НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення розподілених систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Програмно – апаратний комплекс

моніторингу та управління

активностями охоронної системи»

Виконав (-ла):
студент (-ка) IV курсу, групи ТВ-61
Лєбєдєв Валерій Іванович _____

Керівник:
доцент, кандидат технічних наук
Ковальчук Артем Михайлович _____

Рецензент:
доцент, кандидат технічних наук
Гагарін Олександр Олександрович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського"

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
"___"_____2020р.

ЗАВДАННЯ

на дипломну роботу студенту
Лєбєдєв Валерій Іванович
(прізвище, ім'я, по батькові)

1. Тема роботи «Програмно – апаратний комплекс моніторингу та управління активностями охоронної системи»
керівник роботи Ковальчук Артем Михайлович доцент, к.т.н.
(прізвище, ім'я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від "25" травня 2020р. № 1168-с

2. Строк подання студентом роботи «10» червня 2020р.
3. Вихідні дані до роботи мови програмування — Python, Javascript, середовище розробки — Visual Studio Code, Arduino IDE
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) розглянути та проаналізувати існуючі програмно-апаратні рішення предметної області, розробити апаратне програмне забезпечення, розробити інтерфейс користувача, запроектувати схему бази даних, розробити серверний додаток, реалізувати зв'язок серверного додатку з клієнтським та апаратним додатком, реалізувати можливість авторизації адміністраторів за допомогою клієнтського додатку, забезпечити адміністраторів системи можливістю редагування таблиць баз даних засобами клієнтського додатку, зробити висновки за результатами роботи.
5. Перелік ілюстративного матеріалу 1. Мета роботи. 2. Актуальність теми роботи. 3. Постановка задачі перед проектом. 4. Аналоги систем. 5.Клієнт-

серверна архітектура. 6. Апаратна реалізація. 7. Серверний додаток. 8. Графічний інтерфейс - клієнт. 9. Розгортання додатку. 10. Результат роботи системи головна сторінка 11. Результат роботи системи керування робітниками. 12. Результат роботи системи відстеження руху. 13. Висновки. 14. Дякую за увагу.

7. Дата видачі завдання "1" грудня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	14.12.2019 — 23.12.2019	
2.	Вивчення та аналіз задачі	27.03.2020 – 03.04.2020	
3.	Розробка архітектури та загальної структури системи	27.03.2020 – 03.04.2020	
4.	Розробка структур окремих підсистем	09.04.2020 – 16.04.2020	
5.	Програмна реалізація системи	17.04.2020 – 12.05.2020	
6.	Оформлення пояснювальної записки	03.05.2020 – 06.06.2020	
7.	Захист програмного продукту	25.05.2020	
8.	Передзахист	10.06.2020	
9.	Захист	15.06.2020	

Студент _____ Лєбєдєв В.І.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Ковальчук А.М.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою даної роботи є дослідження схожих за функціями систем що реалізують ідею теми дипломної роботи а також практичне оволодіння прийомами та навичками у створенні програмно-апаратних рішень з використанням сучасних технологій розробки. Для створення клієнтського додатку, його графічного інтерфейсу використано засоби фреймворку Vue.js та мову програмування Javascript. Для створення серверного додатку використано фреймворк Flask на базі мови програмування Python.

Записка містить 68 сторінок, 22 рисунків, 2 таблиці, 3 додатки і 11 посилань.

Ключові слова: REST, API, PYTHON, ФРЕЙМВОРК, БІЗНЕС-ЮНІТ

ABSTRACT

The aim of the work is research similar in function systems that implement the idea of the work and the practical mastery of techniques and skills in creating software and hardware solutions using modern development technologies. Vue.js framework tools and Javascript programming language were used to create the client application and its graphical interface. The Flask framework based on the Python programming language was used to create the server application.

The note contains 68 pages, 22 pictures, 2 tables, 3 attachments and 11 links.

Keywords: REST, API, PYTHON, FRAMEWORK, BUSINESS-UNIT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

REST (Representational State Transfer) — архітектурний стиль взаємодії компонентів додатку в мережі.

API (Application Programming Interface) — набір класів, процедур, функцій завдяки яким одна програма може взаємодіяти з іншою.

RFID (Radio Frequency Identification) — метод автоматичної ідентифікації за допомогою радіочастотного сигналу.

RESTful — сервіс побудований на REST архітектурі.

ВСТУП

На сьогоднішній час більшість компаній, підприємств більш ретельно підходять до вирішення проблеми моніторингу та автоматизації процесів навколо їх діяльності. Даний підхід дозволяє відстежувати, контролювати та приймати часом більш вигідні рішення у тій чи іншій справі.

Досліджуючи вже наявні рішення моніторингу саме охоронних систем можна зробити висновки що існують готові програмно-апаратні комплекси які вирішують проблеми по охороні різного виду об'єктів, приміщень, але у цьому випадку не враховують людей а також людський фактор який присутній у цій справі. Саме тому була розглянута проблема моніторингу активності охоронної системи яка матиме реалізацію за допомогою програмно-апаратного комплексу.

Серед різноманіття як клієнтських так і серверних додатків не має таких які не були б побудовані за використанням різного виду фреймворків. Вони не є «панацеєю» для всіх рішень, але їх використання поліпшує розробку, а часом навіть зменшує витрати на реалізацію проектів. Коли мова йде про розробку рішень що матимуть змогу виконувати роботу по обробці даних, аналізу, зберігання їх в базу даних то відразу виникає потреба у побудові серверної частини. Але для взаємодії з сервером необхідно також розуміти що розробка клієнтської частини також має сенс. Тому для вирішення даної проблеми необхідне рішення що містить у собі архітектуру клієнт-сервер. При цьому важливим аспектом при розробці клієнтської частини є реалізація необхідного для повного функціювання системи функцій користувача а також чіткого зрозумілого інтерфейсу з можливістю надання необхідної інформації про стан надісланих запитів та інформування інших частин системи про їх успішно або ж неуспішну обробку.

Мета виконання даної дипломної роботи полягає у розробці програмно-апаратного комплексу, а саме інформаційної, автоматизованої системи по контролю

та моніторингу активностей охоронної системи з використанням сучасних технологій розробки клієнт-серверних додатків, їх інсталяція на сервері.

При розробці програмно-апаратних додатків використано середовище розробки Visual Studio Code, програмне забезпечення для автоматичного розгортання та керування додатками Docker.

Для розробки програмного додатку було обрано клієнт-сервер архітектуру яка дозволяє проектувати та розроблювати ефективні додатки за допомогою різноманітних інструментів, технологій без прив’язки до конкретної мови програмування.

Апаратний додаток являє собою запрограмований мікроконтролер, що базується на основі модулю Esp8266. Плата використовуватиметься для управління ідентифікацією користувачів за допомогою методу радіочастотної ідентифікації. Даний вибір дозволить з легкістю зв’язати апаратний додаток з програмним.

1.ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

Метою розробки є реалізація клієнт-серверної архітектури для програмно-апаратного комплексу який надаватиме користувачам можливість виконувати роботу по моніторингу та управлінню активностями охоронної системи. Для досягнення цієї мети перед студентом постають наступні питання:

1. Проаналізувати існуючі програмно-апаратні рішення даної предметної області.
2. Розробити інтуїтивно зрозумілий інтерфейс користувача-адміністратора системи, клієнтський додаток.
3. Запроектувати схему бази даних серверного додатку.
4. Розробити серверний додаток.
5. Налагодити зв’язок додатків між собою.
6. Реалізувати можливість авторизації користувачів адміністраторів за допомогою клієнтського додатку.
7. Забезпечити адміністраторів системи можливістю редагування таблиць баз даних засобами клієнтського додатку.
8. Клієнтський додаток повинен надавати користувачу системи стислу статистику роботи системи.
9. Адміністратор повинен мати змогу додавати звіти згідно якості роботи співробітників.
10. Адміністратор системи повинен мати змогу ідентифікувати співробітників за допомогою карт-доступу.

1.1. Серверний додаток.

Серверний додаток міститиме в собі ряд модулів які будуть відповідати за

реалізацію моделей, взаємодію з базою даних, обробку та відправку запитів на клієнтський додаток, аутентифікації та регістрації користувачів системи, отримання різноманітної інформації, статистики використання, яка потім може бути використана іншими частинами системи (рисунок 1.1).

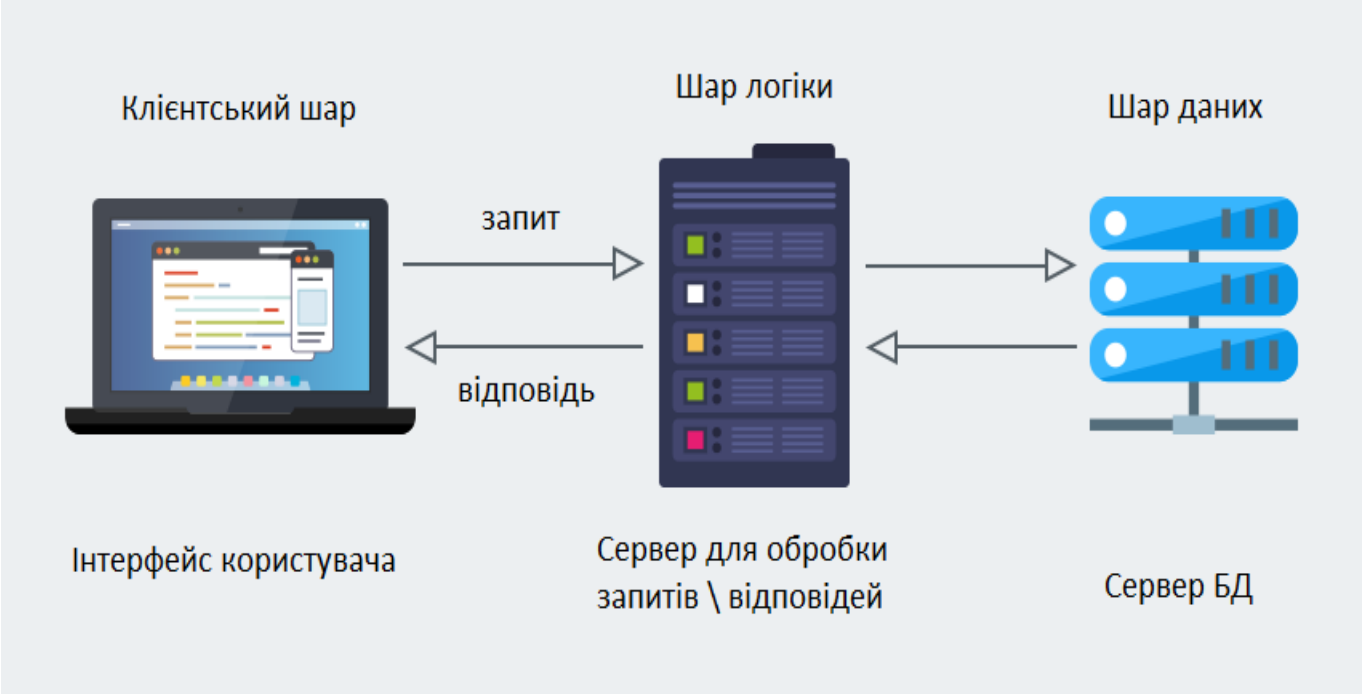

Рисунок 1.1 – Типова архітектура клієнт-сервер

Необхідні можливості, функції, які повинні бути наявні у додатку, є:

— аутентифікація вже існуючих користувачів;

— регістрація нових користувачів за допомогою електронної адреси та паролю;

— можливість збереження у базі даних інформації про користувачів;

— можливість додання безконтактних карт у базу даних та присвоєння їх окремим користувачам;

— додання маршрутів та збереження їх у базі даних;

— надання адміністраторам-керівникам інформації щодо ефективності та якості виконання роботи їх співробітниками у вигляді звітів.

— відстеження у реальному часі місцезнаходження робітників.

1.2. Апаратний додаток.

Апаратний додаток матиме реалізацію на базі мікроконтролеру який буде передавати дані на сервер для їх подальшої обробки. Однією з основних функцій мікроконтролеру для даної системи буде прийом безконтактних карт та відправка їх ідентифікаторів на сервер (рисунок 1.2).

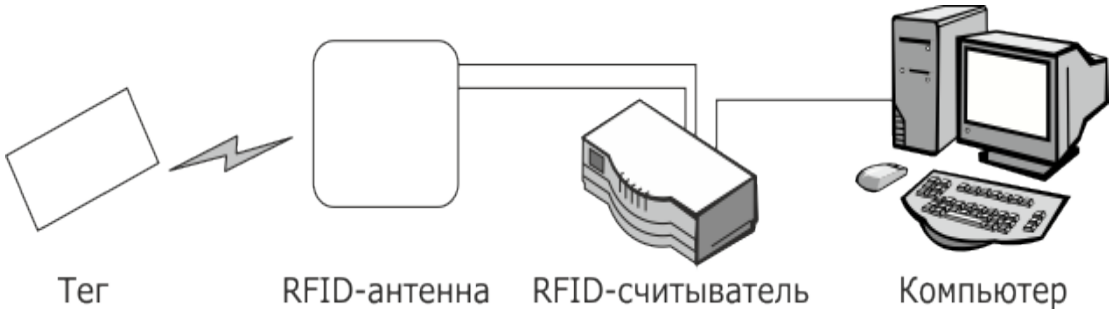

Рисунок 1.2 – Базові компоненти системи радіочастотної ідентифікації

Апаратний додаток також повинен мати функцію сигналізування робітнику успішності відправлених серверу даних а також якості виконання ним роботи. Таким чином оператори які матимуть змогу в реальному відстежувати робітників не повинні будуть спостерігати за якістю роботи так як за них це зробить додаток.

1.3. Клієнтський додаток

Використання клієнтського додатку має сенс для налаштування злагодженого ланцюгу існування програмно-апаратному комплексу адже саме завдяки логічному відображенню обробленої інформації дозволятиме аналізувати та приймати рішення. Розробка додатку вирішує також проблему взаємодії звичайного користувача системи з сервером.

Кожен клієнт містить реалізацію лише користувацького інтерфейсу та не займається безпосередніми маніпуляціями базою даних. При настанні необхідності виконання певних дій, що потребують втручання у віддалену частину системи, виконується запит до серверу, що містить всю необхідну бізнес-логіку для обробки

даних. Сервер в свою чергу делегує виконання безпосередніх операцій над даними в базі даних серверу баз даних, що займається виконанням необхідних команд над інформацією, яка знаходиться в базі.

Необхідні елементи, функції, можливості які повинні бути присутні у клієнтському додатку:

— інтерфейс має бути інтуїтивно-зрозумілий навіть для звичайного користувача;

— повинна бути присутня можливість аутентифікація користувача за допомогою типової сторінки;

— можливість додавання співробітників через спеціальну форму;

— можливість ознайомлення з попередніми звітами по роботі;

— можливість додавання апаратних додатків у базу даних серверу;

— форма для зв'язку з адміністратором системи;

— сторінка зі схемою взаємодії між апаратними додатками;

— сторінка для відстежування стану апаратної частини додатків;

Таким чином, кожен рівень програмно-апаратного комплексу повинен відповідати лише за окремий функціонал призначений для нього та делегувати виконання операцій, які йому не належать, іншому рівню. Це забезпечує можливість паралельної розробки кожного рівня та окремого тестування кожного з них.

Підхід дозволить в майбутньому більш ефективно налаштовувати розробку продукту шляхом розподілення роботи між окремими людьми. Це дозволить як для розробника так і для підприємства використовувати людські ресурси в потрібному для них руслі.

