Оновлене замовлення з API в кабінеті

3.9.5. Запит видалення даних

Видалення даних значно відрізняється від вставки, отримання чи оновлення. В першу чергу, важливо розуміти, що дані не можна повністю видаляти. Їх можна лише приховувати від користувачів та не відображати, оскільки так користувачі можуть обманювати систему видаляючі коректні замовлення та повертати собі комісію платформи. Для того щоб відображати ті чи інші дані (фільтрувати статус замовлення) існує колонка status в таблиці замовлень. Вона має чітко визначені статуси, які може набувати в бекенд частині сайту. Для того, щоб додати запит можна додати запис з статусом видалення в бекенд основного сайту, а потім при отриманні даних в API повертати всі замовлення крім цього статусу. Однак, між цим потрібно зробити перевірки на відповідність замовлення до користувача. В випадку успішного виконання

запиту в відповідь отримаємо значення true. Приклад запиту який повинен надсилатись в тілі предсталено наступним чином:

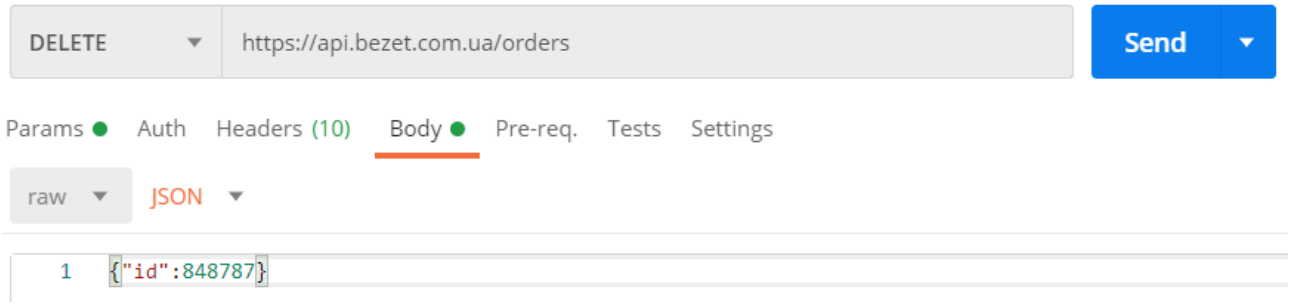


Рис. 3.34. Запит на видалення замовлення

Згідно запиту стає зрозуміло, що він включає в себе лише ID для видалення, а перевірка користувача буде відбуватись за токеном який передається в загаловку.

Код який відповідає за видалення товарів складається з всіх уже існуючих функцій та представлений на рис. 3.35.

```php
public function ActionDELETE(){
    $user = new User();
    $userdata = $user->getUserByToken();
    $data = json_decode(file_get_contents( filename: 'php://input'), associativ
    $orderdata = new Order();

    if(!\R::count( type: 'order', addSQL: 'brand=?',[$userdata['order']])){
        echo('Not your order!');
        http_response_code( response_code: 400);
        die();
    }

    $orderdata->attributes['status'] = 'deleted';
    $orderdata->update( table: 'order',$data['id']);

    echo('true');
    die();
}
```

Рис. 3.35. Видалення даних

Один з варіантів успішної перевірки – подивитись які дані зберігаються в БД. Для цього можна зайти напряму на сервер, та подивитись, яке значення має статус замовлення з ID 848787, який був переданий в тілі. Відповідь продемонстрована на рис. 3. 36.

id 1	user_tel	user_name	status
848787	380631234567	Сульжик Роман	deleted

Рис. 3.36. Дані в БД про замовлення

3.10. Документація для розробників

Очевидно, щоб інші розробники могли використовувати API для них необхідна документація. Існує безліч генераторів документації та сервісів, однак всі вони або платні, або мають безліч зайвого функціоналу. Для зручності, актуальності оновлення та незалежності від зовнішніх ресурсів прийнято рішення написати свою документацію, да зверстати її статичної сторінкою. За основу та дизайн взято популярний сервіс документації та тестування API – swagger.

Для швидкого пошуку всієї документації зручно розмістити її в корені піддомену API. В такому випадку, кожен з розробників завжди матиме посилання на документацію та зможе одразу бачити всі запити для отримання, створення, оновлення чи видалення даних.

Для створення видів прийнято рішення використати готовий CSS фреймворк bootstrap, який в вільному доступі. Це значно прискорить розробку, її подальшу підтримку та дозволить уникнути розробки дизайну. Отже, для початку необхідно створити файл шаблону, який включає в себе head, footer та весь контент, який може дублюватись на різних сторінках. Для цього в файлі documentation.php створюється стандрартний набір html тегів, де підключається

bootrstap та додаткові файли js, css, в яких можна переписувати стилі чи JS скрипти. В коді це представлено на рис. 3.37.

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <base href="/">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="s
    <link rel="stylesheet" href="/main.css">
</head>
<body>

<?= $content ?>


<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" inte
<script src="/main.js"></script>
</body>
</html>
```

Рис. 3.37. Шаблон documentation

Оскільки весь шаблон документації буде статичною сторінкою з уже заповненими даними, то зручно розділити шаблон на сайдбар з меню та навігацією, та основним контетом. Bootstrap дозволяє це робити завдяки своїм вбудованим класам, тегам та стилям. Також, в сучасних браузерах існує кешування. Це створює певні проблеми як при розробці, так і при оновленні даних (як наприклад не оновлення css на стороні клієнту без очищення кешу). Щоб обійти цю особливість в рядок підключення файлу можна додати get запитом версію файлу. А версія файлу буде дорівнювати мікросекундам останнього оновлення файлу. Така функція є в php, та має назву filemtime. Отже, при оновленні файлу – час оновлення завжди буде новий, і буде завантажуватись файл з кодом останнього оновлення. Цей метод вирішить проблему автоматичного скидання кешу на стороні клієнту та є доволі зручним для розробників. Результатуючий шаблон матиме вигляд, що зображено на рис. 3.38.

