МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Кафедра комп'ютерної інженерії

ІВАНІЦКИЙ Назар Юрійович

«Алгоритми ретрансляції звукових повідомлень в реальному часі для системи вуличного радіо "розумного" міста / Real-time audio message retransmission algorithms for a smart city street radio system»

Спеціальність: 123 «Комп’ютерна інженерія»
Освітньо-професійна програма: Комп’ютерна інженерія
Кваліфікаційна робота

Виконав студент групи КІм-21
Назар ІВАНІЦКИЙ

_____

Науковий керівник:
к.т.н., доц. Юрій БАТЬКО

_____

Кваліфікаційну роботу допущено
до захисту:
"___" _____ 20___ р.

Завідувач кафедри
_____ Олег БЕРЕЗЬКИЙ

Тернопіль — 2021

ЗМІСТ

Студент групи КІм-21 _____ Назар ІВАНІЦКИЙ
(підпис)

Керівник магістерської роботи _____ Юрій БАТЬКО
(підпис)

ВСТУП

Актуальність теми. “Розумні” міста функціонують на основі ефективності та прозорості та демонструють широкий спектр нових технічних засобів і послуг. Одним із ключових запитів громадян є інформування за допомогою сучасних, цифрових та персоналізованих засобів про проблеми, які можуть вплинути на: відключення комунальних послуг, зміни в громадському транспорті, перевищення рівня забруднювальних речовин та інші надзвичайні події. Задача реалізації алгоритмів ретрансляції звукових повідомлень в реальному часі для системи вуличного радіо, що забезпечить потреби громадян “розумного” міста є актуальною.

Швидка урбанізація є однією з найважливіших сил сучасного суспільства. Щотижня в міста переїжджає 1,3 мільйона людей, і до 2040 року очікується, що 65% населення світу буде жити в містах, причому 90% цього зростання міського населення буде відбуватися в Африці та Азії. У світі вже є 21 мегаполіс, у кожному з яких проживає понад 10 мільйонів людей. Необхідно, щоб ми почали працювати над нашими містами вже сьогодні, щоб створити екологічне середовище, яке буде придатним для життя в майбутньому.

Цифровізація дозволила інфраструктурі стати розумнішою. Оптимальне використання фізичного простору та енергії, активне управління користувачами, активами та процесами, ефективне функціонування бізнесу та компаній поступово виграли від процесу цифровізації. Країни, які запровадили цифровізацію не лише у виробничому секторі, як-от використання робототехніки чи Інтернету речей (IoT), але й у всебічних рішеннях розумних міст, включаючи електронну охорону здоров’я чи розумні мережі, отримали вигоду від збільшення валового внутрішнього продукту (ВВП) внаслідок зниження кінцевих операційних витрат (OPEX).

Основною проблемою впровадження «розумного» міста є його визначення та обсяг; Концепція «розумного» міста варіюється від чистих електронних рішень,

інтегрованих в інфраструктуру інформаційно-комунікаційних технологій (ІКТ) «розумного» міста, таку як Інтернет речей, хмара або мобільні телефони та великі дані з аналітикою, яка в реальному часі вимірює використання його статусу, що дозволяє приймати рішення вищого рівня. Крім того, “розумне” місто може бути реалізоване як багатопрофільне середовище співпраці уряду, що надає своїм громадянам можливості.

Мета і завдання дослідження. Метою кваліфікаційної роботи є розроблення алгоритмів ретрансляції звукових повідомлень в реальному часі для системи вуличного радіо "розумного" міста.

Об’єктом дослідження є процес ретрансляції звукових повідомлень в реальному часі.

Предметом дослідження є методи та алгоритми ретрансляції звукових повідомлень в реальному часі для системи вуличного радіо.

Для досягнення поставленої мети необхідно розв'язати наступні задачі:

1. Проаналізувати технології Інтернету речей та "розумного міста".
2. Проаналізувати алгоритми ретрансляції звукових повідомлень в реальному часі в системі вуличного радіо.
3. Розробити алгоритми ретрансляції звукових повідомлень в реальному часі.
4. Програмно реалізувати менеджмент-сервер для системи вуличного радіо.
5. Реалізувати станцію системи вуличного радіо.
6. Розробити інтерфейс адміністратора системи вуличного радіо.
7. Провести дослідження розроблених алгоритмів.

Методи дослідження базуються на використанні побудови інфраструктури розумного міста (для аналізу системи розумного міста); стрімінгу звукових повідомлень (для передачі звукових повідомлень та їх відтворення на клієнті вуличного радіо); об'єктно-орієнтованого та функціонального програмування (для проєктування та реалізації програмних засобів системи вуличного радіо).

Наукова новизна одержаних результатів. Розроблено алгоритми ретрансляції звукових повідомлень в реальному часі для системи вуличного радіо, що дозволило інтегрувати надалі цю систему зі системою "розумного" міста.

Практичне значення отриманих результатів. Реалізовано алгоритми відтворення звукових повідомлень для системи вуличного радіо "розумного" міста. Розроблено систему вуличного радіо, яка складається з: станції вуличного радіо, сервера для менеджменту станціями вуличного радіо, інтерфейсу адміністратора системи, системи відтворення звукових повідомлень згідно з чергою, системи відтворення звукових повідомлень в режимі реального часу, зберігання черги відтворення в БД. Розроблювані алгоритми реалізовано у вигляді менеджмент сервера станцій вуличного радіо та клієнтів (станцій) на мові Python, та бази даних PostgreSQL.

Публікації та апробація випускної кваліфікаційної роботи. Отримані результати апробовані в межах V науково-практичної конференції молодих вчених і студентів «Інтелектуальні комп'ютерні системи та мережі» Західноукраїнського національного університету та опубліковано три тези доповідей за темою роботи [3,4,5].

Кваліфікаційна робота складається із трьох розділів, висновків, списку використаної літератури та додатків. У першому розділі систематизовано алгоритми системи вуличного радіо та "розумного" міста.

В другому розділі розроблено алгоритми ретрансляції звукових повідомлень в реальному часі для системи вуличного радіо.

У третьому розділі розроблено та реалізовано алгоритми ретрансляції звукових повідомлень в реальному часі для системи вуличного радіо, інтерфейс адміністратора системи. Описана структура програмного забезпечення. Проведено дослідження по захисту розроблених алгоритмів.

1. АНАЛІЗ АЛГОРИТМІВ СИСТЕМИ ВУЛИЧНОГО РАДІО В СИСТЕМІ "РОЗУМНОГО" МІСТА

1.1. Технології, концепції та стандарти системи "розумного" міста

Попри різницю у визначеннях, загальне бачення концепції у світі таки склалося. Серед базових характеристик «розумних міст» — стійкість та екологічність, участь суспільства в управлінні, ефективне використання даних, прагнення підвищити якість сервісів та рівень життя.

Ключові характеристики концепції «розумне» місто:

- Людиноцентричне — місто орієнтоване на мешканців, бізнес, працівників, туристів.
- Добре кероване.
- Доступне та відкрите для людей та нових ідей.
- Розкриває дані про свою діяльність.
- Захищає персональні дані.
- Засноване на інтегрованих сервісах та інфраструктурі.
- Проактивне у навчанні та розвитку громадян.

На основі визначень Міжнародної організації зі стандартизації, Міжнародної електротехнічної комісії, Міжнародної спілки електрозв'язку, Європейського інституту стандартизації електрозв'язку та інших профільних організацій.

Розумні міста аналізуються як середовище відкритих і керованих користувачами інновацій для експериментування та перевірки майбутніх дослідницьких послуг Інтернету та міських пробних програм [36]; ефективне використання спільних ресурсів з метою створення інноваційних екосистем вимагає стійкого партнерства та стратегій співпраці між зацікавленими сторонами.

Створені професійні спільноти для сприяння взаємному навчанню між багатопрофільними представниками архітектури, планування, інженерії, транспорту, комунального господарства, інформаційних технологій, операційних

досліджень, соціальних наук, географії та екології, державних фінансів та політики, а також комунікацій [25]; результатом є нова теорія міст, заснована на інноваційних і широких джерелах інформації про те, що відбувається в місті майже в реальному часі. Набір загальних багатовимірних компонентів, що лежать в основі концепції «розумного міста», включаючи (технології, людей та інституції) [31], а також основні фактори успішної ініціативи «розумного міста» описується як інтеграція інфраструктури та послуг, пов'язаних із технологіями, соціальне навчання для зміцнення людського потенціалу. інфраструктура та управління для інституційного вдосконалення та залучення громадян.

Інформаційно-комунікаційні технології в системі "розумного міста".

Розумне місто визначається як місто, в якому ІКТ об'єднані з класичними інфраструктурами, скоординовані та інтегровані за допомогою нових цифрових технологій [37]. Розумне бачення складається шляхом визначення цілей, додавання завдань дослідження та розгляду сценаріїв у межах проєктних областей. Концепція розумних та підключених спільнот базується на IoT, розпізнаванні натовпу та кіберфізичних хмарних обчисленнях для забезпечення всеосяжної мережі підключених пристроїв [27]; крім того, розумні датчики та аналітика великих даних, щоб забезпечити перехід від IoT до контролю в реальному часі.

Міський Інтернет речей розроблений для підтримки бачення розумного міста, яке використовує найсучасніші комунікаційні технології для надання послуг з додатковою вартістю для адміністрації міста та для громадян [28]; міський Інтернет речей оптимізує технології, протоколи та архітектуру, використовуючи найкращі технічні рішення та практичні рекомендації. Мобільні телефони з сенсорною технологією, такою як глобальна система позиціювання (GPS), гіроскоп, мікрофон, камера, акселерометр [10], також надають нові інноваційні послуги в архітектурі та структурі Smart City.

Концепція Sensing as a Service (рисунок 1.1) досліджується з технологічних, економічних та соціальних аспектів [35]; дослідження також визначає основні відкриті виклики та проблеми, включаючи методи його взаємозв'язку в поточній

інфраструктурі Інтернету речей, платформах і програмних додатках, які пропонуються як послуги з використанням хмарних технологій. Нещодавнє бачення Інтернету майбутнього (FI) та його окремих компонентів, IoT та Internet of Services (IoS) можуть стати будівельними блоками для просування до єдиної платформи ІКТ у міському масштабі, яка перетворює розумне місто на відкриту інноваційну платформу [36]; Пропонована загальна реалізація заснована на моделі Ubiquitous Sensor Network (USN), яка відповідає вимогам відкритих, об’єднаних і надійних платформ. Основними проблемами, які можуть перешкодити IoT суттєво підтримувати сталий розвиток майбутніх розумних міст, є невідповідність між підключеними об’єктами та ненадійний характер супутніх послуг [16]; когнітивну структуру, яка динамічно змінюється об'єкти реального світу представлені у віртуалізованому середовищі, і де пізнання та близькість використовуються для вибору найбільш релевантних об'єктів для цілей програми інтелектуальним та автономним способом.

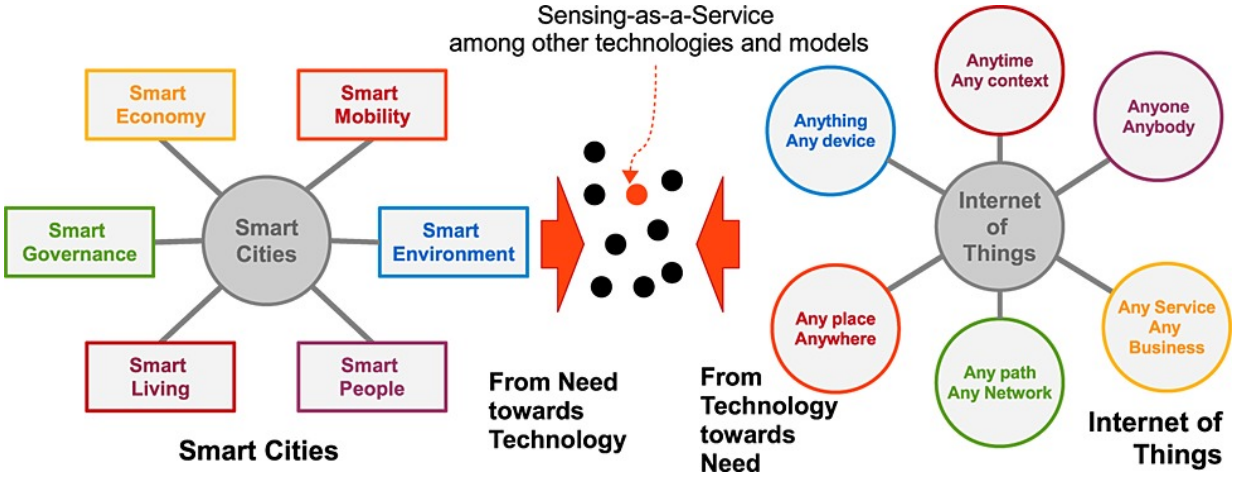

Рисунок 1.1 Концепція Sensing as a Service

Великі дані (Big Data).

Інфраструктура розумного міста базується на концепції цифрового міста, що об’єднує Інтернет речей та технології хмарних обчислень [29], щоб забезпечити автоматичний контроль та розвідувальні служби для людей та логістику у фізичних містах. Застосування великих даних у розумних містах включає

розгалужені сенсорні мережі з проблемами та викликами, які можна пом’якшити за допомогою розгортання хмарних обчислень і аналізу даних.

Існує прогалина в об’єднанні сучасного рівня техніки в інтегрованих аналітичних додатках Big Data і машинного навчання для IoT і додатків Smart City [46], що допомогло б зменшити витрати на розробку та надати нові аналітичні послуги для громадян і тих, хто приймає рішення в містах.

Аналітика великих даних має величезний потенціал для покращення послуг розумного міста [7]; однак розуміння вимог, які підтримують його впровадження та застосування, є ключем до ефективного аналізу та використання для досягнення успіху у сферах послуг розумного міста.

Поєднання Інтернету речей і великих даних є ще незрілим невивченим дослідженням з новими та цікавими викликами для досягнення мети майбутніх розумних міст [44]. Труднощі зосереджені насамперед на усвідомленні основних характеристик розумного середовища та співпраці між технологіями та бізнес-сервісами, які дозволяють розумним містам покращити своє бачення та принципи.

Додатки та потенційні можливості Big Data Smart City зможуть зрозуміти його середовище, аналізуючи його дані, щоб негайно вносити зміни для розв'язання проблем та покращення якості життя мешканців [7]; дані повинні бути зібрані з усіх мереж, пристроїв і датчиків, вбудованих в його інфраструктуру; він стає цінним, проходячи різні етапи обробки, застосовуючи передовий аналіз даних платформ Big Data і, нарешті, експортуючи дані в корисну платформу, щоб покращити програмне забезпечення будь-якої системи міста.

Великі міські дані, які надходять від міських датчиків, стануть джерелом інформації про розумне місто [11], що дозволить довгострокове стратегічне планування замість короткострокового мислення про те, як міста функціонують і як ними можна керувати. Дві тісно пов'язані нові технологічні рамки, Інтернет речей і великі дані можуть зробити розумні міста ефективними та чуйними [30]; однак, ці технологічні досягнення вимагають інтеграції з точки зору фізичної інфраструктури, де цифрові технології перетворюються на кращі суспільні

послуги для мешканців та краще використання ресурсів при зменшенні впливу на навколишнє середовище.

Енергія.

Ієрархічна, централізовано керована енергетична мережа не відповідає потребам розумних міст. Розумні мережі збільшують пропускну здатність та ефективність шляхом оптимізації попиту, енергії та доступності мережі. Щоб надавати надійну інформацію моніторингу в режимі реального часу, яка забезпечує гнучкість роботи, Smart Grids вимагають безпеки, якості обслуговування (QoS) в мережах передачі даних і технологічних стандартів для взаємодії [39], де ІКТ відіграють фундаментальний елемент у їхньому розвитку та продуктивності.

Надійна та швидка комунікаційна інфраструктура [47] необхідна для з'єднання між великою кількістю розподілених елементів, таких як генератори, підстанції, розподіл, передача, зберігання та користувачі [8]. Розумні мережі також можуть бути включені в рамки, які спрямовані на повернення інвестицій між виробниками, операторами та клієнтами; ці фреймворки поділені на три інтерактивні смарткомпоненти [41]: розумні центри керування, розумні мережі передачі та розумні підстанції.

Основні проблеми для Smart Grids включають взаємодію між Smart Grid і Smart Meter, а також надійність, доступність, затримку та якість обслуговування в бездротових середовищах, які піддаються атакам типу «Відмова в обслуговуванні» (DoS) [9]. Зусилля зі стандартизації для гармонізації комунікаційних стандартів і протоколів координуються через Європу, США та Китай [38].

Огляд Smart Grids [40] (рисунок 1.2) складається з трьох основних систем: розумної інфраструктури, управління та захисту. Дослідження охоплює розумні підсистеми енергії, інформації, безпеки та комунікацій на додаток до цілей управління, таких як підвищення енергоефективності, профілювання попиту, максимізація корисності, зниження вартості та контроль викидів.

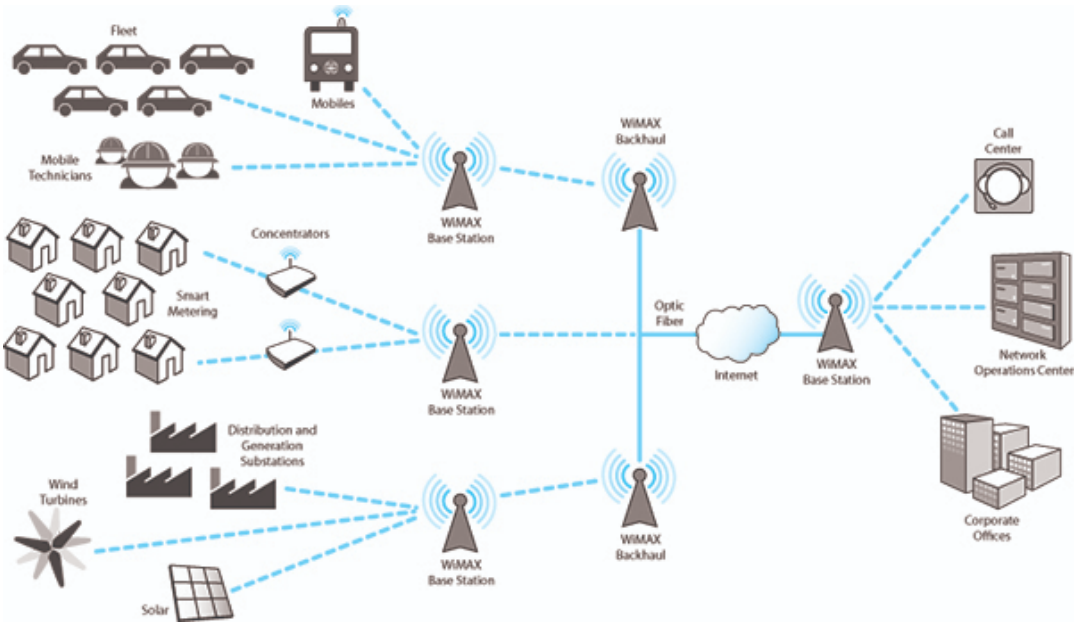

Рисунок 1.2 Підсистема Smart Grids

На додаток до розумних міст, ІКТ-активатор також потребує все більшої кількості розсіяної енергії для своїх центрів обробки даних і обладнання для передачі зв'язку. Energy Packet Networks (EPN) використовує накопичувачі енергії для адаптації до нерегулярного постачання відновлюваних джерел енергії та періодичного попиту серверів хмарних обчислень [20].

Модель EPN зберігає та розподіляє квантовані енергетичні одиниці, або енергетичні пакети, до та від великого діапазону пристроїв, заснованих на механізмах, аналогічних комп'ютерним мережам [21]. Енергетичні пакети керуються та оптимізуються центрами енерго-диспетчеризації, які отримують запити від центрів зберігання споживачів, які бажають поповнити [19].

Blockchain в системі "розумного міста".

Blockchain дозволяє оцифрувати контракти, оскільки забезпечує аутентифікацію між сторонами та шифрування інформації, яка поступово збільшується під час їх обробки в децентралізованій мережі. Технологія BlockChain може порушити світ банківської справи, забезпечуючи криптовалюти [48] глобальні грошові перекази, платіжні рішення [32] розумні контракти [23], автоматизовані банківські записи та цифрові активи на додаток до анонімності користувачів [24].

Децентралізовані системи управління персональними даними, засновані на блокчейні, забезпечують користувачам власний контроль над своїми даними [50] та розповсюдження цифрового контенту, що керується правами користувачів [43]. Децентралізація методу консенсусу, який використовує оцінку правдивості, застосовується до управління контрактами, наприклад управління цифровими правами [13].

Блокчейн вже застосовувався в розумних мережах, забезпечуючи безпеку енергетичних транзакцій у децентралізованій торгівлі [6], інтелектуальні транспортні системи на семи рівневий концептуальній моделі, яка імітує модель OSI [15], розумні пристрої забезпечують безпечну комунікаційну платформу в розумному місті [12]. Керувати та налаштовувати пристрої для Інтернету речей [26], розумних будинків [14] та цифрової документації.

