МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук, фізики та математики
Кафедра інформатики, програмної інженерії та економічної кібернетики

ПРОЄКТУВАННЯ ТА РОЗРОБЛЕННЯ WEB-ЗАСТОСУНКУ "СЕРВІС ДОСТАВКИ ЇЖІ"

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти "бакалавр"

Виконав: студент 4 курсу
Спеціальності 121 Інженерія
програмного забезпечення
Освітньо-професійної програми
«Інженерія програмного забезпечення»
першого (бакалаврського) рівня освіти
Корзун Валентин Віталійович
Керівники старший викладач
Черненко Ірина Євгенівна
кандидат фізико-математичних наук,
доцент Єрмолаєв Вадим Анатолійович
Рецензент кандидат фізико-
математичних наук, доцент
Бистрянцева Анастасія Миколаївна

Херсон – 2020

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

REST	Representational State Transfer
URL	Uniform Resource Locator
URI	Universal Resource Identifier
HTTP	HyperText Transfer Protocol
DRF	Django REST framework
JSON	JavaScript Object Notation
AJAX	Asynchronous Javascript and XML
API	Application Programming Interface

ВСТУП

Актуальність. З появою інформаційних технологій, мир змінився назавжди. Зараз інформаційні технології впроваджені в економічну, соціальну, політичну та духовну сферу життя людини. Їх використовувати надзвичайно просто, але внутрішня складова є досить складною.

Одним з найвідоміших інформаційно-технічних засобів є всесвітня мережа Інтернет. Більшість населення Землі кожен день взаємодіє з Інтернетом, оскільки він дає змогу задовольнити практично всі потреби людини. Наприклад, більше не треба витрачати час на те, щоб дістатися до бібліотеки та знайти необхідну книгу, бо вся необхідна інформація є в Інтернеті. Був час, коли ця технологія була на ранньому етапі, але зараз вже можливо навіть за допомогою мобільних телефонів отримувати інформацію досить швидко. Ще один приклад. За допомогою Інтернету можна створити власний бізнес та заробляти кошти не виходячи з дому.

Інтернет допомагає навіть у самих базових потребах, зокрема, у споживанні їжі. За допомогою Web-застосунків в Інтернеті, можна замовляти їжу у будь-який час та місце. Це дуже зручно, так як не треба витрачати час на те, щоб доставити їжу самостійно. Для людей, у яких не має часу на те, щоб приготувати їжу вдома — це необхідність. Однак, існують деякі проблеми, які мають такі сервіси. Це, насамперед, обмеження на замовлення їжі тільки з одного харчового підприємства, що у свою чергу не дає змогу людині одночасно замовити їжу з декількох харчових підприємств, що призводить до втрати часу. Також, більшість таких сервісів вимагає значний відсоток коштів з харчових підприємств.

Тому було прийнято рішення розробити Web-застосунок, у якому буде можливість замовляти їжу одночасно з декількох підприємств. Для підвищення конкурентоспроможності, буде знижено відсоток коштів з

харчових підприємств. Також, у Web-застосунку буде функціонал, який дасть змогу заробляти кошти шляхом виконання замовлень.

Ще однією особливістю цієї системи є те, що вона буде реалізована на основі архітектури REST. Це дає можливість взаємодіяти нашій системі з компонентами інших систем за допомогою встановлених правил API.

Об'єкт дослідження – технології розробки Web-застосунків.

Предмет дослідження – Web-застосунок.

Метою даної роботи є проєктування та розроблення Web-застосунку “Сервіс доставки їжі”.

Досягнення зазначеної мети здійснюється шляхом вирішення таких **основних завдань**:

1. Дослідити літературу по REST, Web-service, Restful, API, HTTP.
2. Налаштувати інтегроване середовище для розробки та спроєктувати моделі в базі даних.
3. Розробити серверну частину Web-застосунку.
4. Розробити клієнтську частину Web-застосунку.
5. Налаштувати проєкт на сервері.

Структура роботи складається з вступу, двох розділів, висновку і списку літератури.

РОЗДІЛ 1
КОНЦЕПЦІЇ ПОБУДОВИ WEB-СЕРВІСІВ З АРХІТЕКТУРОЮ REST

1.1. Web-service

Web-service – це програмні системи, розроблені для підтримки взаємодії між іншими програмами через протоколи. За допомогою набору відкритих стандартів на основі розширеної мови розмітки (XML), таких як Web Service Description Language (WSDL), SOAP, ми можемо підтримувати взаємодію між запитувачем і постачальником Web-послуг [1].

Технологія Web-сервісів – це стандартний стек технологій, що підтримує Server Oriented Architecture (SOA).

Основна мета Web-сервісів – зробити доступними функціональні блоки за допомогою стандартних Інтернет-протоколів, незалежних від платформ та мов програмування. Ці сервіси можуть бути новими програмами або просто застарілими системами.

1.2. RESTful Web Service

Клієнтські програми, які повинні обмінюватися інформацією з будь-яким доступним Web-сервісом, мають використовувати інтерфейс прикладних програм (API) незалежно від архітектурного стилю, що використовується. API відкриває набір даних і функцій для полегшення взаємодії між комп'ютерними програмами. Протокол передачі репрезентативного стану (REST) застосовується для проєктування таких API. Добре розроблені API REST можуть бути корисними для провайдерів та розробників Web-служб (через збільшення викликів запитів). Representational State Transfer (REST) являє собою абстракцію архітектурних елементів в рамках розподіленої системи гіпермедіа [8]. REST ігнорує деталі реалізації компонентів та синтаксис протоколів, щоб

зосередити увагу на ролях компонентів, обмеженнях їх взаємодії з іншими компонентами та інтерпретації значущих елементів даних. Він включає в себе основні обмеження на компоненти, роз'єми, а також дані, які визначають основу Web-архітектури, і, таким чином, визначають суть його поведінки в мережі. API Web-служб, які дотримуються архітектурних обмежень REST, називаються RESTful [8]. Одні з них:

1. Клієнт-сервер – система Web-комунікацій може прив'язувати лише ті сутності, які дотримуються встановлених стандартів для здійснення взаємодії між собою. Наприклад, для взаємодії клієнта з сервером, навіть якщо суб'єкти реалізуються різними мовами чи технологіями, вони повинні дотримуватися стандартів взаємодії один з одним.

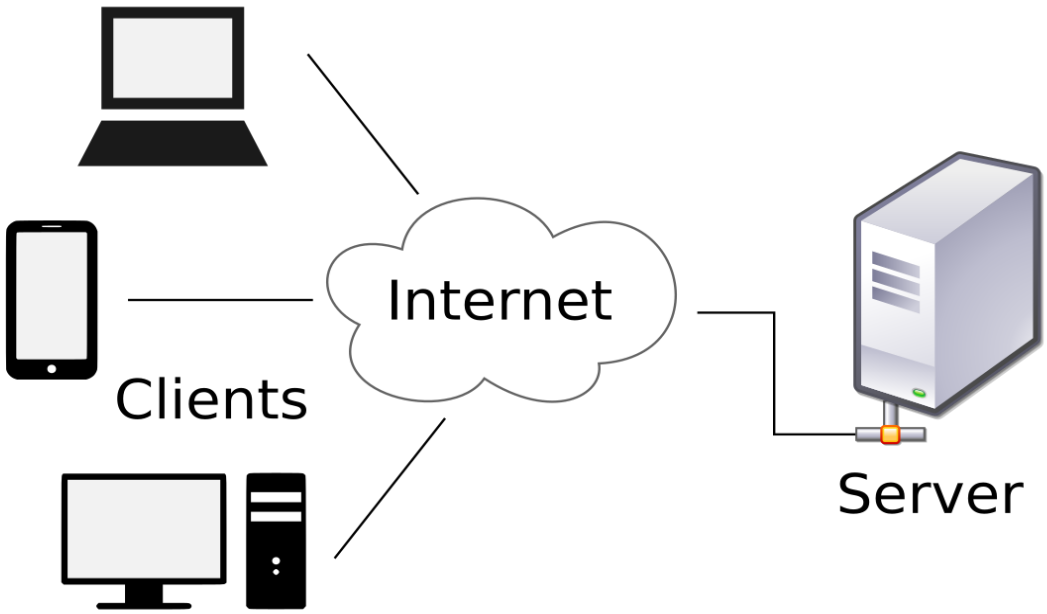

Рис. 1.1 Клієнт-серверна архітектура

2. Багатошарова система – для забезпечення безпеки, керування навантаженням на сервер, або реалізації кешування використовуються такі компоненти, як проксі та шлюзи. Шаруваті системи зменшують взаємодію між декількома шарами, приховуючи внутрішні шари від всіх, крім сусіднього зовнішнього шару, тим самим покращуючи здатність до

еволютивності і повторного використання. Такий багаторівневий стиль системи дозволяє архітектурі складатися з ієрархічних шарів, обмежуючи поведінку між компонентами таким чином, що кожен компонент не може «бачити» за межами безпосереднього шару, з якими вони взаємодіють. Обмежуючи знання про систему одним шаром, ми ставимо обмеження на загальну складність системи та сприяємо незалежності підшару системи.

3. Кеш – є одним із важливих обмежень, який знижує вартість мережі. Кешування даних дозволяє збільшити загальну кількість відповідей, надійність додатку. Це позволяє контролювати навантаження на сервер. Основна перевага кешу в тому, що він дозволяє зменшити середній час очікування запита до серверу. Кешовані дані легко можуть бути переміщені на новостворені ресурси [8].

4. Відсутність стану – це обмеження звільняє сервер від додаткового навантаження, так як не треба запам’ятовувати стан клієнтського запиту.

5. Код за запитом. Це обмеження має тенденцію до створення технології зв'язку між Web-серверами і клієнтами, так як клієнт повинен бути в змозі зрозуміти і виконати код, який він завантажує з сервера.

