Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ІНФОРМАЦІЙНА СИСТЕМА МОНІТОРИНГУ ТА АНАЛІЗУ ДАНИХ КОРПОРАТИВНИХ РЕСУРСІВ

INFORMATION SYSTEM FOR MONITORING AND ANALYZING DATA OF CORPORATE RESOURCES

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-42
Коваль Віталій Сергійович

(підпис)

Керівник:
к.т.н., доцент
Христинець Наталія Анатоліївна

(підпис)

Кваліфікаційну роботу
допущено до захисту
« 11 » червня 2024 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
« 10 » 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Ковалю ВіталіюСергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Інформаційна система моніторингу та аналізу даних корпоративних ресурсів*

Керівник роботи *к.т.н., доцент Христинець Наталія Анатоліївна*

затверджені наказом закладу вищої освіти від «30» грудня 2023 року № 459/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 11.06.2024р.

3. Вихідні дані до роботи *Методички та книги на тему алгоритмів та програмування, публікації про архітектурні концепції та шаблони програмування, інтернет-ресурси з різних джерел на тему авіації, документація з мов програмування та CMS Drupal*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Аналіз предметної області
Засоби та інструменти CMS Drupal
Розробка та реалізація інтерфейсу для моніторингу та аналізу даних
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Аналітичні дослідження з питань розробок
Технології CMS Drupal та прикладні програмні інтерфейси
Архітектура інформаційної системи для аналізу та моніторингу даних
Інтерфейс інформаційної системи
Схема роботи програмнго продукту

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Христинець Н.А., доцент*		
Засоби та інструменти CMS Drupal	*Христинець Н.А., доцент*		
Розробка та реалізація інтерфейсу для моніторингу та аналізу даних	*Христинець Н.А., доцент*		
Нормоконтроль	*Багнюк Н.В., доцент*		
Гарант ОП	*Лавренчук С.В., доцент*		
Показник запозичень тексту	***___%***		
Академічна доброчесність	*Міскевич О.І.,асистент*		

7. Дата видачі завдання 10.01.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Розділ 1. Аналіз предметної області*	до 15.02.2024 р.	Виконано
2.	*Розділ 2. Засоби та інструменти CMS Drupal*	до 15.03.2024 р.	Виконано
3.	*Розділ 3. Розробка та реалізація інтерфейсу для моніторингу та аналізу даних*	до 04.05.2024 р.	Виконано
4.	*Висновки та пропозиції*	до 07.05.2025 р.	Виконано
5.	*Формування списку використаних джерел*	до 10.05.2024 р.	Виконано
6.	*Формування додатків*	до 15.05.2024 р.	Виконано
7.	*Оформлення ілюстративного матеріалу*	до 20.05.2024 р.	Виконано
8.	*Нормоконтроль*	до 01.06.2024 р.	Виконано
9.	*Інструментальна перевірка на академічний плагіат*	до 04.06.2024 р	Виконано
10.	*Представлення кваліфікаційної роботи бакалавра до захисту*	до 11.06.2024 р.	Виконано

Здобувач вищої освіти _____ Коваль В.С.
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ Христинець Н.А.
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Коваль В.С. Інформаційна система моніторингу та аналізу даних корпоративних ресурсів. Рукопис.

Кваліфікаційна робота магістра (бакалавра) ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2024.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, додатків.

У першому розділі проведено аналіз предметної області, а саме: аналіз існуючої проблеми, складено порівняльну характеристику інструментів для розробки інформаційної системи та здійснено обґрунтований вибір технологій. Проаналізовано визначення API, та надано характеристику. Складено архітектуру інтерфейсу для інформаційної системи.

У другому розділі розглянуто загальні методики розробки додатків на основі використання модульної системи керування вмістом Drupal. Розглянуто такі інструменти як: бібліотека Jquery, шаблонізатор Twig, спеціальна мова стилю сторінок CSS та роботу Drupal з стороніми API.

Третій розділ присвячено опису розробки та реалізації інтерфейсу, та функціонал моніторингу та аналізу даних. Наведено зображення програмних інтерфейсів платформи та додано приклади написання коду для функціоналу моніторингу та аналізу даних.

Ключові слова: система, інтерфейс, дані, корпоративні ресурси, авіація, система керування вмістом, друпал, база даних, веб-розробка, програмний інтерфейс.

ANNOTATION

Koval V.S. Information system for monitoring and analyzing data of corporate resources. Manuscript.

Qualification work for master's (bachelor's) degree in Computer Engineering, specialty 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2024.

Qualification work consists of an introduction, three sections, conclusions, a references, five appendices.

The first chapter analyzes the subject area, namely: an analysis of the existing problem, a comparative description of tools for developing an information system, and a reasonable choice of technologies. The API definition is analyzed and characterized. The architecture of the interface for the information system is drawn up.

The second section discusses general methods of application development based on the use of the modular content management system Drupal. Such tools as the Jquery library, the Twig templating engine, the special CSS page style language, and Drupal's work with third-party APIs are considered.

The third section describes the development and implementation of the interface, as well as the functionality of monitoring and data analysis. Images of the platform's program interfaces are provided and examples of writing code for the monitoring and data analysis functionality are included.

Keywords: system, interface, data, corporate resources, aviation, content management system, drupal, database, web development, software interface.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі аналіз даних та їх моніторинг є невід'ємною частиною. Тому, розробка інформаційної системи для моніторингу та аналізу даних є досить актуальною темою, як для спостереження, так і для дослідження активностей польотів, прийняття рішень для бізнесу та урядових установ, аналізу пасажиропотоків, покращення ефективності роботи аеропортів тощо.

Стан вивченості проблеми. Проблема правильного та швидкого аналізу та моніторингу даних у різних сферах. В основному, дослідженнями авіаційних сполучень займаються інжеренери в галузі літакобудування. Програмування веб-інтерфейсів для таких досліджень потребує знань в галузі комп'ютерної інженерії.

Метою кваліфікаційної роботи є розробка веб-інтерфейсу для моніторингу та аналізу даних авіаційних корпорацій.

Об'єктом дослідження є дані моніторингу авіаційних компаній.

Предметом дослідження є технології інструменти системи керування вмістом Drupal для побудови веб-інтерфейсу моніторингу корпоративних ресурсів.

Для досягнення мети було поставлено наступні завдання:

– проаналізувати технології для моніторингу корпоративних даних та визначити систему керування вмістом та шаблонізатор для побудови веб-інтерфейсу;

– дослідити інтерфейси та аргументувати вибір одного з них для отримання даних з віддалених сховищ;

– визначити інструменти для роботи з локальним середовищем проектування та вибрати мову програмування для побудови функціоналу, а також утиліти для забезпечення продуктивності роботи веб-інтерфейсу;

– розробити архітектуру функціоналу на основі вибраних інструментів та програмно спроектувати моніторинг та аналіз ресурсів;

– протестувати роботу веб-інтерфейсу і забезпечити його злагоджену роботу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз питань обробки даних авіаційної індустрії

У світі, насиченому величезним обсягом інформації, корпорації стикаються з безпрецедентними викликами щодо збору, зберігання, обробки та аналізу даних. Розвиток технологій і цифровізація супроводжується зростанням обсягу даних, що вимагає від компаній не лише зберігання цих даних, а й їхню ефективну обробку та використання для прийняття стратегічних рішень. В наш час важливіше не саме зберігання даних, а їх швидке оновлення. Погодьтеся, всі хочуть бачити найсвіжіші дані, наприклад, новини. Те саме відноситься і до корпоративних ресурсів – постійне оновлення даних дає змогу тій чи іншій корпорації зростати, та бути актуальними для людей. Застарілі і непроаналізовані дані призводять до неактуальності того чи іншого ресурсу. Уявіть, ми дізнались новину від сусіда, а через два дні, ми читаємо цю ж новину на якомусь ресурсі. Такий ресурс буде відвідувати менше і менше людей, а в кінцевому результаті, він впаде в пошукових системах Google, і люди забудуть, що таке джерело взагалі існувало.

В цьому випадку, прийнято рішення аналізувати і моніторити дані з корпоративно-авіаційних ресурсів. Авіаційна сфера стрімко росте, з кожним роком, люди все більше пересуваються саме літаками. Ще 100 років тому, ніхто уявити не міг, що з Парижу до Мадриду можна дібратись за кілька годин по небу, проте, це є реальність. На жаль, є багато проблем, наприклад, не завжди люди можуть отримувати актуальну інформацію про аеропорти, авіалінії, рейси та перельоти, ще кілька років тому, щоб дізнатись розклад рейсів, потрібно було їхати в аеропорт, або моніторити спеціальні розклади, які оновлювались раз в день.

Авіаційна індустрія – складна система, яка об’єднує різні аспекти, такі як перельоти, аеропорти, літаки, рейси та пасажирські потоки. За допомогою сучасних технологій, таких як системи GPS, супутникове зображення та сенсори,

збираються величезні обсяги даних про пересування літаків та діяльність аеропортів. Однак, існують деякі значні проблеми, які стають перешкодою для ефективного аналізу цих даних.

Однією з ключових проблем у сфері аналізу даних авіаційної індустрії є обмежена доступність даних. Багато даних про рух літаків, перельоти та діяльність аеропортів є комерційною власністю компаній і можуть бути недоступні для загального використання. Це ускладнює аналіз та дослідження в цій галузі, оскільки дослідники та аналітики мають обмежений доступ до даних. Одна з цілей – отримувати свіжі дані, аналізувати їх та подавати інформацію публічно. Щоб, будь яка людина, могла відстежувати рейси та перельоти літаків в реальному часі.

Проте, є і інші проблеми, наприклад – неоднорідність даних. Ця проблема може виникати через різні формати збереження даних, використання різних систем або неповноту даних. Наприклад, інформація про рейси цілком може бути представлена у різних форматах або різним шляхом, що ускладнює їхній аналіз. Тому, одна з цілей – отримувати дані і конвертувати їх в один формат, це значно полегшує подальшу роботу з цими даними та аналізу.

Напевно, одна з головних проблем – точність даних. Точність даних також є ключовим моментом в аналізі та моніторингу даних авіаційної індустрії. Навігаційні дані, такі як координати GPS, можуть містити помилки або неточності, що може призвести до похибок в аналізі даних. І таким чином, призвести до некоректного пересування літаків на гугл-мапі в реальному часі. Головна ціль – отримувати найсвіжі дані з надійного ресурсу через API. Використовуючи такий підхід, буде надано точні дані для користувачів про перельоти або рейси.

Також, багато хто вважає, що аналіз авіаційних даних може призвести до проблем конфідеційності та безпеки даних. Що нібито збір та обробка даних про пересування літаків може стати об’єктом атаки для тих, хто шукає можливості для кібератак або навіть терористичних актів. Забезпечення конфіденційності та безпеки цих даних є критичним завданням для забезпечення безпеки авіаційної

системи в цілому. Проте, хочу зазначити, що авіаційні дані надаються безкоштовно багатьма ресурсами, такими як FlightRadar, а також більше даних надаються іншими API, але за певні кошти. Тому, якщо і хакери захочуть отримати ці дані, вони це зроблять. В цьому випадку, якісь захищені дані не зберігаються і не обробляються. Тут отримуються публічні дані, потім йде правильний аналіз та моніторинг, щоб полегшити життя людей, які зацікавлені в подорожах, та і в загальному в авіаційній сфері.

Тому, основне завдання, це розробити інтерфейс, який буде виводити проаналізовані дані, та моніторити певні об'єми даних, які оновлюються майже щосекунди. Таким чином, відбувається трансформація конкретних закодованих даних у вигляді масивів, в об'єкти візуалізації, які є зрозумілими для простих користувачів. Проте, є одна проблема – не можуть бути проаналізовані абсолютно всі дані, причина дуже проста – нестача ресурсів для такої задачі. Для аналізу та моніторингу великих об'ємів даних, потрібні величезні і швидкі сервера. Тому, будуть отримуватись данні в якомусь конкретному контексті, наприклад, дані авіаліній Нідерландів, та моніторинг рейсів цих авіаліній. Таким чином, люди які проживають в Нідерландах і хочуть кудись полетіти, або ж, навпаки, люди які хочуть відвідати Нідерланди, можуть скористатись цим інтерфейсом, та відслідковувати ті чи інші рейси.

Узагальнюючи, існує безліч проблем у сфері аналізу даних авіаційної індустрії, які постійно еволюціонують і вимагають уваги та інноваційних підходів для їх вирішення.

1.2 Обгрунтування вибору технологій та інструментів для моніторингу та аналізу корпоративних даних

На даний момент, технології дуже стрімко розвиваються, щоб розробити, наприклад, простий інтерфейс, може бути використаний простий конструктор сайтів. Проте, в цьому випадку, це не просто інтерфейс, а складна платформа, яка здатна отримувати, аналізувати, обробляти та моніторити дані. Задача –

поєднати зручний інтерфейс з складною логікою на «Back-end», і при цьому зведення до мінімуму використання додаткових технологій, та використання ресурсів на сервері. Для таких цілей найкраще підходять CMS системи, їх не мало, але потрібна надійна, перевірена роками і швидка CMS. Однією з таких, або, можливо, навіть і єдина, це – Drupal CMS, основні характеристики якого наступні:

– гнучкість у налаштуванні: Drupal – це гнучка система управління контентом, яка може бути легко налаштована для відповідності конкретним потребам в аналізі та моніторингу даних. Вона дозволяє створювати різноманітні типи контенту, включаючи форми для введення даних та звіти для візуалізації результатів аналізу, також працює і з клієнт частиною, що дозволяє динамічно змінювати візуалізацію тих чи інших даних;

– безпека: Drupal відомий своїм надзвичайно високим рівнем безпеки, що являється критично важливим для зберігання, моніторингу та обробки конфіденційної інформації, яка може бути зібрана в процесі аналізу та моніторингу даних;

– інтеграція з іншими системами: Drupal легко інтегрується з іншими системами, такими як бази даних, сервіси веб-аналітики, CRM-системи та інші інструменти, що дозволяє об’єднати дані з різних джерел для комплексного аналізу та моніторингу.

Саме це і потрібно, оскільки, платформа буде працювати із зовнішніми системами і API для отримання тих чи інших авіаційних даних. Крім цього, Drupal дозволяє робити безпечні запити до зовнішніх ресурсів, за допомогою сторонніх модулів та основного функціоналу. Платформа буде робити дуже багато запитів щоденно, і навіть щохвилинно, для отримання нових даних.

Як відомо, щоб щось отримати, потрібно щось попросити. В даному випадку «попросити» це і є GET запит, який отримає найсвіжіші дані. Drupal є open-source CMS (рис. 1.1), що надає можливість постійно підтримувати та розвивати цю CMS, а також втілювати нові ідеї, що в свою чергу означає, що ця CMS ніколи не втратить свою цінність, і завжди буде актуальною. Всі ці критерії

є вагомим аргументом, чому повинна бути використана Drupal CMS в цьому випадку. Такі критерії як: безпека, надійність, підтримка, швидкість, гнучкість та актуальність, дозволяють побудувати те що потрібно. Це лише фундамент, базуючись на цій системі розроблено інтефейс для моніторингу та аналізу даних, проте, задіяні додаткові інструменти та технології для повної побудови якісного продукту. Для безпечного та якісного виконання запитів, використаний REST API. Для цього завантажено базовий модуль який надає Drupal – RESTful Web Services API. Тобто, це основа, але потребує додаткових налаштувань. Саме через REST API отримуються всі авіаційні дані.

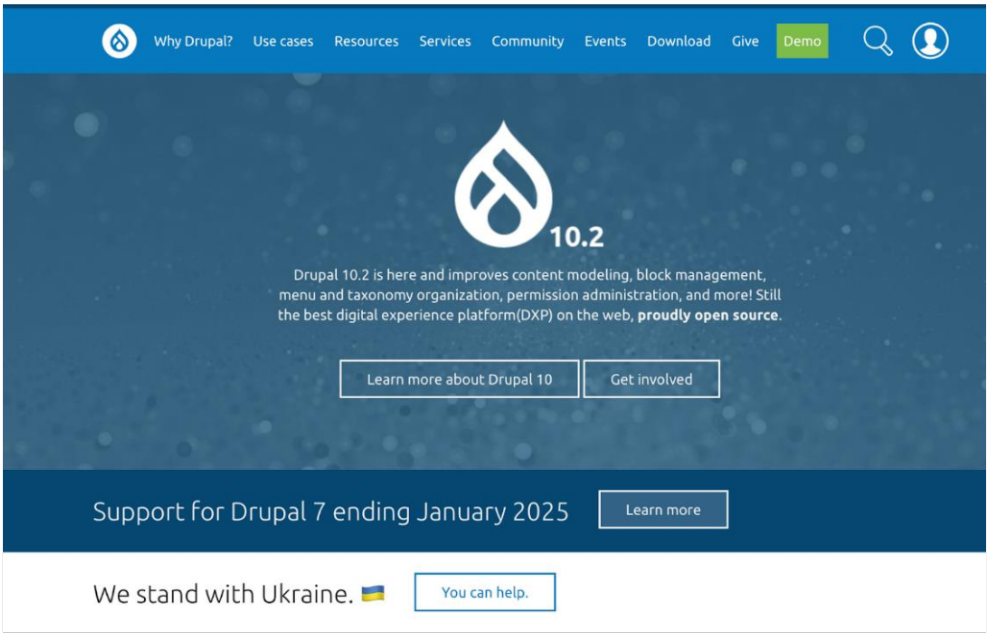

Рисунок 1.1 – Офіційна сторінка Drupal CMS на d.org [1]

Rest API надає гнучкий спосіб для отримання даних, в цьому випадку передача і отримання даних відбувається поступово виконувавши певні кроки, які потрібні для того, щоб ці дані прийшли цілі, в повному об’ємі та швидко.

В загальному, Rest API відповідає за надання актуальних даних. В цьому випадку, задача полягає в тому, щоб отримувати різні об’єми даних паралельно, а саме: авіалінії та їх структури, аеропорти, дані рейсів літаків, їхні координати, тощо. Прооблена така маніпуляція з даними (рис. 1.2). Для візуалізації

моніторингу даних, використана доволі стара, але надійна JavaScript бібліотека – Jquery, адже, друпал уже підлаштований під неї.

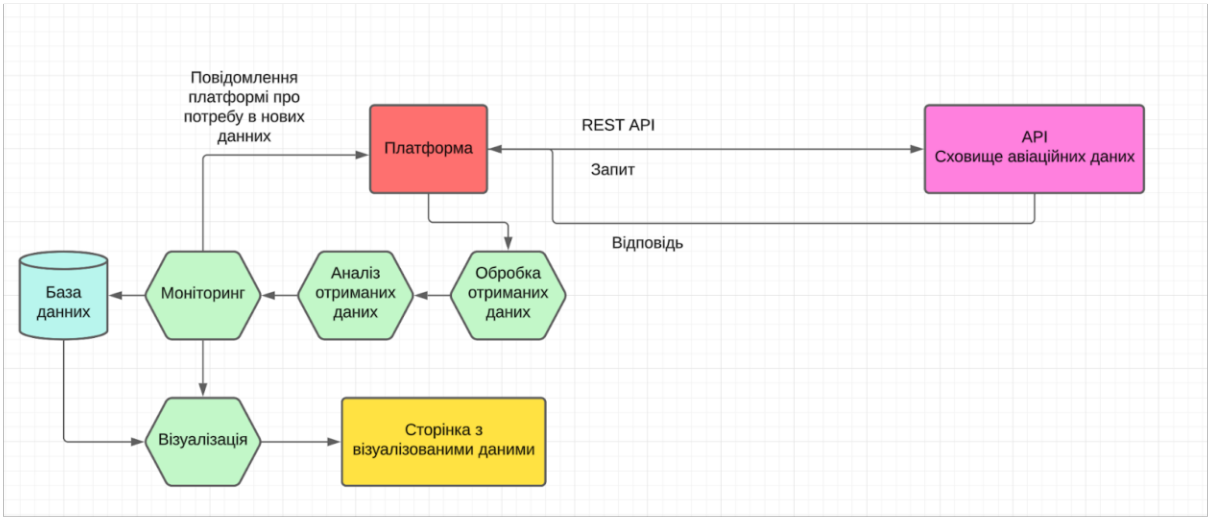

Рисунок 1.2 – Цикл роботи з даними

Так історично склалось, що з самого початку, тобто, заснування самої CMS, потрібно було додавати якісь прості інтерактивні функції, в той час, не було потужних фреймворків як зараз. Саме тому, і по цей день використовується бібліотека Jquery. В цьому випадку, написано код саме на основі цієї бібліотеки, який відповідає за динамічне переміщення літаків в реальному часі згідно координатів. Таким чином, навіть без оновлення сторінки, відбувається візуалізація моніторингу даних. В наш час це важливо, адже користувачі звикли спостерігати за чимось, не докладавиши додаткових зусиль.

Зараз є багато сайтів, на яких користувачі мають змогу спостерігати рейси літаків в реальному часі, наприклад, «FlightRadar» (рис. 1.3). Це досить потужні моніторингові системи, але через велику кількість асинхронних дій, браузер користувача може просто закриватись, або підвисати. Це відбувається тому, що кожний літак, це окрема асинхронна функція, і окремий запит на оновлення даних, в цьому випадку це – координати. Тут це було передбачено, і саме тому, користувач зможе фільтрувати рейси по авіалініям. Таким чином, на карті в один і той же момент, не буде сотні літаків, а тому і картинка і візуалізація є якіснішою і швидшою, яку браузер зможе відтворити без жодних проблем. Також, було

передбачено швидкість інтернету користувача, якщо вона буде повільною, то і візуалізація літаків буде плавною.

Рисунок 1.3 – FlightRadar 24 з моніторингом всіх польотів [2]

Для відображення супутникової гугл-мапи, використана бібліотека Leaflet, яка буде оновлювати карту світу кожного дня, згідно нових супутникових даних. Таким чином, надано гарантії користувачам про актуальність геоданих аеропортів та міст.

Drupal працює з мовою програмування PHP, тому аналіз та обробка даних написана за допомогою коду PHP. Дані отримуються у форматі JSON, що є актуально, адже мові програмування PHP комфортно працювати з форматами такого роду. Оскільки, ядро цієї мови надає безліч функцій та методів, для роботи з об’ємними масивами даних, і це в свою чергу забезпечує швидкість та надійність обробки цих даних. Це означає, що не прийдеться писати багато коду, що може призвести до похибок при маніпуляції з даними. Потрібно віддати належне засновникам PHP, які кожного року стараються покращити цю мову.

Після аналізу і обробки даних, потрібно тимчасово їх десь зберігати. Drupal працює з системою керування реляційними базами даних – MySql. Коли будуть отримуватись нові дані, і утворюватись нові об’єкти на основі цих даних, наприклад, нова статична авіалінія або тимчасовий об’єкт рейсу, ці дії будуть сприйматись як інструкції для Drupal, які він повинен виконати. Ці інструкції

передаються до бази даних через спеціальний шар програмного забезпечення, який відомий як «драйвер бази даних». Коли MySQL отримує ці інструкції, він обробляє їх відповідно до вказаних правил та зберігає дані у відповідній таблиці бази даних. Наприклад, дані про нову авіаланію можуть бути збережені у таблиці «airline», а деталі про перельоти – у таблиці flights. Коли йде звернення до веб-сайту, щоб переглянути сторінку всіх авіаланій Нідерландів, Drupal використовує MySQL, щоб отримати необхідні дані з бази даних. Він формує запит до MySQL, отримує відповідь та відображає вміст на веб-сайті. Отже, спілкування між Drupal і MySQL подібне до тісної співпраці: кожна сторона знає свої кроки, і разом вони створюють «коллобарацію», яку ми бачимо на екранах наших пристроїв. Ця спільна взаємодія між Drupal і MySQL забезпечує ефективну та надійну роботу веб-сайту, надаючи користувачам зручний та швидкий доступ до необхідного контенту. Тобто, база даних, є ключовим посередником між аналізом даних та їх візуалізацією.

Підсумовуючи – друпал буде використаний як основа для платформи, REST API відповідає за отримання та захист даних, мова програмування PHP аналізує та оброблює дані, MySql зберігає статичні та тимчасові дані, а JavaScript відповідає за їх моніторинг.

1.3 Порівняння різних технологій розробки для виконання поставленого завдання

Технології дуже стрімко розвиваються. З кожним роком з’являються якісь нові фреймворки, і навіть мови програмування. Проблема полягає в тому, що конкуренція дуже велика, і дуже рідко якась нова технологія може витіснити стару, але перевірену і стабільну технологію. Наприклад, на мові програмування PHP побудовано 80% всіх сайтів у світі. Проте світ адаптовується, і багато хто переробляє сайти зі старих технологій на нові, а дехто не може це робити, адже для цього потрібно дуже багато ресурсів, тому ці сайти лише підтримуються.

В основному сайти будуються на основі фреймворків або CMS. Перший варіант зустрічається набагато частіше. Фреймворків є багато, але варто зазначити найважливіші.

Найпопулярніший фреймворк мови програмування Python - Django. Даний фреймворк повністю диктує структуру проекту і призначається, щоб розробники не відволікалися на прості речі. Якщо коротко, то цей фреймворк надає якісь вже базові речі, і розробникам не доведеться витрачати на це час. Цей фреймворк використовується багатьма людьми, це стабільна річ, немає вразливих місць, є багато додаткових бібліотек які полегшать життя розробникам, та можуть бути використані для різноманітних потреб. Проте є і недоліки, наприклад, доволі високий поріг входження, навіть тоді коли запускається якийсь простий проект, наприклад який виводить тільки текст «Hello world», то через складну структуру використовується дуже багато ресурсів і пам’яті.

Прямий конкурент вищезгаданого фреймворку – Flask, проте це повна протилежність, основна мета – повна мінімізація. На перший погляд все здається простим і зрозумілим, але якщо потрібно написати складну бізнес-логіку, налаштувати базу даних, починаються проблеми: якщо є потреба зробити щось за допомогою Flask, потрібно робити все власноруч. Все через менш розвинену спільноту, тому багато документації, викладеної в мережі, або не працює, або є застарілою;

Фреймворк побудований на мові програмування PHP - Laravel, який використовується для створення сайтів будь-якої складності. Перш за все, це є плафторма з відкритим вихідним кодом, що значно скорочує час розробки. Також, цей фреймворк є одним з найбезпечніших фреймворків. Розробка сайтів на Laravel дуже популярна. Фреймворк використовує інтерфейс командного рядка Artisan, що в свою чергу значно прискорює процес розробки сайтів та додатків. Крім того, у фреймворку вбудована система аутентифікації, завдяки якій, Laravel однією командою створить стандартні функції, що використовуються майже на кожному сайті (наприклад, як сторінки входу або скидання пароля).

