

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій інформатики факультету інформатики



**Побудова багаторівневого веб-застосування на платформі
Docker-контейнерів**

**Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» - 121**

Керівник курсової роботи

Кандидат технічних наук,

Старший викладач

Черкасов Д. І.

(підпис)

“ ____ ” _____ 2021 р.

Виконала студентка ІПЗ-4

Бутенко І.С.

“ ____ ” _____ 2021р.

«_____»_____2020 року

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

Студентки факультету інформатики

спеціальності «Інженерія програмного забезпечення» 4 курсу

Бутенко Ірині Сергіївні

Тема: Побудова багаторівневого веб-застосування на платформі Docker-
контейнерів

Вихідні дані:

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Календарний план

Анотація

Вступ

Розділ 1. Огляд існуючих рішень

Розділ 2. Структурна розробка власного рішення

Розділ 3. Детальна розробка компонента

Висновки

Додатки

Список використаної літератури

Дата видачі «_____»_____2020 року Керівник Черкасов Д. І. _____ (підпис)

Завдання отримала _____ (підпис)

Календарний план виконання роботи:

| № п | Назва етапу курсової роботи | Термін виконання етапу | Примітка |
|--------|---|---------------------------|----------|
| 1. | Отримання завдання курсової роботи. | 10.11.2020 | |
| 2. | Дослідження предметної області. | 03.12. 2020 | |
| 3. | Пошук матеріалу і написання першого розділу | 1.03.2021 | |
| 4. | Побудова структури веб-застосунку | 14.03.2021 | |
| 5. | Розробка архітектури веб-застосунку | 20.03.2021 | |
| 6. | Виконання практичної частини курсової роботи | 30.03.2021 | |
| 7. | Написання опису компонентів застосунку (третій розділ) | 04.04.2021 | |
| 8. | Аналіз виконаної роботи з куратором | 06.04.2021 | |
| 9. | Створення презентації і написання доповіді | 08.04.2021 | |
| 10. | Внесення змін в роботі | 10.04.2021 | |
| 11. | Здача роботи на перевірку на плагіат. | 12.04.2021 | |
| 12. | Захист курсової роботи | 19.04.2021 | |

Студент **Бутенко І.С.**Керівник **Черкасов Д. І.** “_____” _____

Зміст

| | | |
|-------|---|----|
| 1 | ВСТУП..... | 4 |
| 2 | Огляд існуючих рішень..... | 5 |
| 2.1 | Огляд архітектури різного рівня | 6 |
| 2.1.1 | Огляд однорівневої архітектури..... | 6 |
| 2.1.2 | Огляд дворівневої архітектури..... | 7 |
| 2.1.3 | Огляд тривірневої архітектури..... | 8 |
| 2.1.4 | Огляд багаторівневої архітектури | 9 |
| 2.1.5 | Порівняльний аналіз | 9 |
| 3 | Структурна розробка власного рішення..... | 11 |
| 3.1 | Структурна схема архітектури застосунку | 12 |
| 3.2 | Опис компонентів застосунку | 15 |
| 3.2.1 | Компонент API бекенду (api_backend)..... | 15 |
| 3.2.2 | Компонент адмін-панелі (admin_panel) | 16 |
| 3.2.3 | Компонент користувацького інтерфейсу (user_frontend)..... | 16 |
| 3.2.4 | Компонент веб-серверу та проксі-серверу (nginx_server)..... | 17 |
| 3.2.5 | Компонент бази даних (postgres_db)..... | 17 |
| 3.3 | Опис функціонування системи..... | 17 |
| 4 | Розробка компоненту API (api_backend) | 18 |
| 4.1 | Розробка маршрутизації API запитів urls.py..... | 18 |
| 4.2 | Розробка контролерів API views.py..... | 19 |
| 4.3 | Розробка моделі даних models.py..... | 20 |
| 4.4 | Розробка Dockerfile | 22 |
| 4.5 | Розробка файлу конфігурації для docker compose | 23 |
| 5 | Висновок..... | 23 |
| 6 | Список використаної літератури | 24 |
| 7 | Додатки..... | 26 |

1 ВСТУП

Через пандемію COVID-19 напрямок доставки їжі зафіксував колосальний зріст в 2020 році. Згідно з даними із ain.ua в 2021 році швидка доставка готової їжі з ресторанів виросте ще в 3-4 рази за рахунок зростання цифрових сервісів. Триватиме зростання доставки також з супермаркетів: в 6-8 разів. Багато ритейлерів і рестораторів будуть організовувати власні доставки. Збільшиться кількість закладів, які працюють в форматі dark kitchen. Попит на доставку їжі та продуктів у 2021 році буде рости, навіть якщо пандемія закінчиться.

Зважаючи на вищенаведені факти, для ресторану чи кафе в сучасних реаліях досить актуально мати інтернет-сервіс для доставки їжі. Для того, щоб платформа могла користуватись попитом та конкурувати з іншими, вона повинна задовольняти наступним критеріям:

- Застосунок повинен бути доступним за будь-яких умов;
- Представляти користувачам та адміністрації зрозумілий та зручний інтерфейс;
- Швидко реагувати на зміни бізнес-вимог, випереджати конкурентів;
- Давати можливість розробникам здійснювати зміни швидко, безпечно, щоб зберігати швидкість розвитку продукту.

Щоб реалізувати вищесказані вимоги, я вирішила використати концепцію мікросервісів. Архітектурний стиль мікросервісів — це підхід, при якому єдиний додаток будується як сукупність невеликих, самодостатніх, незалежних, не тісно зв'язаних сервісів, що спілкуються між собою за допомогою легких механізмів, наприклад: AMQP, HTTP, gRPC. Ці сервіси побудовані навколо бізнес-потреб (кожен відповідальний за конкретний процес) та розгортаються незалежно з використанням повністю автоматизованого середовища. Самі по собі сервіси можуть бути написані різними мовами і використовувати різні технології зберігання даних.

Існує великий вибір платформ для розробки застосунків на основі

контейнерів. Контейнери — це програмні пакунки, які містять всі необхідні компоненти для виконання застосування та використовують ядро батьківської операційної системи. Завдяки використанню контейнерів існує можливість максимальної ізоляції застосувань, або навіть компонентів застосувань, які виконуються на одному сервері, одне від одного та від самої батьківської системи.

