ДИПЛОМНИЙ ПРОЕКТ

ДП.ІПЗ-10.ПЗ

Група ІПЗ-41

Мазур Сергій

2020

Державний вищий навчальний заклад
“Прикарпатський національний університет імені Василя Стефаника”
Кафедра інформаційних технологій

УДК 004

ДИПЛОМНИЙ ПРОЕКТ

Тема: Розробка веб-сайту “Система управління службами міста Smartcity”

Спеціальність 121 “Інженерія програмного забезпечення”
код і назва спеціальності

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДП.ІПЗ-10.ПЗ
(позначення)

Рецензент
доц. Лазарович І.М.
(посада) (підпис) (дата) (розшифровка підпису)

Студент
ІПЗ-41 Мазур С.А.
(шифр групи) (підпис) (дата) (розшифровка підпису)

Нормоконтролер
доц. Лазарович І.М.
(посада) (підпис) (дата) (розшифровка підпису)

Керівник дипломного проекту
доц. Іщеряков С.М.
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту

Завідувач кафедри

доц. Козленко М.І.
(посада) (підпис) (дата) (розшифровка підпису)

2020
(рік)

Державний вищий навчальний заклад
«Прикарпатський національний університет імені Василя Стефаника»
Факультет математики та інформатики Кафедра інформаційних технологій
Спеціальність 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Завідувач кафедри Козленко М.І.
_____
“___” _____ 20___ р.

ЗАВДАННЯ
НА ВИКОНАННЯ ДИПЛОМНОГО ПРОЕКТУ

Студенту Мазуру Сергію Анатолійовичу
(прізвище, ім’я, по батькові студента)

1. Тема проекту Розробка веб-сайту “Система управління службами міста Smartcity”

затверджена розпорядженням по факультету математики та інформатики від 25 жовтня 2019 р., №7

2. Термін здачі студентом закінченого проекту 22 травня 2020 р.

3. Вихідні дані до дипломного проекту Стандарт кафедри Інформаційних Технологій ПНУ “Вимоги до змісту та оформлення”, UNE 178301:2015: 2015 – Smart cities – Open Data

4. Зміст пояснювальної записки (перелік питань, що їх належить опрацювати)
1. Аналіз розвитку технологій розумних міст
2. Обґрунтування вибору моделей та створення таблиць бази даних
3. Розробка та тестування аплікації
4. Економічне обґрунтування проекту

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень): “Мета роботи”, “Поняття Smart city”, “Ознаки технології, яку можна віднести до Smart city”, “UML Roles”, “Database schema”, “3-tier architecture”, “Spring Security authentication process”, “Spring Context”, “Швидкий старт з Spring Boot”, “Робота з базою даних”, “JDBC template”, “Тестування аплікації”

6. Дата видачі завдання 11.09.2019 р.

Керівник _____ Іщеряков С.М.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Мазур С.А.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів дипломного проекту	Термін виконання етапів проекту	Примітка
1. Аналіз розвитку технологій розумних міст	02.11.2019	Виконав
2. Обґрунтування вибору моделей та створення таблиць бази даних	15.02.2020	Виконав
3. Розробка та тестування аплікації	17.04.2020	Виконав
4. Економічне обґрунтування проекту	11.05.2020	Виконав
5. Оформлення пояснювальної записки	18.05.2020	Виконав

Студент _____ Мазур С.А.
(підпис) (розшифровка підпису)

Керівник проекту _____ Іщеряков С.М.
(підпис) (розшифровка підпису)

РЕФЕРАТ

Пояснювальна записка: 90 с., 56 рисунків, 16 джерел, 1 додаток.

Ключові слова: СИСТЕМА РОЗУМНЕ МІСТО, JAVA, ANGULAR, SPRING, USER, DBMS, CRUD, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, UML-МЕТОДОЛОГІЯ, ПРОГРАМНА РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ, MOCKITO.

Тема: моделі, методи, алгоритми інформаційно-програмного забезпечення для обробки даних та організації роботи міських установ.

Об’єкт дослідження: інформаційна модель системи “Розумне місто”: складові – інформаційні технології – безпека.

Мета роботи: розроблення моделей, методів, алгоритмів інформаційно-програмного для функціонування системи “Розумне місто”.

Предмет дослідження: інформаційні технології керування організаціями як складовими розумного міста і забезпечення безпеки при взаємодії між компонентами системи.

Результати дослідження: Проаналізовано складові системи Розумне місто, інформаційні технології підтримки функціонування і, на цій, основі інформаційну модель розумного міста. Обґрунтовано метод криптографічного захисту даних на основі хеш-функції. Вибрано принципи архітектури програмного забезпечення та проаналізовано вимоги до якості WEB - застосунків. Розроблено програмну реалізацію управління розумним містом, реалізовано авторизацію на основі JWT - технології і виконано тестування програмного продукту.

В результаті досліджень отримано програмну реалізацію системи безпечного управління розумним містом на основі: технології UML - моделювання, JWT-технології авторизації користувачів, технології захищеного підключення TLS, що уможливлює безпечну взаємодію складових розумного міста.

ABSTRACT

Explanatory note: 90 p., 56 figures, 16 references, 1 Appendix.

Keywords: SMART CITY SYSTEM, JAVA, ANGULAR, SPRING, USER, DBMS, CRUD, SOFTWARE ARCHITECTURE, UML METHODOLOGY, SOFTWARE IMPLEMENTATION, TESTING, MOCKITO.

Topic: models, methods, algorithms of information software for data processing and organization of work of city institutions.

Object of research: information model of the "Smart city" system: components-information technologies-security.