По-суті, ці фреймворки які відповідають за «Back-end» частину. Проте, для «Front-end» частини використовуються зовсім інші типи фреймворків, які побудовані на основі мови програмування JavaScript. Vue.js, React, Angular це три схожих між собою інструменти для розробки візуалізації інтерфейсів. Якщо говорити про моніторинг даних, то ці фреймворки за допомогою асинхронних методів JavaScript можуть постійно оновлювати ті чи інші дані. Якщо Django, це про аналіз даних і зберігання їх в базі даних, то JavaScript фреймфорки, це про візуалізацію та моніторинг даних. Наприклад, на офіційному сайті фонду «Повернись живим» є секція про звітність (рис. 1.4).

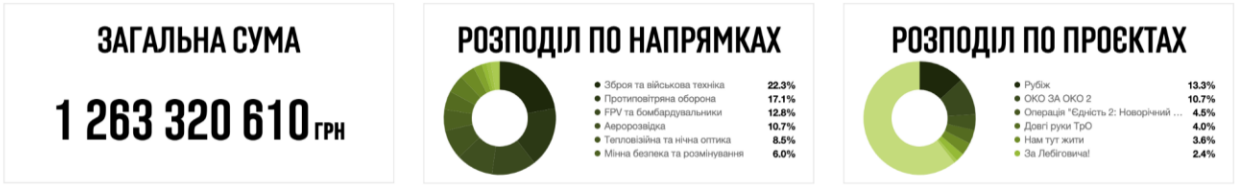

Рисунок 1.4 – Візуалізація проаналізованих данних про витрати [3]

Виходячи з рисунку 1.4 можна зазначити, що збір даних та їх аналіз виконує фреймворк для «Back-end» частини, а от виводу цих діаграм, та їх оновлення в реальному часі виконує «Front-end» фреймворк.

CMS – це технологія, яка значно відрізняється від фреймворків згаданих вище. По-перше, це комбінація двох типів фреймворків, тобто, ця CMS надає функціональність для візуалізації даних, аналізу та маніпуляцію з цими даними і зберігання них в базі данних. Joomla, Drupal, MODx та WordPress є одними з самих популярних CMS. З цих потрібно відзначити саме Drupal, на якому власне і побудовано інтерфейс для моніторингу та логіку аналізу даних, а також запити для отримання нових даних. Це є безкоштовна CMS, яка придатна для розробників і дуже зручна у використанні. Тобто, проста людина, яка не розуміється в програмуванні, може побудувати простий сайт за допомогою адміністративної панелі. Проте, за допомогою мови програмування PHP та модульної системи, можна розроблювати різний функціонал, починаючи від простої візитки і закінчуючи корпоративними порталами.

Очевидно, що ці всі технології можуть бути використані у аналізі та моніторингу даних. Проте, CMS є більш привабливим варіантом, оскільки, надає багато функціоналу одразу, надає інтерфейс для управління сайтом прямо з адмін-панелі, а також зберігає, аналізує і візуалізує дані одночасно.

1.4 Аналіз веб-інтерфейсів з аналізом і моніторингом даних

Зараз у світі є безліч різних різного роду сайтів, платформ з моніторингом та аналізом данних. Наприклад, одною з таких є Flashscore.

Flashscore – інформаційна система, яка збирає дані результатів всіх спортивних подій. Окрім того, що ця система архівує результати, вона ще моніторить події, які відбуваються в реальному часі. Працює ця система через різні веб-сокети та веб-хуки. Тобто, наприклад, коли якась команда забиває гол, спрацьовує веб-сокет, який надсилає оновлені дані спортивної події в якесь сховище даних, і за це відповідає спеціальне API. Flashscore підключене до цього API, тобто, підключене до сховища даних, які оновлюються через веб-сокет. В кінцевому результаті, на Flashscore встановлений веб-хук, який реагує на зміну даних на віддаленому середовищі, і потім ці дані аналізуються та передаються до клієнт-частини, яка оновлює рахунок спортивної події, за допомогою JavaScript, і це все відбувається в одній сесії, тобто, не потрібно перезавантажувати сторінку, щоб побачити оновлений результат.

Рисунок 1.5 демонструє те, як працюють веб-хуки і веб-сокети для оновлення даних. По-суті, запит на отримання даних відбувається тільки тоді, коли це дійсно потрібно, а інакше запити будуть відбуватись регулярно, але в більшості випадків, вони нічого не повертатимуть, бо рахунок змінюється кілька разів за матч, а не кожної хвилини.

Коротко, йде спортивна подія, якщо змінюється рахунок, йде передача даних до API сховища через веб-сокет, на оновлення даних у сховищі спрацьовує веб-хук, який витягує ці дані і перенаправляє на аналіз. Після аналізу даних на сервері, вони перенаправляються на обробку, тобто, змінюється формат даних,

щоб клієнт-частина змогла прийняти та візуалізувати. Отже, за допомогою візуалізації і відбувається моніторинг даних, тому що, наприклад, коли команда забила гол, потрібно змінити рахунок на таблі за допомогою методів JavaScript.

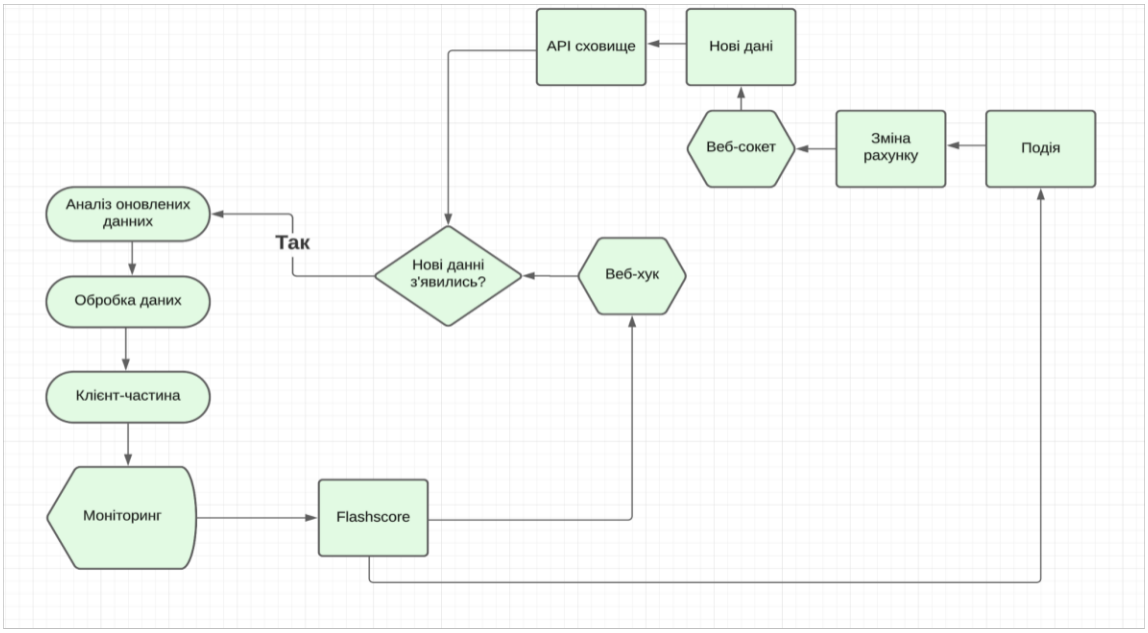

Рисунок 1.5 – Весь цикл отримання та аналізу даних для їх моніторингу

Проте, є і інший тип моніторингу даних, що отримує дані стабільно, в цьому випадку є гарантія того, що дані будуть нові і оновлені, і неважливо скільки пройшло часу. Якщо в попередньому випадку дані могли і не змінитись, оскільки рахунок матчу міг залишитись тим самим, наприклад, 0-0, то в цьому випадку, дані будуть оновлюватись постійно. Саме і цей тип моніторингу застосовано в дипломній роботі. Як і було зазначно раніше, побудовано моніторинг рейсів які відбуваються в реальному часі. Тобто, йде запит на отримання нових даних з різних корпоративних ресурсів кожних 5-7 секунд. Тут вже не потрібні веб-сокети і веб-хуки. Буде просто йти запит до API сховища для отримання нових данних, а потім ці дані будуть аналізуватись, оброблятись і відправлятись до клієнт-частини для моніторингу, як вказано раніше на рисунку 1.3.

1.5 API та його використання у цілях аналізу та моніторингу даних корпоративних ресурсів

В сучасному світі майже кожна програма, сайт або застосунок використовують якусь API, для покращення якості продукту, або для отримання специфічних даних. Наприклад, всім відомий сайт «Rozetka», використовує Stripe API, яке в свою чергу дозволяє розробникам інтегрувати функціональність онлайн-платежів у свої веб-сайти або додатки. Або ж, є ще Instagram API, яке дозволяє надсилати нові дописи на різні сайти. Відома популярна соціальна мережа Facebook, теж має своє API, але при цьому використовує і інші.

Інтерфейс прикладного програмування, або ж API, має чітко відрізнятися від інтерфейсу користувача. Інтерфейс користувача приймає дані від користувачів, передає їх додатку для обробки та повертає результати користувачеві. API не взаємодіє з користувачем, а обробляє дані, отримані від одного модуля програми, і передає результати назад в інший модуль. Ось як це відбувається – принцип роботи API зазвичай виражається через зв’язок запит-відповідь між клієнтом і сервером. Клієнт – це будь-яка зовнішня програма, з якою взаємодіє користувач. Сервер відповідає за серверну логіку та операції з базою даних. У цьому сценарії API працює як проміжний рівень між клієнтом і сервером, що дозволяє надсилати запити даних і відповіді. Це саме те, що потребує задача, адже для отримання нових даних з авіаційної сфери, потрібно робити запит-відповідь, майже кожної хвилини. В цьому випадку, клієнт, це інтерфейс платформи, адже користувач з нею взаємодіє.

API бази даних забезпечують зв’язок між програмою та системою керування базою даних. Розробники працюють із базами даних, пишучи запити для доступу до даних, змінення таблиць тощо. Ми будемо використовувати API цього типу, щоб отримувати статичні дані аеропортів, які розташовані в Нідерландах. Такі дані зберігаються в базах даних, оскільки вони є статичними та потребують додаткового захисту, оскільки це цивільна інфраструктура.

Віддалені API визначають стандарти взаємодії для програм, що працюють на різних машинах. Іншими словами, один програмний продукт отримує доступ до ресурсів, розташованих за межами пристрою, який їх запитує, що пояснює назву. Оскільки дві віддалені програми з’єднані через комунікаційну мережу, зокрема через Інтернет, більшість віддалених API написані на основі веб-стандартів.

В цій кваліфікаційній роботі використано два типи API. Для отримання статичних даних використано API сховищ. Проте, підключено і віддалені API, для отримання динамічних даних, таких як рейсів або авіаліній (рис. 1.6).

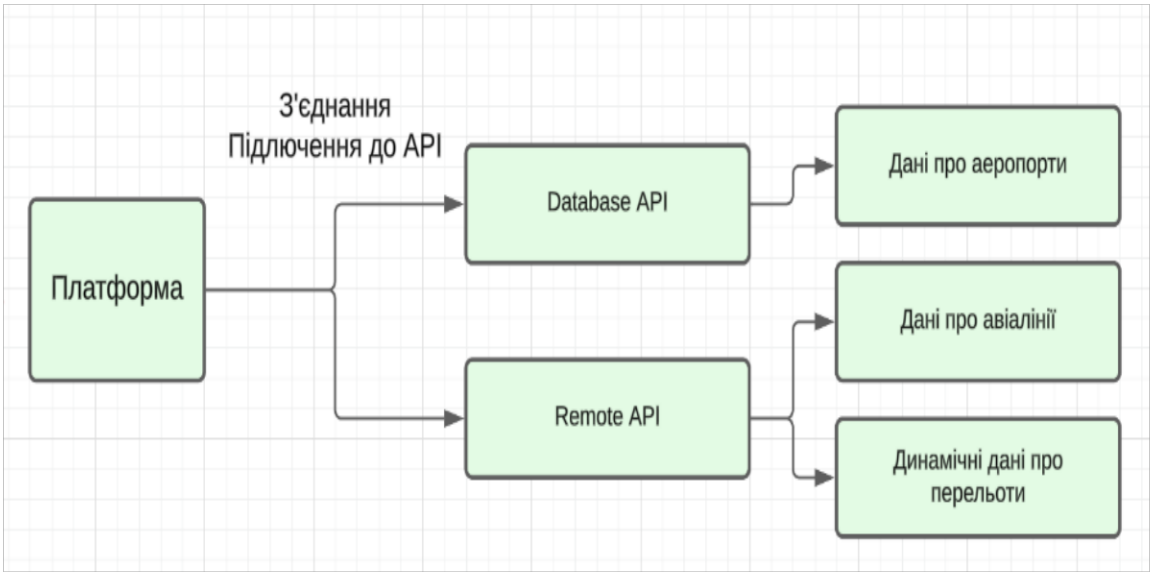

Рисунок 1.6 – Використання API на платформі розробки

API є ключовим моментом у аналізі і моніторингу корпоративних даних. Наприклад – AviationEdge, щоб відслідковувати координати польотів, які відбуваються в реальному часі, потрібно робити запит кожної секунди в якесь сховище, для отримання нових координатів літака. В даному випадку відбуваються запити до Remote API, куди також надходять дані, але вже від державних авіаційних інституцій (рис. 1.7). Можна отримувати різні дані, і робити різну кількість запитів, але при цьому потрібно заплатити певну частину коштів. На жаль, послуги більшості API не є безкоштовними, що є логічно.

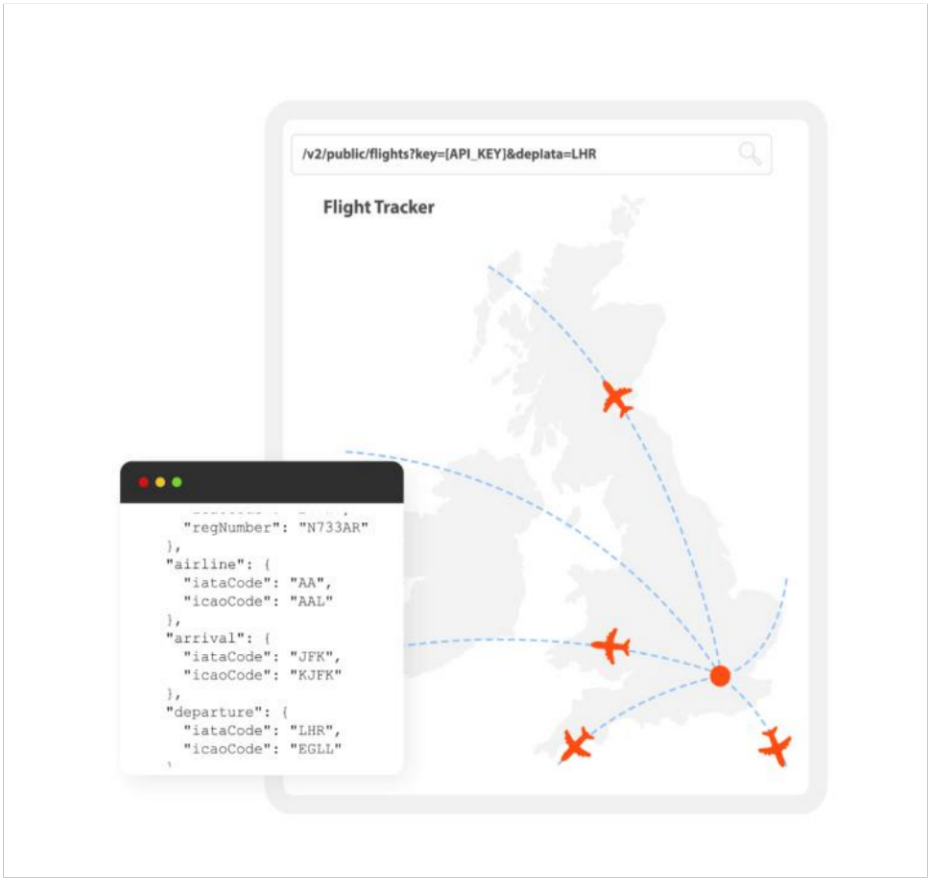

Рисунок 1.7 – AviationEdge API для отримання всіх авіаційних даних [4]

Підсумовуючи, можна зазначити, що API відіграє ключому роль в аналізі і моніторингу корпоративних даних. Адже, для того щоб щось аналізувати або робити моніторинг, потрібно мати основу, в цьому випадку це авіаційні дані, які і надходять з API.

1.6 Інструмент GitHub у контексті проекту

В сучасній розробці різного програмного забезпечення не обійтись без GitHub. GitHub – це справжній оазис для розробників, де кожен може знайти натхнення, навчатися та співпрацювати з однодумцями з усього світу. Це платформа, яка об’єднує мільйони розробників та команд для спільної роботи над проектами будь-якої складності. На Гітхабі кожен проект – це історія, яка наповнюється кодом, ідеями та співпрацею. Тут можна знайти все, починаючи від відкритих бібліотек та невеликих інструментів, і закінчуючи великими

відкритими проектами та корпоративними розробками. В цьому випадку, буде створено репозиторій для зберігання коду платформи, також слід зазначити, що буде зберігання не тільки коду, а й документації, тестів, статичних ресурсів, такі як іконки літаків, аеропортів і так далі. Є можливість створювати різні гілки для різних функцій, експериментувати з новими ідеями та об'єднувати зусилля через систему пул-реквестів. Якби цю платформу розробляло кілька розробників, тоді кожен член команди міг би зробити свій внесок та спостерігати за тим, як проект розвивається в реальному часі. А потім, коли розробка буде готовою до публікації, команда може використовувати можливості Гітхабу для автоматизації процесу випуску та розгортання. Це дозволяє швидко та ефективно поширювати їх продукт серед користувачів. Таким чином, Гітхаб – це не лише платформа для зберігання коду, але й вірний супутник для будь-якого розробника, який прагне зробити світ кращим через технології.

Мета дуже проста, це зберігати різні версії продукту на віддаленому сховищі, і у випадку якщо щось зламається, є можливість повернутись до старішої версії, на якій додаток працює корректно. Також, будуть зберігатись різні статичні ресурси в спеціальному сховищі на GitHub. Наприклад, іконки літаків, які будуть відображатись на гугл-мапі з моніторингом даних про перельоти. Або ж іконки аеропортів та авіаліній. Насправді, зберігання коду платформи на Гіт-хаб дає гарантію, що ця платформа зможе бути розгорнута на будь-якому середовищі, та використана багатьма людьми.

В цій кваліфікаційній роботі цей інструмент залучено не лише для зберігання коду, але і для зберігання статичних ресурсів на спеціальному сервері. Гітхаб надає безкоштовні сервери для зберігання даних, таким чином, будь-хто зможе отримати ці дані.

Загалом, Гітхаб є стабільною платформою для розробників усіх рівнів. Він сприяє співпраці, інноваціям та відкритості, створюючи місце, де кожен може знайти відгалуження свого шляху розвитку. Через можливості спільної роботи, відкритості, відданість вільному програмному забезпеченню, Гітхаб стимулює розвиток технологій та збільшує доступність інновацій для всіх.

1.7 Структура та вигляд інтерфейсу для аналізу і моніторингу даних

Розроблено інтерфейс для візуалізації проаналізованих даних та моніторингу. На кожному сайті повинна бути головна сторінка, адже це є візиткою. Окрім вигляду, вона повиина бути структуризована так, аби пошукові системи Google, могли її сканувати. На самому верху сторінки є меню, це є свого роду навігація по сайту. Зліва меню – логотип сайту. Нижче статичний банер, який символізує сферу авіації, це дає змогу користувачам краще зрозуміти, що це за інформаційна система. Здебільшого такі банери роблять для людей з вадами зору, а також для естетики сайту. Далі розташована секція з контентом, в цій секції є наступні компоненти: блок з поточними рейсами, з авіалініями, з аеропортами та з новинами. Внизу сторінки є футер, з меню та іншими деталями, які притаманні звичайним інформаційним системам. Футер є видимий на всіх сторінках. Також передбачено і адаптивну верстку, для того, щоб сайт працював на мобільних пристроях. В такому випадку, розташування компонентів змінюється залежно від розміру екрана користувача. В загальному схема головної сторінки виглядає згідно рисунку 1.8.

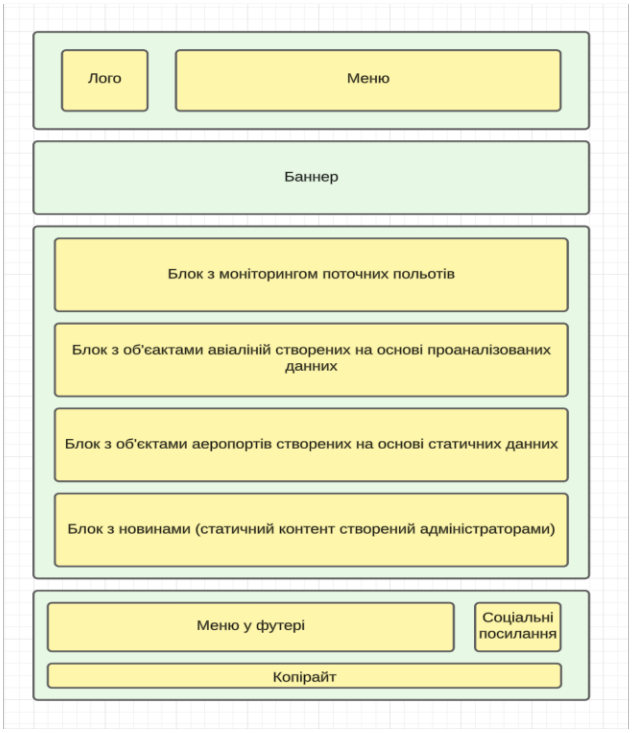

Рисунок 1.8 – Головна сторінка інтерфейсу

Важливо, щоб ці сторінки завантажувались і відображались швидко, тому потрібно максимально мінімізувати запити до бази даних, та використати правильну стратегію кешування. Схема інтерфейсу виглядає згідно рисунку 1.9.

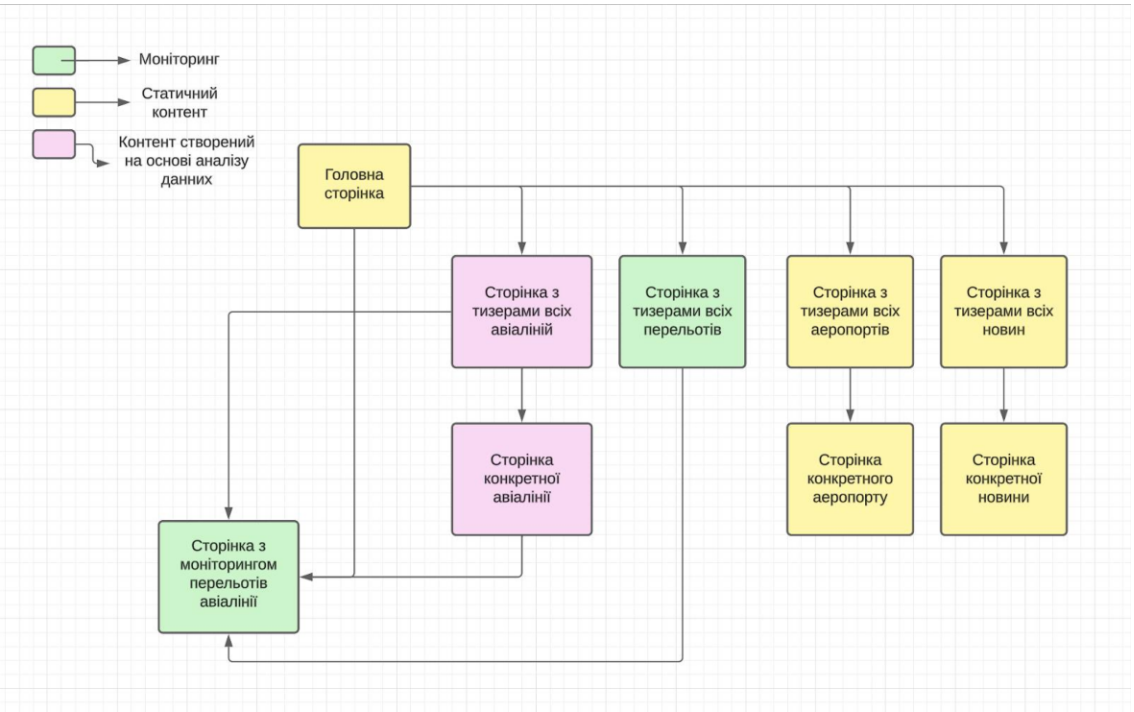

Рисунок 1.9 – Структура інтерфейсу з аналізом та моніторингом даних

1.8 Постановка завдання на кваліфікаційну роботу бакалавра

На основі аналізу проблем, можна поставити наступні практичні завдання на кваліфікаційну роботу:

– розгорнути чистий сайт для інтерфейсу на основі CMS Drupal;

– знайти та підключити API для отримання авіаційних даних;

– написати функціонал для аналізу даних використовуючи мову програмування PHP;

– написати функціонал для моніторингу даних використовуючи мову програмування JavaScript;

– розробити інтерфейс для візуалізації проаналізованих даних та моніторингу використовуючи внутрішні інструменти Drupal.

РОЗДІЛ 2
ЗАСОБИ ТА ІНСТРУМЕНТИ CMS DRUPAL

2.1 Вибір CMS Drupal

CMS Drupal – як і більшість систем керування вмістом є безкоштовною. Для того щоб користуватись друпалом, потрібно лише завантажити повну версію, та встановити її на сервері або локальному середовищі. Це може бути чисте ядро, тобто, система без додаткових компонентів та розширень, або ж готова збірка з набором стандартних модулів, які полегшують роботу в подальшому. Завантажувати CMS краще з офіційного сайту, але простіше встановлювати на хостингу через адмін-панель. У другому випадку не доведеться створювати базу даних та налаштовувати систему. Потрібно лише прикріпити доменне ім’я, зареєстроване заздалегідь, та вибрати його, щоб працювати з контентом. Двигун системи написаний мовою програмування PHP та використовує для своєї роботи бази даних MySQL та PostgreSQL.