Найпопулярнішою платформою є Docker, тому розробка веб-сервісу для доставки їжі на цій платформі стала темою моєї курсової роботи.

Суть моєї роботи полягає в наступному:

- Розробити веб-сайт доставки їжі, який зможе успішно конкурувати із подібними сервісами;
- Застосунок повинен бути доступним 365/24/7 (навіть під час оновлення функціоналу)
- Система повинна мати низьку зв'язність між основними компонентами системи та високе зчеплення коду в окремо взятому сервісі;
- Користувачі мають мати можливість, здійснювати замовлення без реєстрації;
- Для адміністрація має бути доступна зручна адмін-панель для відстеження замовлень та процесу їх виконання.

2 Огляд існуючих рішень

Розробку застосунку слід почати із вибору архітектурного рішення. Існують однорівневі, дворівневі, трирівневі та багаторівневі архітектури застосунків. Різниця між ними залежить від того, яким чином поділяються на частини ці компоненти. В однорівневій архітектурі вони всі є частинами однієї програми. У дворівневій архітектурі ці компоненти розділені на дві окремі частини. У трирівневій архітектурі компоненти розділені на три окремі частини.

В цьому розділі я пропоную розглянути наявні архітектурні рішення, їх

порівняти та обрати необхідну архітектуру.

2.1 Огляд архітектури різного рівня

2.1.1 Огляд однорівневої архітектури

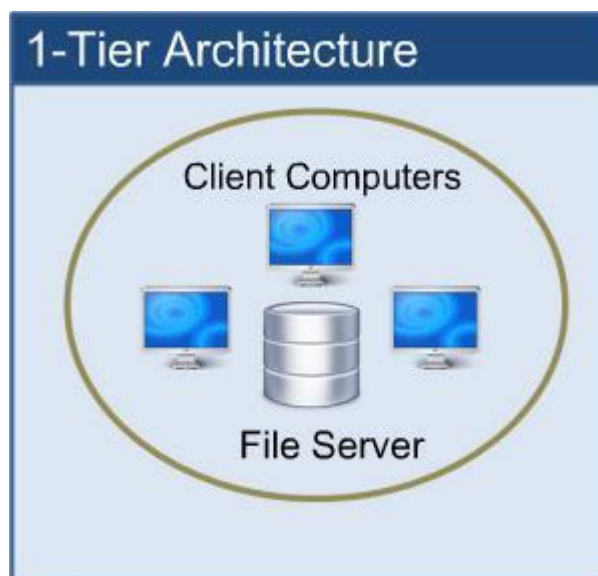


Рисунок 1— модель однорівневої архітектури

Однорівнева архітектура передбачає розміщення всіх необхідних компонентів, таких як інтерфейс, проміжне програмне забезпечення та back-end-файли, для програмного додатка або технології на одному сервері або платформі. У даній архітектурі клієнт здійснює тільки відображення інформації, що надається сервером, який несе все обчислювальне навантаження.

Даний тип системи є найпростішим і може мати сенс для простих програм, що містяться на одній платформі.

2.1.2 Огляд дворівневої архітектури

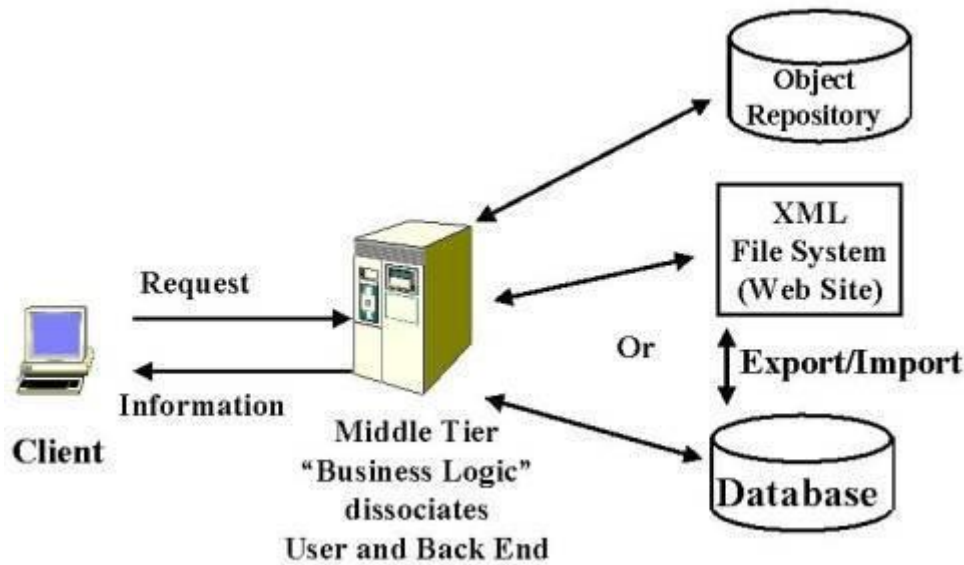


Рисунок 2 — модель дворівневої архітектури

У будь-якій сучасній мережі, яка використовує сучасні технології, присутні елементи клієнт-серверного взаємодії. З однорівневої архітектури вона стає дворівневою, оскільки між клієнтом та сервером необхідний розподіл трьох основних компонентів: взаємодію з користувачем, бізнес-логіку та управління ресурсами.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів — клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Такий підхід дозволяє застосунку мати більшу надійність та можливість для масштабування.

2.1.3 Огляд трирівневої архітектури

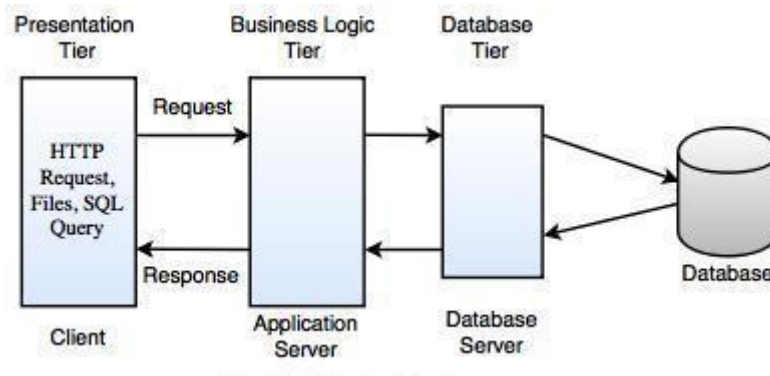


Рисунок 3— модель трирівневої архітектури

Подальший розвиток архітектури клієнт-сервер призвів до трирівневої архітектури. У такій архітектурі програми, які входять до складу рівня обробки, буде винесено на окремий сервер (сервер застосувань), але крім цього, можуть частково розміщені на машинах клієнтів і серверів.