```
    <base href="/">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrit
    <link rel="stylesheet" href="/main.css?v=<?=filemtime( filename: 'main.css');?>">
</head>
<body>

<main>
    <div class="container-fluid">
        <div class="row">
            <div class="col-3">
                <div class="d-flex flex-column flex-shrink-0 p-3 text-white bg-dark"...>
            </div>
            <div class="col-9">
                <div class="documentation-content">
                    <?= $content ?>
                </div>
            </div>
        </div>
    </div>
</main>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-MrcW6
<script src="/main.js<?=filemtime( filename: 'main.js');?>"></script>
```

Рис. 3.38. Основний documentation.php

Додавши за допомогою тегів li меню, отримаємо готовий шаблон, який має привабливий вигляд, та включає в себе всі необхідні пункти в меню для описання документацій. Навігація всередині документу буде здійснюватись завдяки якорям в тегах “<a>”. Загальний вигляд меню без контенту представлени на рис. 3.39.

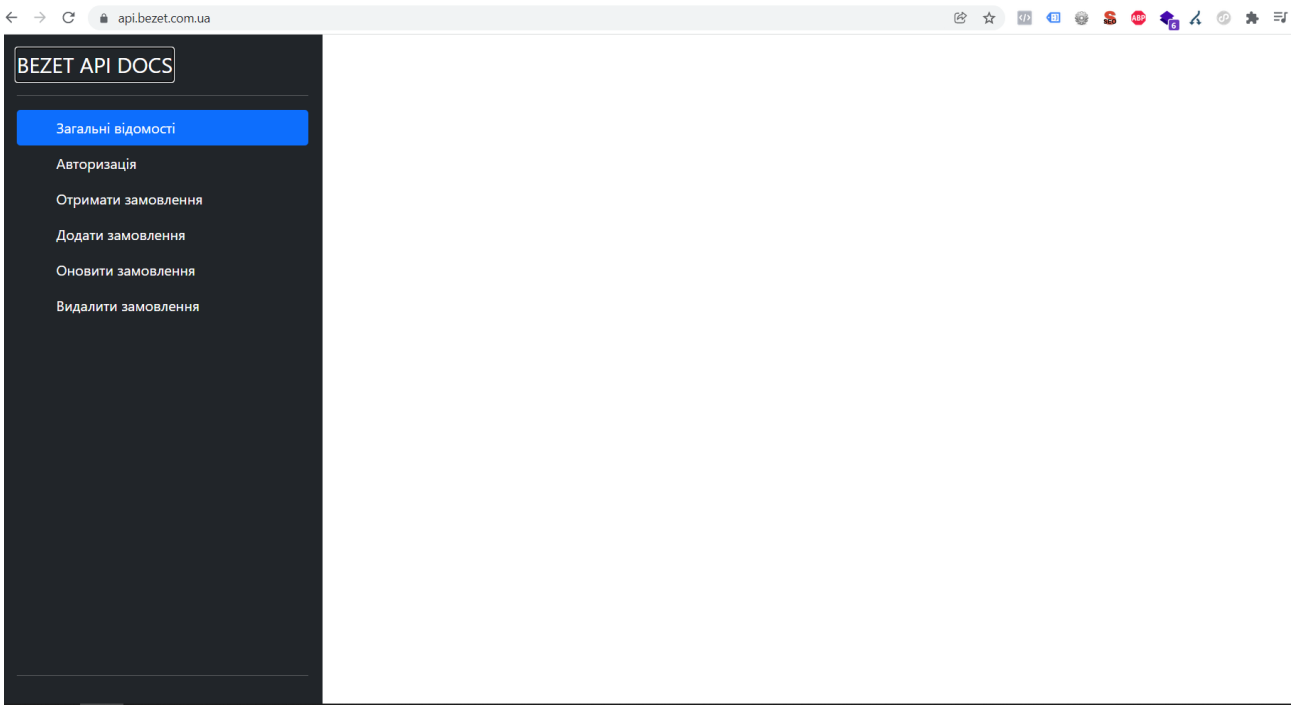


Рис. 3.39. Інтерфейс документації API

Для повної зручності лишилось зробити статичний блок сайдбару, а блок з контентом – з адаптивною висотою, яка буде скролитись окремо від висоти екрану. Для цього використано css властивість overflow-y: auto, яка буде робити скрол при перевищенні висоти блоку в висоту вікна. Для задання висоти вікна використано властивість hight с значенням 100vh. Для перевірки даних можна використати цикл в виведенні чисел від 1 до 500, що буде відповідати довгому контенту в фінальному представленні. Візуально це матиме наступний вигляд:

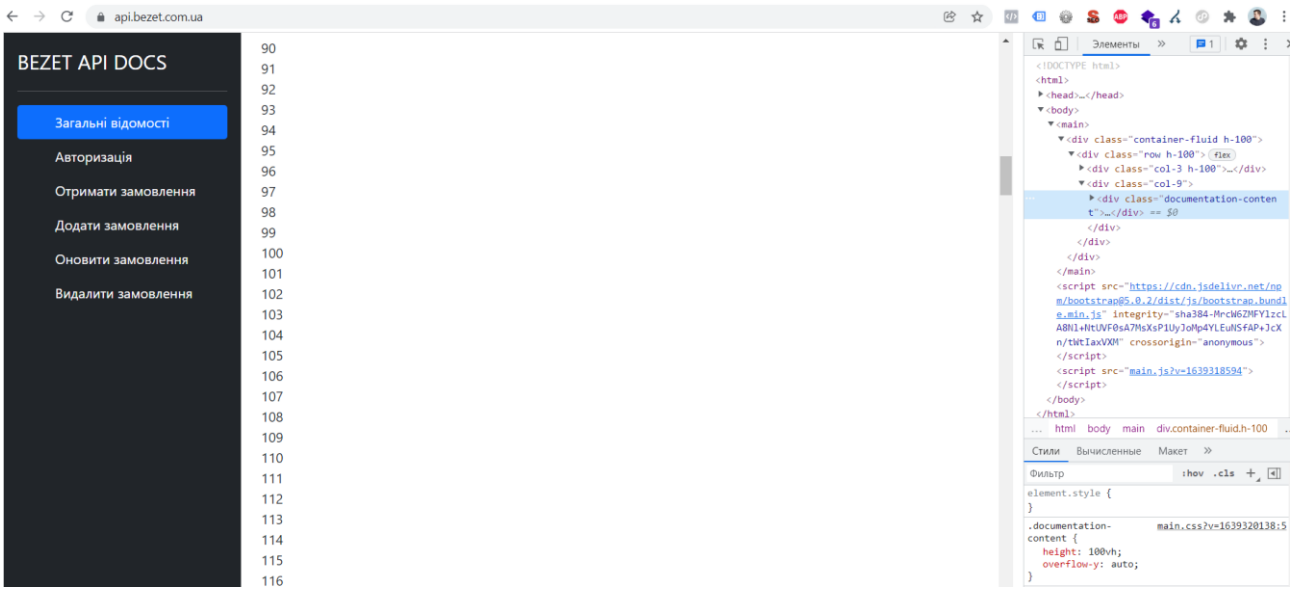


Рис. 3.40. Інтерфейс з тестовими даними та скролом

Після того як шаблон оформлення документації створено – описана загальна інформація по роботі з API. На момент релізу підготовлена документація на мові програмування PHP, однак через те, що REST API працює завдяки HTTP протоколу, всі запити легко можна інтерпретувати в інші мови. Основним критерієм роботи API є JSON формат обміну даних. На PHP всі запити відбуваються завдяки cURL. cURL — це утиліта для організації вибірки даних з вебу, що надає можливість гнучкого формування запиту із завданням таких параметрів, як cookie, user_agent, referrer і будь-яких інших заголовків.

3.10.1. Авторизація

Очевидно, що для будь якої дії з API в першу чергу необхідна авторизації, для перевірки рівня доступу, відповідності того хто запитує, до функцій обробки чи результатів. Для авторизації існує два етапи. Перший – це отримання публічного та приватного ключі, які генерує менеджер, та які доступні в кабінеті кожного з користувачі. Другий – це отримання токену авторизації, з яким будуть здійснюватись всі подальші дані. Кожен з методів в документації можна розбити візуально на дві частини, перша – інформація про запит, типи полів та їх варіанти, а друга – приклад коду на вибраній мові програмування.

Будь яка з інструкцій повинна включати в себе:

1. HTTP метод передачі
2. Посилання на яке відправляти запит
3. Опис всіх полів які можуть бути надіслані
4. Приклад запиту на обраній мові програмування
5. Короткий опис що відбувається
6. Варіанти відповідей

Отже, для текст для документації матиме наступний вигляд:

Для авторизації ви повинні зробити POST запит на посилання **/auth**. Передати потрібно два значення. При успішній валідації даних в відповідь ви отримаєте токен, який в подальшому потрібно включати в header при запитах.

Серед заголовків для отримання даних перевірка здійснюється лише на заголовок існування токену при всіх запитах крім авторизації. Однак для коректної роботи рекомендовано додати заголовок Accept з значенням json, або всі дані «*/*».

Для створення запиту рекомендовано використовувати curl. В коді це виглядатиме наступним чином:

```
$ch = curl_init( $url );
$data = json_encode( array( "pk"=> 'PKHERE',"sk"=> 'SKHERE') );
curl_setopt( $ch, CURLOPT_POSTFIELDS, $data );
curl_setopt( $ch, CURLOPT_HTTPHEADER, array('Accept:*/*'));
curl_setopt( $ch, CURLOPT_RETURNTRANSFER, true );
$result = curl_exec($ch);
curl_close($ch);
//do smthing with response
```

В результаті блок з авторизацією матиме вигляд в документації, який зображений на рис. 3.41.

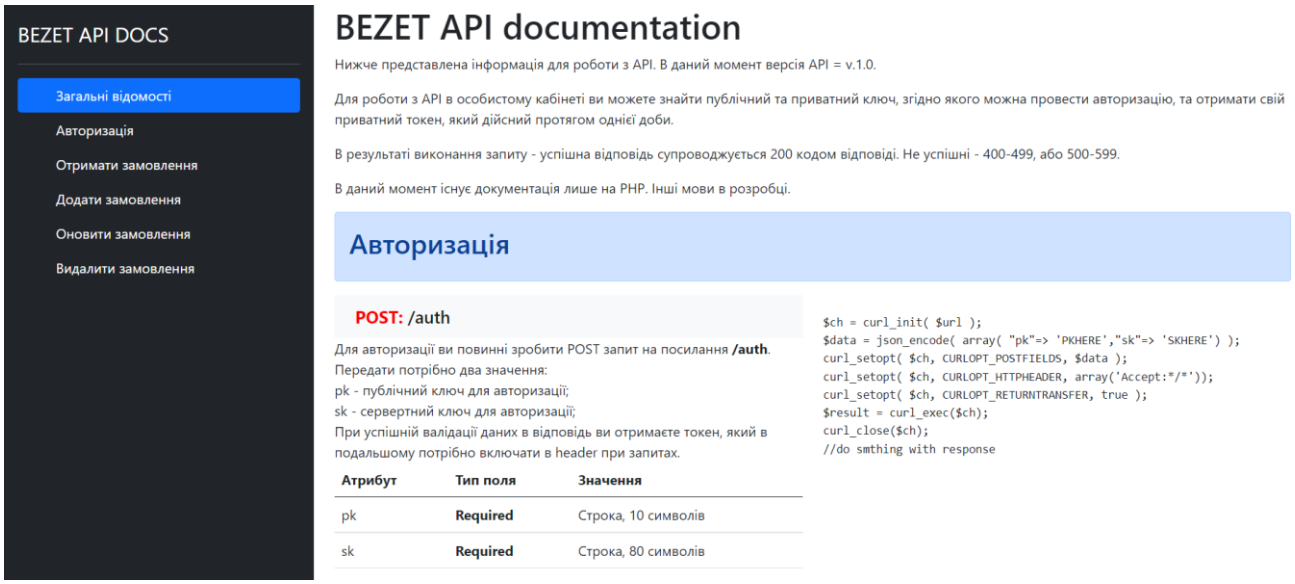


Рис. 3.41. Документація авторизації

3.10.2. Отримання замовлення

Для отримання замовлень, та всіх подальших дій в заголовку запитів уже необхідно додавати раніше сформований токен. Для цього, ключ, який отримано в попердньому запиті потрібно додати в масив headers. В коді це виглядатиме наступним чином:

```
curl_setopt( $ch, CURLOPT_HTTPHEADER, array('Accept:*/*', 'Token:
TOKENVALUE'));
```

В значення TOKENVALUE подтрібно підставити ключ, який отримано на попередньому кроці. Запит на отримання замовлень є самим простим, та в

відповідь користувач отримає список 100 останніх замовлень, в JSON форматі, які матимуть в собі дані про замовлення та про кожен з товарів, які є в цьому замовленні. В обробці даних можливі помилки в вигляді відсутності ключа, або невірного ключа, тоді сервер надасть 405 помилку невірної авторизації.

На сторінці документації дані про отримання замовлення виглядають як на рис. 3.42.

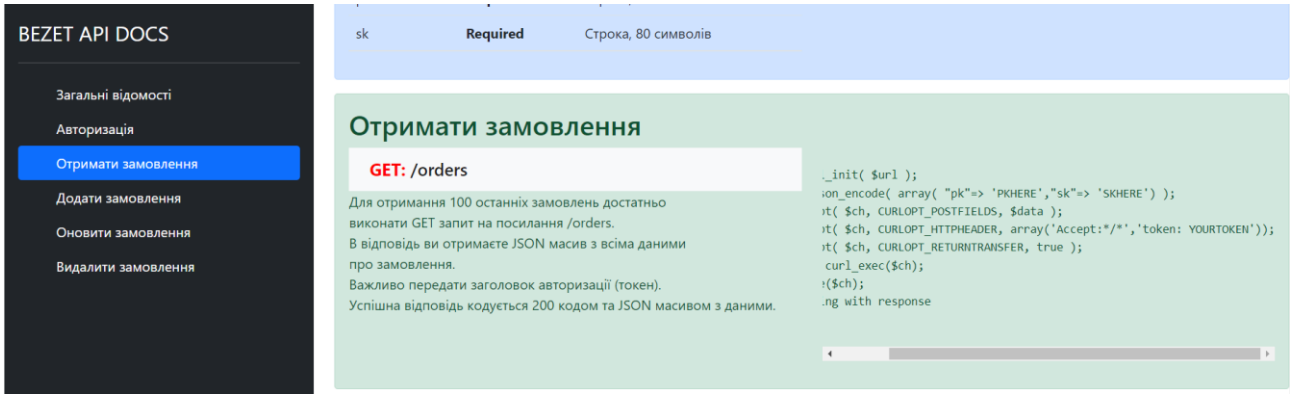


Рис. 3.42. Документація отримання замовлень

3.10.3. Створення замовлення

Створення замовлення є однією з найважчих та в той же час найважливіших функцій в існуванні API. Такий функціонал дозволяє напряму пов'язувати CRM клієнта та адміністративну частину основного веб сайту без будь якої взаємодії напряму. Для створення замовлення, як і для отримання в API необхідно передавати значення токену. Однак, крім того в тілі запиту тепер з'являються поля, які можуть приймати те чи інше значення, згідно документації, яка буде описана нижче. При створенні замовлення існує дву сутності, які поєднані, та одна є дочірньою до іншої. Маються на увазі товари, які є складовими будь якого замовлення. При цьому, в випадку відсутності товарів буде видана помилка, що к-ть товарів в масиві повинно бути як мінімум одне, що вже створено в обробнику збереження даних.

Всі поля та їх значення зображені на рис. 3.43. Для коректної роботи системи та правильної передачі даних необхідно дотримуватись всіх правил та значень в передачі.

Атрибут	Тип поля	Значення
user_tel	**Required**	Номер телефону в форматі +380123456789
user_name	**Reruired**	Текстове поле, до 192 символів
note	**Optional**	Текстове поле, до 62000 символів
city	**Required**	Місто відправлення
warehouse	**Required**	Відділення для відправки
track	**Optional**	Номер накладної для відстеження
typedel	**Optional**	Тип поштової служби. npdel = Нова пошта, ukrdel = У
call	**Optional**	int (1 - потрібно телефонувати, 0 - ні)
products	**Required**	Масив товарів. Описаний в таблиці нижче

Атрибут	Тип поля	Значення
qty	**Optional**	Число, >1, за замовчування = 1. Кількість товару.
price	**Required**	Ціна продажу, Decimal 15,2
title	**Required**	Назва на сайті BEZET
sku	**Optional**	Артикул на сайті BEZET
size	**Optional**	Розмір товару, текстове поле

Рис. 3.43. Типи полів при створенні замовлення

Сам POST запит для додавання замовлень буде одним з найбільших серед всіх, адже необхідно передати весь масив даних. Для зручності PHP дозволяє перетворювати готовий масив в JSON тіло, та передавати його. Тому, для передачі та отримання всіх даних виброна саме наступний метод. Приклад коду який відповідає за це представлено на рис. 3.44.