Purpose: development of models, methods, algorithms of information and software for the functioning of the "Smart city" system.

Subject of research: information technologies for managing organizations as components of a smart city and ensuring security in the interaction between system components.

Research result: The components of the Smart city system, information technologies to support the functioning and, on this basis, the information model of the smart city are analyzed. The method of cryptographic data protection based on hash function is proved. The principles of software architecture are selected and the quality requirements for WEB applications are analyzed. A software implementation of smart city management has been developed, authorization based on JWT technology has been implemented, and the software product has been tested.

Conclusion: As a result of research, the software implementation of a secure smart city management system based on: UML - modeling technology, JWT - user authorization technology, secure TLS connection, which makes it possible to safely interact with the components of a smart city.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ВНЗ – вищий навчальний заклад

БД – база даних

СУБД – система управління базами даних

REQUEST – запит до серверу

RESPONSE – відповідь сервера

USER – користувач

UML – Unified Modeling Language – мова моделювання

AJAX – Asynchronous JavaScript And XML - підхід до побудови користувацьких веб інтерфейсів

JSON – JavaScript Object Notation(стандарт обміну інформацією)

XML – eXtensible Markup Language (розширювана мова розмітки)

CRUD – Create Read Update Delete operations (операції створення, редагування та видалення)

ВСТУП

Актуальність роботи

Національна програма інформатизації, Стратегія сталого розвитку “Україна – 2020” та Концепція 4.0 розгортають процеси розроблення та впровадження технологій інтелектуалізації у всіх предметних сферах суспільства, серед яких технології, що формують інформаційну систему “Розумне місто”. Складовими системи “Розумне місто” є: розумні енергосистеми, розумний транспорт і логістика, розумна медицина, розумні екосистеми, розумні будівлі. Технології функціонування “розумного міста” – автоматизація, інтернет речей, машинне навчання.

Автоматизація процесів реалізується комплексом автоматизованих систем. Автоматизовані системи управління, інформаційно-аналітичні системи. Інтелектуальні системи керування, системи підтримки прийняття рішення, експертні системи функціонально спрямовані на контроль, оброблення інформації від об’єктів і, на цій основі, прийняття рішення на управління станом об’єкта.

Розглянуто технологію інтернет речей як мережі фізичних об’єктів з вбудованими давачами для реєстрації та передавання даних про стан різнорідних об’єктів, середовища та структури взаємодії “об’єкт – середовище”.

Описано роль машинного навчання (у контексті аналізу даних) як методу, який застосовують для побудови складних моделей та алгоритмів, які є підґрунтям для передбачення. Інтелектуалізація інформаційної інфраструктури суспільства пов’язана із розробленням веб-застосунків, що дозволяють користувачу виконувати дії, та веб-сайтів (веб-сторінок), які в основному є інформативними. Веб-застосунок отримує запит від клієнта і виконує обчислення, після цього формує веб-сторінку і відправляє її клієнтові мережею з використанням протоколу HTTP. Для безпечного передавання інформації

використовують протокол HTTPS, в якому дані передаються в зашифрованому вигляді.

Порівняння роботи з відомими розв'язаннями проблеми

Основними напрямами розвитку розумних міст в Празі є: розумне планування; розумне середовище; Smart Estate – використання розумних технологій для збору та аналізу даних про нерухомість для оптимізації циклів обслуговування та попередження проблем; розумне життя; Smart Community - використання аналізу даних та ІКТ, для кращого розуміння потреб мешканців та їх залучення, зростання ролі громади, а також надання громадам можливість брати більше відповідальності за співпрацю у своєму життєвому середовищі.

В Україні розумні міста повинні включати проекти-рішення з урахуванням: прозорості і доступності даних для громадян через відкритий портал даних або мобільний-застосунок. У цьому зв'язку є актуальним застосування сучасних захищених комунікаційних систем та захищених протоколів обміну даними.

Мета і задачі дослідження

Метою дипломної роботи є розроблення моделей, методів, архітектури інформаційно-програмного забезпечення для функціонування системи "Розумне місто".

Задачі дослідження:

1. створити інформаційну модель безпечного функціонування розумного міста у просторі: складові – технології підтримки – безпека;

2. аналіз архітектури програмного забезпечення та вибір алгоритму для проектування системи "розумне місто";

3. програмна реалізація системи "Розумне місто" та її тестування.

Об'єктом дослідження є інформаційна модель системи "Розумне місто": складові – технології – безпека.

Предметом дослідження є інформаційні технології керування організаціями як складовими розумного міста і забезпечення безпеки при взаємодії між компонентами системи.

Новизна одержаних результатів

Спроектовано архітектуру програмного забезпечення функціонування системи розумного міста відповідно до інформаційної моделі розумного міста, принципів проектування, вимог якості WEB-застосувань та вимог захисту інформації від несанкціонованого доступу. Реалізовано системний підхід до розроблення інформаційно-програмного забезпечення функціонування системи у просторі протоколів захищеного обміну даних.

Особистий внесок

1. проведено аналіз існуючих рішень та інформаційну модель розумного міста у просторі “складові – технології підтримки – безпека”;

2. запропоновано архітектуру, алгоритм функціонування та реалізацію системи «Розумне місто» з високим рівнем безпеки та надійності, яка використовує обмін даними через мережу інтернет.

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ РОЗУМНИХ МІСТ

1.1 «Розумне місто»: сутність поняття, багаторівнева модель системи

1.1.1 Сутність поняття, структура, принципи

Згідно Стратегії сталого розвитку “Україна – 2020” серед актуальних задач інтелектуалізації суспільства України, що обумовлені проблемами, які виникають у великих містах, серед яких: енергетика, відходи і вода, охорона здоров’я, громадський транспорт, безпека і захищеність, освіта, послуги, робочі місця та економічний розвиток - проектування інформаційних систем “Розумне місто” (Smart City) відповідно до складових системи і технологій забезпечення їх функціонування.