В такій трирівневій архітектурі:

- програма-клієнт реалізує інтерфейс користувача, передає запити серверу застосувань і приймає від нього відповідь;
- сервер застосувань реалізує бізнес-логіку і звертається із запитами до сервера "третього рівня" (наприклад, сервера бази даних за даними);
- сервер третього рівня обслуговує запити сервера застосувань.

Ця архітектура була найпоширенішою протягом останніх років при створенні веб-застосувань, але останнім часом, з ростом популярності веб-технологій виникла потреба в зменшенні навантаження на веб-сервер, тому з'явилася потреба в розробці "товстих клієнтів" в рамках трирівневої архітектури.

2.1.4 Огляд багаторівневої архітектури

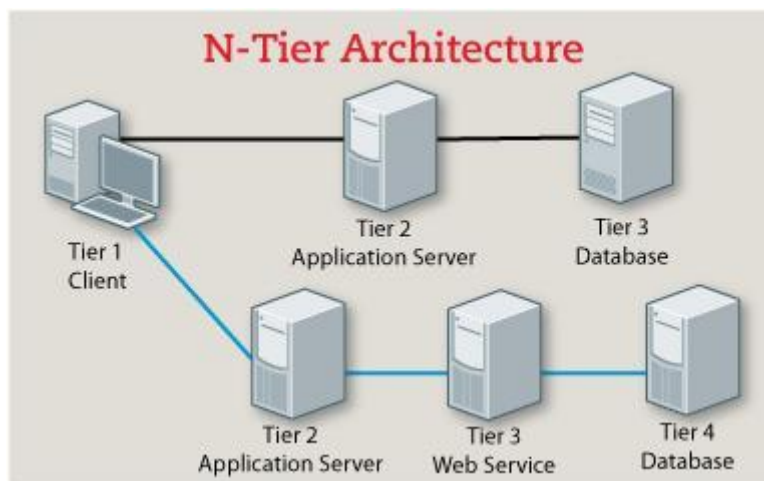


Рисунок 4— модель багаторівневої архітектури

Трирівнева архітектура може бути розширена до багаторівневої, шляхом виділення додаткових серверів для кожного рівня, кожен з яких буде представляти власні сервіси і надавати змогу користуватися послугами інших серверів різного рівня. N-рівнева архітектура додатка надає модель, за якою розробники можуть створювати гнучкі і повторно використовувані програми. Поділяючи додаток на рівні абстракції, розробники набувають можливість внесення змін в якийсь певний шар замість того, щоб переробляти весь додаток повністю. Трирівнева архітектура зазвичай складається з рівня уявлення, рівня бізнес-логіки й рівня зберігання даних.

2.1.5 Порівняльний аналіз

Порівняємо переваги та недоліки вищезгаданих архітектурних рішень.

| Тип архітектури | Переваги | Недоліки |
|-------------------------|--|---|
| Однорівнева архітектура | Простота застосунку, як і в розробці, так і в розгортанні та налаштуванні; | Невеликі можливості для масштабування; Проблеми із безпекою даних. |

| | | |
|------------------------|--|---|
| | Робота усього застосунку на одному пристрої; Необов'язковість мережевого з'єднання для роботи. | |
| Дворівнева архітектура | Можливість існування централізованого адміністрування прикладного програмного інтерфейсу; Розподілення навантаження на клієнт та сервер; Значне зниження трафіку в мережі, так як передаються виклики збережених процедур. | Невеликі можливості для масштабування; Розміщення бізнес-логіки на клієнтському ПЗ ; Можливі проблеми із безпекою користувацьких даних. |
| Трирівнева архітектура | Високу продуктивність, так як усі завдання розподіляються між різними серверами; Високий рівень безпеки, так як можна налаштувати свій принцип захисту на кожному з рівнів; Масштабованість; Низькі вимоги до продуктивності і технічних характеристик терміналів (клієнтів), як наслідок зниження їхньої вартості. | Більш висока складність створення додатків, в розгортанні та адмініструванні; Висока вартість обладнання; Адміністрування даної системи вимагає кваліфікованого професіонала. |
| Багаторівнева | Високий ступінь | Ще вища складність при |

| | |
|-------------|--|
| архітектура | масштабованості та створення додатків, в розподіленості; розгортанні та Високий рівень безпеки адмініструванні; даних; Висока вартість серверного обладнання; Високі можливості для налаштування специфікацій для кожного із серверів. |
|-------------|--|

Таблиця 1 — переваги та недоліки архітектурних рішень

Для розробки онлайн сервісу замовлення і доставки їжі я вирішила використати багаторівневу архітектуру, так як вона відповідає вимогам розроблюваного додатку та надає можливість розподіляти навантаження між компонентами.

3 Структурна розробка власного рішення

Основними вимогами до веб-сервісу для замовлення їжі, що має бути розроблений, є:

- Зв'язок між його компонентами повинен бути реалізований через HTTP протокол та REST API;
- Сервіс back-end повинен мати зручний API, щоб в майбутньому мати можливість для реалізації або інтеграції інших сервісів. Наприклад, інтеграція мобільних аплікацій з цим API;
- Компоненти повинні бути реалізовані на перевірених та стабільних фреймворках, технологіях;
- Наявність зручного та простого інтерфейсу, як для користувачів, так і для адміністрації веб-сервісу;
- Усі дані повинні зберігатись у реляційній базі даних;

Зважаючи на усі ці вимоги, я вирішила використати JavaScript- бібліотеку

React для побудови користувацького інтерфейсу. React - найпопулярніша бібліотека JavaScript для розробки користувацького інтерфейсу (UI). Також при розробці front-end був використаний CSS- фреймворк Bootstrap, котрий пропонує великий вибір різноманітних компонентів та класів для стилізації.