Drupal має відкритий вихідний код, а це означає, що розвитком даної системи може займатися будь-хто, і це є одною з головних переваг друпалу над іншими системами, адже таким чином, вона буде постійно розвиватись і покращуватись.

Як і багато CMS, Drupal має можливість розширення функціональності завдяки установці додаткових модулів. Причому, можна використати вже існуючі модулі, або створити свої, за допомогою мови програмування PHP, тому в цій дипломній роботі створено кілька модулів які надають функціонал для аналізу та моніторингу даних. Ці модулі можна тримати у файловій системі, але при цьому не використовувати на сайті.

За 23 роки існування цієї системи, друпал нажив переваги і недоліки:

– безкоштовне використання, оскільки ця система є відкритою, це є однією з причин чому друпал так швидко став популярним;

– гнучкість, бо система надає можливість створити сайт будь-якої складності, що є вагомою перевагою у порівнянні з іншими безкоштовними

системами, тобто, на цій системі можна створити як і сайт-візитку, так і велику платформу для державної корпорації;

– швидкість, проте багато хто має стереотип, що друпал повільний, оскільки має досить об’ємну файлову систему, але він має вбудовану систему кешування, хоча інші CMS не мають такої переваги, адже для кешування потрібно встановлювати додаткові плагіни і модулі, які не завжди є безкоштовними;

– модулі, адже завдяки відкритому коду, розробник може знайти модуль для будь-якої потреби (рис. 2.1), крім того, він може зробити свій модуль, і викласти у свій профіль на офіційному сайті друпалу, це в свою чергу означає що інші люди можуть використати цей модуль на своїх сайтах.

– безпека, оскільки друпал має продуману систему захисту і з багатьма перевірками та модулями, які захищають сайт від різного роду хакерських атак.

Проте варто зазначити, що існують і суттєві недоліки:

– складність оновлення, оскільки більшість CMS можуть оновлювати модулі прямо з адміністративної панелі, варто лише нажати на кнопку оновлення, проте, Drupal не володіє такою властивістю, тому що, для оновлення модулів потрібно задіювати розробників і різні інструменти, наприклад, Composer та Github;

– вимогливість, адже Drupal вимагає великих ресурсів, тому розмістити сайт на самому дешевому хостингу не вийде;

– складний інтерфейс, через що новичкам дуже тяжко вивчити інтерфейс друпалу через велику кількість різних функцій.

Після встановлення Drupal, є вже стандартний дизайн пустого сайту, в цьому є одна з переваг. За це відповідають різні теми, в кожній темі є свій дизайн, і можна перемкнути тему прямо з адмін-панелі. Як і у випадку з модулями є можливість завантажувати додаткові теми з офіційного сайту CMS Drupal та використовувати їх сайті.

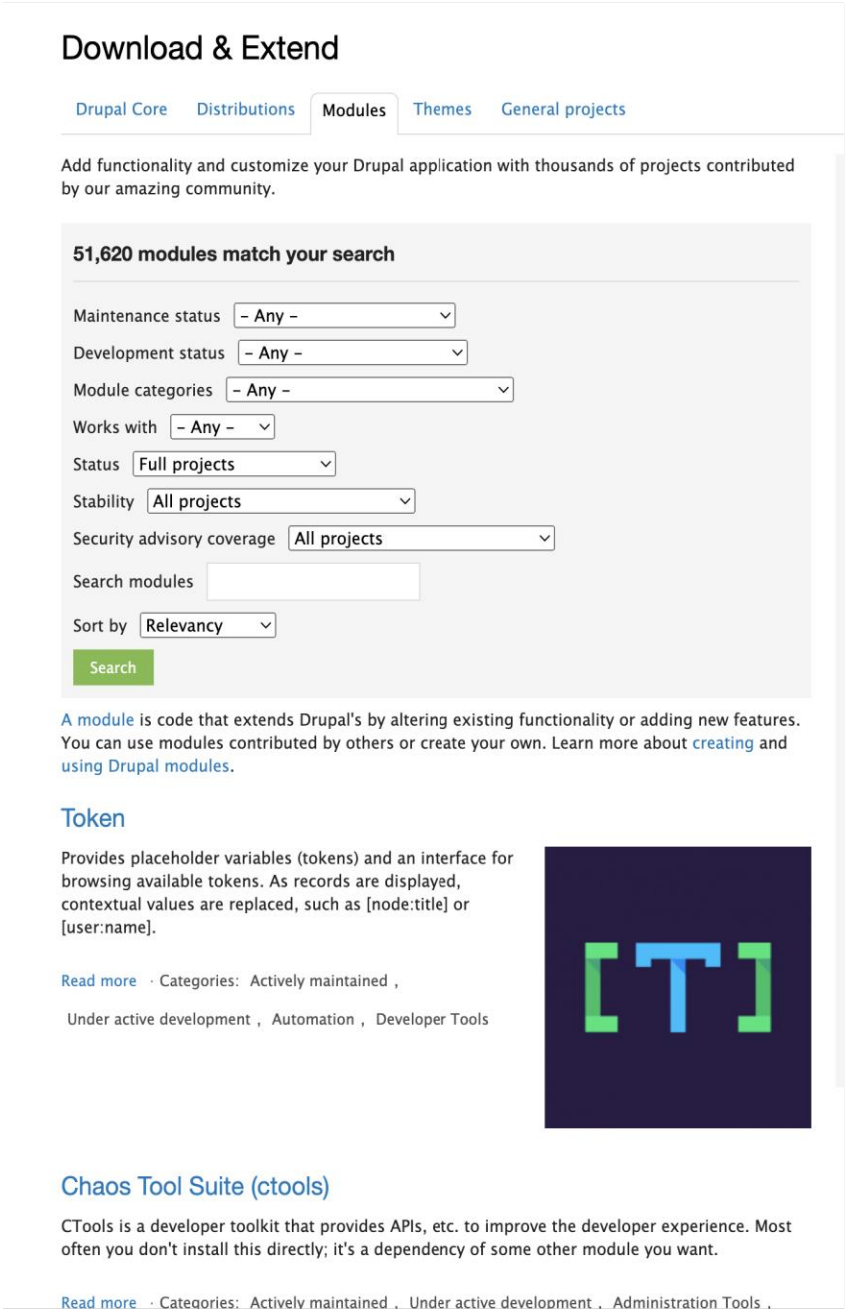

Рисунок 2.1 – Сторінка з доступними модулями [5]

Крім того, будь-яку тему можна взяти як за основу нової, це означає, що є можливість взяти вже готовий код з компонентами, і використати це у власній темі, таким чином можна розвивати нову тему з новим дизайном, але мати вже якісь готові компоненти.

Підсумовуючи, можна твердно заявити, що CMS Drupal дозволяє зробити сайт будь-якої складності. Цей сайт буде безпечним, надійним та швидким. Якщо потрібна додаткова функціональність, завжди є можливість знайти відкриті модулі на офіційному сайті та завантажити їх. Крім того, не потрібно переживати

про актуальність та застарілість, адже це є відкрита система з розвиненою спільнотою, яка буде підтримувати і покращувати цю систему.

2.2 Інструменти для роботи з локальним середовищем Drupal та їх застосування

DDEV – це інструмент із відкритим кодом для запуску локальних середовищ веб-розробки за лічені хвилини і навіть менше. В контексті друпалу, цей інструмент дозволяє швидко запустити середовище для роботи. Крім того, цей інструмент автоматично підключає базу даних до сайту, та інші сервери, наприклад, SOLR. Також, він автоматично створює директорію з конфігураційними файлами, які можна змінити під свої потреби. Завдяки цьому інструменту, є можливість запускати кілька середовищ паралельно, це означає, що якщо є потреба в тому, щоб два сайти працювали одночасно і були синхронізовані, то це можна налаштувати.

Для успішної роботи з цим інструментом, потрібно лише його завантажити на комп'ютер, він працює на Windows, macOS та Linux. І за допомогою лише кількох команд, створюється локальне середовище з розгорнутим сайтом. Проте, цей інструмент має залежність від іншого – Docker.

Docker – це програмне забезпечення, яке відокремлює певну кількість ресурсів під контейнер та керує ним. У межах цієї ізольованої частини можна встановити потрібну операційну систему й змусити код працювати так, як це потрібно. В одному контейнері може бути встановлений Linux, а в другому – Mac, і плюсом є те, що один контейнер ніяк не впливає на інший.

Docker працює за системою контеринізації, по суті, це є свого роду упакування програми у віртуальний контейнер разом зі всіма речами, які потрібні для її запуску. Тому, в цьому випадку є можливість запустити два різних сайта на Drupal в одній системі.

Docker працює за принципом трьох етапів – побудова, доставка та запуск.

Перший етап – створюється основний файл докеру та різні доповнення. Потрібно лише один раз визначити як налаштувати програму і що це за програма, та на чому вона повинна працювати. У випадку з Drupal, потрібно визначити, яка версія мови програмування PHP та яка версія бази даних.

Другий етап – доставка образу на сервер. Це потрібно для того, якщо є потреба розгорнути той самий сайт в іншому середовищі, або на іншому комп'ютері. Наприклад, два розробника працює над одним і тим же продуктом, то в цьому випадку це потрібно, щоб не було відмінностей і конфліктів у налаштуваннях. Кінцевим етапом є запуск контейнерів

Як було згадано раніше, Drupal працює за принципом модульної системи. На офіційному сайті, можна знайти будь-який модуль та завантажити його. Наприклад, є потреба у завантажені модулю «Menu item extras», для кращого налаштування пунктів меню на сайті. Тоді, для його завантаження, потрібно перейти на його сторінку на офіційному сайті друпалу, в самому внизу буде секція, яка вказує як завантажити модуль (рис. 2.2).

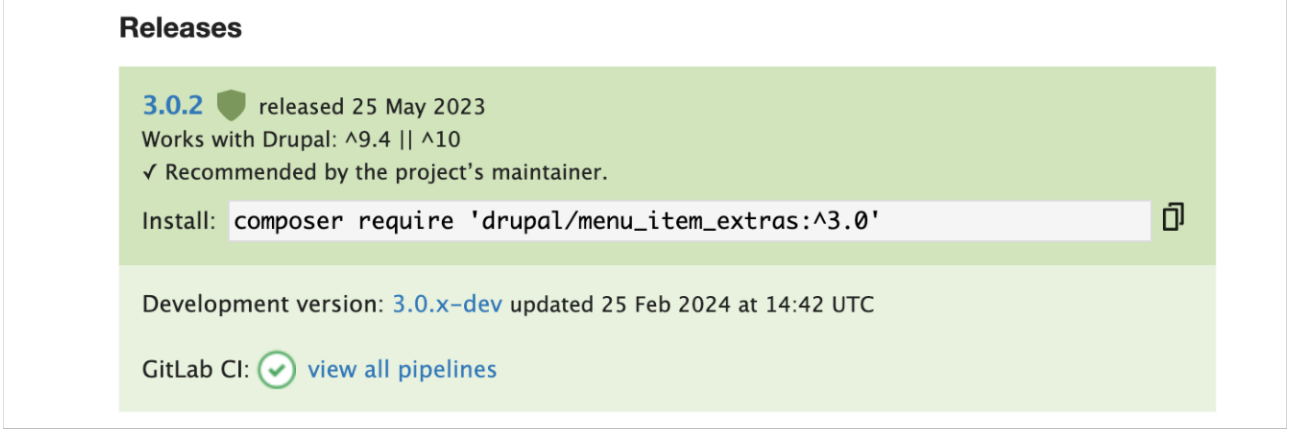

Рисунок 2.2 – Інформація про останню версію модулю [6]

Тобто, є спеціальна команда для завантаження модулю. Для цього потрібен інструмент – Composer. Цей інструмент управління залежностями в проектах, які побудовані на основі мови програмування PHP. По-суті, цей інструмент дає можливість ефективно керувати різними бібліотеками, завантажувати різні необхідні пакети та модулі.

Утиліта Composer використовує файл composer.json для опису і тримання залежностей проекту та автоматичного встановлення бібліотек і модулів. Тобто, з рисунку 2.2, видно що модуль використовує версію 3.0, тому після завантаження цього модулю, у файл composer.json буде автоматично додано залежність від цього модулю з версією 3.0. Таким чином, є можливість завантажувати різні модулі, теми та бібліотеки лише однією командою. А щоб інші розробнкии мали можливість розгорнути таке ж саме середовище, з такими ж модулями, достатньо мати файл composer.json та виконати команду – composer install.

Drush – це інтерфейс командного рядка для Drupal. Він надає різноманітні команди для покращення роботи в розробці Drupal, автоматизації робочих процесів і загалом полегшення створення сценаріїв для різних частин робочого процесу Drupal. Це зручний інструмент для всіх, а не лише для розробників. Потужний командний API Drush дозволяє зосередитися на обробці спеціальної логіки та дозволити Drush працювати зі звичайними завданнями CLI, такими як збір вхідних даних від користувачів, форматування виводу та відображення довідкових повідомлень. Також, використовуючи Drush можна активувати щойно завантажений модуль, отримати посилання на вхід в адміністративну панель, зробити копію бази данних, оновити базу данних або імпортувати нову базу данних.

В загальному, ці всі інструменти знадобляться в розробці системи, яка буде аналізувати дані. Адже, система була розроблена локально, для покращення розробки використана утиліта Drush, а для завантаження сторонніх бібліотек і модулів використаний Composer.

2.3 Архітектура Drupal

Як і було зазначено раніше, Drupal це скоріше модульна система. Тобто, це як пазл, де кожна деталь, це якийсь модуль, який відповідає за функціонал.

Основою всіх основ є ядро – потужний фундамент, який створює основу

для системи управління контентом світового класу. Ядро Drupal – це сукупність PHP-скриптів, що забезпечують такі основні функції, як керування користувачами, створення контенту та базові функції сайту. Отже, в ядрі є наступні компоненти:

– директорія config, в якій розташований файл core.extension.yml, це конфігураційний файл який імпортується при інсталяції сайту, в цьому файлі описано які модулі або повинні включитись після встановлення ядра;

– директорія lib, в якій розташовані всі бібліотеки, за замовчуванням там одна бібліотека – Drupal, яка виконує стандартні функції;

– директорія modules, в якій розташовані модулі, ці модулі не можна видалити або змінити, також туди не можна додати нові модулі. Це модулі за замовчуванням, і деякі з них будуть увімкнені після встановлення ядра;

– директорія scripts, в якій розташовані всі скрипти, наприклад, скрипт який генерує контент, або скрипт який експортує базу данних.

– директорія tests, в якій розташовані всі файли тестів, які запускаються для перевірки чи не зламалась функціональність ядра друпал;

– директорія themes, в якій розташовані всі теми.

Ядро – база друпалу, воно запускає всі процеси та співпрацює з багатьма іншими серверними додатками. Ядро побудоване за допомогою модульної системи, модулі бувають трьох типів:

– ті що знаходяться в ядрі, контрібні і кастомні (рис. 2.3), вони не можуть бути змінені самостійно, тому що це відкритий код, ці модулі завантажуються за допомогою вже згаданого інструменту composer;

– контрібні модулі – це сторонні модулі, які були зроблені іншими розробниками і завантажені на офіційний сайт друпалу, що дає змогу будь-кому завантажити цей модуль і використовувати його;

– кастомні модулі – це власний функціонал, який був розроблений під конкретний проект, і є приватною власністю розробника, ці модулі, як правило є специфічними, і не можуть бути використані в інших проектах.

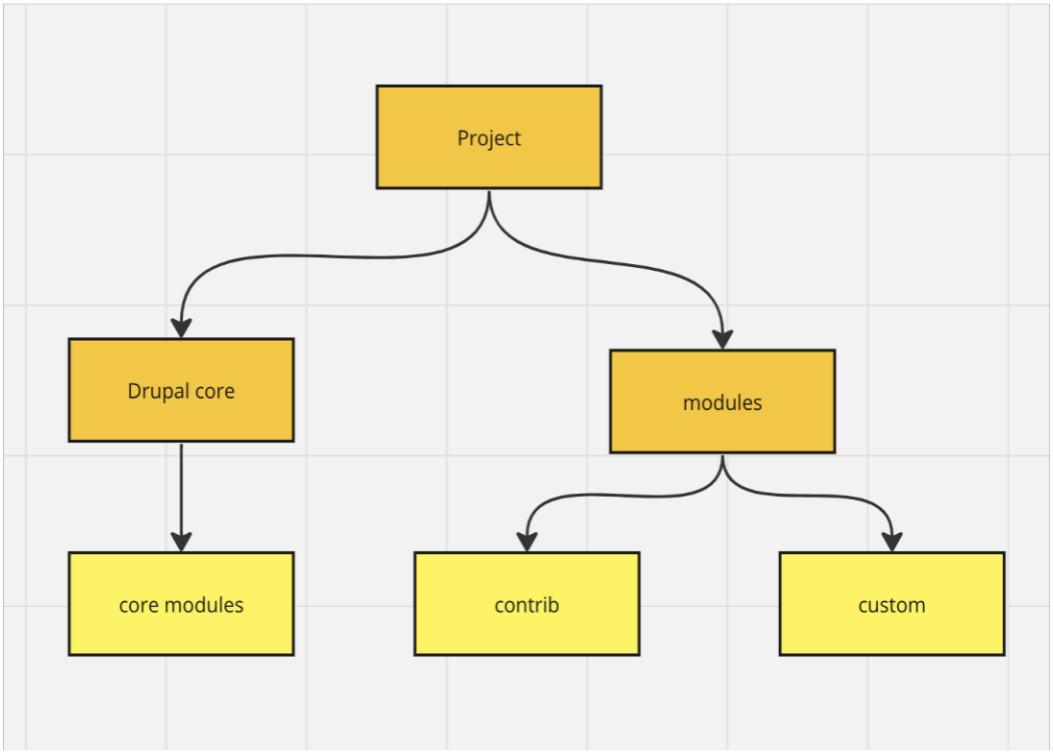

Рисунок 2.3 – Архітектура модульної системи

Для того, щоб модуль працював, він повинен містити певні файли. Наприклад, в цій дипломній роботі створено «diplom_airline» - кастомний модуль, який надає весь функціонал пов'язаний з авіалініями.

Отже, для того, щоб система розпізнавала цей модуль, потрібно створити «diplom_airline.info.yml» файл, і в цьому файлі описати базові налаштування модулю, такі як: ім'я модулю, тип, опис, версія ядра друпалу, залежності, конфігурації (рис. 2.4).

```yaml
name: diplom_airline
type: module
description: 'Provides an airline entity.'
package: Custom
core: 8.x
core_version_requirement: ^8 || ^9
dependencies:
  - drupal:text
configure: entity.airline.settings
```

Рисунок 2.4 – Базові налаштування модулю

З цим файлом, система повинна розпізнати цей модуль, і дає можливість активувати його. Є інші додаткові файли і директорії, які виконують якусь роль, але при цьому модуль може працювати і без них, в загальному структура модулю вигядає згідно рисунку 2.5.

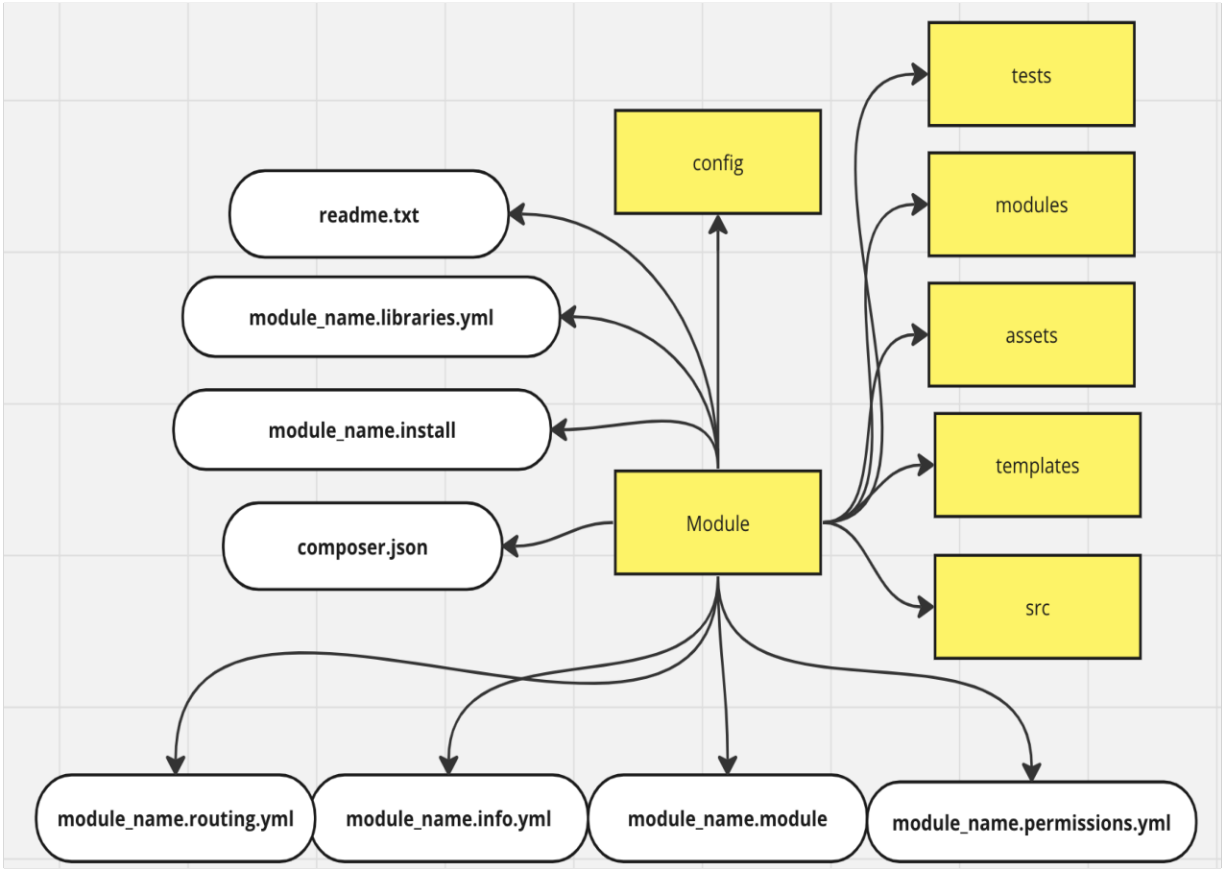

Рисунок 2.5 – Архітектура модулю

З огляду на рисунок 2.5, можна відзначити наступне:

– readme.txt – це текстовий файл, який описує який функціонал надає модуль, це зроблено для того, щоб інші розробники змогли легко зрозуміти суть того чи іншого модулю;

– module_name.libraries.yml – цей файл ініцілізує внутрішні бібліотеки. Як правило, ці бібліотеки надають стилізацію або JavaScript код, ці бібліотеки можуть бути перевикористані в інших місцях, проте це відбувається дуже рідко;

– module_name.install – цей файл спрацьовує лише один раз – коли модуль вмикається на сайті, зазвичай, в цьому модулі додається код, який повинен спрацювати лише один раз, наприклад, створення сутностей або встановлення необхідних конфігурацій, без яких модуль не може працювати;

– composer.json – цей файл ініцілізує залежності від зовнішніх пакетних сутностей або бібліотек;

– module_name.routing.yml – в цьому файлі ініцілізуються маршрути;

– module_name.module – в цьому додаються «хуки»;

– module_name.permissions.yml – в цьому файлі ініцілізуються дозволи;

– config – ця директорія містить додаткові конфігурації, які надає модуль;

– tests – ця директорія містить файли для тестування, щоб бути впевненим, що цей модуль надає корректну функціональність, яка не зламає інші модулі;

– modules – ця директорія містить саб-модулі, тобто, дочірні модулі;

– assets – ця директорія, як правило, містить CSS та JavaScript код;

– templates – ця директорія містить шаблони тих чи інших компонентів, які надає модуль;

– src – ця директорія містить PHP код, тобто основну функціональність модулю, наприклад, плагіни, сервіси, контроллери і так далі.

Ахітектура тем, працює майже ідентично до модульної. Теми так само можуть бути як контрібні, кастомні та ті що розташовані в ядрі. Проте, є певні відмінності, та і самі поняття суттєво відрізняються;

Теми – це частина Drupal, яку ви та будь-хто інший, хто відвідує вашу програму на базі Drupal, бачите під час перегляду будь-якої сторінки у своєму браузері. Це свого роду як шар, на зразок екрана, який існує між вмістом Drupal і користувачами сайту. Щоразу, коли запитується сторінка, Drupal збирає вміст для відображення в структуровані дані, які потім передаються на кінцевий етап це є відображення цих даних. В темі можуть бути:

– css стилі, які відповідають за зовнішній вигляд сайту;

– шаблони компонентів, які відповідають за структуру сторінок, а потім на кінцевому етапі конвертуються в HTML;

– JavaScript код, який відповідає за інтерактивні речі на сайті;

– різного роду ресурси, такі як: картинки, іконки і так далі.

Основною особливістю тем є наслідування. Тобто, є можливість взяти будь-яку тему, і зробити її як за основу, тобто, батьківською. Це надає весь функціонал теми, але з можливістю розширення. В цій дипломній роботі, було взято за основу тему «Bootstrap».

Майже кожен основний компонент, з якого складається сайт Drupal, є сутністю того чи іншого типу, наприклад, ноди, користувачі та блоки є типами сутностей. Ці сутності можна налаштувати, додавши до них поля. Розуміння того, як працює система Entity, є основним для розуміння як працює друпал.

Є можливість змінити існуючі типи сутностей, та підлаштувати їх під сайт, також можна створювати власні типи сутностей, щоб інкапсулювати поведінку певних структур даних у програмі. За замовчуванням, друпал надає такі типи сутностей: нода, користувач та блок. Всі ці типи сутностей розташовані в ядрі, та можуть використовуються по замовчуванню.

Є два типи сутностей – конфігураційна сутність та контент сутність:

– контент сутності зберігаються в базі данних, і не можуть бути використані на іншому середовищі, тільки у разі перенесення бази данних;

– конфігураційна сутність зберігаються у файлах з розширенням yml. Таким чином, ці конфігурації можуть переноситись з середовища на інше, за допомогою імпорту та експорту.

Термін «нода» у друпалі використовується для позначення основної одиниці зберігання контенту. Кожна стаття, блоговий запис, сторінка або будь-який інший тип контенту в друпалі є нодою.