Архітектурні властивості, які пропонує REST:

1. Незалежність даних – REST надає можливість використовувати можливий формат для ресурсу. Дані одного ресурсу можуть бути в різних форматах. Це означає, що запит можна форматувати (в заголовку), отже, дозволяє узгодити вміст.

2. Масштабованість. Вся необхідна інформація про стан взаємодії міститься в самому запиті. Запит може бути поданий будь-яким сервером з відповідним змістом кешу, що дозволяє викликати запит через кластер серверів. Отже, забезпечити систему масштабування.

3. Переносимість – додаток повинен запускатися в неоднорідних середовищах, оскільки архітектура, орієнтована на сервіси, підтримує надання послуг для використання служб, незалежних від технології та

платформи. Використання будь-якої послуги повинно бути незалежним від середовища, на якому вона працює.

4. Простота – щоб програма була зі зрозумілою структурою та простою реалізацією, треба дробити складні задачі на прості. Це дозволяють легко реалізувати основні методи CRUD.

5. Висока продуктивність – кожен ресурс має уніфікований ідентифікатор (URI). Тому це дає змогу швидше ідентифікувати ресурс за допомогою уніфікованого локатора ресурсів (URL). Файл – найпоширеніший вид ресурсу.

1.3. RESTful API Design

Основна функція API – це можливість взаємодіяти клієнтській програмі з сервером. У свою чергу, сервер має бути розміщений на хостингу, наприклад, Heroku. Для клієнтських програм існують API, які містять в собі функції, які полегшують взаємодію між комп'ютерними програмами та дозволяють їм обмінюватися інформацією [8]. API REST називають такі API, які відповідають архітектурному стилю REST. Існують певні практики розроблення API REST, які притримуються стандарту HTTP. Для використання сервісу, його функціональних можливостей, треба правильно спроєктувати APIs. Для розроблення API використовуються компоненти API – ресурси. У API REST вони є фундаментальними одиницями. Отже, щоб мати змогу звернутися до ресурсу, треба розробити URI. Варто пам'ятати, що за допомогою методів HTTP та розробленого URI, можна виконувати будь-які дії над ресурсом.

1.4. API Design Rules

Існують правила, які узгоджуються на певному рівні та сприяють використанню "чистого" API. Ці правила звільняють від плутанини розробника, а це призводить до скорочення часу на написання коду. Є багато правил, які фактично стали стандартом, але існують і такі, які

можуть трохи відрізнятися (для своїх потреб) [8]. Основні правила такі:

• символ / повинен використовуватися для розділення двох ієрархічних компонентів;

• символ / не повинен використовуватися в найменуванні компонентів;

• при формуванні URL, слід використовувати літери з нижнім регістром, наприклад: http://localhost:8000;

• можливість задати користувачем формат відповіді, яку він очікує;

• кожен компонент в REST API, розділений символом (/), повинен відповідати унікальній моделі в ієрархії моделей;

• в URI не повинні міститися назви методів HTTP, наприклад GET;

• при формуванні URL запиту, повинно бути чітко сформована назва того, з чим цей запит взаємодіє. Наприклад, якщо при запиті GET повинен повернутися єдиний об’єкт, то тоді використовують найменування /object. Якщо при запиті GET повертається список об’єктів, то слід використовувати запис /objects;

• при використанні query params для GET запиту, ми можемо відфільтрувати відповідь за тими параметрами, які потрібно. Наприклад, /objects – поверне нам список усіх об’єктів. /objects?active=true - поверне нам список об’єктів, відфільтрованих за параметром active зі значенням true;

• методи CRUD мають використовуватися за призначенням, тобто метод POST повинен використовуватися для створення та збереження нових даних на сервері, метод GET для отримання даних, PUT для оновлення для оновлення даних з тілом запиту, які саме дані повинно оновити та метод DELETE для видалення даних;

• у кожного створеного об'єкту повинен бути унікальній URI для доступу до нього;

Отже, щоб розробити масштабований, надійний, функціональний

REST API, треба притримуватися зазначених правил при розробленні самого продукту.

1.5. Протокол передачі гіпертексту HTTP

HTTP – це протокол без збереження стану. Це означає, що клієнт та сервер не можуть бути обізнані з попередніми запитами та відповідями [6]. Саме цей протокол передає представлення ресурсів через мережу. HTTP протокол має обмеження для сервера, а саме: підтримувати стан взаємодії між клієнтом та сервером, а також забезпечити передачу відповідної інформації в межах запиту від кожного клієнта до сервера. Це обмеження покращує властивості видимості, надійності та масштабованості. Це означає, що проксі-сервери та інші посередники зможуть брати участь у комунікаційних схемах. Загибель сервера та повернення до попереднього стану не спричинить проблеми з синхронізацією стану сеансу [8]. Надійність покращується, оскільки вона полегшує можливість відновлення роботи після часткових збоїв. Масштабованість покращується, оскільки сервісам не потрібно зберігати стан та споживати пам'ять, що представляє активну взаємодію, отже, дозволяє серверному компоненту швидко звільняти ресурси.

HTTP – базовий мережевий протокол, який використовується всесвітньою мережею. Це визначає, яким чином має відбуватися передача повідомлень разом із поясненням, які дії повинні виконувати Web-сервери та браузери у відповідь на різні команди. Наприклад, вводячи URL-адресу у браузері, фактично надсилає на Web-сервер команду HTTP, спрямовуючи її на отримання та передачу запитуваної Web-сторінки [8]. REST вважається простим способом організації взаємодії між різними системами. REST має можливість використовувати HTTP протокол.

Отже, HTTP – базовий мережевий протокол, який не зберігає стан. Це означає, що сервер при здійсненні запиту до клієнта, не зберігає ніякої інформації про нього, включаючи вміст, метадані, заголовки тощо.

1.5.1. Коди статусів

Коди статусів HTTP – це стандартні відповіді на запит до сервера або клієнта у виді цифр. Ці коди допомагають виявити причину проблеми, коли Web-сторінка чи інший ресурс не завантажуються належним чином, або навпаки. Термін коду статусу HTTP насправді є загальним терміном, який включає як код статусу HTTP, так і HTTP – фразу-причину або відповідне повідомлення. Ця фраза призначена для короткого текстового опису. Коди HTTP вказують, чи успішно виконано запит HTTP. Ці відповіді згруповані у п’ять класів: інформаційні відповіді, успішні відповіді, переправлення, помилки клієнта та помилки сервера. У табл. 1.1 наведено деякі основні коди.

Таблиця 1.1

Таблиця кодів стану HTTP

Код	**Текст**	**Опис**
Інформаційні		
100	Continue	Тільки частина запиту була отримана сервером. Клієнт повинен продовжувати запит.
101	Switching Protocols	Сервер вибирає протокол.
Успішні відповіді		
200	OK	Запит успішний
201	Created	Запит виконаний, новий ресурс створений.
204	No Content	Код статусу і заголовок передані у відповіді, але дані відсутні.
Переправлення		
305	Use Proxy	Доступ до запитуваної сторінці повинен бути запитаний через проксі, який вказаний в заголовку Location.
Помилка на стороні клієнта		
401	Unauthorized	Запитувана сторінка вимагає авторизації

Продовження табл. 1.1

403	Forbidden	Доступ до запитуваної сторінки заборонений.
Помилка на стороні сервера		
500	Internal Server Error	Запит не вдався на стороні серверу.

1.5.2. Основні методи HTTP

RESTful Web-сервіси завантажують ресурси через Web-URI, і вони використовують чотири основні методи HTTP, що називаються CRUD, а саме GET, POST, UPDATE та DELETE для створення, отримання, оновлення та видалення даних.

- Метод HTTP GET використовується для отримання даних. В успішному шляху запиту, GET повертає представлення у XML або JavaScript Object Notation (JSON) та код відповіді HTTP 200 (OK). У випадку помилки він повертає 404 (NOT FOUND) або 400 (BAD REQUEST) для клієнтських помилок, тоді як 500 (Internal SERVER ERROR) для помилки на сервері. Відповідно до конструкції специфікації HTTP, GET запити використовуються лише для зчитування даних, а не для їх зміни. Тому при використанні цього способу вони вважаються безпечними. Тобто їх можна викликати без ризику зміни або пошкодження даних. Виклик одного разу має той самий ефект, що і при багаторазовому виклику. Рекомендовано, що GET запити ніколи не повинні змінювати дані [8].

- Метод HTTP POST найчастіше використовується для створення нових даних. Після успішного створення, POST повертає посилання на новостворений ресурс з кодом 201. POST запит не є безпечним, тому рекомендується застосовувати додаткові заходи забезпечення цілісності даних, наприклад, crsf token. При здійсненні двох однакових запитів відбудеться те, що створяться 2 різні об’єкти, які містять однакову інформацію.

- Метод DELETE використовується для видалення даних по ідентифікованому URI. Після успішного видалення, запит поверне HTTP – статус 200 (OK) разом із тілом відповіді. Також, можливе повернення HTTP – статусу 204 (NO CONTENT). Це означає, що запит повернувся без тіла відповіді. При неодноразовому виклику по ідентифікованому URI, запит поверне відповідь – 404 (NOT FOUND), так як дані вже видалені.

- Метод PUT використовується для оновлення даних по ідентифікованому URI. В тілі запиту передаються дані для оновлення. Після успішного оновлення, запит повертає відповідь 200 або 204 (якщо не повертає жодного вмісту). PUT – це небезпечна операція, оскільки вона змінює (або створює) дані на сервері. При неодноразовому виклику по ідентифікованому URI, запит кожен раз буде вертати той самий стан, що і у першому виклику.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБЛЕННЯ WEB-ЗАСТОСУНКУ

2.1. Детальний опис функціоналу Web-застосунку

У web-застосунку можливо зареєструватися трьома різними способами. Як клієнт, адміністратор ресторану або водій.

Клієнти мають можливість швидко та зручно знаходити потрібну їжу з ресторанів, додавати в корзину та формувати замовлення. Як вже було сказано, клієнти не мають обмеж в оформленні замовлень з різних ресторанів. До того ж, клієнт має можливість відслідковувати стан та історію всіх замовлень. Система оплати відбувається через сервіс Stripe.