2.АНАЛІЗ КОМПЛЕКСІВ МОНІТОРИНГУ ТА УПРАВЛІННЯ АКТИВНОСТЯМИ ОХОРОННОЇ СИСТЕМИ

Призначення даного комплексу є вирішення проблеми моніторингу активностей охоронної системи, контроль за якістю виконанням працівниками їх роботи за допомогою апаратної ідентифікації з використанням безконтактних карт. Система повинна мати зрозумілий інтерфейс для керівників що мають контролювати виконання робіт їх співробітниками а також управління вже створеними системами. Використання саме «програмно-апаратного» комплексу матиме в собі переваги у кінцевій ціні даного рішення та «діджиталізації» в міру.

На даний момент часу на різноманітних підприємствах, компаніях керівництво користується рішенням ідентифікації своїх робітників за допомогою радіочастотної ідентифікації. Їх використовують для того щоб слідкувати за людьми в приміщеннях, контролювати доступ до тих чи інших приміщень, відстежувати час коли співробітники прибувають на робоче місце, коли його покидають. В цілому, дане рішення для подібних задач має право на існування. Але досліджуючи мережу інтернет на предмет подібних рішень у сфері охоронних систем одним із основних способів використання було знайдено лише як для контролю доступу людей. Рішень, які дозволяли б використовуючи технологію радіочастотної ідентифікації як для моніторингу та управління охоронною системою не було знайдено.

До частково аналогічних рішень можна відмітити рішення які присутні і по цей час на більшості підприємствах, компаніях як мінімум у нашій країні. Також як це і працює на базі НТУУ «КПІ імені Ігоря Сікорського» де на території кампусу розташований центральний сервер з якого ведеться відео нагляд за територією та корпусами. В корпусах розташовані турнікети через які контроль відбувається за допомогою радіочастотної ідентифікації. Але у даному рішенні наявний сильний

вплив людського фактору на ситуацію в охоронній системі. Багато факторів залежить як і від людини яка спостерігає зі сторони серверу так і від людей які знаходяться в корпусах, на території зовні (рисунок 1.3).

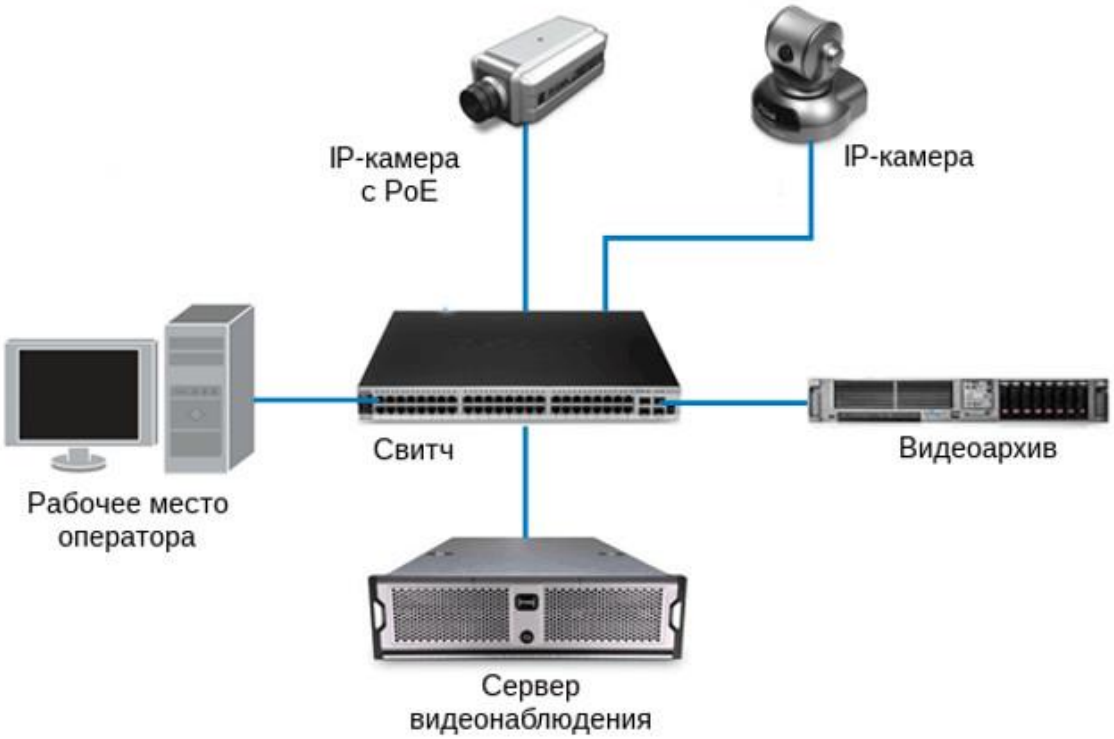

Рисунок 1.3 – Типова схема охоронної системи на базі відео нагляду

В реальному часі не має можливості хоча б приблизно дізнатися без застосування спеціальних засобів зв'язку на якій частині території знаходиться та чи інша людина, наприклад, у випадку якщо вона за межею огляду камер відео спостереження. Також, не має можливості відстежити якість виконання роботи людьми. Що має свій сенс для контролю за виконаною роботою.

Рішення що мають право на існування та відстежувати в реальному часі місцезнаходження людей можуть бути занадто дорогими у промислових масштабах, коли охоронна система знаходиться на великій території і в один момент часу знаходиться багато людей на чергуванні.

3.ЗАСОБИ РОЗРОБКИ ПРОГРАМНО-АПАРАТНОГО КОМПЛЕКСУ

Основним середовищем розробки було обрано середовище розробки Visual Studio Code від компанії Microsoft, що є одним із популярних середовищ для розробки додатків як для серверної частини так і для апаратної. Для налаштування середовища розробки був використаний вбудований сервіс пошуку розширень.

3.1. Середовище розробки Visual Studio Code

Visual Studio Code — редактор вихідного коду, розроблений Microsoft для Windows, Linux і macOS. Позиціонується як «легкий» редактор коду для розробки веб-і хмарних додатків а також і для різних платформ. Включає в себе відладчик, інструменти для роботи з Системою контролю версій, підсвічування синтаксису, засоби для рефакторингу. Має широкі можливості для налаштувань: призначання для користувача теми, поєднання клавіш і файли конфігурації. Розповсюджується безкоштовно, розробляється як програмне забезпечення з відкритим вихідним кодом, але готові збірки розповсюджуються під проприєтарною ліцензією.

Visual Studio Code заснований на Electron і реалізується через веб-редактор Monaco, розроблений для Visual Studio Online.

Тож можна виділити основні аспекти які мали вплив на вибір середовища програмування:

— палітра команд;

— управління робочим простором;

— система контролю версій;

— управління настройками;

— написання коду для різних додатків;

3.2. Середовище розробки Arduino IDE

В рамках платформи Ардуіно програма Arduino IDE робить те ж — допомагає програмістам писати програми. З її допомогою скетч, написаний на мові Arduino, перевіряється, перетворюється в C++, компілюється, завантажується в Arduino. Теоретично, ви можете обійтися без цієї програми, але Arduino IDE ідеальний вибір для розробки під мікроконтроллери бази Node MCU. Тому перше, що ви повинні зробити - це знайти і встановити собі цю середу програмування. Це абсолютно не важко і абсолютно безкоштовно[2].

Цикл програмування Arduino спрощено виглядає так:

— підключення плати до комп'ютеру;

— написання програми-скетчу;

— завантаження написаного скетчу на плату через USB-з'єднання;

— виконання платою написаного скетчу.

3.3. Пакет програмного забезпечення Docker

Docker — програмне забезпечення для автоматизації розгортання і управління додатками в середовищах з підтримкою контейнеризації. Дозволяє «упакувати» додаток з усім його оточенням і залежностями в контейнер, який може бути перенесений на будь-яку Linux-систему з підтримкою cgroups в ядрі, а також надає середовище з управління контейнерами. Спочатку використовував можливості LXC, з 2015 року застосовував власну бібліотеку, що абстрагує віртуалізаційних можливості ядра Linux - libcontainer. З появою Open Container Initiative почався перехід від монолітної до модульній архітектурі[3] (рисунок 3.1).

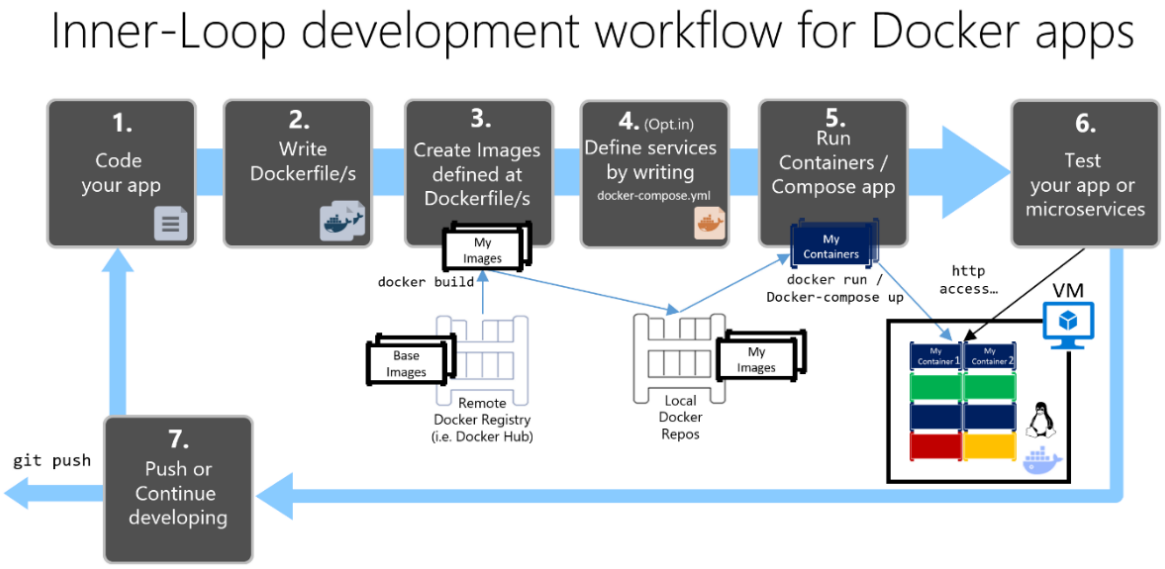

Рисунок 3.1 – робочий процес розробки для додатків Docker

Але використання Docker`ру без його інструментарію не має сенсу. Отже також для розробки був обраний інструмент Docker-Compose. Compose інструмент для створення і запуску багатоконтейнерних Docker додатків. У Compose, ви використовуєте спеціальний файл для конфігурації ваших сервісів у додатки. Потім, використовується проста команда, для створення і запуску всіх сервісів з конфігураційного файлу.

Compose чудовий для розробки, тестування і налаштування середовища, а також безперервної інтеграції. Ви цьому розділі ви можете отримати додаткову інформацію про розв'язуваних завданнях.

3.4. Система контролю версій — Git

Система контролю версій або VCS може значно полегшити роботу розробників, які намагаються проаналізувати зміни і вклади в загальний код. Простіше кажучи, система контролю версій - це ключовий елемент в системі управління настройками програмного забезпечення, які відповідають потребам проекту. VCS дають можливість призначати для певних змін / ревізій / оновлень

літерні або числові значення. Також можуть надати інформацію про тимчасові мітки і ідентифікатор людини внесла зміних[4] (рисунок 3.2).

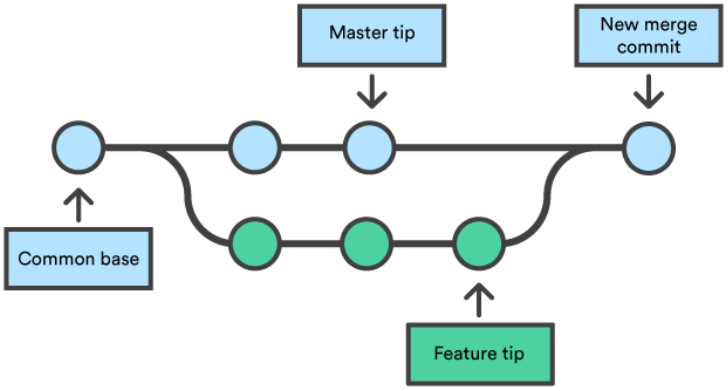

Рисунок 3.2 – схема розгалуження розробки за допомогою Git

Основі переваги використання Git:

— легко масштабувати роботу за проектом, легко включати в процес потрібних співробітників. При грамотному веденні сховища в найкоротші терміни можна розгорнути додатковий майданчик або видалити зайвого розробника з проекту;

— проект легко передавати з команди в команду, від розробника до розробника: просто відправите посилання на репозиторій;

— нічого не втрачається. Всі версії файлів зберігаються, і ситуацій, при якій може «загубитися» безповоротно та чи інша розробка, не буває;

— можна паралельно налаштувати роботу між розробниками і цілими командами: розподіліть навантаження між співробітниками - так ви значно підвищите швидкість виконання проектних робіт;

— можна відстежувати, як йде робота, записувати її в так звані «лог» файли, і в будь-який момент можна побачити, хто і коли вніс зміни;

Підводячи підсумок: Git - це система роботи команди програмістів, при якій всі вони можуть вносити зміни одночасно, не побоюючись за працездатність проекту. Git необхідний, якщо над сайтом працюють відразу кілька команд

розробників, і сам сайт при цьому вельми складний в архітектурі і складається з великого числа файлів.

3.5. Мова програмування Python

Легкість коду, що робить мову Python особливо найкращим вибіром для новачків в програмуванні, - один з принципів філософії Python, яку можна узагальнити наступним чином.

Оскільки Python орієнтований на легкість коду, в ньому часто використовуються ключові слова англійською мовою там, де інші мови програмування зазвичай використовують розділові знаки. Особливе його відмінність полягає в тому, що для угруповання інструкцій в блоці коду Python використовує відступи, а не ключові слова або знаки пунктуації. У мові Pascal, наприклад, початок блоків позначається ключовим словом begin і закінчується ключовим словом end, в той час як програмісти на C використовують фігурні дужки для позначення блоків коду. Дуже часто такий підхід угруповання блоків відступами критикується програмістами, знайомими з іншими мовами, але, безсумнівно, використання відступів в Python дозволяє програмам виглядати менш нагромадженими[11].

Використання даної мови програмування має свої переваги:

— Python безкоштовний так як це вільне програмне забезпечення з відкритим вихідним кодом.

— Python легкий у вивченні - він має простий синтаксис.

— Python легкий в обслуговуванні - має модульну структуру.

— Python має багатий «арсеналом» - він пропонує велику стандартну бібліотеку, яка легко інтегрується в програми.

— Python інтерпретується - компіляція не потрібно.

— Python є високорівневим мовою - він має статичний розподіл пам'яті.

— Python розширюваний - дозволяє додавати низькорівневі модулі.

— Python універсальний - підтримує як процедурний, так і об'єкт але орієнтований методи програмування.

— Python гнучкий у використанні - з його допомогою можна створювати консольні програми, додатки графічного інтерфейсу, а також сценарії для взаємодії зовнішніх програм з веб-сервера.

3.6. Технології для розробки інтерфейсу

Кожна сторінка в Інтернеті, яку ви відвідуєте, будується на виконанні окремих інструкцій крок за кроком. Ваш браузер (Chrome, Firefox, Safari і т.д .... Якщо використовуєте Internet Exploler, не читайте далі, вимкніть комп'ютер і йдіть гуляти) відіграє колосальну роль у відображенні коду і тим, що ми можемо бачити на наших екранах і навіть взаємодіяти. Пам'ятайте, що код без браузера - це просто текстовий файл - це коли ви ставите цей текстовий файл в браузер, що відбувається диво. Коли ви відкриваєте веб-сторінку, браузер відображає HTML і інші мови програмування в максимально зрозумілому для вас форматі.