```
$ch = curl_init( $url );
$data = [
'user_tel'=>'',
'user_name'=>'',
'city'=>'',
'warehouse'=>'',
'products'=>
[
['title'=>'',
'price'=>'',
],
],
];
$data = json_encode($data);
curl_setopt( $ch, CURLOPT_POSTFIELDS, $data );
curl_setopt( $ch, CURLOPT_HTTPHEADER, array('Accept:*/*','token: YOURTOKEN'));
curl_setopt( $ch, CURLOPT_RETURNTRANSFER, true );
$result = curl_exec($ch);
curl_close($ch);
//do smthing with response
```

Рис. 3.44. Запит на створення замовлення

В результаті сторінка з документацією про додавання замовлення матиме дві таблиці з необхідними полями, та простим запитом з порожніми даними, однак існуючими обов'язковими полями для передачі даних. Сама сторінка в документації має вигляд, що представлено на рис. 3.45.

Рис. 3.45. Документація додавання нового замовлення.

3.10.4. Оновлення замовлення

Запит оновлення замовлення практично аналогічний до запиту створення замовлення. Тому він не потребує додаткової документації, за виключенням додання поля orderid в основні поля з даними про замовлення.

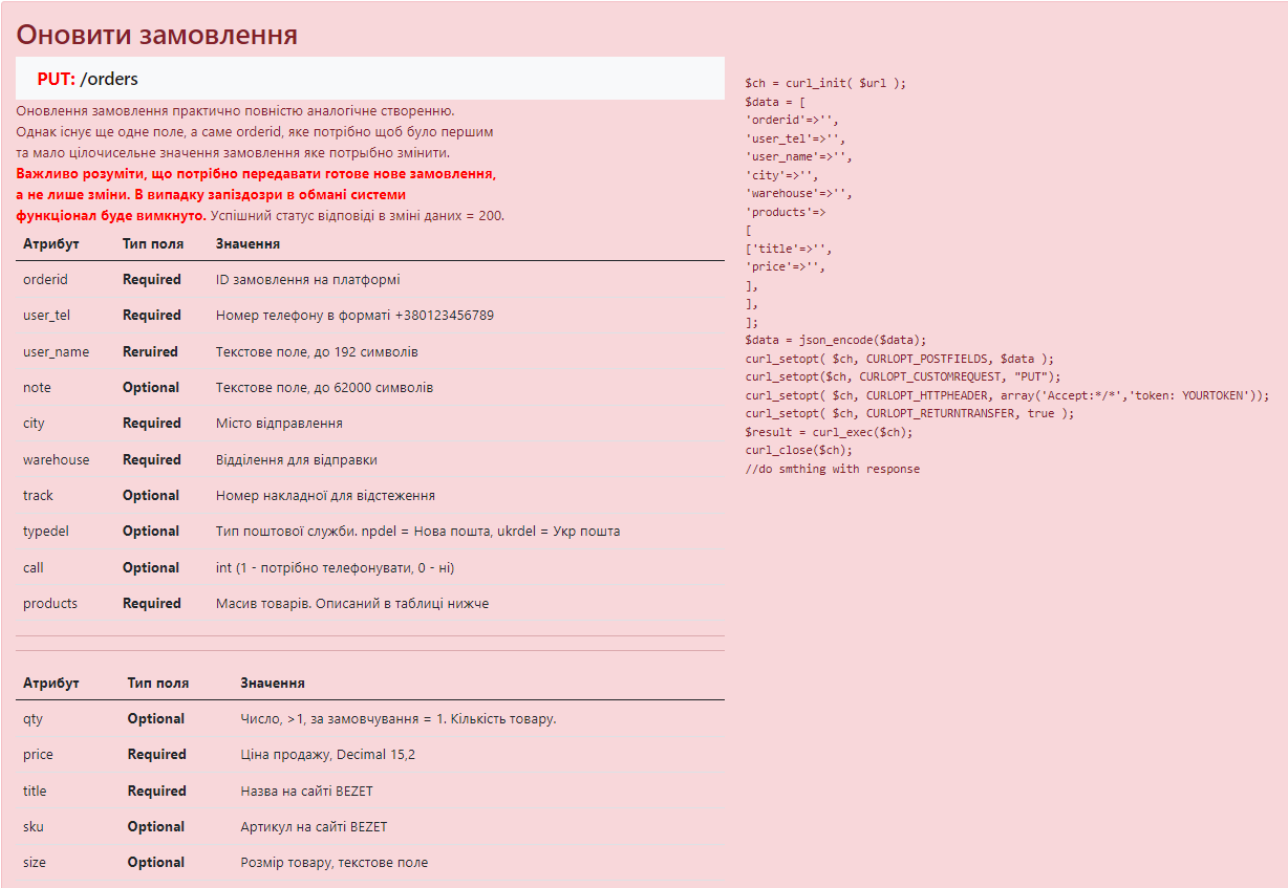
Це поширений метод в розробленні API документації, адже перед тим як оновлювати дані – було б добре повністю розуміти процедуру та кроки створення замовлення. Для цього часто використовують примітку «див. попередній крок».

З основного оновлення тут також змінюється HTTP метод запиту, який не є стандартно браузерним. Адже методи GET та POST широко використовуються в HTML та звичайних сайтах. Метод же PUT без проблем можна створити за допомогою ключа в cURL генерації запиту. Приклад коду який в такому випадку представлено наступним чином:

```
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "PUT");
```

Ключ та запит перекладається як «встанови новий тип запиту як PUT». Завдяки такому ключу сервер буде розуміти новий тип запиту.

Отже, фінально блок документації про додавання замовлення буде включати в себе оновлення одного блоку, та посилання на інструкцію попереднього блоку з додаванням замовлення. На самому сайті документації це виглядає як на рис. 3.46.



Оновити замовлення

PUT: /orders

Оновлення замовлення практично повністю аналогічне створенню.
Однак існує ще одне поле, а саме orderid, яке потрібно щоб було першим
та мало цілочисельне значення замовлення яке потрібно змінити.
Важливо розуміти, що потрібно передавати готове нове замовлення,
а не лише зміни. В випадку запіздозри в обмані системи
функціонал буде вимкнуто. Успішний статус відповіді в зміні даних = 200.

Атрибут	Тип поля	Значення
orderid	**Required**	ID замовлення на платформі
user_tel	**Required**	Номер телефону в форматі +380123456789
user_name	**Reruired**	Текстове поле, до 192 символів
note	**Optional**	Текстове поле, до 62000 символів
city	**Required**	Місто відправлення
warehouse	**Required**	Відділення для відправки
track	**Optional**	Номер накладної для відстеження
typedel	**Optional**	Тип поштової служби. npdel = Нова пошта, ukrdel = Укр пошта
call	**Optional**	int (1 - потрібно телефонувати, 0 - ні)
products	**Required**	Масив товарів. Описаний в таблиці нижче

Атрибут	Тип поля	Значення
qty	**Optional**	Число, >1, за замовчування = 1. Кількість товару.
price	**Required**	Ціна продажу, Decimal 15,2
title	**Required**	Назва на сайті BEZET
sku	**Optional**	Артикул на сайті BEZET
size	**Optional**	Розмір товару, текстове поле