Ноди можуть містити різні поля (такі як заголовок, текст, зображення тощо) і можуть бути організовані у різні типи та категорії. Ноди є основною будівельною одиницею для створення та керування контентом в друпалі. Вони дозволяють створювати, редагувати та видаляти контент на веб-сайті, а також надають можливість встановлювати різні права доступу до цього контенту для різних користувачів.

Отже, сама нода це є контент, це може бути як стаття або ж новина. Проте налаштування ноди є конфігурації, такі як: тип ноди, форма для заповнення ноди, форма для виводу ноди, поля і так далі. Рисунок 2.6 ілюструє ієрархію сутностей.

В друпалі є типи сутностей та «бандли». Наприклад, користувач та нода, це різні типи сутностей. Проте, стаття або новини є бандлами типу сутності ноди. Тобто, це є свого роду класифікація. Наприклад, береза, клен та дуб є деревами, тому, умовно кажучи, всі вони мають тип – дерево, проте сам вид дерева, це бандл.

В сутності можуть бути додаткові елементи, такі як поля. Наприклад, сутність користувача, має такі поля як: ім’я, електронна скринька та пароль користувача. Кожне поле записується в базу даних, та прив’язується до конкретного об’єкту через ідентифікатор.

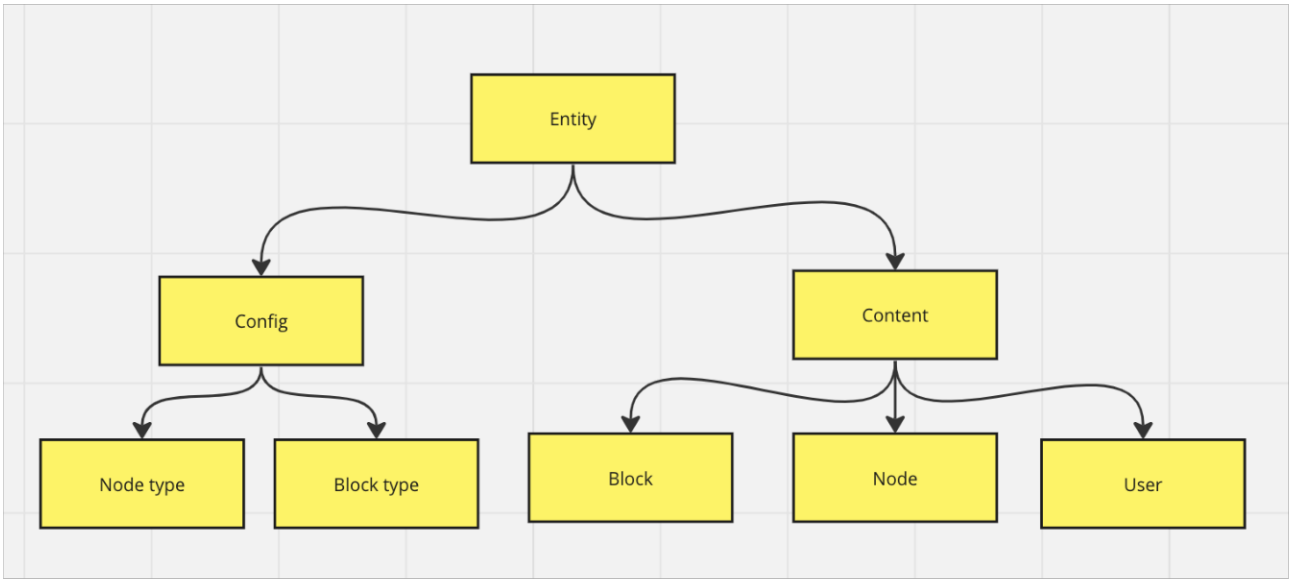

Рисунок 2.6 – Архітектура Entity API.

Наприклад, на сайті створено дві сутності авіаліній, в кожної є свій ідентифікатор в базі даних, але обидва мають спільні поля, тому значення полів записуються в окремі таблиці в базі даних, та прив’язуються до кожного об’єкта авіалінії (рис. 2.7).

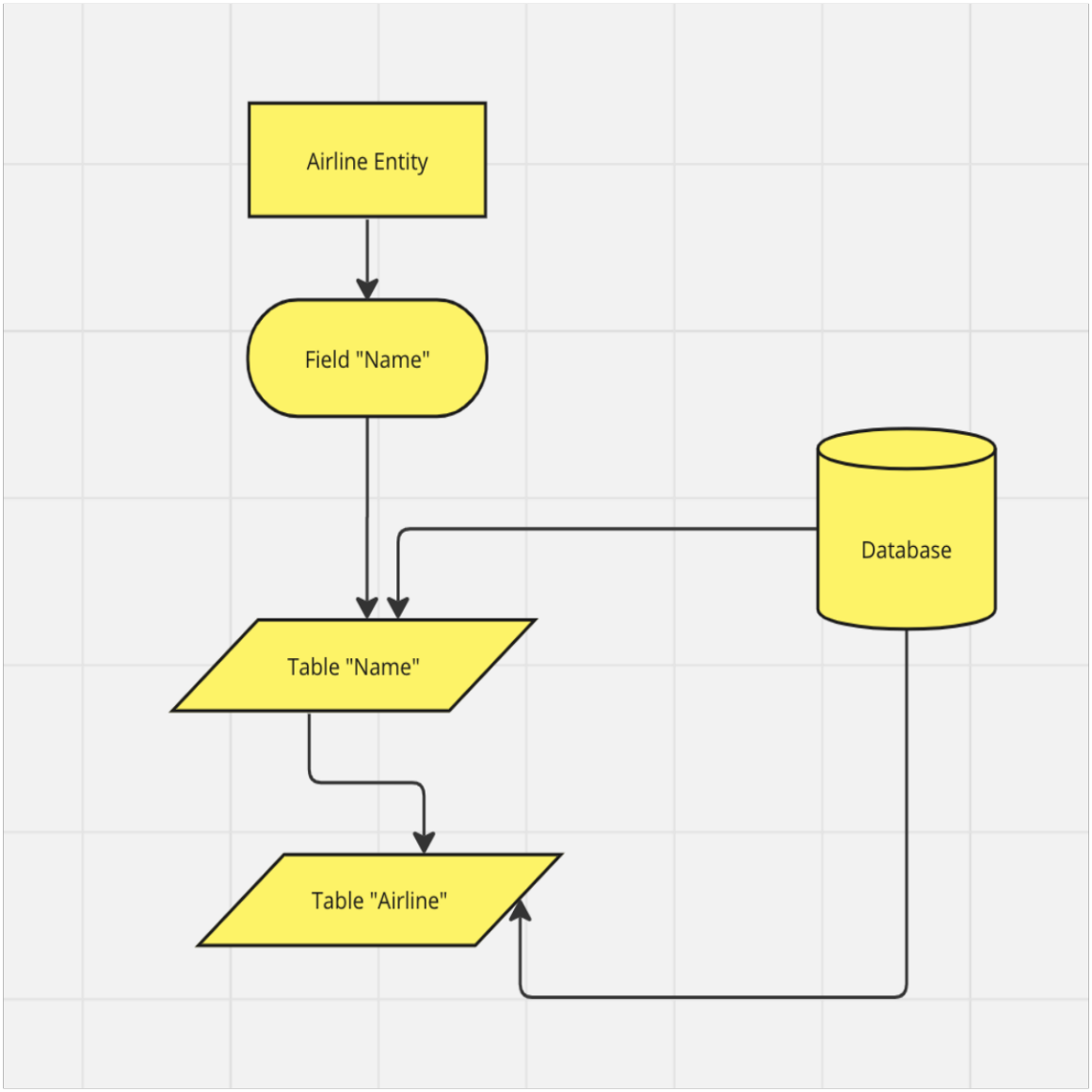

Рисунок 2.7 – Робота полів з базою даних

Ще одними з основних компонентів архітектури друпалу є – дозволи і ролі. По-суті, вони надають можливість контролювати доступ до різних сторінок, блоків і ресурсів. Наприклад, деякі ресурси можна надавати тільки авторизованим користувачам, а деякі всім. Це все легко налаштовується через конфігурації в адміністративній панелі.

За замовчуванням, друпал надає три ролі це – анонімний, авторизований користувач та адміністратор, але є можливість додавати нові ролі, наприклад, модератор або контент менеджер. Ці ролі можна призначати специфічним користувачам.

Наприклад, як адміністратор я можу редагувати профіль іншого користувача, і там я можу призначити нові ролі або забрати старі ролі (рис. 2.8).

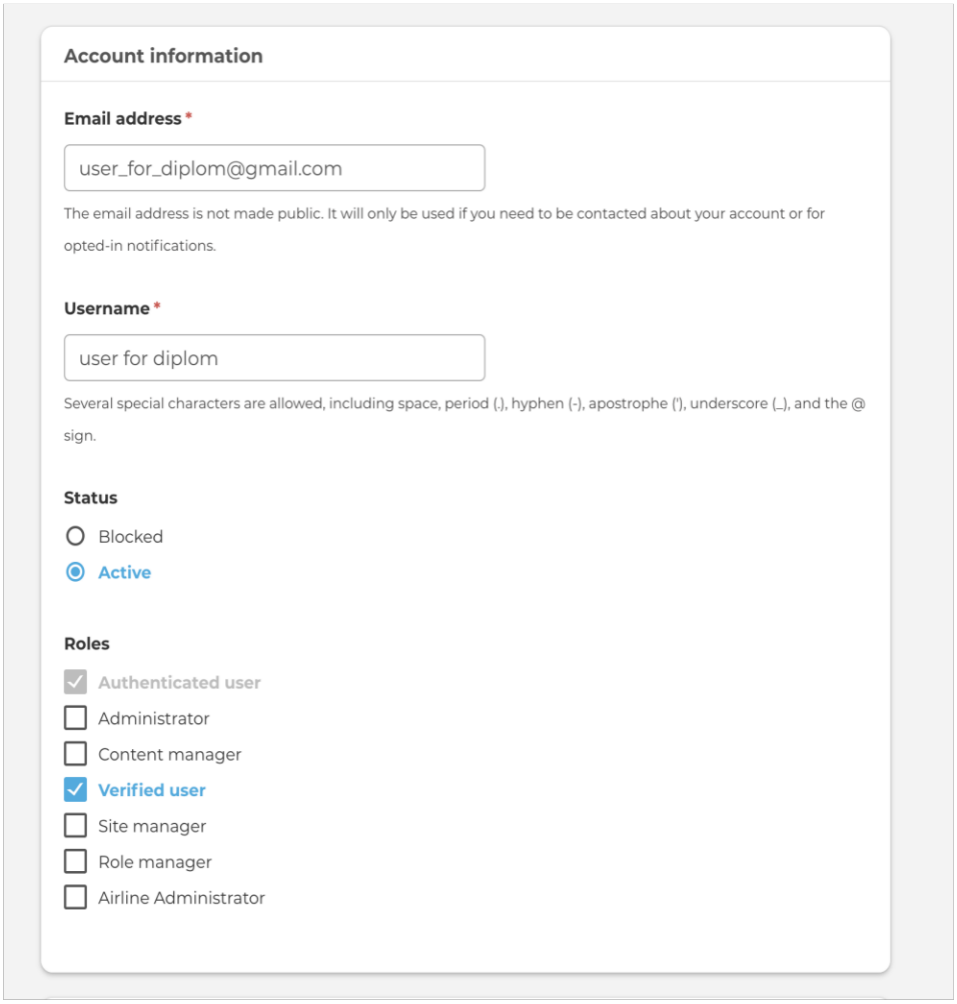

Рисунок 2.8 – Надання нової ролі користувачу

До кожної ролі, можна призначати специфічні дозволи. Наприклад, для перегляду вмісту новин і блогу існують різні дозволи. Або ж, для перегляду сторінок авіаліній і сторінок аеропортів є теж різні дозволи. Це також налаштовується в адміністративній панелі друпалу, і це є конфігурації.

Сутність тісно співпрацює з дозволами, адже, практично для кожної дрібниці на сайті можна створити дозвіл, за допомогою якого буде контролюватись доступ для групи тих чи інших користувачів.

Наприклад, якщо користувач реєструється на подію, ми надаємо йому доступ до адреси цієї події, тільки після того як цей же користувач туди

зареєструється і отримає нову роль. За адресу відповідає якесь поле, яке є одиницею типу вмісту – подія, і для цього поля наданий окремий доступ у вигляді специфічного дозволу.

Кожний компонент в друпалі взаємодіє один з одним. Це в свою чергу утворює «двигун» на якому і працює сайт. В загальному архітектура друпалу, виглядає приблизно як на рисунку 2.9.

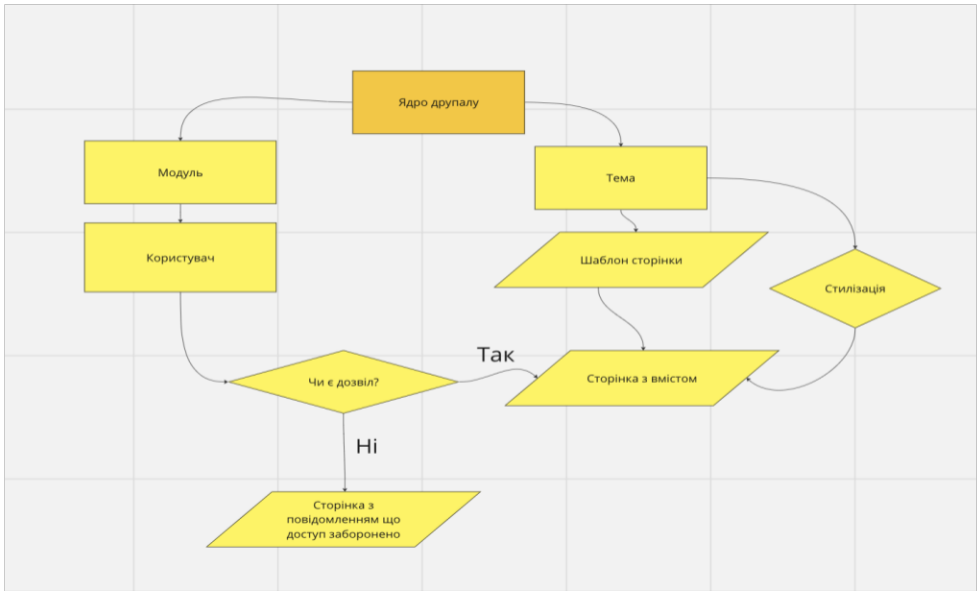

Рисунок 2.9 – Архітектура друпалу

Можна зазначити, що архітектура друпалу дуже гнучка, і її можна навіть змінити під свої потреби. Для цього можна взяти ядро друпалу як основу, і модифікувати її під свої потреби.

Весь функціонал пишеться за допомогою мови програмування PHP. Це скриптова мова програмування, застосовується для розробки додатків та сайтів. По-суті, ця мова виконує сценарії – якісь спеціально написанеі програми на сервері, що справцьовують у відповідь на запит від користувача. Тобто PHP є свого роду інтерпретатором, а сам процес відповідно, називається інтерпретацією. У випадку з друпалом, абсолютно весь код, скрипти, різні сценарії і ядро написано за допомогою мови програмування PHP. Ця мова програмування стрімко розвивається, і з кожним оновленням з’являються все нові і нові методи, які полегшують життя розробникам. Наприклад, прості цикли

для пошуку якогось значення в масиві, тепер можна вирішити за допомогою методу array_search().

2.4 Шаблонізатор Twig та його роль в Drupal розробці та зовнішнього вигляду інтерфейсу

Одним з ключових інструментів друпалу є шаблонізатор Twig. Друпал не може працювати з якимось іншим інструментом, окрім Twig, тому він використовується і до цих пір, хоча в старих версіях друпалу він не використовувався.

Twig – це потужний шаблонізатор, який дозволяє створювати дуже складні проекти з сотнями розширених шаблонів, тисячами блоків, фільтрів та функцій. Також Twig пропонує пісочницю, унікальну, але не широко використовувану функцію, яка дозволяє створювати безпечні шаблони, що редагуються користувачем. Крім того, в екосистемі Twig є ряд зручних вбудованих і сторонніх інструментів для спрощення щоденного робочого процесу.

Синтаксис твігу є специфічним, зміні записуються у подвійні фігурні дужки. Також твіг має такі оператори як: «if» та «for». Тобто, це дає змогу динамічно будувати різні шаблони (рис. 2.10).

В кінцевому результаті, шаблон трансформується у HTML розмітку яку може прочитати браузер та вивести, а змінні перетворюються на відповідні значення.

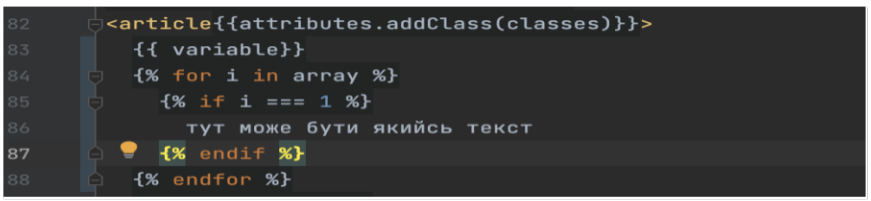

Рисунок 2.10 – Синтаксис шаблонізатору Twig

За кожну деталь на сайті в друпалі, відповідає конкретний шаблон. Наприклад, за меню відповідає один шаблон, а за пункт меню інший. Ці всі

шаблони Twig динамічно генерує на стороні серверу. Наприклад, за основну сторінку в друпал відповідає шаблон html.twig.html, проте цей шаблон може бути перезаписаний іншими шаблономи, залежно від контексту. Схема на рисунку 2.11 описує, які можуть бути шаблони і за що вони відповідають.

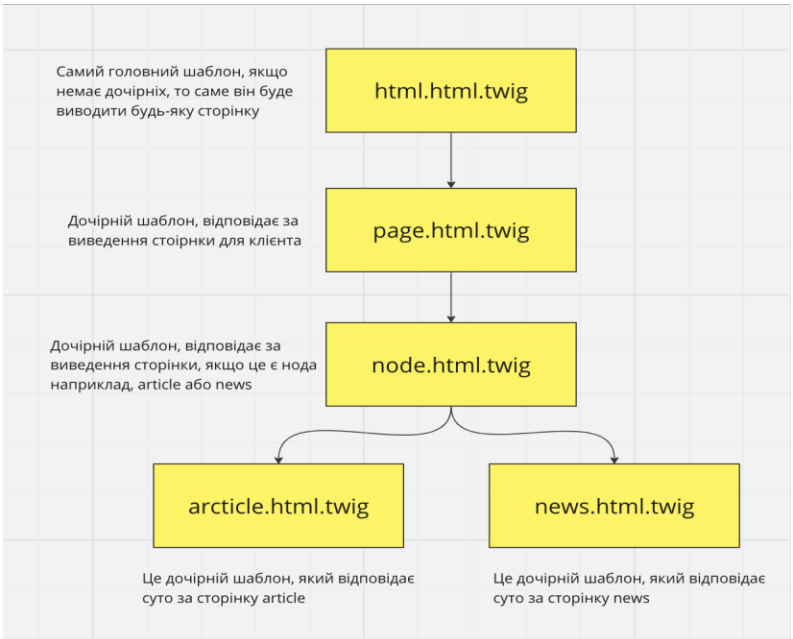

Рисунок 2.11 – Іеархія шаблонів в друпал

В основному Twig працює в друпал складно. Занадто багато різних процесів залучено для побудови шаблонів, які відповідають за виведення тої чи іншої сторінки.

Наприклад, було згадано, що в шаблоні можуть бути якісь зміни – ім’я ноди або адреса. За побудову і передачу цих змінних до шаблону відповідальний Render API. За компіляцію шаблонів відповідальний спеціальний сервіс Compiler. Також, в друпалі є різні хуки, за допомогою яких можна змінити відображення того чи іншого шаблону. Основний з них, є хук препроцес, це по-суті, кінцевий етап, де можна змінити або додати нову зміну до шаблону. Зазвичай, ці препроцеси існують для того, щоб динамічно змінювати вміст сторінки, залежно від якихось умов. Render API містить 14 кроків, кожен крок за щось відповідає,

наприклад, є крок, який надає кеш для тих чи інших елементів, а препроцес є останнім кроком. Процес обробки шаблонів проілюстровано на рисунку 2.12.

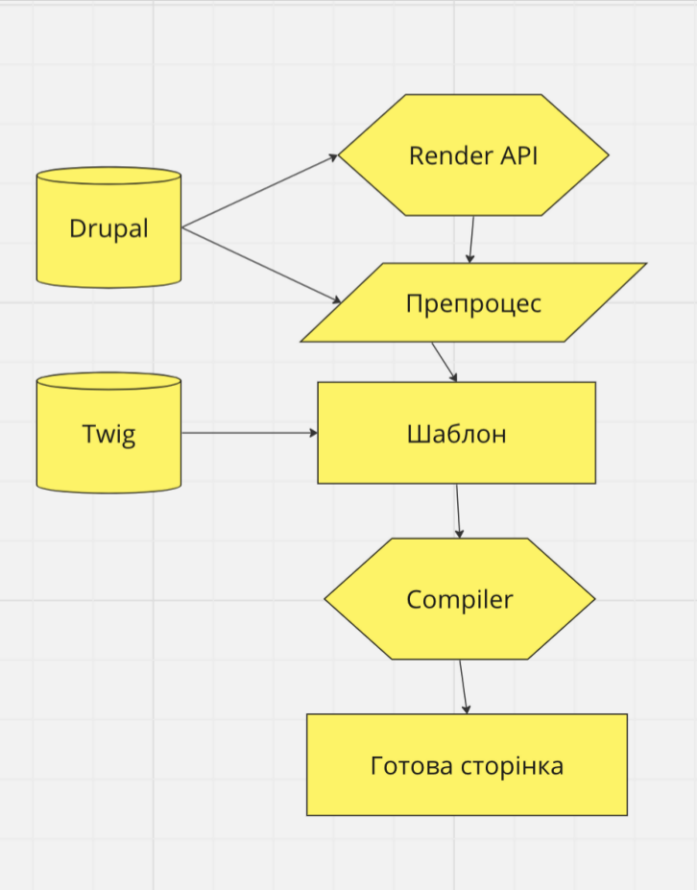

Рисунок 2.12 – Взаємодія друпалу з Twig

З вище вказаного можна зазначити чотири ключові моменти:

– розмітка сторінок: Twig створює ті ж самі HTML шаблони для сторінок сайту, це означає, що можна легко відокремлювати відображення від бізнес-логіки;

– використання змінних та фільтрів: Twig дозволяє використовувати змінні для вставки динамічного контенту в шаблони, наприклад, дані з бази даних, або динамічні змінні з PHP-коду, та також надає фільтри для обробки цих змінних, таких як форматування дати чи видалення тегів або зайвих пробілів;

– наслідування шаблонів: Twig підтримує концепцію наслідування шаблонів, що дозволяє створювати базові шаблони та наслідувати їх для

конкретних сторінок або компонентів, що в свою чергу зменшує обсяг зайового коду та файлів;

– блоки та включення: Twig дозволяє вбудовувати блоки коду та включати один шаблон в інший, що в свою чергу також допомагає у впорядкуванні та повторному використанні коду.

2.5 Інструменти Theme API, CSS та JavaScript для розробки функціоналу зовнішнього вигляду веб-інтерфейсу

Drupal як і всі CMS, будують серверну і клієнтну частину. Це означає, що не потрібно задіювати додаткові фреймворки для побудови «Front-end». Drupal надає внутрішні інструменти для побудови візуалізації, такі як Theme API та Render API, та також співпрацює з такими інструментами як JavaScript, Jquery, SCSS та CSS.

Theme API – функціональність, яка відповідає за зовнішню частину інтерфейсу. На сайт можна встановити ту чи іншу тему з різним дизайном. Кожна тема надає стандартні стилі, JavaScript код та шаблони. Варто зазначити, що існують два типи тем, це теми для адмін панелі та для клієнта.

HTML – це мова гіпертекстової розмітки документів, яка працює через систему тегів і структурує елементи на сторінках. Цей інструмент завжди порівнюють з будинком, тобто, кімната або меблі – це елементи (рис. 2.13). І якщо порівнювати це з браузером, то HTML описує структуру, а саме: де має бути заголовок, текст, картинка, посилання тощо. Це каркас, на основі якого будуються всі сайти. В контексті друпалу, немає чистого HTML, як і було зазначено раніше є шаблони, які компілюються в HTML розмітку автоматично.

Проте, кожна кімната може мати свої кольори або розміри. Саме за допомогою інструменту CSS це можна зробити.

CSS – це такий набір команд, які відповідальні за візуалізацію сторінки. Наприклад, кольори тексту або якихось елементів, ширина або висота певних елементів, розміри та адаптивність картинок, це все робить CSS. Варто

зазначити, що CSS працює тільки з браузером, і це надає можливість прямо в браузері надавати якусь стилізацію для елементів.