HTML і CSS насправді всього лише структура сторінки і інформація про стилі. Перш ніж перейти до JavaScript і інших мов програмування, необхідно знати основи HTML і CSS, оскільки вони знаходяться на передній частині кожної веб-сторінки і додатки.

На самому початку 1990-х років HTML була єдиною мовою, доступним в Інтернеті. З тих пір багато чого змінилося і тепер одним з найпоширеніших мов програмування є JavaScript.

В першу чергу, необхідно зрозуміти, що HTML - основа кожної веб-сторінки, незалежно від складності сайту або кількості задіяних технологій. Це важливий навик для будь-якого веб-професіонала і відправна точка для всіх, хто має відношення до створення або редагування контенту в Інтернеті. І, на щастя для нас, йому дивно легко вчитися.

CSS - це каскадні таблиці стилів. Ця мова розмітки визначає, як HTML-елементи веб-сайту повинні відображатися на інтерфейсі сторінки.

У той час як HTML є основною структурою нашого сайту, CSS - це те, що дає всьому нашому сайту стиль. Кольори, цікаві шрифти і фонові зображення - все це заслуга CSS. Ця мова впливає на весь настрій веб-сторінки, що робить його неймовірно потужним інструментом і важливим навиком для веб-розробників. Він також дозволяє веб-сайтам адаптуватися до різних розмірів екрану і типів пристроїв.

JavaScript є більш складною мовою, ніж HTML або CSS, і він не був випущений в бета-версії до 1995 року. В даний час JavaScript підтримується всіма сучасними веб-браузерами і використовується практично на кожному сайті в Інтернеті для більш потужних і складних функцій.

JavaScript - це логічна мова програмування, який можна використовувати для зміни вмісту веб-сайту і змусити його поводитися по-різному у відповідь на дії користувача. Загальне використання JavaScript включає в себе вікна підтвердження, заклики до дії і додавання нових ідентифікаторів до існуючої інформації.

Для повноцінної реалізації клієнтського додатку, його інтерфейсу, було обрано для використання сучасний фреймворк Vue.js.

Vue.js — це JavaScript-фреймворк для створення користувацьких інтерфейсів. Він добре інтегрується з іншими бібліотеками або існуючими проектами. Це робить його придатним інструментом як для невеликих проектів, так і для складних односторінкових додатків.

API, або інтерфейс прикладного програмування, є програмним посередником, який дозволяє двом додаткам спілкуватися один з одним. API часто надає дані, які інші розробники можуть використовувати в своїх власних додатках, не турбуючись про бази даних або відмінностях у мовах програмування. Розробники часто отримують дані з API в форматі JSON, які вони інтегрують в інтерфейсні програми. Vue.js відмінно підходить для використання цих видів API.

Досить багато фреймворків мають вбудований апі для відправки HTTP запитів. Angular 2 має http модуль, JQuery має $ .ajax, і до версії Vue 2.0, Vue.js мав vue-resource. В Vue 2.0 розробники фреймворка вирішили що наявність вбудованого http клієнта є надмірною і буде краще використовувати сторонні бібліотеки. Однією з найпопулярніших бібліотек є Axios.

Axios це потужний http клієнт. Він використовує Проміс за замовчуванням і працює як на стороні клієнта так і на стороні сервера (що робить можливим завантаження даних під час рендеринга на сервері). Також цю бібліотеку досить легко використовувати в Vue, так як вона використовує Проміс, можна комбінувати її з async / await щоб отримати лаконічний не громіздкий код відправлення запитів.

3.7. Технології для розробки серверного додатку

Існує безліч способів підняти свій власний веб-сервер, який буде обробляти HTTP запити користувачів і повертати їм в браузери результат.

Оскільки ми використовуємо Python в якості основного мови, бібліотеку, яка спрощує нам створення веб-сервера, виберемо теж зі світу Python.

Flask - це інструмент для веб-сайтів на мові Python. Являє собою мікрофреймворк з вбудованим веб-сервером. Домовимося, що ви використовуєте Linux в якості операційної системи, або знаєте як виконати аналоги команд в Windows.

Написання веб-додатку передбачає розробку HTTP-запитиів на свій сервер, щоб отримати дані для заповнення динамічних частин всього сервісу.

Отже для цих задач було обрано Python 3, Flask та Connexion для створення запитів API REST, які можуть включати перевірку вводу та виводу, та надавати документацію Swagger як бонус. Також включений простий, але корисний веб-додаток на одній сторінці, який демонструє використання API з JavaScript та оновлення DOM за допомогою нього[5] (рисунок 3.3).

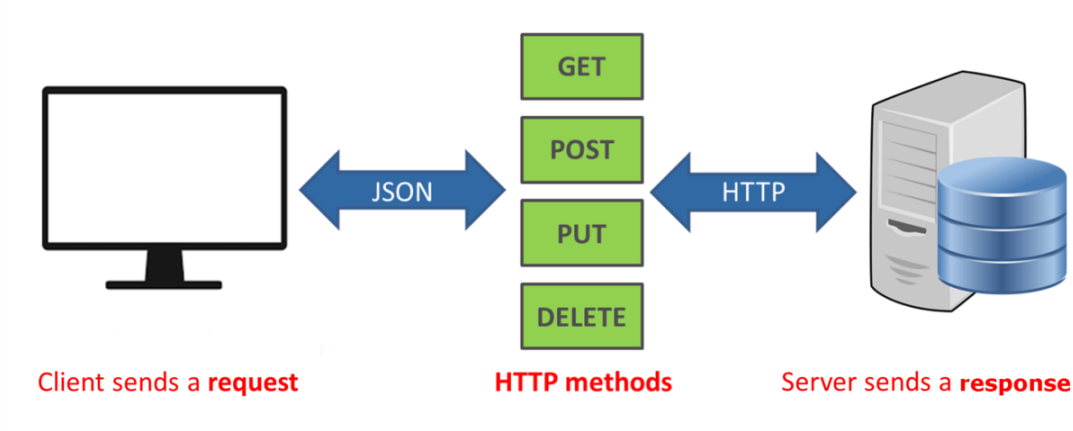

Рисунок 3.3 – схема запитів REST API між сервером та клієнтом

API REST обслуговуватиме просту структуру даних, а будь-які оновлення позначаються новою міткою часу.

Ці дані можуть бути представлені в базі даних, збережені у файлі або бути доступними через якийсь мережевий протокол, але для нас структура даних в пам'яті працює добре. Однією з цілей API є від'єднання даних від програми, яка їх використовує, приховуючи деталі реалізації даних.

Спосіб використання REST API передбачає собою налаштування клієнта на відправку запитів згідно вище вказаних методів на кінцеві точки серверного додатку. Таким чином ми отримаємо клієнт-серверний додаток який дозволить в майбутньому більше сконцентруватися на розробці певної частини всього додатку.

3.8. Формування вимог до системи

До вимог згідно використання додатків можна віднести цілий пакет програмного забезпечення, бібліотек, фреймворків, що потребують встановлення для своєї роботи.

3.8.1. Вимоги до програмного забезпечення

Програмне забезпечення являє собою комплекс апаратного, серверного та клієнтського додатків, засобів які необхідні для їх запуску.

Клієнт: Vue.js: 3.0

СУБД: версія PostgreSQL: 12.2

Docker, Docker-compose: версія 19.03

Сервер: версія Flask 1.1.2

Інструмент розробки: Visual Studio Code

Операційна система: Ubuntu LTS 18.04

3.8.2. Вимоги до апаратного забезпечення

Процесор: Intel (R) Xeon (R)

Процесор: E5-2680 v4 при 2,40 ГГц

ОЗУ: 8 Гб.

Місце на жорсткому диску: 500 Гб

Архітектура: x86-64

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

В процесі розробки та проектування базового шаблону проекту використовувалися різноманітні технології, засоби розробки, методи. Отже, що саме буде покривати наш програмний комплекс:

— налаштування середовища розробки в docker-compose;

— створення серверної частини на базі фреймворку Flask;

— створення додатку клієнтської частини за допомогою фреймворку Vue.js;

— розміщення створеної кодової бази у так звані «контейнери» для спрощення розгортання на сервері в подальшому;

— налагодження зв'язку клієнт-сервер за допомогою REST API;

— ідентифікація користувачів за допомогою RFID технології;

Додаток розроблений за схемою дворівневої архітектури «Клієнт-Сервер», яка в подальшому може бути масштабована без додаткової зміни програмного коду для колективної роботи користувачів (рисунок 4.1).

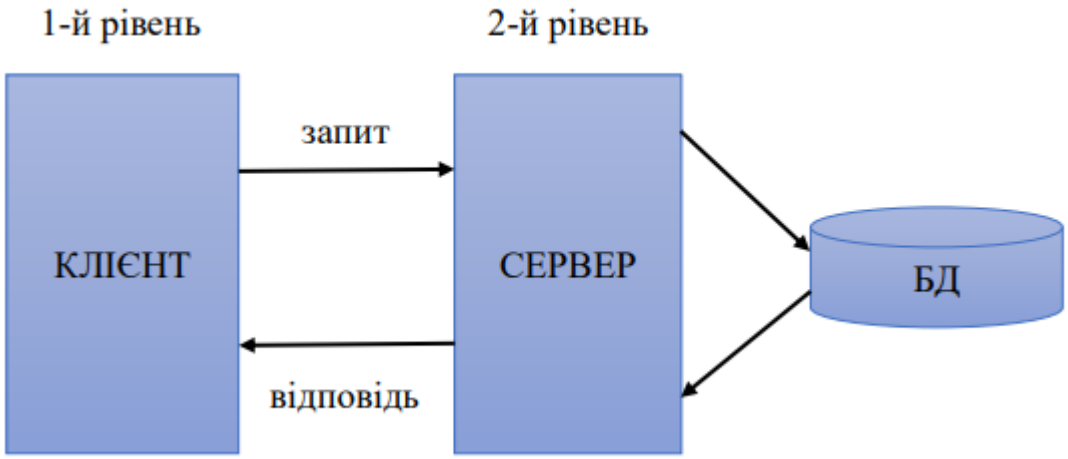

Рисунок 4.1 – Схема дворівневої архітектури

Перевагами даної схеми є:

— можливість розподілу функцій обчислювальної системи між декількома незалежними серверами.

— збереження всіх даних на окремому захищеному сервері баз даних.

— можливість одночасної роботи групи користувачів.

— гарантія цілісності даних.

4.1. Структура проекту

Найгірша помилка великих додатків - це архітектура моноліту у вигляді величезної кодової бази з великою кількістю залежностей, така структура сильно уповільнює розробку особливо впровадження нових функцій. Тому було організовано розподілення наступним чином. Для кожного компонента виділено власну папку для модулів компонента. Важливо щоб кожен модуль залишився маленьким і простим.

В іншому випадку: було б складно розвивати продукт - додавання нового функціоналу і внесення змін в код будуть проводитися повільно і мати високий шанс поломки інших залежні компонент. Вважається, що якщо бізнес-юніти не розділені, то можуть виникнути проблеми з масштабуванням додатків.

В нашому випадку кожен компонент повинен має «шари», наприклад, для роботи з Інтернетом, бізнес-логікою, доступом до БД, ці шари мають свій власний формат даних не змішаний з форматом даних сторонніх бібліотек. Це не тільки чітко розділяє проблеми, а й значно полегшує перевірку та тестування системи.

Для більш логічної структури наші додатки були розподілені наступним чином(рисунок 4.2).

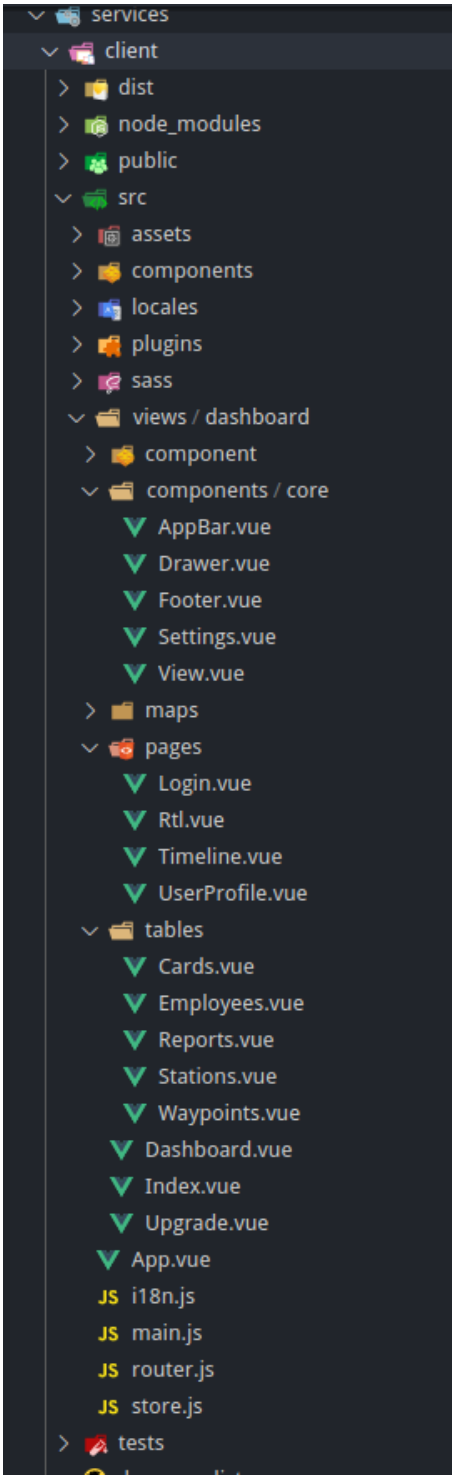

Рисунок 4.2 – Структура клієнтського додатку та його компоненти

Для серверного додатку було обрано розподілення яке покращить розробку та в загалом розуміння самої бізнес-логіки додатку та проекту в цілому(рисунок 4.3).

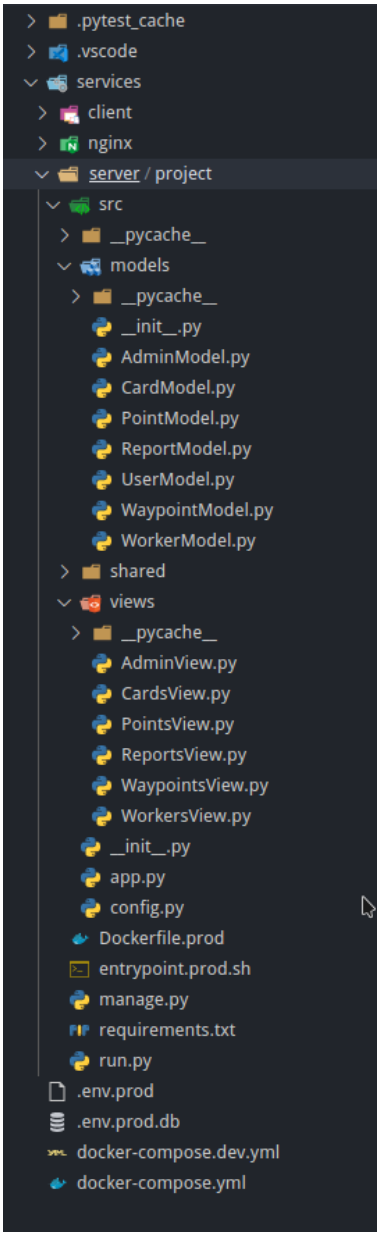

Рисунок 4.3 – Структура клієнтського додатку та його компоненти

4.2. RESTful веб-сервіс Flask

REST - це архітектура, тобто принципи побудови розподілених гіпермедіа систем, того що іншими словами називається World Wide Web, включаючи універсальні способи обробки і передачі станів ресурсів по HTTP[6].