А для побудови сервісу backend був вибраний Python-фреймворк Django. Django описують як «веб-фреймворк для перфекціоністів з дедлайнами». Він був створений для якомога швидшої можливості переходу від прототипів до готових сервісів. Django містить величезну кількість функцій для вирішення більшості завдань веб-розробки, наприклад : ORM, міграції бази даних, аутентифікація користувача, панель адміністратора, форми та інше.

Для комунікації між back-end та front-end був використаний REST API, що реалізований за допомогою Django REST Framework.

В якості реляційної бази даних була вибрана PostgreSQL, так як вона має наступні переваги:

- підтримку БД необмежених розмірів;
- потужні і надійні механізми транзакцій ,реплікації;
- розширювану систему вбудованих мов програмування і підтримку завантаження C-сумісних модулів;
- легку масштабованість.

Щоб не писати вручну “сирі” SQL запити ,було вирішено використати Django ORM. Django ORM (Object Relational Mapping) є однією з найпотужніших особливостей Django-фреймворку, що дозволяє взаємодіяти з базою даних, використовуючи код Python, а не SQL.

3.1 Структурна схема архітектури застосунку

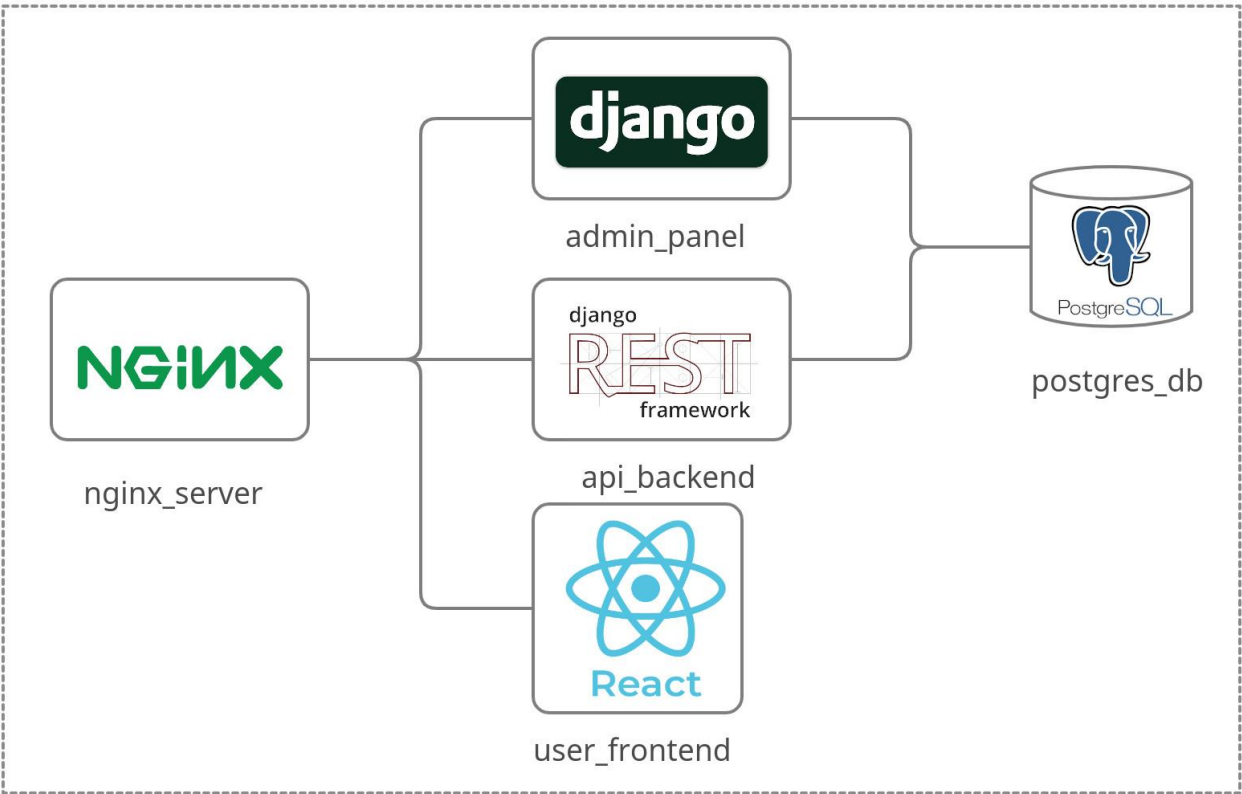


Рисунок 5— структурна схема веб-сервісу

| | | |
|-----|---------------------|---|
| 100 | Continue | Продовжити. Означає, що сервер отримав заголовок запиту, і клієнт повинен надіслати тіло запиту. |
| 101 | Switching Protocols | Перемикання протоколів. Говорить про те, що запитуюча сторона надсилає запит до сервера на здійснення процедури перемикання протоколів. |
| 200 | OK | Стандартна відповідь про успішне виконання HTTP запиту. |
| 201 | Created | Створено. Запит був виконаний і в результаті нього був створений новий ресурс. |
| 204 | No Content | Немає вмісту. Сервер успішно обробив запит, |

| | | |
|-----|--------------------|--|
| | | але не повертає вміст. |
| 301 | Moved Permanently | Ресурс переміщено назавжди. Цей і всі наступні запити повинні бути спрямовані по вказаному новому URI. |
| 302 | Found | Ресурс тимчасово переміщено. Використовується для виконання перенаправлення на іншу сторінку |
| 400 | Bad Request | Неправильний запит. Запит не може бути виконаний з причини невірної синтаксису. |
| 401 | Unauthorized | Несанкціонований доступ. Використовується спеціально у разі необхідності аутентифікації користувача, яка відбулася невдало |
| 403 | Forbidden | Заборонено. Запит був коректним, але сервер відмовляється відповідати на нього. |
| 404 | Not Found | Ресурс не знайдено, але він може бути доступний в майбутньому. |
| 405 | Method Not Allowed | Неприпустимий метод. Метод, з яким виконувався запит, не дозволено використовувати для заданого ресурсу. |

Таблиця 2— найпоширеніші статус коди

3.2 Опис компонентів застосунку

3.2.1 Компонент API бекенду (api_backend)

