МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна підтримка роботи малого підприємства»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Тиркусова Н.В.**

Студента групи ІНЗ – 61с — **Павловський О.В.**

СУМИ 2020

РЕФЕРАТ

Записка:63 стор., 19 рис., 18 джерел.

Об’єкт дослідження — організація технологічного процесу для міні-пекарні.

Мета роботи — розробити інформаційну систему технологічного і комерційного циклу міні-пекарні.

Методи дослідження — методи проектування, як для масштабу малого підприємства, так і можливості міграції проекту на більш потужні технології рівня підприємства.

Результати — створена інформаційна система для роботи міні-пекарні на CMS Wordpress, а також був розроблений пілотний проект для переходуна більш потужні інформаційні технології (Oracle-19c + Django Rest Framework (Python) + Vue(JavaScript)+DevOpps(Docker, Kubernetes)).

ВЕБ-СИСТЕМА ІНТЕРНЕТ-МАГАЗИНУ, ІНТЕРНЕТ МАГАЗИН, СИСТЕМА УПРАВЛІННЯ КОНТЕНТОМ, ХОСТИНГ,WEB,ADOBE, BRACKETS, HTML, CSS, PHP, JAVASCRIPT, OPENCART

ЗМІСТ

ВСТУП

На даний час, перед країною стоїть питання побудови нової, сучасної економічної моделі. Дивлячись на країни, які взяли гарний старт у розвитку, є країни які використовують в дуже багато сучасних технологій. Одним з гарних прикладів є електронні сервіси в виробництві малого і середнього бізнесу. Це стартапи, які не потребують на початку потужних капіталовкладень. А також в подальшому (в разі позитивних показників), можуть швидко з мінімальними затратами трансформуватися в щось потужне рівня підприємства наприклад. Або трансформуватись в систему публічного використання сервісів.

Мета роботи полягає в автоматизації початкової ланки малого бізнесу: міні пекарні і процесів, що дозволяють вести облік матеріалів необхідних для виробництва денної випічки, складеної на основі щоденних замовлень торгових точок.

Під час автоматизації бізнесу пекарні або булочної, завжди варто враховувати тенденції ринку. Наприклад, раніше формат магазину з булками або ж міні пекарні, переформувались в затишні заклади такі як кафе або ж кав'ярні з власним виробництвом булочних виробів. Міні пекарні стали масовим бізнесом, так як різноманітність асортименту пропонованих товарів, на багато більше ніж у магазині. Так само для управління великою кількістю виробів навіть в міні пекарні, потрібно сучасна система обліку. Адже облік в міні-пекарні - це найголовніша частина, що б можна було зайняти свою нішу в конкурентному середовищі.

Проект також пов’язує де-кільки ланок виробничого, та комерційного складу.

Робота виконана на замовлення підприємства «Міні-пекарня».

Проект використовує нові, сучасні в ІТ компаніях технології, що сприяє росту кількісних та якісних показників компанії.

1 Інформаційний огляд

1.1 Історія розвитку Електронної комерції

Галузь електронної комерції виникла з появою механізмів здійснення безготівкових операцій з віддаленим доступом до систем оплати. В якості транспортної системи для передачі даних, як правило, використовується всесвітня мережа Інтернет, характеристики якої не відповідають введеному стандарту Secure Electronic Transactions (SET), окрім локальних банківських систем з чітким розмежуванням доступу. Стрімкий розвиток і розширення мережі електронних магазинів потребує впровадження новітніх технологій.[6]

Електронна комерція - придбання або продаж товару (здійснення трансакції) за допомогою електронних носіїв чи через комп'ютерну мережу. Дане поняття може включати в себе замовлення, оплату та доставку товарів або послуг

Електронна комерція є одним з видів електронного бізнесу. Відповідно до документів ООН, бізнес класифікується як електронний, якщо хоча б дві його складові з чотирьох (виробництво товару або послуги, маркетинг, доставка і розрахунки) здійснюються за допомогою Інтернету. Тому в такій інтерпретації вважається, що покупка відноситься до електронної комерції, якщо, як мінімум, маркетинг (організація попиту) і розрахунки проводяться засобами Інтернету. Більш вузьке трактування поняття "електронна комерція" характеризує системи безготівкових розрахунків на основі пластикових карт.

Електронна трансакція представляє собою певну послідовність операцій, що ініціюється клієнтом (покупцем) або електронним магазином та виконується у віртуальній платіжній системі, наприклад: E-Gold (всесвітня); PayPal (Європа); WebMoney Transfer, CyberPlat (Росія); PayCash (Росія, Україна, Латвія, США та Великобританія).[6]

Системи електронних платежів користуються популярністю, про що свідчать показники кількості їх клієнтів. Наприклад, обіг універсальної платіжної системи CyberPlat за 2009 рік склав 160 млрд. рублів (123,5 млрд.

доларів). Також за даними НСМЕП (Національної Системи Масових Електронних Платежів), що діє в Україні, річний безготівковий обіг результатів електронних трансакцій за платіжними картками 2009 року склав 590,6 млн. гривень (для порівняння: у 2001році - 8,2 млн. гривень).

Отже, все більше користувачів бажають використовувати електронні платіжні системи, проте все більш актуальним стає питання забезпечення безпеки самого користувача - його персональних даних, номера рахунку і пароля доступу до платіжної системи. [6]

1.2 Постановка задачі

Вимоги до програмного продукту з боку замовника були такі: програма повинна давати можливість роботи декількох робочих (керівника підприємства, бухгалтера, економіста, постачальника, старшого пекаря, точок реалізації, відповідального за інформаційне забезпечення) місць, а саме:

- на основі попиту покупців формувати замовлення торгових точок, що працюють з пекарнею. Замовлення повино формуватися в режимі онлайн і надходити в пекарню на затвердження наступної випічки.

- на основі зводу прийнятих і затверджених замовлень, які складаються з виду готової продукції і їх кількості повинен формуватися розрахунок партії випічки і повинні бути враховані можливі виробничі витрати.

- при розрахунку виробничих витрат необхідно розрахувати складові інгредієнти для рецептів готової продукції.

- при розрахунку складові інгредієнтів необхідно врахувати їх кількість на складі на момент підготовки до випічки і визначити необхідні придбання в разі недостачі.

- сформувати поточну партію закупівель і логістику їх придбання, з можливим пропорційним збільшенням витрат на найближчий період закупівель.

- при визначенні кількості матеріалів і виробничих потужностей зробити випічку з розрахунком списання відповідних матеріалів зі складу і отримання готової продукції.

- формування відпустки поточної готової продукції по торговим точкам.

- при відпуску готової продукції необхідно враховувати можливі залишки і передачу між торговими точками, а також реалізацію товару.

- при заготівлі товарів на склад для реалізації випічки необхідно вести його облік і списання за нормами відповідно до рецептури.

- програма повинна давати можливість відстежувати можливість руху товару для контролю і аналізу за будь-який заданий період. А також при необхідності і можливості перерахунку і згортання залишків на будь-який період.

- для розробки готової продукції програма повинна давати можливість створення рецептів і їх інгредієнтів з нормою списання відповідних матеріалів.

- програма повинна давати можливість отримання електронних і друкованих копій до низки друкованих документів з можливою деталізацією.

- програма повинна давати можливість отримання аналітичного і статистичного звіту на зростання і зниження видів продукції за вказаний період і по окремих точках реалізації.

- програма повинна давати можливість роботи з одним документом різних служб з можливими блокуваннями коректованих значень або іншої логіки змін з урахуванням усунення конфліктів.

- програма повинна мати можливість для створення резервних копій на рівні адміністрування програми з можливістю отримання різних точок відновлення і моделювання ситуацій.

- в програмі повинен бути врахована можливість ведення накладних витрат для розрахунку собівартості.

- програма повинна мати можливість налаштувань для роботи декількох окремих пекарень і зміни їх реквізитів.

- програма повинна мати облік користувачів, у яких повинні бути різні операційні можливості і впусти доступу.

- програма повинна мати можливість масштабування для проведення реінжинірингу в робочих умовах з мінімальними витратами часу і фінансів.

- після визначення архітектури, її імплементації, і тестування в рамках наявних можливостей, необхідно визначити можливі шляхи та етапи розвитку програми. Можлива її стандартизація компонентів і дистрибуція для можливого тиражування.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Програмні засоби для поставленої задачі

Для реалізації вимог замовника до програмного забезпечення. З урахуванням вимог роботи різних професійних груп, матеріального становища і різних правових форм, бюджету та проект розроблений за допомогою системи керування вмістом сайту з відкритим вихідним кодом WordPress. Плагінів для WordPress, сторінок, написаних на PHP і сервер бази даних - MySQL. У написанні деяких сторінок використана бібліотека (JS) JQuery, для створення в деяких формах більш динамічного їх функціонування і перерахунку підсумкових рядків і колонок, а також асинхронної логіки читання і збереження даних (AJAX).[1]

Вибір WordPress був обумовлений з урахуванням його вже готових можливостей, а також популярності та підтримки на всіх хостингових платформах.

Так WordPress має вже реалізовану можливість роботи з базою даних і веденням користувачів, а також є можливість встановлення вже готових тим, плагінів, адмін панелі, з готовими і красивими дизайнами сторінок і багаторівневих призначених для користувача меню. Що з урахуванням часу і фінансування є одним з оптимальних виборів для проекту.[2]

Користування хостингом стало в разі завдання і також одним з оптимальних можливостей для малого бізнесу з відсутністю достатнього фінансування роботи власного веб-сервера а також сервера баз даних. А також постійної їх підтримки та багатьох функцій надаються хостингової платформою:

функції і настройки веб-сервера, сервера FTP, послуг резервування доменного імені, архівних копіювання, і інше використовуємо хостинг з наданої адміністративної панеллю.[3]

2.2 Система управління контентом

Будь-який сайт – це, перш за все, згрупований певним чином контент. Всі ці меню, блоки, графічні оформлення служать для того, щоб зручно організувати інформацію та підкреслити найцікавіші моменти. Виходячи з цього, дамо визначення CMS (Content Management System) – це сімейство платформ, завдяки яким можна доволі зручно створити сайт, тобто належним чином керувати контентом. Системи управління контентом на сьогоднішній день займають міцну позицію серед різноманітних сайтбілдерів.[7]

Система управління контентом – це спеціальна програма (сайтовий движок), яка встановлюється на хостинг (віддалений сервер, під´єднаний до Інтернету), і покликана виконувати 2 функції:

Показувати сторінки сайту користувачам, динамічно формуючи їх вміст із заздалегідь визначених шаблонів оформлення контенту (тобто текстів, малюнків, таблиць та інших матеріалів, що знаходяться у базі даних). Важливо розуміти, що при такій схемі сайту як набору сторінок не існує. Окремо існує дизайн (шаблон) і набір різноманітного типу матеріалів (файли, текст і т. п.) CMS будує сторінку користувачу у момент його запиту. Тобто движок генерує сторінки в залежності від внесених до матеріалів змін, а також індивідуального статусу користувача (наприклад, вміст кошика у випадку магазину). При цьому CMS намагається обробити якнайбільше запитів за одиницю часу, а також заважає спамерам засмічувати базу даних, слідкує за безпекою та виконує у фоновому режимі безліч інших дій, життєво необхідних для нормальної роботи сайту.[7]

Допомогти користувачу, який не має навичок програмування, просто і зручно керувати сайтом: публікувати сторінки, новини, викладати відео, робити посилання та багато іншого. Коротше кажучи, CMS забезпечує можливість створення сайту для більшості необізнаних у цьому питанні людей.[5]

Якими бувають CMS?

По-перше, існують платні (DLE, 1C Bitrix та ін.) та безкоштовні (WordPress, Drupal, Joomla, Opencart та ін.). Насправді їх набагато більше, ми лише навели приклади найбільш поширених платформ. По-друге, деякі з них мають вузьку спеціалізацію (Opencart, наприклад, заточений під створення магазинів, DLE найкраще себе показує на новинних порталах), але є і багато універсальних (Drupal, Joomla, MODX, Wordpress). Остання позиціонується як блоговий движок, але завдяки наявності великої кількості плагінів, на ньому можна створити навіть магазин.[7]

Взагалі, однією з основних фішок CMS є можливість встановлення допоміжних модулів/компонентів/плагінів, здатних значно розширити початковий функціонал. Простіше за все для створення певного типу сайту використовувати спеціалізовану CMS. Так ви отримаєте більшість важливих для реалізації проекту функцій вже з коробки без необхідності додатково насичувати систему плагінами, які можуть призвести до виникнення вразливостей.[7]

2.3 Сучасні варіанти CMS для інтернет-магазинів

Платні движки для інтернет-магазину:

1С-БІТРИКС

Ця CMS займає перше місце за популярністю серед платних движків інтернет-магазину, широко розсповсюджена, насамперед, у корпоративному секторі. Її відносять до систем промислового рівня, а це означає, що для відповідності вимогам власника сайта і зручного управління інтернет-магазином, необхідно добре налаштувати сам движок. Про це говорить і програма сертифікації фахівців від розробників. Також ця платформа – одна з найбільш вимогливих до ресурсів сервера і, щоб движок інтернет-магазину працював стабільно, необхідно використовувати потужний хостинг.[8]

Переваги 1С-Бітрікс:

Вартість ліцензії на редакції для інтернет-магазину: 9845 гривень і 17 800 гривень.