Архітектура REST розроблена щоб відповідати протоколу HTTP використовується в мережі Інтернет.

Центральне місце в концепції RESTful веб-сервісів це поняття ресурсів.

Ресурси представлені URI. Клієнти відправляють запити до цих URI використовуючи методи представлені протоколом HTTP, і, можливо, змінюють стан цих ресурсів.

Методи HTTP спроектовані для впливу на ресурс стандартним способом наведені в таблиці 4.1.

Метод	Дія	Приклад
GET	Отримати інформацію про ресурс	example.com/api/orders (отримати список замовлень)
GET	Отримати інформацію про ресурс	example.com/api/orders/123 (отримати замовлення #123)
POST	Створити новий ресурс	example.com/api/orders (створити нове замовлення із переданих даних)
PUT	Оновити ресурс	example.com/api/orders/123 (оновити замовлення #123 даними переданими запитом)
DELETE	Видалити ресурс	example.com/api/orders/123 (видалити замовлення #123)

Таблиця 4.1. Можливі методі REST

Дизайн REST не дає рекомендацій яким конкретно повинен бути формат даних переданих з запитами. Дані передані в тілі запиту можуть бути у форматі JSON, або передані за допомогою аргументів в URL. Використання технології REST в даному випадку передбачає також застосування в цілях проекту схеми розподілення даних додатку, його логіку та інтерфейсу MVC схема якої наведена на наступному малюнку(рисунок 4.4).

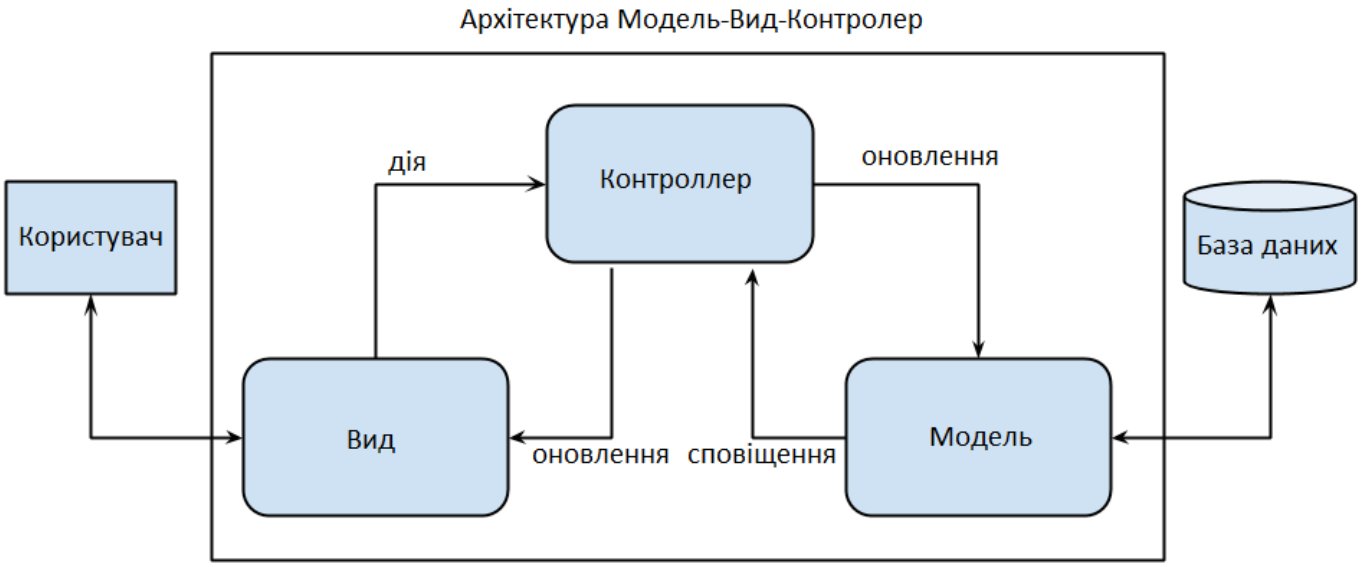

Рисунок 4.4 – деревоподібна ієрархія компонентів Vue.js

Тому для використання даної архітектури було обрано фреймворк Flask на основі мови програмування Python. Flask — це веб-мікроструктура, побудована на базі Python. Вона називається мікроструктурою, тому що не вимагає для роботи спеціальних інструментів або плагінів. Мікроструктура Flask відрізняється компактністю, гнучкістю і високим рівнем структурування, за рахунок чого вона більш краща у порівнянні з іншими програмними структурами.

4.3. Vue.js клієнт

Vue.js - це прогресивний фреймворк, що підходить для створення користувацьких інтерфейсів. Назва фреймворка співзвучно з view, тобто з поданням (якщо говорити про моделі MVC).

Фреймворк використовується для вирішення завдань саме рівня уявлення, його просто інтегрувати з іншими бібліотеками та проектами. Іншими словами, Vue.js - це інструмент, який можна впроваджувати поступово. Можливість впровадження пов'язана з тим, що Vue.js прагне до прогресивності: підтримку Vue можна додати у вже існуючий проект, завдяки чому його функціональність буде значно розширена. І це відрізняє його від інших фреймворків[7].

Взагалі простота інтеграції - це одне з основних переваг даного фреймворка, особливо в поєднанні з можливістю інтеграції з бекенд фреймворками.

Інші сильні сторони Vue.js - це простота в освоєнні і хороша документація, а також висока продуктивність.

Центральна концепція Vue.js - це концепція компонентів, тобто невеликих частин інтерфейсу користувача, які можна використовувати повторно. Таким чином, і сам додаток складається з частин-компонентів. Один компонент може включати кілька інших компонентів, тобто використовується деревоподібна ієрархія[8] (рисунок 4.5).

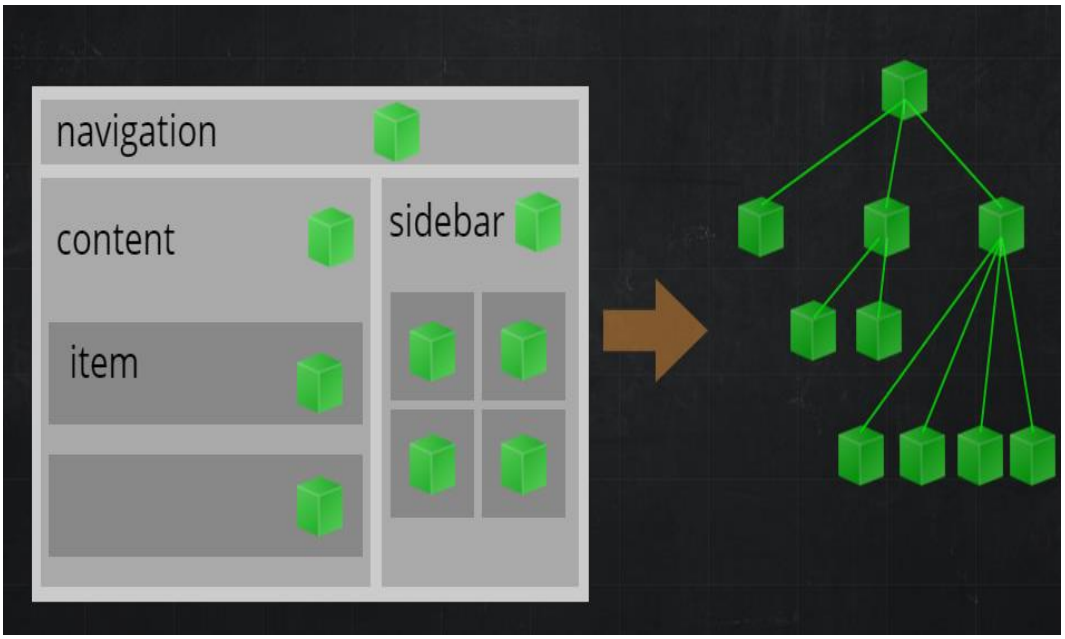

Рисунок 4.5 – деревоподібна ієрархія компонентів Vue.js

Vue.js - це реактивний MVC фреймворк: уявлення (view) автоматично змінюється при зміні змінної (моделі). Vue.js має дуже хорошу екосистему, яка включає в себе безліч корисних сторонніх компонентів та інструментів.

4.4. Серверна база даних

Для фреймворку Flask існує спеціально створене розширення що називається Flask-Migrate, . Це розширення є Flask-обгорткою для Alembic, основою

для міграції бази даних SQLAlchemy. Робота з міграціями баз даних додає трохи роботи спочатку, але це невелика ціна, щоб заплатити за надійний спосіб внесення змін до нашої базу даних в майбутньому.

Для взаємодії з міграцією необхідно використовувати Flask-SQLAlchemy, розширення, яке забезпечує Flask-дружню оболонку до популярного пакету SQLAlchemy, який є Object Relational Mapper або ORM. ORM дозволяють додаткам керувати базою даних з використанням об'єктів високого рівня, таких як класи, об'єкти і методи, а не таблиці і SQL. Завдання ORM - перевести операції високого рівня в команди бази даних[9].

Найприємніше в SQLAlchemy полягає в тому, що це ORM не для одного, а для багатьох реляційних баз даних. SQLAlchemy підтримує довгий список движків баз даних, включаючи популярні MySQL, PostgreSQL.

Дані, які будуть зберігатися в базі даних, будуть представлені набором класів, зазвичай званих моделями баз даних. Рівень ORM в SQLAlchemy буде виконувати переклади, необхідні для зіставлення об'єктів, створених з цих класів, в рядки у відповідних таблицях бази даних.

Але в міру того, як додаток продовжує рости, потрібно змінити структуру, яка, швидше за все, додасть нові сутності, але іноді також може змінювати або видаляти елементи. Alembic (інфраструктура міграції, яка використовується Flask-Migrate) зробить ці зміни схеми таким чином, щоб не було потрібно відтворювати базу даних з нуля.

Щоб виконати цю роботу Alembic підтримує репозиторій міграції, який є каталогом, в якому зберігається його сценарії міграції. Кожен раз, коли в схему бази даних вносяться зміни, в репозиторій додається сценарій міграції з докладними відомостями про зміну. Щоб застосувати міграції до бази даних, ці сценарії міграції виконуються в тій послідовності, в якій вони були створені.

На даний момент додаток знаходиться в зародковому стані, але це не завадить обговорити, що буде в стратегії міграції бази даних в майбутньому. Уявіть, що у нас є додаток на нашій машині розробки, а також є копія, розгорнута на виробничому сервері, який знаходиться в мережі і використовується.

Припустимо, що для наступної версії нашої програми нам потрібно внести зміни в свої моделі, наприклад, потрібно додати нову таблицю. Без міграції нам потрібно буде з'ясувати, як змінити схему нашої бази даних, як на локальному хості, так і на нашому сервері, і це може бути великою проблемою.

Але з підтримкою міграції бази даних, після зміни моделей в додатку ми створюємо новий сценарій міграції (flask db migrate), переглянувши його, щоб переконатися, що автоматичне створення зробило правильні речі, застосуймо зміни в базі даних розробки (flask db upgrade). Ми додасмо сценарій міграції в систему управління версіями і зафіксовуємо його[10].

Коли будемо готові випустити нову версію програми на свій production сервер, все, що нам потрібно зробити, це захопити оновлену версію програми, яка буде включати в себе новий сценарій міграції і запустити flask db upgrade. Alembic виявить, що база даних не оновлена до останньої редакції, і виконає всі нові сценарії міграції, створені після попереднього випуску.

4.5. Схема бази даних

Схема бази даних уявляє собою візуальне представлення таблиць в базі даних. Для створення таблиць і відносин між ними інструментів безліч. Побудова схеми дозволяє спростити представлення роботи яку потрібно виконати розробнику системи під час проектування бізнес-логіки(рисунок 4.6).

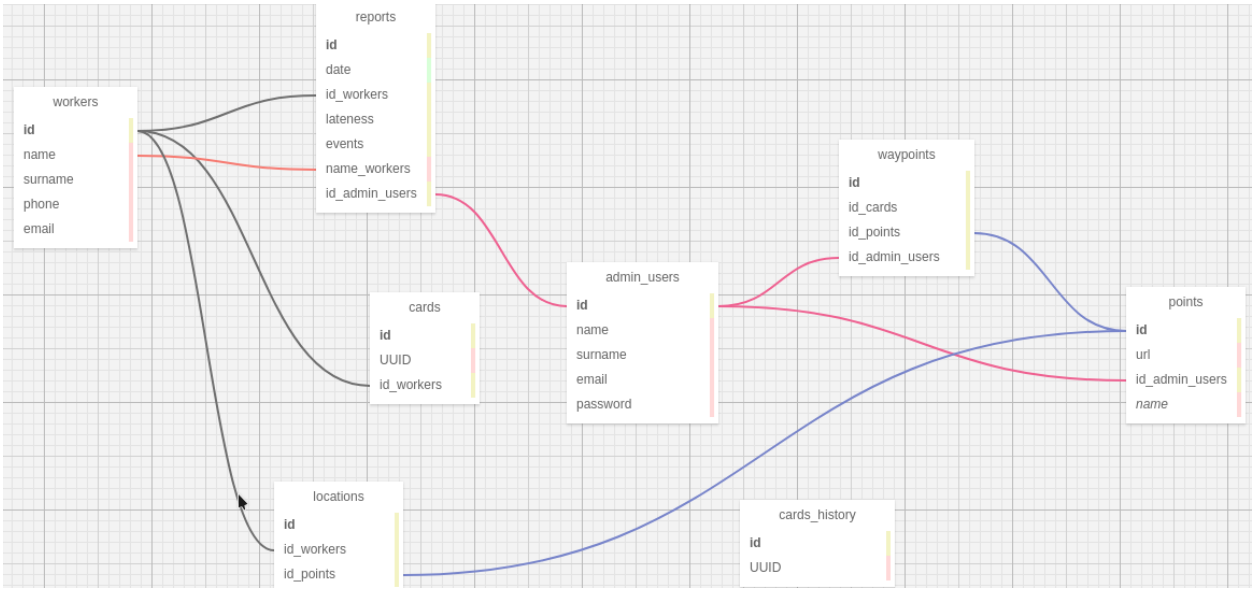

Рисунок 4.6 – Загальна схема бази даних

Таким чином створення схеми дозволяє полегшити життя всім іншим членам команди розробки продукту. Також для розробника після проектування бази даних сучасні інструменти дозволяють згенерувати код для автоматичного створення СУБД вказаних таблиць та їх відношень.

4.6. Проектування мовою UML

Для більш детального представлення процесів які відбуваються в додатках було обрано уніфіковану мову моделювання UML. Використовуючи систему позначень можна здійснити процедури об’єктно орієнтованого аналізу і проектування. В даному випадку мова використовується для візуалізації процесів які відбуваються в програмній системі. На наступному малюнку наведена діаграма прецедентів(рисунок 4.7).