Інформаційна система “Розумне місто” – це взаємозв’язана структура розумних складових, розумних технологій їх функціонування, елементів інформаційної безпеки, комплексно спрямованих на забезпечення безпечного, екологічного, комфортного функціонування інфраструктури суспільства та управління проблемними ситуаціями.

Суб'єкти розумного міста – міська влада і державні установи; мешканці, бізнес-установи. Об'єкти інфраструктури і послуг розумного міста: освіта, здоров'я, навколишнє середовище, транспорт, водопостачання, перероблення сміття, енергія, захист. Технології забезпечення функціонування розумного міста: аналіз даних, кібербезпека, геоінформаційна система, фізична інфраструктура та мережа, управління змінами, планування, управління. Розгалужена структура розумного міста: розумна енергетика, розумна мобільність , розумна економіка, розумне довкілля, розумна культура, розумна медицина, розумна безпека, розумне правосуддя, розумні органи управління [1].

Принципи системи Розумне місто:

1) інформаційно-орієнтоване: збір, інтеграція, доповнення, зберігання і передавання даних для вдосконалення процесу прийняття рішень;

2) взаємозв’язане – ефективне використання інтернету речей (IoT) для покращання можливостей міста сприймати та реагувати на нагальні потреби мешканців;

3) інтелектуальне – використання аналітики та когнітивних інструментів для стимулювання поведінкових змін;

4) об’єднане – об’єднання міських службовців і мешканців для співпраці, де ключовими аспектами є прозорість та відкритість даних;

5) чітко організоване – надання мешканцям, установам організованих послуг.

Функціональність системи Розумне місто чітко пов’язана з трьома технологіями концепції Індустрія 4.0 (четвертої промислової революції) у сегментах: автоматизації, інтернету речей, машинного навчання. Основними засобами інтелектуалізації є: інтернет речей та інформаційно-комунікаційні технології (ІКТ). Інтернет речей (IoT – інфраструктура) – комплекс сенсорів, давачів, пристроїв, вимірювальних приладів, які реєструють/вимірюють дані з об’єктів фізичного середовища, порівнюють з встановленими нормами, передають оброблену інформацію на управління в узагальнену платформу.

IoT – інфраструктура та ІКТ функціонально пов’язані. IoT – інфраструктура пов’язана із: збиранням та підготовленням даних – поглиблений аналіз даних, штучні нейронні мережі, прогностичні механізми; наданням послуг – моделювання міського середовища, інформаційні панелі, додатки, інтерфейси; підключені мешканці та установи міста – розумні додатки, побудові пристрої, підключені об’єкти; приймання, внесення, управління даними – метадані, опрацювання подій, віртуалізація даних, тиражування, синхронізація, якість даних; зв'язок та обмін інформацією – провідні та безпровідні технології, шина передавання повідомлень, WEB-сервіси, програмний інтерфейс. WEB-сервіси забезпечують взаємодію програмних систем незалежно від платформи і

засновані на базі відкритих стандартів та протоколів. У цьому контексті актуальним є використання захищених протоколів, один з яких SSL (Secure Sockets Layer) і його наступник TLS (Transport Layer Security) - це протоколи для встановлення аутентифікованих і зашифрованих з'єднань між мережевими комп'ютерами [2].

1.1.2 Досвід розвитку інтелектуальних систем

Світовий досвід – місто Прага. Чеська Республіка - країна з населенням 10 мільйонів чоловік, розташована в Центральній Європі. Столиця, є найбільшим містом в країні і вносить близько 30 відсотків економіки держави. Прага стикається з багатьма з тих же проблем, з якими стикаються і інші міста. Місто зростає, і ті, хто хоче жити в місті, очікують комфортного життя, покладаючи великі надії на все - від житла і транспорту до утилізації відходів та енергетики. За даними ОЕСР, до 2050 року приблизно 70% населення світу буде жити в містах. Неухильно зростаюча щільність населення - і пов'язане з цим збільшення трафіку - вимагають нових підходів до міського планування. Наприклад, комунальні послуги тепер розглядаються не окремо, ізольовано, а як єдине мережеве ціле. При такому цілісному підході міська інфраструктура розглядається як свого роду центральна нервова система. Сучасне місто: система систем.

Енергетика, водопостачання, телекомунікації, транспорт... До цих пір кожна система оптимізувалася окремо, найчастіше без урахування пов'язаних систем. Саме Мережева взаємодія цих систем робить міське середовище "розумною". Результат: поліпшення зв'язку між системами і регулярний обмін інформацією. Але ця складна мережева структура вимагає використання стандартів і специфікацій. У Празі проектами "Розумного міста" займається окрема компанія, що належить державі. Тут Павло Тесар, представник цієї

компанії, розповідає, як працює механізм і приклади проектів, над якими вони працювали.

Державна компанія була створена для задоволення потреб IT та смарт-сіті Праги. Перевага полягає в тому, що компанія здатна швидко і ефективно працювати по міських запитах і не відчуває проблем з бюджетом. З іншого боку, Павло каже, що отримання дозволів може бути складним, оскільки іноді міські організації не дуже добре співпрацюють, а сама компанія не має прямої ролі у прийнятті рішень.

Компанія створила політику, яку внутрішньо вони називають конвеєрною. Ця політика управління процесами допомагає, тому що вона спрямовує ідеї до реалізації.