Велика кількість опцій для адміністративної частини та інтерфейсу користувача.

Маркетплейс із великою кількістю модулів і платних шаблонів.

Безшовна інтеграція з 1С для автоматизації роботи.

Движок інтернет-магазину надає готові рішення із коробки без несподіванок і темних плям.[8]

Технічна підтримка на безпрецедентно високому рівні.

Движок для онлайн магазину 1С-Бітрікс гарантує більш високу безпеку, ніж у будь-яких open source систем.

Недоліки 1С-Бітрікс:

Велика кількість розробників, але ціни на роботу вище середніх на ринку.

Користувачі відзначають певну перевантаженість інтерфейсу управління і вимогливість системи до ресурсів сервера.

Чудовий (багато у чому) движок для інтернет-магазину пропонує шаблонні рішення (ви зустрінете десятки таких самих), а оплата індивідуального дизайнера (плюс програміст і верстка) обійдеться у кругленьку суму.[8]

Фактично вартість розробки інтернет-магазину на “Бітрікс” починається з 20 000 гривень, а середня ринкова ціна – приблизно на рівні 70 000 гривень.

UMI-CMS

Складаючи рейтинг “Популярні движки інтернет-магазинів”, друге місце за популярністю серед розробників інтернет-магазинів доведеться віддати UMI.CMS. Вартість редакцій із функціоналом електронної комерції нижча, порівняно з «Бітрікс», – 22 900 рублів і 34 900 рублів. Інтерфейс системи простіший для освоєння новачком, і вимогливість до ресурсів менша. Правда, незважаючи на три десятки доступних модулів, відсутній маркетплейс доповнень, а значить, для реалізації нестандартних додаткових робіт доведеться шукати розробників на стороні.

Переваги UMI.CMS:

Нескладна в освоєнні адміністративна панель, додавання товарів і управління каталогом реалізовані досить зручно.

Хороша техпідтримка, структурована документація як у текстовому, так і в відеоформаті.

Недоліки UMI.CMS:

У безкоштовному доступі шаблонів дизайну практично не знайти, платних теж мало.

Для доопрацювання системи можна скористатися послугами не кожного фахівця, що відзначається на ціні.

За великої кількості сторінок у структурі керувати сайтом стає незручно.

Відсутній маркетплейс доповнень, що накладає обмеження при розвитку проекту.

Потрібно відзначити, що, на відміну від російського ринку, в Україні ця система не досить популярна. Одна з причин цього – специфічна програмна мова, яка використовується при розробці платформи.

Безкоштовний скрипт інтернет-магазину: для кого він підійде та приклади

MAGENTO

Одна з найпопулярніших CMS для інтернет-магазинів в США та інших західних ринках. У крахнах колишнього СНГ на цій безкоштовній платформі магазини створюють рідше, оскільки більша частина документації доступна тільки англійською, і далеко не кожен фахівець працює з цим движком. Одна з ключових переваг Magento – велика кількість можливих для використання шаблонів, плагінів та інших розширень, які значно спрощують додавання нової функціональності. Щоправда, більшість із них платні.[8]

Переваги движка Magento:

Розвинена платформа з великою функціональністю і широкими можливостями налаштування.

Велика кількість доступних розширень в маркетплейсі як платних так і безкоштовних.

Потенціал масштабування дозволяє не боятися можливих проблем при розвитку інтернет-магазину в майбутньому.

Недоліки движка Magento:

Документація переважно англійською, що створює певний мовний бар'єр.

Для високої продуктивності і швидкості роботи знадобиться виділений сервер, замість звичайного хостингу.

Погано підходить для малого бізнесу, це рішення зазвичай використовується для великих підприємств.

Вартість розробки інтернет-магазину на Magento у середньому за ринком становить $2000 і вище. При цьому додаткові складнощі виникають у плані підтримки і в процесі просування.

OPENCART

Один із найпростіших в управлінні движків для інтернет-магазинів, який досить популярний в Рунеті, посідає третє місце в рейтингах. Основна причина цього – легке освоєння адмінпанелі, простота додавання та управління товарами. Щоправда, мінусів у OpenCart теж вистачає 1.5, у новій 2.3 все набагато краще. Це і конфлікти нових версій зі старими доповненнями, генерація дублів сторінок (негативно відзначається на SEO), обмеженість у плані масштабування. Загалом, якщо ви плануєте не зупинятися на інтернет-магазині з кількома сотнями товарів, ця CMS вам, вочевидь, не підходить. Простота запуску обертається витратами, пов'язаними з необхідністю постійних доробок і виправлення помилок.[8]

Переваги OpenCart 1.5:

Відкритий код.

Простота управління інтернет-магазином з боку користувача.

Багато плагінів як платних, так і безкоштовних.

Сама система проста і «мало важить», без проблем буде працювати і на віртуальному хостингу.

Невисока вартість додаткових робіт, так як фахівців, що працюють з даною системою, багато.

для функціоналу магазину движок пропонує багато корисних речей – від знижок, купонів і акцій до характеристик і груп користувачів.

Недоліки OpenCart 1.5:

Дуже рідкісні оновлення.

Недостатня тех. підтримка навіть на комерційних модулях.

Шаблони дизайну зазвичай адаптовані під певну версію движка, що створює проблеми при їхньому використанні з виходом нового релізу.

Велика кількість дублів сторінок, які генеруються движком – однин з основних його недоліків.

Базовий функціонал – мінімальний.

Загалом, це варіант тільки для маленьких і обмежених в плані функціоналу сайтів. Їхні власники можуть з повним правом казати, що OpenCart – найкраща cms для інтернет-магазинів. Але при необхідності розвитку інтернет-магазину під управлінням OpenCart потрібно чимало доробок і складності з масштабуванням будуть ключовою проблемою, а відсутність необхідної підтримки створить проблеми при самостійному використанні движка.[8]

DRUPAL

Гарний движок для інтернет-магазину, написаний на мові PHP. Цей open source продукт захищений ліцензією GPL. Його розвитку сприяють програмісти-ентузіасти із різних країн світу. На офіційному сайті ви можете побачити й оцінити приклади веб-ресурсів, створених на базі цієї CMS (виберіть розділ «Who Uses Drupal» і вкажіть у фільтрах “електронна комерція”).

Переваги CMS Drupal:

Чудове ком’юніті.

На офіційному сайті є безліч (понад 30 тисяч) безкоштовних модулів, за допомогою яких ви легко зможете значно збільшити функціонал магазину.

Численне відкрите співтовариство грамотних розробників і користувачів.

Гідна система внутрішнього пошуку, цілком придатна для роботи.

Мінуси використання Drupal:

Для користувачів початківців платформа може виявитися складною і недостатньо доброзичливою.

Навчання займає багато часу.

Кількість якісних тем обмежена.

Періодично виникають критичні вразливості, для швидкого усунення яких бажано включити модуль автоматичного оновлення.[8]

Платформа Drupal, незважаючи на певні недоліки, – одна із кращих безкоштовних CMS (і найбільш функціональних). При цьому не можна рекомендувати її для розробки інтернет-магазину значних розмірів, хоча б через згадані вразливості, навіть враховуючи те, що усунути їх можна оперативно.[8]

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Режими роботи програми

Згідно вимог замовника були розроблені наступні режими роботи програми:

3.1.1 Номенклатура

Режим, що дозволяє вести наявність матеріалів для випічки і їх загальну ціну.

Для ведення існують режими:

- Створити нову номенклатуру - за допомогою якої можливо створювати нову позицію номенклатури її найменування.

- Режим редагування дає можливість редагування полів при неправильному введенні.

- Режим видалення існує для видалення рядка номенклатури (якщо в режимі рецептів немає посилання на цей товар).

Номенклатура:Создать новую номенклатуру

Ид.записи	Наименование	Количество		Цена			
1	Ванілін	0.061	кг	15	грн.	Редактировать	Удалить
2	Висівки пшеничні	37.835	кг	160.8	грн.	Редактировать	Удалить

Рисунок 3.1 – Режим «Номенклатура»

3.1.2 Рецепт.

Режим дозволяє вести список рецептів і список продуктів, що складають ці рецепти. Список рецептів ведеться подібно списку номенклатур в якому є можливість Створити, Редагувати і видалити рецепт. Також існує можливість введення роздрібної ціни. Введення складових можливий при розкритті списку продуктів, що входять в рецепт (режим: Рецепт показати / приховати). Де також можливе

введення, коригування товарів, що входять в рецепт і їх кількість в рецепті.

Рисунок 3.2– Режим «Рецепт»

3.1.3 Розрахунок

Режим дозволяє ввести замовлення по складеної рецептурою для випічки або зібрати замовлення (Кнопка: Підрахувати замовлення для пекарні з замовлень торгових точок) з замовлень торгових точок (Режим: Накладна колонка Замовлення). І підрахувати витрата товарів (Кнопка: Розрахувати залишки товарів на складі відповідно до замовлення) з можливістю відображення залишків товару і їх нестачу (Червоний колір).

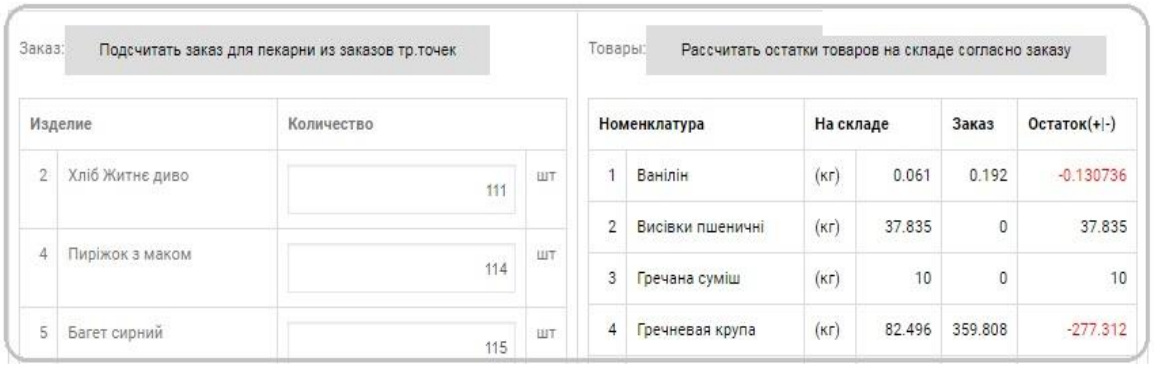


Рисунок 3.3 – Режим «Розрахунок»

3.1.4. Торгові точки.

Режим дозволяє вести список торгових точок їх найменування, відповідальна особа і логін відповідальної особи.


Торговые точки:Создать новые торговые точки

Ид.записи	Наименование	ФИО ответственного	Логин ответственного		
1	Магазин	[illegible]	mag1	Редактировать	Удалить
2	[illegible]	[illegible]	mag2	Редактировать	Удалить

Рисунок 3.4 – Режим «Торгові точки»

3.1.5 Накладні витрати.

Режим дає можливість ведення наявних накладних витрат на одиницю продукції. Накладні витрати виводяться в режимі друку рецептів і допомагають скласти можливість розрахунку собівартості продукції (Режим: Вихідні форми пункт: Рецепт).


Номенклатура:Создать новую номенклатуру

Ид.записи	Наименование	Количество		Цена			
1	Ванілін	0.061	кг	15	грн.	Редактировать	Удалить
2	Висівки пшеничні	37.835	кг	160.8	грн.	Редактировать	Удалить

Рисунок 3.5 – Режим «Накладні витрати»

3.1.6 .Передача товару

Режим дозволяє організувати і контролювати передачу товару з однієї торгової точки на іншу. При вході в режим в випадаючому списку вибирається торгова точка з якої відбувається передача товару (відповідальні по торгових точках можуть вибрати тільки свою торговельну точку). Вводиться нова передача. Вибирається зі списку торгова точка на яку передається товар, вибирається сам товар і

вводиться його кількість. Є можливість відредагувати і видалити створену передачу до того часу поки що приймає торгова точка не підтвердить приймання товару і введе своє кількість прийнятого товару. Після цього приймання закривається для редагування і режими Редагувати і Видалити стають недоступними. І передає бачить реакцію приймаючої сторони (кількість прийнятого товару). Є також тимчасові позначки передачі і приймання товару. Можливо робити будь-яку кількість передач товару в день.