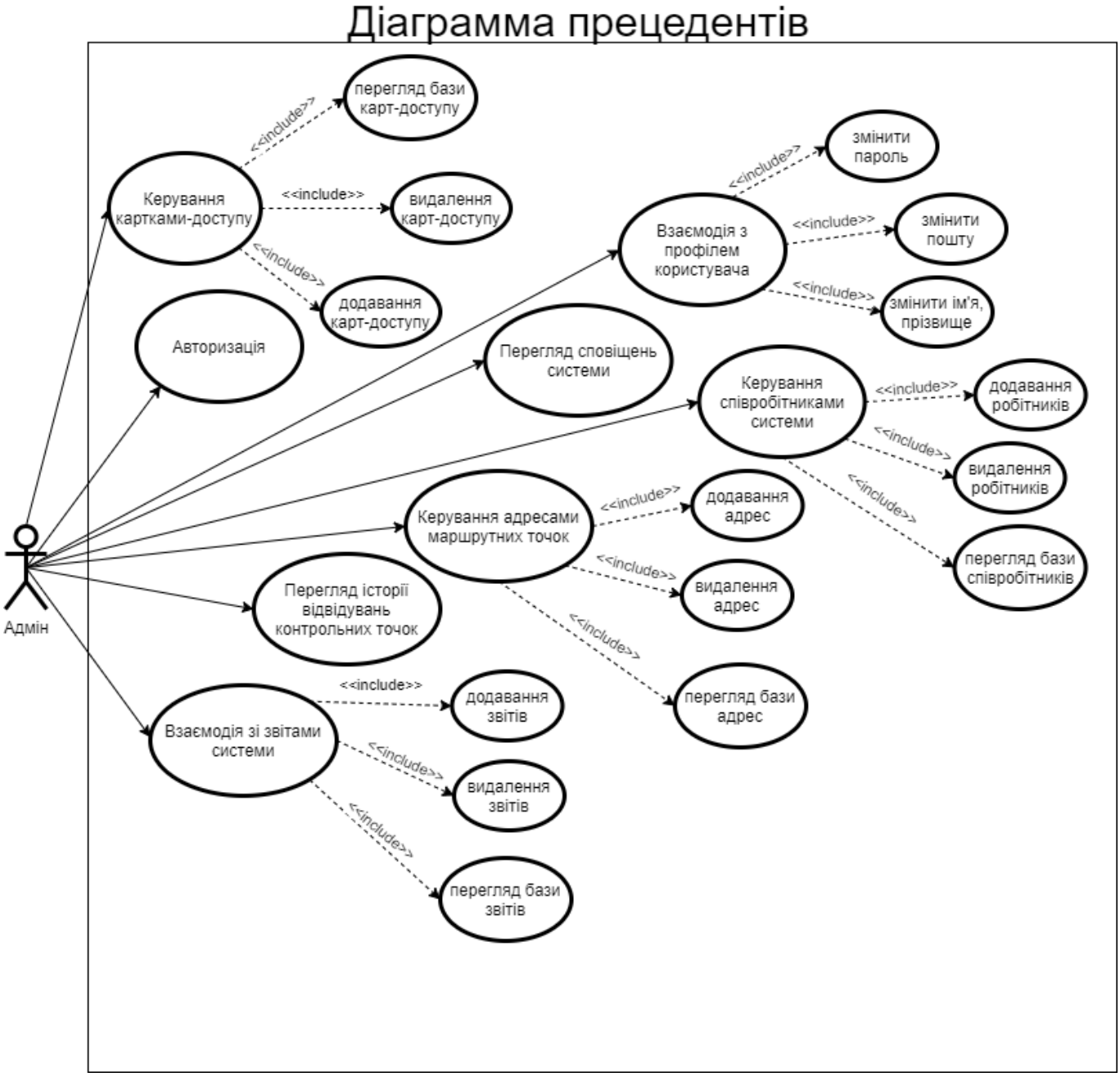

Рисунок 4.7 – Діаграма прецедентів

За допомогою даного представлення ми можемо зрозуміти відношення між акторами та прецедентами нашої системи. Прецедент в даному випадку являє собою можливості модельованої системи.

Для описання процесів, фаз, дій які відбуваються під час авторизації було обрано діаграму послідовностей(рисунок 4.8).

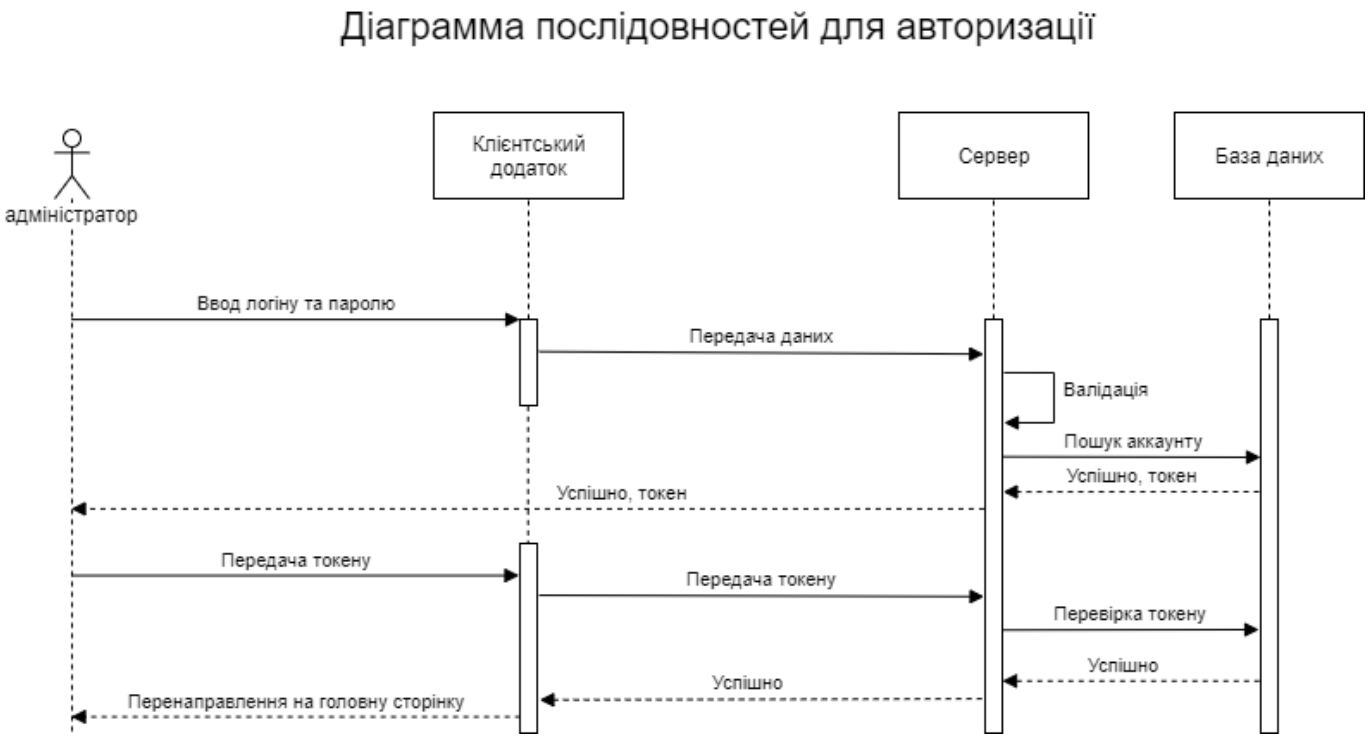

Рисунок 4.8 – Діаграма послідовностей для авторизації

Використання даного підходу дозволяє поетапно зрозуміти як саме в системі виконується в даному випадку процес авторизації користувача. Діаграма включає в себе об’єкти які взаємодіють між собою в рамках сценарію, повідомлення якими вони обмінюються та результати які повертає система.

В клієнтському додатку присутня функція сповіщень про надзвичайні ситуації, тому для відображення зв’язків які присутні у системі сповіщень було обрано відображення за допомогою діаграми класів(рисунок 4.9).

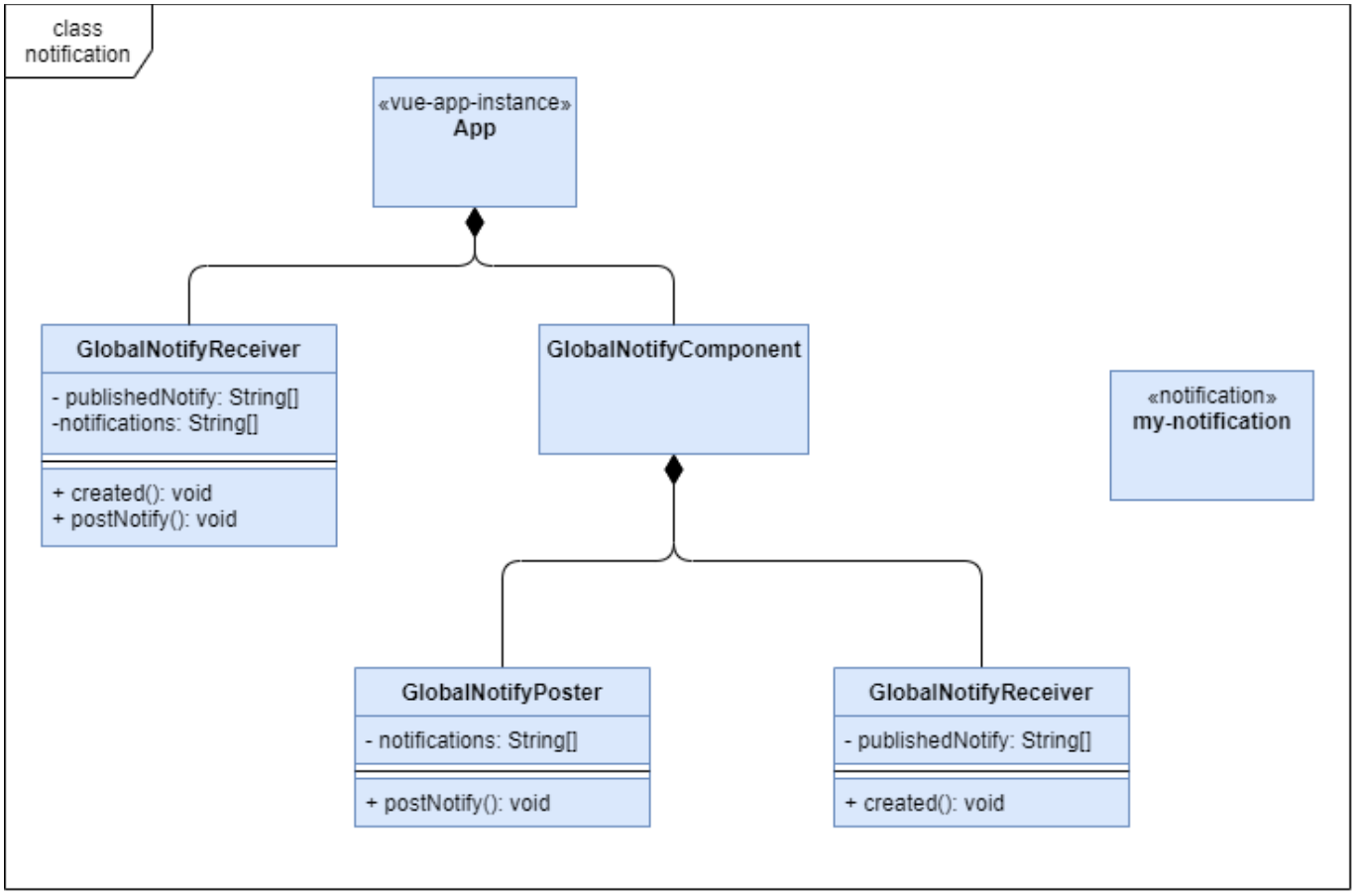

Рисунок 4.9 – Діаграма класів системи сповіщень

В даному випадку діаграма класів відображає зв'язки у системі сповіщень між компонентами присутніми в модулі. Кожний компонент моделюється як клас UML. Клас в свою чергу показує поля даних та атрибути, а методи операції.

Таким чином використання даних видів представлень дозволила першочергово спроектувати програмну систему, зрозуміти принцип її роботи, та в подальшому полегшила розробку самого продукту.

5. РОБОТА КОРИСТУВАЧА ІНТЕРФЕЙСОМ ПРОГРАМНОЇ СИСТЕМИ

Користувальницький інтерфейс - це перегляд додатків, з яким користувач взаємодіє з метою використання програмного забезпечення. Користувач може керувати програмним забезпеченням, а також апаратними засобами за допомогою інтерфейсу користувача. Сьогодні користувальницький інтерфейс є майже в кожному місці, де існує цифрова технологія, прямо від комп'ютерів, мобільних телефонів, автомобілів, музичних плеєрів, літаків, кораблів тощо.

Користувацький інтерфейс є частиною програмного забезпечення та розроблений таким чином, що передбачається забезпечити користувачеві уявлення про програмне забезпечення. Користувальницький інтерфейс забезпечує фундаментальну платформу для взаємодії людина-комп'ютер.

Користувальницький інтерфейс може бути графічним, текстовим, аудіо-відео, залежно від базової апаратної та програмної комбінації. Інтерфейс користувача може бути апаратним чи програмним забезпеченням або комбінацією обох.

Більшість програмних продуктів, особливо прикладного характеру, орієнтованих на кінцевого користувача, працюють в діалоговому режимі взаємодії з користувачем таким чином, що ведеться обмін повідомленнями, що впливають на обробку даних.

Нерідко замість терміна «графічний інтерфейс користувача» використовують інші, більш короткі: «призначений для користувача інтерфейс» або просто «інтерфейс». Це не зовсім вірно, однак з контексту, як правило, можна зрозуміти, про що йде мова. Наприклад, коли фахівці в своїх звітах згадують «інтерфейс програми» або складають «вимоги до інтерфейсу», то, як правило, мають на увазі саме графічний інтерфейс.

Графічний інтерфейс користувача не вичерпується графічним дизайном системи. Рішення про кольорову гаму, шрифтах і про конкретний втіленні елементів

інтерфейсу (тобто, власне про графічний дизайн) приймаються тільки на останньому етапі розробки. До цього необхідно провести серйозну підготовчу роботу: вивчити потреби користувачів; зрозуміти, які вимоги користувачі пред'являють до системи; створити нізкодеталізірованние, а потім і високодеталізовані макети екранів системи з урахуванням виявлених вимог; протестувати макети на реальних або потенційних користувачів системи.

Протестовані макети можна віддавати дизайнеру, який в результаті і визначить кінцевий вигляд системи. Невдалі дизайнерські рішення, наприклад, використання не читаються шрифтів або відразливих кольорових поєднань, можуть звести нанівець результати всієї проведеної роботи. Тому бажано, щоб дизайнер працював спільно з юзабіліті-фахівцем.

Створений відповідно до такою методологією графічний інтерфейс користувача буде максимально зручним, тобто відповідати вимогам юзабіліті. Це гарантує бізнес-ефективність системи.

5.1. Інсталяція додатку на сервер

Інсталяція програмного додатку передбачає собою налаштування середовища docker, та використання пакету docker-compose для автоматичного розгортання на будь якому сервері.

Отже, перш за все створюється файл з назвою docker-compose.yml у директорії проекту. Його конфігуруємо згідно синтаксису описаному в документації[3] (рисунок 5.1).