В даний час в місті працює 60 проектів, деякі з яких вже завершені, а інші- абсолютно нові. Роботою займається група з 120 співробітників.

Компанія залучає зацікавлених осіб для надання допомоги в реалізації проектів. Це стосується і економічної сфери, і наукових кіл. Вони не вважають себе кращими, але знають, що, працюючи з зацікавленими сторонами, вони можуть мати доступ до найяскравіших умів у країні та за її межами.

Стратегія для проектів розумного міста розділена на п'ять областей- мобільність - розумні будівлі, місто без відходів, люди в міському середовищі і туризм. Шоста область - платформа даних, яка є флагманським проектом.

Проекти включають в себе вищезгадану платформу даних, а також сенсорні проекти для аналізу трафіку та навколишнього середовища, регіональну транспортну систему та моніторинг/економію енергії.

Павло каже, що будь-яка ідея IT або Smart City, яка успішно проходить через Раду, вирішується його командою. Хоча вони досягли прогресу в багатьох областях, вони все ще працюють над пошуком хороших рішень інших проблем, включаючи необхідність у більшій кількості зарядних станцій для забезпечення електронної мобільності в місті.

Ці приклади розумних міст малюють картину того, що відбувається на місцях в ініціативах розумних міст по всьому світу, де технології та інновації

використовуються для вирішення міських проблем і поліпшення якості життя громадян.

Якщо метою розумних міст є підвищення ефективності надання муніципальних послуг, а в цілому-підвищення якості життя, то розумні спільноти прагнуть до процвітання в умовах широкосмугової економіки, її рушійної сили і сенсу існування. У цьому сенсі зосередження уваги на громаді, як вказує той же термін, прагне зробити кращі міста, роблячи ставку на варіант прогресу, який шукає баланс між місцевою економікою і глобалізацією.

Незалежно від розміру міста, будь то міський або сільський, мета полягає в підвищенні його конкурентоспроможності і рівня життя. Інформаційно-комунікаційні технології використовуються для побудови "інклюзивного процвітання, вирішення соціальних проблем і підвищення якості життя в нашому пов'язаному столітті", згідно концепції форуму інтелектуальної спільноти.

Важливу роль в обміні досвідом посідає форум Smart City - створений, зокрема, представниками державного управління, президентами і директорами провідних компаній, експертами з-за кордону, які представляють інноваційні рішення, що зарекомендували себе, наприклад, в Барселоні, Сінгапурі або Торонто. Це надзвичайно плідна дискусія про проблеми, що стоять практично перед усіма областями: від транспорту та інформаційних технологій через екологію, енергетику, будівництво до медицини і людського спілкування. В ході зустрічі фахівці обговорюють найбільш ефективні моделі реалізації конкретних інвестицій [3].

1.1.3 Багаторівнева модель системи

Динамічне і стійке місто - це екосистема, що складається з людей, організацій і підприємств, політики, законів і процесів, об'єднаних разом для досягнення бажаних результатів. Це місто адаптивне, чуйне і завжди актуальне для всіх тих, хто живе, працює і відвідує місто. Розумне місто інтегрує технології для прискорення, полегшення та перетворення цієї екосистеми.

В екосистемі розумного міста існує чотири типи творців цінності. Вони створюють і споживають цінність навколо одного з результатів, перерахованих на рисунку 1.1.

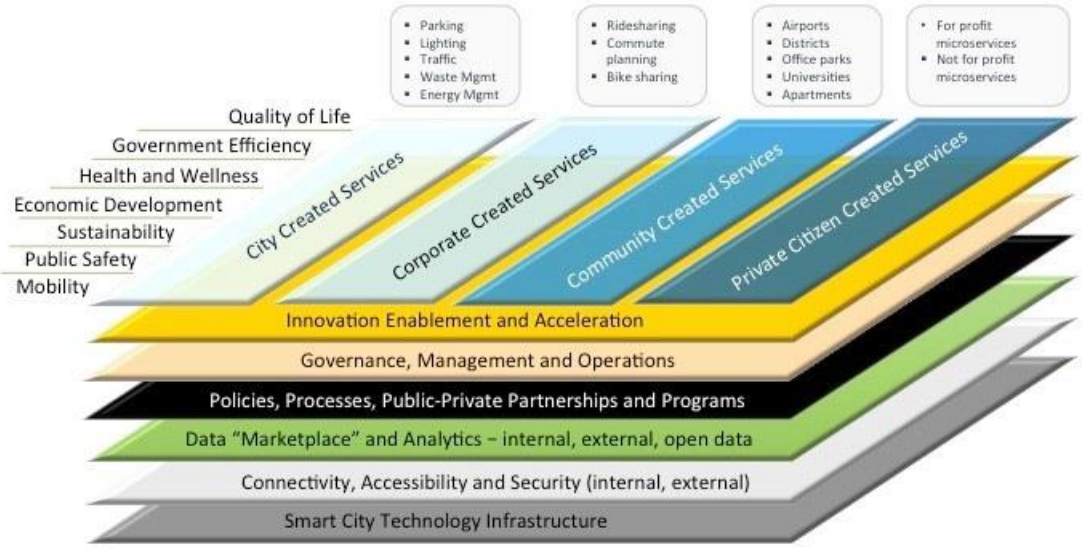

Рисунок 1.1 – Шари Розумного міста

Коли люди думають про розумне місто, вони автоматично думають про послуги, надані муніципальними та квазідержавними установами, такими як розумне паркування, розумне управління водними ресурсами, Розумне освітлення тощо.

Підприємства та організації можуть створювати послуги, які використовують та створюють інформацію для досягнення результатів для своїх

зацікавлених сторін. Деякі приклади" розумного " бізнесу включають Uber і Lyft для особистої мобільності, NextDoor для обміну інформацією і Waze / Google для планування трафіку і поїздок на роботу.