Адміністратор ресторану має можливість додавати, видаляти та оновлювати характеристики певної їжі. Відслідковувати всі замовлення, що надійшли до системи, змінювати їх стан. Аналізувати кількість замовлень та їх вартість, які були здійснені протягом неділі. Аналізувати статистику найпопулярнішої їжі ресторану та найефективніших водіїв.

Водій має можливість відслідковувати стани замовлень з різних ресторанів, оформляти обране замовлення як поточне та змінювати його стан на “Виконаний” у випадку, коли водій привіз замовлення до пункту призначення. До того ж, у водія є певна статистика виконаних замовлень, яку він може аналізувати.

У web-застосунок буде впроваджена архітектура REST. По-перше, за допомогою встановлених вимог у архітектурі REST, web-застосунок стане надійним, захищеним, масштабованим. По-друге, за допомогою API запитів, є можливість зробити динамічне оновлення контенту сторінок. Це покращить взаємодію користувачів з системою. По-третє, за допомогою тих самих API запитів можливо впровадити взаємодію різних систем, програм з web-застосунком (наприклад, розробити мобільний застосунок).

2.2. Огляд технологій, які будуть використані при реалізації Web-застосунку

Від розробки до розгортання проєкту на сервері, будуть використані різноманітні технології та інструменти. Їх опис буде представлений у цьому параграфі. Особливу увагу приділено Web-фреймворку Django.

2.2.1. Django фреймворк

Це Web-фреймворк, розроблений за допомогою мови програмування python. Базується на відомому шаблоні проєктування MVC (Model – моделі, View – відображення, Controller – контролер). За допомогою Django можна вирішувати звичайні завдання в Web-розробці. Він є надзвичайно швидким та масштабованим, безпечним і неймовірно універсальним.

Django дотримується правил, при яких різні частини системи слабко залежать одна від одної. Це означає, що кожну частину системи можна легко замінити іншими подібними функціональними компонентами, роблячи компоненти ортогональними. Також, шаблон проєктування MVC дозволяє легко з’єднати моделі, відображення, та контролери. Наприклад, шаблони використовуються для відображення HTML, у моделях зберігаються дані системи, які передаються до встановленої бази даних в системі, а в відображеннях відбуваються маніпуляції з даними моделей. Як же було сказано, Django притримується правил, при яких компоненти слабко залежать одне від одного. Це дозволяє різним компонентам системи працювати в згуртованості, незалежно один від одного, коли и де це можливо. Наприклад, url – адреси в Django не залежать від написаного коду, який генерує представлення даних, тобто, їх можна змінювати, не змінюючи жодного рядка у відображеннях.

Не повторюй свій код – ще одне правило в Django, яке сприяє використанню якомога менше коду. Короткі, чисті та доступні

функціональні строки коду є результатом правила в Django. Наприклад, успадкування шаблонів – це концепція в Django, яка дає можливість не повторювати кожен раз написання одного шаблону в інших. Для цього, використовуються такі теги, як: {% extends %}, {% include %} і т.д., про які ми поговоримо пізніше. Також, Django перейняла філософію з Python, а саме: «явне краще, ніж неявне».

Переваги використання Django Framework.

Як вже було зазначено раніше, Django Framework легко справляється з повсякденними завданнями в розробці. Використання Django дозволяє розробникам підкреслювати конкретні аспекти програм, яку вони розробляють, а не застосовувати загальні аспекти Web-розробки. Також, Django містить в собі реалізовані функції, яких може не бути в більшості інших фреймворків.

Крім того, Django має велику спільноту та чітку документацію – це полегшує починаючим розробникам реалізовувати функціонал проєкту. Django широко використовується в багатьох компаніях, має відкритий вихідний код з кількома сторонніми пакетами, які завжди оновлюються. Всю інформацію можна знайти на офіційному сайті https://www.djangoproject.com.

Так як Django притримується філософії DRY «не повторюй себе», написаний код в фреймворку є чітким, читабельним та коротким. Розгорнути проєкт можна на любій платформі, яка підтримує мову програмування Python. До того ж, Django має можливість взаємодіяти з багатьма базами даних в одному стилі, а саме ORM (object relational mapper), це ще більше підсилює його портативність.

Django має вбудовану панель адміністратора, яка потрібна для того, щоб взаємодіяти з моделями в встановленій базі даних. В панелі можливо виконувати всі дії CRUD над об’єктами в моделях. Інтерфейс панелі буде зрозумілий як для розробника, так і для звичайного користувача.

Найголовніша особливість в Django – вбудована система

тестування та масштабованість. Сайти з великим трафіком, такі як: DropBox, Instagram, Reddit, Quora і т.д., використовують Django, як основну технологію. Оскільки модулі в системі слабко залежать один від одного, їх можливо підключати та відключати під різні потреби. Розгортання проєкту зі складною архітектурою на сервері не займає багато часу, так як Django розроблявся так, що процес розгортування проходив швидко.

Процес обробки запитів в Django.

HTTP запит в браузері використовується для побудови певного об'єкта HttpRequest, який передається наступним компонентам системи. Django має проміжний компонент «middleware framework», який перехоплює обробку запиту або відповіді, таким чином, має можливість змінювати вхідні, вихідні дані [10]. Наприклад, AuthenticationMiddleware перехоплює запити та пов'язує їх з конкретними користувачами за допомогою сеансів. Функція «view» є останньою в порядку обробки. Якщо «HttpRequest» не обробляється, то сервер повертає HTTP 404 або HTTP 500 відповідь. Функція «view» є однією з важливих концепцій в розробленні Web-сайтів на Django. Процес обробки запиту / відповіді більш детальніше можна подивитися на рис. 2.1.

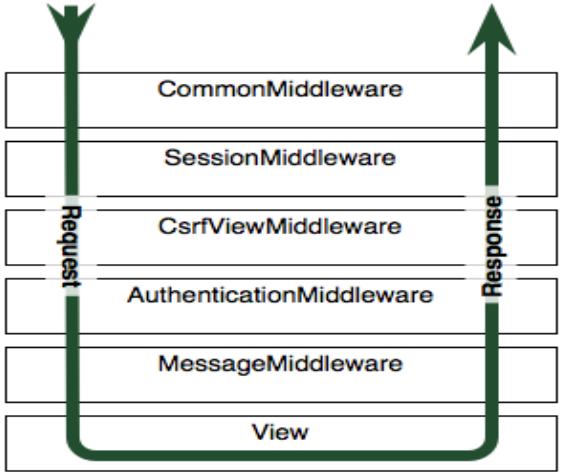

Рис. 2.1 Django Request/Response Cycle

Останній етап обробки відповіді / запиту проходить, коли

програмне забезпечення Response Middleware обробляє HttpResponse для повернення до браузера [10].

2.2.2. Django REST framework

Django Rest Framework (DRF) – це бібліотека, яка працює зі стандартними моделями Django для створення гнучкого і потужного API для проєкту [11]. DRF складається з 3-х комнонентів: «serializer», «view» і «url».

- Serializer: перетворює інформацію, що зберігається в базі даних в певний формат, який легко і ефективно передається через API.
- ViewSets: визначає функції (читання, створення, оновлення, видалення), які будуть доступні через API.
- Url: визначає URL-адреси, які будуть надавати доступ до кожного «view».

Serializer.

Моделі Django представляють дані, що зберігаються в базі в складній структурі, але API повинен передавати інформацію в простій структурі. Хоча дані будуть представлені як екземпляри класів Model, для зручності, їх необхідно перевести в формат JSON для передачі через API.

Всі ці функції виконує serializer DRF. Коли користувач передає інформацію через API, serializer бере дані, перевіряє їх на правильність і перетворює в коректний формат даних для екземпляра моделі. Аналогічним чином, коли користувач звертається до даних через API, відповідні об’єкти моделей передаються в serializer, який перетворює їх в формат JSON, для зручності роботи з даними.

Найпоширеніша форма serializer є клас serializers.ModelSerializer, який прив’язується до моделі:

```
class RestaurantSerializer(serializers.ModelSerializer):
    user = serializers.StringRelatedField()

    class Meta:
        model = Restaurant
        fields = "__all__"
```

Рис. 2.2 клас serializers.ModelSerializer

Параметр «fields» вказує які поля доступні для serializer. Також, існує параметр «exclude», який, навпаки, включає доступні поля крім тих, які вказані в «exclude».

Serializer – це неймовірно гнучкий і потужний компонент DRF. Хоча підключення serializer до моделі є найбільш поширеним, serializer можуть використовуватися для створення будь-якої структури даних Python відповідно до визначених параметрів.

ViewSets.

Serializer потрібні для форматування даних в певний формат при читанні або запису, а ViewSets – це класи з різними функціями, які визначають методи, які матиме користувач при доступі до певного URL, з яким зв’язаний ViewSets. Основні методи:

- create() – створення нового об’єкта;
- retrieve() – отримання об’єкта;
- update() – оновлення даних в об’єкті;
- destroy() – видалення об’єкта;
- list() – повернення списку об’єктів;

Кожен з цих методів вже реалізований в базовому ViewSets, потрібно лише перевизначити методи у своєму класі. На рис. 2.3 це продемонстровано.

```python
class UserViewSet(viewsets.ViewSet):
    """
    A simple ViewSet for listing or retrieving users.
    """
    def list(self, request):
        queryset = User.objects.all()
        serializer = UserSerializer(queryset, many=True)
        return Response(serializer.data)

    def retrieve(self, request, pk=None):
        queryset = User.objects.all()
        user = get_object_or_404(queryset, pk=pk)
        serializer = UserSerializer(user)
        return Response(serializer.data)
```

Рис. 2.3 клас viewsets.ViewSet

У класа UserViewSet реалізовані 2 методи. Метод list повертає усіх user, які є в моделі. Метод retrieve повертає одного user по ключу pk. Усі методи повертають дані в форматі JSON.