1.2. Аналіз концептів та технологій інтернету речей

Якщо ми перейдемо до визначення Інтернету речей, ми з’ясуємо, що це концепція міжмережевого об’єднання фізичних пристроїв, під’єднаних пристроїв, розумних пристроїв та інших пристроїв, вбудованих з електронікою, програмним забезпеченням, датчиками, приводами та мережевим з’єднанням. які дозволяють цим об’єктам збирати та обмінюватися даними[26].

Це означає, що об’єкт можна відчути або керувати дистанційно через наявну мережеву інфраструктуру, що забезпечує пряму інтеграцію фізичного світу в комп’ютерні системи, що, покращує їх ефективність і точність і зменшує людське втручання.

Якщо доповнити його датчиками та виконавчими механізмами, ця технологія стане прикладом більш загального класу кібер-фізичних систем, який також охоплює такі технології, як розумні мережі, віртуальні електростанції, розумні будинки, інтелектуальний транспорт та розумні міста.

Ключові концепції Інтернету речей (IoT):

● Апаратне забезпечення — серцем Інтернету речей є мільярди взаємопов’язаних пристроїв із підключеними датчиками та виконавчими механізмами, які відчувають і керують фізичним світом. На додаток до мережевого підключення для передачі даних, які вони збирають, ці пристрої потребують деяких базових можливостей обробки та зберігання, які надаються мікроконтролером або інтегральною схемою, наприклад, системою на чіпі (SoC) або програмованою користувачем вентильною матрицею (FPGA)[11].

● Вбудоване програмування — пристрої IoT є вбудованими пристроями, і їх можна створити прототипом за допомогою платформ мікроконтролерів, таких як Arduino, зі спеціально розробленими друкованими платами (PCB), розробленими пізніше. Створення прототипів за допомогою цих платформ вимагає навичок проєктування схем, програмування мікроконтролерів і глибокого розуміння протоколів апаратного зв’язку, таких як послідовний, I2C або SPI, що використовуються для встановлення зв’язку між мікроконтролером і підключеними датчиками та виконавчими механізмами. Вбудовані програми часто розробляються на C++ або C, однак Python і JavaScript стають все більш популярними для створення прототипів пристроїв IoT[11].

● Безпека — безпека є однією з найважливіших проблем в Інтернеті речей, тісно пов’язаною з етикою даних, конфіденційністю та відповідальністю. Він повинен бути вбудований на кожному кроці проєктування системи. Оскільки щодня підключаються мільйони нових пристроїв, кількість потенційних точок атаки зростає щодня. Оскільки на кону так багато, навички інженерної безпеки, включаючи оцінку загроз, етичний злом, шифрування для забезпечення цілісності даних, захист мережевих архітектур і додатків, а також моніторинг подій, реєстрацію активності та аналіз загроз, виходять на передній план розвитку проєктів Інтернету речей.

● Мережева та хмарна інтеграція — проєктування та керування мережею є важливими для IoT через величезний обсяг підключених пристроїв і через вплив, який можуть мати рішення щодо проєктування мережі. Підключення дозволяє

пристроям спілкуватися з іншими пристроями, а також із програмами та службами, які працюють у хмарі. Хоча хмарні обчислення та Інтернет речей – це дві дуже різні технології, потокова передача даних у реальному часі та хмарна інтеграція мають вирішальне значення для належного функціонування Інтернету речей. Хмарна інфраструктура використовується для зберігання, обробки та аналізу даних, а також для реалізації бізнес-логіки IoT-додатків.

● Аналітика даних і прогнозування — кількість пристроїв IoT, які передають дані, щодня збільшується, що перетворює великі дані на величезні дані. Розробникам потрібно безпечно та надійно отримувати, зберігати та запитувати величезну кількість різнорідних даних, що надходять із цих пристроїв. Крім того, оскільки багато пристроїв IoT генерують дані із затримкою або чутливими до часу, також необхідно фільтрувати або відкидати нерелевантні дані.

● Машинне навчання та штучний інтелект — щоб доставити цінність і зрозуміти величезні обсяги даних, які генеруються пристроями Інтернету речей, машинне навчання та навички штучного інтелекту є найважливішими навичками для розробників Інтернету речей. Щоб бути по-справжньому розумним, аналітика великих даних має застосовувати когнітивні обчислювальні методи, отримані з аналізу даних, моделювання, статистики, машинного навчання та штучного інтелекту. Ці методи можна застосовувати в режимі реального часу як для потоків даних датчиків для прогнозного аналізу, так і для автономного прийняття рішень у відповідь на вхідні дані. Крім того, машинне навчання можна застосувати до історичних даних для виявлення закономірностей або аномалій у даних[35].

Інтернет речей та індустрія.

Вплив Інтернету речей на промислові екосистеми призвів до появи надійного кібер-фізичного з’єднання, яке називають четвертою промисловою революцією, також відомою як промисловий Інтернет речей (IIoT). Підключена екосистема охоплює фізичні підключені активи: у тому числі ті, що знаходяться у виробничому цеху, а також виготовлені підключені пристрої, такі як підключені автомобілі або прилади.

IIoT поєднує технологію великих даних і машинне навчання; підключення даних датчика, зв'язку «машина-машина» (M2M) та робототехніки; в промисловому виробництві.

Внаслідок IIoT клієнти більше зосереджуються на операційних витратах (OPEX)[18], ніж на капітальних витратах (CAPEX). Наприклад, на нафтогазових, хімічних і нафтохімічних підприємствах клієнти переходять на новітні технології, такі як техніка перебудовуваної діодної лазерної спектроскопії (TDLS), щоб зменшити загальні експлуатаційні витрати. Аналізатори TDLS замінюють традиційні рішення, такі як діоксид цирконію та парамагнітні, для вимірювання кисню та технологічні газові хроматографи для вимірювання легких вуглеводнів.

Thermo Fisher Scientific підвищила продуктивність продукту, системи ультра високоефективної рідинної хроматографії Vanquish (UHPLC). Корпорація Waters заявляє, що «Пропускна здатність, селективність і специфічність, досягнуті за допомогою системи ACQUITY UPLC®, забезпечують покращені результати хроматографії, споживаючи менше розчинника, ніж традиційна ВЕРХ – для більш ефективної лабораторної роботи». Продуктивність має вирішальне значення для кількісного та якісного аналізу в таких областях, як аналіз фармацевтичних препаратів, моніторинг навколишнього середовища, тестування харчових продуктів та моніторинг якості води. Виробники аналітичного обладнання повинні все більше працювати зі своїми клієнтами та впроваджувати інноваційні продукти, які підвищують продуктивність процесів клієнтів і знизять загальні експлуатаційні витрати, щоб клієнти могли збільшити прибуток.

Промислова трансформація змусила клієнтів вимагати спрощених процесів та аналітичного обладнання, які можуть зменшити ручне втручання та спростити весь процес. Як приклад інноваційного продукту, лазерна іонізаційна мас-спектрометрія з електророзпиленням (LAESI) є передовою технікою іонізації, яка дозволяє аналізувати біологічні зразки без необхідності підготовки зразків. Підготовка зразків є складною для клієнтів через тривалий процес і відсутність кваліфікованої людської праці. Техніка LAESI є вигідною для клієнтів, оскільки вона розв'язувати їхні проблеми. Як приклад технологічних інновацій у

переробних галузях, вимірювання кисню в процесі є ключем до забезпечення безпеки навколишнього середовища. Вимірювання видобутку газу може бути складним, оскільки воно спричиняє часті вимоги до технічного обслуговування, менший час відгуку та часте засмічення лінії. Таким чином, клієнти переходять на вбудовану обробку, оскільки це знижує вартість володіння, дає швидшу окупність, зменшує обслуговування та дозволяє безперервно використовувати (навіть у найсуворіших умовах).

У зв’язку з впливом інноваційних процесів і продуктів IIoT, або Industry 4.0 (рисунок 1.3)[22], на ринок аналітичних приладів клієнти впроваджують аналізатори процесів для поєднання виробництва та контролю якості в деяких програмах. Аналізатори процесів можуть допомогти користувачам негайно виправити будь-які помилки у виробництві та знизити витрати, пов’язані з проведенням окремого лабораторного аналізу. Традиційне аналітичне тестування в лабораторії не вимагається в деяких випадках, оскільки аналіз проводиться на етапі виробництва. Наприклад, під час аналізу харчових продуктів клієнти переходять на онлайн-аналізатор ближнього інфрачервоного випромінювання, оскільки вони можуть знизити експлуатаційні витрати, виключивши аналіз контролю якості, який виконується окремо в лабораторії. Корпорація Buchi придбала компанію під назвою NIR-Online для надання онлайн-аналізатора NIR.

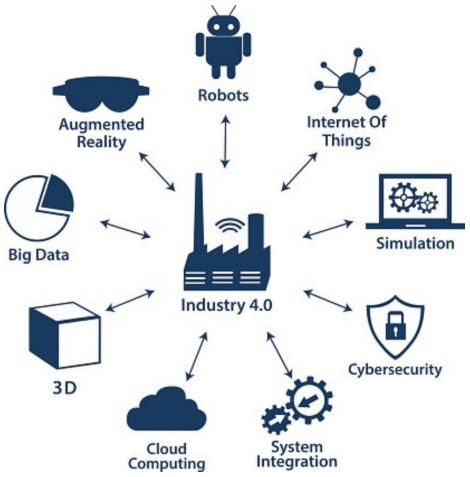

Рисунок 1.3 Індустрія 4.0

Цифрова трансформація з IoT/IIoT у виробництві забезпечує надійне спілкування, розуміння та можливості для інновацій, які передбачають

організацію речей, людей і систем у наскрізних автоматизованих потоках створення вартості.

IoT в охороні здоров'я, або Інтернет медичних речей.

Подібно до промисловості, IoT має потенціал для перевизначення системи охорони здоров'я. Інтернет медичних речей (IoMT) — це фактично набір медичних пристроїв і програм, які підключаються до IT-систем охорони здоров'я через комп'ютерні мережі онлайн.

Зі збільшенням кількості підключених медичних пристроїв потужність IoMT зростає разом із розширенням сфери його застосування, будь то дистанційний моніторинг пацієнтів із хронічними або тривалими захворюваннями, відстеження замовлень пацієнтів на ліки або носінні пристрої mmHealth пацієнтів, які можуть надсилати інформацію особам, які здійснюють догляд. Ця нова практика використання пристроїв IoMT для дистанційного моніторингу пацієнтів у їхніх домівках позбавляє їх від поїздок до лікарні щоразу, коли у них виникають медичні питання або зміна їхнього стану, що змінює всю екосистему охорони здоров'я та налаштування спілкування між лікарем і пацієнтом.

1.3. Програмно-апаратні засоби для реалізації системи вуличного радіо

Для реалізації системи вуличного радіо можна використати різні технології, проте на мою думку, у цьому проєкті найбільш доцільно використовувати такий стек технологій:

- Програмне забезпечення сервера вуличного радіо: Python з бібліотекою Django та REST API.
- Апаратна реалізація клієнта вуличного радіо: Raspberry PI.
- Програмне забезпечення клієнта вуличного радіо: Python.
- Інтерфейс адміністратора системи: Next.js.
- Система обміну повідомленнями (відкладені завдання): RabbitMQ.

- База даних: PostgreSQL.
- Сховище об'єктів: MinIO.

Raspberry Pi.

Raspberry Pi — це серія недорогих програмованих комп’ютерів, які включають набір GPIO, або «Інтерфейс введення/виведення загального призначення», контактів, які можна використовувати для підключення та керування зовнішніми електричними пристроями, а також для створення Інтернету речей (IoT). рішення.

Точна кількість і роль контактів змінюється між окремими моделями, але зазвичай поділяється на контакти живлення, заземлення та контакти загального призначення. Виводи живлення та заземлення не програмуються. Вивід живлення забезпечує постійне живлення 3,3 В/5 В, тоді як контакт заземлення використовується для підключення до катода схеми.

Виводи загального призначення повністю програмуються і можуть використовуватися як у виведення, так і в режимі введення. У режимі виведення контакти забезпечують постійне живлення 3,3 В, яке можна вмикати та вимикати. У режимі входу штифт зчитує струм, що постачається ланцюгом, і повертає логічне значення, яке вказує, отримує він живлення 3,3 В чи ні.

Звичайно, ці можливості не є новими та були широко доступні для розробників через мікроконтролери, такі як Arduino або NodeMCU[22]. Однак ці пристрої зазвичай мали обмежену пам’ять і обчислювальну потужність і вимагали використання певних мов програмування.

Python.

Python — це високорівнева об'єктноорієнтована мова для програмування загального призначення. Інтерпретована мова Python є кросплатформною. Багато операційних систем, таких як Ubuntu, включають Python як стандартний компонент.

Python був задуманий наприкінці 1980-х одним зі співробітників Centrum Wiskunde & Informatica, Гвідо ван Россумом, і вийшов у 1991 році. Враховуючи цей факт, ми можемо зробити висновок, що Python – це мудра мова з великою

кількістю готових до використання бібліотеки та фреймворки. За всі роки існування він став популярним і створив величезну спільноту, яка завжди готова допомогти вам. В Інтернеті можна знайти багато документації та прикладів реалізації.

Python має суворий узгоджений синтаксис і модульну структуру програми. Можна легко визначити, чи ви програміст вперше, чи маєте досвід роботи з іншими мовами. Не загромаджений візуальний макет допомагає програмістам створювати легко читабельний код.

Ядро Python має мінімалістичний синтаксис, але стандартна бібліотека містить багато корисних функцій, таких як доступ до системних ресурсів, файлової системи тощо.

На Python вплинули інші мови програмування. Для цього знадобилася об'єктноорієнтована парадигма Smalltalk, функціональне програмування Lisp (лямбда, карта тощо), робота з масивами та складна арифметика Fortran, а також декоратори та обробники винятків Java. Python підтримує всі основні дані, які використовуються в сучасному програмуванні.

Як універсальна мова, Python дозволяє вирішувати широкий спектр завдань, таких як автономний скрипт, програми, що керують автоматизацією та роботами, створення вебдодатків та вебсайтів. Для конкретних завдань можна використовувати розширення, написане, наприклад, мовою C.

Можливо, хтось вважатиме недоліком низьку швидкість виконання коду в порівнянні зі скомпільованим, але, судячи з обчислювальної потужності сучасних ПК, навряд чи це буде настільки критично.

Next.js.

Next.js — це фреймворк для React.js, це означає, що це обгортка поверх React, надаючи загальні функції React з додатковими можливостями.

Щоб створити повноцінний вебдодаток з React з нуля, необхідно враховувати багато важливих деталей:

Код потрібно об’єднати за допомогою пакета, такого як webpack, і трансформувати за допомогою компілятора, такого як Babel.

В Next.js відтворення на стороні клієнта динамічно керує маршрутизацією, не оновлюючи сторінку щоразу, коли користувач запитує інший маршрут. Але рендеринг на стороні сервера може зобразити повністю заповнену сторінку під час першого завантаження для будь-якого маршруту вебсайту, тоді як візуалізація на стороні клієнта спочатку зображає порожню сторінку.

Особливості Next.js[42]:

- Інтуїтивно зрозуміла система маршрутизації на основі сторінок (з підтримкою динамічних маршрутів).
- Попереднє відтворення, як статична генерація (SSG), так і рендеринг на стороні сервера (SSR) підтримуються на основі кожної сторінки.
- Автоматичне розділення коду для пришвидшення завантаження сторінки.
- Маршрутизація на стороні клієнта з оптимізованою попередньою вибіркою.
- Вбудована підтримка CSS і Sass, а також підтримка будь-якої бібліотеки CSS-in-JS.
- Середовище розробки з підтримкою швидкого оновлення.
- Маршрути API для створення кінцевих точок API з безсерверними функціями.
- Повністю розширюваний.

RabbitMQ.

AMQP (Advanced Message Queuing Protocol) – відкритий протокол для менеджменту чергами повідомлень між елементами системи. Основна ідея полягає в тому, що окремі підсистеми можуть обмінюватися довільним чином повідомленнями через AMQP брокер, який здійснює маршрутизацію, можливо, гарантує доставлення, розподіл потоків даних, підписку на потрібні типи повідомлень[17].

Протокол AMQP вводитиме три поняття (рисунок 1.4):

- Exchange (обмінник або точка обміну) — надсилаються повідомлення. Обмінник розподіляє повідомлення на одну або кілька черг. Він спрямовує повідомлення в чергу на основі створених зв'язків (binding) між ним та чергою.

● Queue (черга) — структура даних на диску або оперативної пам'яті, яка зберігає посилання на повідомлення та віддає копії повідомлень consumers (споживачам). Одна черга може використовуватися кількома споживачами.

● Binding (прив'язка) — правило, яке повідомляє точці обміну в яку черги ці повідомлення повинні попадати. Обмінник та черга можуть бути пов'язані кількома прив'язками.

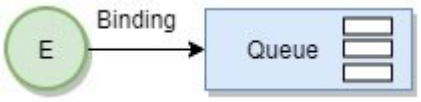

Рисунок 1.4 Візуалізація протоколу AMQP

Протокол AMQP працює поверх TCP/IP.

RabbitMQ – це брокер повідомлень із відкритим вихідним кодом. Він маршрутизує сполучення за всіма базовими принципами протоколу AMQP, описаними в специфікації. Відправник передає повідомлення брокеру, а той доставляє його одержувачу. RabbitMQ реалізує та доповнює протокол AMQP.

Основна ідея моделі обміну повідомленнями у RabbitMQ полягає в тому, що producer (видавець) не надсилає повідомлення безпосередньо в чергу. Насправді й досить часто видавець навіть не знає, чи повідомлення взагалі буде доставлене в будь-яку чергу.

Натомість видавець може надсилати повідомлення лише на обмін. З одного боку, обмін отримує повідомлення від видавців, з другого — відправляє їх у черги. Обмін повинен точно знати, що робити з отриманим повідомленням. Чи має воно бути додане у певну чергу? Чи має бути додано до кількох черг? Або повідомлення потрібно ігнорувати.

Коротко роботу RabbitMQ можна описати так (рисунок 1.5):

1. Видавець надсилає повідомлення певному обміннику.

2. Обмінник, отримавши повідомлення, маршрутизує його в одну або кілька черг відповідно до правил прив'язки між ним та чергою.

3. Черга зберігає посилання цього повідомлення. Саме повідомлення зберігається в оперативній пам'яті або на диску.

4. Як тільки споживач готовий отримати повідомлення з черги, сервер створює копію повідомлення за посиланням та надсилає.

5. Споживач отримує повідомлення та надсилає брокеру підтвердження.

6. Брокер, отримавши підтвердження, видаляє копію повідомлення з черги. Потім видаляє з оперативної пам'яті та з диска.

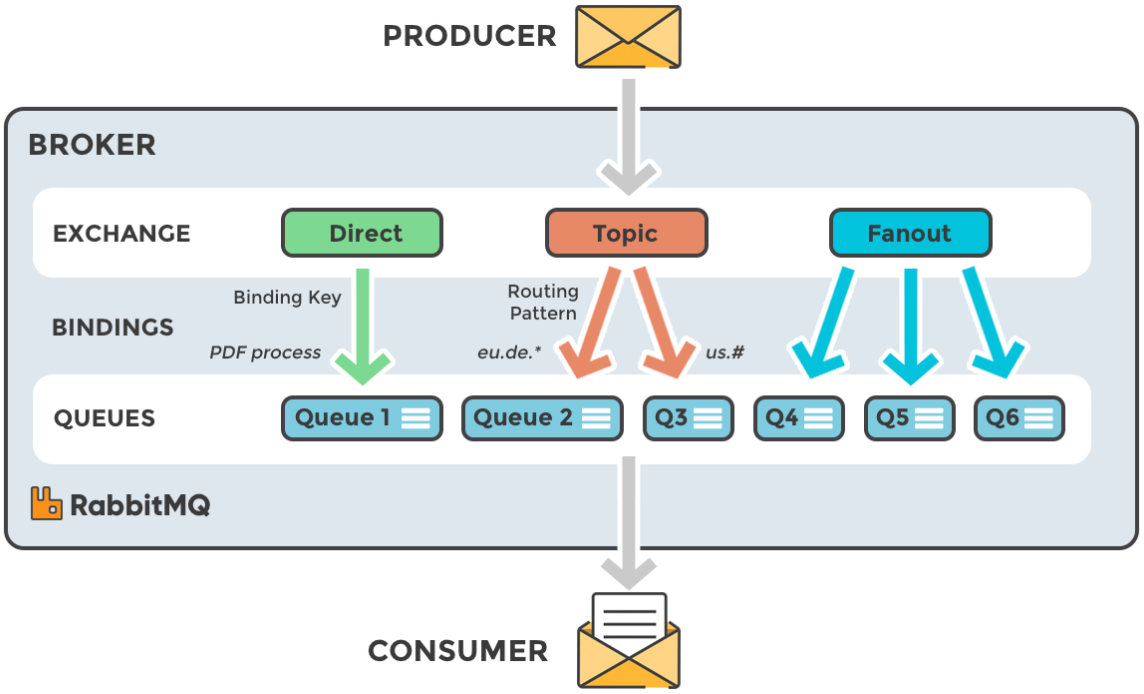

Рисунок 1.5 Візуалізація роботи RabbitMQ

В системі вуличного радіо використовується база даних PostgreSQL для зберігання даних необхідних даних для роботи менеджмент сервера, дані про станції клієнти, дані черги відтворення.