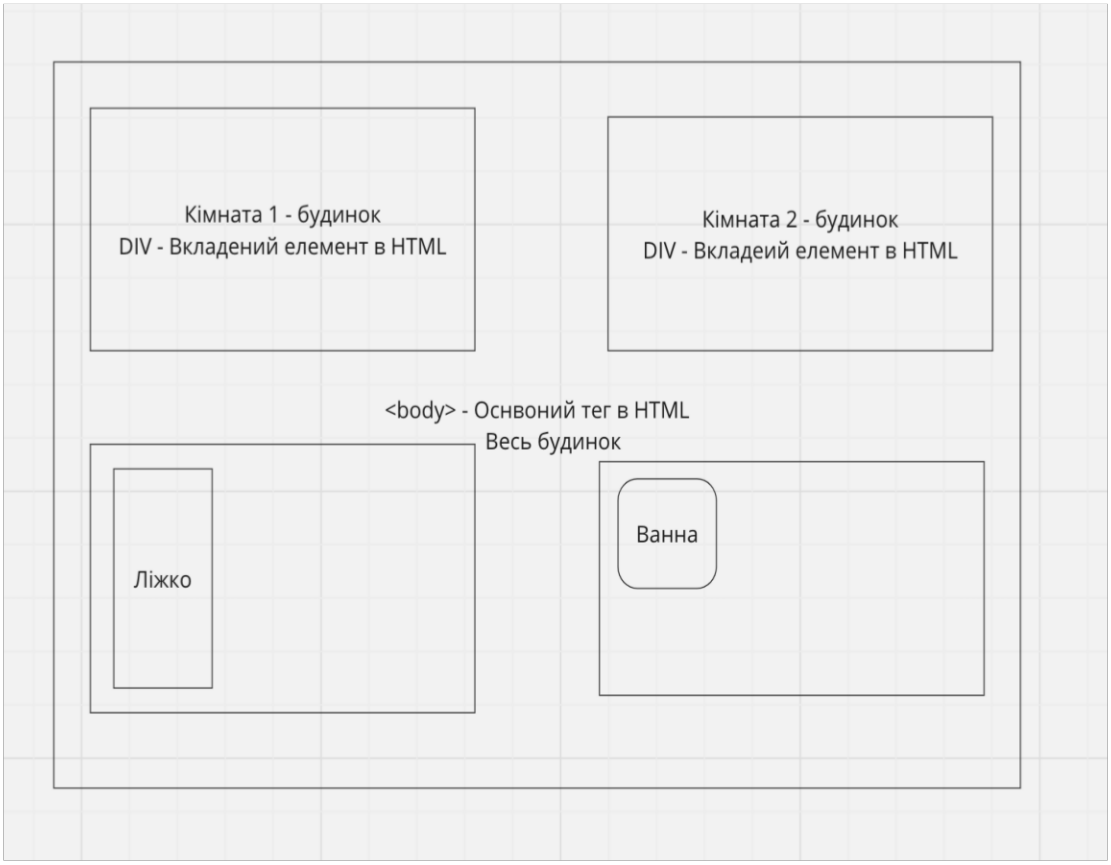

Рисунок 2.13 – Каркас будинку на основі HTML

Це дуже корисно для розробників, для цього використовується інспектор, який є в кожному браузері. CSS як і HTML може бути компільованим. Наприклад, є спеціальні «мови препроцесори», які покращують читання та написання команд для стилізації. Тобто, це по-суті, математичні мови програмування, за допомогою яких можна писати різні формули, наприклад, для обчислення ширини якогось елементи. І в кінцевому результаті цей код компілюється в CSS код. Проте, потрібно залучити додаткові інструменти, такі як GULP або npm.

По-суті, за допомогою HTML і CSS можна будувати прості сторінки в різних інтерфейсах, але потрібно розуміти, що за допомогою цих інструментів можна побудувати статичні, але ніяк не динамічні сторінки. За динамічні дії

відповідає мова програмування JavaScript. JavaScript – це скриптова мова програмування, яка широко використовується для створення інтерактивних інтерфейсів, застосунків та сайтів. Ця мова програмування додає різного роду анімацію і спливаючі повідомлення, валідує форми і змушує інтерфейс реагувати на наші дії. Наприклад, поширений випадок – при наведенні курсору на кнопку, вона змінює колір, або зникає після натискання. Тобто, JavaScript робить сторінки динамічними. Якщо ж знову проводити аналогію з будинком, то можна зазначити, що відкривання дверей до якоїсь кімнати, це є свого роду інтерактивність з будинком, це саме те що робить JavaScript на сайті.

В контексті друпалу, саме «чистий» JavaScript рідко де використовується, тільки в сторонніх бібліотеках. Насправді, вся інтеракція в друпалі побудована на коді бібліотеки Jquery, яка є досить старою, але при цьому ефективною. Саме за допомогою цієї бібліотеки побудовано багато інтерактивного коду в інтерфейсі для моніторингу та аналізу даних. Підключати цю бібліотеку не потрібно, вона автоматично встановлюється разом з ядром друпалу.

У поєднанні ці всі інструменти надають середовище для розробки зовнішнього вигляду інтерфейсу. Theme API в друпалі працює зі всіма сучасними інструментами розробки. Навіть, якщо потрібно підключити додаткові інструменти, наприклад, фреймворк React, це запросто можна зробити, адже існує безліч модулів, які автоматично підключають різні технології та надають базу для подальшої розробки.

2.6 Робота Drupal із зовнішніми API

Drupal дуже гнучка CMS, але дуже часто потрібно підключати додаткові сервіси для розробки. Наприклад, для оплати простих покупок на сайті, потрібно підключати сторонні API, це можуть бути і MonoBank, PayPal та інші API. Тут важливо – правильно налаштувати цей API.

Майже всі API підключаються за допомогою API ключа. За допомогою цього ключа можна отримати дані з одного сховища в інше (рис. 2.14), в цьому

випадку – сайт і зовнішня API. У розділі 3 чітко описано, як отримуються та підключаються ключі до даного інтерфейсу. Проте, це доволі простий спосіб, і майже не використовується. Уявіть, що хтось може вгадати ключ, і отримувати захищені дані. Це майже неможливо, але ймовірність така є. Саме з цієї причини, у більшості випадків, аутентифікація відбувається через додаткові захисні методи. Їх може бути безліч, так само і рівнів перевірки. Наприклад, в Drupal є модуль «SimpleOauth», він надає способи для з'єднання через різні токени. Працює це дуже просто – на сайті генерується токен та клієнт, після чого він записується у файл, шифрується на зберігається в якомусь конкретному місці. Потім, після першого з'єднання із зовнішнім API цей токен запам'ятовується. Це відрізняється тим, що API запам'ятовує токен та клієнта, який з'єднався, що означає, що ніхто інший ніколи не зможе отримати дані, навіть якщо він і вгадає токен.

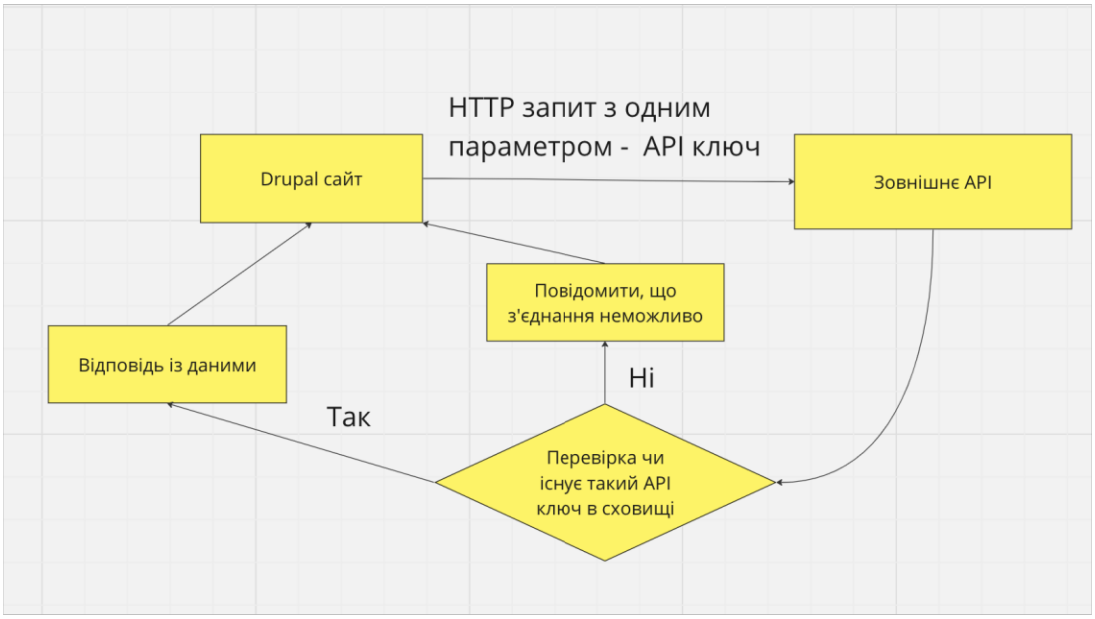

Рисунок 2.14 – З'єднання за допомогою HTTP протоколу з API ключем

Всі дані конвертуються у деякі формати, та зашифровуються спеціальними ключами. Інколи трапляється, що зловмисники все-таки перехоплюють трафік, та отримують всі дані. Тому, більшість API передають дані за допомогою протоколу HTTPS. Це ще один додатковий захист, адже, існує два протоколи

HTTP та HTTPS. Протокол HTTP використовує технологію клієнт-сервер: клієнт відправляє запит, сервер отримує цей запит, формує відповідь і повертає клієнту, самі дані ніяк не захищені. HTTPs розширює протокол HTTP, що робить його безпечним. Протокол HTTPs вимагає, що під час встановлення з'єднання клієнт і сервер використовують певний тимчасовий клюа, за допомогою якого будуть зашифровуватись та розшифровувати дані. Тому всі з'єднання між друпалом і сторонніми API працюють через протокол HTTPS.

РОЗДІЛ 3
РОЗРОБКА ТА РЕАЛІЗАЦІЯ ІНТЕРФЕЙСУ ДЛЯ МОНІТОРИНГУ ТА АНАЛІЗУ ДАНИХ

3.1 Проектування та стилізація веб-інтерфейсу

Для побудови «Front-end» інформаційної системи за допомогою CMS Drupal є багато різних шляхів. Як і було зазначено раніше, є кілька способів – це використовувати внутрішні інструменти Drupal, такі як Theme API та Headless, де є можливість використання додаткових фреймворків таких як: React.js або Vue.js, але оскільки другий шлях потребує додаткових ресурсів, то буде використано стандартний шлях.

Отже, для початку потрібно створити саб-тему. Для цього, потрібно виконати наступні кроки:

– вибрати тему для наслідування;

– створити папку для саб-теми в в папці themes/custom у файловій системі проекту;

– створити основний конфігураційний файл сабтеми;

– підключити бібліотеку з css стилями;

– включити сабтему на сайті.

Щоб, вибрати яку батьківську тему використовувати, потрібно зважити всі за і проти. В друпал, є загалом кілька тем з «коробки», це – classy, bartik, olivery та claro. Проте, на drupal.org, є змога знайти ще додаткові теми, наприклад, «Bootstrap4». Ця базова тема долає розрив між Drupal і Bootstrap Framework. Якщо коротко, то ця тема надає вже готові стилі, які можуть бути використані у проекті, наприклад – стилі кнопок, хедеру, футеру та багато інших. Це означає, що не потрібно писати стандартну стилізацію з нуля, а буде можливість використовувати існуючу.

Далі було завантажено цю тему у файлову систему проекту, за допомогою інструменту composer. На цій ж самій сторінці є секція з останніми випусками цієї теми.

Далі було додано директорію «diplom_bootstrap4» в директорії themes/custom, та створено конфігураційний файл «diplom_bootstrap4.info.yml» файл в середині цієї директорії, де є можливість прописувати налаштування щойно створеної теми, такі як: підключення css бібліотеки, ініцілізація регіонів, версія друпалу, ім'я та опис теми. В цьому випадку цей файл виглядає ось так (рис. 3.1). Проте, є завжди можливість додавати або видаляти якісь налаштування, або змінювати існуючі, це вже залежить від того, яка структура повинна бути побудована, та які стилі будуть використані.

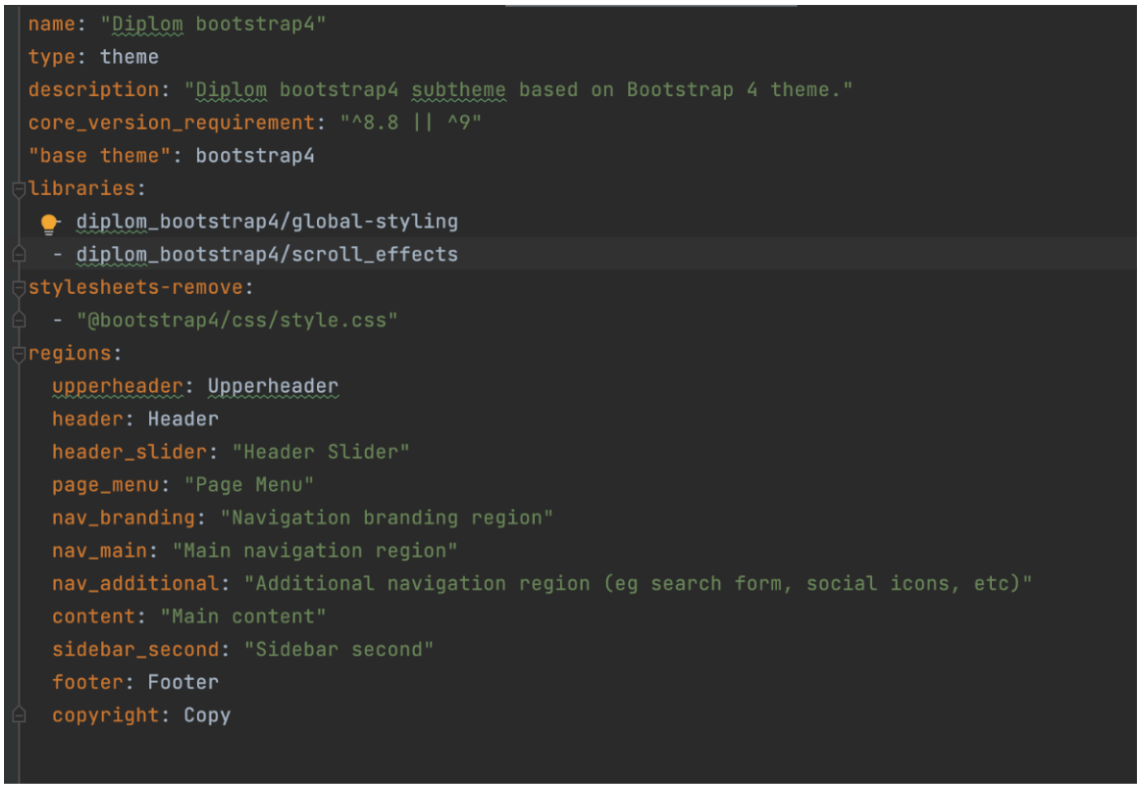

Рисунок 3.1 – Налаштування власної теми в основному конфігураційному файлі

Тут була підключена бібліотека з css стилями та JavaScript кодом за допомогою ключа libraries. Також, була ініцілізована стратегія з регіонами які будуть використані на сайті за допомогою ключа regions. Також варто зауважити, що тема залежить від версії ядра друпалу. Якщо ця версія несумісна з поточною версією сайту, тоді ця сабтема не буде працювати. Це зроблено для того, щоб не виникало фатальних помилок, через застарілі речі, потипу бібліотек в яких використовується код, який вже не підтримується.

В кожному регіоні може бути багато різних компонентів, це зручно, адже це може бути змінено в будь-який момент, наприклад, зміна розміщення блоків з адмін панелі, або додавання або видалення нових компонентів. Отже, кінцева структура сайту, буде виглядати згідно рисунку 3.2.

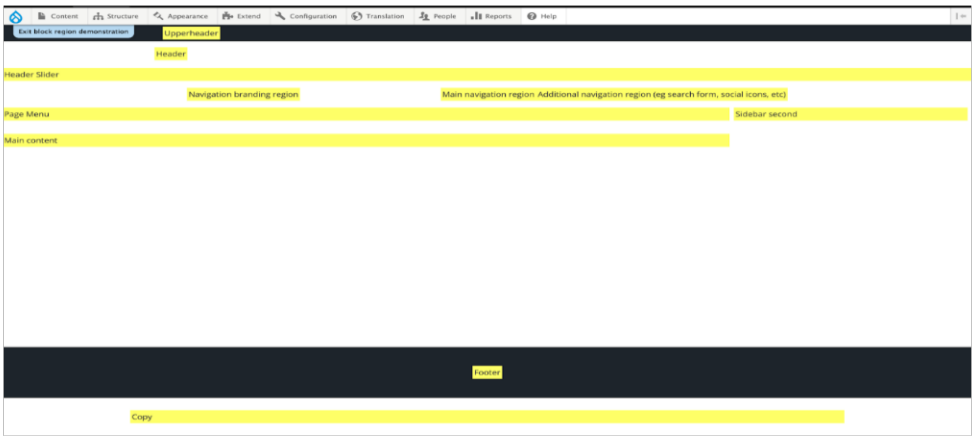

Рисунок 3.2 – Налаштування розміщення компонентів на сайті

Отже, для того щоб перевірити, чи ця сабтема працює, потрібно перейти в адміністративну панель та переконатись, що щойно створена і налаштована тема вже включена, і використовується на сайті.

3.2 Підключення API для отримання сторонніх даних

В кваліфікаційній роботі розроблена платформа, яка моніторить авіалінії Нідерландів, і також відслідковує рейси всіх літаків цих авіаланій. Для цього потрібно мати джерело, звідки будуть братись нові дані. В цьому випадку використано два API: Aviation Edge та Flight Radar.

Aviation Edge надає безліч можливостей. По-суті, це хаб для розробників для всіх авіаційних даних, від відстеження рейсів у реальному часі до актуальних розкладів аеропортів, маршрутів авіакомпаній та багато іншого. Ця API надає як і статичні дані, так і динамічні (рис. 3.3). Це саме те, що потрібно – отримати дані всіх авіалінії Нідерландів, та всіх аеропортів розташованих в Нідерландах. Статичні дані, хоч і статичні, але час від часу оновлюються.

Наприклад, назви аерпорту, або авіалінії можуть змінитись. Тому час від часу, навіть і статичні дані змінюються.

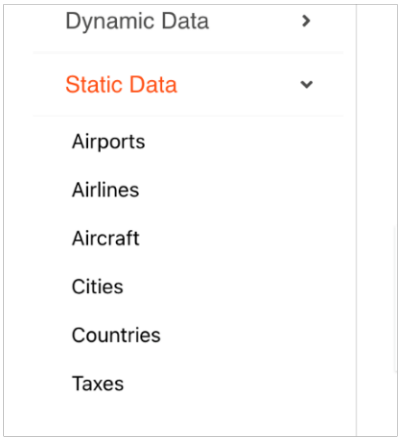

Рисунок 3.3 – Розділи з усіма статичними та динамічними типами даних [7]

Ще одна API - Flight Radar, оскільки Aviation Edge повільно оновлює даних. Тобто, воно може оновлювати дані не кожних 20 секунд, а кожних 5-10 хвилин. Оскільки, задача полягає в тому, щоб моніторити дані польотів, і переміщати літаки в реальному часі, 5-10 хв це досить довго. Тому, потрібен Flight Radar, для того щоб оновлювати дані польотів раз в 1-7 секунд.

Для підключення Aviation Edge до інформаційної системи, потрібно мати ключ. Ця API не є безкоштовною, тому для того щоб отримати API ключ, за допомогою якого будуть отримуватись дані, потрібно заплатити 7$ на місяць (рис. 3.4). Ця підписка надає 30 тисяч запитів, що є доволі непогано, виходячи з того, що будуть отримуватись і аналізуватись здебільшого статичні дані авіаліній та аеропортів. Тому, буде куплена підписка для отримання API ключа, який буде використано для отримання даних. Ключ є, тепер все що потрібно зробити, це побудувати GET запит в коді (рис. 3.4), який буде повертати дані з Aviation Edge. Цей запит буде виконуватись кожних три години, для отримання поновених даних, по-суті, це виконує проста крон функція (Додаток А). Тут використано найпростішу аутентифікацію між API та інформаційною системою,

оскільки ця інформаційна система не буде у просторах інтернету, то захищати нічого не потрібно.

За допомогою HTTP запиту та API ключа, отримується JSON дані, а функція мови програмування PHP присвоює ці дані змінній, для того щоб зберігати ці дані, та в подальшому мати можливість їх аналізувати.

```
function diplom_airline_cron() {
  $airline_json_data = file_get_contents( filename: 'https://aviation-edge.com/v2/public/airlineDatabase?key=b32028-a4ac39&codeIso2Country=NL');
  $airline_data = json_decode($airline_json_data,  associative: TRUE);
```

Рисунок 3.4 – GET запит з ключем до Aviation Edge, для отримання всіх авіаліній Нідерландів

Підключення Flight Radar API дещо відрізняється від Aviation Edge. По-перше, є можливість зробити певну кількість запитів безкоштовно. По-друге, ця API знаходиться на https://rapidapi.com (рис. 3.5). По-третє, як було зазначено раніше, цей API набагато швидше оновлює дані, а саме – кожної секунди. Отже, для підключення цієї API, потрібно просто зареєструватись на сайті https://rapidapi.com, підписатись на API, та отримати ключ.

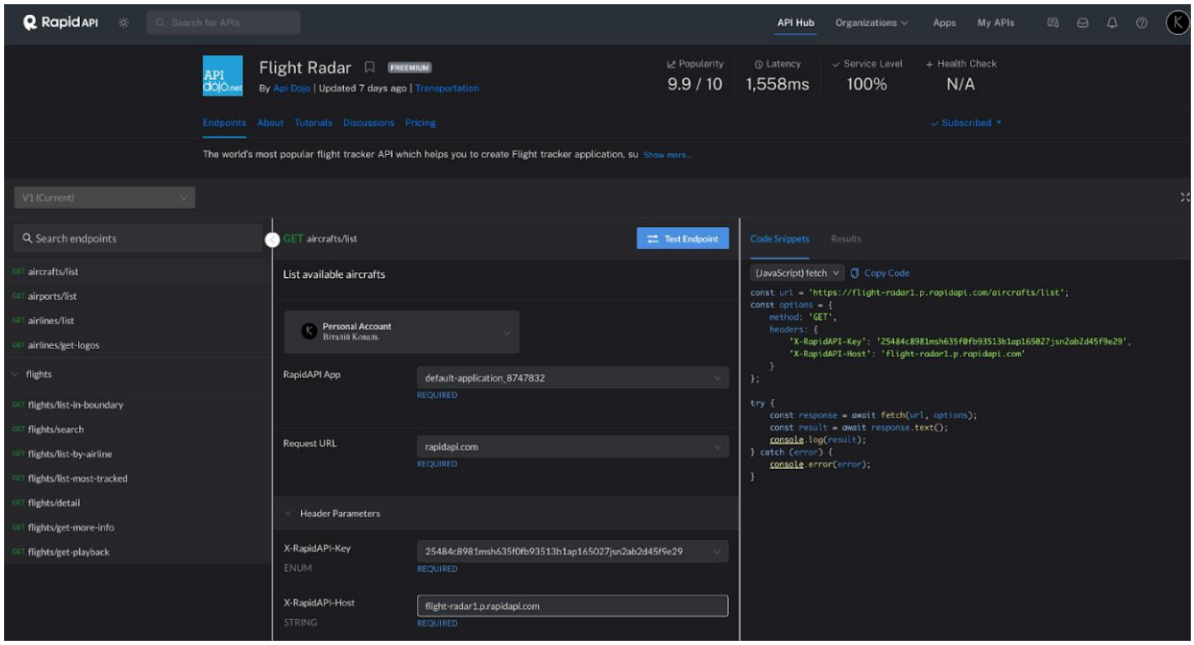

Рисунок 3.5 – Сторінка Flight Radar API на rapidapi сайті [8]

Дані з цього API використовуються в JavaScript коді (рис. 3.6), для моніторингу даних, а саме переміщення літаків, згідно координатів.

```
const url = 'https://flight-radar1.p.rapidapi.com/flights/list-by-airline?airline=' + drupalSettings.airline_id;
const options = {
  method: 'GET',
  headers: {
    'X-RapidAPI-Key': '25484c8981msh635f0fb93513b1ap165027jsn2ab2d45f9e29',
    'X-RapidAPI-Host': 'flight-radar1.p.rapidapi.com'
  }
};
```

Рисунок 3.6 – GET запит з ключем до Flights Radar, для отримання геоданих польотів конкретної авіаланії Нідерландів

На данний момент, було підключено до платформи два API, та наштований процес, для отримання даних у вигляді JSON, які надають змогу їх обробляти, аналізувати та моніторити. Є багато різних способів підключення API до сайтів. Є захищені через додаткові ключі і токени та є незахищені, які напряму просять доступ, як у цьому випадку. Оскільки, цей сайт не є опублікований в інтернеті, а лише використаний локально, тоді немає сенсу робити захищені запити до різних платформ.

3.3 Проектування архітектурних моделей та побудова структуризації бази даних

Створено програмним кодом дві власні сутності. Це «Airline» та «Flight» (Додаток Б). Ці сутності будуть пов’язані між собою. Оскільки, Airline може мати безліч польотів. Сутність авіалінії, буде мати певну кількість полів, такі як: IATA код, ICAO код, назва авіалінії і так далі. Кожне поле, це таблиця в базі даних, яка буде під’єднана до таблиці самої Airline за допомогою ідентифікатора. Таким чином, надається можливість отримувати дані по API, аналізувати їх, та за допомогою цих даних оновлювати, або створювати нові сутності авіаліній, вставляючи оброблені дані у поля цієї сутності.

Отже, було створено новий власний модуль для авіалінії, з назвою «Diplom Airline», та з машинним ім’ям «diplom_airline», це є свого роду ідентифікатором для пошукової системи друпалу. Оскільки, це модуль надаватиме нову сутність для авіалінії, то було створено нову директорію src/Entity та файл Airline.php в цій директорії. Якщо коротко, то цей файл відповідає за функціонал власної сутності – авіалінії. Була написана аннотація в цьому файлі та код для додавання полів які потребує сутність авіалінії. Важливо, щоб ці поля були ідентичними з даними які надходять з API. В підсумку структура файлу виглядає згідно рисунку 3.7. Потрібно зазначити, що цей файл може бути модифікованим в будь-який момент, це означає, що є змога розширити, додати, або змінити функціонал цієї сутності, без будь-яких проблем.

Виходячи з файлу, можна зазначити, що можна додати, або видалити поля цієї сутності, а саме у методі «baseFieldDefinitions». Також додавати нові методи, такі як «setTitle», який програмно оновлює ім’я авіалінії. Ці методи можна використати будь-де у коді програми, що є добре, адже це надає гнучкості, та не порушує правила SOLID. Також, додано код, який створює форму для редагування авіалінії, хоч, і будуть створюватись авіалінії автоматично з даними які будуть надходити з API, але потрібно мати форму, для того щоб редагувати ці авіаланії.