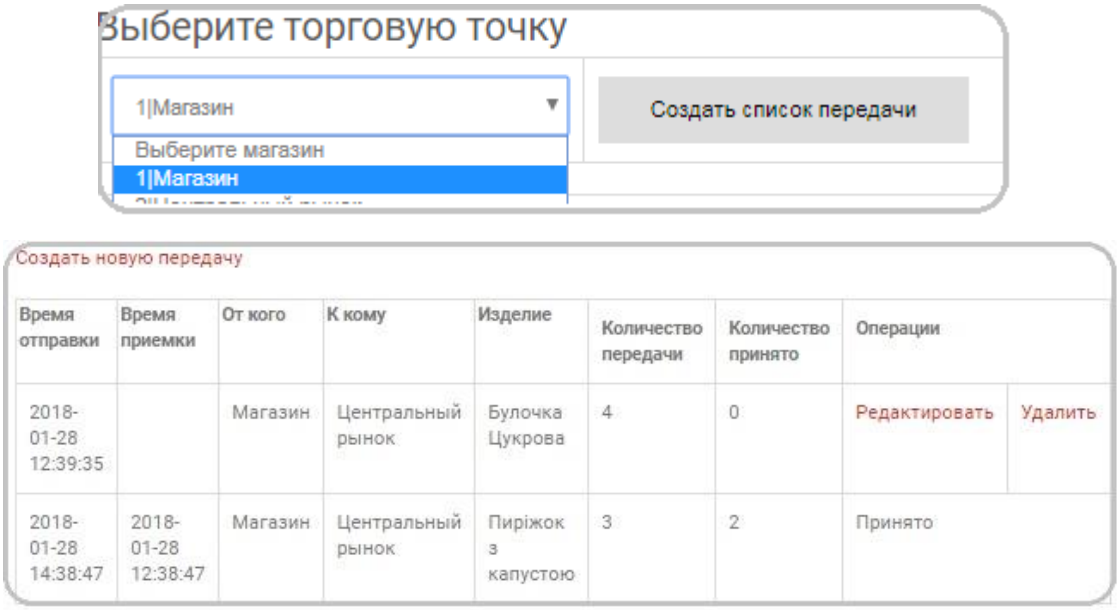


Рисунок 3.6– Режим «Передача товару»

3.1.7 Прийом товару.

Режим дозволяє переглядати всі запити на передачу товару і вимагають підтвердження і вже підтверджені. При вході в режим в випадаючому списку вибирається торгова точка з якої відбувається передача товару (відповідальні по торгових точках можуть вибрати тільки свою торговельну точку). Потребують підтвердження запити на передачу товару дають можливість редагувати позицію приймання. Якщо кількість прийнятого товару відповідає переданому вводиться та ж цифра, що і на передачу або та, яка була фактично. Після введення, що підтверджує

кількість передача вважається закритою. Показники по передачі, що йдуть до звіту вважаються підтвердженими.

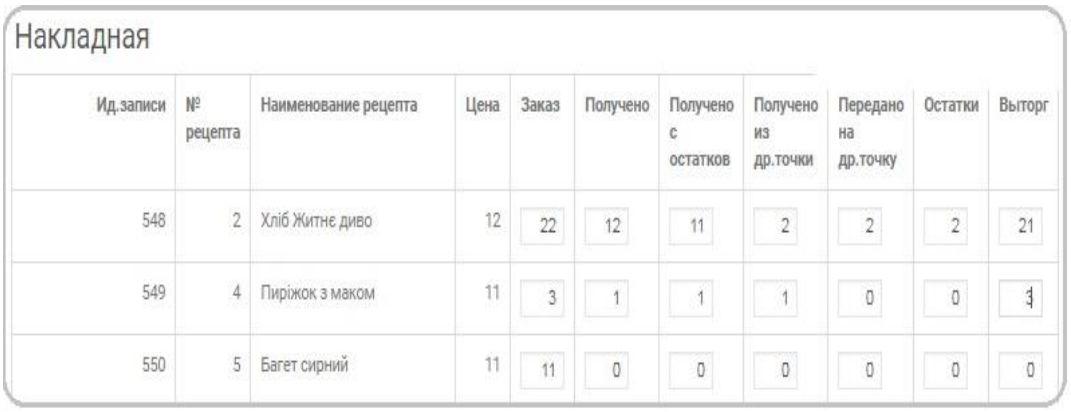
Приемка товара

Время отправки	Время приемки	От кого	К кому	Изделие	Количество передачи	Количество принято	Операции
2018-01-28 12:39:35		Магазин	Центральный рынок	Булочка Цукрова	4	0	Редактировать
2018-01-28 14:38:47	2018-01-28 12:38:47	Магазин	Центральный рынок	Пиріжок з капустою	3	2	Принято

Рисунок 3.7– Режим «Прийом товару»

3.1.8 Накладна.(Заказ)

При вході в режим в випадаючому списку вибирається торгова точка з якої відбувається передача товару (відповідальні по торгових точках можуть вибрати тільки свою торговельну точку). Робота в даному режимі ведеться з накладними за поточну дату і дані для кожного магазину можливо вводити по кожній товарній позиції. Якщо робота зі звітом ведеться перший раз в день, то відбувається автоматичне створення рядків накладної для обраного магазину по кожній товарній позиції (Кнопка: Створити / редагувати накладну). Якщо накладна вже створена, то ведеться редагування вже створеної і заповненої накладної. Є можливість створити нову накладну за поточний день (Кнопка: Створити нову накладну). Режим дозволяє вести звітність по торговій точці для прийнятого, переданого і реалізованого товару, а також можливість здійснити замовлення. Колонки приймання передачі заповнюються автоматично при вході. Кількість товару вводиться по кожній товарній позиції. Для проведення замовлення в графу Замовлення - вводиться кількість товару, що замовляється, яке використовується для планування замовлень пекарні.



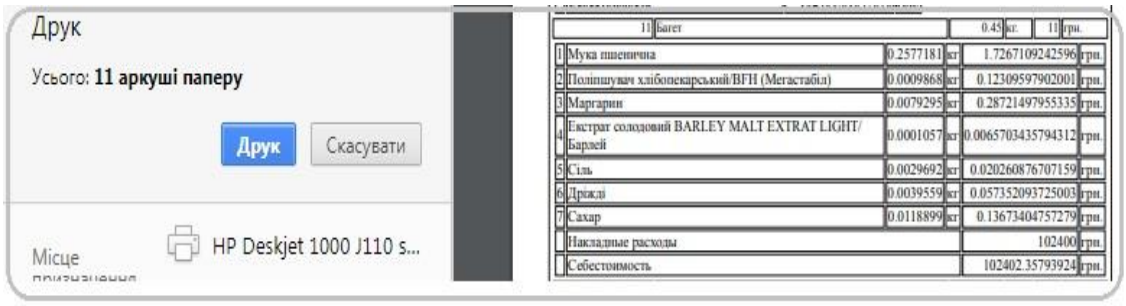
Накладная

Ид.записи	№ рецепта	Наименование рецепта	Цена	Заказ	Получено	Получено с остатков	Получено из др.точки	Передано на др.точку	Остатки	Выторг
548	2	Хліб Житнє диво	12	22	12	11	2	2	2	21
549	4	Пиріжок з маком	11	3	1	1	1	0	0	3
550	5	Багет сирний	11	11	0	0	0	0	0	0

Рисунок 3.8– Режим «Накладна»

3.1.9 Вихідні форми.

Режим дає можливість отримання таблиць вихідних форм і при необхідності їх збереження в файл (Кнопка: Друк). Вибір форм проводиться зі списку. Форми доступні згідно з визначеними для користувача ролей.



11	Багет			0.45 кг.	11 грн.
1	Мука пшенична	0.2577181	кг	1.72671092425961	грн.
2	Поліпшувач хлібопекарський/ВГН (Мегастабіл)	0.0009868	кг	0.123095979020011	грн.
3	Маргарин	0.0079295	кг	0.287214979553351	грн.
4	Екстракт солодовий BARLEY MALT EXTRAT LIGHT/ Ячмінь	0.0001057	кг	0.006570343579431	грн.
5	Сіль	0.0029692	кг	0.020260876707159	грн.
6	Дріжді	0.0039559	кг	0.057352093725003	грн.
7	Сахар	0.0118899	кг	0.136734047572791	грн.
	Накладные расходы			102400	грн.
	Себестоимость			102402.357939241	грн.

Рисунок 3.9 – Режим «Вихідні форми»

3.2 Опис структури даних

Опис бази даних для проекту:

1. Таблиця номенклатури товарів на складі.

```
CREATE TABLE `wp_tt_nomenkl` (
`ts` timestamp DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
 `id` bigint(20) ,
 `name` varchar(50) ,
```

```
  `cn` double ,
  `zn` double ,
  `dat` timestamp NULL DEFAULT NULL,
  `cn_base` double ,
  `zn_base` double ,
  `dat_base` timestamp  DEFAULT CURRENT_TIMESTAMP,
  `ed` varchar(10) ,
  `zakaz` double ,
  `zakaz_zn` double ,
  `avg_cost` double ,
  `avg_cost_base` double ,
  `rizn_cn` double
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

2. Таблица готових виробів:

```
CREATE TABLE `wp_tt_recept` (
  `ts` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `id` bigint(20) NOT NULL,
  `name` varchar(50) NOT NULL,
  `w` double NOT NULL,
  `zakaz` int(11) NOT NULL,
  `koef` int(11) NOT NULL,
  `rozn` double NOT NULL,
  `fakt` bigint(20) NOT NULL,
  `fakt_zn` double NOT NULL,
  `ind` int(11) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

3. Таблиця складових готових виробів:

```
CREATE TABLE `wp_tt_recept_staf` (
```

```
`ts` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
`id_recept` bigint(20) NOT NULL,
`id_nom` bigint(20) NOT NULL,
`id` bigint(20) NOT NULL,
`name` varchar(50) NOT NULL,
`cn` double(24,7) DEFAULT NULL,
`ed` varchar(10) NOT NULL,
`zakaz` int(11) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

4. Таблиця для випічки:

```
CREATE TABLE `wp_tt_vipechka` (
`ts` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
`id_recept` bigint(20) NOT NULL,
`dat` date NOT NULL,
`zakaz` bigint(20) NOT NULL,
`fakt` bigint(20) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

5. Таблиця накладних відпустки товару:

```
CREATE TABLE `wp_tt_magaz_nakl` (
`ts` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
`id` bigint(20) NOT NULL,
`id_recept` bigint(20) NOT NULL,
`id_magaz` bigint(20) NOT NULL,
`name_recept` varchar(50) NOT NULL,
`rozn` double NOT NULL,
`date_nakl` date NOT NULL,
`zakaz` int(11) NOT NULL,
```

```
`poluch` int(11) NOT NULL,
`poluch_ostat` int(11) NOT NULL,
`from_point` int(11) NOT NULL,
`to_point` int(11) NOT NULL,
`ostat` int(11) NOT NULL,
`vitorg` int(11) NOT NULL,
`xzakaz` double NOT NULL,
`xpoluch` double NOT NULL,
`xpoluch_ostat` double NOT NULL,
`xfrom_point` double NOT NULL,
`xto_point` double NOT NULL,
`xostat` double NOT NULL,
`xvitorg` double NOT NULL,
`ind` int(11) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

6. Таблиця магазинів:

```
CREATE TABLE `wp_tt_magaz` (
`ts` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
`id` bigint(20) NOT NULL,
`name` varchar(50) NOT NULL,
`fio` varchar(50) NOT NULL,
`login` varchar(50) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

7. Інші менш важливі таблиці проекту (в тому числі і таблиці для створення архіву *_arc) і допомагаючі таблиці WordPress.