База даних корпоративного класу, PostgreSQL може похвалитися такими складними функціями, як керування кількома версіями одночасного керування (MVCC), відновлення моменту часу, табличні простори, асинхронна реплікація, вкладені транзакції, онлайн/гаряче резервне копіювання, складний планувальник

запитів/оптимізатор, а також попереднє записування журналів для відмовостійкості.

PostgreSQL працює на більшості популярних операційних систем – майже на всіх дистрибутивах Linux і Unix, Windows, Mac OS X. Його природа з відкритим кодом дозволяє легко оновлювати або розширювати його. У PostgreSQL ви можете визначати власні типи даних, створювати власні функції та навіть писати код іншою мовою програмування (наприклад, Python) без перекомпіляції бази даних. І, звісно, PostgreSQL безплатний!

Надійна база даних.

PostgreSQL не просто реляційний, він об'єктно-реляційний і підтримує складні структури та широкий спектр вбудованих і визначених користувачем типів даних. Він забезпечує велику місткість даних і користується довірою за його цілісність даних. Це дає йому деякі переваги перед іншими базами даних SQL з відкритим вихідним кодом, такими як MySQL, MariaDB і Firebird. PostgreSQL має багато функцій, які допомагають розробникам створювати програми, адміністраторам — захищати цілісність даних і створювати відмовостійкі середовища[33].

Він пропонує своїм користувачам величезну (і висхідну) кількість функцій. Вони допомагають програмістам створювати нові програми, адміністраторам краще захищати цілісність даних, а розробникам створювати стійкі та безпечні середовища. PostgreSQL дає своїм користувачам можливість керувати даними, незалежно від того, наскільки велика база даних.

Розширювана база даних.

Окрім того, що PostgreSQL є безплатний і з відкритим вихідним кодом, він дуже розширюваний. Є дві області, які PostgreSQL сяє, коли користувачам потрібно налаштувати та контролювати свою базу даних. По-перше, він у високій мірі відповідає стандартам SQL. Це підвищує його сумісність з іншими програмами.

По-друге, PostgreSQL надає користувачам контроль над метаданими. PostgreSQL є розширюваним, оскільки його робота керується каталогом. Однією з

ключових відмінностей PostgreSQL від стандартних систем реляційних баз даних є те, що PostgreSQL зберігає набагато більше інформації у своїх каталогах: не тільки інформацію про таблиці та стовпці, а й інформацію про типи даних, функції, методи доступу тощо. Ці таблиці можуть бути змінені користувачем, і оскільки PostgreSQL базує свою роботу на цих таблицях, це означає, що PostgreSQL може бути розширений користувачами. Для порівняння, звичайні системи баз даних можна розширити лише шляхом зміни жорстко закодованих процедур у вихідному коді або завантаження модулів, спеціально написаних постачальником СУБД.

PostgreSQL має багату історію підтримки розширених типів даних і підтримує рівень оптимізації продуктивності, який зазвичай асоціюється з комерційними аналогами баз даних, такими як Oracle і SQL Server. PostgreSQL використовується як основне сховище даних або сховище даних для багатьох веб, мобільних, геопросторових та аналітичних додатків.

PostgreSQL може зберігати структуровані та неструктуровані дані в одному продукті. Неструктуровані дані, що містяться в аудіо, відео, електронних листах та публікаціях у соціальних мережах, можуть бути використані для покращення обслуговування клієнтів, виявлення нових вимог до продуктів і пошуку способів запобігти скупчення клієнтів серед незліченних інших видів використання[48].

PostgreSQL також має чудові можливості обробки онлайн-транзакцій (OLTP) і може бути налаштований на автоматичне відключення і повне резервування, що робить його придатним для фінансових установ і виробників. Бувши високоздатною аналітичною базою даних, її можна ефективно інтегрувати з математичним програмним забезпеченням, таким як Matlab і R. Завдяки можливостям реплікації PostgreSQL вебсайти можна легко масштабувати до будь-якої кількості серверів баз даних, скільки вам потрібно.

PostgreSQL при використанні з розширенням PostGIS підтримує географічні об’єкти й може використовуватися як сховище геопросторових даних для служб на основі розташування та геоінформаційних систем (ГІС).

Моделювання даних — це процес створення візуального уявлення або всієї інформаційної системи, або її частин для передачі зв'язків між точками даних і структурами. Мета полягає в тому, щоб проілюструвати типи даних (рисунок 1.6), які використовуються та зберігаються в системі, взаємозв'язки між цими типами даних, способи групування та організації даних, а також їх формати та атрибути.

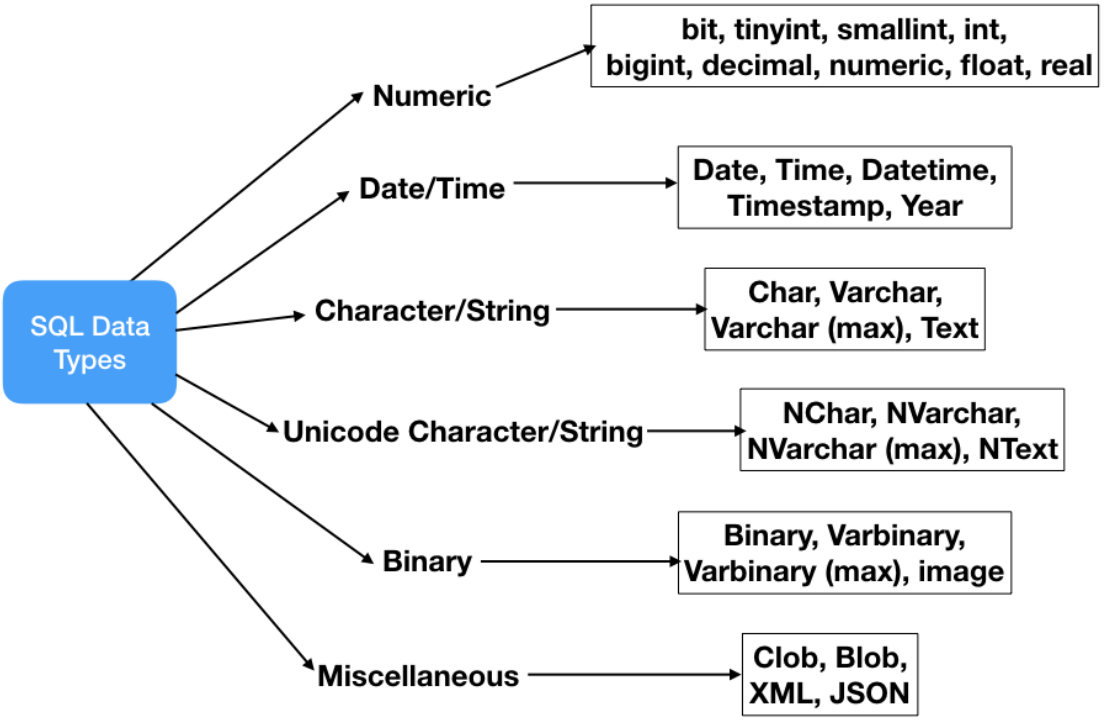

Рисунок 1.6 Типи даних SQL

Моделі даних будуються на основі потреб бізнесу. Правила та вимоги визначаються заздалегідь на основі зворотного зв'язку з зацікавленими сторонами бізнесу, щоб їх можна було включити в дизайн нової системи або адаптувати під час наявної ітерації.

Дані можна моделювати на різних рівнях абстракції. Процес починається зі збору інформації про бізнес-вимоги від зацікавлених сторін і кінцевих користувачів. Ці бізнес-правила потім перекладаються в структури даних для

формування конкретного дизайну бази даних. Модель даних можна порівняти з дорожньою картою, планом архітектора або будь-якою формальною схемою, яка полегшує глибше розуміння того, що проєктується.

Моделювання даних використовує стандартизовані схеми та формальні методи. Це забезпечує загальний, послідовний і передбачуваний спосіб визначення ресурсів даних і керування ними в організації або навіть за її межами.

Моделювання даних розвивалося разом із системами керування базами даних, причому складність типів моделей зростала в міру зростання потреб підприємств у зберіганні даних. Ось кілька типів моделей:

Ієрархічні моделі даних представляють відносини один до багатьох у деревоподібному форматі. У цьому типі моделі кожен запис має один корінь або батьківський елемент, який зображається на одну або більше дочірніх таблиць. Ця модель була реалізована в IBM Information Management System (IMS), яка була представлена в 1966 році й швидко знайшла широке застосування, особливо в банківській справі. Хоча цей підхід менш ефективний, ніж нещодавно розроблені моделі баз даних, він все ще використовується в системах Extensible Markup Language (XML) і географічних інформаційних системах (GIS).

Реляційні моделі даних спочатку були запропоновані дослідником IBM Е. Ф. Коддом у 1970 році. Вони досі реалізуються в багатьох різних реляційних базах даних, які зазвичай використовуються в корпоративних обчисленнях. Реляційне моделювання даних не вимагає детального розуміння фізичних властивостей використовуваного сховища даних. У ньому сегменти даних чітко об’єднуються за допомогою таблиць, що зменшує складність бази даних[33].

MinIO.

MinIO — це сервер зберігання об’єктів (рисунок 1.7), який реалізує той самий публічний API, що й Amazon S3. Це означає, що програми, які можна налаштувати на розмову з Amazon S3, також можна налаштувати на розмову з Minio. Сховище об’єктів, наприклад Minio, можна використовувати для зберігання неструктурованих даних, таких як фотографії, відео, файли журналів, резервні копії та зображення контейнерів/віртуальних машин. Розмір окремого об’єкта

може варіюватися від кількох КБ до максимум 5 ТБ. Файли організовані в так званих «секціях», які є логічними роздільниками ваших збережених даних, і їх потрібно передати вашій програмі разом із вашим ключем доступу, секретним ключем і HTTP-адресою екземпляра Minio.

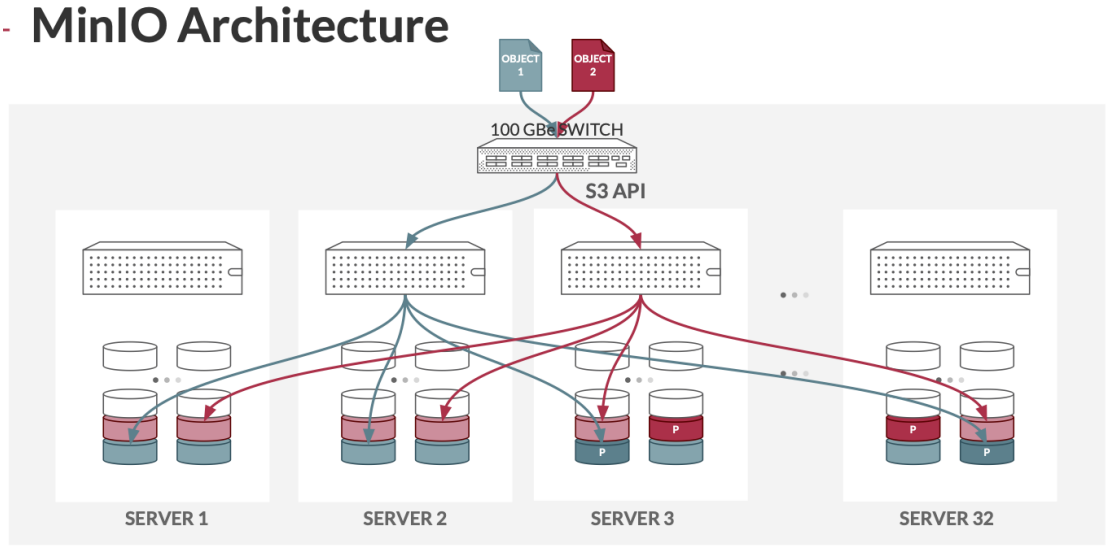

Рисунок 1.7 Архітектура MinIO

1.4. Висновки до розділу

Побудова розумного міста – це системний процес, який ведеться з часом, крок за кроком. Розробка та впровадження додатків розумного міста є надзвичайно складною, а вибір та використання відповідних методів та інструментів DM для комунікації між реальним і цифровим світами відіграє вирішальну роль для успіху роботи. У даному розділі було надано вичерпне уявлення про дослідження, опубліковані в літературі, пов’язані з алгоритмами DM для розумних міст, на основі бібліометричного аналізу з використанням даних за період з 2017 по лютий 2021 року.

Дослідження показало, що алгоритми DM для розумних міст є сферою досліджень, що розвивається і швидко розвивається, оскільки вони багаторазово зростали за останні 8 років. Ця тема була найбільш популярною серед дослідників у Китаї, США, Індії, Іспанії, Великобританії та Греції та стала благодатним полем для співпраці між дослідниками з різних країн, особливо між Китаєм та США.

Інтеграція різних технологій, які використовуються в додатках і сервісах розумного міста, залишається найскладнішою проблемою, яку необхідно подолати через обсяг, неоднорідність і складність зібраних даних.

Майбутні тенденції в розумних містах свідчать про безперервну автоматизацію його транспорту за допомогою автономних засобів самоуправління та транспортних засобів. Інфраструктура буде все більше взаємопов’язана з цифровізацією її потреб в енергії, представлених у Smart Grid. Великі дані дозволятимуть громадянам або користувачам краще використовувати простір, послуги та структури розумного міста.

Апаратні та програмні технології, доступні для реалізації вуличного радіо, в останні роки швидко розвиваються. Як і у всіх технологічних проектах, потрібно провести лінію та впровадити наявне обладнання. Доступні сьогодні опції дозволяють простій інфраструктурі досягти тих же результатів. Численні оновлення мікропрограми та оптимізації налаштувань перетворилися більше на «мистецтво», а не на «науку». Тим не менш, для міст відкриваються великі переваги отримання аналогової та статусної інформації з віддалених пристроїв. Крім того, функція віддаленого адміністративного доступу виявилася чудовою перевагою.

2. АЛГОРИТМИ ТА МЕТОДИ РЕТРАНСЛЯЦІЇ ЗВУКОВИХ ПОВІДОМЛЕНЬ В РЕАЛЬНОМУ ЧАСІ ДЛЯ СИСТЕМИ ВУЛИЧНОГО РАДІО "РОЗУМНОГО" МІСТА

2.1. Опис системи вуличного радіо та системи “розумного міста”

Система вуличного радіо передбачає відтворення звукових повідомлень в режимі реального часу та згідно з чергою яку задав адміністратор системи. Також ця система може бути використана для відтворення сповіщень в “розумному” місті, за умови інтеграції цих систем.

Система сповіщень – це набір протоколів і процедур, які можуть залучати як людину, так і комп’ютерні компоненти. Мета цих систем — генерувати та надсилати своєчасні повідомлення особі чи групі людей. Прості системи сповіщень використовують єдиний засіб зв’язку, наприклад електронну пошту або текстове повідомлення. Складніші системи, призначені для розсилання важливої інформації, зазвичай використовують інші методи комунікації, а також можуть включати людські елементи, щоб гарантувати, що кожна людина дійсно отримує повідомлення[17]. Системи оповіщення про надзвичайні ситуації, якими керують державні установи, можуть використовувати телефонну систему, телевізійне мовлення та широкий спектр інших методів зв’язку (рисунок 2.1).

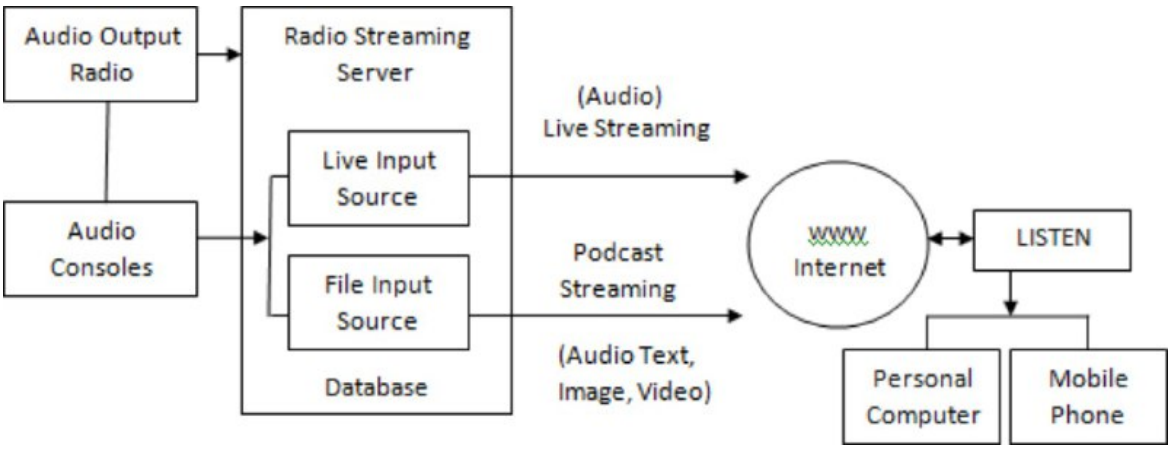

Рисунок 2.1 Структурна схема роботи радіо системи

Основна мета системи сповіщень – попередити людину про те, що сталася конкретна подія. Прикладом простої системи сповіщень є те, як користувач може налаштувати вебсайт для надсилання електронного листа, коли відбувається оновлення або інша подібна подія. Комп'ютери, телефони та інші пристрої використовують прості системи сповіщень, щоб сповіщати користувачів про вхідні текстові повідомлення, електронні листи та інші події. Більшість із цих систем мають лише один спосіб сповіщення, особливо якщо інформація, яка передається, не вважається критичною.

Складніші системи сповіщень зазвичай використовуються в ситуаціях, коли дуже важливо негайно зв'язатися з людьми. Багато компаній використовують якийсь тип розширеної системи сповіщень, щоб гарантувати, що критично важливі співробітники завжди будуть під рукою. Якщо виникла надзвичайна ситуація, яка вимагає уваги одного або кількох співробітників, можна активувати систему оповіщення. Початковий контакт зазвичай здійснюється за допомогою текстового повідомлення, електронної пошти або телефону. Якщо початковий контакт не вдається, деякі системи можуть перейти до інших співробітників, які можуть замість цього вирішити ситуацію.

Уряди також використовують складні системи сповіщень, щоб попередити громадян про майбутні катастрофи чи інші важливі події. Двома методами, які часто включають в цей тип системи сповіщень, є телевізійне мовлення та телефонні системи. У деяких випадках телефонну систему екстреної допомоги можна використовувати для виклику жителям певного району, хоча це може не попередити людей, які не мають стаціонарних телефонів. Деякі органи місцевого самоврядування підтримують бази даних системи екстрених сповіщень, які дозволяють громадянам залишати номери мобільних телефонів, адреси електронної пошти та іншу контактну інформацію. Іноді в системах сповіщень також присутній людський елемент, і в такому випадку можна очікувати, що люди попередять своїх друзів і сусідів або навіть певний список людей.

Кому потрібна система оповіщення про надзвичайні ситуації.

Надзвичайні ситуації можуть виникнути з будь-якого місця в будь-який час. Будь-яке підприємство з більш ніж кількома співробітниками повинно розглядати систему оповіщення про надзвичайні ситуації як невіддільну частину своїх планів безперервності бізнесу. Для організацій зі співробітниками, які працюють віддалено, подорожують, їздять або працюють з різних корпоративних об'єктів, дуже важливо, щоб вони могли миттєво зв'язатися з кількома каналами.

Системи оповіщення про надзвичайні ситуації ідеально підходять для підприємств, медичних закладів і лікарень, шкіл, некомерційних організацій, членських організацій та будь-якої іншої організації, яка потребує єдиної комунікаційної платформи. Ваші плани спілкування повинні включати кожного співробітника, кожного студента, кожного співробітника, кожного віддаленого працівника. Найкраще програмне забезпечення для оповіщення про надзвичайні ситуації дозволить будь-якій організації, приватній чи державній, швидко використовувати свої комунікаційні можливості з будь-якою аудиторією[45].

Залежно від того, де розташована ваша організація, приміщення та співробітники, будуть різні загрози. Однак найпоширеніші загрози здатні вплинути на кожного. Не дивно, що збої в IT є найпоширенішою причиною, чому організації активують свої плани екстрених сповіщень. Ось як оцінюються інші надзвичайні ситуації в організаціях:

- 50% – відключення IT
- 49% – інциденти, пов'язані з погодою
- 47% – відключення електроенергії
- 45% – стихійні лиха
- 42% – пожежа
- 38% – інциденти з управління об'єктами
- 33% – питання безпеки
- 32% – інциденти зі здоров'ям та безпекою
- 28% – інциденти кібербезпеки
- 24% – збої в дорозі

Кожна організація схильна до відключень IT та електроенергії. Ці збої можуть тривати від секунд до днів, спричиняючи втрату компаній мільйонів доларів, частки ринку клієнтів та гарної репутації. Кожна сфера бізнесу може постраждати, оскільки процеси та операції різко зупиняються. Сучасний споживач став нетерплячим і не терпить перебоїв у роботі. Чим швидше компанія зможе розповсюдити та передати інформацію своїм співробітникам про відключення, тим швидше вона зможе повернутися до Інтернету або, принаймні, вжити активних заходів для подолання кризи. Повідомлення про надзвичайну ситуацію для всіх співробітників може допомогти організаціям ефективно керувати такими подіями.