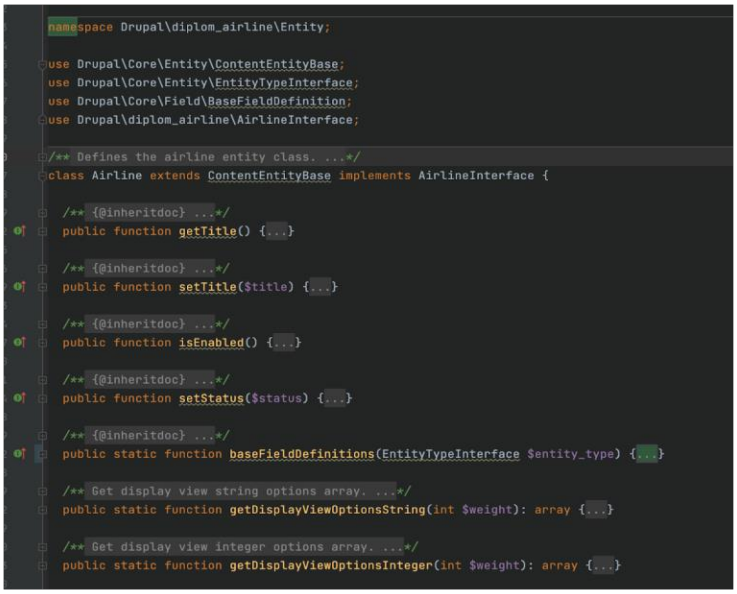

Рисунок 3.7 – структура файлу Airline.php який описує сутність Airline

Доступ до форми надано лише для ролі адміністратора. Для, того щоб бачити створені авіаланії в адміністративній панелі, створено список авіаланій за допомогою «ListViewBuilder » класу у файлі «AirlineListBuilder».

Також створено і інші файли, такі як «diplom_airline.module» де виконуються всі базові функції (Додаток В) та «diplom_airline.permissions.yml» для створення дозволів на перегляд, редагування якихось речей. Також було створено diplom_airline.routing.yml для налаштування системи шляхів. Структура кастомного модулю є такою (рис. 3.8).

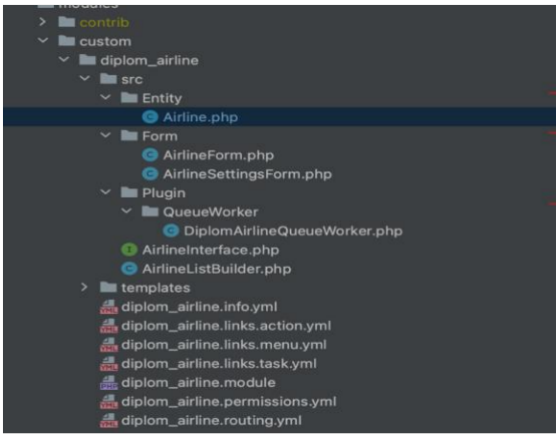

Рисунок 3.8 – Структура модулю diplom_airline

Схема сутності «Airline» є наступною (рис. 3.9). Якщо size дорівнює нулю, або ж status є «historical», тоді ми не створюється новий об’єкт сутності, оскільки не потрібна історична авіалінія, або авіалінія без літаків.

Отже, є поля та таблиці в базі даних, є форма для редагування, і найголовніше сам тип сутності авіалінії, який дає змогу створювати нові об’єкти авіаліній, із даними які зберігаються в полях.

Наступним кроком є створення нового типу сутності – Flights. Він має кілька полів: посилання на сутність авіалінії, довгота, широта і статус. Також, цей тип сутності не має форми для створення або редагування. Оскільки об’єкти перельотів будуть створюватись і оновлюватись автоматично через код.

В результаті тип сутності Flights є дочірнім типом Airline. В такому випадку їх може бути безліч, і це логічно, тому що одна Airline може мати багато

польотів, оскільки має багато літаків. По-суті, об’єкти польотів є тимчасові, адже видаляються з бази даних, коли якийсь літак приземлиться.

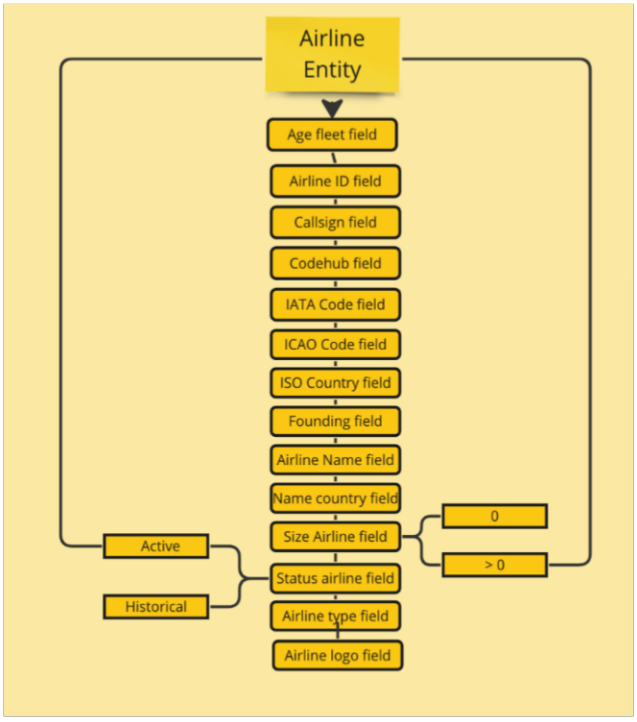

Рисунок 3.9 – Схема моделі Airline зі всіма полями

3.4 Створення функціоналу для аналізу даних

Було визначено, які дані будуть відслідковуватись. Flight Radar API оновлює дані щосекунди, і це підходить, адже запит виконується кожних 7 секунд, для отримання даних польоту в реальному часі, а потім за допомогою цих даних, буде переміщуватись літак, згідно координатів які надходять разом з цими даними. Для цього, потрібно розробити функціонал для моніторингу даних (Додаток Г).

Для початку, було створено функцію «hook_cron» у власному модулі. В цій функції робиться GET запит до «AviationEdge API», аби отримати дані всіх авіаліній Нідерландів. Після отримання цих даних, потрібно конвертувати json в масив з елементами кожної авіалінії (рис. 3.10). Тут виконується запит, для отримання даних. Спочатку йде аналіз цих даних, тобто, відбувається відсів авіаліній, які є історичними, що означає що вони не є активними, та ті які мають

значення ключа розміру нуль, що означає, що ця авіалінія не працює, оскільки немає літаків.

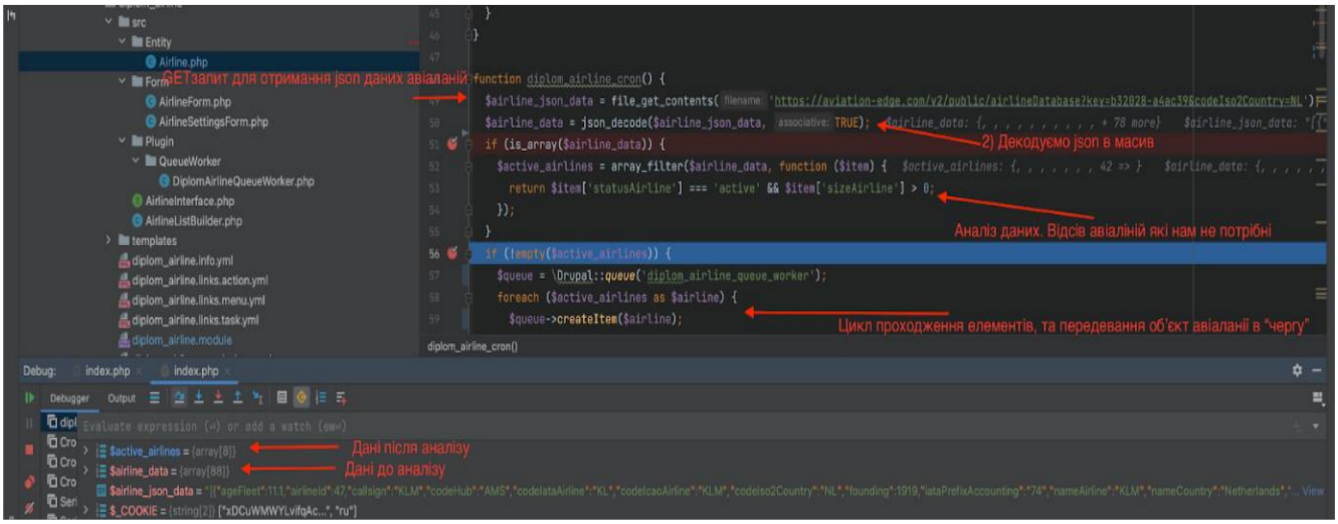

Рисунок 3.10 – Обробка та аналіз даних авіаліній

Наприкінці функції використовується метод createItem(), це означає що передається масив з даними авіалінії в чергу для обробки. В друпалі за це відповідає плагін «queueWorker», цей плагін оброблює об’єкти з даними поступово, тобто, опрацьовує один об’єкт, і переходить до наступного. Це відрізняється від звичайного циклу тим, що тут на опрацювання кожного об’єкту виділяється певний проміжок часу, зазвичай це 30 секунд. Це дає гарантію, що якщо даних буде дуже багато, то система витримає обробку та аналіз цих даних, а в інакшому випадку, просто пропустить ці об’єкти та повідомить в чому проблема. Далі, в черзі, отримується цей масив з даними, та на основі цих даних, створюється об’єкт авіалінії (рис. 3.11). На 61 стрічці в коді є перевірка чи вже існує авіалінія в базі данних з таким ID, якщо так, то пропускається ітерація, якщо ні, то створюється авіалінія з даними які обробляються.

Тепер є сутності авіаліній які безпосередньо належать до Нідерландів. Вони будуть оновлюватись кожних 24 години, якщо є якісь зміни. В цьому випадку, зміни будуть дуже рідко, адже це майже статична інформація. Тобто, максимум що може змінитись це ім’я, або статус. Тому по крону, отримуються

найсвіжіші дані авіаліній, і якщо є якісь зміни між старими даними і тими які тільки були отримані, то відбувається оновлення значення авіаліній в базі даних.

```php
public function processItem($item) {
  $airline_storage = $this->entityTypeManager->getStorage('airline');

  if (!empty($item['airlineId'])) {
    $airline = $airline_storage->getQuery()
      ->accessCheck(FALSE)
      ->condition('airline_id', $item['airlineId'])
      ->execute();

    if (!empty($airline)) {
      return;
    }

    $airline_date = Airline::create([
      'age_fleet' => $item['ageFleet'],
      'airline_id' => $item['airlineId'],
      'callsign' => $item['callsign'],
      'code_hub' => $item['codeHub'],
      'code_iata_airline' => $item['codeIataAirline'],
      'code_icao_airline' => $item['codeIcaoAirline'],
      'code_iso_country' => $item['codeIso2Country'],
      'founding' => $item['founding'],
      'iata_prefix_accounting' => $item['iataPrefixAccounting'],
      'name_airline' => $item['nameAirline'],
      'name_country' => $item['nameCountry'],
      'size_airline' => $item['sizeAirline'],
      'status_airline' => $item['statusAirline'],
      'type' => $item['type'],
    ]);
    $airline_date->save();
  }
}
```

Рисунок 3.11 – Створення ентіті об’єкт Airline на основі отриманих даних

Для того, щоб не заходити постійно в базу даних, і шукати по таблицях значення, є можливість перевірити авіалінії в адміністративної панелі як і проілюстровано на рисунку 3.12. Тут є список всіх авіаліній, який адміністратор може змінити прямо з цієї панелі, видалити, або додати нові авіалінії вручну.

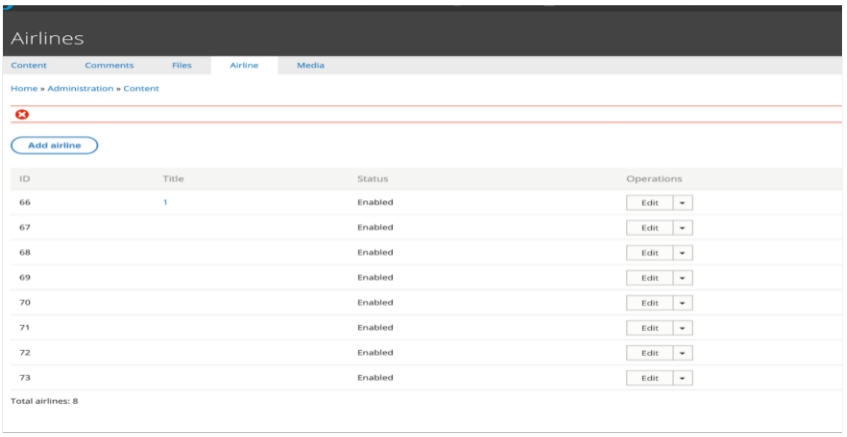

Рисунок 3.12 – Список створених авіаліній на основі проаналізованих даних з API

3.5 Створення функціоналу для моніторингу авіаційних об’єктів на основі проаналізованих даних

У базі даних є авіалінії, які будуть оновлюватись автоматично час від часу. Це всього лиш об’єкти, тому потрібно налаштувати їхній вивід. З точки зору користувача, було б зручно бачити список всіх авіаліній у вигляді тизерів. На цьому тизері, можна побачити короткий огляд авіалінії, кнопка з посиланням на повний огляд авіалінії, та найважливіше – посилання на Flight Tracker. Тобто, кожна авіалінія буде мати посилання на сторінку з гугл картою, де в реальному часі будуть переміщатись літаки цієї авіалінії згідно координатів які надходитимуть з Flight Radar API. Для початку було створено сторінку з тизерами всіх авіаліній. Для цього використовується Views API, яке працює зсередини CMS Drupal. Для створення цієї сторінки з адміністративної панелі, можна використати шлях – Structure – Views – Add View.

Для відображення елементів на сторінці було налаштовано запит до бази даних, та вивід даних (рис. 3.13).

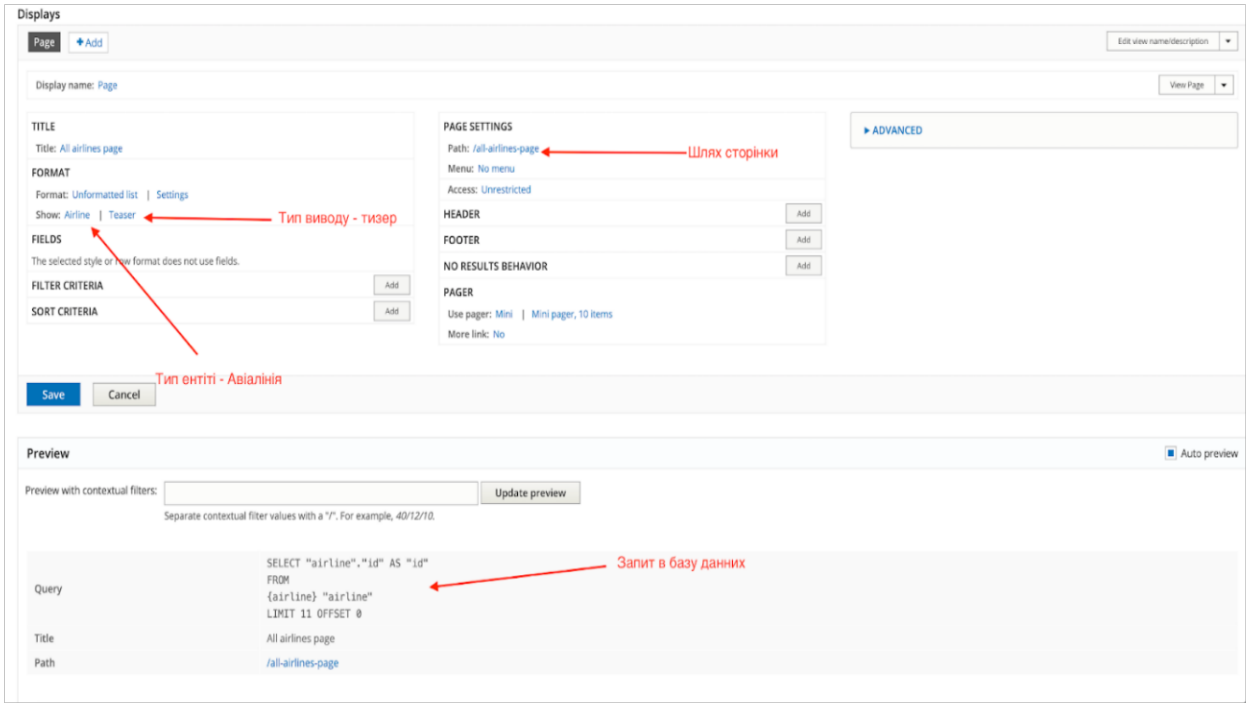

Рисунок 3.13 – Налаштування сторінки з усіма авіалініями за допомогою Views API

Тобто, вибрано тип сутності, в цьому випадку це авіалінія. Потім було включено та налаштовано тип виводу, в цьому випадку це тизер, та прописано шлях до сторінки.

В кінці формується SQL запит до бази даних. Варто, зазначити, що цей запит до бази даних формується автоматично на основі налаштованих конфігурацій. Проте, друпал надає змінювати цей запит у коді, якщо це потрібно, оскільки «Views API» не може покрити абсолютно всі потреби. Проте, в цьому випадку, запит змінювати не потрібно.

Для налаштування тизеру в друпал, були використані певні конфігурації. За замовчуванням, друпал уже надає цей тизер, залишається тільки налаштувати його, а саме додати поля, які будуть відображатись на цьому тизері (рис. 3.14). З огляду на рисунок, можна побачити дві секції – верхню, і нижню «disabled». Це означає, що всі поля які знаходяться у верхній секції, відображаються на тизері, а ті поля які знаходяться в нижній секції, не матимуть можливість відображатись на тизері. Це зроблено для того, щоб мати можливість гнучко оновлювати тизер, прямо з адміністративної панелі. Кожне поле можна налаштувати, а саме: показувати позначку чи ні, та вибрати унікальний вивід для поля.

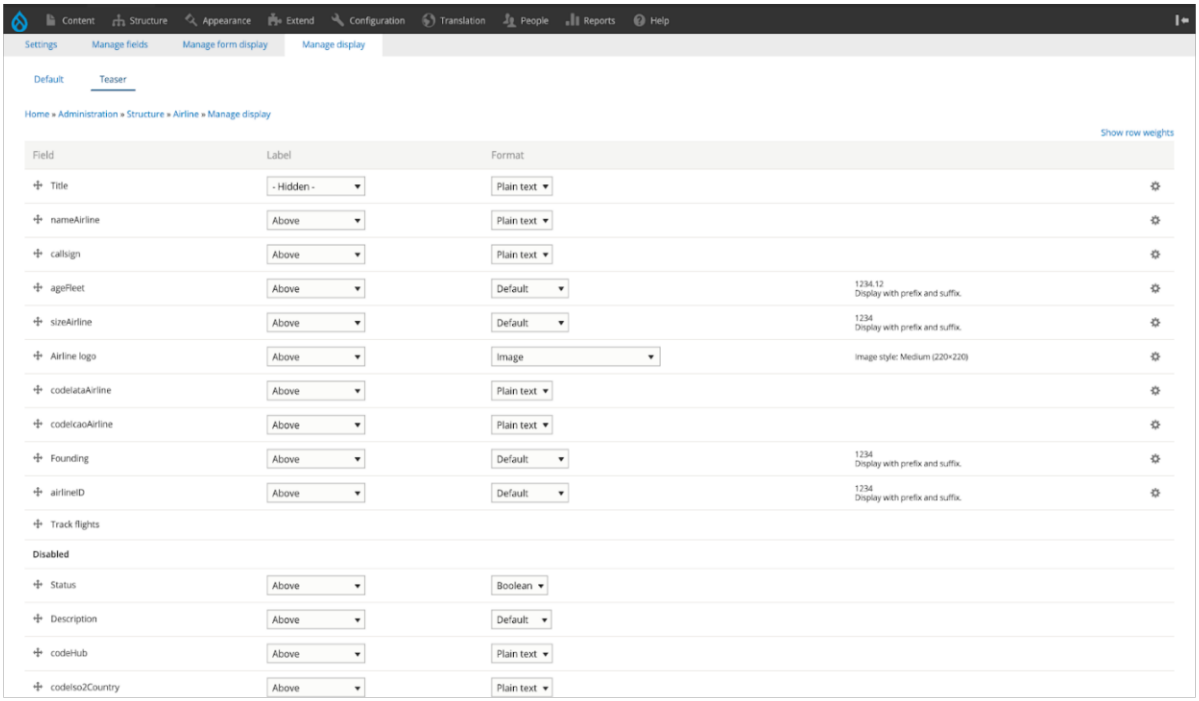

Рисунок 3.14 – Налаштування тизеру для об’єкту авіалінії

Після переходу на сторінку за шляхом /all-airlines-page (рис. 3.15), можна побачити всі авіалінії у вигляді тизерів. Посилання «Track Flights» веде на сторінку з гугл картою, яка буде моніторити дані, і переміщати літаки згідно координатів.

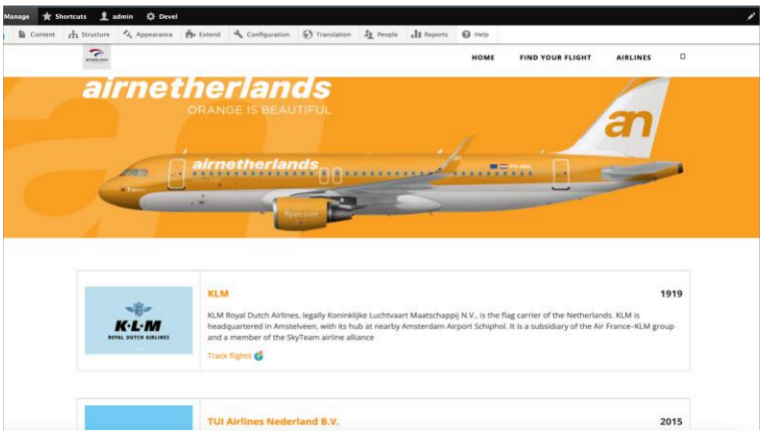

Рисунок 3.15 – Сторінка з оглядом всіх авіаланій

Оскільки тепер побудована сторінка зі всіма авіалініями та налаштований тизер з кнопкою кожної авіалінії, то тепер час зробити і саму сторінку з гугл картою. Перш за все, для того щоб створити якусь сторінку програмно, можна використовувати різні готові методи, які надають базу для створення сторінки. Ця сторінка зроблена через Controller. Для цього було створено власний модуль з назвою flights та новий об’єкт роутингу в flights.routing.yml файлі (рис. 3.16).

```
flights.tracker_by_airline:
  path: 'find-your-flight/{airline_id}/flight-tracker'
  defaults:
    _title: 'Flights Tracker'
    _controller: '\Drupal\flights\Controller\FlightsTrackerByAirlineController::build'
  requirements:
    _permission: 'access content'
```

Рисунок 3.16 – Роутинг який будує сторінку з гугл картою для моніторингу даних

Один з ключових аспектів який показаний на рисунку – path ключ. Це шлях, по якому відкривається сторінка. Динамічний параметр «airline_id» може змінюватись, тому, цей роутинг генерує багато сторінок, різний «airline_id» –

різна сторінка. Тобто, це свого роду – замінювач. Це підходить по тій причині, що на сайті є багато авіаліній, але структура сторінки для них повинна бути одна, єдина відмінність – літаки які будуть відображатись на карті, тому що кожна авіалінія має різні літаки. Наступний важливий момент – ключ «_controller». Саме там прописується шлях до коду, який відповідає за виведення вмісту сторінки. Можна помітити, що є метод build(), що знаходиться в класі «TrackerByAirlineController». Саме цей метод повертає відповідь, та сигналізує як саме повинна будуватись стоірнка. Створивши роутинг, що по-суті є каркасом сторінки, який також з'єднує окрему сторінку з середовищем самого сайту, потрібно додати код та зробити певні кроки, для візуалізації вмісту сторінки:

– створено файл «FlightsTrackerByAirlineController.php» в src/Controller директорії. Це реєстрація цього контроллеру в пошуковому механізму друпалу, інакше цей контроллер просто не буде знайденим, після чого буде фатальна помилка, на будь-якій сторінці сайту;

– у файлі контроллеру ініцілізовано клас, який наслідує ContollerBase, щоб мати вже всі стандартні методи;

– додано власний метод build(), та побудовано логіку візуалізації сторінки, іншими словами – побудова відповіді, на запит;

– додано код згідно рисунку 3.17, в цьому коді метод build() приймає аргумент ідентифікатору, а це саме той динамічний параметр, про який було згадано раніше, за допомогою цього ідентифікатора, загружається об'єкт авіалінії з бази даних використовуючи id, після чого отримати назву авіалінії, та передати у масив відповіді;

– ключова річ в методі build – це все що є в тілі return., це асоціативний масив, який по-суті будує відтворює сторінку;

– ключ theme – назва шаблону, який будує DOM сторінки, за допомогою шаблонізатора twig, що в кінцевому етапі трансформується в розмітку HTML.

Ключ airline_name – змінна, що передає назву поточної авіалінії, потрібна для побудови назви сторінки. Ключ attached – прикріплює бібліотеку з каскадними стилями та JavaScript скриптами до сторінки. Саме в цій бібліотеці

додано весь основний код для моніторингу рейсів різних авіаліній. Останній ключ drupalSettings – передає ідентифікатор поточної авіалінії до клієнт-частини, для побудови запиту. Тобто, це означає, що цей ідентифікатор передано до JavaScript коду, який безпосередньо відіграє роль у побудові посилання для запиту до API.

```php
/**
 * Builds the response.
 */
public function build(string $airline_id = NULL) {
  $airline_storage = $this->entityManager->getStorage('airline');
  $airlines = $airline_storage->loadByProperties(['code_iata_airline' => $airline_id]);
  $airline = reset( &array: $airlines);
  if ($airline instanceof AirlineInterface) {
    $airline_name = $airline->get('name_airline')->getString();
  }

  return [
    '#theme' => 'flight-tracker-page',
    '#airline_name' => !empty($airline_name) ? $airline_name : NULL,
    '#attached' => [
      'library' => [
        'flights/flights_library',
      ],
      'drupalSettings' => [
        'airline_id' => $airline_id,
      ],
    ],
  ];
}

}
```

Рисунок 3.17 – Код який відповідає за побудову Flight Tracker сторінки

Для передачі змінних до клієнт-частини, було додано функцію hook_theme в flights.module файлі (рис. 3.18). В цій функції відбувається реєстрація твіг-шаблону, щоб система Drupal знала, що такий шаблон існує, та з якими змінними і в якому саме місці цей шаблон потрібно викликати.

```php
/**
 * Register template files.
 */
function flights_theme($existing, $type, $theme, $path):array {
  return [
    'flight-tracker-page'  => [
      'variables' => [
        'airline_name' => NULL,
      ],
    ],
```

Рисунок 3.18 – Код який реєструє шаблон в системі Drupal для відтворювання сторінки

Після підготовки всього необхідного з «Back-end» частини, було побудовано структуру сторінки в самому шаблоні. Спочатку створено шаблон з назвою flight-tracker-page.html.twig в директорії /templates, а потім додано розмітку сторінки за допомогою шаблонізатору (рис. 3.19). В шаблоні прописується назва сторінки, та додається пустий елемент з flights-map ідентифікатором. Для того щоб вивести гугл карту на сторінці, було підключено сторонню бібліотеку.

```
<body>
  <div class="flights-map-header">
    {% trans %}
      Track flights of {{ airline_name }} airline.
    {% endtrans %}
  </div>
<div id = "flights-map"></div>
</body>
```

Рисунок 3.19 – HTML розмітка сторінки Flight Tracker

Для цього є бібліотека – Leaflet, яка вже інтегрована в Drupal, та успішно використовується багатьма сайтами. Все що потрібно було зробити, це отримати файли бібліотеки на оригінальному сайті, та перенести ці файли до проекту, а потім підключити їх до внутрішньої бібліотеки flights/flights_library. Після переходу на сторінку за шляхом /find-your-flight/ams/flight-tracker (рис. 3.20), потрібно переконатись, що все працює і карта є.