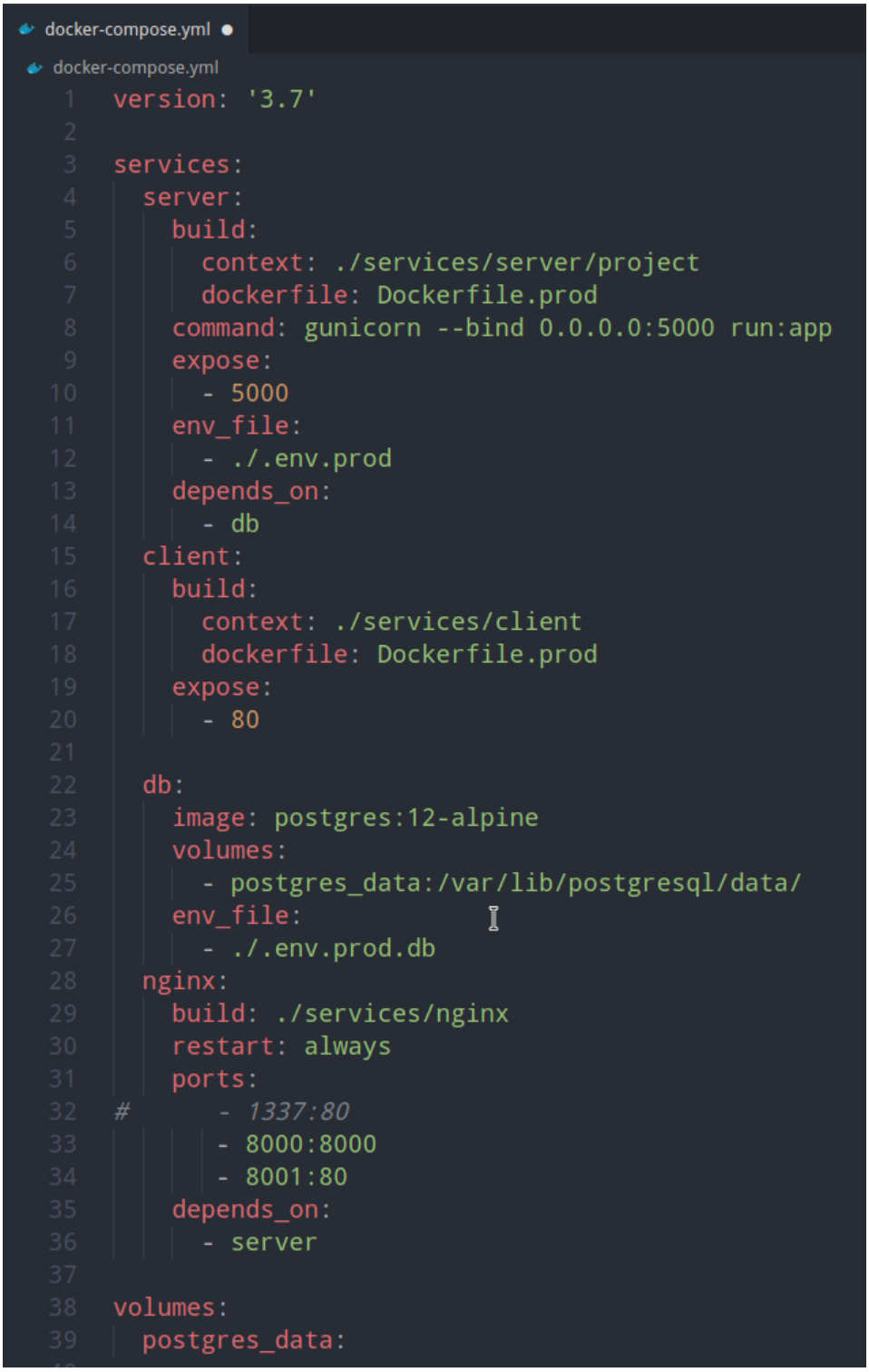

Рисунок 5.1 – необхідна конфігурація docker-compose файлу

Окремо також створюються файли env.prod та env.prod.db. Їх використання передбачає собою запобіжний захід у ході використання системи контролю версій. Сама система налаштовується таким чином щоб дані файли не відображалися для інших у випадку коли буде використовуватися віддалений сервер для зберігання

змін програмного коду. Обидва файли зберігають в собі інформацію стосовно паролів, назв баз даних, порти для доступу до них, директорії, користувачів бази даних. Також даний підхід дозволить з легкістю налаштувати додаток під різні підрозділи, необхідно лише відредагувати вказані вище дані (рисунок 5.2).

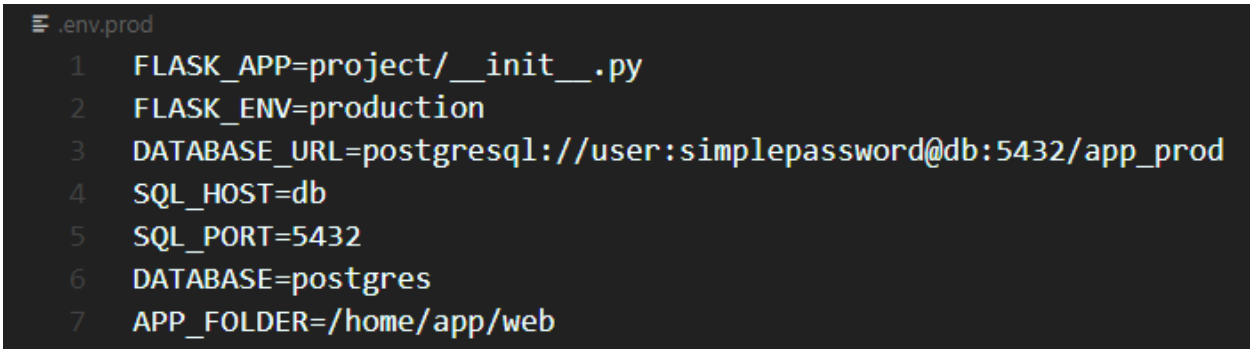

Рисунок 5.2 – необхідна конфігурація файлу .env.prod

В данному файлі адміністратор серверу повинен вказати (змінити) параметри порту, а також у змінній DATABASE_URL замінити user,simplepassword, app_prod на дані вказані у наступному файлі (рисунок 5.3).

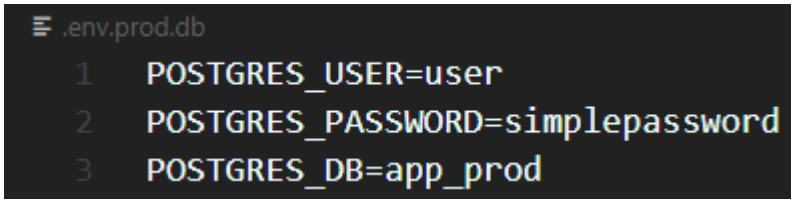

Рисунок 5.3 – необхідна конфігурація файлу .env.prod.db

Даний файл містить в собі дані користувача бази даних та саму назву бази даних.

Дані маніпуляції необхідні лише для первинного налаштування для додатку який буде встановлюватися. Після створення та налаштування файлів від користувача все що буде необхідно це виконати в командному терміналі команду: “docker-compose -f docker-compose.yml up -d –build”.

В результаті її успішного виконання буде створено та запущено у фоновому процесі базу даних PostgreSQL, серверний додаток Gunicorn для більш продуктивного зв’язку з фреймворком Flask та проксі-сервер Nginx.

5.2. Користувацький інтерфейс

Користувацький інтерфейс включає в себе все, з чим взаємодіє користувач при роботі з додатком. Тому його побудова є одним із важливіших кроків у розробці програмного продукту. Так як під час його проектування виявляються слабкі та сильні сторони інтерфейсу, визначається його ефективність, недоліки.

В нашому випадку наш додаток матиме головну сторінку(рисунок 5.4).

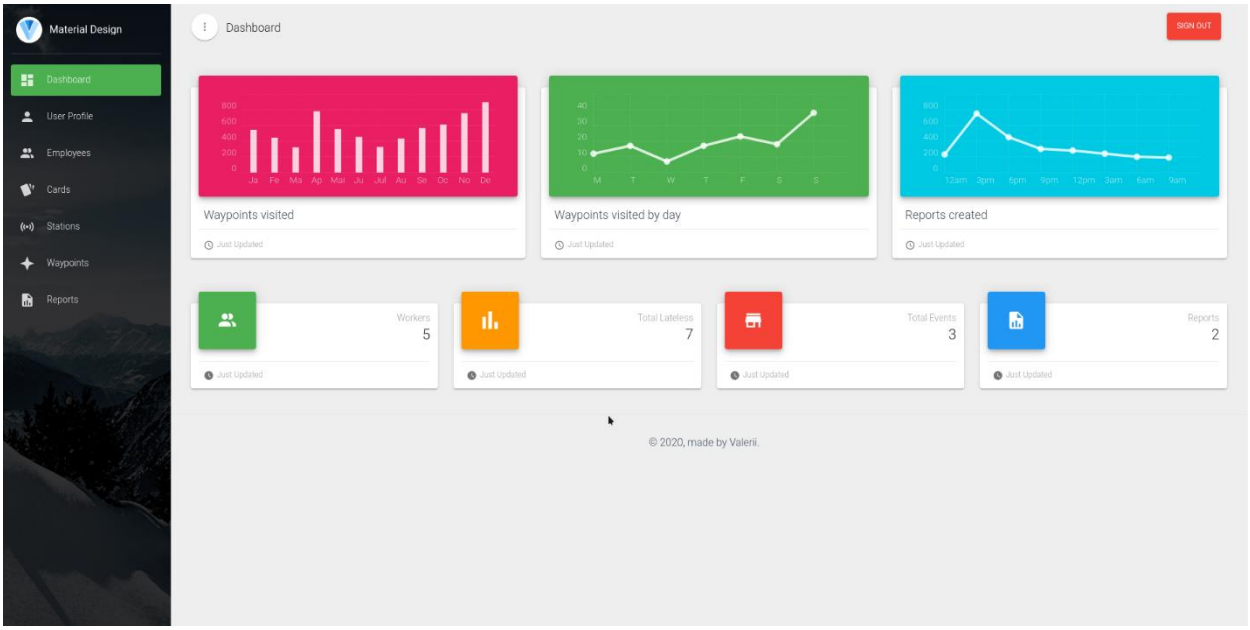

Рисунок 5.4 – головна сторінка клієнтського додатку

На даній сторінці адміністратор проекту матиме змогу спостерігати в реальному часі за статистикою запізнень, кількості звітів, кількість незвичайних подій. На діаграмах зверху можна відстежувати статистику по місяцям, дням тижнів по кількості створених звітів, кількість відвіданих пунктів.

На наступному малюнку користувачу буде представлено окрему сторінку для редагування власних даних(рисунок 5.5).

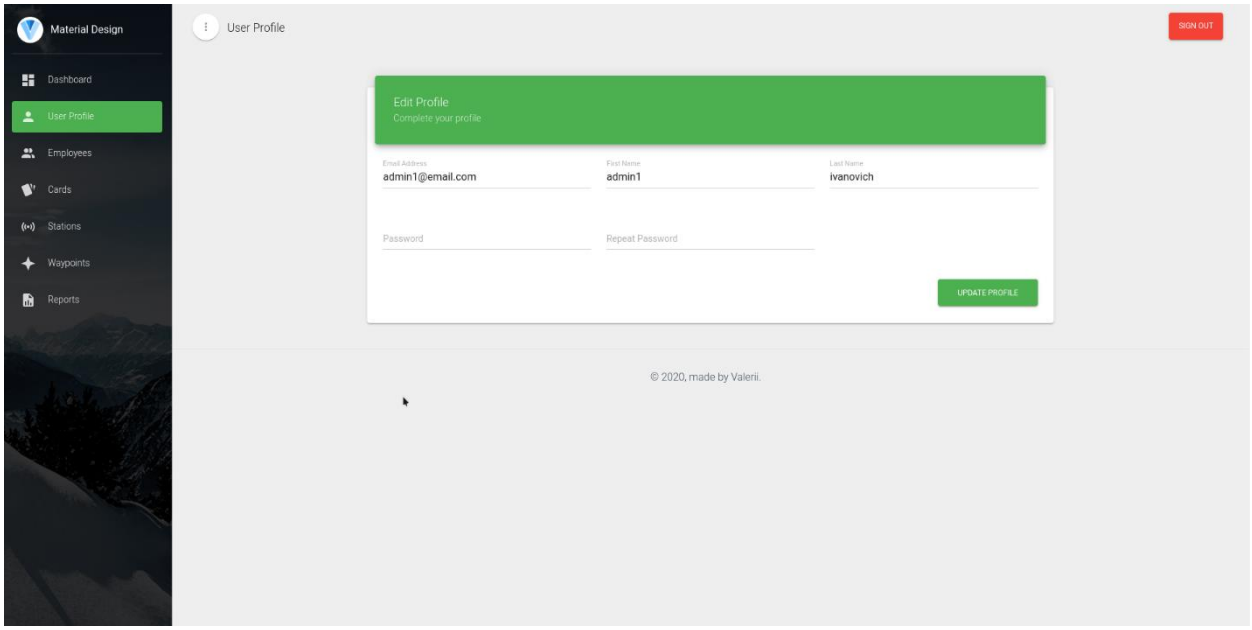

Рисунок 5.5 – сторінка редагування даних адміністратора

Адміністратор системи матиме змогу редагувати власні дані, такі як поштова скринька, його ім’я та прізвище, а також пароль та його підтвердження. Для відвідання даної сторінки та редагування власних даних користувач повинен знаходитися в системі, бути ідентифікованим.

Сторінка користувача повинна бути присутня так як у разі збільшення кількості адміністраторів системи у них виникатиме потреба у зміні власних даних. Також однією з причин додавання функцій зміни паролю є потреба у зв’язку з проблемами безпеки додатків що мають доступ до інтернету.

Наступний малюнок демонструє загальний вигляд бази даних робітників(рисунок 5.6).

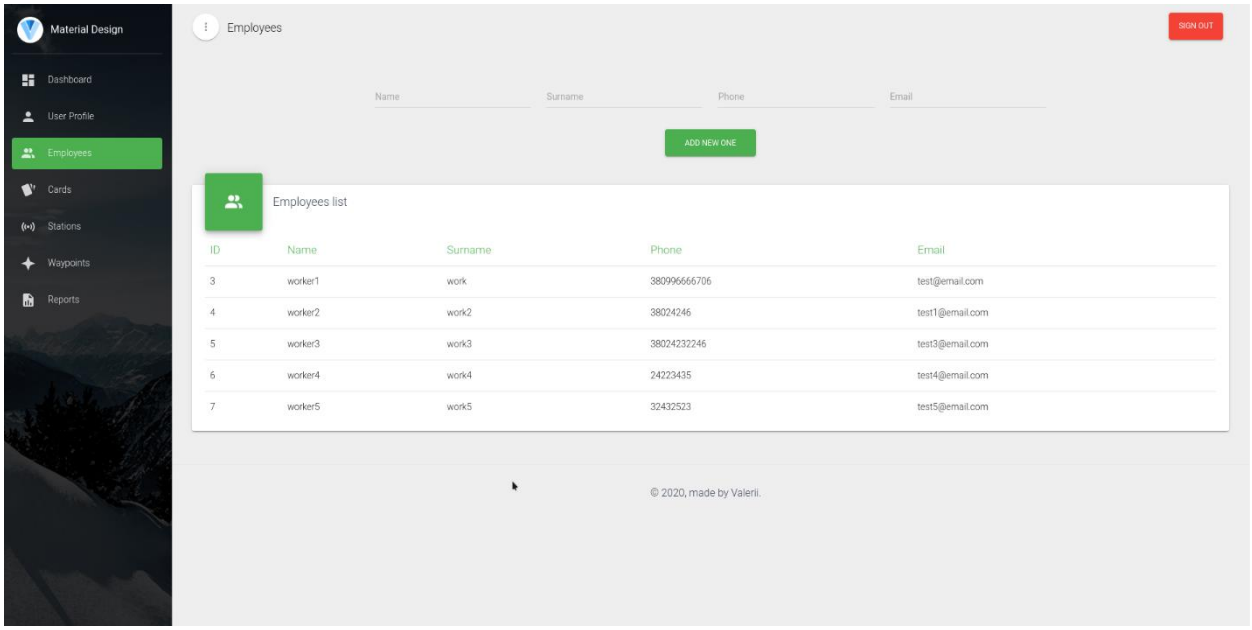

Рисунок 5.6 – сторінка редагування даних адміністратора

На даній сторінці адміністратор системи має змогу відстежувати робітників системи в реальному часі, додавати нових в базу даних за допомогою елементів форм у верхній частині сторінці. Таблиця відображає унікальний ідентифікатор робітника, його ім'я та прізвище, номер телефону та його поштову скриньку яка в майбутньому може бути використана для відправки попереджень або ж особистих звітів. У випадку масштабування системи, її ріст, матиме сенс розподілити адміністраторів за групами контролю. Кожна з груп відповідатиме за окремі дії системи. Таким чином і для контролю за робітниками можна буде виділити окремий вид групи адміністраторів які займалися б лише персоналом.