Інциденти, пов’язані з погодою, відрізняються залежно від місця розташування, але всі організації піддаються ризику. Хоча ваша організація не може запобігти урагану, торнадо, вона може дати своїм співробітникам багато попереджень про небезпеку, що насувається. Він може надати відповідну інформацію про місця укриття, інструкції з евакуації та інші заходи безпеки. Рішення для аналізу загроз та моніторингу може допомогти вам швидко визначити загрози, а також будь-яких співробітників, які постраждали. Як тільки ви виявите загрозу, ваша система оповіщення про надзвичайні ситуації може допомогти вам надіслати вказівки щодо безпеки тим, хто постраждав, або кроки, щоб відновити роботу вашого бізнесу, критично важливому персоналу підтримки.

Пожежа – це ще одна загроза, з якою стикається кожна компанія, будь то лісова пожежа або внутрішня пожежа. У будівлі можуть бути розміщені маршрути евакуації, але ще більш ефективним заходом захисту було б активувати систему екстрених сповіщень для електронної пошти, текстових повідомлень, дзвінків, надсилання push-повідомлень, оновлення інтернет-сайтів та сайтів соціальних мереж, а також надсилання сповіщень за користувацькими каналами. У таких ситуаціях корисно використовувати програмне забезпечення, яке дозволяє адміністраторам сегментувати співробітників за їх місцем розташування, оскільки не кожен працівник може постраждати. Такі повідомлення повинні отримувати лише ті працівники, які знаходяться в безпосередній небезпеці. Після події

систему також можна використовувати для інформування всіх співробітників про те, що сталося.

Кіберзагрози — це надзвичайні ситуації, про які ми часто не думаємо відразу, коли розглядаємо системи сповіщення про надзвичайні ситуації, але вони все частіше становлять значну загрозу для будь-якої організації. Дані, особливо дані клієнтів, мають великий попит. Хакери постійно шукають способи використати слабкі місця безпеки для доступу до цієї інформації. Насправді за оцінками експертів, кібератаки вже коштують бізнесу 400 мільярдів доларів на рік, і, як очікується, до 2025 року ця цифра зросте до понад 10 трильйонів доларів на рік.

Організації повинні серйозно ставитися до цих загроз і мати план ефективної комунікації зі співробітниками у разі атаки. Компанії потрібно буде використовувати систему сповіщень про надзвичайні ситуації, щоб інформувати співробітників, а також надавати їм інформацію, необхідну для підтримки бізнесу, якнайшвидшого відновлення систем і боротьби з неминучими суспільними негативними впливами. Так само як відключення IT або електроенергії, кіберзагрози можуть перервати або навіть зупинити бізнес-операції. Співробітники повинні знати, як реагувати як на споживачів, партнерів, постачальників та інших зацікавлених сторін, так і на критично важливі бізнес-системи.

Адміністрування системою вуличного радіо відбувається за допомогою інтерфейсу адміністратора системи. В ньому адміністратор може переглядати статус станцій, керувати чергою відтворення та транслювати звукові повідомлення в режимі реального часу.

Надалі система вуличного радіо може бути інтегрована з системою “розумного” міста за допомогою API. Саме ця інтеграція дозволить місту, в реальному часі, сповістити мешканців в разі виникнення такої необхідності[6].

2.2. Алгоритми реалізації системи вуличного радіо для системи “розумного” міста

Вебфреймворк пропонує набір інструментів багаторазового використання компонентів, що дозволяє розробникам зосередитися на швидкому створенні свого додатка, не винаходячи велосипед. У кожній вебпрограмі виникають проблеми, як-от як зіставити певну URL-адресу з набором коду або як динамічно створити сторінку за деякими даними з вашої бази даних. Вебфреймворк дозволить вам зробити це швидко, щоб ви могли зосередитися на написанні коду для даної URL-адреси або вмісту певної сторінки. Ці дві основні проблеми називаються відповідно маршрутизацією та шаблонуванням і є основними компонентами багатьох вебфреймворків. Фронтенд і вебфреймворки абсолютно різні[16].

Django — один з найпопулярніших і величезних вебфреймворків з відкритим кодом. Він написаний на Python, і такі компанії, як Instagram, Doordash і Disqus, використовують його у своїх системах, чого, сподіваємося, достатньо, щоб показати, що Django є надійним і масштабованим.

Архітектура Django.

Django дотримується архітектури Model-Template-View (MTV) (рисунок 2.1), яка поділена на три різні частини:

- Модель — це логічна структура даних, що лежить в основі всієї програми, і представлена базою даних (зазвичай реляційними базами даних, такими як MySql, Postgres).
- Шаблон — це інтерфейс користувача — те, що ви бачите у своєму браузері, коли відвідуєте вебсайт. Вони представлені файлами HTML/CSS/Javascript.
- Подання є посередником, який з’єднує подання і модель разом, тобто він передає дані від моделі до представлення.

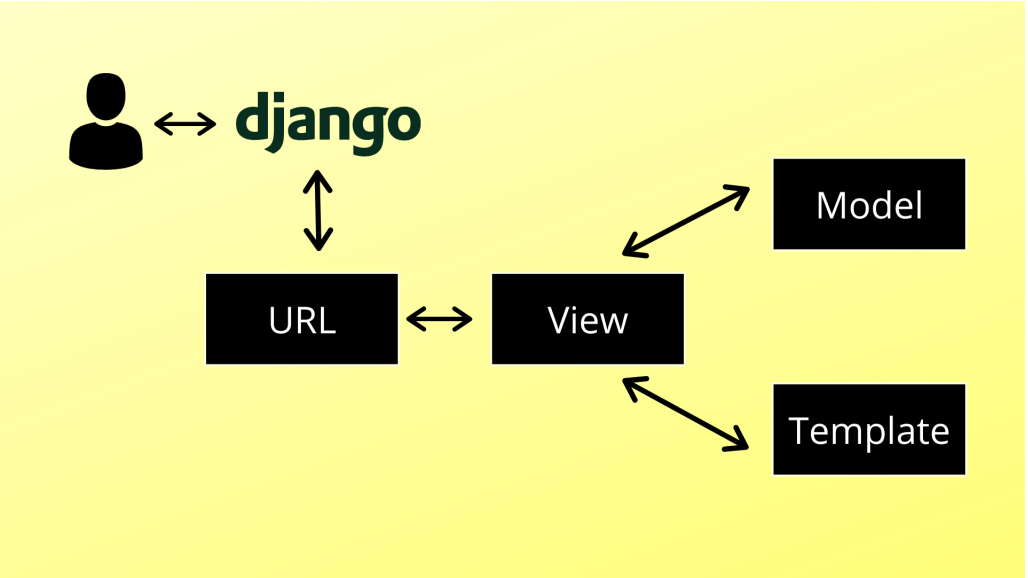

Рисунок 2.2 Model-View-Template

З MTV ваша програма обертатиметься навколо моделі — або зображати її, або маніпулювати нею.

Отже, скажімо, що користувач введе URL-адресу у своєму браузері, цей запит буде проходити через Інтернет-протоколи на ваш сервер, який викличе Django. Потім Django обробить вказаний шлях URL-адреси, і якщо він збігається з URL-шляхом, який ви явно вказали, він викличе контролер, який потім виконає певну дію, наприклад, отримає запис із вашої моделі (бази даних), а потім відтворить Перегляд (тобто: текст JSON, вебсторінка HTML/CSS/JavaScript).

2000 року Рой Філдінг написав докторську дисертацію, де виклав концепцію REST. Там були рекомендації про те, як спроєктувати Сервер, щоб було зручно спілкуватися з Клієнтами. Виділю два головні принципи створення додатків у стилі REST:

- Сервер не повинен нічого знати про стан Клієнта. У запиті від Клієнта має бути вся потрібна інформація для обробки цього запиту Сервером.
- Кожен ресурс на Сервері повинен мати певний Id, а також унікальний URL, яким здійснюється доступ до цього ресурсу.

На цей час ми можемо знайти фреймворк для створення програм у стилі REST практично для кожної мови програмування, що використовується у

веброзробці. Логіка побудови Web API на Сервері у цих фреймворках реалізована однаково.

Дії управління даними прив'язані до певних HTTP-методів. Існує кілька стандартних дій для роботи з даними – це Create, Read, Update, Delete. Часто їх узагальнюють як CRUD. В таблиці 2.1. описано використання HTTP методів в REST API[34].

Таблиця 2.1 — Використання HTTP-методів в REST API

HTTP метод	**Призначення**
GET	Для читання об’єктів
POST	Для створення об'єктів
PATCH	Для часткового оновлення об'єкту
PUT	Для повного оновлення об'єкту
DELETE	Для видалення об'єктів
OPTIONS	Для опису параметрів з'єднання

Для зручної роботи з API для Django існує Django Rest Framework.

Django Rest Framework (DRF) — це бібліотека, яка працює зі стандартними моделями Django для створення гнучкого та потужного API для проєкту.

Основна архітектура.

API DRF складається з 3-х шарів: серіалізатора, подання та маршрутизатора.

Серіалізатор: перетворює інформацію, що зберігається в базі даних та визначену за допомогою моделей Django, у формат, який легко та ефективно передається через API.

Подання (ViewSet): визначає функції (читання, створення, оновлення, видалення), які будуть доступні через API.

Маршрутизатор: визначає URL-адреси, які надаватимуть доступ до кожного виду.

Серіалізатори.

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

Серіалізатор DRF здійснює це перетворення. Коли користувач передає інформацію (наприклад, створення нового екземпляра) через API, серіалізатор бере дані, перевіряє їх і перетворює на те, що Django може скласти в екземпляр моделі. Аналогічно, коли користувач звертається до інформації через API, відповідні екземпляри передаються в серіалізатор, який перетворює їх у формат, який може бути легко переданий користувачеві як JSON.

Найбільш поширеною формою, яку приймає серіалізатор DRF, є той, який прив'язаний безпосередньо до моделі Django:

```
class DataSerializer(serializers.ModelSerializer):
    class Meta:
        model = Data
        fields = ('filed',)
```

Налаштування fields дозволяють точно вказати, які поля доступні для цього серіалізатора. Замість нього може бути встановлений exclude замість fields, яке буде включати всі поля моделі, крім тих, які вказані в exclude.

Подання (ViewSet).

Серіалізатори – це неймовірно гнучкий та потужний компонент DRF. Хоча підключення серіалізатора до моделі є найпоширенішим, серіалізатори можуть використовуватися для створення будь-якої структури даних Python через API відповідно до певних параметрів.

Серіалізатор аналізує інформацію в обох напрямках (читання та запис), тоді як ViewSet - це код, в якому визначені доступні операції. Найбільш поширеним ViewSet є ModelViewSet, який має такі вбудовані операції:

- Створення примірника: create ()
- Отримання / читання примірника: retrieve ()
- Оновлення екземпляра (усі або вибрані поля): update () або partial_update ()

- Знищення / Вилучення екземпляра: destroy ()
- Список екземплярів (з розбивкою за сторінками за замовчуванням): list ()

Кожна з цих функцій може бути перевизначена, якщо потрібна інша поведінка, але стандартна функціональність працює з мінімальним кодом, а саме:

```
class ThingViewSet(viewsets.ModelViewSet):
    queryset = Thing.objects.all()
```

Якщо потрібні додаткові налаштування, можна використовувати загальні уявлення замість ModelViewSet або навіть окремі уявлення.

Маршрутизатори.

І, нарешті, маршрутизатори: вони надають верхній рівень API. Щоб уникнути створення нескінченних URL-адрес: «списки», «деталі» та «змінити», маршрутизатори DRF об'єднують усі URL-адреси, необхідні для цього типу в один рядок для кожного ViewSet.

Авторизація станцій на сервері вуличного радіо.

OAuth 2.0 — це галузевий стандартний протокол авторизації. OAuth 2.0 зосереджується на простоті клієнта для розробників, забезпечуючи конкретні потоки авторизації для вебпрограм, настільних програм, мобільних телефонів і пристроїв у вітальні. Ця специфікація та її розширення розробляються в рамках робочої групи IETF OAuth.

Щоб отримати маркер доступу клієнту і використовувати його для доступу до захищених ресурсів сервера є чотири потоки авторизації (рисунок 2.2):

- Authorization Code Grant: код видається і використовується для отримання access_token. Цей код випускається в інтерфейсну програму (у браузері) після входу користувача. Замість цього на стороні сервера видається access_token, який аутентифікує клієнта за допомогою пароля та отриманого коду.
- Implicit Grant: після того, як користувач входить в систему, access_token видається негайно.
- Client Credential Grant: access_token видається на сервері, аутентифікуючи тільки клієнта, а не користувача.

- Password Grant: access_token видається негайно з одним запитом, що містить всю інформацію для входу: ім'я користувача, пароль користувача, ідентифікатор клієнта та секрет клієнта. Це може здатися легшим у реалізації, але це має деякі складнощі та вразливості.

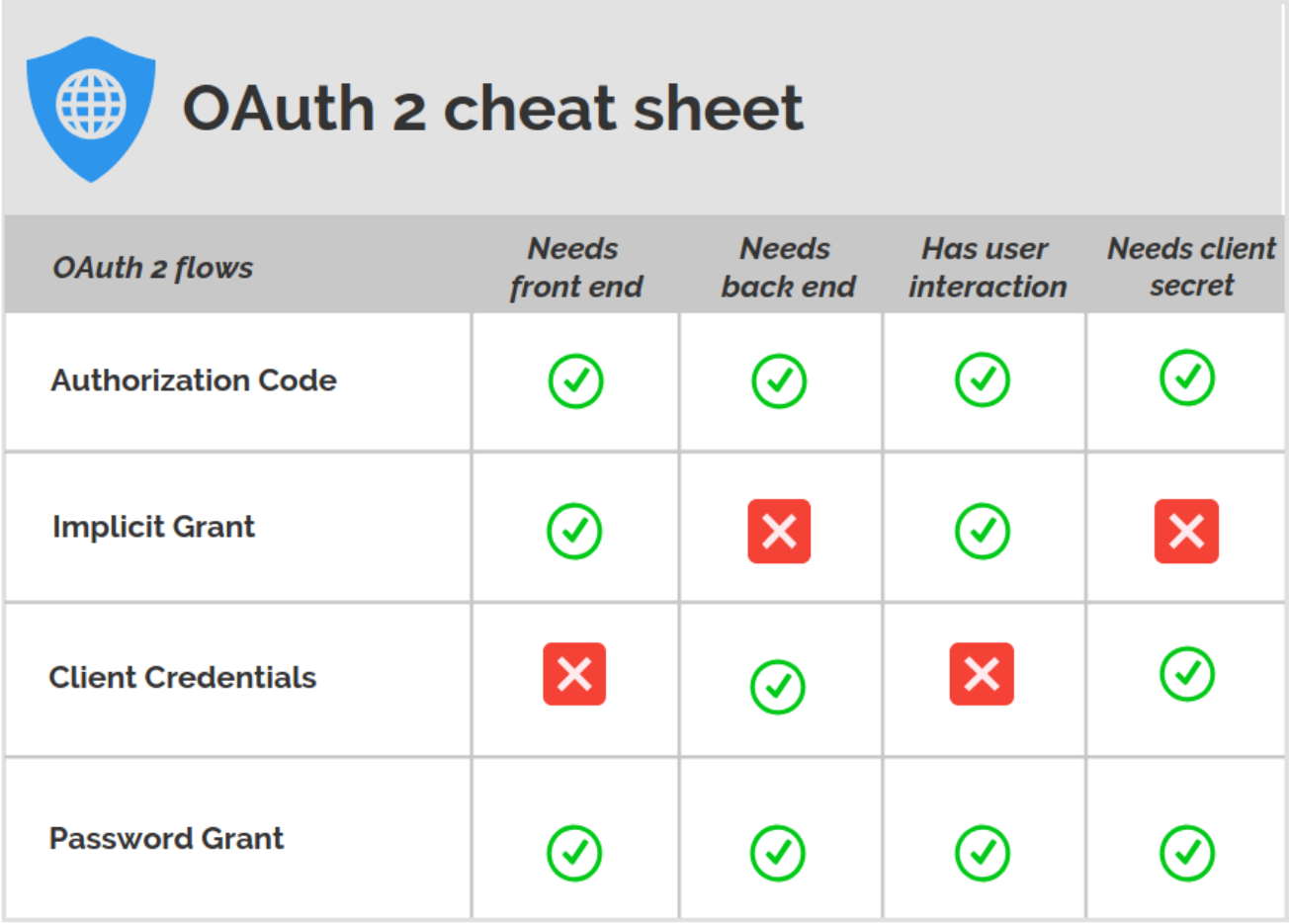

Рисунок 2.3 Вимоги для OAuth2 потоків.

Для клієнта вуличного радіо найкраще підходить потік Client Credential Grant, який не передбачає використання інтерфейсу користувача. Для інтерфейсу адміністратора системи — Authorization Code Grant (рисунок 2.2). Це найбезпечніший і складний потік. Процес входу розділений на два етапи, що забезпечує більшу безпеку.

Модель клієнт-сервер.

Модель клієнт-сервер (рисунок 2.3) — це розподілена структура програми, яка розподіляє завдання або робоче навантаження між постачальниками ресурсу

або служби, які називаються серверами, і запитувачами послуг, які називаються клієнтами. В архітектурі клієнт-сервер, коли клієнт-комп'ютер надсилає запит на дані серверу через Інтернет, сервер приймає запитуваний процес і доставляє запитані пакети даних назад клієнту.

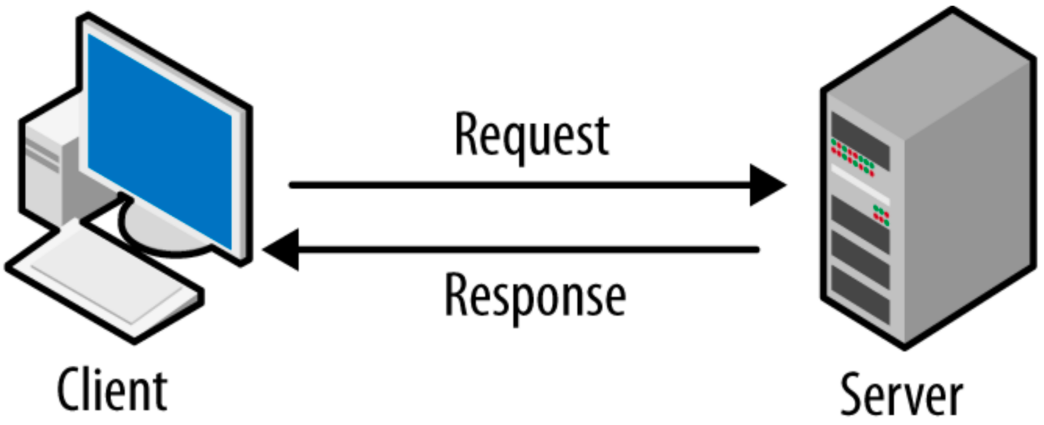

Рисунок 2.4 Модель клієнт-сервер

Клієнт: станція Raspberry PI яка відтворює звукові повідомлення згідно попередньо заданої черги або в режимі реального часу.

Сервер: віддалений Django сервер на якому формується черга відтворення та відбувається менеджмент станціями та адміністраторами вуличного радіо.

Отже, в основному клієнт щось запитує, а Сервер обслуговує це до тих пір, поки клієнт присутній в базі даних.

Вебсокети.

HTTP і WebSocket обидва є протоколами зв'язку, які використовуються в спілкуванні клієнт-сервер (рисунок 2.4)[34].