Громади - це мініатюрні розумні міста, але з дуже локалізованими потребами. Деякі приклади потенційних інтелектуальних спільнот включають Університетські кампуси, офісні парки, аеропорти, Вантажні порти, багатоквартирні будинки або житлові комплекси, Житлові комплекси/квартали, ділові райони і навіть окремі "розумні" будівлі. У них є потреби в інтелектуальних послугах, які можуть бути адаптовані спеціально для їх зацікавлених сторін.

Жителі або окремі громадяни також є постачальниками інтелектуальних послуг в розумному місті. Житель, що живе поблизу небезпечного перехрестя вулиць, може направити камеру на перехрестя і передати цю інформацію в прямому ефірі фахівцям з планування дорожнього руху і поліції. Жителі розміщують датчики вимірювання якості повітря на своїх об'єктах, щоб відстежувати рівень забруднення і пилку в певний час року, і доводять цю інформацію до інших членів спільноти. Резиденти можуть зробити ці інтелектуальні послуги тимчасовими або постійними, а також безкоштовними або платними.

Шари розумного міста

Розумне місто - це екосистема, що складається з декількох "шарів можливостей". Хоча технологія є найважливішим фактором, вона є лише одним з багатьох фундаментальних можливостей, якими повинен володіти кожен розумне місто. Жодна здатність не є більш важливою, ніж інші. Кожна здатність відіграє свою роль в розумному місті. Ці можливості повинні об'єднуватися і координуватися один з одним для виконання своєї місії.

Шар цінностей. Це найбільш помітний шар для жителів міста, підприємств, відвідувачів, робітників, студентів, туристів та інших. Цей шар являє собою каталог послуг "розумного міста" або "варіантів використання", зосереджених

навколо результатів і пропонованих творцями цінності і споживаних зацікавленими сторонами міста.

Інноваційний шар. Щоб залишатися актуальними, творці цінностей в розумному місті повинні постійно впроваджувати інновації та оновлювати свої послуги для зацікавлених сторін. Розумні міста активно сприяють цьому за допомогою різних інноваційних програм, включаючи лабораторії, інноваційні зони, Тренінги, семінари зі створення ідей, розвиток навичок і партнерство з університетами та підприємствами.

Рівень управління, управління та операцій. Розумне місто призводить до цифрової трансформації існуючих процесів і послуг. Моделі управління розумними містами повинні інтегрувати нову екосистему творців цінностей і новаторів. Вони повинні планувати, підтримувати та монетизувати нові бізнес-моделі, процеси та послуги. Вони повинні модернізувати свою існуючу інфраструктуру та процеси управління для підтримки "розумних" послуг. Нарешті, вони повинні вимірювати ефективність роботи міста за допомогою нового набору показників.

Політика, процеси, державно-приватне партнерство та рівень фінансування. Розумне місто не просто чарівним чином з'являється в один прекрасний день. Для побудови, експлуатації та підтримки "розумного міста" потрібен абсолютно новий набір моделей взаємодії, правил, джерел фінансування та партнерів. Міста повинні розробити новий набір "розумних" компетенцій, щоб отримати і залишитися в грі "Розумне місто".

Інформаційний та інформаційний шар. Життєва сила розумного міста - це інформація. Розумне місто має сприяти цьому кількома способами, включаючи ініціативи з відкритих даних, ринки даних, аналітичні послуги та Політику монетизації. Не менш важливо, що вони повинні мати програми, що заохочують обмін даними і Політику Конфіденційності, щоб захистити те, що і як збираються дані.

Рівень підключення, доступності та безпеки. Люди, речі та системи взаємопов'язані в розумному місті. Можливість легко з'єднати всі три, управляти

і перевіряти, хто і що підключено і спільно використовується, при цьому захист інформації і користувачів має вирішальне значення. Найвищі пріоритети для розумних міст-це забезпечення безшовного шару надійних з'єднань.

Рівень технологічної інфраструктури розумного міста. Більшість людей автоматично думають про технологію, коли говорять про розумні міста. Технологічна інфраструктура розумного міста повинна виходити за рамки традиційних муніципальних користувачів і підтримувати новий клас творців цінностей і зацікавлених сторін міста/користувача.

Розумне місто - це складна екосистема людей, процесів, політики, технологій та інших факторів, що працюють разом для досягнення ряду результатів. Розумне місто не" належить " виключно місту. Інші творці цінностей також беруть участь, іноді працюючи у співпраці, а іноді і самі по собі. Успішні та стійкі розумні міста використовують програмний підхід для залучення своїх зацікавлених сторін до екосистеми.

Багато міст не використовують екосистемний підхід до проектів розумних міст. Частково це пов'язано з тим, що проектами "розумного міста" управляє організація інформаційних технологій (IT), статут якої присвячений розробці та розгортанню систем. Навпаки, більш досвідчені Розумні міста керують своїми програмами через внутрішні крос-функціональні трансформаційні або інноваційні організації.