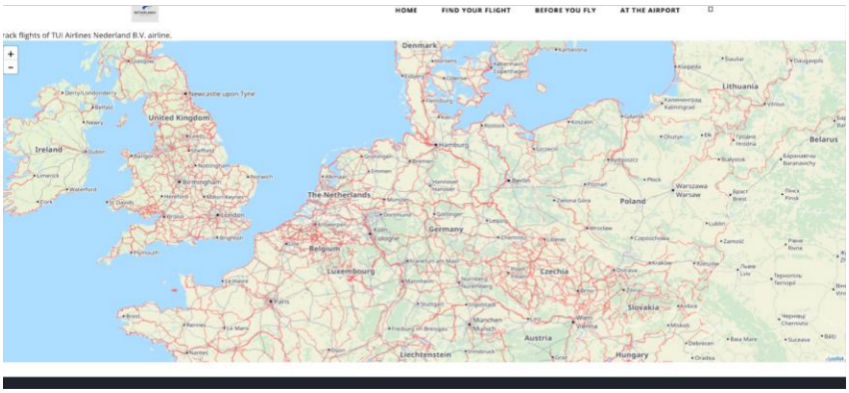

Рисунок 3.20 – Виведення гугл-карти за допомогою сторонньої бібліотеки Leaflet

Для розробки моніторингу даних було додано JavaScript (Додаток Д). Для цього потрібно отримувати дані через API кожних 5 секунд. Піся чого, треба проаналізувати їх, перетворити ці дані в об’єкти в JavaScript коді і переміщати літаки за рахунок цих об’єктів. Весь цей процесс, відбувається за допомогою JS коду, тому потрібно створити новий файл з JavaScript розширенням, та підключити цей файл до бібліотеки, щоб сторінка оброблювала цей код.

Заміна стандартної гугл-карти яку надає Leaflet на ту, яка потрібна, тобто, кастомізована. Цей код (рис. 3.21), створює нову карту, з іншими розмірами, та підписом.

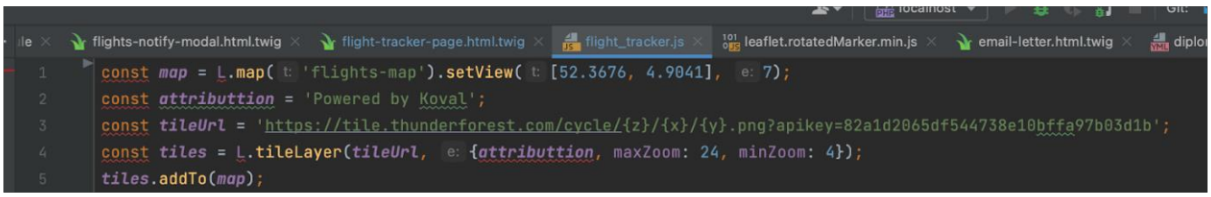

Рисунок 3.21 – код для кастомізації гугл-карти

Потрібно було створити основну асинхронну функцію, в якій будуть відбуватись запити для отримання даних. Далі було ініцілізовано функцію fetchData(url, options), яка приймає два аргументи – перший з яких це посилання для створення запиту, другий це додаткові опції, наприклад, API ключ, за допомогою якого отримується об’єкт з даними (рис. 3.22).

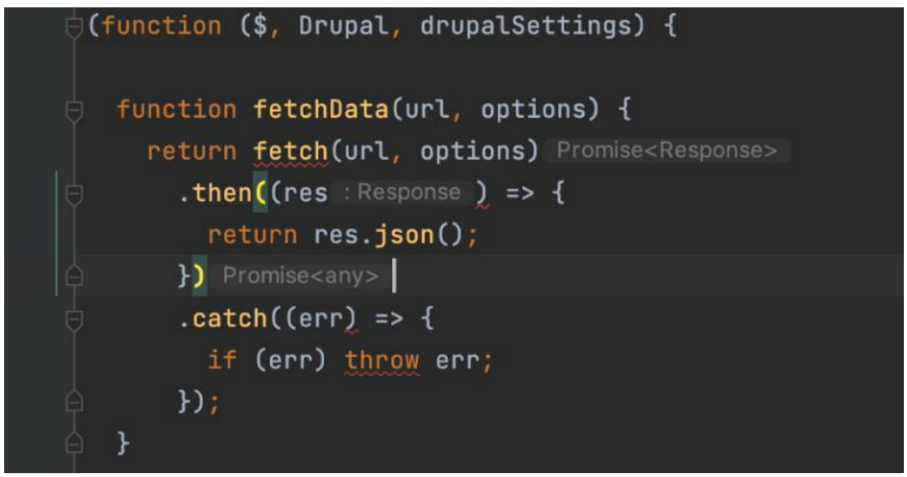

Рисунок 3.22 – Функція для отримання даних по посиланню

Було створено нову функцію – putStatesFlighs (рис. 3.23), яка приймає два аргументи, перший – об'єкт гугл-карти, другий – об'єкт з мітками, в цьому випадку це літаки. Далі, створюється об'єкт для іконки літаку, за це відповідає константа planeIcon. Ну, а нижче будується посилання для отримання нових даних реальних польотів поточної авіалінії.

```
function putStatesFlighs(map, markers) {
  const planeIcon = L.icon({
    iconUrl: 'https://kovalskiy266.github.io/public_flights_data/plane.png',
    iconSize: [24, 50],
    iconAnchor: [25, 16]
  });

  const url = 'https://flight-radar1.p.rapidapi.com/flights/list-by-airline?airline=' + drupalSettings.airline_id;
  const options = {
    method: 'GET',
    headers: {
      'X-RapidAPI-Key': '25484c8981msh635f0fb93513b1ap165027jsn2ab2d45f9e29',
      'X-RapidAPI-Host': 'flight-radar1.p.rapidapi.com'
    }
  };
```

Рисунок 3.23 – побудова запиту для отримання данних реальних польотів поточної авіалінії

Тут будується динамічне посилання за допомогою ідентифікатора авіалінії. Для того, щоб отримувати дані польотів конкретної авіаланії, потрібно використовувати параметр «airline» в посиланні, і підставити туди значення. В даному випадку це IATA код, адже кожна авіалінія має ICAO та IATA код. Саме за цими параметрами відбувається пошук даних у сховищах для аналізу даних. Для того щоб переконатись, як ці дані отримуються, можна використати інструмент Postman, там можна ввести адресу для отримання даних, і подивитись дані у вигляді масиву (рис. 3.24). Отже, можна побачити, що тут будується посилання для отримання даних польотів конкретної авіаланії. Потім дані приходять у JSON форматі, та у масиві aircrafts. Кожний об'єкт у масиві – окремий рейс. Там є наступні дані: ідентифікатор літаку, ідентифікатор рейсу, координати польоту, код аеропорту куди прямує літак, код аеропорту звідки вилетів літак, година відльоту та приземлення літаку, кількість пассажирів та інше.

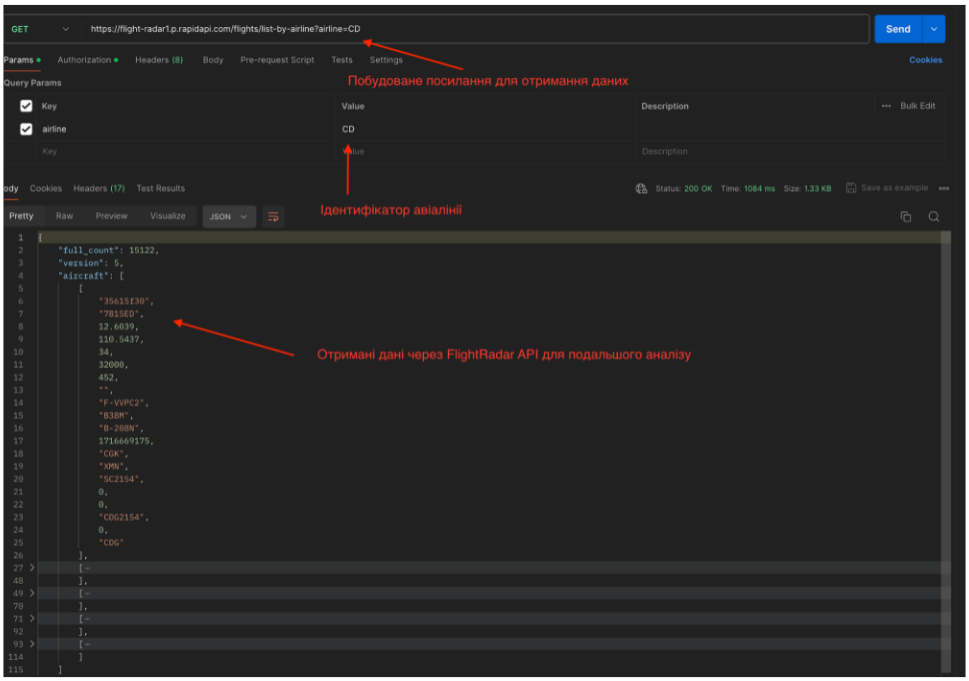

Рисунок 3.24 – Приклад запиту даних у інструменті Postman

В загальному для аналізу та моніторингу даних, знадобляться наступні дані: година приземлення та відльоту літака, координати та коди аеропортів. Отримавши код аеропортів, є можливість зробити новий запит для отримання детальної інформації про аеропорти, але тепер використовуючи «AviationEdge API».

Тепер коли є дані у масиві, потрібно проаналізувати їх, та зробити певні маніпуляції. Потрібно через цикл всіх об’єктів отримувати координати літака, та розмістити літак на гугл-карті, цикл починається на 244 рядку у коді (рис. 3.25). Після чого, проаналізувати напрямок літака, і повернути іконку літака згідно цього напрямку. Далі йде аналіз звідки цей літак вилетів, та куди він прямує. Після аналізу отримуються коди аеропортів, та йде новий запит для отримання детальних даних аеропортів. Далі йде перевірка, чи розміщений вже літак на гугл карті, якщо так, тоді просто оновлюються координати, і тим самим переміщується літак. Проте, якщо це новий рейс, тоді додається новий об’єкт на гугл мапі. Ця ж функція з кодом викликається знову і знову з певним інтервалом, в даному випадку це 7 секунд.

```
fetchData(url, options).then(function (states) {
  if (states.aircraft) {
    states.aircraft.forEach((state) => {
      if (state[0]) {
        let lat= state[2],
          lng = state[3],
          angle = Math.ceil(state[4]);
        if (state[12]) {
          putStatesAirport(map, airportMarkers, state[12]);
        }
        if (state[13]) {
          putStatesAirport(map, airportMarkers, state[13]);
        }
        if (markers[state[0]]) {
          markers[state[0]]
            .setLatLng([lat, lng], { icon: airportIcon, rotationAngle: angle,  draggable: true }).bindPopup( t: '<h3>' + state[12] + ' ----> ' + state[13] + '
        } else {
          markers[state[0]] = L.marker( t: [lat, lng],  e: { icon: airportIcon, rotationAngle: angle, draggable: false }).bindPopup( t: '<h3>' + state[12] + ' -
          markers[state[0]].addTo(map);
        }
      }
    });
  }
  setTimeout(() => putStatesFlighs(map, markers), 7000);
});
}
```

Рисунок 3.25 – Код для переміщення літаків на мапі згідно нових даних

Це називається рекурсія, вона потрібна для того, щоб постійно оновлювати дані, та відповідно, і самі літаки на карті, що є і свого роду – моніторингом, оскільки, в цьому випадку йде моніторинг польотів певної авіалінії в реальному часі.

Коли є літаки, варто подумати і про об’єкти аеропортів. Для отримання аеропортів використовуються статичні дані з AviationEdge. Проте, потрібно підключитись до неї, та побудувати посилання для отримання даних конкретного аеропорту. Логіка виведення аерпортів на мапу майже ідентична до логіки з літаками. Так само отримуються координати, аналізуються та виводяться на карту. Якщо ж знову використати інструмент PostMan, то можна побачити наступне: координати розташування аеропорту, різні коди, назва країни та аеропорту (рис. 3.26).

В даному випадку одні дані залежать від аналізу інших. Наприклад, дані аеропортів залежать від аналізу польотів. Оскільки, отримавши дані рейсів, потрібно проаналізувати, що це за рейс, звідки він вилетів і куди прямує, і на основі цього аналізу отримується код аеропортів, після чого дані самих аеропортів.

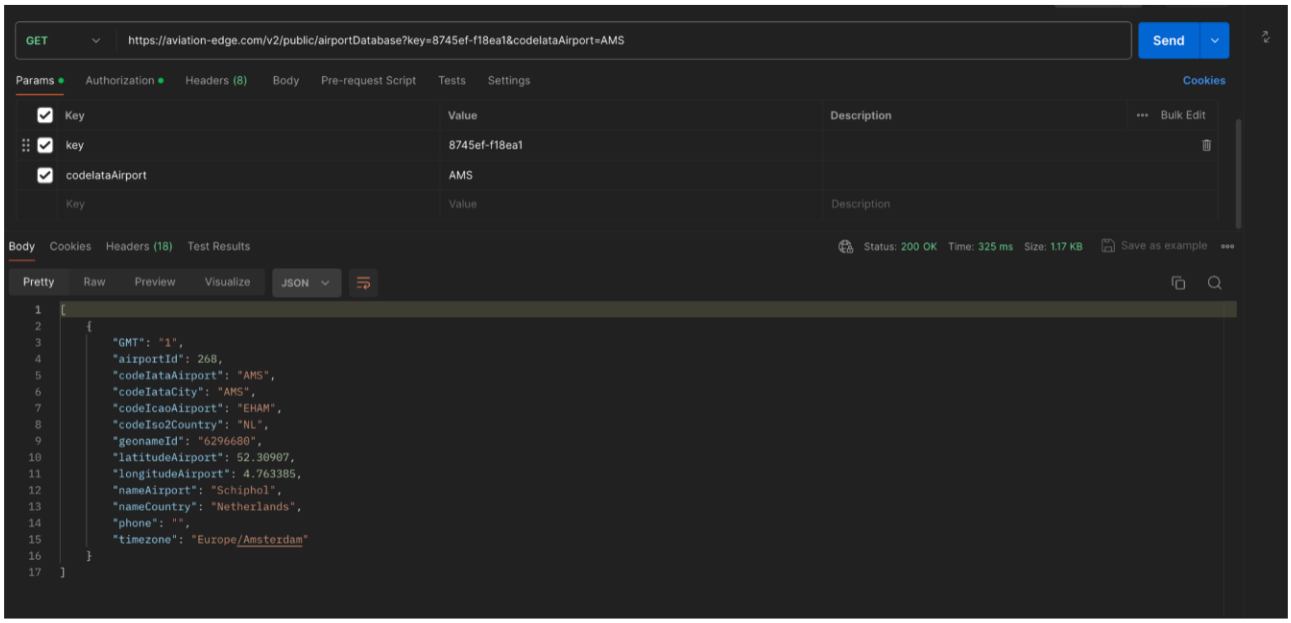

Рисунок 3.26 – Запит до AviationEdge для отримання даних аеропорту

Рисунок 3.27 описує весь шлях аналізу і моніторингу даних в цій інформаційній системі.

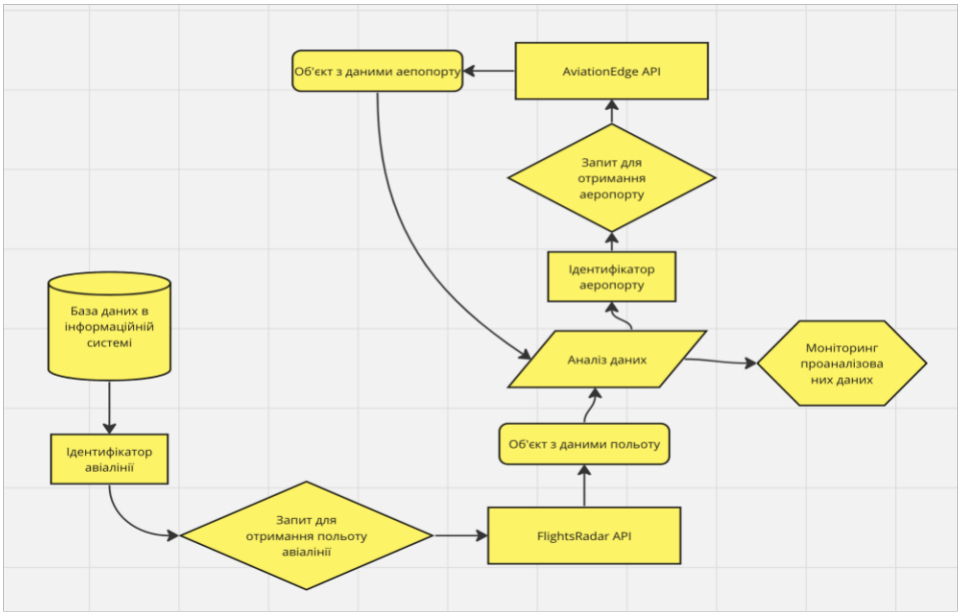

Рисунок 3.27 – Схема аналізу і моніторингу даних

В даному випадку візуалізація аеропортів і літаків відбувається асинхронно. Можна зазначити, що візуалізація аеропортів є сталою, оскільки логічно, що аеропорт нікуди не переміститься на мапі, бо це є будівля. Тепер, якщо перейти на сторінку з гугл мапою, можна побачити різні об’єкти, статичні

і динамічні (рис. 3.28). Відбувається запит на отримання початкових даних, потім йде аналіз, а потім йде моніторинг цих даних, за рахунок динамічного переміщення літаків згідно координат.

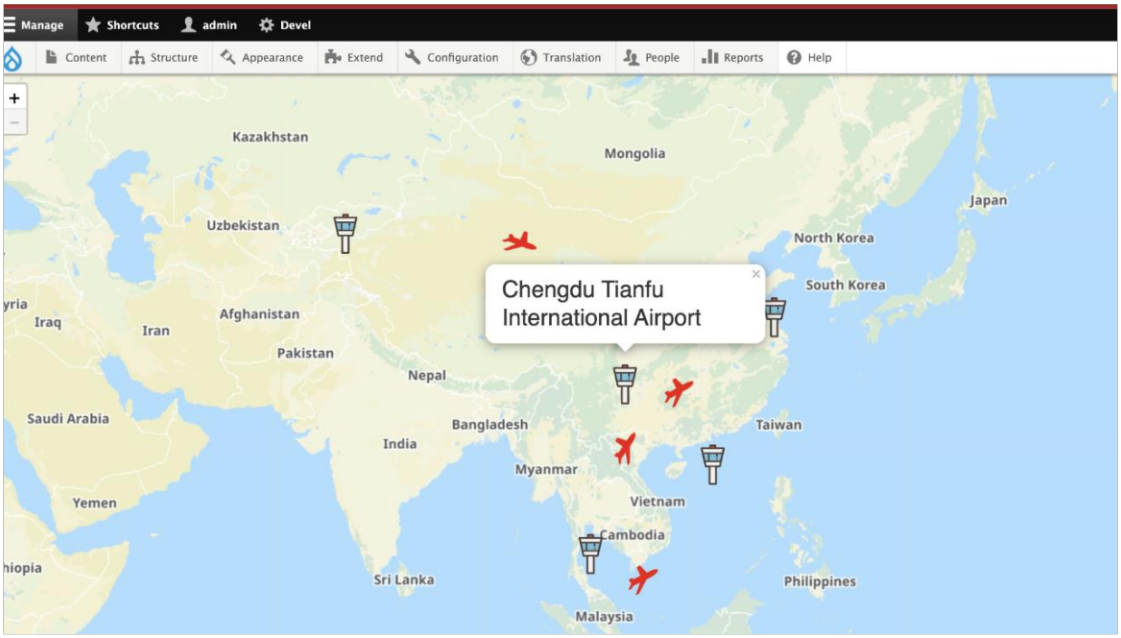

Рисунок 3.28 – Моніторинг польотів у реальному часі

Також, є можливість переглянути кожний об’єкт, просто натиснувши на нього. Наприклад, назва аеропорту, як і показано на рисунку 3.28.

В загальному, аналіз і моніторинг залежать від даних які надходять з API. Дані з корпоративних ерсурсів отримуються постійно, а потім йде аналіз – що це за авіалінії, чи вони актуальні, скільки літаків має та чи інша авіалінія, з якими аеропортами співпрацює тощо. На основі проаналізованих даних, відбувається фільтрація по аеропортам та авіаланіям, та паралельно йде моніторинг даних, за допомогою мапи сервісу «Google» та відслідковування рейсів в реальному часі.

ВИСНОВКИ

У кваліфікаційній роботі побудовано функціонал для аналізу та моніторингу даних з авіаційних ресурсів.

В результаті виконання роботи виконано завдання:

– проаналізовано технології для моніторингу корпоративних даних та визначено систему керування вмістом «Drupal» та шаблонізатор «Twig» для побудови веб-інтерфейсу;

– досліджено прикладні програмні інтерфейси для отримання авіаційних даних та вибрано «AviationEdge» та «FlightRadar» для отримання даних з віддалених сховищ;

– визначено DDEV, Docker, Drush та Composer як інструменти для роботи з локальним середовищем. Проаналізовано та вибрано мови програмування для побудови функціоналу, а також утиліти для забезпечення продуктивності роботи веб-інтерфейсу;

– розроблено архітектуру функціоналу на основі вибраних інструментів та програмно спроектовано моніторинг та аналіз ресурсів;

– протестовано роботу веб-інтерфейсу і забезпечено його злагоджену роботу.