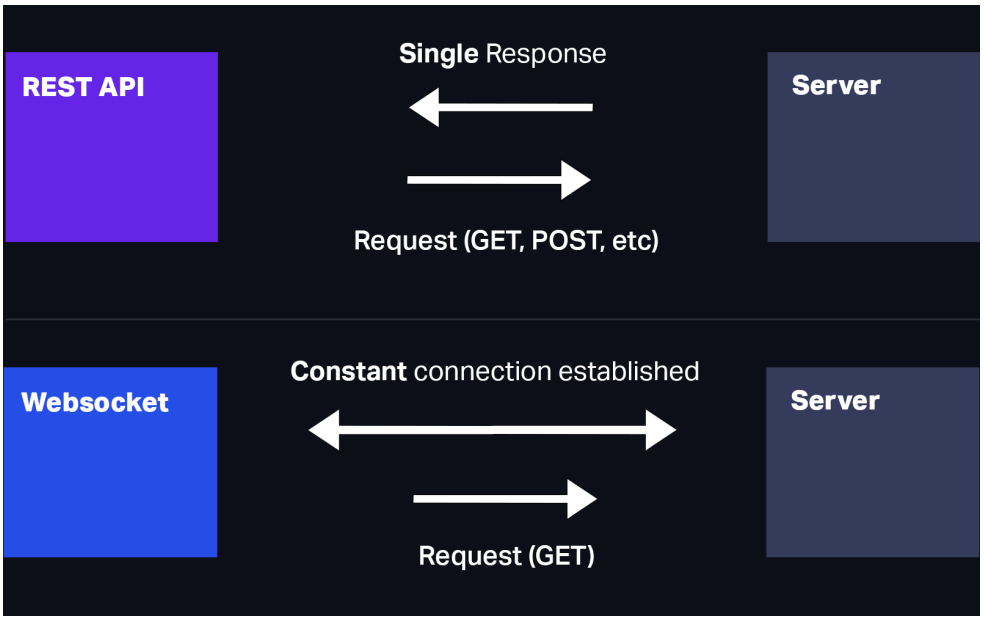

Рисунок 2.5 HTTP (REST API) та WebSocket

Протокол HTTP односпрямований, коли клієнт надсилає запит, а сервер — відповідь. Візьмемо приклад, коли користувач надсилає запит на сервер, цей запит йде у формі HTTP або HTTPS, після отримання запиту сервер надсилає відповідь клієнту, кожен запит асоціюється з відповідною відповіддю, після надсилання відповіді з'єднання закривається, кожен запит HTTP або HTTPS щоразу встановлює нове з'єднання з сервером, а після отримання відповіді з'єднання припиняється саме по собі.

HTTP — це протокол без стану, який виконується на верхній частині TCP, який є протоколом, орієнтованим на з'єднання, він гарантує доставлення пакетів даних за допомогою методів тристороннього рукостискання та повторну передачу втрачених пакетів.

HTTP може працювати поверх будь-якого надійного протоколу, орієнтованого на підключення, такого як TCP, SCTP. Коли клієнт надсилає HTTP-запит на сервер, TCP-з'єднання відкривається між клієнтом і сервером, і після отримання відповіді TCP-з'єднання припиняється, кожен HTTP-запит відкриває окреме TCP-з'єднання до сервера, наприклад, якщо клієнт надсилає 10 запитів — на сервері буде відкрито 10 окремих TCP-з'єднань. і закритися після отримання відповіді/резервного рішення.

Інформація про повідомлення HTTP, закодована в ASCII, кожне повідомлення запиту HTTP складається з версії протоколу HTTP (HTTP/1.1, HTTP/2), методів HTTP (GET/POST тощо), заголовків HTTP (тип вмісту, довжини вмісту), інформації про хост тощо. і тіло, яке містить фактичне повідомлення, яке передається на сервер. Розмір заголовків HTTP варіюється від 200 байтів до 2 КБ, звичайний розмір заголовка HTTP становить 700-800 байт. Коли вебдодаток використовує більше файлів cookie та інших інструментів на стороні клієнта, які розширюють можливості зберігання агента, це зменшує корисне навантаження заголовка HTTP[34].

WebSocket є двонапрямним, повнодуплексним протоколом, який використовується в тому ж сценарії спілкування клієнт-сервер, на відміну від HTTP, він починається з ws:// або wss://. Це протокол зі збереженням стану, що означає, що з'єднання між клієнтом і сервером буде існувати, поки його не розірве будь-яка сторона (клієнт або сервер). після закриття з'єднання клієнтом і сервером з'єднання припиняється з обох кінців.

Візьмемо для прикладу зв’язок клієнт-сервер, є клієнт, який є веббраузером і сервером, щоразу, коли ми ініціюємо з’єднання між клієнтом і сервером, клієнт-сервер здійснив рукостискання і вирішив створити нове з’єднання, і це з’єднання буде залишатися в живих до припинення будь-яким із них. Коли з’єднання встановлено та активне, зв’язок відбувається за допомогою того самого каналу з’єднання, поки воно не буде розірвано.

Ось як після зв’язку клієнт-сервер вирішує встановити нове з’єднання, щоб зберегти його живим, це нове з’єднання буде відоме як WebSocket. Після встановлення каналу зв'язку та відкриття з'єднання обмін повідомленнями буде відбуватися у двонапрямному режимі, доки з'єднання між клієнтом-сервером не буде зберігатися. Якщо хтось із них (клієнт-сервер) загине або вирішить закрити з’єднання, обидві сторони закривають. Спосіб роботи сокета дещо відрізняється від того, як працює HTTP, код статусу 101 позначає протокол перемикання в WebSocket.

2.3. Алгоритми ретрансляції звукових повідомлень в реальному часі для системи вуличного радіо

Технологія прямої трансляції часто використовується для трансляції подій у прямому ефірі, таких як спортивні змагання, концерти та взагалі теле- та радіопрограми, які виходять у прямому ефірі. Пряму трансляцію, яку часто скорочують до простої потокової передачі, — це процес передачі «живого» медіа на комп'ютери та пристрої. Це досить складний предмет, який зароджується з великою кількістю змінних.

Під час потокової передачі мультимедіа в браузер головне, що ми передаємо файл, який створюється на льоту і не має попередньо визначеного початку чи кінця, замість того, щоб відтворювати кінцевий файл.

У цьому випадку ми використовуємо статичний носій для опису медіа, який представлений файлом, будь то mp3 або файл WebM. Цей файл знаходиться на сервері та може бути доставлений — як і більшість інших файлів — у браузер. Це часто називають прогресивним завантаженням.

Медіа, що транслюються в прямому ефірі, не має обмеженого часу початку та завершення, оскільки це не статичний файл, а потік даних, який сервер передає далі в браузер і часто є адаптивним. Зазвичай для цього нам потрібні різні формати та спеціальне програмне забезпечення на стороні сервера.

Одним з основних пріоритетів для прямої трансляції є синхронізація програвача з потоком: адаптивне потокове передавання — це техніка для цього у разі низької пропускної здатності. Ідея полягає в тому, що швидкість передачі даних контролюється, і якщо здається, що вона не встигає, ми опускаємося до потоку з нижчою пропускною здатністю (і, отже, нижчою якістю). Щоб мати цю можливість, нам потрібно використовувати формати, які полегшують це. Формати прямої трансляції зазвичай дозволяють адаптивне потокове передавання,

розбиваючи потоки на серію невеликих сегментів і роблячи ці сегменти доступними з різною якістю та швидкістю передачі даних.

Потокове аудіо та відео на вимогу.

Потокова технологія не використовується виключно для прямих трансляцій. Його також можна використовувати замість традиційного методу прогресивного завантаження для аудіо та відео за запитом.

У цьому є кілька переваг:

- Затримка зазвичай нижча, тому медіафайли почнуть відтворюватися швидше.
- Адаптивне потокове передавання покращує роботу на різних пристроях.
- Медіафайли завантажуються вчасно, що робить використання пропускної здатності більш ефективним.

Саме потокова технологія найкраща для системи вуличного радіо. Вона не затрачає надто багато ресурсів та не потребує постійного ведення ефіру. Це дозволяє відтворювати звукові повідомлення згідно з чергою яку сформував адміністратор системи у відповідному інтерфейсі.

Протоколи потокового передавання.

Хоча статичні медіа зазвичай обслуговуються через HTTP, існує кілька протоколів для обслуговування адаптивних потоків; розгляньмо варіанти.

HTTP.

На цей час HTTP є найбільш поширеним протоколом, який використовується для передачі медіафайлів на вимогу або в режимі реального часу[34].

RTMP.

Протокол обміну повідомленнями в реальному часі (RTMP) — це власний протокол, розроблений Macromedia (тепер Adobe) і підтримується плагіном Adobe Flash. RTMP поставляється в різних варіантах, включаючи RTMPE (зашифрований), RTMPS (безпечний через SSL/TLS) і RTMPT (інкапсульований у HTTP-запитах).

RTSP.

Протокол потокової передачі в реальному часі (RTSP) контролює медіасеанси між кінцевими точками та часто використовується разом з протоколом реального часу транспортного протоколу (RTP) і протоколом керування реальним часом (RTCP) для доставляння медіапотоку. Використання RTP з RTCP дозволяє здійснювати адаптивну потокову передачу. Це поки що не підтримується в більшості браузерів. Деякі постачальники реалізують відповідні транспортні протоколи, такі як RealNetworks та їхній Real Data Transport (RDT).

Окрім постійної потокової трансляції на станції вуличного радіо також повинна бути присутня черга звукових повідомлень, які відтворюватимуться в той час, коли ефір не буде зайнятим.

Schedule є планувальником у процесі виконання періодичних завдань, які використовують шаблон конструктора для конфігурації. Schedule дозволяє періодично запускати функції Python (або будь-які інші викликані) через заздалегідь визначені інтервали, використовуючи простий, зручний для людини синтаксис.

Бібліотека розкладів використовується для планування завдання на певний час кожного дня або певного дня тижня. Ми також можемо встановити час у 24-годинному форматі, коли завдання має виконуватися. По суті, бібліотека розкладів узгоджує час вашої системи з запланованим часом, встановленим вами. Після того, як запланований час і системний час збігаються, викликається функція завдання (командна функція, яка запланована).

Відкладені завдання також можна використовувати для проведення регламентних завдань таких як: синхронізація черги звукових повідомлень, завантаження медіафайлів в той час, коли станція не працює, надсилання на сервер статистичної інформації. Такий підхід забезпечить меншу навантаженість на станцію коли вона працює.

2.4. Подальша інтеграція системи вуличного радіо з системою “розумного міста”

Для інтеграції системи вуличного радіо та системи “розумного” міста використовуватимуться API.

API є всюди. Працюючи у фоновому режимі програмного забезпечення, яке ми використовуємо для підключення до вебсторінки, пристроїв і даних у режимі реального часу, API невидимі для більшості людей. Попри таку невидимість, вони надзвичайно потужні та дозволяють даним, продуктам і процесам безперешкодно працювати разом. Їх можна використовувати практично для чого завгодно, від простих інструментів для вебмаркетингу до відстеження розташування супутників у космосі[44].

API також є критичними для міст. Міста виробляють великі обсяги даних із різних служб, пристроїв, програм та об’єктів. Робота з API дозволяє користувачам підключатися безпосередньо до потрібних даних без повторного пошуку у величезному обсязі, який створюється. API також можуть допомогти з’єднати дані та пристрої разом, щоб створити унікальну спільну інформаційну систему, на якій можна будувати послуги розумного міста.

На практиці API – це ансамбль обчислювальних функцій, за допомогою яких різні програмні продукти можуть взаємодіяти без участі людини. Це дозволяє програмному забезпеченню підключатися один до одного для досягнення конкретних цілей, наприклад, розкриття даних для громадськості або руйнування внутрішніх розрізів.

Загалом, API згладжують зв’язки між програмним забезпеченням, видаляючи людину посередині, що дає змогу використовувати дані організації в режимі реального часу для полегшення аналізу, візуалізації або прийняття рішень. Це дозволяє більшій кількості людей отримувати доступ до даних і використовувати їх з меншими зусиллями. Наприклад, підприємець може

створити нову програму, використовуючи дані міста, яка покладається на безперервне оновлення наборів даних без необхідності постійного оновлення з'єднання чи коду. Це звільняє підприємця працювати над покращенням своїх продуктів і послуг, не витрачаючи час на постійне оновлення даних.

Розумні міста залежать від цих плавних з'єднань між різними даними, пристроями та сервісами, щоб сприяти ширшому використанню даних. Оскільки більше даних підключається та використовується, міста можуть надавати кращі послуги людям у своїх громадах. API можуть допомогти створити більш цілісну спільну інформаційну систему для даних міста, полегшуючи автоматичне коригування на основі певних умов, сповіщення людей, які їх потребують, і кращі рішення для урядів у довгостроковій перспективі.

Як тільки концепція API згладжує з'єднання даних створена, найпростіше побачити їх потенційний вплив на прикладах із реального світу. Розглянемо три приклади того, як API забезпечують розумні міста та покращують послуги для людей. Використання API в контексті розумного міста зазвичай поділяється на три категорії залежно від того, хто використовує API: для внутрішнього використання, зовнішнього використання або гібрид обох.

Внутрішній: внесення змін у режимі реального часу та сповіщень на основі даних лічильників води.

Місто Каламазу, штат Мічиган, встановило «розумні» лічильники води для всіх споживачів своєї системи водопостачання. Кожен із лічильників оснащено програмним забезпеченням для вимірювання споживання води, виявлення будь-яких аномалій та передачі цієї інформації персоналу відділу водопостачання. Розумні лічильники знімають і передають показання споживання води кілька разів на день, замість того, щоб людина вручну перевіряла лічильник на наявність змін.

Оскільки нові лічильники регулярно передають велику кількість інформації, співробітники відділу водних ресурсів можуть використовувати API, щоб полегшити з'єднання між великими обсягами даних і передавати сповіщення в режимі реального часу, якщо виявлені проблеми. Крім того, клієнти можуть легше сповіщати про проблеми, підключившись до програми, яка відстежує

використання води (на основі API), зменшуючи їхні рахунки та кількість води, що витрачається в системі.

Зовнішній: використання API для стимулювання досліджень та безпеки спільноти.

У Чаттанузі, штат Теннессі, місто співпрацює з Chattanooga Smart Community Collaborative (CSCC), розташованим в Університеті Теннессі в Чаттанузі, щоб збирати та підключати дані з різних систем в одному місці. Завдяки таким даним, як безпека пішоходів, транспортний потік та якість повітря, пов'язані через API, місто може проводити симуляції з університетом на своєму «цифровому близнюку», щоб випробувати нові способи організації громадських просторів для підвищення громадської безпеки.

Завдяки партнерству з університетом місто змогло створити центральну платформу з різноманітними типами даних та джерелами. Потім API підключають будь-кого, хто хоче використовувати ці дані, до всього набору, що дозволяє зовнішнім користувачам здійснювати величезну кількість налаштувань і експериментів. Використання API таким чином захоплює ідеї всієї спільноти та дає змогу місту використовувати мозок найкращих і найяскравіших у Чаттанузі.

Гібридний: об'єднання державних даних і даних, створених користувачами, щоб покращити потік трафіку.

У Луїсвіллі, штат Кентуккі, місто брало участь у різноманітних проєктах будівництва доріг у центрі міста з інтенсивним рухом людей. Щоб допомогти проаналізувати вплив цих проєктів, вони співпрацювали з Waze, щоб отримати уявлення про дані про трафік, створені користувачами в режимі реального часу.

І місто, і користувачі Waze виграють від використання даних та API таким чином. Місто отримує додаткову інформацію від користувачів і економить час на перевірку транспортних потоків вручну. Користувачі Waze отримують додаткові дані від міста про перекриття доріг, а також переваги міста, яке може коригувати свої плани руху на основі проблем у реальному часі, щоб покращити транспортний потік.

Завдяки інтеграції системи вуличного радіо та системи “розумного” міста можливо буде, в реальному часі, сповіщати мешканців про будь-якого роду події. Отримувати статистику відтворення зі станцій, та інші масиви даних для їх подальшого опрацювання у BigData.

2.5. Висновки до розділу

Розроблено алгоритм ретрансляції звукових повідомлень в реальному часі для системи вуличного радіо. Основним завданням закладеним в алгоритм є адаптивне передавання медіа даних між передавачем та приймачем, оптимальне споживання енергії без втрати швидкості, адже чим вища смуга пропускання складніше налаштувати систему: положення на кут антени, інші пристрої навколо системи створюють перешкоди на шляху між передавачем та приймачем, усе це впливає на продуктивність.

Необхідно зазначити, що невіддільною частиною «розумного» міста є потужний інструмент API, який прогресивно вносить кардинальні зміни у розвиток міст, покращуючи програмне забезпечення. Ніхто не може бути впевнений, як формуються технології в майбутньому, але, безумовно, посилення співпраці призводить до посилення інновацій. Міжурядові організації, що підштовхують співпрацю між містами, посіяли насіння великих інновацій. Оскільки все більше міст, підприємств та громадян відкривають API, вони прокладуть шлях для розумніших міст завдяки інноваціям, що ведуть співпрацю.

Система вуличного радіо передбачає відтворення звукових повідомлень в режимі реального часу або згідно із чергою яку задав адміністратор. Основною метою цієї системи є попередити людину про те, що сталася конкретна подія. Адже надзвичайні ситуації можуть виникнути у будь-якому місці та в будь-який час і завдяки інтеграції з системою розумного міста за допомогою API, місто, в реальному часі, із легкістю сповістить мешканцям необхідну інформацію.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ВУЛИЧНОГО РАДІО ТА ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ АЛГОРИТМІВ

3.1. Структури системи вуличного радіо

Розроблено структуру програмної та апаратної реалізації системи вуличного радіо (рисунок 3.1) для системи розумного міста. Програма частина складається з наступних елементів: сервер станцій вуличного радіо, клієнт-станція вуличного радіо та інтерфейс адміністратора системи.

Програмне забезпечення станції вуличного радіо поділене на блоки: блок оновлення черги завдань (регламенті роботи) та синхронізації з сервером, блок відтворення звукових повідомлень згідно з чергою отриманою з сервера, блок відтворення повідомлень в режимі реального часу не залежно від черги повідомлень на сервері.

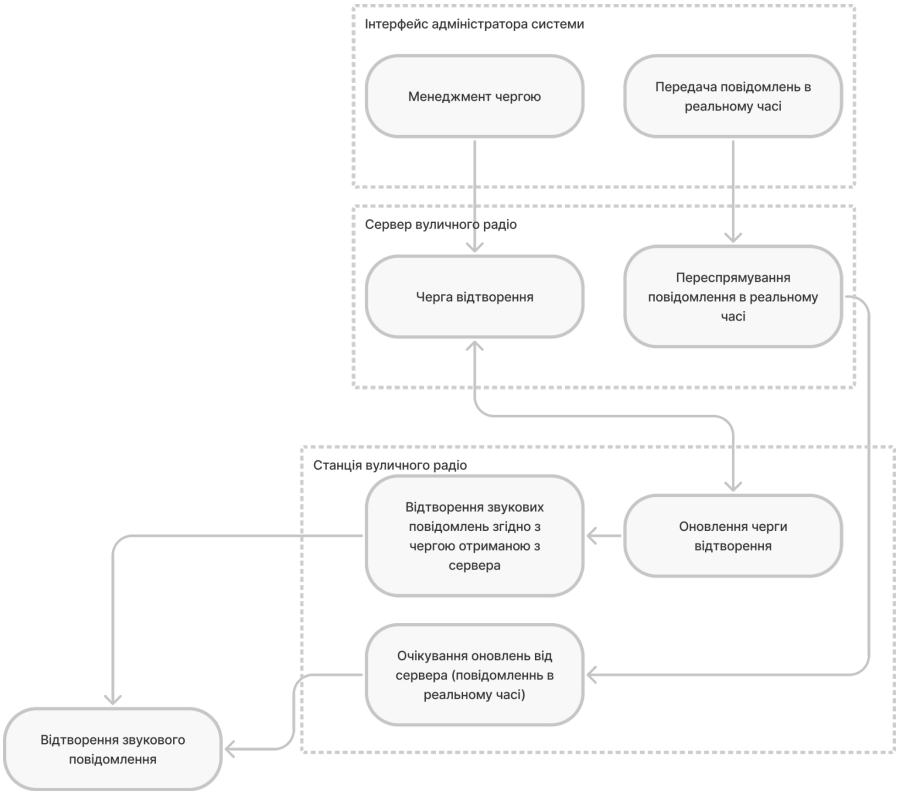

Рисунок 3.1 — Структура схеми роботи вуличного радіо

Для реалізації апаратної частини системи вуличного радіо використано: одноплатний комп'ютер Raspberry PI 4 Model B, всепогодний гучномовець зі ступенем захисту IP66, корпус для захисту одноплатного комп'ютера зі ступенем захисту IP66.

Raspberry Pi 4 Model B — це остання версія недорогого комп’ютера Raspberry Pi. Pi не схожий на ваш типовий пристрій; у найдешевшому вигляді він не має корпусу, а являє собою просто електронну плату розміром з кредитну картку — такого типу, який ви можете знайти всередині ПК або ноутбука, але набагато меншого розміру. Характеристики Raspberry PI 4 Model B описані в таблиці 3.1.

Таблиця 3.1 — Характеристики Raspberry PI 4 Model B

Компонент	**Характеристика**
Плата (system-on-a-chip)	Broadcom BCM2711
Центральний процесор	Quad-core 1.5GHz Arm Cortex-A72 based processor
Графічний процесор	VideoCore VI
Пам'ять	1/2/4GB LPDDR4 RAM
Підключення	802.11ac Wi-Fi / Bluetooth 5.0, Gigabit Ethernet
Відео та звук	2 порти micro-HDMI з підтримкою дисплеїв 4K@60Hz через HDMI 2.0, порт дисплея MIPI DSI, порт камери MIPI CSI, 4-полюсний стереовихід і композитний відеопорт
Порти	2 x USB 3.0, 2 x USB 2.0
Живлення	5 В/3 А через USB-C, 5 В через роз'єм GPIO
Можливість розширення	40-контактний роз'єм GPIO

Raspberry Pi 4 може зробити на диво багато. Для початку любителі технологій використовують плати Raspberry Pi як медіацентр, файлові сервери,

ретро ігрові консолі, маршрутизатори та блокувальники реклами на рівні мережі. Існують сотні проєктів, де люди використовували Raspberry Pi для створення планшетів, ноутбуків, телефонів, роботів, розумних дзеркал, фотографування на краю космосу, проведення експериментів на Міжнародній космічній станції.