Url.

Як вже було зазначено, це просто url-адреса, які будуть надавати доступ до кожного «view».

2.2.3. База даних SQLite

SQLite – це база даних, яка написана на мові програмування C. Вона забезпечує легку за розміром базу даних на диску, яка не потребує окремої серверної обробки та дозволяє отримати доступ до даних за допомогою мови запитів SQL [12]. Вона легко налаштовується у проєкті, тому що не потребує окремих бібліотек для її підтримки. У Django, SQLite встановлена за умовчанням. Досить тільки застосувати 2 команди:

python manage.py makemigrations

python manage.py migrate

Та створити супер користувача:

python manage.py createsuperuser

Для взаємодії з таблицями в базі даних. Також, для більш складних проєктів, можна створити прототип моделей у базі даних SQLite, а потім

перенести всі створені таблиці у більш складу базу даних, наприклад, PostgreSQL, MySQL.

Ще однією особливістю є те, що Django підтримує ORM з SQLite. ORM (Object – Relational Mapping) – це технологія, яка дозволяє взаємодіяти з об’єктами в базі даних за допомогою методів та класів в Django. Це прискорює та полегшує написання коду, так як не треба власноруч прописувати SQL запити, Django це робить самостійно.

2.2.4. Hypertext Markup Language, Cascading Style Sheets

Hypertext Markup Language (HTML) – мова розмітки, за допомогою якої ми можемо бачити сторінку у браузері таку, яка вона є. HTML визначає структуру Web-сторінки [13]. Наприклад, кожна сторінка містить у собі такі елементи, які представлені на рис. 2.4.

```
<! DOCTYPE html>
<html>
   <head>
      <title>Назва</title>
   </head>
   <body>
      <p> Hello world!</p>
   </body>
</html>
```

Рис. 2.4 структура html

Cascading Style Sheets (CSS) – мова стилю, яка потрібна для того, щоб покращити сприйняття та доступність контенту [13]. Використовується тільки разом з HTML.

2.2.5. JavaScript

JavaScript – мова програмування, яка є однією з найпопулярніших мов, які використовуються при створенні сценаріїв для Web-сторінок. Містить у собі парадигми функціонального, об’єктно-орієнтованого

програмування та має динамічну типізацію [14]. Якщо HTML та CSS відповідають за візуальне представлення елементів на сторінці, то JS потрібен для того, щоб:

1. Оновлювати дані, які відображені на сторінці, без перезавантаження сторінки.

2. Мати доступ до кожного елемента на сторінці (змінювати розмір, положення, стилі, контент та ін.)

3. Асинхронно взаємодіяти з сервером через API запити.

4. Зберігати інформацію у кэші браузера при оновленні сторінки.

І це тільки маленька частина того, для чого він потрібен. Існують різноманітні бібліотеки, фреймворки та навіть мова програмування Node.js, основою якої став JavaScript. Всі вони необхідні для виконання різноманітних задач. Це говорить про те, що JS використовується фактично скрізь.

2.2.6. Stripe

Stripe – американська технологічна компанія, що розробляє рішення для прийому і обробки електронних платежів [15]. Stripe створює потужні та гнучкі інструменти для інтернет-комерції. Функціональні API Stripe допомагають створити найкращий можливий продукт для користувачів. Мільйони інноваційних компаній у світі швидко та ефективно масштабують свій бізнес, будуючи його на Stripe.

Stripe постійно вдосконалюється та кожен місяць оновлює свій продукт новими можливостями.

Stripe дозволяє легко впровадити в свій продукт можливість обробки електронних платежів. Технологія має надійний захист від несанкціонованих доступів або перехоплень, так як працює за принципом відкритого / закритого ключа шифрування. Значним плюсом є те, що можна використовувати Stripe безкоштовно.

2.2.7. Heroku

Heroku – це хмарна платформа, яка дозволяє користувачам легко та швидко розгортати проєкти будь-якої складності [16]. Це економить час, а отже і кошти. Heroku дозволяє налаштовувати проєкти на сервері безкоштовно, але такий сервер, звичайно, не має достатніх конфігурацій для того, щоб обслуговувати мільйонів користувачів. Для того, щоб мати таку можливість, треба оформити спеціальну підписку. Технологія підтримує такі мови програмування, як: Node.js, Ruby, Java, PHP, Python, Go, Scala, Clojure [16]. Для кожної мови програмування є чітка документація.

2.2.8. Git

Git – це розподілена система контролю версій, яка дозволяє зберігати всі зміни, внесені в файли, що зберігаються в репозиторії [17]. Коміти – це певні знімки, які містять у собі зміни. Це дуже зручно, так як у випадку збою системи, можна відкотити код до будь-якого збереженого стану. Ще однією важливою особливістю Git є те, що можна взаємодіяти з іншими розробниками. Це як Google Docs, тільки для розробників. Також, Git використовується при налаштуванні проєкту на сервері, так як містить чітку структуру та зручні команди, які можуть завантажити проєкт на будь-який сервер, який містить у собі встановлений Git.

2.3. Розроблення серверної частини Web-застосунку

2.3.1. Налаштування інтегрованого середовища Pycharm

У цьому параграфі буде налаштована базова конфігурація проєкту перед початком розроблення проєкту. Будуть зачеплені такі теми, як настройка віртуального оточення, бази даних, статичних та медіа файлів, IDE PyCharm і т.д.

Перед початком роботи, треба створити папку з проєктом. Потім, за допомогою IDE Pycharm, відкрити цю папку.

PyCharm – це інтегроване середовище розробки (IDE), яке використовується в комп'ютерному програмуванні, спеціально для мови Python [18].

Наступним кроком буде встановлення віртуального оточення. Це можна зробити двома способами, через Pycharm або командну строку.

Virtualenv – важливий інструмент, який створює ізольоване середовище певної версії Python. Це вирішує проблеми залежностей встановлених пакетів та їх версій.

У нашому проєкті буде фігурувати версія Python 3.7. Після цього, потрібно встановити всі пакети, які потрібні для проєкту. Їх список наведено у рис. 2.5. Щоб встановити пакети, треба активувати virtualenv та застосувати команду:

pip install –r requirements.txt, де requirements.txt – текстовий файл з пакетами. Продемонстровано на рис. 2.5.

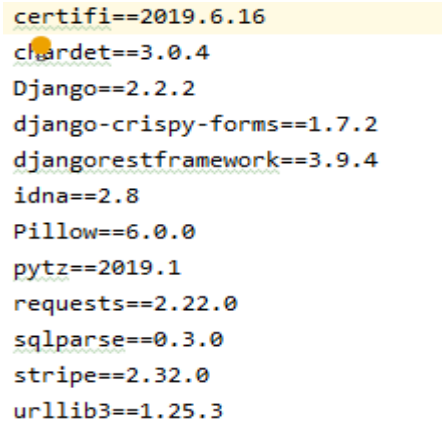

Рис. 2.5 Встановлені пакети

Деякі з них поки що не будуть використовуватися, але, у подальшому, вони будуть потрібні при реалізації frontend частини.

Наступним кроком буде налаштування Django. Застосовуємо команди:

1. django-admin startproject <назва основної папки проєкту> (config);
2. python manage.py startapp <назва компоненту проєкту> (core);

3. python manage.py startapp <назва компоненту проєкту> (restaurant);
4. python manage.py startapp <назва компоненту проєкту> (driver);
5. python manage.py startapp <назва компоненту проєкту> (customer);
6. python manage.py startapp <назва компоненту проєкту> (user);
7. python manage.py makemigrations – створення міграцій для бази даних;
8. python manage.py migrate – мігрування даних в базу даних;
9. python manage.py createsuperuser – створення супер користувача;

Повинна вийти структура проєкту, яка наведена на рис. 2.6.

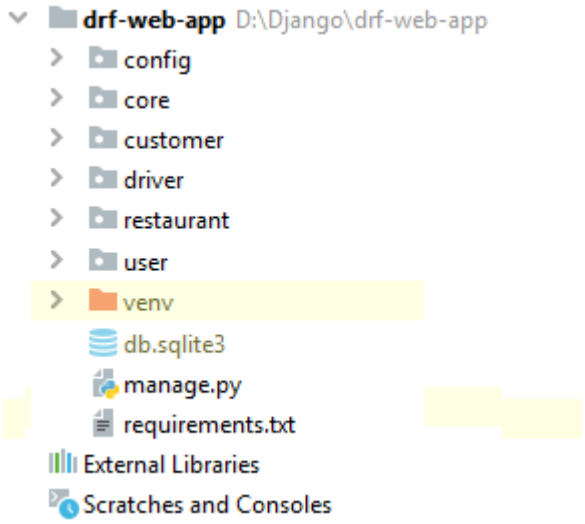

Рис. 2.6 Структура проєкту

В файлі config/settings.py в списку INSTALLED_APPS, треба додати встановлені компоненти та пакети Django REST framework. Кожен компонент створений для того, щоб вирішувати поставлені завдання. Наприклад, в компоненті «core» реалізуються всі моделі, які потрібні нашій системи.

У подальшому, нам потрібно буде створити структуру шаблонів та статичних файлів, налаштувати їх у config/settings.py.

Для перевірки правильного налаштування нашого проєкту,

потрібно застосувати команду:

python manage.py runserver – запускає наш сервер на localhost:8000

Повинна відобразитися сторінка з вмістом на рис. 2.7.