Сторінки на PHP:

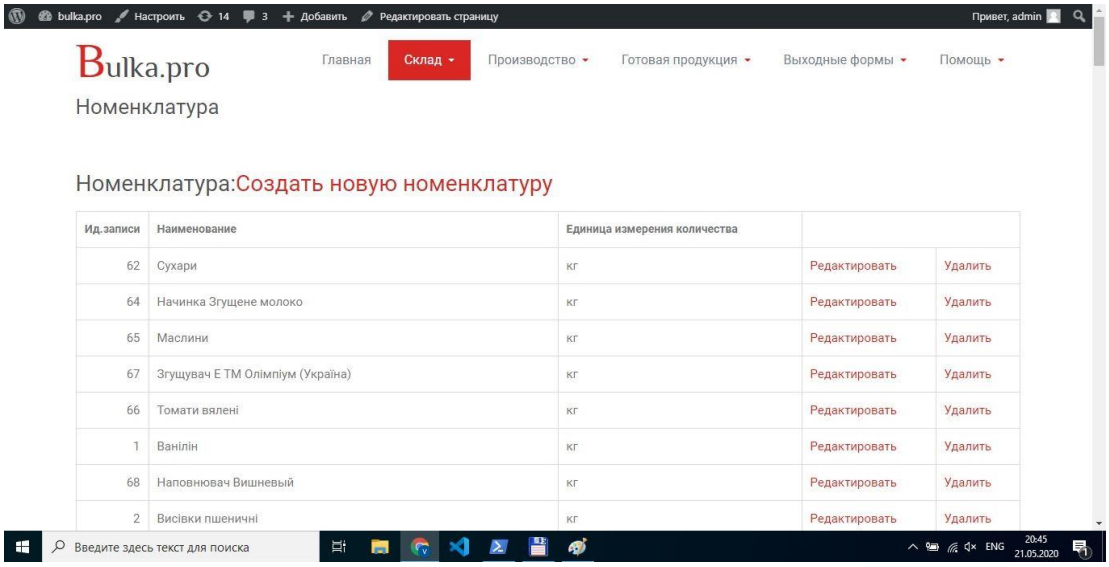


Рисунок 3.10– Сторінка «Номенклатура»

```
[exec]
$current_user = wp_get_current_user();
$cur_user_id = get_current_user_id();
$cls=$wpdb->get_var("SELECT close_on_develop FROM wp_tt_conf");
if($cls==1&&$cur_user_id!=1 ){
    print "<h1>Програма на реконструкції";
}else{

$npg=5;
$h="http://www.bulka.pro/?page_id=$npg";
$actt=$_GET['a'];
$idd=$_GET['idd'];
$fname=$_GET['nam'];
$cn=$_GET['cn'];
$ed=$_GET['ed'];
$zn=$_GET['zn'];
$cur_user_id = get_current_user_id();
```

```
if($cur_user_id>0&&$cur_user_id<3){
print "<h1>Номенклатура:";
print "<a href='".$h."&a=1'>Створити нову номенклатуру </a>";
print "</h1><br>";

if($actt==21){
  $wpdb->query("UPDATE wp_tt_nomenkl SET id = $idd, name='$fname', ed='$ed' WHERE id = $idd ");
  $actt=0;
}
if($actt==3){
  $wpdb->query("DELETE FROM wp_tt_nomenkl  WHERE id = $idd ");
  $actt=0;
}


if($actt==11){
 $wpdb->insert('wp_tt_nomenkl', array(
    'id' => $idd,
    'name' => $fname,
    'ed' => $ed,
));
  $actt=0;
}
$posts = $wpdb->get_results("SELECT id, name, cn, zn, dat, cn_base, zn_base, dat_base, ed FROM wp_tt_nomenkl ");
if($actt==2||$actt==1){
  print "<form name='formSave' id='formSave' action='".$h."' method='GET'>";
  print "<input type='hidden' name='page_id' value='$npg'>";
```

```
}

print "<table>";
print " <tr style='padding: 2px 2px; font-size: 14px;    width: 36px;'>";
 print " <th style='text-align:right; height: 14px; width: 46px;'>Ид.записи</th>";
 print " <th >Найменування </th>";
 print " <th colspan=1>Єдиниця вимірення кількості</th>";
 print " <th colspan=2></th>";
print " </tr>";
if($actt==1){
$maxcn = $wpdb->get_var("SELECT  max(id)+1 FROM wp_tt_nomenkl");

print " <tr>";
print " <td  align='right'><input style='text-align:right;' type=text size=4 name='idd' value='".$maxcn."'></td>";
print " <td><input type=text size=60 name='nam' value='$a->name'></td>";
print " <td><input type=text size=3 name='ed' value='кг'></td>";
print " <td><input type='submit'  value=' Создать '></td>";
print "<input type='hidden' name='a' value='11'>";
print " <td></td>";
print " </tr>";
   print "</form>";
}
foreach ($posts as $a) {

print " <tr>";
print " <td align='right'>$a->id</td>";

print " <td> " ; if($actt==2&&$idd==$a->id){
```

```
        print "<input type=text size=60 name='nam' value='$a->name'>";
      }else{  print "$a->name";} print "</td>";
    print " <td > " ; if($actt==2&&$idd==$a->id){  print "<input type=text size=3
name='ed' value='$a->ed'>";}else{  print "$a->ed";} print "</td>";

    if($actt==2&&$idd==$a->id){
      print " <td colspan=2>";
       print " <input type='submit'  value=' Зберегти '>";
       print "<input type='hidden' name='a' value='21'>";
       print "<input type='hidden' name='idd' value='$idd'>";
       print "</form>";
      print "</td>";
    }else{
       print " <td><a href='".$h."&a=2&idd=$a->id'>Редагувати</a></td>";
       print " <td><a href='".$h."&a=3&idd=$a->id'
onclick='javascript:return(confirm(\"Видалити $a->name ?\"));'
>Видалити</a></td>";
    }
    print " </tr>";
    }
    print " </table>";
    }
    }
    [/exec]
```

3.3 Реінжиніринг. Від простого проекту, до проекту рівня підприємства.

Аналіз причин зміни проекту та пошук шляхів.

Протягом тривалого використання постійно зростаючого проекту. Ми прийшли до висновку, що для уникнення зайвих витрат на розробку і адміністрування проектів нам необхідно перейти на правильне архітектурне рішення.

Причиною вплинення на наше рішення стало те, що через зростання масштабу даних у нас з'явилася необхідність поділу окремих видів ділянок робіт по функціональних групах. А також затребуваність проекту в цілому спричинила появу дистрибутивної версії проекту, здатної структуруватися на різні масштаби та типи функціональних груп учасників. Це і можливість роботи усередині підприємства, а також можливість використання незалежними групами підприємців з можливістю зростання проекту до максимально можливого. Для суміщення на момент перехідної стадії вирішено було використання гібридного варіанту, з можливістю використання сумісно СУБД з міцними функціональними можливостями. А саме: СУБД маючи широкий спектр функцій логічного і фізичного рівня, з великим різновидом типів об'єктів, широким спектром мікро розподілу прав (навіть на рівні записів), що працює з великими даними (таблицями і пов'язаними з нею об'єктами) здатної добре працювати з таблицями в яких мільярди рядків. СУБД, що має хорошу організацію вбудованої логіки і програмного забезпечення на рівні бази даних. пакетів з уже готовими бібліотеками рішень (як з області бізнес логіки, так і пакетів з прикладними рішеннями, наприклад статистики, різних видів аналізу, реалізації штучного інтелекту), аналітики роботи з базою даних, методів настройки продуктивності і ін.

Після проведеного тривалого та різно-факторного аналізу та підбору, максимально масштабованих архітектур проектування, наш вибір припав на архітектуру на основі поєднання логіки СУБД (Oracle) з її логікою(функції, процедури, пакети) і об'єктами, інтерфейсом прикладного програмування (API) Django Rest Framework (DFR), для організації REST сервісного API. І для

розробки фронтенд як окремої незалежної задачі, була обрана повноцінна система для швидкої розробки на Vue.js - Vue CLI.

Крім того для зручності налагодження, масштабування, автоматизації деплоя версій, та балансування навантажень, всі вищезгадані пакети(бібліотеки/СУБД) було вирішено запускати у Docker контейнерах, під управлінням Kubernetes. Використання Kubernetes дасть можливість використовувати стратегії деплоя: rolling, recreate, blue/green, canary, dark , як для frontend, так і для backend версій. Як продакшн, так і девелоп версій. Використання регулювання навантаження додатку(LoadBalanser), перевірку читаності та життєздатності liveness та readiness probes.

Також дасть можливість розширення з використанням хмарних рішень Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP), Oracle Cloud Infrastructure, IBM Cloud та інших.

Коротко про кожну технологію та кроках реалізацій у ній наших потреб був створений пілотний проект. Ми опишемо загальний план та принципи.

3.3.1 Вибір СУБД.

Після проведеного тривалого та різнофакторного аналізу та проведення тестів можливостей та навантажень вибір припав на СУБД Oracle 19c. З варіантом контейнерної організації. Це було зроблено на основі тих вимог, які ми вже перечисляли, СУБД маючий широкий спектр функцій логічного та фізичного рівня, з великим різновидом типів об’єктів, широким спектром мікро розподілом прав (навіть на рівні записів), працюючої з великими даними (таблицями та зв’язаними з нею об’єктами здатної гарно працювати з таблицями, у яких мільярди рядків. СУБД, має хорошу організацію вбудованої логіки і програмного забезпечення на рівні бази даних. Пакетів з уже готовими бібліотеками рішень (як з області бізнес логіки, так і пакетів з прикладними рішеннями, наприклад статистики, різних видів аналізу, реалізації штучного

інтелекту), аналітики роботи з базою даних, методів настройки продуктивності та ін

Рішення розробки широкого спектра прикладних функцій (API) у виді Rest веб сервісів, передбачає роботу з вже готовими рішеннями на рівні бази даних, що дає можливість роботи з великим обсягом даних та складної аналітики цих даних. Для проведення таких робіт потрібна гарна організація даних.

Планування нової информаційної системи передбачає створення крім центрального, двох приєднуючих контейнерів, які повинні працювати незалежно один від одного(Pluggable databases). Один контейнер планується під адміністрування бази даних, адміністрування користувачей, з ролями та системними привілегіями. А також бази для організації виробничого циклу, заказів та реалізації.

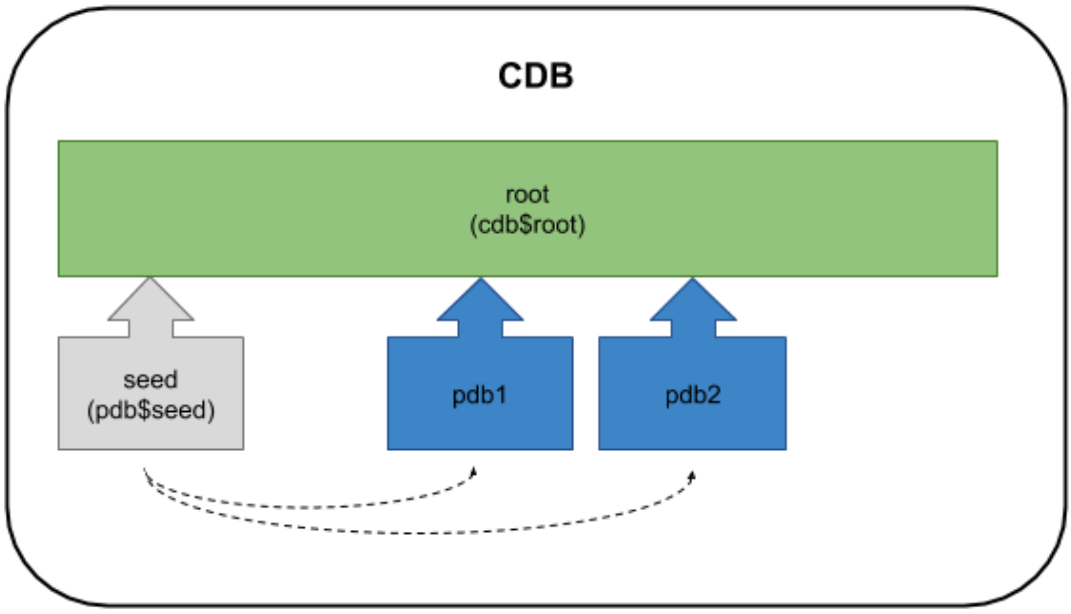


Рисунок 3.11 – Вибір СУБД

CREATE PLUGGABLE DATABASE pdb1 ADMIN USER pdb_adm IDENTIFIED BY Password1 FILE_NAME_CONVERT=('/u01/app/oracle/oradata/cdb1/pdbseed/','/u01/app/oracle/oradata/cdb1/pdb1/');

CREATE PLUGGABLE DATABASE pdb2 ADMIN USER pdb2_adm IDENTIFIED BY Password1 FILE_NAME_CONVERT=('/u01/app/oracle/oradata/cdb1/pdbseed/','/u01/app/oracle/oradata/cdb1/pdb2/');

У адмін контейнері БД диспонуємо (для пілотного проекту) адмін дані Jango Framework, які ми будемо використовувати для створення Rest – webservice.[9]

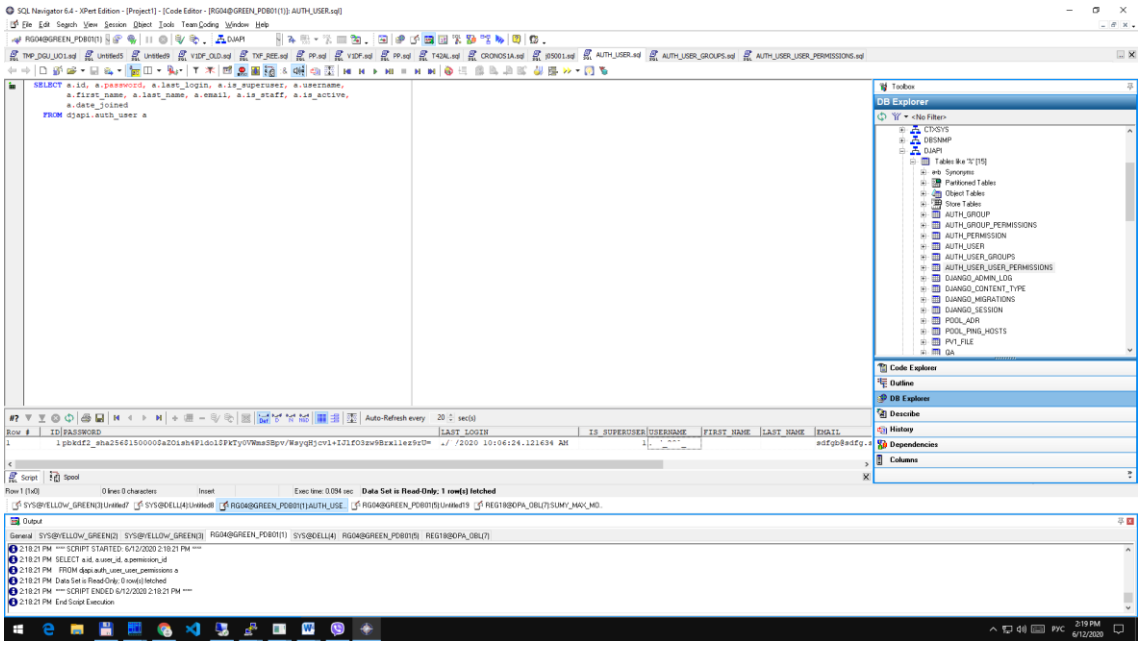


Рисунок 3.12 – Перевірка працездатності Jango Rest Framework

Після розробки груп та їх привілегій готуємо та вносимо у відповідні бази необхідні ролі та привілегії.