Компонент API бекенду написаний на мові Python з використанням фреймворків Django та Django REST Framework.

| Шлях | Тип запиту | Дія |
|------------|------------|---|
| items/ | GET | Надає список усіх наявних страв згідно з їх категоріями |
| cart/ | GET | Повертає вміст кошика: назва замовлення, його ціна та кількість. А також загальну ціну усіх замовлених страв та їх кількість. |
| cart/ | POST | Додає до кошика покупок нове замовлення або змінює кількість порцій уже наявного замовлення. |
| cart/ | DELETE | Очищає увесь вміст кошика. |
| cart/{id}/ | DELETE | Видаляє із кошика певну страву згідно її id. |
| order/ | POST | Надсилає вказані дані користувача на сервер для обробки та запису у базу даних. |

Таблиця 3— API шляхи компоненту api_backend

3.2.2 Компонент адмін-панелі (admin_panel)

Компонент адмін-панелі, подібно до `api_backend`, написаний на Python та Django. В Django є вбудований адміністративний інтерфейс, який підходить для роботи з контентом. Він створюється динамічно за допомогою читання метаданих моделі. Це призводить до появи готового інтерфейсу з авторизацією, який використовується для редагування контенту. Його можна відразу починати використовувати, налаштувавши лише спосіб відображення моделей у файлі `admin.py`.

В результаті цього адміністрація має змогу не лише бачити зроблені замовлення, а й редагувати, публікувати, видаляти інформацію з головної сторінки сайту.

3.2.3 Компонент користувацького інтерфейсу (user_frontend)

Компонент інтерфейсу для користувача реалізований на фреймворку React, а також для стилізації веб-сторінки використовується CSS фреймворк Bootstrap. Оскільки даний компонент не має прямого зв'язку із базою даних, а отримує необхідну інформацію в JSON-форматі, то для формування HTTP-запитів використана бібліотека `axios`.

Через те, що HTTP-протокол не дозволяє підтримувати постійний зв'язок з клієнтом, і кожен новий запит обробляється окремо, без прив'язки до попередніх, компоненту необхідно десь зберігати усі вибрані страви користувачем та їх кількість. Впоратись з цим завданням допомагає сесія браузера. Сесія браузера - це механізм, який дозволяє відстежити запити від одного браузера і зберегти деякі зміни під час переходів по сторінках сайту. З початком сесії на стороні сервера створюється файл, який містить інформацію про користувача, про його дії і події, які відбулися в рамках одного сеансу.

Таким чином, користувачу буде надано зручний спосіб для здійснення замовлення їжі без реєстрації на сайті, замість цього записуючи його

замовлення у сесію. При оформленні замовлення сесія відправляється на backend-сервер, за допомогою axios , де буде оброблена і внесена у реляційну базу даних.

3.2.4 Компонент веб-серверу та проксі-серверу (nginx_server)

Мій веб-застосунок містить три сервіси (API, адмін-панель та користувацький інтерфейс), що можуть обробляти різні запити, тому необхідний проксі-сервер, який би їх розподіляв у необхідні компоненти. Таким сервером є контейнер із Nginx.

Для конфігурації роботи Nginx призначений файл nginx.conf , у якому визначені правила маршрутизації запитів.

Наприклад, запити на URL адресу /admin переспрямовуються на компонент admin_panel, запити на api переспрямовуються на api_backend, а усі всі інші обробляюся за допомогою user_frontend.

3.2.5 Компонент бази даних (postgres_db)

Компонент бази даних postgres_db - Docker контейнер із базою даних PostgreSQL. Оскільки можливі ситуації, коли контейнер буде перезапускатись, я створила для нього том (volume) . Том - це файлова система, яка розташована на хост-машині за межами контейнерів. Таким чином буде забезпечено зберігання даних .Створенням і управлінням томами займається Docker.

3.3 Опис функціонування системи

При переході на початкову сторінку користувачу відкривається головна сторінка, на якій зображенні доступні варіанти страв для замовлення (додаток 1). Також страви поділені згідно з їх категоріями. У разі бажання замовити доставку цієї їжі, користувач може натиснути на кнопку “Додати у кошик”.

Таким чином, вибравши необхідні замовлення, користувач має змогу перейти на сторінку кошика замовлень і переглянути їх кількість та загальну суму (додаток 2). У разі, якщо все необхідне у бажаній кількості знаходиться у кошику, користувач має можливість за допомогою кнопки “Оформити” здійснити замовлення. Після натискання на “Оформити” відкривається модальне вікно, де необхідно ввести наступну інформацію: ім’я та прізвище, електронну адресу, номер телефону, адресу доставки та бажаний час доставки. Якщо усі дані введені коректно, то при натисканні кнопки “Замовити” усі дані про користувача та замовленні страви відправляються у JSON форматі на `api_backend`, де вони серіалізуються та записуються у базу даних. Після цього ці дані з’являються у адмін-панелі.

При вході у адмін-панель запитується логін та пароль (додаток 3), у разі успішної аутентифікації адміністратору буде надана можливість переглядати (додаток 4) категорії їжі, додавати нові, редагувати або видаляти вже наявні. Також є можливість прикріплювати до категорій нові страви і також здійснювати різні дії над ними (додаток 5). У разі необхідності адміністратор може переглядати список здійснених замовлень та їх статус (нове замовлення / виконане / скасоване) (додаток 6) або вибрати певне замовлення для перегляду детальнішої інформації або зміни статусу (додаток 7).

4 Розробка компоненту API (`api_backend`)

4.1 Розробка маршрутизації API запитів `urls.py`

Коли користувач запитує сторінку або API-endpoint на Django, система дотримується наступного алгоритму, щоб визначити, який код Python виконати:

- завантажує `urls.py` модуль і шукає змінну `urlpatterns`, яка є списком функцій `path()`. Кожна `path()` функція асоціює шаблон URL з контролером.

- Django проходить по кожному шаблону URL по порядку і зупиняється на першому, який відповідає URL запиту.
- Django імпортує і викликає відповідний контролер із `views.py`.