Для повноцінного користування системою робітниками необхідно додавати унікальні карти-доступу до окремих пунктів охоронної системи. Сторінка управління картами доступу наведена на малюнку нижче(рисунок 5.7).

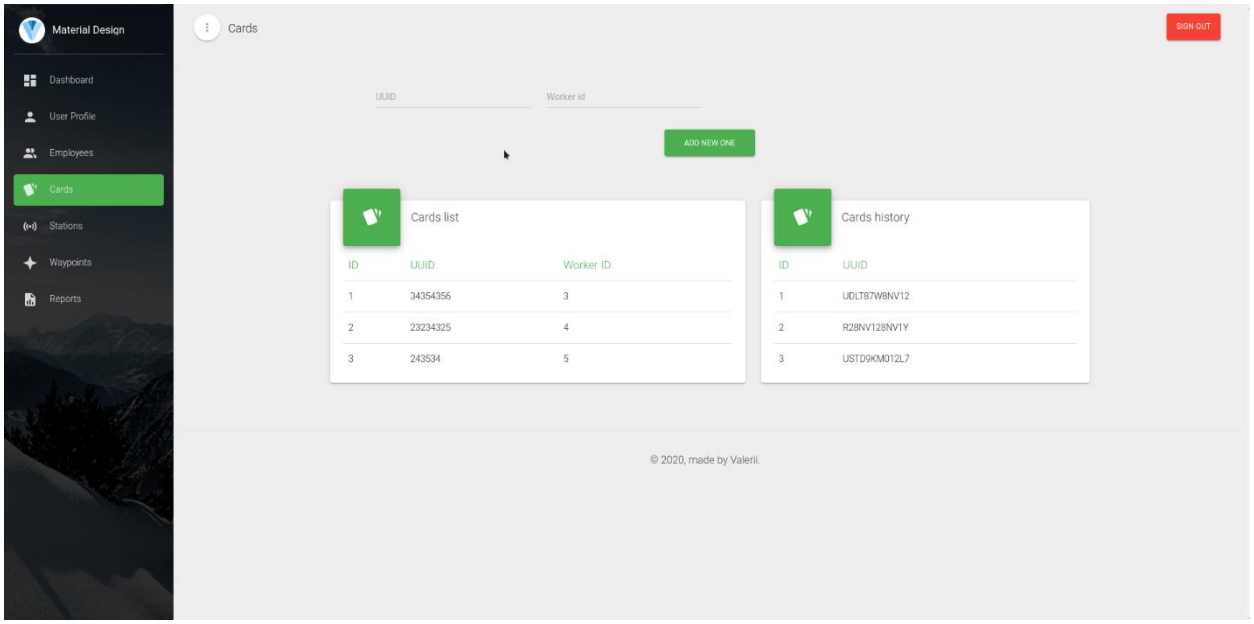

Рисунок 5.7 – сторінка управління картами-доступу

Дана сторінка відображає дві таблиці з даними. Перша зліва таблиця відображає вже додані карти-доступу до користувачів з їх унікальним ідентифікатором. Інша таблиця призначена для відображання загальної історії користування картами, таким чином якщо в системі буде знайдена невідома нова карта-доступу — її дані можна буде скопіювати да додати до певного робітника. Таким чином на одній сторінці адміністратор може керувати картами-доступу, додавати їх до системи та одночасно призначати робітників без них до окремих карт-доступу.

Наступна сторінка необхідна для додання до системи пунктів які являють собою апаратний додаток з власним адресом серверу(рисунок 5.8).

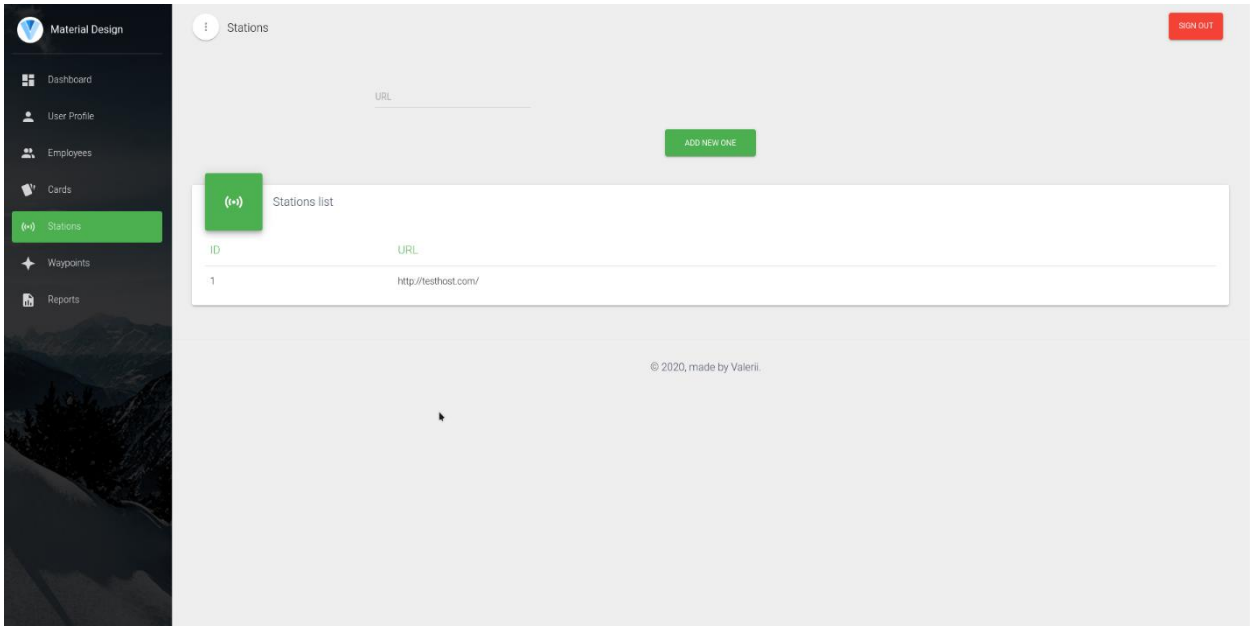

Рисунок 5.8 – сторінка управління картами-доступу

На даній сторінці адміністратору доступні функції додавання адрес пунктів на яких встановлений апаратний додаток, а також перегляд вже доданих пунктів. Дані адрес беруться із конфігурації http-серверу апаратного додатку на якому від буде встановлений. В такому випадку навіть один апаратний додаток може доданий до системи але з різними точками входу до системи.

На наступній сторінці представлено таблицю відвідувань робітниками пунктів яка оновлюється в реальному часі або у випадку її відвідування адміністратором системи(рисунок 5.9).

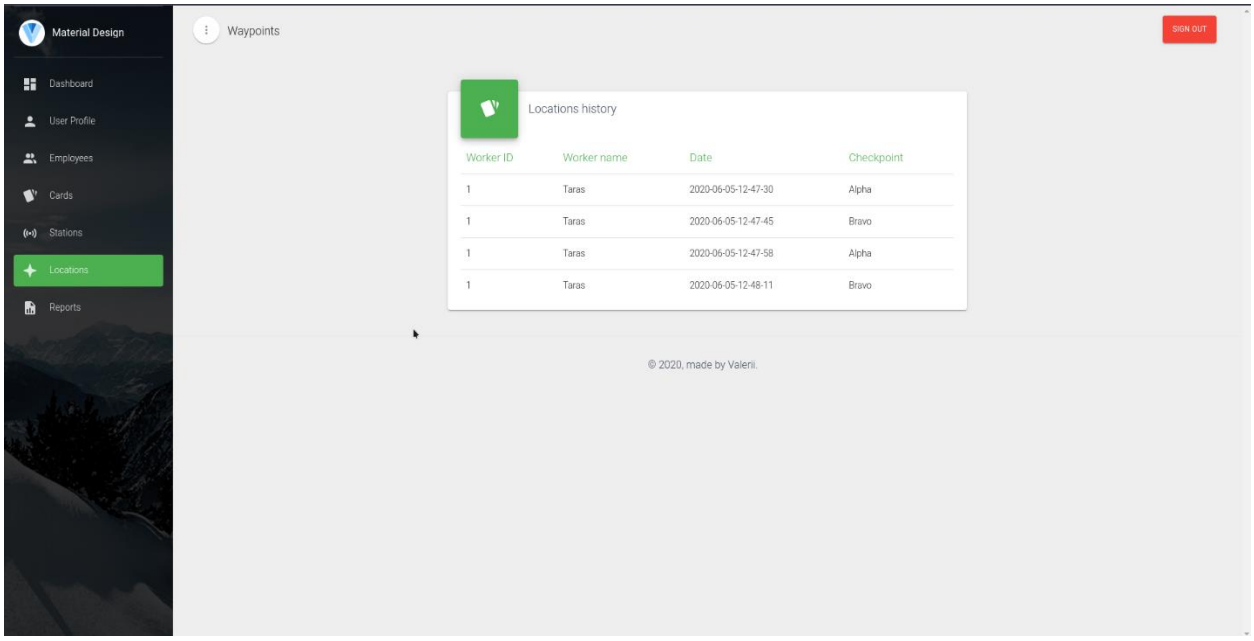

Рисунок 5.9 – сторінка відвідувань пунктів робітниками

На даній сторінці адміністратор системи має змогу відстежувати рух робітників у системі по раніше доданим пунктам. Відстеження часу дозволяє орієнтуватися за поточним часом щоб зрозуміти де робітник був востаннє. В даному випадку додаток налаштований на відображення історії відвідувань у порядку від першого до останнього, але для адміністратора системи налаштовується й відображення історії в зворотньому порядку.

Остання сторінка являє собою представлення бази даних звітів системи(рисунок 5.10).

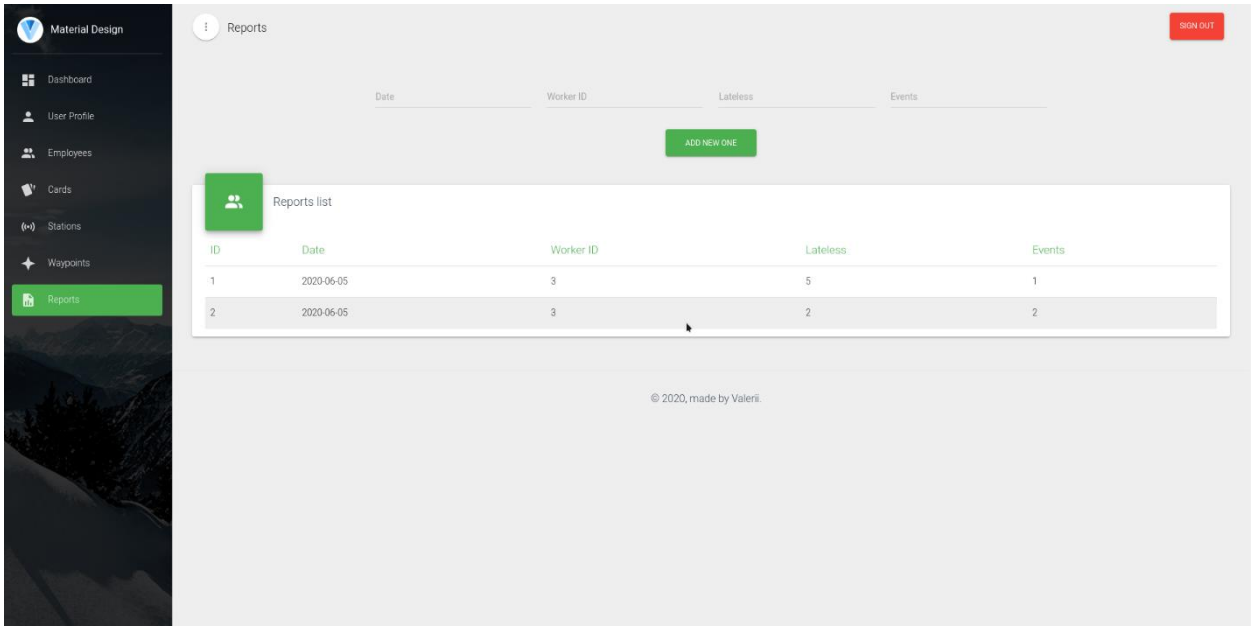

Рисунок 5.10 – сторінка звітів системи

За теперішнім фунціоналом адміністратор системи має змогу сам додавати звіти у випадку якщо це необхідно. Представлення звітів являє собою звіт по даті коли зроблений звіт, ідентифікатор робітника по якому він генерується, кількість його запізнень та загальна кількість незвичайних випадків. У випадку налаштування автоматичної генерації звітів по кожному робітнику системи потрібно налаштувати умови в разі виконання яких звіт буде створений автоматично. Але це потребує більш ретельного підходу кожного підприємства до задачі яка їх цікавить.

Заключною сторінкою клієнтського додатку є сторінка входу до системи(рисунок 5.11).

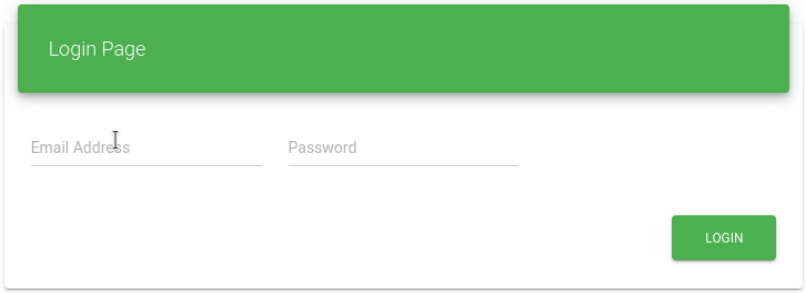

Рисунок 5.11 – сторінка входу до системи

На даній сторінці адміністратор системи має змогу увійти до системи по раніше відомим для нього даним.

Таким чином для побудови клієнтського додатку використовувалися сучасні методи та підходи до його розробки, а також сучасний стиль графічного дизайну Material Design. Інтерфейс інтуїтивно зрозумілий для звичайного користувача і лаконічно представлений.

ВИСНОВКИ

В ході виконання даної роботи було виконано поставлені задачі, а саме:

1. Був проведений поглиблений аналіз існуючих рішень по предметній області в результаті чого не було знайдено аналогів програмних систем подібних до теми дипломної роботи.
2. Був розроблений клієнтський додаток з інтуїтивно зрозумілим інтерфейсом користувача-адміністратора системи.
3. Було запроектовано концептуально схему бази даних та представлено в записці до дипломної роботи.
4. Розроблено серверний додаток з використанням мови програмування Python на базі фреймворку Flask.
5. В процесі розробки додатків було налагоджено взаємодію між додатками, серверний додаток зв’язаний з базою даних та апаратним додатком. Клієнтський додаток зв’язаний з серверним додатком.
6. Реалізовано процес аутентифікації користувачів-адміністраторів у системі за допомогою клієнтського додатку.
7. Для адміністратору системи розроблено функціонал по редагуванню таблиць бази даних.
8. Клієнтський додаток був спроектований таким чином щоб користувачу було максимально зрозуміло призначення елементів графічного інтерфейсу для покращення роботи нових користувачів системи.
9. Для адміністратору системи було спроектовано та розроблено функціонал ідентифікації співробітників за допомогою системи карт, що працюють за методом радіочастотної ідентифікації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Интерфейс. Основы проектирования взаимодействия / К.Носсел, Д. Кронин, Р. Рейман, А. Купер. – Санкт-Петербург: Питер, 2016. – 720 с. – (Питер).