Оскільки Raspberry Pi 4 є швидшим, здатним декодувати відео 4K, користується швидшим зберіганням через USB 3.0 і швидшим мережевим підключенням через справжній Gigabit Ethernet, відкриті можливості для багатьох нових застосувань. Це також перший Raspberry Pi, який підтримує два екрани на одному – до двох дисплеїв 4K@60 – це благо для креативних людей, які хочуть більше місця на робочому столі.

Існують різні способи відтворення аудіофайлів за допомогою Raspberry Pi. Найпростіший спосіб відтворювати аудіо на Raspberry Pi – за допомогою дротових динаміків або навушників. Їх можна під'єднати до Raspberry Pi за допомогою вбудованого гнізда для навушників.

Для відтворення аудіо за допомогою Python було вирішено використовувати mpg123. Дистрибутив mpg123 містить аудіоплеєр/декодер MPEG 1.0/2.0/2.5 у режимі реального часу для рівнів 1, 2 і 3 (найчастіше MPEG 1.0, рівень 3, або MP3), а також бібліотеки декодування та виведення, які можна повторно використовувати. Серед іншого, він працює на GNU/Linux, MacOSX, BSD, Solaris, AIX, HPUX, SGI Irix, OS/2 і Cygwin або на звичайній MS Windows (не всі більш екзотичні платформи тестуються регулярно, але виправлення вітаються). Для python реалізована однойменна бібліотека яка працює з mp123.

3.2 Сервер для менеджменту станціями системи вуличного радіо

Сервер менеджменту вуличного радіо реалізований мовою програмування Python з використанням бібліотек: Dajngo, djangorestframework (для реалізації REST API), drf-yasg (для створення документації до проєкту), django-oauth-toolkit

(для реалізації OAuth2). Реалізація сервера складається з 3 основних django-додатків:

- clients - додаток для керування клієнтами системи вуличного радіо;
- schedule - додаток для керування розкладом та зберігання повідомлень які були передані в режимі реального часу;
- stations - додаток для керування станціями системи вуличного радіо.

Для правильної роботи django сервера необхідно приділити увагу налаштуванням проєкту. Для зберігання секретних даних в налаштуваннях слід використовувати змінні середовища для того, щоб чутливі дані не змогли потрапити в git репозиторій.

На сервері системи вуличного використовується база даних PostgreSQL для зберігання даних про чергу відтворення та станції вуличного радіо. Також для оптимізації запитів до бази даних використовується PgBouncer - пулер підключень для PostgreSQL.

Візуальну структуру таблиць подано в додатку А. Розглянемо призначення таблиць в базі даних, яке описано в таблиці 3.2.

Таблиця 3.2 — Призначення таблиць в базі даних сервера вуличного радіо

Назва таблиці бази даних	**Призначення**
auth_user	Адміністратори сервера системи вуличного радіо
auth_group	Групи адміністраторів системи вуличного радіо
authtoken_token	Токени авторизації
django_admin_log	Інформація про події виконані в інтерфейсі адміністратора сервера
django_migrations	Міграції бази даних
django_session	Сесії адміністраторів
schedule_livemessages	Історія повідомлень записаних в реальному часі

schedule_radioschedule	Черга відтворення
stations_radiostation	Станції вуличного радіо
schedule_radioschedule_involved_stations	M2M зв’язок між чергою та станціями для відтворення
clients_radioclient	Клієнти системи вуличного радіо

На сервері вуличного радіо менеджмент бази даних відбувається через Django ORM, таблиці описуються моделями:

```
class RadioSchedule(models.Model):
    """
        Розклад відтворення для станцій вуличного радіо
    """
    title = models.CharField("Назва запису", max_length=512)
    time_from = models.DateTimeField(
        "Час початку",
    )
    time_to = models.DateTimeField(
        "Час закінчення",
    )
    play_rate = models.PositiveIntegerField(
        'К-сть повторів в одну годину'
    )
    file = models.FileField(
        "Файл для відтворення"
    )
    client = models.ForeignKey(
        RadioClient,
        verbose_name="Замовник",
        on_delete=models.SET_NULL,
        default=None,
        null=True, blank=True
    )
    involved_stations = models.ManyToManyField(
        RadioStation,
        related_name='play_queue',
        verbose_name="Залучені станції",
        blank=True
    )
    created_at = models.DateTimeField(
        "Створено",
        auto_now_add=True,
        editable=False
```

```
    )
    updated_at = models.DateTimeField(
        "Оновлено",
        auto_now=True,
        editable=False
    )
    def __str__(self):
        return self.title

    class Meta:
        verbose_name = "Запис для відтворення"
        verbose_name_plural = "Записи для відтворення"
```

Django ORM (об'єктно-реляційне зображення) є однією з найбільш потужних особливостей Django. Це дозволяє взаємодіяти з базою даних, використовуючи код Python, а не SQL.

Django дозволяє зручно керувати даними, які зберігаються в базі даних, за допомогою інтерфейсу адміністратора (рисунок 3.2). Для авторизації в інтерфейсі необхідно створити суперкористувача консольною утилітою manage.py:

```
(street-radio-service-16gojkmw-py3.10)ivanitskyi@MacBook-Air-Nazar
street_radio_service % python manage.py createsuperuser
    Ім'я користувача (leave blank to use 'ivanitskyi'): ivanitskyi
    Email адреса: n.ivanitskyi@wunu.edu.ua
    Password: ******
    Password (again): ******
    Superuser created successfully.
```

Адміністрування сервера вуличного радіо реалізує основні операції з даними: створення, видалення, редагування та перегляд. Також записуються всі дії адміністраторів з даними на сервері, що дозволяє переглядати та аналізувати дії адміністратора в системі.

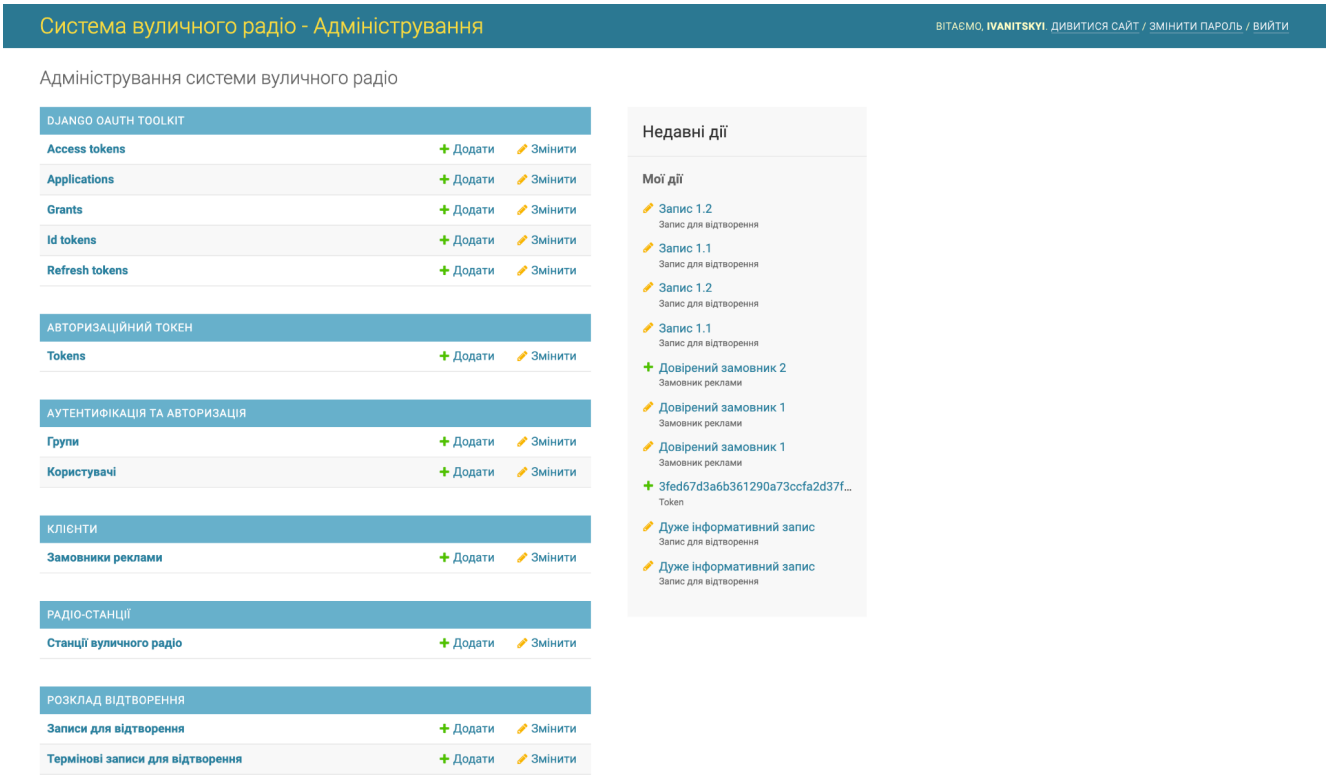

Рисунок 3.2 — Адміністрування даних на сервері системи вуличного радіо

Django Views є одним із найважливіших учасників структури MVT Django. Згідно з документацією Django, функція перегляду — це функція Python, яка приймає вебзапит і повертає вебвідповідь. Цією відповіддю може бути HTML-вміст вебсторінка, або перенаправлення, або помилка 404, або XML-документ, або зображення, будь-що, що може зобразити веббраузер.

Фреймворк Django REST framework дозволяє об'єднати логіку для набору пов'язаних уявлень в одному класі, який називається ViewSet. В інших фреймворках також можна знайти концептуально подібні реалізації під назвою «Ресурси» або «Контролери».

Маршрутизатори за замовчуванням, включені в структуру REST, забезпечать маршрути для стандартного набору дій стилю створення / отримання / оновлення / знищення. При створенні представлення за допомогою бібліотеки drf-yasg та налаштувань можливо створити документацію для нього (рисунок 3.3).

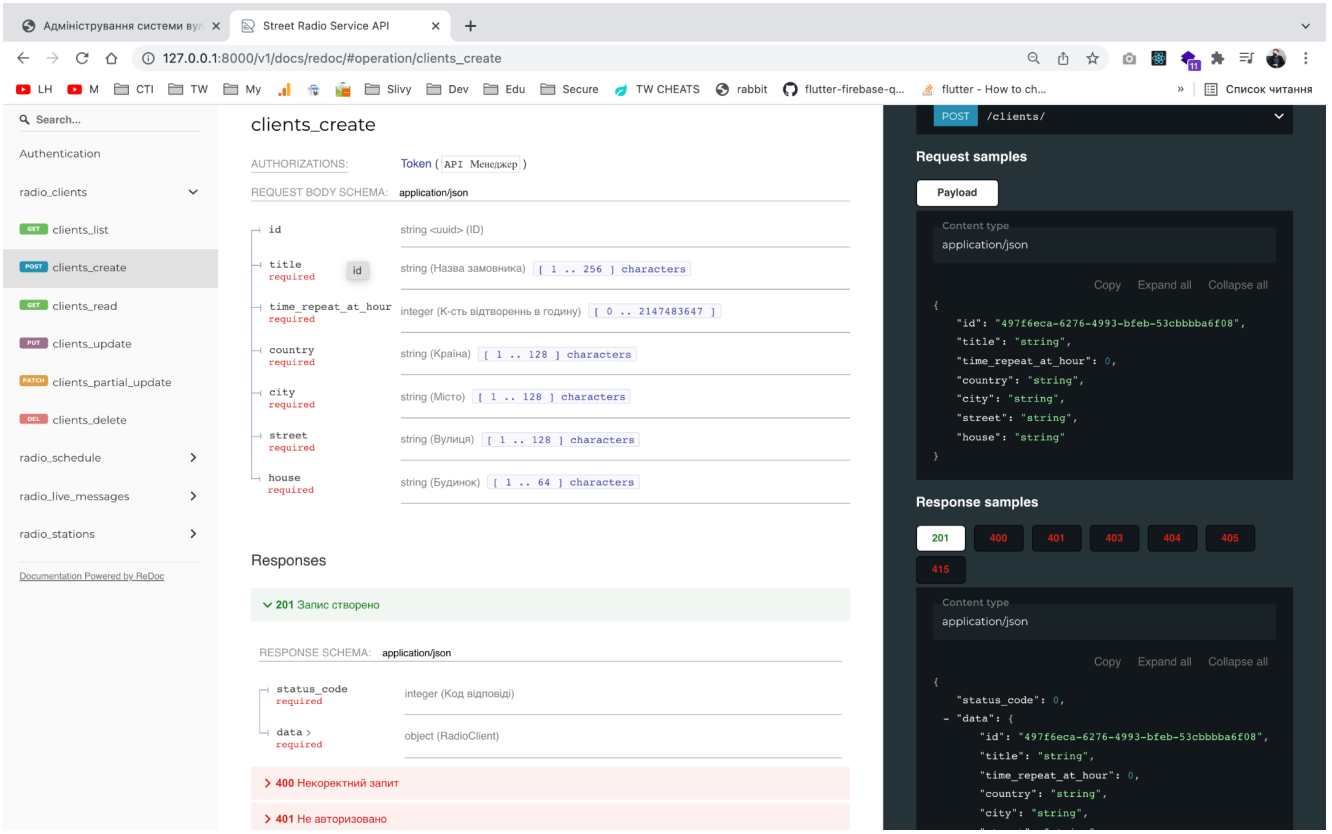

Рисунок 3.3 — Документація для API створення клієнта вуличного радіо.

Приклад ModelViewSet для клієнтів вуличного радіо:

```
from rest_framework import viewsets, permissions
from rest_framework.authentication import TokenAuthentication

from src.api.v1.clients.serializers.clients import
RadioClientSerializer
from src.apps.clients.models import RadioClient


class ClientsViewSet(viewsets.ModelViewSet):
    app_tags = ["radio_clients"]
    yasg_auth = ['token']
    authentication_classes = (TokenAuthentication, )
    permission_classes = [permissions.IsAuthenticated]
    serializer_class = RadioClientSerializer
    queryset = RadioClient.objects.all()
    lookup_field = 'id'

    class Meta:
        model = RadioClient
```

Це представлення для клієнтів вуличного радіо дозволяє проводити з ним маніпуляції за допомогою REST API, після його реєстрації в urls:

```
clients_router = DefaultRouter()
clients_router.register("", views.ClientsViewSet,
basename='ClientsView')
```

Для перетворення даних в Json та з Json в типи даних python Django використовує серіалізатори. Також за їх допомогою сервер може перевіряти всі дані які приходять йому від клієнта. Приклад серіалізатора для черги вуличного радіо:

```
from rest_framework import serializers
from src.apps.schedule.models import RadioSchedule


class RadioScheduleSerializer(serializers.ModelSerializer):

    class Meta:
        model = RadioSchedule
        lookup_field = 'id'
        exclude = ('created_at', 'updated_at')
        extra_kwargs = {
            'url': {'lookup_field': 'id'}
        }
```

Django автоматично включає всі поля моделі в серіалізатор і створює методи create та update.

В Django вебсокети реалізовані за допомогою Channels. Channels – це проєкт, який використовує Django та розширює його можливості за межі HTTP – для обробки WebSockets, протоколів чату, IoT-протоколів та багато іншого. Він побудований на специфікації Python під назвою ASGI.

Він робить це, беручи ядро Django та розміщуючи під ним повністю асинхронний шар, запускаючи сам Django у синхронному режимі, але обробляючи з'єднання та сокети асинхронно (рисунок 3.4), і даючи можливість писати у будь-якому стилі.

Канали (Channels) надають вам Споживачів (Consumers), багату абстракцію, яка дозволяє легко створювати програми ASGI. Споживачі, зокрема, роблять кілька речей: структурують ваш код як серію функцій, які будуть викликатися

щоразу, коли відбувається подія, замість того, щоб змушувати вас писати цикл події.

Споживач — це клас, методи якого можна написати як звичайні функції Python (синхронні) або як очікувані (асинхронні). Асинхронний код не повинен змішуватись із синхронним кодом. Таким чином, є функції перетворення для перетворення з асинхронного в синхронний та назад.

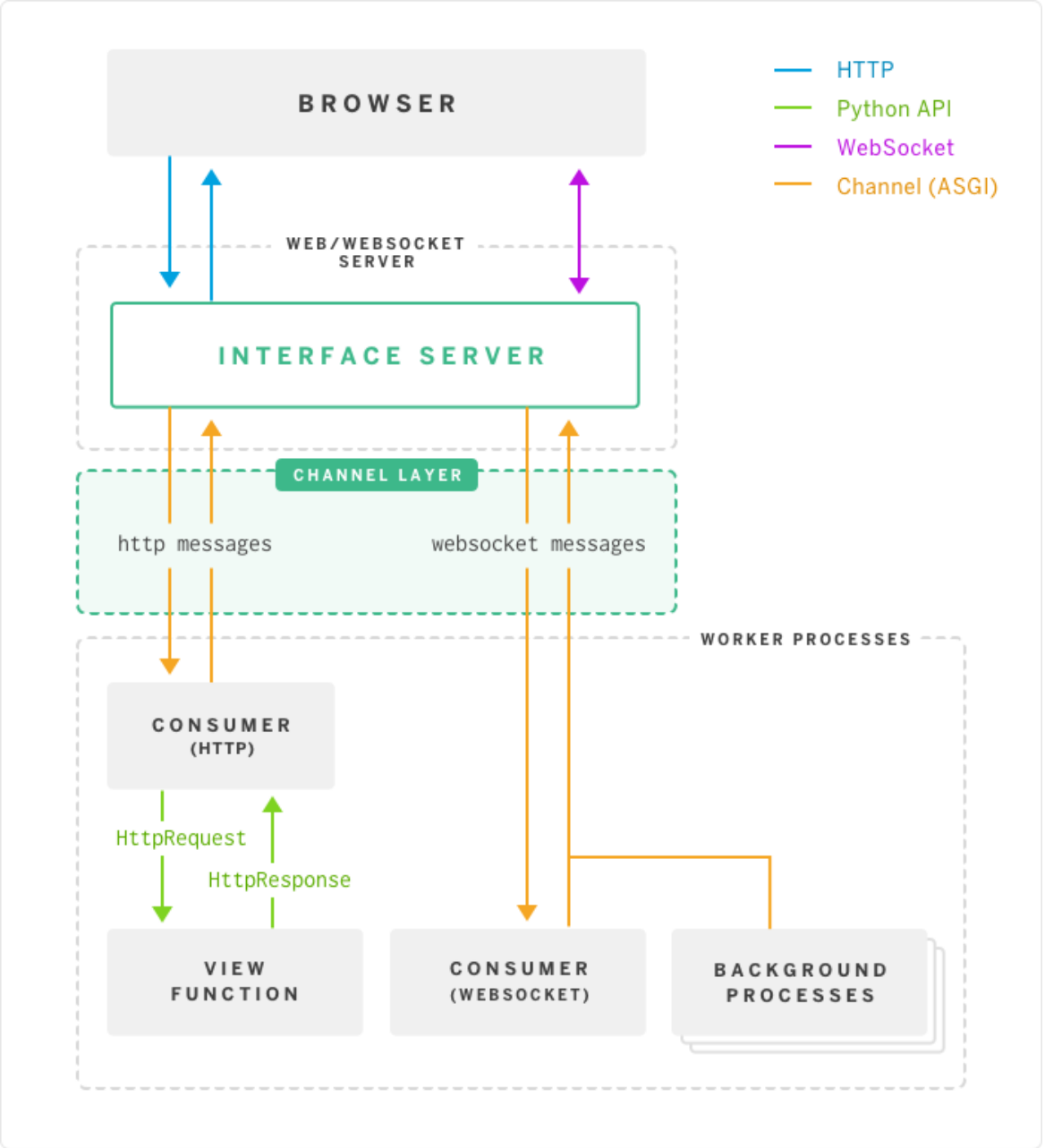

Рисунок 3.4 — Схема роботи Django з Channels

Запуск сервера відбувається через Docker-контейнер. Docker дає можливість запускати ваші програми в контрольованому середовищі, відомому як контейнер, створеному відповідно до визначених інструкцій. Контейнер використовує ресурси машини, так само як традиційна віртуальна машина (VM). Однак контейнери значно відрізняються від традиційних віртуальних машин з точки зору системних ресурсів. Традиційні віртуальні машини працюють за допомогою гіпервізорів, які керують віртуалізацією базового обладнання для віртуальної машини. Це означає, що вони великі з точки зору системних вимог. Переваги Docker:

- Docker не вимагає тривалого процесу встановлення всієї ОС на віртуальну машину, таку як VirtualBox або VMWare.
- Ви створюєте контейнер з кількома командами, а потім виконуєте в ньому свої програми за допомогою Dockerfile.
- Docker керує більшістю віртуалізації операційної системи за вас, тому ви можете продовжувати писати програми та відправляти їх у створеному вами контейнері, як вам потрібно.
- Файли Docker можна надавати іншим користувачам, щоб створювати контейнери та розширювати інструкції в них, створюючи зображення контейнера поверх наявного.
- Контейнери також дуже портативні та працюватимуть однаково незалежно від хост-ОС, на якій вони виконуються. Портативність є величезною перевагою Docker.