У наш другий контейнер за допомогою скрипта конвертера створюємо структури баз даних, та самі дані, нашого проекту та переносимо його у другий контейнер. [9]

Робимо імпорт БД нашого робочого проекту:

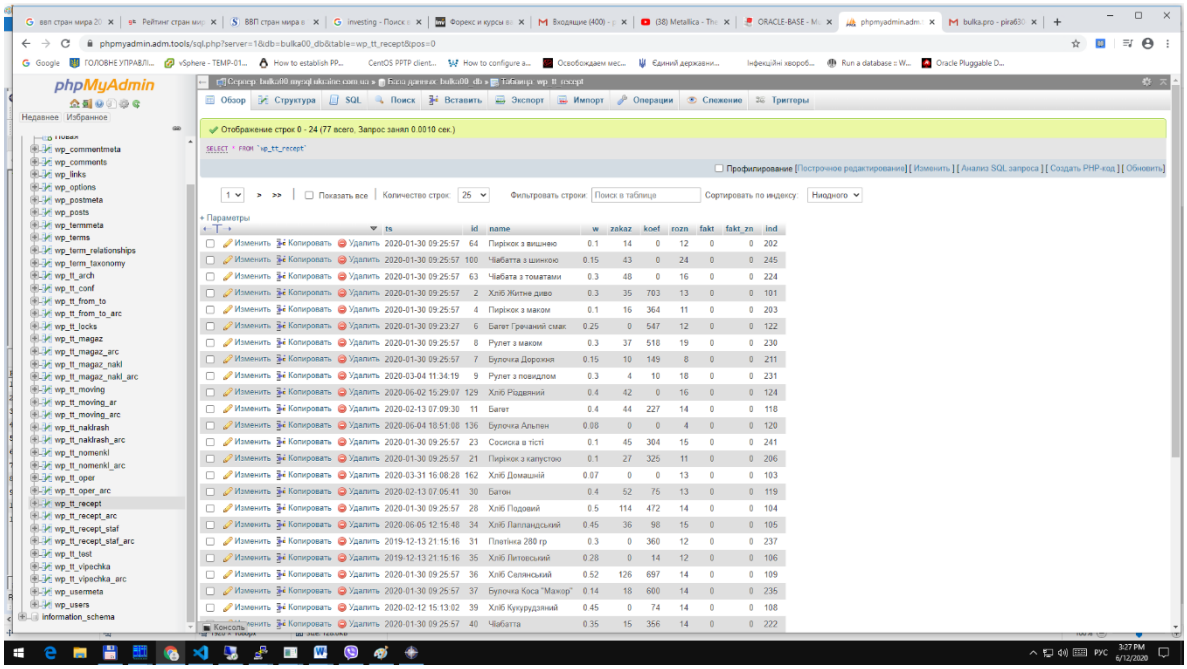


Рисунок 3.13 – Імпорт БД

Готуємо відповідну мінімальну логіку у вигляді: констреінтів, індексів, тригерів, секвенсів, процедур, функцій та пакетів.

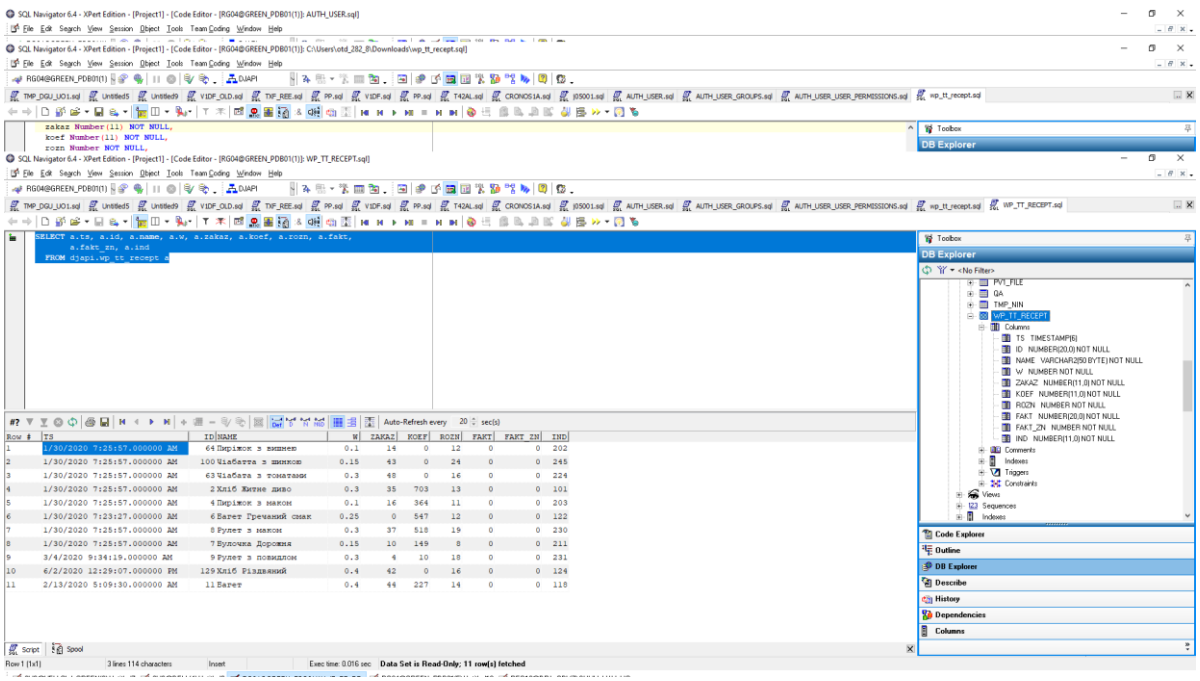


Рисунок 3.14 – Мінімальна логіка

3.3.2 API за допомогою Django REST framework

Після перевірки можливостей СУБД Oracle та підготовки об’єктів і логіки баз даних. Ми віришили створити інтерфейс прикладного програмування відомого як Application Programming Interface (API). API, Application Programming Interface (програмний інтерфейс додатку), - дуже широкі поняття бекенд-розробки. Той API, який ми розглянемо сьогодні, уявляє собою сайт без фронтенд-складової. Замість рендерінга HTML-сторінок, бекенд повертає дані у JSON форматі для використання їх у нативних або веб-додатків. Найпильніша увага при написанні API (як і при написанні веб сайтів) потрібно звернути на те, як він буде використовуватися.

Після аналізу доступних, гарно маштабованих, легких у розробці, підтримуючих розробниками сподобався Django Rest Framework.

Нам знадобиться декілька інструментів: Python 3, Django 2.2, та djangorestframework 3.9 (із репозиторія запустіть pip install -r requirements.txt для установки бібліотек). Також ми завантажили безкоштовну версію Postman. Postman – відмінний інструмент для розробки та тестування API [13].

Для початку ми відкрили папку taskmanager, що містить manage.py, і виконайте python manage.py migrate у командному рядку, щоб застосувати міграції баз даних до дефолтної sqlite бази даних Django. Створюємо суперкористувача за допомогою python manage.py createsuperuser та не забудьте записати ім’я користувача і пароль. Вони знадобляться нам пізніше. Потім виконайте python manage.py runserver для взаємодії з API.

Ми можемо працювати з API двома способами: просматривая фронтенд Django REST фреймворка або виконуючи http-запити. Відкриємо браузер та перейдемо до 127.0.0.1:8000 або до localhost через порт 8000, де Django-проекти запускаються за замовчуванням. Ми побачимо веб-сторінку зі списком доступних кінцевих точок API. Це важливий принцип у RESTful підході до API-розробці: сам API повинен показувати користувачам, що доступно та як це використовувати.

Для початку давайте подивимось на функцію api_index у views.py. Вона містить список кінцевих точок, які ви відвідуєте.

```
@define_usage(returns={'url_usage': 'Dict'})
@api_view(['GET'])
@permission_classes((AllowAny,))
ef api_index(request):
details = {}
for item in list(globals().items()):
if item[0][0:4] == 'api_':
if hasattr(item[1], 'usage'):
details[reverse(item[1].__name__)] = item[1].usage
return Response(details)
```

API функції для кожного уявлення (view в Django) обгорнуті трьома декораторами. Ми ще повернемось до @define_usage.@api_view потрібен для Django REST фреймворка та відповідає за дві речі: шаблон для веб-сторінки, яка у результаті вийде, і HTTP-метод, підтримуючи кінцевою точкою. Щоб дозволити доступ до цього url без перевірки справжності,@permission_classes, також із Django REST фреймворка, заданий як AllowAny. Головна функція API звертається до глобальної області видимості додатка щоб «зібрати» всі певні функції. Так як ми додали до кожної функції уявлення префікс api_, ми можемо легко їх відфільтрувати та повернути словник, що містить інформацію про їх викликів. Деталі викликів надаються призначеним для користувача декоратором, написаним у decorators.py.

```
def define_usage(params=None, returns=None):
def decorator(function):
cls = function.view_class
header = None
# Чи потрібна  аутентифікація для виклика цього уявлення?
if IsAuthenticated in cls.permission_classes:
header = {'Authorization': 'Token String'}
```

```
        # Створюємо лист доступних методів, виключаючи 'OPTIONS'
        methods = [method.upper() for method in cls.http_method_names if method
!= 'options']
        # Створюємо словник для відповіді
        usage = {'Request Types': methods, 'Headers': header, 'Body': params,
'Returns': returns}
        # Захист від побічних ефектів
        @wraps(function)
        def _wrapper(*args, **kwargs):
        return function(*args, **kwargs)
        wrapper.usage = usage
        return _wrapper
        return decorator
```

Щоб верифікувати користувача, метод api_signin запрошує ім’я користувача та пароль і використовує вбудований у Django метод authenticate. Якщо представлені облікові дані вірні, він повертає токен, позволяючи клієнту отримати доступ до захищених кінцевих точок API. Пам’ятаємо про те, що даний токен уявляє ті ж права доступу, що і пароль, тому повинен надійно зберігатися у користувацькому додатку. Тепер ми нарешті можемо працювати з Postman. Відкриємо додаток та використаємо його для відправки post-запиту до /signin/, як показано на скриншоті.