При розробці був створений `urls.py` такого вмісту:

```
from django.urls import path
from .views import ItemsByCategoryAPI, CartAPI, DeleteItemAPI, OrderAPI

urlpatterns = [
    path('items/', ItemsByCategoryAPI.as_view(), name='items-by-category-api'),
    path('cart/', CartAPI.as_view(), name='cart-api'),
    path('cart/<int:id>', DeleteItemAPI.as_view(), name='item-in-cart-api'),
    path('order/', OrderAPI.as_view(), name='create-order-api')
]
```

Тут я визначила 4 шаблони URL-запитів:

- ‘items/’ - для отримання доступних для замовлення страв та їх категорій, що є реалізоване у контролері `ItemsByCategoryAPI`;
- ‘cart/’ - для праці із корзиною, що реалізоване у контролері `CartAPI`;
- ‘cart/<int:id>’ - для праці із певним об’єктом у корзині, що є реалізоване у контролері `DeleteItemAPI`;
- ‘order’ — для комунікації із замовленням загалом, що є реалізоване у контролері `OrderAPI`.

4.2 Розробка контролерів API `views.py`

Відображення (view) або контролер — це місце, в якому міститься логіка роботи. Найпростіший спосіб їх написання, це — функції. Але контролери,

основані на класах, є альтернативний шляхом їх реалізації. Замість функцій тут використовуються об'єкти Python. Основними відмінностями є те, що організація обробки специфічних HTTP методів (GET, POST, і т.д.) рознесена по відповідним методам, замість того, щоб писати певні умови у функції та можливість використовувати міксини, що дозволяють виділяти код в компоненти, які можуть повторно використовуватися.

Усі мої контролери наслідуються від Django REST Framework класу `APIView`. Так як у даному компоненті використовуються лише GET, POST, DELETE запити, то і в контролерах використовуються `get()`, `post()`, `delete()` методи класів.

У файлі `views.py` було реалізовано чотири класи контролерів, для виконання певних типів запитів на заданні адреси:

- `ItemsByCategoryAPI` — підтримує лише GET запити;
- `CartAPI` — підтримує GET, POST, DELETE запити;
- `DeleteItemAPI` — підтримує лише DELETE запити;
- `OrderAPI` — підтримує лише POST запити;

Кожен метод із цих контролерів через моделі даних та ORM працює із базою даних PostgreSQL або із класом `Cart` (файл `cart.py`), у якому реалізована обробка сесій браузера, що дозволяє мати доступ до корзини замовлень.

4.3 Розробка моделі даних `models.py`

Так як компонент `api_backend` повинен працювати із даними з бази, то необхідно це робити, використовуючи мову запитів SQL, але набагато простіше мати певну абстракцію, що дозволяє більшу частину часу працювати зі звичними сутностями Python. Такою абстракцією є Object-Relational Mapping (ORM), що обертає сутності бази даних в структури мови. Але, насамперед, слід створити модель даних. Моделі визначають структуру даних, що зберігаються, включаючи типи полів, їх максимальний розмір, значення за замовчуванням, параметри списку вибору, текст довідки для документації,

текст міток для форм і т. д. Моделі зазвичай визначаються в файлі `models.py` (додаток 8). Вони реалізуються як підкласи `django.db.models.Model`, і можуть включати поля, методи і метадані. Модель може мати довільну кількість полів будь-якого типу - кожен представляє стовпець даних, який ми будемо зберігати в одній з таблиць бази даних. Кожен запис (рядок) бази даних буде складатися з одного значення кожного поля.

Для даного компоненту було розроблено наступні моделі:

- `Category` — модель для категорій. Для даної моделі визначене одне поле — назва категорії;
- `Item` — модель для страв. Дана модель містить поля для назви, опису, зображення, ціни та категорії даної страви;
- `ItemOrder` — модель, що містить об'єкт однієї страви із замовлення та її кількість;
- `Order` — модель замовлення. Містить дані користувача, бажаний час і адресу доставки, а також посилання на об'єкти `ItemOrder`.

Кожне поле Django моделі має свій тип. При реалізації моделей були використані наступні типи полів:

- `CharField` — поле для рядків малого і великого розміру;
- `TextField` — `CharField`, який перевіряє, чи є значення дійсною адресою електронної пошти;
- `DecimalField` — десяткове число з фіксованою точністю, представлене в Python екземпляром `Decimal`;
- `URLField` — `CharField` для URL;
- `ForeignKey` — зв'язок між полями типу “багато-до-одного”;
- `ManyToManyField` - зв'язок між полями типу “багато-до-багатьох”.

Для кожної моделі був реалізований метод `__str__()`, що реалізує текстовий опис об'єкту. Також для моделі `Order` був розроблений метод `get_order()` для обчислення фінальної ціни замовлення.

4.4 Розробка Dockerfile

Кожному контейнер Docker відповідає файл, який називається Dockerfile (додаток 8). Контейнери складаються з шарів. Кожен шар, крім останнього, що знаходиться поверх всіх інших, призначений тільки для читання. Dockerfile повідомляє системі Docker про те, які шари і в якому порядку треба додати в образ.

Ключове слово FROM повідомляє Docker про те, щоб при складанні образу використовувався базовий образ, який відповідає наданому імені. У моєму випадку це є образ Python версії 3.8.

Наступним кроком є визначення робочої директорії у контейнері. Інструкція WORKDIR автоматично створює директорію в тому випадку, якщо вона не існує.

Інструкція RUN дозволяє створити шар в процесі побудови образу. Після її виконання в образ додається новий шар, і його стан фіксується. Я використовую RUN для встановлення додаткових пакетів, що необхідні для коректної роботи через Python код із базою даних PostgreSQL. Також за допомогою цієї інструкції я встановлюю необхідні залежності, котрі перелічені у файлі requirements.txt.

Інструкція COPY повідомляє Docker про те, що потрібно взяти файли і папки з поточного контексту збірки і додати їх у поточну робочу директорію образу. Якщо цільова директорія не існує, ця інструкція її створить. Тому, використовуючи дану інструкцію ,я копіюю програмний код з поточної директорії у поточну директорії образу.

Фінальною інструкцією у моєму Dockerfile є ENTRYPOINT, яка позначає файл для виконання. В даному випадку це entrypoint.sh, котрий здійснює міграції бази даних.