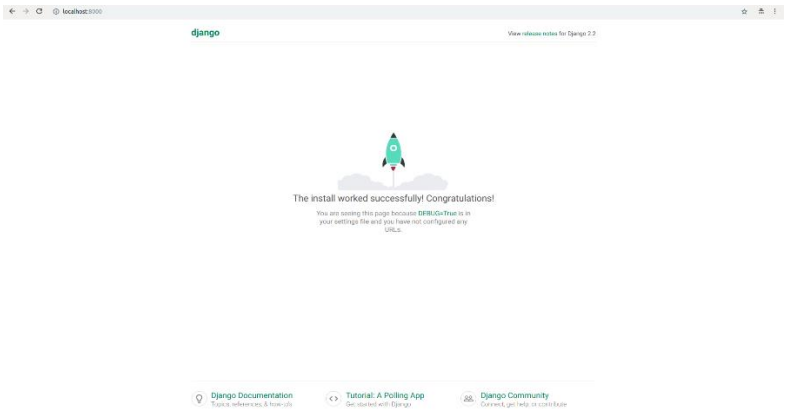

Рис. 2.7 Стартова сторінка Django

2.3.2. Проєктування моделей в базі даних

Важливою частиною кожного проєкту є архітектура програмного забезпечення. Від того, як структура проєкту буде спроєктована, залежить якість програмного забезпечення. Особливо це стосується моделей, які будуть знаходитися в базі даних. Тому, у цьому параграфі будуть спроєктовані моделі системи. Також, потрібно реалізувати діаграми класів, послідовностей для того, щоб розуміти, як моделі повинні взаємодіяти між собою.

У нашій системі представлено 8 моделей. Модель User містить базові поля для всіх типів користувачів. Від неї успадковуються моделі Restaurant, Driver, Customer. Це потрібно для того, щоб не дублювати поля в успадкованих моделях. Meal відповідає за зберігання їжі. StripeSettings потрібно для того, щоб зв’язувати певний ресторан з платіжною системою Stripe. Order відповідає за формування певного замовлення. OrderDetails потрібна для більш детального формування замовлення. Усі моделі, зв’язки між ними продемонстровані в діаграмі класів на рис. 2.8.

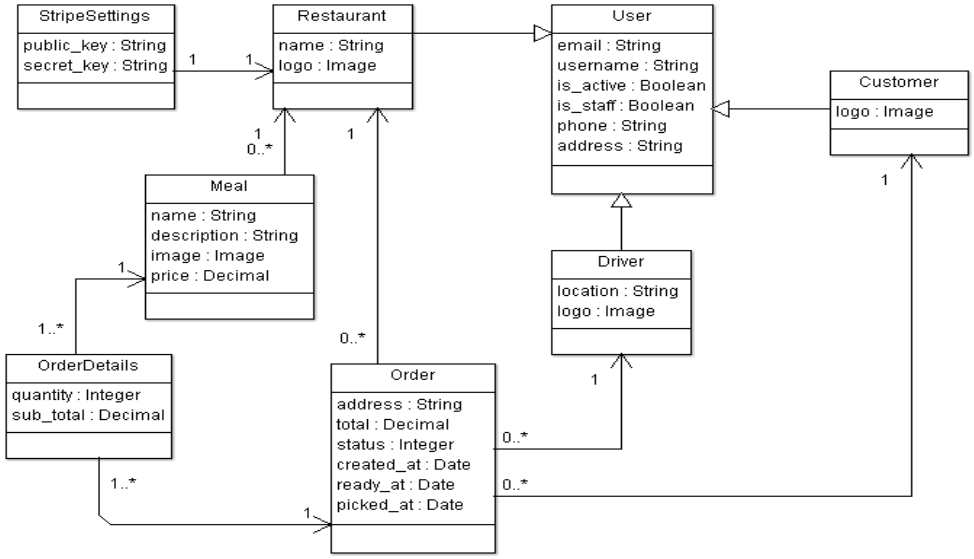

Рис. 2.8 Діаграма класів

Потрібно зазначити, що існують деякі правила, які надають системі тих вимог, які потрібні.

1. Поля email, username повинні бути унікальними.

2. Для всіх атрибутів в моделях повинні бути обмеження в кількості символів.

3. Якщо Order status дорівнює COOKING, то Restaurant може змінити його на READY.

4. Якщо Order status не дорівнює FAIL та DELIVERED, то Restaurant може змінити на FAIL.

5. Якщо у Driver є замовлення, який дорівнює ONTHEWAY, то він не може взяти ще одне замовлення.

6. Якщо виконується умова (5), та Order status дорівнює READY і його ще ніхто не взяв, то Driver може взяти це замовлення.

7. Якщо Order status дорівнює ONTHEWAY, то Drive може змінити його на DELIVERED.

8. Якщо у Customer є Order status, який не дорівнює FAIL або DELIVERED, то він не може взяти ще одне замовлення з того Restaurant, до якого прив’язаний цей Order.

Діаграма послідовностей (рис. 2.9) продемонструє основний функціонал системи. Потрібно зазначити, що Driver, Restaurant, Customer вже авторизовані в системі.

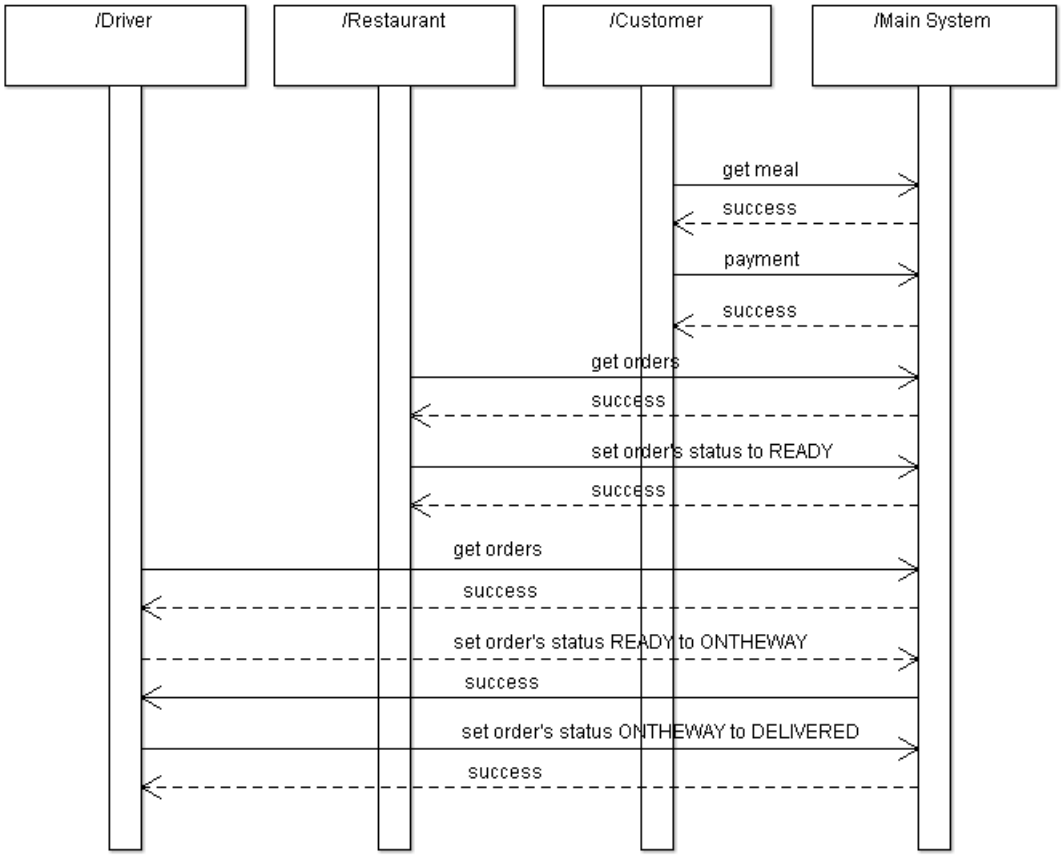

Рис. 2.9 Діаграма послідовності

Переходимо до реалізації. В папці core/models.py потрібно реалізувати спроєктовані моделі. Зробимо це на прикладі моделі Order. Для того, щоб зв’язати нашу модель з базою даних, класу Order потрібно успадкуватися від класу models.Model. Також, потрібно розробити перелічувальний тип для поля status. Реалізований клас представлено на рис. 2.10.

```
class Order(models.Model):
    COOKING_ = 1
    READY_ = 2
    ONTHEWAY_ = 3
    DELIVERED_ = 4
    FAIL_ = 5

    STATUS_CHOICES = (
        (COOKING_, "Cooking"),
        (READY_, "Ready"),
        (ONTHEWAY_, "On the way"),
        (DELIVERED_, "Delivered"),
        (FAIL_, "Fail"))

    customer_ = models.ForeignKey(Customer_, on_delete=models.CASCADE)
    restaurant_ = models.ForeignKey(Restaurant_, on_delete=models.CASCADE)
    driver_ = models.ForeignKey(Driver_, on_delete=models.CASCADE, blank=True, null=True)
    address_ = models.CharField(max_length=256)
    total_ = models.IntegerField()
    status_ = models.IntegerField(choices=STATUS_CHOICES)
    create_at_ = models.DateTimeField(default=timezone.now)
    ready_at_ = models.DateTimeField(blank=True, null=True)
    picked_at_ = models.DateTimeField(blank=True, null=True)

    def __str__(self):
        return str(self.id)
```

Рис. 2.10 Клас Order

Виконуємо вже знайомі команди:

1. python manage.py makemigrations – створення міграцій для бази даних;

2. python manage.py migrate – мігрування даних в базу даних;

2.3.3. Реалізація serializer

Реалізувати свій serializer досить легко. Треба імпортувати в файл restaurant/api/serializers.py бібліотеку rest_framework та успадкуватися від serializers.ModelSerializers. Знову ж таки, зробимо це на прикладі Order.