Незалежно від того, де знаходяться міста в їх подорожі по розумному місту, вони повинні випереджати "криву" з проектами розумного міста. Вони починають з мислення в термінах побудови більш широкої екосистеми, щоб створити стійкий і масштабований Розумне місто. Ключові наступні кроки включають:

– зрозуміти структуру екосистеми розумного міста і адаптувати її до реалій свого конкретного міста. Включити цю модель у розробку свого бачення розумного міста, стратегії та планів реалізації. Щодо структури екосистеми розумного міста - визначення можливостей та прогалин між різними верствами. Зрозуміти, що необхідно для підтримки чотирьох типів творців цінностей;

– оцінка існуючих і нових проектів та ініціатив "розумного міста" в рамках екосистеми. Використати цю структуру, щоб визначити, чого не вистачає в планах проекту і що необхідно, щоб зробити проекти повністю успішними;

– визначення пріоритетів та розвиток компетенцій на різних рівнях екосистеми. Розумне місто вимагає нових навичок і компетенцій. При необхідності треба нарощувати існуючі можливості за рахунок стратегічного партнерства та укладення контрактів з постачальниками послуг.

1.2 Напрямки розумних технологій

Інтегровані комунікації. У більшості випадків дані збираються по модемному з'єднанню, а не по прямому мережевому з'єднанню.

Можливості для удосконалення включають: автоматизацію підстанцій, реагування на попит, автоматизацію розподілу, системи керування та спостереження (SCADA), системи керування енергією, безпровідні меш-мережі, комунікації по лініям електропередачі і оптичному волокну. Інтегровані комунікації дозволяють керування у реальному часі, обмін даними для оптимізації надійності, ефективності використання активів та безпеки [4]. SCADA (Supervisory Control And Data Acquisition – диспетчерське управління і збирання даних) – програмний пакет, призначений для розроблення/ забезпечення роботи в реальному часі систем збирання, оброблення, відображення та архівування інформації про об'єкт моніторингу або управління. Серед завдань SCADA-системи такі: обмін даними з промисловими контролерами і платами вводу-виводу в реальному часі через драйвери; оброблення інформації в реальному часі; логічне управління; Відображення інформації на екрані монітора в зручній і зрозумілій для людини формі; ведення бази даних реального часу з технологічною інформацією; аварійна сигналізація і управління тривожними повідомленнями; підготовлення та генерування звітів

про хід технологічного процесу; здійснення мережевої взаємодії між SCADA ПК; забезпечення зв'язку з зовнішніми додатками (СУБД, електронні таблиці, текстові процесори).

Давачі та вимірювачі. Основними задачами є оцінка стабільності енергосистеми, моніторинг стану обладнання, попередження крадіжки енергії і підтримання стратегії керування.

Технології включають в себе:

– передові мікропроцесорні системи моніторингу та вимірювання (розумні лічильники) і обладнання зчитування даних з лічильників

– системи розподіленого моніторингу -динамічної оцінки ліній; системи вимірювання/ аналізу електромагнітних

– системи вимірювання часу споживання та ціноутворення у реальному часі.

Високотехнологічні компоненти. Цей напрямок забезпечують: інновації у надпровідності, стійкість до відмов, зберігання енергії, діагностичні компоненти, які змінюють фундаментальні властивості мереж.

Водночас актуальними є технології забезпечення: використання кабелів з високотемпературних надпровідників, розподіленого генерування і зберігання енергії, використання композитних провідників "інтелектуальних" приладів/ систем.

Інтелектуальне керування. Автоматизація енергосистеми дозволяє швидке діагностування, точні рішення на порушення у мережі або відключення.

Ці технології спираються на і сприяють кожній з інших чотирьох ключових областей.

Три категорії технологій для інтелектуального керування включають: розподілених інтелектуальних агентів (системи керування), інструменти аналітики (програмне забезпечення та швидкодіючі комп'ютери), операційні застосування (SCADA, автоматизація підстанцій і т.п.).

Удосконалені інтерфейси, підтримка прийняття рішень. Цей напрямок охоплює: технології візуалізації, що зводять велику кількість даних у візуальні

формати, які легко сприймаються; програмні системи, які надають численні можливості коли потрібне втручання оператора; системи аналізу сценаріїв.

1.3 Порівняння роботи з відомими розв'язаннями проблеми

Основними напрямами розвитку розумних міст в Празі є: розумне планування; розумне середовище; Smart Estate – використання розумних технологій для збору та аналізу даних про нерухомість для оптимізації циклів обслуговування та попередження проблем; розумне життя; Smart Community - використання аналізу даних та ІКТ, для кращого розуміння потреб мешканців та їх залучення, зростання ролі громади, а також надання громадам можливість брати більше відповідальності за співпрацю у своєму життєвому середовищі.

Розумні міста також можуть використовувати пристрої Інтернету речей (IoT), такі як датчики підключення, світлові прилади та лічильники для збору та аналізу даних. Потім міста використовують ці дані для поліпшення інфраструктури, комунальних служб і послуг і багато чого іншого. Ще одним прикладом таких сучасних міст є Амстердам. Це місто лише одне з багатьох, особливо в Європі, яка лідирує в світі з розвитку розумних міст. ЄС активно заохочує свої країни-члени до розвитку розумних міст, і Європейська комісія виділила на ці цілі 365 мільйонів євро.

І ці зусилля вже почали приносити свої плоди. Париж дебютував у 2011 році в програмі обміну електромобілями під назвою Autolib, і з тих пір його парк виріс до 3000 автомобілів. Підключені транспортні засоби можна відстежувати за допомогою GPS, а водії можуть використовувати приладову панель автомобіля, щоб заздалегідь зарезервувати паркувальні місця.

Лондон оголосив раніше цього року, що він почне випробування проекту Smart Parking, який дозволить водіям швидко знаходити паркувальні місця і позбавить їх від необхідності тривалих пошуків відкритого місця. Це, в свою

чергу, полегшило б міські затори. Столиця Великобританії також планує протестувати програми обміну електромобілями і велосипедами.

Тим часом Копенгаген почав використовувати датчики для моніторингу велосипедного руху міста в режимі реального часу, що дає цінні дані про поліпшення велосипедних маршрутів в місті. Це дуже важливо, так як більше 40% жителів міста щодня їздять на велосипеді.