Приклад Dockerfile для Django-сервера:

```
FROM python:3.10-slim
…
ENV DEBUG="False" \
    SECURE_BROWSER_XSS_FILTER="True" \
    SECRET_KEY=$django_secret_key \
    API_TOKEN=$api_token \
    DJANGO_ENVIRONMENT="production" \
    CELERY_BROKER_URL=$celery_broker_url

ENV DEFAULT_DB_NAME=$db_name \
```

```
    DEFAULT_DB_SCHEME=$db_scheme \
    DEFAULT_DB_USER=$db_user \
    DEFAULT_DB_PASSWORD=$db_password \
    DEFAULT_DB_HOST=$db_host \
    DEFAULT_DB_PORT=$db_port

ENV MINIO_ENDPOINT_URL=$minio_endpoint \
    MINIO_ACCESS_KEY=$minio_access_key \
    MINIO_SECRET_KEY=$minio_secret_key \
    MINIO_BUCKET_NAME=$minio_bucket_name \
    MINIO_USE_SSL="True"

# Section 3- Compiler and OS libraries
RUN apt-get update \
    && apt-get install -y --no-install-recommends build-essential
libpq-dev libssl-dev libffi-dev python-dev \
    && rm -rf /var/lib/apt/lists/*

# Section 4- Project libraries and User Creation
RUN mkdir /usr/src/radioService
WORKDIR /usr/src/radioService

COPY pyproject.toml poetry.lock /usr/src/radioService/

RUN python -m pip install --upgrade pip \
    && pip install poetry

RUN echo "DJANGO_ENVIRONMENT: $DJANGO_ENVIRONMENT" \
    && poetry config virtualenvs.create false \
    && poetry install --no-dev --no-interaction --no-ansi \
    # Cleaning poetry installation's cache for production:
    && rm -rf "$POETRY_CACHE_DIR" \
    && useradd -U app_user


# Section 5- Code and User Setup
USER app_user:app_user
COPY --chown=app_user:app_user . .
RUN ["chmod", "+x", "./scripts/docker-entrypoint.sh"]
RUN ["chmod", "+x", "./scripts/docker-start.sh"]

EXPOSE 8002

# Section 6- Docker Run Checks and Configurations
ENTRYPOINT [ "./scripts/docker-entrypoint.sh" ]
CMD [ "./scripts/docker-start.sh", "server" ]
```

Для збірки зображення та запуску Docker контейнера, на основі зображення, слід виконати команди:

```
docker build -t radio_service:latest --build-arg … .
docker run -d --net=host --rm --name radio_service radio_service:latest
```

Після запуску команд збірки та створення нового контейнера сервіс буде працювати у свому середовищі та завдяки налаштуванню `--net=host` порти будуть доступними зовні для балансувальника навантаження nginx.

3.3 Клієнт-станція системи вуличного радіо

Станція вуличного радіо реалізована мовою програмування Python як клієнт сервера цієї системи. Відтворення звукових повідомлень відбувається згідно з чергою, яку попередньо склав адміністратор в інтерфейсі менеджменту системою. Також повідомлення можуть відтворюватись не залежно від поточної черги, а саме в режимі реального часу.

Клієнт поділений на два блоки: блок запланованих завдань та блок постійного websocket з'єднання з сервером.

Блок запланованих завдань призначений для відтворення аудіо згідно з чергою та регулярної синхронізації станції вуличного радіо з сервером, для забезпечення актуальності інформації.

Під час синхронізації станція вуличного радіо повинна зберігати дані у своїй локальній базі даних. Для цього вирішено використати базу даних SQLite та SQLAchemy - Python фреймворк для роботи з реляційними базами даних з ORM для зручної роботи з моделями даних. Коли станція проводить синхронізацію з сервером вона отримує повний перелік актуальних записів для відтворення саме на тій станції яка робить запит до сервера та зберігає дані про чергу в локальній базі даних (рисунок 3.5) для оптимізації доставлення повідомлень.

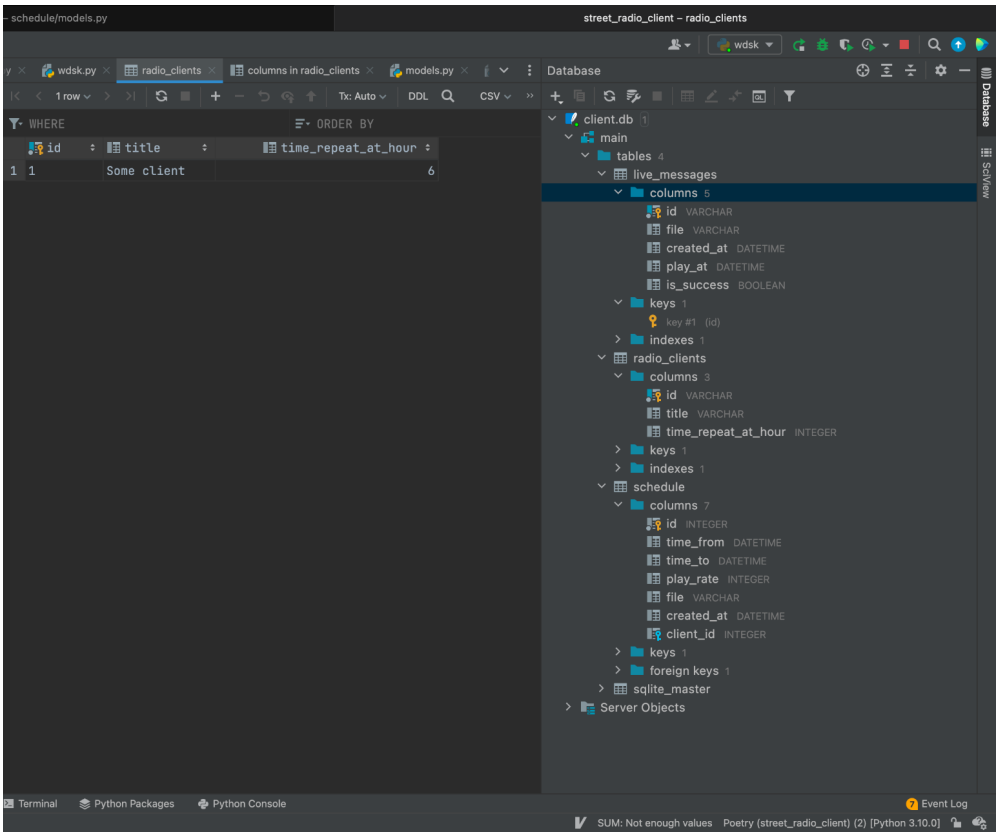

Рисунок 3.5 — Структура таблиць бази даних станції вуличного радіо

Призначення таблиць в базі даних клієнта вуличного радіо:

- live_messages: зберігання повідомлень записані в реальному часі;
- radio_clients: клієнти системи вуличного радіо;
- schedule: розклад відтворення запланованих повідомлень.

Блок websocket з'єднання з сервером забезпечує спілкування з сервером та відтворення звукових повідомлень в режимі реального часу. Під час запуску програмне забезпечення встановлює websocket з'єднання з менеджмент-сервером, сповіщає сервер про підключення, та отримує список подій інших клієнтів. Запуск websocket з'єднання реалізовується таким кодом:

```
if __name__ == "__main__":
    init_db()  # Ініціалізація таблиць бази даних

    websocket.enableTrace(True)

    ws = websocket.WebSocketApp(
        "ws://server-domain/ws/radio/",
        on_open=on_open,
        on_message=on_message,
        on_error=on_error,
        on_close=on_close,
        header=[
            f"Authorization: Token {settings.API_KEY}"
        ]
    )

    ws.run_forever()
```

З прикладу зрозуміло що для забезпечення нормальної роботи websocket з'єднання необхідно створити callback-функції для різних типів подій. В цих функціях і опрацьовуються повідомлення від сервера. Розглянемо callback-функцію опрацювання повідомлень:

```
def on_message(ws, message):
    logging.debug(f"Message received! Message: {message}")
    logging.debug(f"ws: {vars(ws)}")
    try:
        message_data = json.loads(message)
        if 'action' in message_data:
            action = message_data['action']
            if action == 'live':
                new_live_file = download_file(
                    url=message_data['live_file'],
                    filename=message_data['filename'],
                    type='live'
                )
                if new_live_file:
                    play_live(new_live_file)
                    ws.send(
                        json.dumps({
                            "message": 'live_success',
                            "status": True,
                            "station": settings.STATION_ID,
                            "live": datetime.datetime.now()
                        })
                    )
                else:
                    ws.send(
                        json.dumps({
                            "message": 'live_fail',
```

```
                        "status": False,
                        "station": settings.STATION_ID,
                        "live": datetime.datetime.now()
                    })
                )

            elif action == 'healthcheck':
                …
        else:
            print(message_data)

    except Exception as e:
        logging.debug(f"Message error! Message: {message}")
        logging.error(e)
```

Сервер відправляє клієнтам повідомлення після чого клієнт переходить до опрацювання, визначає тип повідомлення та дію яку він повинен виконати. Коли клієнт отримав дію відтворити повідомлення в реальному часі — він зберігає повідомлення в локальній базі даних, завантажує файл повідомлення та розпочинає відтворення файлу. Повідомлення яке потрібно відтворити в режимі реально часу програється незалежно від поточної черги відтворення.

Для авторизації станції на менеджмент сервері вуличного радіо обрано OAuth2 потік Client Credentials (рисунок 3.6).

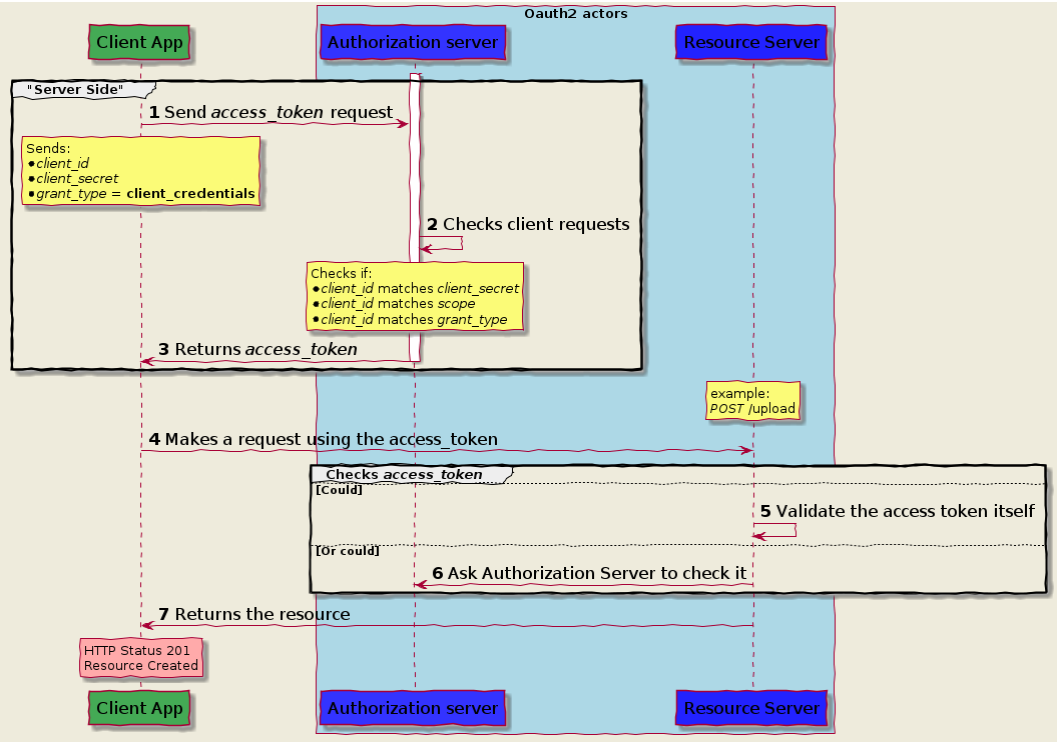

Рисунок 3.6 — Деталізація потоку Client Credentials

Основна відмінність потоку Client Credentials полягає в тому, що в ньому немає взаємодії з користувачем. Програма виконує запит на аутентифікацію безпосередньо до сервера.

Це означає, що програма не може виконувати жодних дій від імені користувача, а лише від себе.

Цей потік відбувається не в браузері, а на сервері. Перенаправлення немає, але виконується виклик HTTP POST, параметри запитів:

- grant_type: client_credentials

```
HTTP POST
    Authorization: Basic *********
    Cache-Control: no-cache
    Content-Type: application/x-www-form-urlencoded
    http://authorization-server/o/token/
    grant_type=client_credentials
```

Відповіддю на цей запит є access_token, який можна використати на сервері менеджменту станціями вуличного радіо для доступу до черги звукових повідомлень та іншого функціоналу.

Для отримання змоги авторизувати себе на сервері вуличного радіо станції необхідно отримати параметри client_id та client_secret, щоб отримати їх потрібно створити новий OAuth 2 додаток на сервері(рисунок 3.7).

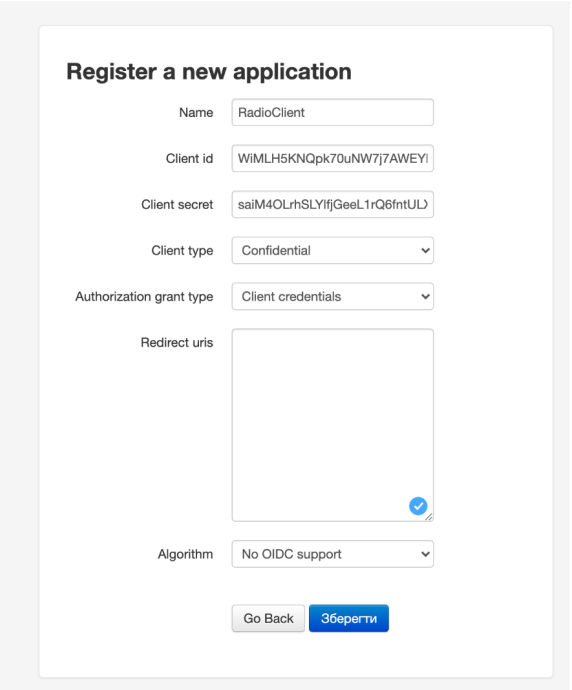

Рисунок 3.7 — Реєстрація нового OAuth2 клієнта з потоком client credentials

Після отримання параметрів клієнт зможе отримувати токен для доступу до сервера для синхронізації та встановлення з'єднання.

3.4 Інтерфейс адміністратора системи вуличного радіо

Інтерфейс адміністратора системи вуличного радіо реалізований з використанням фреймворку Next.js та утилітарного CSS-фреймворку Tailwindcss. Зв'язок між сервером та інтерфейсом адміністратора відбувається за допомогою REST API. Авторизація користувача реалізовується завдяки технології OAuth2, та облікового запису на сервері вуличного радіо з відповідним рівнем доступу.

Next.js має багато стратегій отримання даних (рисунок 3.8). Хоча спочатку Next.js був добре відомий як фреймворк показу на стороні сервера, виявляється, що Next.js має 4 методи отримання даних. Ось коротке пояснення кожного, для ознайомлення з абревіатурою CSR, SSR, SSG, ISR:

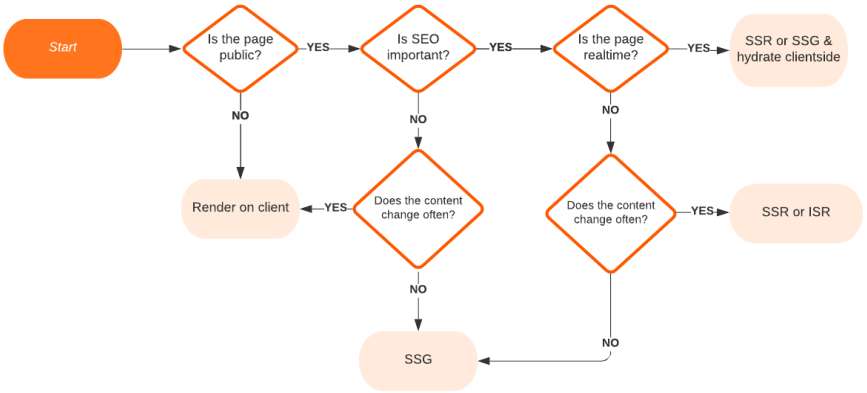

Рисунок 3.8 — Схема роботи Next.JS

- CSR – рендеринг на стороні клієнта, це звичайний вид отримання даних за допомогою useEffect, він отримуватиме дані з API кожного окремого запиту сторінки на стороні клієнта (після того, як сторінка буде відтворена, функція буде запущена).
- SSR – Server-Side Rendering, запускатиме спеціальну функцію для отримання даних з API кожного запиту сторінки на стороні сервера (перед завантаженням сторінки спочатку буде запущена спеціальна функція, створюючи затримку, а потім вона обслуговуватиметься сторінка).
- SSG – Static Site Generation, запускає спеціальну функцію для отримання даних один раз під час створення цієї сторінки.
- ISR – додаткова статична регенерація, це нова річ, коротко кажучи, комбінація SSG і SSR, де вона обслуговується статично, але в певний час і за певних умов ця сторінка перебудує та знову отримує дані з API.

В інтерфейсі адміністратора системи вуличного радіо реалізовано функціонал: авторизації в системі, перегляду статусу станцій, операцій з клієнтами, операцій з чергою повідомлень, запису та передачі повідомлень в реальному часі до станцій вуличного радіо (рисунок 3.9).

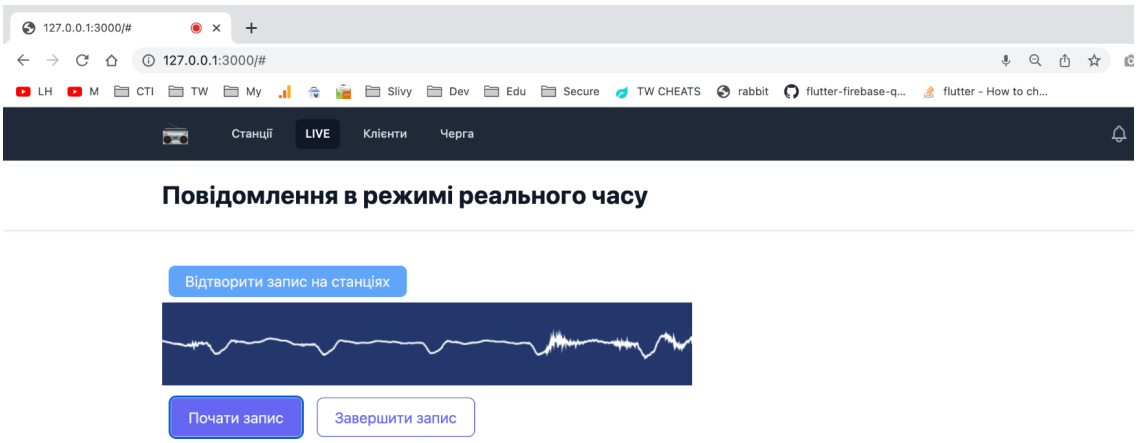

Рисунок 3.9 — Передача повідомлень в режимі реального часу через інтерфейс адміністратора

Для авторизації станції на менеджмент сервері вуличного радіо обрано OAuth2 потік Authorization Code Grant (рисунок 3.10).