Для початку, треба розробити SerializerRestaurant, SerializerMeal, SerializerOrderDetail. Так як SerializerOrder має зв’язок між ними. Реалізовані serializers представлені на рис. 2.11.

```
class SerializerRestaurant(serializers.ModelSerializer):
    user_ = serializers.StringRelatedField()
    class Meta:
        model = Restaurant_
        fields = "__all__"

class SerializerMeal(serializers.ModelSerializer):
    restaurant_ = serializers.StringRelatedField()
    class Meta:
        model = Meal_
        fields = "__all__"

class SerializerOrderDetails(serializers.ModelSerializer):
    meal_ = SerializerMeal()
    class Meta:
        model = OrderDetails_
        fields = "__all__"
```

Рис. 2.11 Реалізовані serializers

Потім, реалізуємо SerializerOrder. Реалізований SerializerOrder представлено на рис. 2.12.

```
class SerializerOrder(serializers.ModelSerializer):
    customer_ = SerializerCustomer()
    driver_ = SerializerDriver()
    restaurant_ = SerializerRestaurant()
    order_details_ = SerializerOrderDetails(many=True)
    status_ = serializers.ReadOnlyField(source="get_status_display")
    payment_ = serializers.ReadOnlyField(source="get_payment_display")

    class Meta:
        model = Order_
        fields = "__all__"
```

Рис. 2.12 SerializerOrder

Клас meta використовується для зв’язку serializer з певною моделлю.

2.3.4. Реалізація views

Перед початком реалізації views, потрібно імпортувати деякі бібліотеки:

1. from .permissions import IsRestaurantUser – дозволяє отримати доступ до views тільки для певного ресторану;

2. from restaurant.api.serializers import SerializerOrder – імпортуємо розроблений раніше serializer;

3. from rest_framework import generics – основний пакет, за допомогою якого можно створити view;

4. from rest_framework.generics import get_object_or_404 – якщо певного об’єкта моделі не існує, повертає код помилки 404;

5. from rest_framework import permissions, authentication – використовується для системи аутентифікації;

Після цього, реалізуємо views, яка повертає у форматі JSON дані усіх замовлень певного ресторану, відсортовані по полю status. Реалізований views представлено на рис. 2.13.

```
class ListApiViewOrder(generics.ListAPIView):
    serializer_class = SerializerOrder
    authentication_classes = (authentication.SessionAuthentication, authentication.TokenAuthentication)
    permission_classes = (permissions.IsAuthenticated, IsRestaurantUser)

    def get_queryset(self):
        restaurant_ = get_object_or_404(Restaurant_, user=self.request.user)
        return Order_.objects.filter(restaurant=restaurant_).order_by('status')
```

Рис. 2.13 ListApiViewOrder

Важливо зазначити, що во всіх views буде використовуватися система аутентифікації двох типів. Це SessionAuthentication та TokenAuthentication.

1. SessionAuthentication – це аутентифікація, яка визначає користувача по сесії в браузері. У нашому випадку, використовується для

AJAX запитів, які будуть реалізовані у frontend частині застосунку [21].

2. TokenAuthentication – це аутентифікація, яка визначає користувача по певному токену, який зберігається в базі даних після реєстрації користувача. Вона потрібна для того, щоб користувач по цьому токену зміг отримати доступ до певного views. Використовується при реалізації android застосунку [21].

Головною метою системи аутентифікації є захист персональних даних кожного користувача.

Останнім кроком буде з‘єднання views з url. В папці restaurant/api/urls.py потрібно імпортувати наступні пакети:

1. from django.urls import path – метод, який створює url запит;

2. from .views import ListApiViewOrder – розроблений views;

Реалізований url представлено на рис. 2.14.

```
urlpatterns = [
    path('order-list/', ListApiViewOrder.as_view(), name='order-list'),

]
```

Рис. 2.14 url “order-list”

2.3.5. Тестування отриманих API

В процесі розроблення, значну увагу приділяють тестуванню застосунку. Це дає змогу підвищити якість програмного забезпечення та відшукати дефекти в системі. Існують багато видів тестування, а саме: функціональне тестування, тестування інтерфейсу, конфігураційне тестування, тестування стабільності або надійності, тестування продуктивності та інші.

Ми будемо використовувати функціональне тестування. Функціональне тестування – це тестування програмного забезпечення, яке спрямоване на виявлення невідповідностей між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і вимог. Розрізняють 2 типи функціонального тестування:

мануальне та автоматичне.

У нашому випадку, використаємо мануальне тестування класу OrderListApiView, який ми реалізували раніше. Розробимо декілька тестів:

- доступ до url “/orders/” неможливий для неавторизованого користувача;
- доступ до url “/orders/” неможливий для авторизованого користувача з типом реєстрації “Customer”;
- доступ до url “/orders/” неможливий для авторизованого користувача з типом реєстрації “Driver”;
- доступ до url “/orders/” можливий для авторизованого користувача з типом реєстрації “Restaurant”. У відповіді містяться усі замовлення, які є у ресторану;

Для прикладу, протестуємо url “/orders/”. По-перше, потрібно створити адміністратора ресторану. По-друге, треба створити замовлення та заповнити його деякими даними для тесту. Та останнє – з’єднаємо замовлення з раніше створеним адміністратором ресторану. Усі дії виконуються в панелі адміністратора Django.

На рис. 2.15 представлено відповідь, які надіслав нам сервер у випадку, коли ми авторизувались як адміністратор ресторану.

Рис. 2.15 Відповідь сервера на запит

Переконуємося у тому, що коректні дані надійшли у цьому запиті.

У разі неочікуваної поведінки, нам потрібно відшукати помилку та виправити її. Але, у нашому випадку, цього робити не потрібно. Ті самі дії потрібно зробити для інших тестів. Однак, щоб ще більше переконатися у тому, що система працює стабільно, треба покрити усі класи / функції такими тестами.

Реалізація внутрішньої структури Web-застосунку закінчена. Переходимо до розроблення графічного інтерфейсу користувача.

2.4. Розроблення клієнтської частини Web-застосунку

У цьому параграфі буде розроблений привабливий графічний інтерфейс для користувача. Так як в нашій системі є три типи користувача, то треба для кожного типу реалізувати власні сторінки в браузері. Для прикладу, зробимо це для користувача типу «адміністратор ресторану».

2.4.1. Проэктування структури html сторінок та їх реалізація.

У нашому проєкті створимо папку «templates». В цій папці будуть зберігатися всі html сторінки, які нам потрібні. На рис. 2.16 представлена структура html сторінок.

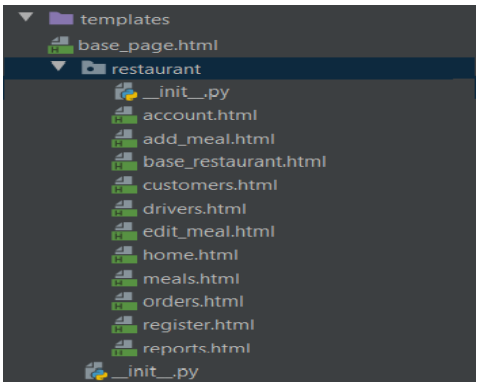

Рис. 2.16 Структура html сторінок для користувача

Кожна html сторінка має власну функцію. Наприклад. Сторінка meals.html потрібна для того, щоб адміністратор міг редагувати / видаляти їжу. Сторінка base_page.html та base_restaurant.html мають базову

структуру, підключені бібліотеки, стилі, які потрібні на кожній сторінці. Таким чином, не потрібно дублювати свій код (філософія django). За допомогою спеціальної мови шаблонів, яка міститься у Django, можливо підключати, включати шаблони один в одного. Це все виконується за допомогою тегів {% extends 'назва шаблону' %} та {% include 'назва шаблону' %}. Також, можливо перевизначити блоки у шаблонах через тег {% block назва блоку %} {% endblock %}.

Розробимо сторінку додавання їжі, а саме add_meal.html. Для цього нам потрібно:

1. Успадкуватися від базових шаблонів, які містять всі необхідні структурні теги, завантажені бібліотеки стилів, скриптів.

2. Перевизначити блоки, які будуть у add_meal.html за допомогою тега {% block %}.

3. За допомогою завантаженої бібліотеки bootstrap, розробити привабливий інтерфейс для додавання їжі.

4. Через представлення у Django, передати форму, яка потрібна для збереження даних у базі даних.

На рис. 2.17 представлена сторінка add _meal.html.

```
{% extends 'restaurant/base_restaurant.html' %}
{% load crispy_forms_tags %}
{% block page_content %}
    <div class="row bg-white has-shadow">
        <div class="col-lg-10 col-sm-10 mx-auto">
            <form method="post" enctype="multipart/form-data">
                {% csrf_token %}
                {{ form|crispy }}
                <button class="btn btn-warning btn-block" type="submit">
                    <strong>CREATE</strong>
                </button>
            </form>
        </div>
    </div>
{% endblock page_content %}
```

Рис. 2.17 add_meal.html

При використанні метода POST необхідно обов'язково вказувати тег {% csrf_token %} для того, щоб зловмисник не зміг підробити запит для користувача і вкрасти у нього дані, або змінити дані на сервері,

використовуючи небезпечні методи, такі як POST, PUT, PATCH та DELETE.

Форма, яку ми отримали з сервера, була передана з певного представлення. Представлення зв’язано з певним url.

На рис. 2.18 продемонстровано приклад коду цього представлення та url.

```python
from django.urls import path
from .views import  AddMeal

app_name = 'restaurant'

urlpatterns = [
    path('add_meal/', AddMeal.as_view(), name='add-meal'),
]

class AddMeal(RequiredRestaurant):
    template_name = 'restaurant/add_meal.html'

    def get(self, request):
        form = MealForm()
        return render(request, self.template_name, {"form": form})

    def post(self, request):
        form = MealForm(request.POST, request.FILES)
        if form.is_valid():
            meal = form.save(commit=False)
            meal.restaurant = request.user.restaurant
            meal.save()
            return redirect("restaurant:meals")
        else:
            messages.error(request, form.errors, 'danger')
            return redirect('restaurant:add-meal')
```

Рис. 2.18 клас AddMeal та url

Приклад розробленої сторінки з додаванням їжі представлено на рис. 2.19.