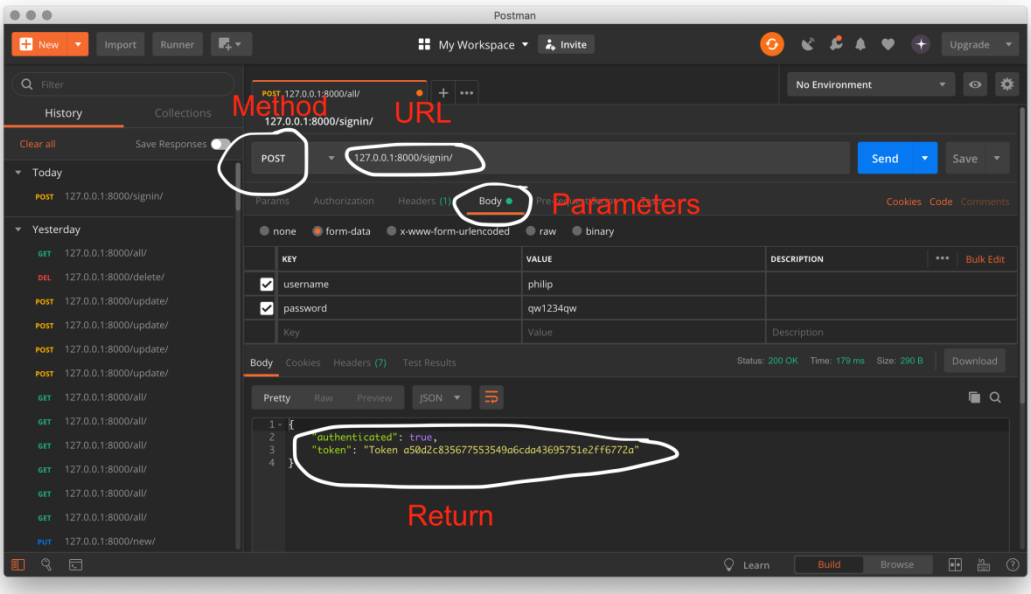


Рисунок 3.15 – Відправка запиту

Тепер нам потрібно створити задачу. Для цього використаємо api_new_task. Зазвичай для створення об’єкту у базі даних використовується PUT-запит. Зверніть увагу, що цей метод, як і два других, не вимагає попередньої серіалізації даних. Замість цього ми передаємо параметри у конструктор об’єкту класа Task, їх ми збережемо у базу даних. Ми відправляємо кількість днів для виконання задачі, так як це набагато простіше, ніж намагатися відправити об’єкт Python-класа Date. Потім у API ми зберігаємо яку-небудь дату у далекому майбутньому. Щоб побачити створений об’єкт, потрібно створити запит до /new/ для створення задачі та повторити запит до /all/.

```
@define_usage(params={'task_id': 'Int', 'description': 'String', 'due_in': 'Int'},
returns={'done': 'Bool'})
@api_view(['POST'])
@authentication_classes((SessionAuthentication, BasicAuthentication,
TokenAuthentication))
@permission_classes((IsAuthenticated,))
def api_update_task(request):
task = request.user.task_set.get(id=int(request.data['task_id']))
```

```
    try:
    task.description = request.data['description']
    except: #Поновлення опису необов’язкове
    pass
    try:
    task.due = date.today() + timedelta(days=int(request.data['due_in']))
    except: #Поновлення дати виконання необов’язкове
    pass
    task.save()
    return Response({'done': True})
```

Для редактування тільки що створеної задачі потрібно створити POST-запит до api_update_task через /update/. Ми включаємо task_id для посилання на правильну задачу з призначеного для користувача task_set. Код буде трішки складніше, так як ми хочемо мати можливість поновлювати опис і/ або дату виконання задачі.

```
    @define_usage(params={'task_id': 'Int'},
    returns={'done': 'Bool'})
    @api_view(['DELETE'])
    @authentication_classes((SessionAuthentication, BasicAuthentication, TokenAuthentication))
    @permission_classes((IsAuthenticated,))
    def api_delete_task(request):
    task = request.user.task_set.get(id=int(request.data['task_id']))
    task.delete()
    return Response({'done': True})
```

Використайте DELETE-запит до api_delete_task через /delete/ для видалення задачі. Цей метод працює аналогічно функції api_update_task, за виключенням того, що замість зміни задачі він видаляє її.

Створив пробні функції для нашого API, св’яжемо фреймворк з СУБД Oracle. Для цього у Database configuration у Django, посилання у Database секції у Django змінимо

```
DATABASES = {

'default': {

'ENGINE': 'django.db.backends.oracle',

'NAME': 'XE',

USER': 'hr',

'PASSWORD': 'hr',

'HOST': 'localhost',

'PORT': '1521'

}

}
```

на дані свого підключення.

Замінимо шляхи на актуальні

```
set PATH=%PATH%;C:\oracle\app\oracle\product\19.3.0\server\bin
set PATH=%PATH%;C:\python\instantclient
```

і проінсталюємо у пітоні драйвер для зв’язку з СУБД Oracle.

pip install cx_Oracle

Після випробування зв’язку перевіримо і додамо функції розроблені у логіці БД. Для цього розробимо на основі зворотнього реінжиніринга моделі та серіалізатори. Додамо необхідні посилання для URL path. Тестуємо API за допомогою Postman, розробивши попередньо тести.

Використовуючи Django REST framework ми отримуємо уніфіцированний та незалежний backend з можливістю раздільної розробки, тестування та використовування. Отримуємо готове адміністрування та готові користувацькі політики. [14]

У подальшому ми розробимо свої політики безпеки і доступів до API.

Стандартний вид рест-сервісів буде таким [14]:

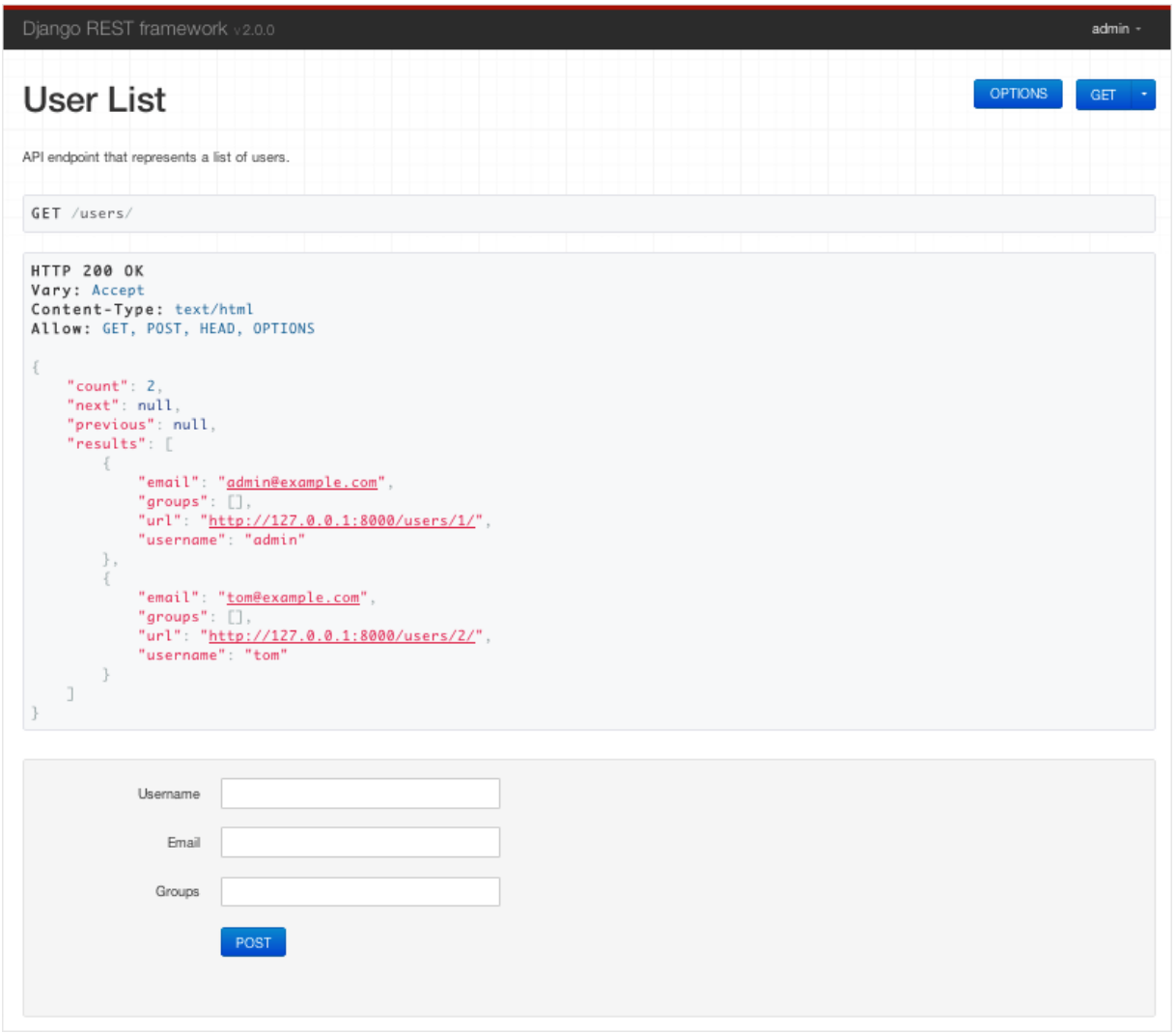


Рисунок 3.16 – Рест-сервіси

Схема роботи фреймворка:

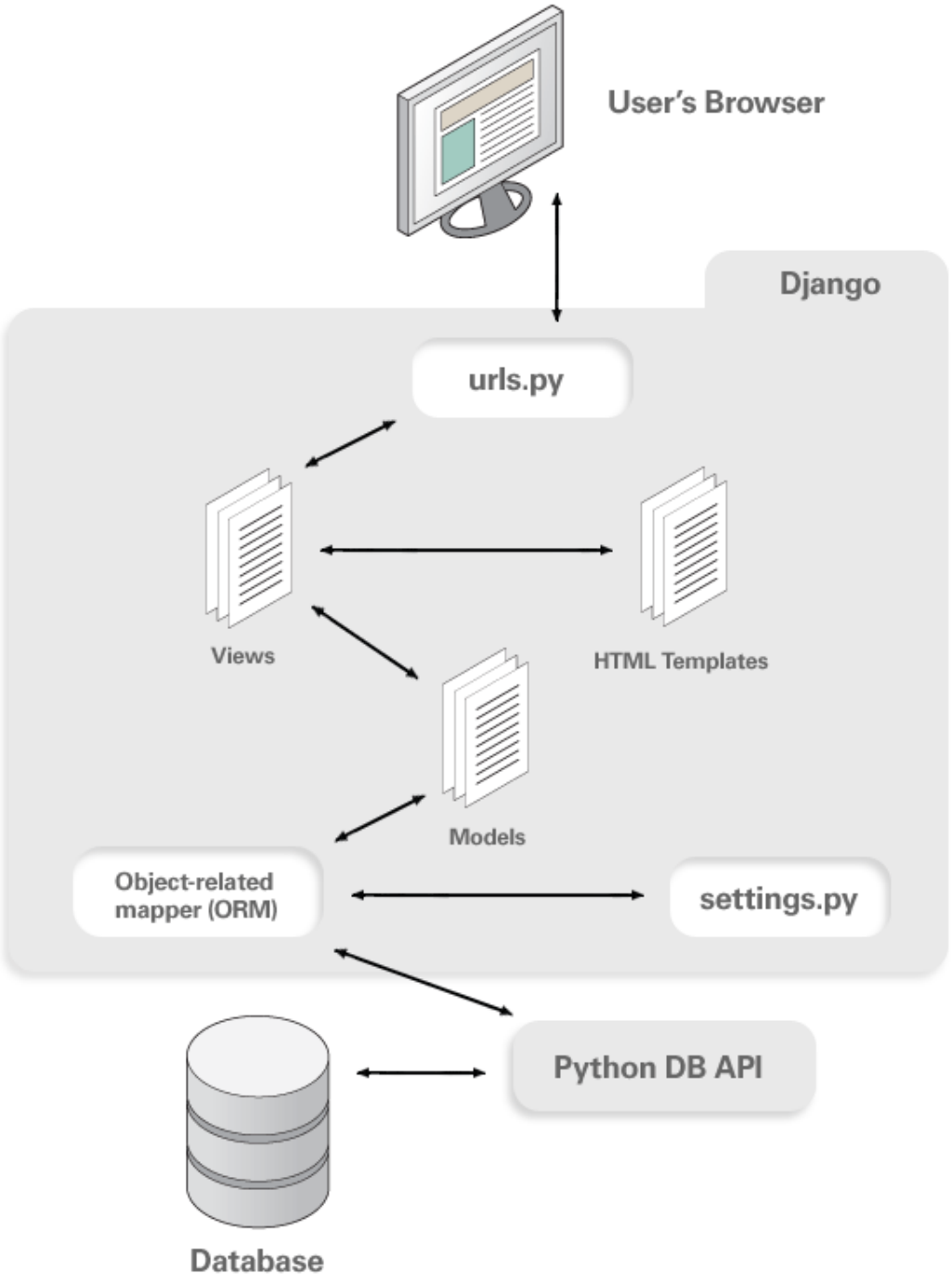


Рисунок 3.17 – Схема роботи фреймворка

Підготувавши API для деяких наших задач та серіалізатори для наших даних. Ми проводимо іспит за допомогою Postman, по аналогії вказаної вище, для основних наших задач.

3.3.3 Фронтенд розробка.

Для фронтенд розробки після тривалих тестів нами був обран Vue CLI. Основними причинами, які переконали нас у правоті нашого вибору стали наступні функції:

- **Архітектура на основі плагінів**. Vue CLI повністю працює на плагінах, що робе його дуже гнучким та розширюваним. Ми можемо вибрати, які з вбудованих плагінів потрібні під час процесу створення нового проекту. Но ми не обмежені тільки цим, ми можемо додати будь-яку кількість плагінів у будь-який момент після створення проекту.
- Vue CLI повністю налаштований**,** расширений **і** поновлюючий інструмент.
- **Набір офіційних встановлених плагінів,** який об'єднує професійні інструменти екосистеми фронтенда (Babel, ESLint, TypeScript, PWA, Jest, Mocha, Cypress та Nightwatch).
- **Один за замовченням пресет**, який ми можемо змінити у відповідно з нашими потребами під час створення проекту або після цього.
- **Не потрібно використовувати витяг залежностей (eject)**. На відміну від CLI-інструментів React і Angular ми можемо безпечно перевірити та налаштувати конфігурацію webpack нашого проекту у будь-який час після створення без необхідності витяг залежностей додатку та переключення на ручній спосіб управління його конфігурації.
- **Підтримка мультісторінок.**
- **Миттєве створення прототипів** без необхідності у будь-якій конфігурації.
- **Різні версії проекту** дозволяють нам створити різні версії проекту, ми можемо использовать таку ж кодову базу для того, щоб використовувати її як додаток, бібліотеку або веб-компоненти.
- **Режим використання сучасних можливостей**. Це означає, що ми можемо збирати наш додаток для сучасних браузерів, но з автоматичною підтримкою для старих.