```
$ch = curl_init( $url );
$data = [
'orderid'=>'',
'user_tel'=>'',
'user_name'=>'',
'city'=>'',
'warehouse'=>'',
'products'=>
[
['title'=>'',
'price'=>'',
],
],
];
$data = json_encode($data);
curl_setopt( $ch, CURLOPT_POSTFIELDS, $data );
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "PUT");
curl_setopt( $ch, CURLOPT_HTTPHEADER, array('Accept:*/*','token: YOURTOKEN'));
curl_setopt( $ch, CURLOPT_RETURNTRANSFER, true );
$result = curl_exec($ch);
curl_close($ch);
//do smthing with response
```

Рис. 3.46. Оновлення даних про замовлення

3.10.5. Видалення замовлення

Вже відомо, що видалення даних насправді не є видаленням. Це лише зміна статусу замовлення на «видалений», що в свою чергу приведе до приховування даних від користувача. Видалення є одним з найпростіших методів оновлення даних, оскільки потребує лише токен в заголовку та номер замовлення в тілі запиту. В відповідь користувач отримає відповідь true в випадку успішного видалення. При цьому, якщо замовлення вже було видалено, користувач все одно

отримає статус true. На сторінці документації таке оформлення матиме вигляд як на рис. 3.47.

Як і в запиті з оновленням замовлення тут використовується нестандартний HTTP метод запиту – DELETE, однак його логіка створення повністю аналогічна з методом PUT. Достатньо написати команду, та додати значення DELETE в ключ.

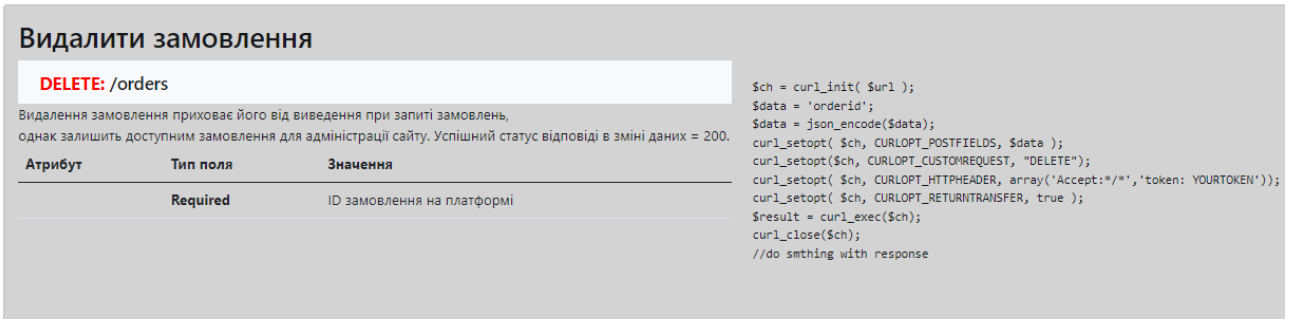


Рис. 3.47. Видалення замовлення

Документація для API не є великою, однак в ній присутні всі існуючі запити які можна використовувати, та водночас вона є дуже легкою для розуміння розробникам. Навіть починаючі розробники з легкістю можуть інтегрувати прийом, додавання чи зміну замовлень на свої платформи, що значно покращить функціонал роботи з маркетплейсом для клієнта.

Висновки до розділу 3

API (прикладний програмний інтерфейс) – опис сбособів (набір класів, процедур, функцій, структур чи констант), якими одна комп'ютерна програма може взаємодіяти з іншою. API приховують більш складний код, та дозволяють за допомогою простих дій виконувати складні набори функцій чи процесів.

В WEB-додатках API розробляються для проектів з якими взаємодіють інші розробники або для того, щоб пов’язати різні CMS чи CRM системи одна з іншою.

API не може бути вже готовою, щоб використання було ідентичне, адже в кожного проекту свої бізнес процеси та логіка внутрішніх функцій та

розрахунків. В зв’язку з великою кількістю партнерів та клієнтів, та складними внутрішніми бізнес процесами на інтернет маркетплейсі BEZET було прийнято рішення в розробці власного API на основі REST.

Розробка виконана на основі готового MVC фреймворку, що значно спрощує задачі розробки. Для методів виклику використовується тип HTTP методу, яким надіслано дані. Всі змінні передаються в форматі об’єктів в JSON форматі, що дозволяє використовувати API на будь яких платформах та мовах.

В процесі розробки створено чотири основні методи для роботи з замовленнями, а саме:

1) отримання;
2) створення;
3) оновлення;
4) видалення.

Такий функціонал дозволяє повноцінно працювати з функціоналом замовлень та інтегрувати систему маркетплейсу в власну CRM систему обробки лідів. В додадок до розробки самих методів розроблено зручно документації з використанням CSS фреймворку Bootstrap для розуміння загальних типів змінних та запитів.

API дозволено використовувати для будь-яких аналогічних проектів. Всі використані скрипти та плагіни відкриті та загальнодоступні.

ВИСНОВКИ

В дипломній роботі було поставлено завдання розробки прикладного програмного інтерфейсу для взаємозв'язку даних з інтернет маркетплейсу BEZET з зовнішніми CRM системами.

В першу чергу, були проаналізовані основні існуючі методи реалізації API та технології, завдяки яким можна реалізувати сучасний та комфортний в підключенні інтерфейс.

Склавши чітке технічне завдання, продумана логіка роботи та взаємодії з інтерфейсом. В якості мови програмування вибрано PHP, вид API – REST, для того, щоб в майбутньму додаток перевести під RESTful інтерфейс. Для роботи з базою даних використовується вже відомий в додатку ORM сервіс RedBeanPHP.

В першу чергу весь проект розгорнуто на окремому піддомені, як окремий мікросервіс, незалежно від основного додатку. Це дозволить не навантажувати основну систему, в той же час мати окремий лог запитів для контролю користувачів, відстеження спроб взлому чи логу помилок.

Для створення авторизації прийнято рішення використовувати загальносвітову практику з публічним та приватним ключем, які в свою чергу генерують тимчасовий токен, який надсилається в HTTP заголовці для будь якого з методів.

Кожен з запитів відповідає HTTP методу передачі даних. Для проекту використовується чотири найпопулярніших методів, а саме: GET, POST, PUT, DELETE. В залежності від запиту який викликано виконується той чи інший метод та функція з назвою ActionMETHODTYPE, де замість METGODTYPE – відповідний метод.

Для розуміння всіх полів та особливостей використання прикладного програмного інтерфейсу створена зручна документація, яка являє собою SPA з прикладами коду для інтеграцій на мові програмування PHP. Сам додаток документації створено з використанням CSS фреймворку BOOTSTRAP та особливостей MVC фреймворку, на якому існує весь мікросервіс з API. Це

зроблено для зручності та можливості підтримки іншими розробниками з команди BEZET.

В результаті розробки отримано повноцінно робочий прикладний програмний інтерфейс, який має зручну авторизацію, документацію та простоту в інтеграції.

В API використано всі сучасні та рекомендовані методи захисту даних від можливого перехоплення та для забезпечення коректності даних між запитом та відповіддю. Також дотримано коди стандартних HTTP відповідей, для зручного розуміння відповідей навіть не враховуючи дані з документації.

Для WEB-розробки даний прикладний програмний інтерфейс може використовуватись вільно та в будь-яких цілях.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Маркетплейс [Електронний ресурс]. Режим доступу: https://horoshop.ua/ua/blog/chto-takoe-marketpleys/ (дата звернення 02.11.2021) – Назва з екрана.
2. API [Електронний ресурс]. Режим доступу: https://dev.by/news/chto-takoe-api-prostym-yazykom (дата звернення 02.11.2021) – Назва з екрана.
3. Види Web API [Електронний ресурс]. Режим доступу: https://dataart.com.ua/news/podhody-k-proektirovaniyu-restful-api/ (дата звернення 02.11.2021) – Назва з екрана.
4. XML-RPC [Електронний ресурс]. Режим доступу: https://coderlessons.com/tutorials/xml-tekhnologii/izuchite-xml-rpc/xml-rpc-kratkoe-rukovodstvo (дата звернення 02.11.2021) – Назва з екрана.
5. SOAP [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/SOAP (дата звернення 02.11.2021) – Назва з екрана.
6. REST API [Електронний ресурс]. Режим доступу: https://habr.com/ru/post/483202/ (дата звернення 02.11.2021) – Назва з екрана.
7. PHP [Електронний ресурс]. Режим доступу: https://www.php.net/manual/ru/intro-whatis.php (дата звернення 02.11.2021) – Назва з екрана.
8. JSON [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/ru/docs/Learn/JavaScript/Objects/JSON (дата звернення 02.11.2021) – Назва з екрана.
9. RESTful [Електронний ресурс]. Режим доступу: https://betacode.net/10773/what-are-restful-web-services (дата звернення 02.11.2021) – Назва з екрана.

10. HTTP methods [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/ru/docs/Web/HTTP/Methods (дата звернення 03.11.2021) – Назва з екрана.
11. HTTP response code [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/ru/docs/Web/HTTP/Status (дата звернення 03.11.2021) – Назва з екрана.
12. POSTMAN [Електронний ресурс]. Режим доступу: https://www.postman.com/ (дата звернення 03.11.2021) – Назва з екрана.
13. Swagger [Електронний ресурс]. Режим доступу: https://swagger.io/ (дата звернення 03.11.2021) – Назва з екрана.
14. HTML [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/ru/docs/Web/HTML (дата звернення 03.11.2021) – Назва з екрана.
15. CSS [Електронний ресурс]. Режим доступу https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics (дата звернення 03.11.2021) – Назва з екрана.

Головна модель додатку