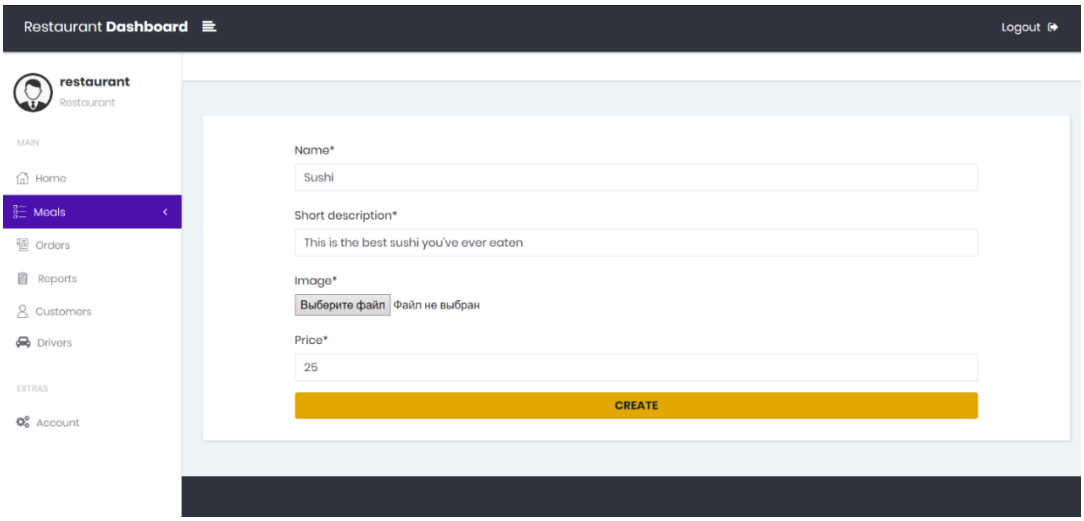

Рис. 2.19 Сторінка з додаванням їжі

2.4.2. Розроблення AJAX запитів для взаємодії з сервером

У попередніх параграфах ми розробляли API, які необхідні для передачі певних даних. Зараз настав час використати ці API для отримання даних. Таким чином, буде розроблена клієнт-серверна взаємодія, яка була описана в теоретичному розділі. Для розроблення цього функціоналу, треба реалізувати AJAX запити на стороні клієнта, які будуть взаємодіяти з сервером.

AJAX, (Asynchronous Javascript and XML) – підхід до побудови користувацьких інтерфейсів, Web-застосунків, за яких Web-сторінка, не перезавантажуючись, у фоновому режимі надсилає запити на сервер і сама довантажує потрібні користувачу дані [22]. Таким чином, користувач не повинен перезавантажувати сторінку кожен раз, а це означає, що користуватися такими Web-застосунками стає зручніше. Значна перевага AJAX запитів у тому, що користувачу не потрібно перезавантажувати сторінку та навантажувати сервер.

Для розроблення графічного представлення системи нам безпосередньо потрібна технологія AJAX. Продемонструємо це при реалізації сторінки замовлень. На ній адміністратор ресторану повинен отримувати актуальні дані замовлень, які надійшли в ресторан та мати можливість змінити стан певного замовлення.

Для отримання даних треба використати вже реалізований API “OrderListApiView”, який містить у собі всі замовлення певного ресторану. Конструкція отримання даних за допомогою AJAX наведена на рис. 2.20.

```
$.ajax({
    url: '/api/restaurant/orders/',
    method: 'GET',
    success: function (data) {
        // ...
    },
    error: function (xhr, textStatus, errorThrown) {
        // ...
    }
});
```

Рис. 2.20 Отримання даних за допомогою AJAX

AJAX запит має достатньо параметрів, але основні такі:

- url – адрес, на якому можна отримати необхідні дані;
- method – тип запиту (GET, POST, PUT і т.д.);
- success – функція, яка виконується у разі успішного отримання даних;
- error – функція, яка виконується у разі повернення сервером помилки;

Слід додати що ці дані може отримати користувач, який аутентифікований у системі, інакше система поверне статус помилки:

```
403 Forbidden "detail": "Authentication credentials were not provided."
```

Однак, якщо у користувача є токен, який ідентифікує його у системі, то він може передати його у параметрах та отримати дані. Як вже було сказано, це зроблено для того, щоб можна було користуватися системою не тільки у браузері, а у телефоні.

Повернемося до реалізації. Після того, як дані отримані, треба їх відобразити. Для цього буде використана бібліотека DataTables [23]. Вона відображає дані у зручній таблиці та містить у собі безліч корисних функцій, серед яких: сортування, пошук даних, пагінація, адаптація під різні монітори та навіть телефони. Для підключення бібліотеки необхідно у base_page.html підключити бібліотеки javascript та css файлів. Після цього, для певної таблиці у js файлі необхідно прописати таку конструкцію:

```
let table_orders_ = $("#table_orders_").DataTable()
```

Після цього плагін вже буде задіяний. Реалізована таблиця на сторінці замовлень наведена на рис. 2.21.

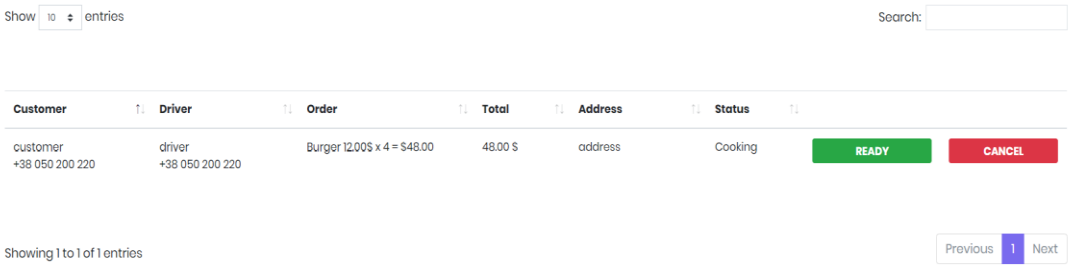

Рис. 2.21 Таблиця замовлень

У цій таблиці є вся необхідна інформація: дані про клієнта, водія, детальна інформація про замовлення, сума, адреса, статус та дві кнопки «Відмінити» та «Готово». Перша кнопка потрібна для того, щоб підтвердити готовність замовлення та передати його до рук водія, а друга щоб відмінити. Ці кнопки також посилають на сервер AJAX запит, тільки з методом POST, так як змінюються дані в базі даних. Пам'ятаємо про те, що цей метод є небезпечним та у параметрах запиту вказуємо crsf_token таким чином:

```
headers: {"X-CSRFToken": csrf_token}
```

Таким чином, майже для кожної сторінки буде використовуватись AJAX запит з методами POST и GET. Тільки для клієнта буде використовуватись сервіс Stripe, який необхідний для того, щоб здійснювати покупку через картку. Але він також реалізується з підтримкою AJAX запитів.

Отже, з розробкою графічного представлення системи закінчено. Для стабільності системи, рекомендовано написати функціональні тести, але ми робити цього не будемо, так як на це треба витратити так само часу, як і на розробку. Останнім кроком буде налаштування Web-застосунку на PaaS платформі Heroku.

2.5. Налаштування проєкту на PaaS платформі Heroku

Для того, щоб налаштувати наш проєкт на платформі Heroku, спочатку потрібно встановити застосунок на комп'ютер та авторизуватись у системі за допомогою команди `heroku login` у терміналі. Після цього у кореневому каталогу проєкту створимо файл Procfile. У ньому потрібно додати такий текст: web: gunicorn drf-app.wsgi --log-file. Це необхідно для того, щоб Heroku розумів, що нам потрібно запускати Web-застосунок за допомогою Web-серверу Gunicorn. Наступним кроком потрібно додати файл runtime.txt та вписати ту версію python, на якому працює проєкт. У нашому випадку це python-3.7. Встановимо додаткові пакети для розгортування командою:

```
pip install gunicorn dj-database-url whitenoise psycopg2
```

- gunicorn – Web-сервер, який буде запускати наш проєкт;
- dj-database-url – пакет для взаємодії з базою даних;
- whitenoise – пакет, який потрібен для правильної роботи статичних файлів;
- psycopg2 – Heroku використовує базу даних Postgres. Цей пакет необхідний для взаємодії з нею;

Обновимо файл settings.py такими строками:

```
import dj_database_url
prod_db  =  dj_database_url.config(conn_max_age=500)
DATABASES['default'].update(prod_db)
MIDDLEWARE = [
    'whitenoise.middleware.WhiteNoiseMiddleware',
    ...
]
```

Створимо застосунок командою:

```
heroku create herokudjangoapp
```

Ініціалізуємо Git та підключимо новий застосунок до віддаленого сховища Heroku Git за допомогою команд:

```
git init
```

```
heroku git:remote -a drf-web-app
```

Додамо файли в git та внесемо зміни

```
git add .
git commit -m "Initial commit"
```

Завантажимо проєкт до віддаленого сховища Heroku.

```
git push heroku master
```

Зробимо міграції бази даних

```
heroku run python manage.py migrate
```

Після цього можна відкрити посилання та побачити, що тепер Web-застосунок є частиною всесвітньої мережі Інтернет.

Це можна побачити на рис. 2.22.