4.5 Розробка файлу конфігурації для docker compose

Надалі образ контейнера обслуговується утилітою docker compose. Docker compose це — інструмент для створення і запуску Docker додатків, що містять декілька контейнерів. Використання її зазвичай розділяється на :

- Визначення сервісів ,з яких буде складатися ваш додаток в docker-compose.yml, надалі вони зможуть бути запуснені всі разом в ізолюваному оточенні;
- Виконання команди docker-compose up, яка запустить увесь застосунок.

У docker-compose.yml (додаток 10) для сервісу api_backend через інструкцію command назначена команда запуску. В результаті виконання цієї команди запускається Gunicorn із адресою 0.0.0.0:8000, який у свою чергу обслуговує Python код. Так як контейнер – ізолюваний компонент, інструкція expose вказує на те, який порт планується відкрити для того, щоб через них можна було б зв'язатися з контейнером, що працює. Для цього використано 8000 порт.

За допомогою інструкції volumes призначається том для довготривалого збереження статичних даних.

Інструкція depends_on вказує, що даний компонент слід запускати лише після успішного запуску бази даних.

5 Висновок

В результаті моєї курсової роботи було створено багаторівневе веб-застосування на Docker-платформі — прототип сайту для замовлення їжі. В застосунку приділяється велика увага розділенню на сервіси та можливості працювати над кожним окремо від інших.

Цей проект демонструє реалізацію багаторівневої мікросервісної архітектури, що містить 5 контейнерів. Для створення контейнерів, що

відповідають за представлення API та обробки даних, були використані наступні технології: Python, Django, Django ORM, Django REST Framework та Gunicorn. Для реалізації фронтенду: HTML/CSS, Bootstrap, JavaScript, React, Axios. Контейнер із проксі-сервером використовує Nginx, а контейнер зі базою даних — PostgreSQL.

Через те, що даний застосунок побудований на мікросервісній архітектурі, для управління контейнерами був використаний Docker Compose. Docker Compose дозволяє працювати із проектами, що складаються із декількох контейнерів, так як надає зручні інструменти для розгортання та налагодження проекту.

6 Список використаної літератури

1. Django Development with Docker Compose and Machine - <https://realpython.com/django-development-with-docker-compose-and-machine>
2. Отримуємо максимум від Django ORM - <https://codeguida.com/post/1952>
3. How to use Django with Gunicorn - <https://docs.djangoproject.com/en/3.2/howto/deployment/wsgi/gunicorn/>
4. Как использовать Django, PostgreSQL и Docker - <https://webdevblog.ru/kak-ispolzovat-django-postgresql-i-docker/>
5. Get started with Docker Compose - <https://docs.docker.com/compose/gettingstarted/>
6. Docker documentation - <https://docs.docker.com/get-started/>
7. How To Use the Official NGINX Docker Image - <https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/>
8. React in Docker with Nginx, built with multi-stage Docker builds, including testing - <https://tiangolo.medium.com/react-in-docker-with-nginx-built-with->

multi-stage-docker-builds-including-testing-8cc49d6ec305

9. Django REST Framework - <https://www.django-rest-framework.org/>
10. Python 3.8.9 documentation - <https://docs.python.org/3/>
11. How to use Axios with React (Everything you need to know) - <https://designrevision.com/react-axios/>
12. Axios documentation - <https://axios-http.com/docs/intro/>

7 Додатки



Доставка їжі Головна



Корзина

Паста



Паста Карбонара

Карбонара паста з жовтком і прошуто у білому вині з пармезаном. Інгредієнти: Паста, вершки, прошуто, дегідрований солоний жовток, бульйон італієць



Бурюкова паста з креветками

Паста-фреш буряка зі смаженими креветками та копченою фетою у поєднанні з міксом салату у білому вині. Інгредієнти: Паста



Паста з консервованим тунцем, томатами і оливками

Паста з тунцем та овочами у поєднанні з міксом салату та оливками у білому вині. Інгредієнти:



Паста з прошуто

Класична паста з прошуто, томатами чері та сиром пармезан. Інгредієнти: Фітучіні, томати чері, прошуто, вершки, бульйон італієць, сир біле вино, сир



Доставка їжі Головна



Корзина



Салат Фінський

Салат мікс, лосось гриль у кисло-солодкому соусі, лимон, картопля гриль, буряк запечений, аспарагус, огірок слайсами, паприка шматочками, соус Фіш

Додати у кошик



Салат з макреллю і картоплею

Салат мікс, макрель гриль, лимон, картопля гриль, кабачок гриль, цукровий горох, помідор, сушена цибуля, соус Ред

Додати у кошик



Салат з в'яленою олениною

Салат мікс, в'ялена оленина, бринза, в'ялені томати, помідори чері, крутони з житнього хліба, мікрогрін, соус з журавлини

Додати у кошик




Салат Грецький


Класичний Грецький салат зі свіжих овочів, сиром фетою та оливками у бальзамічному соусі. Інгредієнти: Томати, перець болгарський, сир фета, огірки очищені, томати чері очищені, маринована цибуля оливки, мікс салату, оливкова олія, бальзамічний чорний оцет, суміш перців, сіль, базилік сушений, орегано.

Додати у кошик




Додаток 1 (Головна сторінка веб-сервісу)

 Доставка їжі

Головна

 Корзина

Корзина

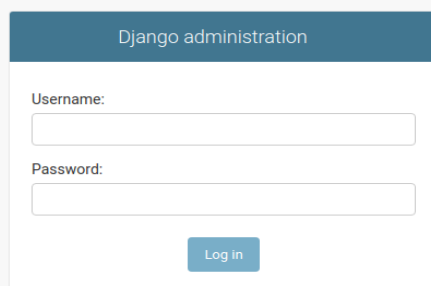
| Продукт | Ціна | Кількість | Загальна ціна |
|--|----------|---|---------------|
| Бограч | 90 грн. | 1 шт.  | 90 грн. |
| Крем-суп із зеленого горошку з беконом | 150 грн. | 1 шт.  | 150 грн. |
| Крем-суп з грибами | 255 грн. | 1 шт.  | 255 грн. |

Кількість порцій: 3 шт.
Загалом: 495 грн.