- **Повномасштабний графічний інтерфейс** для створення, поновлення та управління складними проектами без складностей.
- **API для користувацького інтерфейса плагінів**. Vue UI представляє API для плагінів, який ми можемо використовувати для додавання власних функціональних можливостей до GUI-версії командного рядка.
- **Багато корисних плагінів від суспільства.**

Для написання пілотного проекту на Vue CLI ми встановили Node.js. Потім з командного рядка

npm install -g @vue/cli

Потім ми встановимо глобальний пакет Vue CLI Service:

npm install -g @vue/cli-service-global

Далі все готово для створення нашого проекту:

vue create vuecli-project

Відповівши при інсталяції на всі питання зв’язані з установкою сумісності (Babel), контроля синтаксиса (Linter) і CSS препроцесора, ми отримали структуру проекту[15]:

В директорії проекту выще показано, що у нас є наступні файли і директорії:

- Директорія **node_modules** містить пакети, які потребують додаток та інструменти розробки.
- Директорія **public** містить статичні ресурси проекту, які не будуть включені при створенні збірки проекту.
- В директорії **src** міститься додаток Vue.js з усіма ресурсами.
- Файл **.gitignore** містить список файлів і папок, які не враховуються системою управління версіями Git.
- Файл **babel.config.js** містить параметри конфігурації компілятора Babel.
- Файл **package.json** містить список пакетів, необхідних для разробки на Vue.js, а також команди, використувані для інструментів розробки.

- Файл **package-lock.json** містить повний список пакетів, необхідних у рамках проекту та їх залежностей.
- Файл **README.md** містить загальну інформацію про проект.

А в директорії src, є наступні файли і директорії:

- Директорія **assets** використовується для статичних ресурсів, необхідних додатку та які будуть включені у процес збірки.
- Директорія **components** використовується для компонентів додатка.
- Директорія **views** використовується для компонентів, які будуть відображатися за допомогою можливості маршрутизації URL-адрес.
- Файл **App.vue** — корневий компонент.
- Файл **main.js** — це файл JavaScript, який створює об’єкт екземпляра Vue.
- Файл **router.js** використовується для налаштування маршрутизатора Vue.
- Файл **store.js** використовується для налаштування сховища даних, створеного за допомогою Vuex.

Запуск, збірка та перевірка проекту можлива за допомогою команд serve, build, inspect .

Стартувавши проект npm run serve спостерігаємо вдачний старт:

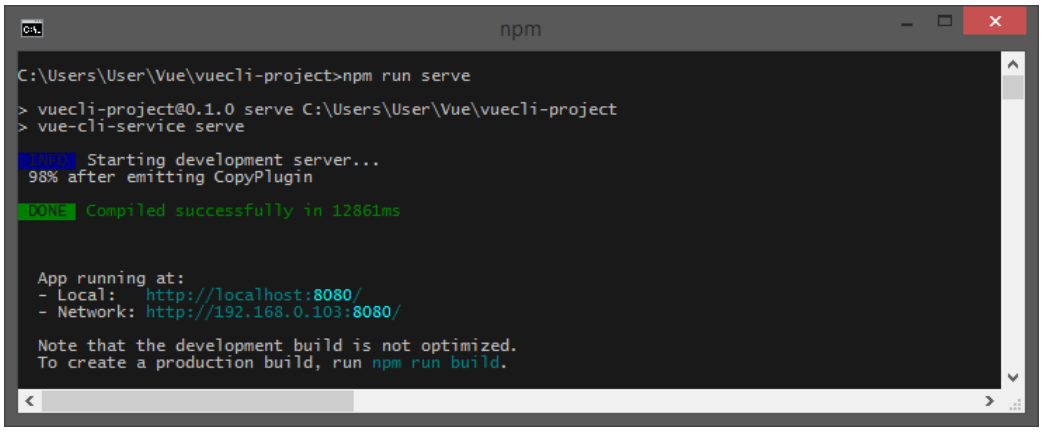


Рисунок 3.18 – Старт проекту

На порту 8080 локального хосту бачимо:

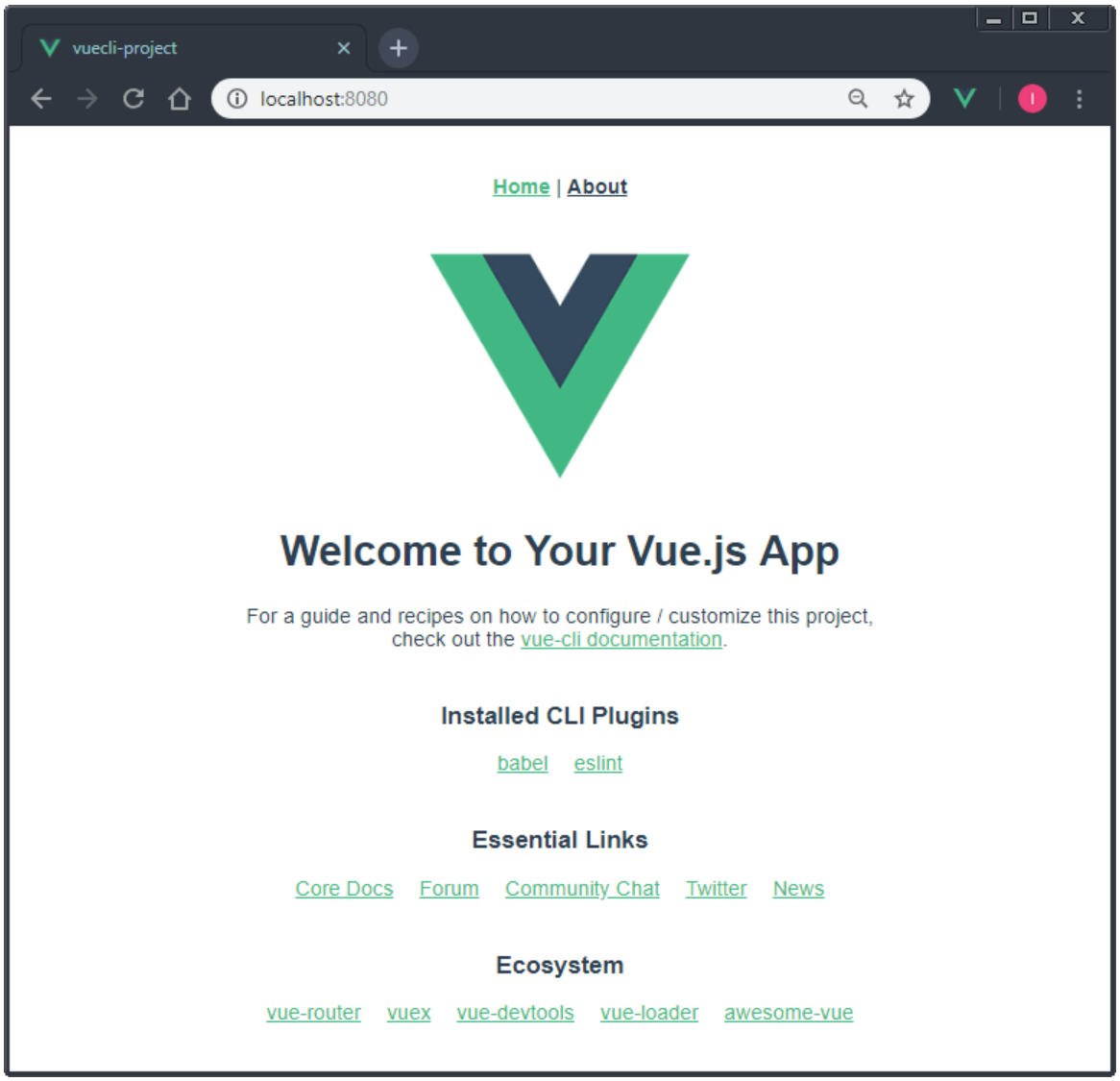


Рисунок 3.19 – Сторінка вітання vue.js

Створюємо компоненти меню та тематичні сторінки.[16]

У файлі App.vue реалізуємо роутинг і меню.

```
import Vue from 'vue'
import Router from 'vue-router'
import PageIPList from '/components/PageReceptList'
import PageParse from '/components/PageRecept'
Vue.use(Router)
export default new Router({
routes: [
{
path: '/receprlist',
name: 'Список рецептов',
component: PageReceptList'
},
{
path: '/recept',
name: 'Рецепт',
component: PageRecept
}
]
})
<template>
<div id="app">
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
<a class="navbar-brand" href="#"></a>
<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarColor01" aria-controls="navbarColor01" aria-expanded="false" aria-label="Toggle navigation">
<span class="navbar-toggler-icon"></span>
</button>
<div class="collapse navbar-collapse" id="navbarColor01">
```

```
<ul class="navbar-nav mr-auto">
<li class="nav-item active">
<a class="nav-link" href="#"> <span class="sr-only">(current)</span></a>
</li>
<li class="nav-item">
<router-link class="nav-link" to="/receptlist">Список рецептів</router-link>
</li>
<li class="nav-item">
<router-link class="nav-link" to="/recept">Редактування рецептів</router-link>
</li>
</ul>
<form class="form-inline my-2 my-lg-0">
<input class="form-control mr-sm-2" type="text" placeholder="Search">
<button class="btn btn-secondary my-2 my-sm-0" type="submit">Search</button>
</form>
</div>
</nav>
<router-view/>
</div>
</template>
<script>
import Vue from 'vue'
import { BootstrapVue, IconsPlugin } from 'bootstrap-vue'
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'
// Install BootstrapVue
Vue.use(BootstrapVue)
```

```
// Optionally install the BootstrapVue icon components plugin
Vue.use(IconsPlugin)
export default {
name: 'app'
}
</script>
<style>
@import url(https://fonts.googleapis.com/css?family=Eczar);
@import url(https://fonts.googleapis.com/css?family=Work+Sans);
body {
font-family: "Work Sans", "Segoe UI", "Helvetica Neue", sans-serif;
}
h1, h2, h3, h4, h5, h6 {
font-family: "Eczar", sans-serif;
}
</style>
```

Сторінка списку рецептів:

```
<template lang="pug">
#app
.card(v-for="recpt  in recepts.id")
.card-header
div id="obj-{{ recept.id }}" class="obj-{{ recept.id }}"
button.btn.btn-clear.float-right(@click="deleteNote(recept)")
button.btn.btn-info.float-right(@click='toggle = !toggle')
.card-title {{ recept.name+' : '+ recept.w+'-'+ recept.rozn }}
.card-body(v-show='toggle' )
div(v-if='toggle' )
host-list(v-bind:vo= recept.name')
</template>
<script>
```

```
import { mapGetters } from 'vuex'
import LList from './LList'
export default {
name: recept -list',
data () {
return {
toggle: false,
idp: 0
}
},
components: {
recept -list': recept
},
computed: mapGetters([' recept ']),
methods: {
deleteNote (recept) {
// Викликаємо `deleteRecept` з нашого сховища, яке
// намагатиметься видалити рецепт з нашої бази даних, відправивши запит до API
this.$store.dispatch('deleteRecept', recrpt)
}
},
beforeMount () {
// Перед тим як завантажити сторінку, нам потрібно отримати список всіх
// х рецептів. наявні для цього ми викликаємо дію `getRecepts` з
// нашого сховища
this.$store.dispatch('getNoteRecepts')
}
}
```

```
</script>
<style>
header {
margin-top: 50px;
}
</style>
```

Сторінка створення рецепту:

```
<template Рецепт:
.col-9
input.form-input(type="text" v-model="name" placeholder="Назва рецепту")
.form-group
.col-1
label.form-label Вес
.col-1
input.form-input(type="text" v-model="w" placeholder="1")
.col
Цена реалізації
.col-1
input.form-input(type="text" v-model="rozn" placeholder="10")
.form-group
.col-1
.col-3
button.btn.btn-primary(type="submit") Створити
</template>
<script>
export default {
name: 'create-note',
data () {
return {
```

```
'name': '',
'w': '',
'rozn': ''
}
},
methods: {
submitForm (event) {
this.createRecept()
this.name = ''
this.w = ''
this.rozn = ''
// preventDefault потрібно для того, щоб сторінка
// не перезавантажувалась після натискання кнопки submit
event.preventDefault()
},
createNote () {
// Викликаємо дію `createRecept` з сховища, яке
// відправить запит на створення нової замітки до нашого API.
this.$store.dispatch('createRecept', { name: this.name, w: this.w, rozn: this.rozn })
}
}
}
</script>
```

Як ви замітили ми використали препроцесор pug. Аналогічно ми створили всі наші сторінки з працюючого проекту. Згідно використаної технології Vuex ми створюємо особистий стор у якому визначаємо всі використані нами стани, гетери, дії та мутації.