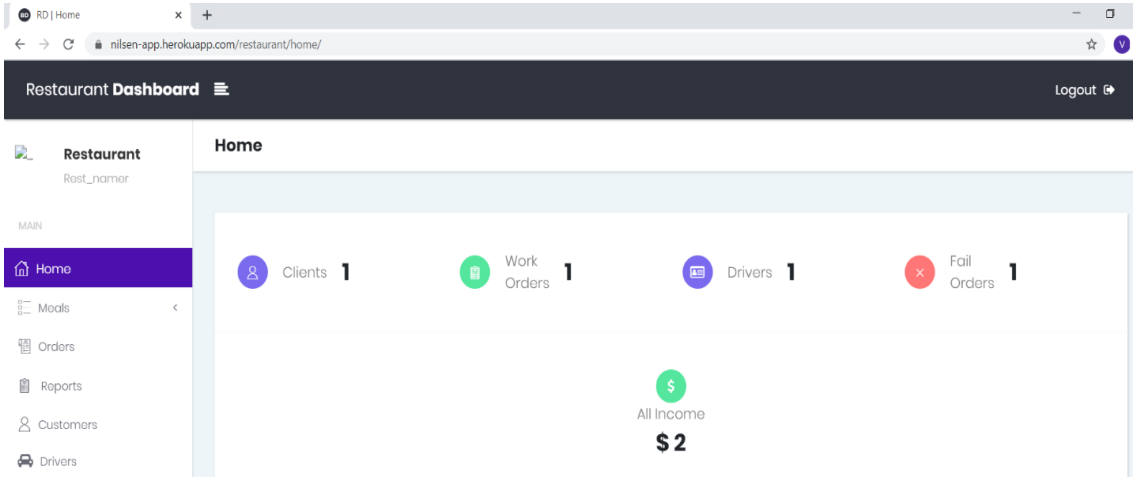

Рис. 2.22 Web-застосунок в Інтернеті

На цьому кроці можна завершити розроблення Web-застосунку.

ВИСНОВКИ

У ході кваліфікаційної роботи були виконані наступні завдання:

- Досліджено літературу по Web-service, RESTful Web Service, RESTful API Design, API, HTTP, які були представлені та детально описані в першому розділі.
- Налаштовано інтегроване середовище розробки Pycharm, створено ізольоване віртуальне середовище Virtualenv, встановлені всі необхідні пакети для розроблення проєкту. Встановлено фреймворк Django та бібліотека DRF. Спроєктовано та реалізовано моделі в базі даних SQLite.
- Розроблена серверна частина Web-застосунку. Підтвердженням цього є реалізовані API запити, які дозволяють користувачам взаємодіяти з Web-застосунком за допомогою уніфікованого локатора ресурсу. За допомогою мануального тестування, ми перевірили надійність та стабільність наших API запитів.
- Розроблена клієнтська частина Web-застосунку. Для кожного типу користувачів створені персональні Web-сторінки. За допомогою технології AJAX здійснена взаємодія клієнта з сервером. Впроваджена технологія Stripe, яка дозволяє користувачам здійснювати оплату замовлень через картку.
- Налаштовано проєкт на PaaS платформі Heroku. Тепер кожна людина може здійснити запит до нашого Web-застосунку за допомогою персонального комп’ютера, мобільного телефона, планшета тощо.

Отже, можна стверджувати, що мета даної роботи була повністю виконана. Наразі, за https://nilsen-app.herokuapp.com існує розроблений Web-застосунок “Сервіс доставки їжі”, який вже може задовольняти потреби користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Чен Р. Інтелектуальні обчислення та інформатика: комунікації в галузі комп'ютерів та інформатики: конф. : 8-9 січня 2011 р., Чунцін, 2011. – 115 с.
2. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software / Eric Evans., 2011.
3. Філдінг Р. Т. Архітектурні стилі та дизайн архітектури програмного забезпечення / Р. Т. Філдінг. – Ірвін : каліфорнійський університет, 2000.
4. Robinson I. REST in Practice: Hypermedia and Systems Architecture / I. Robinson, J. Webber, S. Parastatidis., 2010.
5. Masse M. REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces / M. Masse. – California : O'Reilly Media, 2011.
6. Arana S. Web Services Platform Architecture / S. Arana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. – New Jersey : Prentice Hall, 2005.
7. Allamaraju S. RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity / Subbu Allamaraju., 2010.
8. Richardson L. Restful Web Services / L. Richardson, S. Ruby. – California : O'Reilly Media, 2007.
9. Список кодов HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Список_кодов_состояния_HTTP.
10. Kumar S. Django: Request/Response Cycle [Електронний ресурс] / Sarthak Kumar. – 2019. – Режим доступу до ресурсу: https://medium.com/@ksarthak4ever/django-request-response-cycle-2626e9e8606e.
11. Django REST framework [Електронний ресурс]. – Режим доступу до ресурсу: https://www.django-rest-framework.org.
12. SQLite Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://www.sqlite.org/docs.html.

13. HTML/CSS [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://en.wikibooks.org/wiki/HyperText_Markup_Language/CSS.

14. JavaScript [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/JavaScript.

15. Stripe [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Stripe.

16. Heroku [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Heroku.

17. Git [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Git.

18. Pycharm [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/PyCharm.

19. Документация Django 3.0 на русском языке [Електронний ресурс]. – Режим доступу до ресурсу: https://django.fun/docs/django/ru/3.0/.

20. Nield T. Getting Started with SQL: A Hands-On Approach for Beginners / Thomas Nield., 2016. – 134 с.

21. DRF Authentication [Електронний ресурс]. – Режим доступу до ресурсу: https://www.django-rest-framework.org/api-guide/authentication/.

22. AJAX [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/AJAX.

23. DataTables [Електронний ресурс] // Вікіпедія. – Режим доступу до ресурсу: https://datatables.net.

24. Gunicorn - Python WSGI HTTP Server for UNIX [Електронний ресурс]. – Режим доступу до ресурсу: https://gunicorn.org.

25. Getting Started on Heroku with Python [Електронний ресурс]. – Режим доступу до ресурсу: https://devcenter.heroku.com/articles/getting-started-with-python.

26. Gheorghiu G. Python for DevOps / G. Gheorghiu, A. Deza, K. Behrman. – California : O'Reilly Media, 2019. – 506 с.

27. Percival H. Test-Driven Development with Python / Harry Percival. –

California: O’Reilly Media, 2016. – 624 с.

28. Дронов В. А. Django: практика создания Web-сайтов на Python / В. А. Дронов. – Петербург: Профессиональное программирование, 2016. – 528 с.

29. Уязвимость CSRF. Введение [Електронний ресурс]. – Режим доступу до ресурсу: https://intsystem.org/security/learn-about-csrf-intro.

30. Библиотека Django [Електронний ресурс]. – Режим доступу до ресурсу: https://github.com/w3prog/GoodTravel/wiki/Библиотека-Django.

31. Введение · Django Girls Tutorial [Електронний ресурс]. – Режим доступу до ресурсу: https://tutorial.djangogirls.org/ru/.

ДОДАТКИ

Додаток А

КОДЕКС АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ХЕРСОНСЬКОГО
ДЕРЖАВНОГО УНІВЕРСИТЕТУ

Я, Корзун Валентин Віталійович,
учасник(ця) освітнього процесу Херсонського державного університету, **УСВІДОМЛЮЮ**, що академічна доброчесність – це фундаментальна етична цінність усієї академічної спільноти світу.

ЗАЯВЛЯЮ, що у своїй освітній і науковій діяльності **ЗОБОВ'ЯЗУЮСЯ**:

– дотримуватися:

- вимог законодавства України та внутрішніх нормативних документів університету, зокрема Статуту Університету;
- принципів та правил академічної доброчесності;
- нульової толерантності до академічного плагіату;
- моральних норм та правил етичної поведінки;
- толерантного ставлення до інших;
- дотримуватися високого рівня культури спілкування;

– надавати згоду на:

- безпосередню перевірку курсових, кваліфікаційних робіт тощо на ознаки наявності академічного плагіату за допомогою спеціалізованих програмних продуктів;
- оброблення, збереження й розміщення кваліфікаційних робіт у відкритому доступі в інституційному репозитарії;
- використання робіт для перевірки на ознаки наявності академічного плагіату в інших роботах виключно з метою виявлення можливих ознак академічного плагіату;

– самостійно виконувати навчальні завдання, завдання поточного й підсумкового контролю результатів навчання;

– надавати достовірну інформацію щодо результатів власної навчальної (наукової, творчої) діяльності, використаних методик досліджень та джерел інформації;

– не використовувати результати досліджень інших авторів без використання покликань на їхню роботу;

– своєю діяльністю сприяти збереженню та примноженню традицій університету, формуванню його позитивного іміджу;

– не чинити правопорушень і не сприяти їхньому скоєнню іншими особами;

– підтримувати атмосферу довіри, взаємної відповідальності та співпраці в освітньому середовищі;

– поважати честь, гідність та особисту недоторканність особи, незважаючи на її стать, вік, матеріальний стан, соціальне становище, расову належність, релігійні й політичні переконання;

– не дискримінувати людей на підставі академічного статусу, а також за національною, расовою, статевою чи іншою належністю;

– відповідально ставитися до своїх обов'язків, вчасно та сумлінно виконувати необхідні навчальні та науково-дослідницькі завдання;

– запобігати виникненню у своїй діяльності конфлікту інтересів, зокрема не використовувати службових і родинних зв'язків з метою отримання нечесної переваги в навчальній, науковій і трудовій діяльності;

– не брати участі в будь-якій діяльності, пов'язаній із обманом, нечесністю, списуванням, фабрикацією;

– не підроблювати документи;

- не поширювати неправдиву та компрометуючу інформацію про інших здобувачів вищої освіти, викладачів і співробітників;

- не отримувати і не пропонувати винагород за несправедливе отримання будь-яких переваг або здійснення впливу на зміну отриманої академічної оцінки;

– не залякувати й не проявляти агресії та насильства проти інших, сексуальні домагання;

– не завдавати шкоди матеріальним цінностям, матеріально-технічній базі університету та особистій власності інших студентів та/або працівників;

– не використовувати без дозволу ректорату (деканату) символіки університету в заходах, не пов'язаних з діяльністю університету;

– не здійснювати і не заохочувати будь-яких спроб, спрямованих на те, щоб за допомогою нечесних і негідних методів досягати власних корисних цілей;

– не завдавати загрози власному здоров'ю або безпеці іншим студентам та/або працівникам.

УСВІДОМЛЮЮ, що відповідно до чинного законодавства у разі недотримання Кодексу академічної доброчесності буду нести академічну та/або інші види відповідальності й до мене можуть бути застосовані заходи дисциплінарного характеру за порушення принципів академічної доброчесності.

09.04.2020 (дата) _____ (підпис) Корзун Валентин (ім'я, прізвище)