Оформити

2020 - Доставка їжі

Додаток 2(Вікно кошику)



Django administration

Username:

Password:

Log in

Додаток 3 (Вікно входу у адмін-панель)

Home › Delivery › Categorys

AUTHENTICATION AND AUTHORIZATION

Groups

+ Add

Users

+ Add

DELIVERY

Categorys

+ Add

Items

+ Add

Order items

+ Add

Orders

+ Add

«

Select category to change

ADD CATEGORY +

Action: Go 0 of 4 selected

☐ НАЗВА КАТЕГОРІЇ

☐ Пасты

☐ Десерты

☐ Супы

☐ Салаты

4 categorys

Додаток 4(Список категорій у адмін-панелі)

Home › Delivery › Items › Бограч

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

DELIVERY

Categories + Add

Items + Add

Order items + Add

Orders + Add

Change item

HISTORY

Бограч

Заголовок:

Ціна:

Категорія: Супи ✎ +

Опис:

Наваристий та пряний суп з копченостями та яловичиною.

Зображення:

Currently: https://stayhome.kiev.ua/image/cache/data/hatinka/photo_2020-11-24_07-10-59-600x600.jpg

Change:

Додаток 5(Форма редагування страви у адмін-панелі)

Home › Delivery › Orders

AUTHENTICATION AND AUTHORIZATION

Groups

+ Add

Users

+ Add

DELIVERY

Categorys

+ Add

Items

+ Add

Order items

+ Add

Orders

+ Add

The order "Сергій Глибокий" was added successfully.

Select order to change

ADD ORDER +

Action: Go 0 of 3 selected

| <input type="checkbox"/> | ПРИЗВИЩЕ ТА ІМ'Я | НОМЕР ТЕЛЕФОНУ | БАЖАНИЙ ЧАС ДОСТАВКИ | ЗАГАЛЬНА ЦІНА | СТАТУС ЗАМОВЛЕННЯ |
|--------------------------|------------------|----------------|----------------------|---------------|-------------------|
| <input type="checkbox"/> | Сергій Глибокий | 06346578453 | 14 | 470.00 | Нове замовлення |
| <input type="checkbox"/> | Олена Українка | 0325457342 | 11:20-11:50 | 195.00 | Доставлено |
| <input type="checkbox"/> | Олексій Іваненко | 0634212322 | 12:00 | 445.00 | Скасовано |

3 orders

Додаток 6(Загальний огляд замовлень)

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

DELIVERY

Categorys + Add

Items + Add

Order items + Add

Orders + Add

Сергій Глибокий

Статус замовлення: Нове замовлення

Продукти:

Крем-суп з грибами * 1
Салат з в'яленою олениною * 2
Паста Карбонара * 1
Чіа з манговим кюлі і сезонними ягодам * 1
Чіа з манговим кюлі і сезонними ягодам * 3
Салат Грецький * 1

Hold down "Control", or "Command" on a Mac, to select more than one.

Прізвище та ім'я: Сергій Глибокий

Email адреса: serhil@gmail.com

Номер телефону: 06346578453

Адреса доставки: Київ вул. Хрещатик 7

Бажаний час доставки: 14

Delete

Save and add another

Save and continue editing

SAVE

Додаток 7(Перегляд та редагування здійсненого замовлення)

Додаток 8 models.py

```
from decimal import Decimal
from django.db import models
```

```
class Category(models.Model):
```

```
    name = models.CharField(verbose_name='Назва категорії', max_length=100)
```

```
    def __str__(self):
```

```
        return self.name
```

```
class Item(models.Model):
```

```
    title = models.CharField(verbose_name='Заголовок', max_length=100)
```

```
    price = models.DecimalField(verbose_name='Ціна', max_digits=10,
decimal_places=2)
```

```
    category = models.ForeignKey(verbose_name='Категорія', to=Category,
on_delete=models.CASCADE, related_name='items')
```

```
    description = models.TextField(verbose_name='Опис', blank=True)
```

```
    image = models.URLField(verbose_name='Зображення')
```

```
    def __str__(self):
```

```
        return self.title
```

```
class OrderItem(models.Model):
```

```
    item = models.ForeignKey(verbose_name='Продукт', to=Item,
on_delete=models.CASCADE )
```

```
    quantity = models.IntegerField(verbose_name='Кількість порцій', default=1)
```

```
def __str__(self):
    return f'{self.item.title} * {self.quantity}'
```

```
def get_total_item_price(self):
    price = self.quantity * self.item.price

    return price
```

```
class Order(models.Model):
    class StatusOfOrder(models.TextChoices):
        NEW = 'NW', 'Нове замовлення'
        DELIVERED = 'DL', 'Доставлено'
        CANCELED = 'CN', 'Скасовано'

    status = models.CharField(verbose_name='Статус замовлення', max_length=2,
choices=StatusOfOrder.choices, default=StatusOfOrder.NEW)
    items = models.ManyToManyField(verbose_name='Продукти', to=OrderItem)
    name = models.CharField(verbose_name='Прізвище та ім\''я', max_length=50)
    email = models.EmailField(verbose_name='Email адреса')
    phone_number = models.CharField(verbose_name='Номер телефону',
max_length=12)
    address = models.CharField(verbose_name='Адреса доставки', max_length=100)
    time = models.CharField(verbose_name='Бажаний час доставки',
max_length=20)

    def get_total(self):
        total = Decimal()
        for order_item in self.items.all():
```

```
total += order_item.get_total_item_price()
```

```
return total
```

Додаток 9 Dockerfile

```
# pull official base image
```

```
FROM python:3.8.3-alpine
```

```
# set work directory
```

```
WORKDIR /usr/src/backend
```

```
# set environment variables
```

```
ENV PYTHONDONTWRITEBYTECODE 1
```

```
ENV PYTHONUNBUFFERED 1
```

```
# install psycpg2 dependencies
```

```
RUN apk update \
```

```
&& apk add postgresql-dev gcc python3-dev \
```

```
musl-dev libffi-dev libc-dev linux-headers openssl-dev cargo
```

```
# install dependencies
```

```
RUN pip install --upgrade pip
```

```
COPY ./requirements.txt .
```

```
RUN pip install -r requirements.txt
```

```
# copy entrypoint.sh
```

```
COPY ./entrypoint.sh .
```

```
# copy project
```

COPY ..

run entrypoint.sh

ENTRYPOINT ["/usr/src/backend/entrypoint.sh"]