Північна Америка відстала, хоча це самий урбанізований регіон у світі, де більше 80% населення проживає в міських центрах. Тим не менш, в цих країнах існує безліч проектів "розумного міста", особливо в тому, що стосується громадської безпеки та дорожнього руху.

Нью-Йорк протестував технологію виявлення вогнепальної зброї в поліцейських дільницях Брукліна і Бронкса, і мер хоче розширити це тестування по всьому місту. Кемден, штат Нью-Джерсі, впровадив аналогічну технологію.

Нью-Йорк також пілотував програму «Connected car» в 2015 році з метою вивчення того, де водії часто роблять жорсткі гальма або різкі повороти через рух. Потім чиновники могли б використовувати ці дані для поліпшення дорожніх умов і полегшення дорожнього руху.

Нарешті, Сан-Дієго почав використовувати камери, вбудовані в підключені вуличні ліхтарі, щоб стежити за рухом пішоходів і перенаправляти автомобілі в години пік, щоб уникнути нещасних випадків з пішоходами і зменшити затори.

В Україні розумні міста повинні включати проекти-рішення з урахуванням: прозорості і доступності даних для громадян через відкритий портал даних або мобільний-застосунок. У цьому зв’язку є актуальним застосування сучасних захищених комунікаційних систем та захищених протоколів обміну даними [5].

1.4 Майбутнє Інтернету речей і Розумних міст

Потенціал розумних міст практично безмежний, і зростання цих міст має тільки прискоритися в найближчі роки. Але це не єдина область, в якій IoT в найближчому майбутньому кардинально зміниться. Ось чому Bi-Intelligence зібрала звіт про дослідження Інтернету речей. Ось п'ять складових розумного міста і їх вплив в епоху Інтернету речей:

1. Інтелектуальна інфраструктура

– міста повинні створювати умови для безперервного розвитку: цифрові технології набувають все більшого значення, міська інфраструктура і будівлі повинні плануватися більш ефективно і стійко;

– викиди CO2 повинні бути якомога нижчими, наприклад, інвестиції в електромобілі та самохідні транспортні засоби;

– розумні міста використовують інтелектуальні технології для створення енергоефективної та екологічно чистої інфраструктури;

– розумне освітлення повинно давати світло тільки тоді, коли хтось дійсно проходить повз них, наприклад, встановлюючи рівні яскравості і відстежуючи щоденне використання, щоб зменшити потребу в електроенергії.

2. Інструмент Управління Міським Повітрям (CyAM)

Компанія Siemens розробила повний хмарний програмний комплекс "The City Air Management Tool": фіксує дані про забруднення в режимі реального часу і прогнозує викиди. Прогнози з точністю до 90% можна отримати за викидами на найближчі три-п'ять днів. Саме прогнозування забруднення атмосферного повітря з вимірюванням ефективності і використовуваних технологій робить інструмент управління міським повітрям унікальним. Прогноз заснований на алгоритмі, який працює зі штучною нейронною мережею.

CyAM - це хмарний програмний комплекс з приладовою панеллю, яка відображає інформацію в режимі реального часу про якість повітря, виявленому датчиками по всьому місту, і прогнозує значення на найближчі три-п'ять днів.

Міста можуть вибрати з 17 заходів для моделювання наступних трьох-п'яти днів (вплив якості повітря на майбутні три-п'ять днів). Наслідки: введення нових екологічних зон(зон з низьким рівнем викидів), обмеження швидкості руху або безкоштовний громадський транспорт. CyAM заснована на MindSphere, хмарній відкритій операційній системі Siemens для Інтернету речей (IoT) [6].

3. Управління рухом

Завдання для великих розумних міст - оптимізувати трафік. Лос-Анджелес: будучи одним з найжвавіших міст у світі, місто впровадило інтелектуальне транспортне рішення для управління транспортним потоком. Вбудовані в дорожнє покриття датчики посилають в режимі реального часу оновлену інформацію про дорожній потік на центральну платформу управління дорожнім рухом, яка аналізує дані і автоматично налаштовує світлофори відповідно до дорожньої ситуації протягом декількох секунд. Він використовує історичні дані, щоб передбачити, куди може піти трафік-все без участі людини [7].

4. Смарт - Паркінг

Інтелектуальні рішення щодо паркування визначають, коли автомобіль покинув стоянку. Датчики в землі повідомляють через смартфон водієві, де вони можуть знайти вільне місце для паркування. Інші використовують зворотний зв'язок з транспортними засобами, щоб точно визначити, де знаходяться отвори, і підштовхують очікують автомобілі до шляху найменшого опору

Розумні парковки - це реальність сьогоднішнього дня і не вимагають складної інфраструктури і високих інвестицій, що робить їх ідеальними для розумного міста середнього розміру. [8]

5. Інтелектуальне Управління Відходами

Рішення з управління відходами допомагають оптимізувати ефективність збору відходів, знизити експлуатаційні витрати і краще вирішувати екологічні проблеми, пов'язані з неефективним збором відходів.

Контейнер для відходів отримує датчик рівня; при досягненні певного порогу платформа управління водія вантажівки отримує повідомлення на

смартфон. Повідомлення, мабуть, спорожняє повний контейнер, що дозволяє уникнути напівпорожніх зливів [9].