Отже, кваліфікаційна робота бакалавра виконана в повному обсязі, відповідно до теми та мети роботи. Поставлені цілі досягнуті.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна сторінка Drupal. Open Source CMS Drupal.org. URL: https://www.drupal.org/ (дата звернення: 08.02.2024)

2. Live Flight Tracker - Real-Time Flight Tracker Map. Flightradar24. URL: https://www.flightradar24.com/ (дата звернення: 09.02.2024)

3. Благодійний фонд «Повернись живим» - допомога ЗСУ. URL: https://savelife.in.ua/ (дата звернення: 01.03.2024)

4. Free & Premium Aviation API - Aviation database and API. URL: https://aviation-edge.com/premium-api/ (дата звернення: 04.03.2024)

5. Пошук модулів. Розділ 11. Розширення. URL: https://www.drupal.org/uk/docs/user_guide/uk/extend-module-find.html (дата звернення: 08.03.2024)

6. Завантаження та розширення. Menu Item Extras. URL: https://www.drupal.org/project/menu_item_extras (дата звернення: 12.03.2024)

7. Developers – Aviation database and API. URL: https://aviation-edge.com/developers/ (дата звернення: 14.03.2024)

8. Flight Radar API Documentation. RapidAPI. API Hub - Free Public & Open Rest APIs. URL: https://rapidapi.com/apidojo/api/flight-radar1/ (дата звернення: 14.03.2024)

9. Async function in development – JavaScript documentation. URL: https://developer.mozilla.org/enUS/docs/Web/JavaScript/Reference/Statements/async_function (дата звернення: 18.02.2024)

10. CMS Quickstarts – Drupal DDEV Documentation. URL: https://ddev.readthedocs.io/en/latest/users/quickstart/#drupal (дата звернення: 22.02.20)

11. Docker overview. Docker Documentation. URL: https://docs.docker.com/get-started/overview/ (дата звернення: 24.02.2024)

12. Documentation – Leaflet – a JavaScript library for interactive maps. URL: https://mourner.github.io/Leaflet/reference.html (дата звернення: 01.03.2024)

13. GitHub Documentation. URL: https://docs.github.com/en (дата звернення: 04.03.2024)

14. Installing and Uninstalling Modules with Composer. URL: https://modulesunraveled.com/drupal-8-composer-and-configuration-management/installing-and-uninstalling-modules-composer (дата звернення: 05.03.2024)

15. PHP Array Functions W3Schools Online Web Tutorials. URL: https://www.w3schools.com/php/php_ref_array.asp (дата звернення: 16.03.2024)

16. Twig for Drupal 8 Development: Twig Templating. Electric Citizen. URL: https://www.electriccitizen.com/citizen-blog/twig-drupal-8- development-twig-templating-part-1-2 (дата звернення: 22.04.2024)

17. What Is Data Monitoring and Why Does It Matter. Secoda. The Modern Data Management Platform. URL: https://www.secoda.co/blog/what-is-data-monitoring-and-why-is-it-important (дата звернення: 25.04.2024)

ДОДАТКИ

Додаток А

Функція для циклічного отримання авіаційних даних

```
function diplom_airline_cron() {
 $airline_json_data = file_get_contents('https://aviation-
edge.com/v2/public/airlineDatabase?key=b32028-a4ac39&codeIso2Country=NL');
 $airline_data = json_decode($airline_json_data, TRUE);
 if (is_array($airline_data)) {
   $active_airlines = array_filter($airline_data, function ($item) {
     return $item['statusAirline'] === 'active' && $item['sizeAirline'] > 0;
   });
 }
 if (!empty($active_airlines)) {
   $queue = \Drupal::queue('diplom_airline_queue_worker');
   foreach ($active_airlines as $airline) {
     $queue->createItem($airline);
   }
 }
}
```

Додаток Б

Код сутностей польоту та авіалінії

```
namespace Drupal\diplom_airline\Entity;

use Drupal\Core\Entity\ContentEntityBase;
use Drupal\Core\Entity\EntityTypeInterface;
use Drupal\Core\Field\BaseFieldDefinition;
use Drupal\diplom_airline\AirlineInterface;

/**
* Defines the airline entity class.
*
* @ContentEntityType(
*   id = "airline",
*   label = @Translation("Airline"),
*   label_collection = @Translation("Airlines"),
*   handlers = {
*     "view_builder" = "Drupal\Core\Entity\EntityViewBuilder",
*     "list_builder" = "Drupal\diplom_airline\AirlineListBuilder",
*     "views_data" = "Drupal\views\EntityViewsData",
*     "form" = {
*       "add" = "Drupal\diplom_airline\Form\AirlineForm",
*       "edit" = "Drupal\diplom_airline\Form\AirlineForm",
*       "delete" = "Drupal\Core\Entity\ContentEntityDeleteForm"
*     },
*     "route_provider" = {
*       "html" = "Drupal\Core\Entity\Routing\AdminHtmlRouteProvider",
*     }
*   },
*   base_table = "airline",
*   admin_permission = "administer airline",
*   entity_keys = {
*     "id" = "id",
*     "label" = "title",
*     "uuid" = "uuid"
*   },
*   links = {
*     "add-form" = "/admin/content/airline/add",
*     "canonical" = "/airline/{airline}",
*     "edit-form" = "/admin/content/airline/{airline}/edit",
*     "delete-form" = "/admin/content/airline/{airline}/delete",
*     "collection" = "/admin/content/airline"
*   },
*   field_ui_base_route = "entity.airline.settings"
* )
*/
class Airline extends ContentEntityBase implements AirlineInterface {

 /**
  * {@inheritdoc}
```

```
 */
public function getTitle() {
  return $this->get('title')->value;
}

/**
 * {@inheritdoc}
 */
public function setTitle($title) {
  $this->set('title', $title);
  return $this;
}

/**
 * {@inheritdoc}
 */
public function isEnabled() {
  return (bool) $this->get('status')->value;
}

/**
 * {@inheritdoc}
 */
public function setStatus($status) {
  $this->set('status', $status);
  return $this;
}

/**
 * {@inheritdoc}
 */
public static function baseFieldDefinitions(EntityTypeInterface $entity_type) {

  $fields = parent::baseFieldDefinitions($entity_type);

  $fields['title'] = BaseFieldDefinition::create('string')
    ->setLabel(t('Title'))
    ->setDescription(t('The title of the airline entity.'))
    ->setRequired(TRUE)
    ->setSetting('max_length', 255)
    ->setDisplayOptions('form', [
      'type' => 'string_textfield',
      'weight' => -5,
    ])
    ->setDisplayConfigurable('form', TRUE)
    ->setDisplayOptions('view', [
      'label' => 'hidden',
      'type' => 'string',
      'weight' => -5,
    ])
    ->setDisplayConfigurable('view', TRUE);

  $fields['status'] = BaseFieldDefinition::create('boolean')
```

```
  ->setLabel(t('Status'))
  ->setDescription(t('A boolean indicating whether the airline is enabled.'))
  ->setDefaultValue(TRUE)
  ->setSetting('on_label', 'Enabled')
  ->setDisplayOptions('form', [
    'type' => 'boolean_checkbox',
    'settings' => [
      'display_label' => FALSE,
    ],
    'weight' => 0,
  ])
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayOptions('view', [
    'type' => 'boolean',
    'label' => 'above',
    'weight' => 0,
    'settings' => [
      'format' => 'enabled-disabled',
    ],
  ])
  ->setDisplayConfigurable('view', TRUE);

$fields['description'] = BaseFieldDefinition::create('text_long')
  ->setLabel(t('Description'))
  ->setDescription(t('A description of the airline.'))
  ->setDisplayOptions('form', [
    'type' => 'text_textarea',
    'weight' => 10,
  ])
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayOptions('view', [
    'type' => 'text_default',
    'label' => 'above',
    'weight' => 10,
  ])
  ->setDisplayConfigurable('view', TRUE);

$fields['age_fleet'] = BaseFieldDefinition::create('float')
  ->setLabel(t('Fleet age'))
  ->setRequired(TRUE)
  ->setRevisionable(TRUE)
  ->setDisplayOptions('view', static::getDisplayViewOptionsString(10))
  ->setSetting('min', 0)
  ->setDisplayConfigurable('view', TRUE)
  ->setDisplayConfigurable('form', FALSE);

$fields['airline_id'] = BaseFieldDefinition::create('integer')
  ->setLabel(t('airlineID'))
  ->setDisplayOptions('view', static::getDisplayViewOptionsInteger(20))
  ->setDisplayConfigurable('view', TRUE)
  ->setDisplayConfigurable('form', FALSE)
  ->setRequired(TRUE);
```

```
$fields['callsign'] = BaseFieldDefinition::create('string')
  ->setLabel(t('callsign'))
  ->setSetting('max_length', 255)
  ->setDisplayOptions('view', static::getDisplayViewOptionsString(30))
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['code_hub'] = BaseFieldDefinition::create('string')
  ->setLabel(t('codeHub'))
  ->setSetting('max_length', 50)
  ->setDisplayOptions('view', static::getDisplayViewOptionsString(40))
  ->setDisplayConfigurable('form', FALSE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['code_iata_airline'] = BaseFieldDefinition::create('string')
  ->setLabel(t('codeIataAirline'))
  ->setSetting('max_length', 50)
  ->setDisplayOptions('view', static::getDisplayViewOptionsString(50))
  ->setDisplayConfigurable('form', FALSE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['code_icao_airline'] = BaseFieldDefinition::create('string')
  ->setLabel(t('codeIcaoAirline'))
  ->setSetting('max_length', 50)
  ->setDisplayOptions('view', static::getDisplayViewOptionsString(60))
  ->setDisplayConfigurable('form', FALSE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['code_iso_country'] = BaseFieldDefinition::create('string')
  ->setLabel(t('codeIso2Country'))
  ->setSetting('max_length', 50)
  ->setDisplayOptions('view', static::getDisplayViewOptionsString(70))
  ->setDisplayConfigurable('form', FALSE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['founding'] = BaseFieldDefinition::create('integer')
  ->setLabel(t('Founding'))
  ->setSetting('min', 0)
  ->setDisplayOptions('view', static::getDisplayViewOptionsInteger(80))
  ->setDisplayConfigurable('view', TRUE)
  ->setDisplayConfigurable('form', FALSE)
  ->setRequired(TRUE);

$fields['iata_prefix_accounting'] = BaseFieldDefinition::create('string')
  ->setLabel(t('iataPrefixAccounting'))
  ->setSetting('max_length', 50)
  ->setDisplayOptions('view', static::getDisplayViewOptionsString(90))
  ->setDisplayConfigurable('form', FALSE)
```

```
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['name_airline'] = BaseFieldDefinition::create('string')
  ->setLabel(t('nameAirline'))
  ->setSetting('max_length', 255)
  ->setDisplayOptions('view', static::getDisplayViewOptionsString(100))
  ->setDisplayConfigurable('form', FALSE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['name_country'] = BaseFieldDefinition::create('string')
  ->setLabel(t('nameCountry'))
  ->setSetting('max_length', 100)
  ->setDisplayOptions('view', static::getDisplayViewOptionsString(120))
  ->setDisplayConfigurable('form', FALSE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['size_airline'] = BaseFieldDefinition::create('integer')
  ->setLabel(t('sizeAirline'))
  ->setSetting('min', 0)
  ->setDisplayOptions('view', static::getDisplayViewOptionsInteger(130))
  ->setDisplayConfigurable('view', TRUE)
  ->setDisplayConfigurable('form', FALSE)
  ->setRequired(TRUE);

$fields['status_airline'] = BaseFieldDefinition::create('string')
  ->setLabel(t('statusAirline'))
  ->setSetting('max_length', 50)
  ->setDisplayOptions('view', static::getDisplayViewOptionsString(140))
  ->setDisplayConfigurable('form', FALSE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['type'] = BaseFieldDefinition::create('string')
  ->setLabel('type')
  ->setSetting('max_length', 255)
  ->setDisplayOptions('view', static::getDisplayViewOptionsString(150))
  ->setDisplayOptions('form', static::getDisplayViewOptionsString(150))
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['airline_logo'] = BaseFieldDefinition::create('image')
  ->setLabel(t('Airline logo'))
  ->setDisplayOptions('form', [
    'type' => 'image_image',
    'weight' => 1,
    'settings' => [
      'progress_indicator' => 'throbber',
      'preview_image_style' => 'thumbnail',
    ],
```

```
    ])
    ->setDisplayOptions('view', [
      'label' => 'hidden',
      'type' => 'image',
      'weight' => 0,
      'settings' => [
        'image_style' => 'default',
        'image_link' => '',
      ],
    ])
    ->setDisplayConfigurable('form', TRUE)
    ->setDisplayConfigurable('view', TRUE);

  return $fields;
}

/**
 * Get display view string options array.
 */
public static function getDisplayViewOptionsString(int $weight): array {
  return [
    'label' => 'above',
    'type' => 'string',
    'weight' => $weight,
  ];
}

/**
 * Get display view integer options array.
 */
public static function getDisplayViewOptionsInteger(int $weight): array {
  return [
    'label' => 'above',
    'type' => 'number_integer',
    'weight' => $weight,
  ];
}

}


namespace Drupal\diplom_new_flight\Entity;

use Drupal\Core\Field\BaseFieldDefinition;
use Drupal\Core\Entity\ContentEntityBase;
use Drupal\Core\Entity\EntityTypeInterface;

/**
* Defines the Flight entity.
*
* @ingroup flights
*
* @ContentEntityType(
```

```
*   id = "new_flight",
*   label = @Translation("New Flight"),
*   handlers = {
*    "views_data" = "Drupal\views\EntityViewsData",
*   },
*   base_table = "new_flight",
*   admin_permission = "administer flight entities",
*   entity_keys = {
*     "id" = "id",
*     "uuid" = "uuid",
*   },
* )
*/
class NewFlight extends ContentEntityBase {

 /**
  * {@inheritdoc}
  */
 public static function baseFieldDefinitions(EntityTypeInterface $entity_type) {
   $fields = parent::baseFieldDefinitions($entity_type);

   $fields['number'] = BaseFieldDefinition::create('string')
     ->setLabel(t('Flight number'))
     ->setDescription(t('The flight number.'))
     ->setSettings([
       'max_length' => 8,
     ])
     ->setDefaultValue('')
     ->setDisplayOptions('view', [
       'label' => 'hidden',
       'type' => 'string',
       'weight' => -4,
     ])
     ->setDisplayOptions('form', [
       'type' => 'string_textfield',
       'weight' => -4,
     ])
     ->setDisplayConfigurable('form', TRUE)
     ->setDisplayConfigurable('view', TRUE)
     ->setRequired(TRUE);

   $fields['origin'] = BaseFieldDefinition::create('string')
     ->setLabel(t('Origin'))
     ->setDescription(t('Origin point.'))
     ->setSettings([
       'max_length' => 50,
     ])
     ->setDefaultValue('')
     ->setDisplayOptions('view', [
       'label' => 'hidden',
       'type' => 'string',
       'weight' => -3,
     ])
```

```
  ->setDisplayOptions('form', [
    'type' => 'string_textfield',
    'weight' => -3,
  ])
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(FALSE);

$fields['time'] = BaseFieldDefinition::create('timestamp')
  ->setLabel(t('Time'))
  ->setDescription(t('Departure and Arrivals time.'))
  ->setSettings([
    'default_value' => '',
  ])
  ->setDisplayOptions('view', [
    'label' => 'hidden',
    'type' => 'timestamp',
    'weight' => 0,
  ])
  ->setDisplayOptions('form', [
    'type' => 'timestamp',
    'weight' => 0,
  ])
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(FALSE);

$fields['uri'] = BaseFieldDefinition::create('string')
  ->setLabel(t('URI'))
  ->setDescription(t('The URI to the logo'))
  ->setSetting('max_length', 255)
  ->setSetting('case_sensitive', TRUE)
  ->addConstraint('FileUriUnique')
  ->setRequired(FALSE);

$fields['airline'] = BaseFieldDefinition::create('string')
  ->setLabel(t('Airline'))
  ->setDescription(t('Name of the airline.'))
  ->setSettings([
    'max_length' => 24,
  ])
  ->setDisplayOptions('view', [
    'label' => 'hidden',
    'type' => 'string',
    'weight' => 2,
  ])
  ->setDisplayOptions('form', [
    'type' => 'string_textfield',
    'weight' => 2,
  ])
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(FALSE);
```

```
$fields['status_plane'] = BaseFieldDefinition::create('string')
  ->setLabel(t('Status plane'))
  ->setDescription(t('Is the plane flying or has it already landed.'))
  ->setSettings([
    'max_length' => 16,
  ])
  ->setDisplayOptions('view', [
    'label' => 'hidden',
    'type' => 'string',
    'weight' => 2,
  ])
  ->setDisplayOptions('form', [
    'type' => 'string_textfield',
    'weight' => 2,
  ])
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(FALSE);

$fields['flight_status'] = BaseFieldDefinition::create('string')
  ->setLabel(t('Status'))
  ->setDescription(t('Status of the flight.'))
  ->setSettings([
    'max_length' => 16,
  ])
  ->setDisplayOptions('view', [
    'label' => 'hidden',
    'type' => 'string',
    'weight' => 2,
  ])
  ->setDisplayOptions('form', [
    'type' => 'string_textfield',
    'weight' => 2,
  ])
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['aircraft_iata_code'] = BaseFieldDefinition::create('string')
  ->setLabel(t('Aircraft IATA code'))
  ->setDescription(t('Aircraft IATA code.'))
  ->setSettings([
    'max_length' => 16,
  ])
  ->setDisplayOptions('view', [
    'label' => 'hidden',
    'type' => 'string',
    'weight' => 2,
  ])
  ->setDisplayOptions('form', [
    'type' => 'string_textfield',
    'weight' => 2,
```

```
  ])
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

$fields['airline_iata_code'] = BaseFieldDefinition::create('string')
  ->setLabel(t('Airline IATA code'))
  ->setDescription(t('Airline IATA code.'))
  ->setSettings([
    'max_length' => 16,
  ])
  ->setDisplayOptions('view', [
    'label' => 'hidden',
    'type' => 'string',
    'weight' => 2,
  ])
  ->setDisplayOptions('form', [
    'type' => 'string_textfield',
    'weight' => 2,
  ])
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

// Arrival (iata code of arrival airport).
$fields['arrival_iata_code'] = BaseFieldDefinition::create('string')
  ->setLabel(t('Arrival IATA code'))
  ->setDescription(t('Arrival IATA code.'))
  ->setSettings([
    'max_length' => 16,
  ])
  ->setDisplayOptions('view', [
    'label' => 'hidden',
    'type' => 'string',
    'weight' => 2,
  ])
  ->setDisplayOptions('form', [
    'type' => 'string_textfield',
    'weight' => 2,
  ])
  ->setDisplayConfigurable('form', TRUE)
  ->setDisplayConfigurable('view', TRUE)
  ->setRequired(TRUE);

// Arrival (iata code of arrival airport).
$fields['departure_iata_code'] = BaseFieldDefinition::create('string')
  ->setLabel(t('Departure IATA code'))
  ->setDescription(t('Departure IATA code.'))
  ->setSettings([
    'max_length' => 16,
  ])
  ->setDisplayOptions('view', [
    'label' => 'hidden',
```

```
      'type' => 'string',
      'weight' => 2,
    ])
    ->setDisplayOptions('form', [
      'type' => 'string_textfield',
      'weight' => 2,
    ])
    ->setDisplayConfigurable('form', TRUE)
    ->setDisplayConfigurable('view', TRUE)
    ->setRequired(TRUE);

  // Arrival (iata code of arrival airport).
  $fields['flight_iata_number'] = BaseFieldDefinition::create('string')
    ->setLabel(t('Flight IATA number'))
    ->setDescription(t('Flight IATA number.'))
    ->setSettings([
      'max_length' => 16,
    ])
    ->setDisplayOptions('view', [
      'label' => 'hidden',
      'type' => 'string',
      'weight' => 2,
    ])
    ->setDisplayOptions('form', [
      'type' => 'string_textfield',
      'weight' => 2,
    ])
    ->setDisplayConfigurable('form', TRUE)
    ->setDisplayConfigurable('view', TRUE)
    ->setRequired(TRUE);

  return $fields;
 }

}
```

Додаток В

Основні функції авіаліній

```
/**
* @file
* Provides an airline entity type.
*/

use Drupal\Core\Entity\Display\EntityViewDisplayInterface;
use Drupal\Core\Entity\EntityInterface;
use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Queue\QueueFactory;
use Drupal\Core\Queue\QueueInterface;
use Drupal\Core\Render\Element;
use Drupal\Core\Url;
use Drupal\diplom_airline\AirlineInterface;

/**
* Implements hook_theme().
*/
function diplom_airline_theme() {
 return [
   'airline' => [
     'render element' => 'elements',
   ],
   'airline__teaser' => [
     'base hook' => 'airline',
   ],
 ];
}

/**
* Prepares variables for airline templates.
*
* Default template: airline.html.twig.
*
* @param array $variables
*   An associative array containing:
*   – elements: An associative array containing the airline information and any
*     fields attached to the entity.
*   – attributes: HTML attributes for the containing element.
*/
function template_preprocess_airline(array &$variables) {
 foreach (Element::children($variables['elements']) as $key) {
   $variables['content'][$key] = $variables['elements'][$key];
 }
}

/**
```

```
* Implements hook_entity_view().
*/
function diplom_airline_entity_view(
 array &$build,
 EntityInterface $entity,
 EntityViewDisplayInterface $display,
 string $view_mode
): void {
 if (!$entity instanceof AirlineInterface || !in_array($view_mode, ['teaser'])) {
  return;
 }

 $links = [];
 if ($entity->hasField('code_iata_airline') &&
  ($airline_id = $entity->get('code_iata_airline')->getString()) !== '') {
  $url = Url::fromRoute("flights.tracker_by_airline", ['airline_id' => $airline_id])->toString();
 }


 if ($display->getComponent('link_to_flights_tracker')) {
  $build['link_to_flights_tracker'] = [
   '#markup' => '<div class = "track-flights"><a href=' . $url . '>Track flights</a></div>',
  ];
 }

}

/**
* Implements hook_entity_extra_field_info().
*/
function diplom_airline_entity_extra_field_info(): array {
 $items = [];

 $items['airline']['airline']['display']['link_to_flights_tracker'] = [
  'label' => t('Track flights'),
  'visible' => TRUE,
  'weight' => 100,
 ];

 return $items;
}

/**
* Implements hook_theme_suggestions_HOOK().
*/
function diplom_airline_theme_suggestions_airline(array $variables) {
 $suggestions = [];
 $suggestions = $variables['theme_hook_original'] . '__' . $variables['elements']['#view_mode'];
 return $suggestions;
}
```

Додаток Г

Функціонал для аналізу авіаційних даних

```
namespace Drupal\flights\Plugin\QueueWorker;
use Drupal\Core\Entity\EntityTypeManagerInterface;
use Drupal\Core\Plugin\ContainerFactoryPluginInterface;
use Drupal\Core\Queue\QueueWorkerBase;
use Drupal\flights\Entity\Flight;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
* Sample Queue Worker.
*
* @QueueWorker(
*   id = "diplom_flights_queue_worker",
*   title = @Translation("Sample Queue Worker: Flights"),
*   cron = {"time" = 60}
* )
*/
class DiplomFlighsQueueWorker extends QueueWorkerBase implements
ContainerFactoryPluginInterface
{

 /**
  * The entity type manager.
  */
 protected EntityTypeManagerInterface $entityTypeManager;

 /**
  * SendPhaseStatisticsByEmailWorker constructor.
  *
  * @param array $configuration
  *   The configuration.
  * @param string $plugin_id
  *   The plugin id.
  * @param array $plugin_definition
  *   The plugin definition.
  * @param \Drupal\Core\Entity\EntityTypeManagerInterface $entity_type_manager
  *   The entity type manager.
  */
 public function __construct(array $configuration, $plugin_id, array $plugin_definition,
EntityTypeManagerInterface $entity_type_manager)
 {
   parent::__construct($configuration, $plugin_id, $plugin_definition);

   $this->entityTypeManager = $entity_type_manager;
 }

 /**
  * {@inheritdoc}
  */
```

```
 public static function create(ContainerInterface $container, array $configuration, $plugin_id,
$plugin_definition): self
 {
  return new static(
   $configuration,
   $plugin_id,
   $plugin_definition,
   $container->get('entity_type.manager'),
  );
 }

 public function processItem($item)
 {
  $airline_storage = $this->entityTypeManager->getStorage('flight');

  if ($item === 'No Record Found' || empty($item['status'])) {
   return;
  }

  if (!empty($item['flight']['iataNumber'])) {
   $flight = $airline_storage->getQuery()
    ->accessCheck(FALSE)
    ->condition('number', $item['flight']['iataNumber'])
    ->execute();
  }
  if (empty($flight)) {
   $flight_date = Flight::create([
//       'number' => $item['flight']['number'] ?? '',
    'number' => $item['flight']['iataNumber'],
    'airline' => $item['airline']['iataCode'],
    'flight_status' => $item['status'],
    // Replace status plane with departure iata_code.
    'status_plane' => $item['arrival']['iataCode'],
    'origin' => $item['departure']['iataCode'],
   ]);
   $flight_date->save();
  } else {
   $current_flight = $airline_storage
    ->loadByProperties(['number', $item['flight']['iataNumber']]);
   $current_flight->set('flight_status', $item['status'])->save();
  }

 }
}
```

Додаток Д

Функціонал для моніторингу рейсів

```
const map = L.map('flights-map').setView([52.3676, 4.9041], 7);
const attributtion = 'Powered by Koval';
const tileUrl =
'https://tile.thunderforest.com/cycle/{z}/{x}/{y}.png?apikey=82a1d2065df544738e10bffa97b03d1b
';
const tiles = L.tileLayer(tileUrl, {attributtion, maxZoom: 24, minZoom: 4});
tiles.addTo(map);
const COUNTRY_CODE='PT';
const airportGetEndpoint = 'https://aviation-edge.com/v2/public/airportDatabase?key=' +
API_KEY;
const airportIataCodes = [
 "AMS", "DHR", "EIN", "ENS", "GLZ", "GRQ", "LEY",
 "LWR", "MST", "QRH", "QWZ", "QYC", "QYE", "QYH",
 "QYI", "QYL", "QYM", "QYP", "QYT", "QYV", "QYZ",
 "RTM", "UDE", "WOE", "ZXY", "ZYA", "ZYE", "ZYH",
 "ZYM", "ZYO", "ZYT", "ZYU", "QEK", "UTC",
];
const planeIcon = L.icon({
 iconUrl: 'https://kovalskiy266.github.io/public_flights_data/airplane-flight-svgrepo-com.svg',
 iconSize: [24, 50],
 iconAnchor: [25, 16],
});

(function ($, Drupal, drupalSettings) {

 function fetchData(url, options) {
  return fetch(url, options)
    .then((res) => {
      return res.json();
    })
    .catch((err) => {
      if (err) throw err;
```

```
    });
}


function putStatesFlighs(map, markers) {
  const airportMarkers = {};

  const airportIcon = L.icon({
    iconUrl: 'https://kovalskiy266.github.io/public_flights_data/icons8-airplane-100.png',
    iconSize: [24, 40],
    iconAnchor: [25, 16]
  });

  const specificIcon = L.icon({
    iconUrl: 'https://kovalskiy266.github.io/public_flights_data/icons8-airplane-48.png',
    iconSize: [24, 40],
    iconAnchor: [25, 16]
  });

  const flameIcon = L.icon({
    iconUrl: 'https://kovalskiy266.github.io/public_flights_data/flames.png',
    iconSize: [24, 40],
    iconAnchor: [25, 16]
  });

  L.marker([55.751999, 37.617734], { icon: flameIcon, draggable: false }).bindPopup('<h3>На болотах все стабільно</h3>').addTo(map);



  const urlParams = new URLSearchParams(window.location.search);
  const myParam = urlParams.get('flight');

  const url = 'https://flight-radar1.p.rapidapi.com/flights/list-by-airline?airline=' + drupalSettings.airline_id;
```

```
  const options = {
   method: 'GET',
   headers: {
    'X-RapidAPI-Key': '25484c8981msh635f0fb93513b1ap165027jsn2ab2d45f9e29',
    'X-RapidAPI-Host': 'flight-radar1.p.rapidapi.com'
   }
  };

  fetchData(url, options).then(function (states) {
   if (states.aircraft) {
    states.aircraft.forEach((state) => {
     if (state[0]) {
      let lat= state[2],
       lng = state[3],
       angle = Math.ceil(state[4]);
      if (state[12]) {
       putStatesAirport(map, airportMarkers, state[12]);
      }
      if (state[13]) {
       putStatesAirport(map, airportMarkers, state[13]);
      }
      if (markers[state[0]]) {
       markers[state[0]]
        .setLatLng([lat, lng], { icon: airportIcon, rotationAngle: angle,  draggable: true
}).bindPopup('<h3>' + state[12] + ' ----> ' + state[13] + '</h3>');
      } else {
       markers[state[0]] = L.marker([lat, lng], { icon: airportIcon, rotationAngle: angle, draggable:
false }).bindPopup('<h3>' + state[12] + ' ----> ' + state[13] +  '</h3>');
       markers[state[0]].addTo(map);
      }
     };
    });
   }
   setTimeout(() => putStatesFlighs(map, markers), 7000);
  });
```

```
};

function putStatesAirport(map, markers, airport_iata_code) {
  const airportIcon = L.icon({
    iconUrl: 'https://kovalskiy266.github.io/public_flights_data/airport-svgrepo-com.svg',
    iconSize: [48, 105],
    iconAnchor: [25, 16]
  });
  fetchData('https://aviation-edge.com/v2/public/airportDatabase?key=8745ef-
f18ea1&codeIataAirport=' + airport_iata_code).then(function (states) {
    states.forEach((state) => {
      let lat= state.latitudeAirport,
        lng = state.longitudeAirport;

      if (markers[state.airportId]) {
        markers[state.airportId]
          .setLatLng([lat, lng]);
      } else {
        markers[state.airportId] = L.marker([lat, lng], { icon: airportIcon, draggable: false
}).bindPopup('<h3>' + state.nameAirport + '</h3>');
        markers[state.airportId].addTo(map);
      }
    });
  });
}

const flightsMarkers = {};
putStatesFlighs(map, flightsMarkers);

})(jQuery, Drupal, drupalSettings);
```