2. NodeMCU Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://nodemcu.readthedocs.io/en/master/.

3. Docker Docs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/.

4. Чакон С. Git для профессионального программиста / Скотт Чакон. – Санкт-Петербург: Питер, 2016. – 496 с. – (Питер).

5. Эспозито Д. Разработка современных веб-приложений: анализ предметных областей и технологий / Дино Эспозито. – Москва: Вильямс, 2017. – 464 с. – (Вильямс).

6. Elman J. Lightweight Django: Using REST, WebSockets, and Backbone 1st Edition / Julia Elman. – Себастопол: O'Reilly Media, 2014. – 246 с. – (O'Reilly Media).

7. Хэнчетт Э. Vue.js в действии / Э. Хэнчетт, Б. Листоун. – Санкт-Петербург: Питер, 2019. – 304 с. – (Питер).

8. Vue.js Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/v2/guide/.

9. SQLAlchemy 1.3 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.sqlalchemy.org/en/13/.

10. Flask-Migrate extension [Електронний ресурс] – Режим доступу до ресурсу: https://flask-migrate.readthedocs.io/en/latest/.

11. МакГрат М. Программирование на Python для начинающих / Майк МакГрат. – Москва: ООО Эксмо, 2015. – 189 с. – (ООО Эксмо).

ДОДАТОК 1

Програмно – апаратний комплекс моніторингу та управління активностями охоронної системи

СПЕЦИФІКАЦІЯ

УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТВ6146_20Б

Аркушів 2

Київ 2020

Позначення	Найменування	Примітки
Документація		
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТВ6146_20Б 81-1	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТВ6146_20Б 12-1	docker-compose.yml	Основний файл конфігурації-запуску додатків
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТВ6146_20Б 12-2	WorkersView.py	Один із основних компонентів представлення серверного додатку
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТВ6146_20Б 12-3	WorkersModel.py	Один із основних компонентів відображення серверного додатку
УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТВ6146_20Б 12-4	Employees.vue	Один із основних компонентів клієнтського додатку

ДОДАТОК 2

Програмно – апаратний комплекс моніторингу та управління активностями охоронної системи

ТЕКСТ ПРОГРАМНОГО МОДУЛЮ

УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТВ6146_20Б

Аркушів 9

Київ 2020

Текст серверного додатку програмно – апаратного комплексу моніторингу та управління активностями охоронної системи

Текст ініціалізуючого модулю серверного додатку.

```
from flask import Flask
from flask_cors import CORS
from .models import db, bcrypt
from .views.CardsView import card_api as card_blueprint
from .views.AdminView import admin_api as admin_blueprint
from .views.PointsView import point_api as point_blueprint#
from .views.ReportsView import report_api as report_blueprint
from .views.WaypointsView import waypoint_api as waypoint_blueprint
from .views.WorkersView import worker_api as worker_blueprint
from .views.LocationsView import location_api as location_blueprint


def create_app():
  """
  Create app
  """

  # app initiliazation
  app = Flask(__name__)
  cors = CORS(app)

  app.config.from_object("src.config.Config")

  # initializing bcrypt and db
  bcrypt.init_app(app)
  db.init_app(app)

  app.register_blueprint(card_blueprint, url_prefix='/api/v1/cards')
  app.register_blueprint(admin_blueprint, url_prefix='/api/v1/admins')
  app.register_blueprint(point_blueprint, url_prefix='/api/v1/point')
  app.register_blueprint(report_blueprint, url_prefix='/api/v1/report')
  app.register_blueprint(waypoint_blueprint, url_prefix='/api/v1/waypoints')
  app.register_blueprint(worker_blueprint, url_prefix='/api/v1/workers')
  app.register_blueprint(location_blueprint, url_prefix='/api/v1/locations')


  @app.route('/', methods=['GET'])
  def index():
    return 'Null endpoint'

  return app
```

Текст представлення точки входу для керування робітниками.

```
from flask import request, json, Response, Blueprint, g
from ..models import WorkerModel, WorkerSchema
from ..shared.Authentication import Auth

worker_api = Blueprint('workers', __name__)
workers_schema = WorkerSchema()


@worker_api.route('/', methods=['POST'])
def create():
    """
    Create a single worker
    """
    req_data = request.get_json()
    data = workers_schema.load(req_data)

    worker_in_db = WorkerModel.get_worker_by_email(data.get('email'))
    if worker_in_db:
        message = {'error': 'worker already exist, please supply another email address'}
        return custom_responce(message, 400)


    worker = WorkerModel(data)
    worker.save()
    print(worker)
    ser_data = workers_schema.dump(worker)

    return custom_responce(ser_data, 201)


@worker_api.route('/', methods=['GET'])
#@Auth.auth_required
def get_all():
    """
    Get all workers
    """
    workers = WorkerModel.get_all_workers()
    ser_workers = workers_schema.dump(workers, many=True)
    return custom_responce(ser_workers, 200)

@worker_api.route('/<int:worker_id>', methods=['GET'])
#@Auth.auth_required
def get_a_worker(worker_id):
    """
    Get a single worker
    """
    worker = WorkerModel.get_one_worker(worker_id)
    if not worker:
        return custom_responce({'error': 'worker not found'}, 404)
```

```
    ser_worker = workers_schema.dump(worker)
    return custom_responce(ser_worker, 200)

@worker_api.route('/me', methods=['PUT'])
#@Auth.auth_required
def update():
    """
    Update me
    """
    req_data = request.get_json()
    data = workers_schema.load(req_data, partial=True)

    worker = WorkerModel.get_one_worker(g.worker.get('id'))
    worker.update(data)
    ser_worker = workers_schema.dump(worker)
    return custom_responce(ser_worker, 200)

@worker_api.route('/me', methods=['DELETE'])
#@Auth.auth_required
def delete():
    """
    Delete a worker
    """
    worker = WorkerModel.get_one_worker(g.worker.get('id'))
    worker.delete()
    return custom_responce({'message': 'deleted'}, 204)#add responce


def custom_responce(res, status_code):
    return Response(
        mimetype="application/json",
        response=json.dumps(res),
        status=status_code
    )
```

Текст моделі керування робітниками.

```
from .UserModel import UserModel
from marshmallow import fields, Schema


class WorkerModel(UserModel):
    #miss db.Model inheritence
    __tablename__ = "workers"


    @staticmethod
    def get_all_workers():
        return WorkerModel.query.all()
```

```
    @staticmethod
    def get_one_worker(id):
        return WorkerModel.query.get(id)

    @staticmethod
    def get_worker_by_email(value):
        return WorkerModel.query.filter_by(email=value).first()

class WorkerSchema(Schema):
    id = fields.Int(dump_only=True)
    name = fields.Str(required=True)
    surname = fields.Str(required=True)
    phone = fields.Str(required=True)
    email = fields.Email(required=True)
    #password = fields.Str(required=True, load_only=True)

from . import db, bcrypt

class UserModel(db.Model):
    __abstract__ = True #check for error if missing
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(128), nullable=False)
    surname = db.Column(db.String(128), nullable=False)
    phone = db.Column(db.String(128), unique=True)#auth for both
    email = db.Column(db.String(128), unique=True, nullable=False)
    password = db.Column(db.String(128))

    def __init__(self, data):
        self.name = data.get('name')
        self.surname = data.get('surname')
        self.phone = data.get('phone')
        self.email = data.get('email')
        #self.password = self.__generate_hash(data.get('password'))

    def save(self):
        db.session.add(self)
        db.session.commit()

    def delete(self):
        db.session.delete(self)
        db.session.commit()

    def update(self, data):
        for key, item in data.items():
            if key == 'password':
                self.password = self.__generate_hash(item)
                setattr(self, key, self.password)
            else:
                setattr(self, key, item)
        db.session.commit()

    def __generate_hash(self, password):
```

```
        return bcrypt.generate_password_hash(password, rounds=10).decode("utf-8")

    def check_hash(self, password):
        return bcrypt.check_password_hash(self.password, password)

    def __repr__(self):#repl?
        return '<id {}>'.format(self.id)
```

Текст представлення відображення сторінки робітників клієнтського додатку.

```
<template>
  <v-container
    id="regular-tables"
    fluid
    tag="section"
  >
    <!-- <base-v-component
      heading="Employees list"
      link="components/simple-tables"
    /> -->
    <v-row justify="center">
      <v-col
        cols="12"
        md="8"
      >
        <v-container class="py-0">
          <v-row>
            <v-col
              cols="12"
              md="3"
            >
              <v-text-field
                v-model="newItem.name"
                label="Name"
                class="purple-input"
              />
            </v-col>
            <v-col
              cols="12"
              md="3"
            >
              <v-text-field
                v-model="newItem.surname"
                label="Surname"
                class="purple-input"
              />
            </v-col>
            <v-col
              cols="12"
              md="3"
            >
```

```
          <v-text-field
            v-model="newItem.phone"
            label="Phone"
            class="purple-input"
          />
        </v-col>
        <v-col
          cols="12"
          md="3"
        >
          <v-text-field
            v-model="newItem.email"
            label="Email"
            class="purple-input"
          />
        </v-col>
      </v-row>
      <v-row justify="center">
        <v-btn
          color="success"
          class="mr-0"
          @click="addNewOne"
        >
          Add new one
        </v-btn>
      </v-row>
    </v-container>
  </v-col>
</v-row>
<base-material-card
  icon="mdi-account-multiple"
  title="Employees list"
  class="px-5 py-3"
>
  <v-simple-table>
    <thead>
      <tr>
        <th class="primary--text">
          ID
        </th>
        <th class="primary--text">
          Name
        </th>
        <th class="primary--text">
          Surname
        </th>
        <th class="primary--text">
          Phone
        </th>
        <th class="primary--text">
          Email
        </th>
```

```
        </tr>
      </thead>

      <tbody>
        <tr
          v-for="item in employees"
          :key="item.id"
        >
          <td>{{ item.id }}</td>
          <td>{{ item.name }}</td>
          <td>{{ item.surname }}</td>
          <td>{{ item.phone }}</td>
          <td>{{ item.email }}</td>
        </tr>
        <!-- <tr>
          <td>1</td>
          <td>Dakota Rice</td>
          <td>Niger</td>
          <td>Oud-Turnhout</td>
          <td class="text-right">
            $36,738
          </td>
        </tr> -->
      </tbody>
    </v-simple-table>
  </base-material-card>

  <div class="py-3" />
 </v-container>
</template>
<script>
 import axios from 'axios'
 export default {
  data () {
   return {
    editedIndex: -1,
    newItem: {
     name: '',
     surname: '',
     phone: '',
     email: '',
    },
    defaultItem: {
     name: '',
     surname: '',
     phone: '',
     email: '',
    },
    employees: [],
    id: null,
    name: null,
    surname: null,
```

```
      phone: null,
      email: null,
    }
  },
  mounted:
    function () {
      axios
        .get('http://localhost:5000/api/v1/workers/')
        .then(responce => {
          this.employees = responce.data
        })
        .catch(function (e) {
          console.log(e)
        })
    },
  methods: {
    close () {
      this.$nextTick(() => {
        this.newItem = Object.assign({}, this.defaultItem)
        this.editedIndex = -1
      })
    },
    addNewOne () {
      axios
        .post('http://localhost:5000/api/v1/workers/', this.newItem)
        .catch(function (e) {
          console.log(e)
        })
      this.employees.push(this.newItem)
      this.close()
    },
  },
}
</script>
```

ДОДАТОК З

Програмно – апаратний комплекс моніторингу та управління активностями охоронної системи

ОПИС ПРОГРАМНОГО МОДУЛЮ

УКР.КПІім.ІгоряСікорського_ТЕФ_АПЕПС_ТВ6146_20Б

Аркушів 9

Київ 2020

АНОТАЦІЯ

Додаток містить опис компонентів клієнтського та серверного додатків системи, що розроблена для моніторингу та управління активностями охоронної системи. Створені компоненти реалізують функціонал який був поставлений в задачах дипломного проекту. Наведені вище компоненти реалізують такі функції:

— зберігання даних про робітників системі в базі даних;

— додавання та видалення вже існуючих робітників;

— взаємодія клієнтського додатку з серверним, обмін даними;

Клієнтський додаток отримує дані через протокол HTTP відправляючи запити до серверу. Отримавши дані клієнтський додаток виводить дані на графічному інтерфейсі веб-додатку реалізованого за допомогою фреймворку Vue.js.

При розробці обох програмних компонентів використовувалась мова Python та Javascript у середовищі розробки Visual Studio Code.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис опис компонентів клієнтського та серверного додатків системи, що розроблена для моніторингу та управління активностями охоронної системи. У додатку 2 міститься програмний код даних модулів.

Програмний продукт працює в будь-якій операційній системі (Windows, Linux та MacOS) і потребує встановленого на ПК додатку Docker та для клієнтського додатку встановленого сучасного браузеру.

При розробці обох програмних компонентів використовувалась мова Python та Javascript у середовищі розробки Visual Studio Code.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблені компоненти виконують завдання по зберіганню даних про робітників системи в базі даних. Клієнтський модуль дозволяє взаємодіяти з сервером шляхом відправки HTTP запитів. Серверний додаток виконує функцію збереження даних шляхом взаємодії з сервером бази даних.

Дані модулі з легкістю можна використати для формування інших подібних модулів додатку, що реалізується.

.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Основним з інструментів для створення додатків був фреймворк Vue.js для клієнтського додатку та фреймворк Flask для серверного додатку.

Обидва додатки написані на мовах програмування Javascript та Python відповідно.

Також до проекту підключаються зовнішні залежності, які управляються менеджером залежностей npm та pip для клієнтського та серверного додатків.

.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для забезпечення повноцінної роботи та покращення ефективності проектування та розробки проекту а в подальшому і демонстрації роботи у рамках реалізації поставлених цілей було обрано середовище розробки Visual Studio Code яке показало себе з найкращої сторони так як на сьогоднішній час являється одним з найгнучкіших середовищ розробки.

Програмний продукт працює в будь-якій операційній системі (Windows, Linux та MacOS) і потребує встановленого на ПК додатку Docker та для клієнтського додатку встановленого сучасного браузеру.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблені додатки для своєї інсталяції не потребують додаткового втручання та ручне налаштування програм в операційній системі. Лиш необхідно мати встановлене програмне забезпечення Docker та запустити в директорії проекту команду “docker-compose -f docker-compose.yml up -d –build”.

Результатом виконання команди буде запуск клієнтського, серверного, додатків а також додатку бази даних та проксі-серверу.

.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для розроблених додатків буде інформація яка може зчитуватися як з допомогою клієнтського додатку за спеціально заповненими полями так і за спеціальними запитами до серверного додатку із заповненими полями у форматі JSON.

Вихідними даними для клієнтського додатку буде вивід обробленої інформації у відповідні поля графічного інтерфейсу. Для серверного додатку вихідними даними буде код успішного виконання запиту та наступний вивід вже обробленої інформації сервером.