```
<?php
        namespace bezet\base;


    use bezet\Db;
    use Valitron\Validator;


    abstract class Model{


        public $attribites = [];
        public $errors = [];
        public $rules = [];


        public function __construct(){
            Db::instance();
        }


        public function load($data){
            foreach($this->attributes as $name => $value){
                if(isset($data[$name])){
                    $this->attributes[$name] = $data[$name];
                }
            }
        }


        public function save($table){
        $tbl = \R::dispense($table);
        foreach($this->attributes as $name => $value){
            $tbl->$name = $value;
        }
        return \R::store($tbl);
    }
```

```
        public function validate($data){
            $v = new Validator($data);
            $v->rules($this->rules);
            if($v->validate()){
                return true;
            }
            $this->errors = $v->errors();
            return false;
        }

        public function getErrors(){
            $errors = '<ul>';
            foreach($this->errors as $error){
                foreach($error as $item){
                    $errors .= "<li>$item</li>";
                }
            }
            $errors .= '</ul>';
            $_SESSION['error'] = $errors;
        }

        public function update($table, $id){
            $bean = \R::load($table, $id);
            foreach($this->attributes as $name => $value){
                if ($value != "") $bean->$name = $value;
            }
            return \R::store($bean);
        }
    }
?>
```

Клас Router

```
<?php

namespace bezet;
        class  Router{

    protected static $routes = [];
    protected static $route = [];

    public static function add($regexp,$route = []){
        self::$routes[$regexp] = $route;
    }

    public static function getRoutes(){
        return self::$routes;
    }
    public static function getRoute(){
        return self::$route;
    }

    public static function dispatch($url){
        $url = self::removeQueryString($url);
                if(self::matchRoute($url)){
                $controller = 'app\controllers\\' . self::$route['prefix'] . self::$route['controller'].'Controller';
                if(class_exists($controller)){
                    $controllerObject = new $controller(self::$route);
                    $action = 'Action'.self::upperCamelCase($_SERVER['REQUEST_METHOD']);
                    if (method_exists($controllerObject,$action)){
                        $controllerObject->$action();
                        $controllerObject->getView();
                    } else {
```

```
                throw new \Exception("Метод $controller::$action не найден", 404);
                        }
                    }else {
                        throw new \Exception("Контроллер $controller не найден", 404);
                    }
                }
                else {
                    throw new \Exception("Страница не найдена " .$url." На адресе - "."https://$_SERVER[HTTP_HOST]$_SERVER[REQUEST_URI]", 404);
                }
            }

            public static function matchRoute($url){
                foreach (self::$routes as $pattern => $route){
                    if (preg_match("#{$pattern}#i", $url, $matches)){
                        foreach($matches as $k => $v)
                        {if(is_string($k)){
                            $route[$k]=$v;
                            }
                        }
                        if(empty($route['action'])){
                            $route['action']='';
                        }
                        if (!isset($route['prefix'])){
                            $route['prefix']='';
                        }
                        else {
                            $route['prefix'] .='\\';
                        }
                        $route['controller'] = self::upperCamelCase($route['controller']);
                        self::$route = $route;
                        return true;
```

```
            }
             }
             return false;
        }
        protected static function upperCamelCase($name){
            return str_replace(' ','',ucwords(str_replace('-',' ', $name)));
        }

        protected static function lowerCamelCase($name){
            return lcfirst(self::upperCamelCase($name));
        }

        protected static function removeQueryString($url){
            if ($url){
                $params = explode('&',$url,2);
                if (false === strpos($params[0], '=')) {
                    return rtrim($params[0],'/');
                } else return '';
            }
        }
    }
    ?>
```