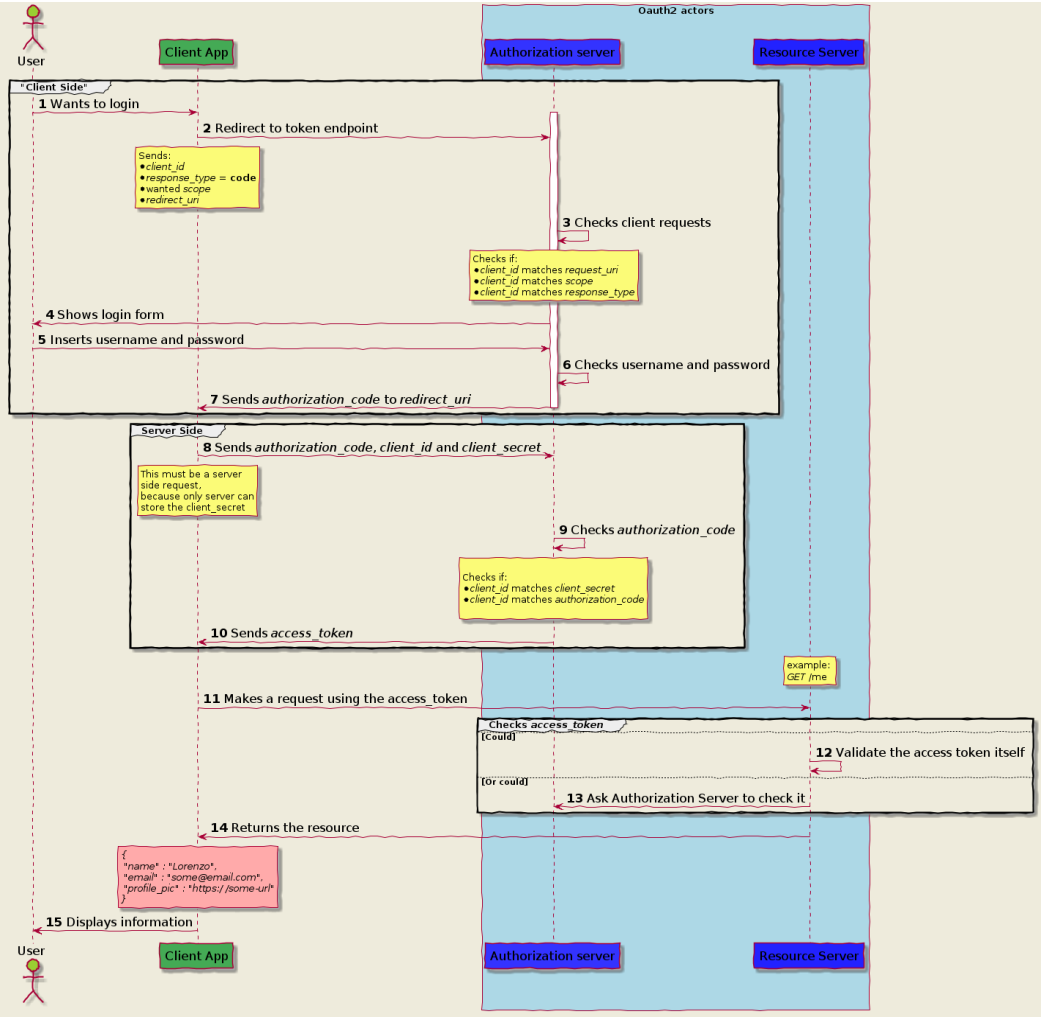

Рисунок 3.10 — Деталізація потоку Authorization Code Grant

Детально розглянемо потік Authorization Code Grant.

1. Користувач бажає увійти в інтерфейс адміністратора системи вуличного радіо.

Класичний сценарій для цього потоку відтворюється в браузері користувача. Користувач натисне кнопку «Увійти за допомогою OAuth», а клієнт згенерує та надішле запит на вхід на сервер авторизації (менеджмент-сервер вуличного радіо).

2. Користувача буде перенаправлено на Сервер авторизації.

Клієнт створює запит на вхід для сервера авторизації. Запит буде надіслано у формі HTTP Redirect, а інформація буде надіслана як GET параметри.

```
GET /token

Location:
https://authorization-server/token?client_id=[the_client_id]&redirect_uri=[a redirect uri]&response_type=code&scope=[list of scopes]&state=[some client parameter]
```

Параметри наступні:

- client_id: для ідентифікації програми, що викликає;
- redirect_uri: URL-адреса, на яку Сервер авторизації надішле (через перенаправлення) код авторизації після входу користувача;
- response_type: визначає тип відповіді, яку поверне Сервер авторизації. Це значення code, як правило, у потоці коду авторизації;
- scope: список дозволів, які програма запитує у користувача. Наприклад: view_schedule, write_log. Користувачу буде запропоновано надати ці дозволи. Це буде корисно, коли клієнт отримуватиме доступ до сервера ресурсів. Він вирішить, чи дозволити або заборонити доступ;
- state: цей необов'язковий параметр буде повернено клієнту як є після процесу входу. Його можна використовувати для отримання інформації про клієнтську програму, наприклад, про сеанс користувача.

3. Запит на підтвердження.

Сервер авторизації повинен перевірити всі параметри запиту:

- client_id: чи існує клієнт із цим ідентифікатором? Чи дозволено клієнту виконати цей запит?
- redirect_uri: чи може клієнт використовувати цей URI переспрямування? Чи пов’язаний цей URI переспрямування з цим клієнтом?
- response_type: чи дозволено клієнту використовувати цей тип відповіді?
- scope: чи дозволено клієнту використовувати ці гранти?

Щоб виконати ці перевірки, сервер авторизації повинен попередньо зареєструвати всіх клієнтів, які матимуть доступ. Підключення та обслуговування клієнтів виходять за рамки OAuth.

4. Форма входу.

Сервер авторизації показує форму входу, і адміністратор повинен ввести ім’я користувача та пароль, щоб здійснити вхід (крок 5 на рисунку 3.).

Після перевірки даних (крок 6 на рисунку 3.) сервер авторизації запитує у користувача згоду, зазначену в областях. Користувач вирішує надати чи ні одну чи декілька областей доступу.

7. Перенаправлення на клієнтську сторону з кодом авторизації сервер авторизації генерує код авторизації (authorization_code) і надсилає його клієнту за URI, зазначеним у запиті. Усі ці операції відбувалися на стороні клієнта, у браузері (або в мобільному додатку), для наступних кроків клієнт повинен використовувати серверну частину.

8 і 9. Перевірка коду авторизації.

Тепер клієнт повинен викликати сервер авторизації, щоб перевірити отриманий код. Для виконання цієї операції пройде перевірки:

- authorization_code, який потрібно перевірити;
- client_id: це потрібно разом із client_secret, щоб переконатися, що запит надходить від цього клієнта, і маркер не «викрадений», коли ви були на інтерфейсі;
- client_secret — це пароль клієнта, і він повинен зберігатися в безпечному місці, тому вам потрібен сервер.

Сервер авторизації виконує всі перераховані вище перевірки, перевіряючи, чи authorization_code недоторканий, не змінений, не прострочений і виданий для цього конкретного клієнта.

10. Клієнт отримує маркер доступу сервер авторизації створює access_token і повертає його клієнту. Існує багато типів токенів, вони мають термін дії і їх можна оновити. Зазвичай разом з маркером повертається додаткова інформація: token_type: одним із найвідоміших є Bearer, що означає: надати доступ носія цього маркера; expires_in: тривалість токена; refresh_token: інший маркер, щоб поновити access_token, коли термін його дії закінчиться.

11. Клієнт використовує access_token. Тепер, коли клієнт отримав access_token, він може використовувати його для автентифікації на сервері ресурсів. Сервер ресурсів є загальним компонентом і може обслуговувати багато різних типів ресурсів: REST API, послуги SOAP, вебсторінки тощо. Залежно від типу ресурсу, метод надсилання access_token може змінюватися. Найпоширеніший спосіб надіслати його проти REST API – це використовувати заголовок HTTP: Authorization, об’єднавши тип token_type з access_token. Приклад:

```
Authorization: Bearer eyJhbGciOlJUZjI2NiIhInR9cCL8IkpKCMJ
```

12 і 13. Перевірка токена

Коли сервер ресурсів отримує запит, він повинен перевірити наявність і цілісність маркера.

Перевірка токена, особливо на основі його типу, може бути виконана багатьма способами. У деяких випадках сервер ресурсів може перевірити його, в деяких інших повинен викликати сервер авторизації.

14 і 15. Доступ до захищених ресурсів.

Як тільки сервер ресурсів успішно перевірить маркер, він поверне ресурс клієнту, і він може показати його користувачеві.

Реєстрація клієнта, для можливості авторизації через обліковий запис адміністратора на сервері вуличного радіо (рисунок 3.11), здійснюється на

менеджмент-сервері за допомогою бібліотеки django-oatuh-toolkit. При створенні клієнта потрібно обрати Authorization code потік, вказати його назву та ввести посилання яке опрацьовуватиме запити від сервера авторизації на стороні клієнта.

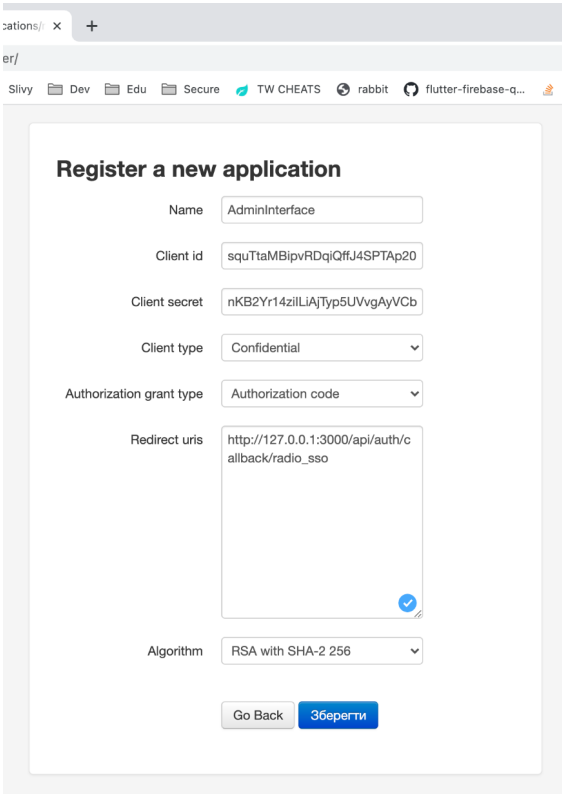

Рисунок 3.11 — Реєстрація нового OAuth2 клієнта з потоком Authorization code grant

Після реєстрації клієнта на сервері необхідно налагодити процес авторизації адміністраторів в системі. Для цього найкраще підходить бібліотека авторизації NextAuth, вона найкраще підходить для Next.js додатків. Після додавання бібліотеки до проєкту її необхідно налаштувати, для цього створимо файл конфігурації та додамо сервер вуличного радіо як провайдер авторизації. Приклад конфігурації провайдера авторизації:

```
{
  id: "radio_sso",
  name: "Radio Server Auth",
  type: "oauth",
  version: "2.0",
  params: { grant_type: "authorization_code" },
  accessTokenUrl: `${process.env.baseApiPath}/o/token/`,
  requestTokenUrl: `${process.env.baseApiPath}/o/authorize/`,
```

```
                                                        authorizationUrl:
`${process.env.baseApiPath}/o/authorize/?response_type=code`,
  profileUrl: `${process.env.baseApiPath}/o/userinfo/`,
  authorization: { params: { scope: "openid profile admin_action" } },
  idToken: true,
  checks: ["pkce", "state"],
  profile(profile) {
    return {
      id: profile.sub,
      name: profile.name,
      email: profile.email,
      image: profile.picture,
    }
  },
}
```

Після налаштувань інших частин бібліотеки адміністратори зможуть авторизуватись в інтерфейсі адміністратора за допомогою облікового запису на сервері вуличного радіо.

3.5. Висновки до розділу

За допомогою розроблених алгоритмів було реалізовано систему вуличного радіо для системи “розумного” міста. Основною метою якого було забезпечення можливості відтворення звукових повідомлень в режимі реального часу.

Відтворення звукових повідомлень здійснюється за допомогою одноплатного комп'ютера Raspberri PI 4, менеджмент-сервера системи вуличного радіо та інтерфейсу адміністратора системи.

Застосування алгоритмів дозволило реалізувати систему та надалі інтегрувати систему вуличного радіо з системою “розумного” міста, для якісного та вчасного інформування громадян про події в місті.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи одержані наступні результати:

1. Проведено аналіз концептів та стандартів інтернету речей, наявних систем вуличного радіо та “розумного” міста.

2. Проаналізовано наступні алгоритми: ретрансляції повідомлень в режимі реального часу, реалізації системи вуличного радіо для «розумного» міста.

3. Сформульовано вимоги до функцій системи вуличного радіо.

4. Реалізовано систему вуличного радіо для розумного міста з достатнім ступенем захисту від пилу і води.

5. Програмно реалізовано станцію вуличного радіо для відтворення звукових повідомлень мовою програмування Python. Менеджмент-сервер системи вуличного радіо для керування чергою відтворення та клієнтами системи за допомогою мови програмування Python, бібліотеки Django та djangorestframework. Інтерфейс адміністратора системи вуличного радіо для авторизації адміністраторів, керування чергою та передачі повідомлень на станції в режимі реального часу мовою програмування JavaScript та фреймворком Next.js.

6. Проведено тестування системи вуличного радіо реалізованої розробленими алгоритмами та визначено напрямки інтеграції вуличного радіо з системою “розумного” міста.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дубчак Л. О., Гураль І. В. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп’ютерна інженерія» / ред. О. М. Березький. Тернопіль : ТНЕУ, 2019. 33 с.

2. Дубчак Л. О., Мельник Г. М. Методичні рекомендації до виконання кваліфікаційної роботи з освітнього ступеня “Магістр”. Спеціальність: 123 - Комп’ютерна інженерія. Магістерська програма — Комп’ютерна інженерія" / ред. О. М. Березький. Тернопіль : ЗУНУ, 2020. 32 с.

3. Іваніцкий Н.Ю., Кривко Р.В. Аналіз методів та алгоритмів шифрування повідомлень. V Науково-практична конференція молодих вчених і студентів «інтелектуальні комп’ютерні системи та мережі». 02 грудня 2021, Тернопіль, Україна.Тернопіль: ЗУНУ, 2021, с. 49.

4. Іваніцкий Н.Ю., Крисак Д.М. Модернізація продуктивності postgresql для використання з системою моніторингу zabbix. V Науково-практична конференція молодих вчених і студентів «інтелектуальні комп’ютерні системи та мережі». 02 грудня 2021, Тернопіль, Україна.Тернопіль: ЗУНУ, 2021, с. 50.

5. Іваніцкий Н.Ю., Сидяга В.В. Цифрові системи в інфраструктурі розумного міста. V Науково-практична конференція молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі». 02 грудня 2021, Тернопіль, Україна. Тернопіль: ЗУНУ, 2021, с. 48.

6. Aitzhan N. Z., Svetinovic D. Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams. IEEE transactions on dependable and secure computing. 2018. Vol. 15, no. 5. P. 840–852.

7. Applications of big data to smart cities / E. Al Nuaimi et al. Journal of internet services and applications. 2015. Vol. 6, no. 1.

8. A survey on smart grid potential applications and communication requirements / V. C. Gungor et al. IEEE transactions on industrial informatics. 2013. Vol. 9, no. 1. P. 28–42.

9. A survey on the critical issues in smart grid technologies / I. Colak et al. Renewable and sustainable energy reviews. 2016. Vol. 54. P. 396–405.

10. Balakrishna C. Enabling technologies for smart city services and applications. 2012 6th international conference on next generation mobile applications, services, and technologies (NGMAST), Paris, France, 12–14 September 2012.

11. Batty M. Big data, smart cities and city planning. Dialogues in human geography. 2013. Vol. 3, no. 3. P. 274–279.

12. Biswas K., Muthukkumarasamy V. Securing smart cities using blockchain technology. 2016 IEEE 18th international conference on high performance computing and communications; IEEE 14th international conference on smart city; IEEE 2nd international conference on data science and systems (hpcc/smartcity/dss), Sydney, Australia, 12–14 December 2016.

13. Blockchain contract: securing a blockchain applied to smart contracts / H. Watanabe et al. 2016 IEEE international conference on consumer electronics (ICCE), Las Vegas, NV, USA, 7–11 January 2016.

14. Blockchain for IoT security and privacy: the case study of a smart home / A. Dorri et al. 2017 IEEE international conference on pervasive computing and communications: workshops (percom workshops), Kona, HI, 13–17 March 2017.

15. Blockchain in intelligent transportation systems / D. Cocîrlea et al. Electronics. 2020. Vol. 9, no. 10. P. 1682.

16. Enabling smart cities through a cognitive management framework for the internet of things / P. Vlacheas et al. IEEE communications magazine. 2013. Vol. 51, no. 6. P. 102–111.

17. Enterprise Messaging with JMS and AMQP / M. Lui et al. Pro spring integration. Berkeley, CA, 2011. P. 397–450.

18. Friedli T., Bellm D. OPEX: a definition. Leading pharmaceutical operational excellence. Berlin, Heidelberg, 2013. P. 7–26.

19. Gelenbe E., Ceran E. T. Energy packet networks with energy harvesting. IEEE access. 2016. Vol. 4. P. 1321–1331.

20. Gelenbe E. Energy packet networks: ICT based energy allocation and storage. Lecture notes of the institute for computer sciences, social informatics and telecommunications engineering. Berlin, Heidelberg, 2012. P. 186–195.

21. Gelenbe E. Energy packet networks: smart electricity storage to meet surges in demand. Fifth international conference on simulation tools and techniques, Desenzano del Garda, Italy, 19–23 March 2012.

22. Gupta M. Z. Home Automation System using NodeMCU. International journal for research in applied science and engineering technology. 2020. Vol. 8, no. 4. P. 372–377.

23. Hawk: the blockchain model of cryptography and privacy-preserving smart contracts / A. Kosba et al. 2016 IEEE symposium on security and privacy (SP), San Jose, CA, 22–26 May 2016.

24. Heilman E., Baldimtsi F., Goldberg S. Blindly signed contracts: anonymous on-blockchain and off-blockchain bitcoin transactions. Financial cryptography and data security. Berlin, Heidelberg, 2016. P. 43–60.

25. Hoyler M., Harrison J. Global cities research and urban theory making. Environment and planning A: economy and space. 2017. Vol. 49, no. 12. P. 2853–2858.

26. Huh S., Cho S., Kim S. Managing IoT devices using blockchain platform. 2017 19th international conference on advanced communication technology (ICACT), Pyeongchang, Kwangwoon Do, South Korea, 19–22 February 2017.

27. Internet of things and big data analytics for smart and connected communities / Y. Sun et al. IEEE access. 2016. Vol. 4. P. 766–773.

28. Internet of things for smart cities / A. Zanella et al. IEEE internet of things journal. 2014. Vol. 1, no. 1. P. 22–32.

29. Li D., Cao J., Yao Y. Big data in smart cities. Science china information sciences. 2015. Vol. 58, no. 10. P. 1–12.

30. Mohanty S. P., Choppali U., Kougianos E. Everything you wanted to know about smart cities: The Internet of things is the backbone. IEEE consumer electronics magazine. 2016. Vol. 5, no. 3. P. 60–70.

31. Nam T., Pardo T. A. Conceptualizing smart city with dimensions of technology, people, and institutions. The 12th annual international digital government research conference, College Park, Maryland, 12–15 June 2011. New York, New York, USA, 2011.

32. Peters G. W., Panayi E. Understanding modern banking ledgers through blockchain technologies: future of transaction processing and smart contracts on the internet of money. Banking beyond banks and money. Cham, 2016. P. 239–278.

33. Pilicita Garrido A., Borja López Y., Gutiérrez Constante G. Rendimiento de MariaDB y PostgreSQL. Revista científica y tecnológica UPSE. 2020. Vol. 7, no. 2. P. 09–16.

34. Reid F. Http. Network programming in .NET. 2004. P. 87–130.

35. Sensing as a service model for smart cities supported by Internet of Things / C. Perera et al. Transactions on emerging telecommunications technologies. 2013. Vol. 25, no. 1. P. 81–93.

36. Smart cities at the forefront of the future internet / J. M. Hernández-Muñoz et al. The future internet. Berlin, Heidelberg, 2011. P. 447–462.

37. Smart cities of the future / M. Batty et al. The european physical journal special topics. 2012. Vol. 214, no. 1. P. 481–518.

38. Smart grid communications: overview of research challenges, solutions, and standardization activities / Z. Fan et al. IEEE communications surveys & tutorials. 2013. Vol. 15, no. 1. P. 21–38.

39. Smart grid technologies: communication technologies and standards / V. C. Gungor et al. IEEE transactions on industrial informatics. 2011. Vol. 7, no. 4. P. 529–539.

40. Smart grid – the new and improved power grid: a survey / X. Fang et al. IEEE communications surveys & tutorials. 2012. Vol. 14, no. 4. P. 944–980.

41. Smart transmission grid: vision and framework / F. Li et al. IEEE transactions on smart grid. 2010. Vol. 1, no. 2. P. 168–177.

42. Thakkar M. Next.js. Building react apps with server-side rendering. Berkeley, CA, 2020. P. 93–137.

43. The blockchain-based digital content distribution system / J. Kishigami et al. 2015 IEEE fifth international conference on big data and cloud computing (bdcloud), Dalian, China, 26–28 August 2015. 2015.

44. The role of big data in smart city / I. A. T. Hashem et al. International journal of information management. 2016. Vol. 36, no. 5. P. 748–758.

45. The scientific activities of the organizations of integration associations in conditions of emergencies / A. Kovler et al. Journal of foreign legislation and comparative law. 2020. Vol. 6, no. 4. P. 1.

46. Towards a big data analytics framework for iot and smart city applications / M. Strohbach et al. Modeling and Processing for Next-Generation Big-Data Technologies. Cham, 2015. P. 257–282.

47. Wang W., Xu Y., Khanna M. A survey on the communication architectures in smart grid. Computer networks. 2011. Vol. 55, no. 15. P. 3604–3629.

48. Wright C. S. Bitcoin: a peer-to-peer electronic cash system. SSRN electronic journal. 2008.

49. Zyskind G. Efficient secure computation enabled by blockchain technology: Thesis. 2016. 128 p.

50. Zyskind G., Nathan O., Pentland A. Decentralizing privacy: using blockchain to protect personal data. 2015 IEEE security and privacy workshops (SPW), San Jose, CA, 21–22 May 2015.