Зв’язок з нашим API будемо вести через бібліотеку Axios.

```
import axios from 'axios'
```

```
export const HTTP = axios.create({
baseURL: 'http://localhost:8000/'
})
```

З його допомогою визначаємо методи звернення до нашого API розписаному в Urls.py Jango:

```
import { HTTP } from './common'
export const Recept = {
create (config) {
return HTTP.post('/reseprt/', config).then(response => {
return response.data
})
},
delete (recept) {
return HTTP.delete('/recept/?id=' + recept.id).then(response => {
return response.data
})
},
list () {
return HTTP.get('/recept/').then(response => {
return response.data
})
}
}
```

Створив основні модулі нашого фронтенда, ми закінчили цикл створення повного стека нашої нової архітектури.

Для хорошої конфігурації середовища, швидкого деплоя, ми вирішили використовувати Docker контейнери для всіх вище описаних верств нашого проекту. А для автоматичного використання потрібної кількості копій додатків, балансування навантаження, автоматичної зміни версій ми будемо використовувати Kubernetes.[10]

У даній роботі ми розповімо про роботу в контейнері нашої бази даних Oracle-19c. Але ми розробляємо наш проект з використанням балансувальника навантажень, фронтенда, і використання Jango.

Використання СУБД Oracle в контейнері, можливо кількома способами: один з яких скачати вже готовий імідж з Docker сховища встановити його і налаштувати. Другий спосіб встановивши Oracle на потрібну платформу (у нас це Oracle Linux) і зробити білд іміджу. Третій спосіб яким ми скористаємося це на Git-репозиторії скористатися скриптами для створення білда в контейнері, що подаються виробником (https://github.com/oracle/docker-images/tree/master/OracleDatabase/SingleInstance/dockerfiles/19.3.0).

```
# LICENSE UPL 1.0
#
# Copyright (c) 2018, 2020 Oracle and/or its affiliates.
#
# ORACLE DOCKERFILES PROJECT
# --------------------------
# This is the Dockerfile for Oracle Database 19c
#
# REQUIRED FILES TO BUILD THIS IMAGE
# ----------------------------------
# (1) db_home.zip
#Download Oracle Database 19c Enterprise Edition or Standard Edition 2 for Linux x64
```

```
# from http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html
#
# HOW TO BUILD THIS IMAGE
# -----------------------
# Put all downloaded files in the same directory as this Dockerfile
# Run:
#  docker build -t oracle/database:19.3.0-${EDITION} .
#
# Pull base image
# ---------------
FROM oraclelinux:7-slim as base
# Labels
# ------
LABEL "provider"="Oracle"                                       \
"issues"="https://github.com/oracle/docker-images/issues"      \
"volume.data"="/opt/oracle/oradata"                             \
"volume.setup.location1"="/opt/oracle/scripts/setup"            \
"volume.setup.location2"="/docker-entrypoint-initdb.d/setup"    \
"volume.startup.location1"="/opt/oracle/scripts/startup"        \
"volume.startup.location2"="/docker-entrypoint-initdb.d/startup" \
"port.listener"="1521"                                          \
"port.oemexpress"="5500"
# Environment variables required for this build (do NOT change)
# -------------------------------------------------------------
ENV ORACLE_BASE=/opt/oracle \
ORACLE_HOME=/opt/oracle/product/19c/dbhome_1 \
INSTALL_DIR=/opt/install \
INSTALL_FILE_1="LINUX.X64_193000_db_home.zip" \
INSTALL_RSP="db_inst.rsp" \
```

```
        CONFIG_RSP="dbca.rsp.tmpl" \
        PWD_FILE="setPassword.sh" \
        RUN_FILE="runOracle.sh" \
        START_FILE="startDB.sh" \
        CREATE_DB_FILE="createDB.sh" \
        SETUP_LINUX_FILE="setupLinuxEnv.sh" \
        CHECK_SPACE_FILE="checkSpace.sh" \
        CHECK_DB_FILE="checkDBStatus.sh" \
        USER_SCRIPTS_FILE="runUserScripts.sh" \
        INSTALL_DB_BINARIES_FILE="installDBBinaries.sh"
        # Use second ENV so that variable get substituted
        ENV
PATH=$ORACLE_HOME/bin:$ORACLE_HOME/OPatch/:/usr/sbin:$PATH \
        LD_LIBRARY_PATH=$ORACLE_HOME/lib:/usr/lib \
        CLASSPATH=$ORACLE_HOME/jlib:$ORACLE_HOME/rdbms/jlib
        # Copy files needed during both installation and runtime
        # ------------
        COPY $SETUP_LINUX_FILE $CHECK_SPACE_FILE $INSTALL_DIR/
        COPY $RUN_FILE $START_FILE $CREATE_DB_FILE $CONFIG_RSP
$PWD_FILE $CHECK_DB_FILE $USER_SCRIPTS_FILE $ORACLE_BASE/
        RUN chmod ug+x $INSTALL_DIR/*.sh && \
        sync && \
        $INSTALL_DIR/$CHECK_SPACE_FILE && \
        $INSTALL_DIR/$SETUP_LINUX_FILE && \
        rm -rf $INSTALL_DIR
        ############################################
        # ------------------------------------------
        # Start new stage for installing the database
        # ------------------------------------------
        ############################################
```

```
FROM base AS builder
ARG DB_EDITION
# Copy DB install file
COPY --chown=oracle:dba $INSTALL_FILE_1 $INSTALL_RSP $INSTALL_DB_BINARIES_FILE $INSTALL_DIR/
# Install DB software binaries
USER oracle
RUN chmod ug+x $INSTALL_DIR/*.sh && \
sync && \
$INSTALL_DIR/$INSTALL_DB_BINARIES_FILE $DB_EDITION
################################################
# ------------------------------------------
# Start new layer for database runtime
# ------------------------------------------
################################################
FROM base
USER oracle
COPY --chown=oracle:dba --from=builder $ORACLE_BASE $ORACLE_BASE
USER root
RUN $ORACLE_BASE/oraInventory/orainstRoot.sh && \
$ORACLE_HOME/root.sh
USER oracle
WORKDIR /home/oracle
HEALTHCHECK --interval=1m --start-period=5m \
CMD "$ORACLE_BASE/$CHECK_DB_FILE" >/dev/null || exit 1
# Define default command to start Oracle Database.
CMD exec $ORACLE_BASE/$RUN_FILE
```

Викачуємо дистрибутив LINUX.X64_193000_db_home.zip і назва бази і паролів запускаємо docker build -t oracle / database: 19.3.0.0 який створює імідж.

Запустити його можна створивши контейнер:

```
docker run --rm --name oracle-docker \-p 1521:1521 \
-p 5500:5500 \
-v /opt/oracle/oradata :/opt/oracle/oradata
--shm-size="10g" \
oracle/database:19.3.0.0
```

Як ми бачимо в контейнер проброшени два прота: 1521 для зв'язку з бази даних з іншими додатками і 5500 - для роботи Oracle Enterprise Manager.

І папки для файлів зберігання даних.

Запускаємо контейнер і в базі даних створюємо всі що необхідно для роботи проекту, що ми описали вище.

Коли контейнер готовий зберігаємо його і готуємо yaml [11] файл для деплоя і створення сервісу бази даних:

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
name: oracle-db-enterprise
spec:
replicas: 1
minReadySeconds: 30
selector:
matchLabels:
app: oracle-db-enterprise
template:
metadata:
labels:
app: oracle-db-enterprise
spec:
hostname: oracle-db-enterprise
containers:
```

```
- name: oracle-db-enterprise
image: oracle/database:19.3.0.0
env:
- name: ORACLE_SID
value: "ORCLCDB"
- name: ORACLE_PDB
value: "ORCLPDB1"
- name: ORACLE_PWD
value: “******”
volumeMounts:
- name: oracle-db-config
mountPath: /opt/oracle/scripts/setup
ports:
- containerPort: 1521
livenessProbe:
tcpSocket:
port: 1521
initialDelaySeconds: 300
periodSeconds: 30
volumes:
- name: oracle-db-config
configMap:
name: oracle-db-config
---
apiVersion: v1
kind: Service
metadata:
name: oracle-db-enterprise-1
spec:
ports:
```

- port: 1521

targetPort: 1521

protocol: TCP

selector:

app: oracle-db-enterprise-1

За аналогією створюємо контейнери для інших верств Vuex, Jango і також використовуємо сервіс Ingress входить в Kubernetes як лоадбалансер для наших фронтенд[12]:

kind: Service

apiVersion: v1

metadata:

name: service-test

spec:

type: LoadBalancer

selector:

app: service_test_pod

ports:

- port: 8080

targetPort: http

Також знаючи широкий спектр API Ingress можливо при робочого навантаження ми будемо використовувати з кількома бекенд-сервісами.

На цьому наші дослідження в плані переробки нашого проекту на нову архітектуру закінчено. Далі ми створюємо графік розробки, міграції проведення тестів. Після узгодження з усіма зацікавленими сторонами починаємо розробку.

ВИСНОВКИ

У роботі був розглянутий дійсно працюючий проект виробничої та комерційної ланки малого бізнесу. Був розглянутий проект, який має різні етапи виробничого процесу та взаємозв'язків різних типів учасників проекту.

В проекті було виконано:

- пошук шляхів сучасних рішень для відповідного масштабу проектів. Та приведені обґрунтування нашого вибору – використання CMS (Content Management System), саме Wordpress.
- На комерційному українському хостингу встановлений Wordpress налаштовані обрана тема для проекту, створені відповідні налаштування для проекту та основне меню проекту.
- Створена на тому же хостінгу база даних (MySQL) та розроблені об'єктів баз даних: таблиці, індекси, констреинти.
- Розроблені скрипти (PHP, JQuery), сторінок та вихідних форм, форм статистики, документація до проекту.
- Була встановлена на система керування базами даних Oracle-19c та була встановлена екземпляр центральної бази даних для контейнерної бази даних, а також встановленні дві контейнерні бази даних.
- За допомогою Django Restframework проведено створення об'єктів моделей бази даних, створені сеарілізатори об'єктів та маршрутизатори для кожного API.
- Створений один з режимів, для нашого фронтенду на Vue CLi.
- Створений за допомогою Docker контейнер для СКБД Oracle.

Після проведення вищезгаданих робіт був проведений аналіз діяльності та зроблені відповідні висновки, які задовольнили замовників та дали можливість приступити до більш масштабної реалізації в даному напрямку.

Список літератури

1. Бардзелл Джеффри Macromedia Dreamweaver MX 2015 с ASP, ColdFusion и PHP. Из первых рук (+ CD-ROM); Эком - М., 2016. - 560 с.

2. Бенкен Елена PHP, MySQL, XML. Программирование для Интернета; БХВ-Петербург - М., 2017. - 336 с.

3. Гизберт Дамашке PHP и MySQL; НТ Пресс - М., 2016. - 569 с.

4. Дронов В. PHP, MySQL и Dreamweaver. Разработка интерактивных Web-сайтов; БХВ-Петербург - М., 2016. - 480 с.

5. Дунаев В.В. HTML, скрипты и стили; БХВ-Петербург - М., 2017. - 527 с.

6.https://pidru4niki.com/15350818/informatika/osnovi_elektronnoyi_komertsiyi

7. https://mozok.net/systema-upravlinya-contentom

8. https://lemarbet.com/ua/otkrytie-internet-magazina/vybiraem-dvizhok-dlya-internet-magazina-kakaya-cms-luchshe/

9. Oracle для профессионалов: архитектура, методики программирования и основные особенности версий 9i, 10g, 11g и 12c. 3-е издание - Томас Кайт.

10. Философия DevOps. Искусство управления IT Дженнифер Дэвис. 2016 O'Reilly.

11. «Continuous delivery. Практика непрерывных апдейтов» — Эберхард Вольф.

12. «Site Reliability Engineering. Надежность и безотказность как в Google» — Бетси Бейер, Крис Джоунс, Дженнифер Петофф.

13. Джефф Форсье. Django. Разработка веб-приложений на Python: учебник / Джефф Форсье, Пол Биссекс, Уэсли Дж. Чан. Пер. с англ. - М: Символ-Плюс, 2013 – 456 с.

14. Django REST Framework [Электронный ресурс]. Режим доступа: http://www.django-rest-framework.org/ (дата обращения: 3.06.2018) Callum Macrae. Vue.js: Up and Running. — O'Reilly, 2017. — 219 с. — ISBN 9781491997246.

15. Olga Filipova. Learning Vue.js 2 Learn how to build amazing and complex reactive web applications easily with Vue.js. — Packt Publishing Ltd, 2016. — 334 с. — ISBN 9781786461131.

16. Alex Kyriakidis, Kostas Maniatis. The Majesty of Vue.js. — Packt Publishing Ltd, 2016. — 230 с. — ISBN 9781787125209.