Майбутнє Інтернету речей - безмежне. Вона надає рішення у всіх секторах, включаючи виробництво, моду, ресторан, охорону здоров'я, освіту і т.д. Розумні міста можуть спільно використовувати загальну платформу розумного міста, що має сенс особливо для невеликих міст. Хмарний характер рішень Інтернету речей для розумних міст доречний при спільному використанні платформи, заснованої на відкритих даних. Малі міста можуть утворити загальну міську екосистему. Таким чином, рішення малих і великих розумних міст об'єднуються в мережу і управляються через центральну хмарну платформу. Нарешті, що ще важливіше, розмір міста не є перешкодою на шляху до того, щоб стати "розумним". Міста в кожній групі можуть отримати вигоду з інтелектуальних технологій.

1.4.1 IoT технології

Система Інтернету речей може обробляти і передавати інформацію тільки в режимі онлайн. Наприклад, коли пристрої в IoT безпечно підключені до комунікаційних мереж. Тут виникає питання про те, наскільки можливе з'єднання і які види з'єднань доступні для того, щоб ці тисячі пристроїв могли спілкуватися один з одним. Відповіддю на всі ці питання є інтернет протоколи. Протоколи дозволяють цим пристроям взаємодіяти один з одним, безпечно обмінюватись інформацією. До цього часу було створено велику кількість протоколів, але на цьому рух не зупиняється, вводяться більш сучасні та безпечніші.

1. Bluetooth

Однією з найбільш широко використовуваних бездротових технологій ближньої дії є Bluetooth. Є можливість швидко підключитись до пристроїв з Bluetooth, які пропонують зручну технологію сполучення з розумними

гаджетами. Нещодавно представлений протокол Bluetooth серед протоколів IoT є іонним або низькоенергетичним протоколом Bluetooth. Він дозволить собі діапазон звичайного Bluetooth в поєднанні з більш низьким енергоспоживанням переваги. Місце Bluetooth в IoT представлено на рисунку 1.2.

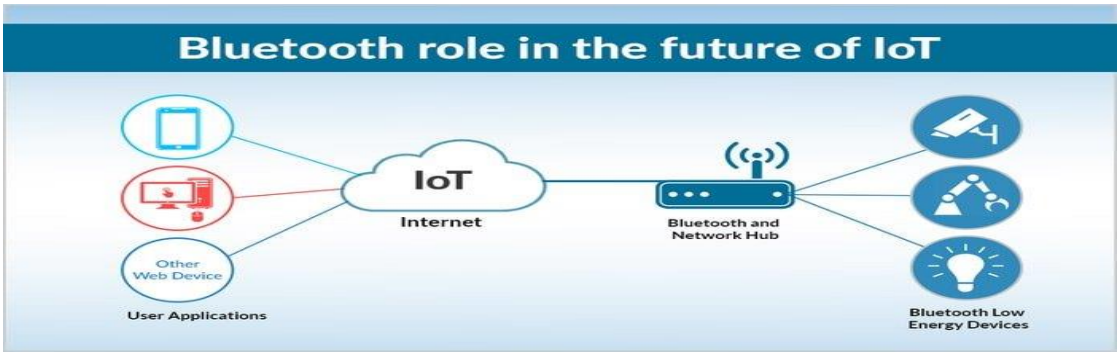

Рисунок 1.2 - Роль Bluetooth в IoT

Треба пам'ятати, що BLE не призначений для передачі великих файлів і буде відмінно поєднуватися з невеликими порціями даних. Саме з цієї причини Bluetooth лідирує в протоколах Інтернету речей цього століття.

2. Wi-Fi

Wi-Fi є найпоширенішою технологією на думку багатьох електронних дизайнерів. Це відбувається через інфраструктуру, яку він несе. Він має велику швидкість передачі даних, а також здатність працювати з великою кількістю даних. В поширеному стандарті Wi-Fi 802.11 представлена можливість передачі даних в сотні мегабіт за секунду. Можливості використання Wi-Fi представлено на рисунку 1.3.

Рисунок 1.3 – Приклади використання Wi-Fi

Єдиним недоліком цієї технології є те, що він може споживати надмірну кількість електроенергії для деяких пристроїв Інтернету речей. Він коливається приблизно в 50 м і поряд з роботою над стандартами Інтернет-протоколу включає в себе доступ до хмарної інфраструктури Інтернету речей. Ці частоти мають діапазони 2,4 ГГц і 5 ГГц.

3. ZigBee

Так само, як і Bluetooth, існує велика користувацька база ZigBee. Серед протоколів Інтернету речей, ZigBee призначений більш для використання в промисловому середовищу ніж в домашньому. Він зазвичай працює на частоті 2,4 ГГц. Ідеально підходить для промислових об'єктів, де дані, як правило, передаються невеликими порціями між будівлями [10]. Галузі використання ZigBee представлено на рисунку 1.4.

Рисунок 1.4 – Сфери використання ZigBee

ZigBee і його пульт дистанційного керування є популярними у використанні за свою можливість підтримки безпечних, малопотужних, масштабованих систем з високою кількістю вузлів.

4. MQTT IoT

MQTT IoT - це протокол транспортування черги повідомлень. Він був розроблений в 1999 році Арленом Ніппером (Arcom) і Енді Стенфорд-Кларком (IBM). Він в основному використовується для моніторингу з віддаленої області

в IoT. Основне завдання, яке виконує MQTT - отримання даних від великої кількості електронних пристроїв.

Він працює на вершині TCP протоколу для забезпечення надійних, але простих потоків даних. Цей протокол MQTT складається з трьох основних компонентів або механізмів: Subscriber, Publisher, and Broker. Робота Publisher'a полягає в генеруванні даних і передачі їх передплатнику за допомогою брокера. Забезпечення безпеки - це робота брокера. Він робить це, авторизацію передплатників і видавців. Робота MQTT з пристроями представлена на рисунку 1.5.

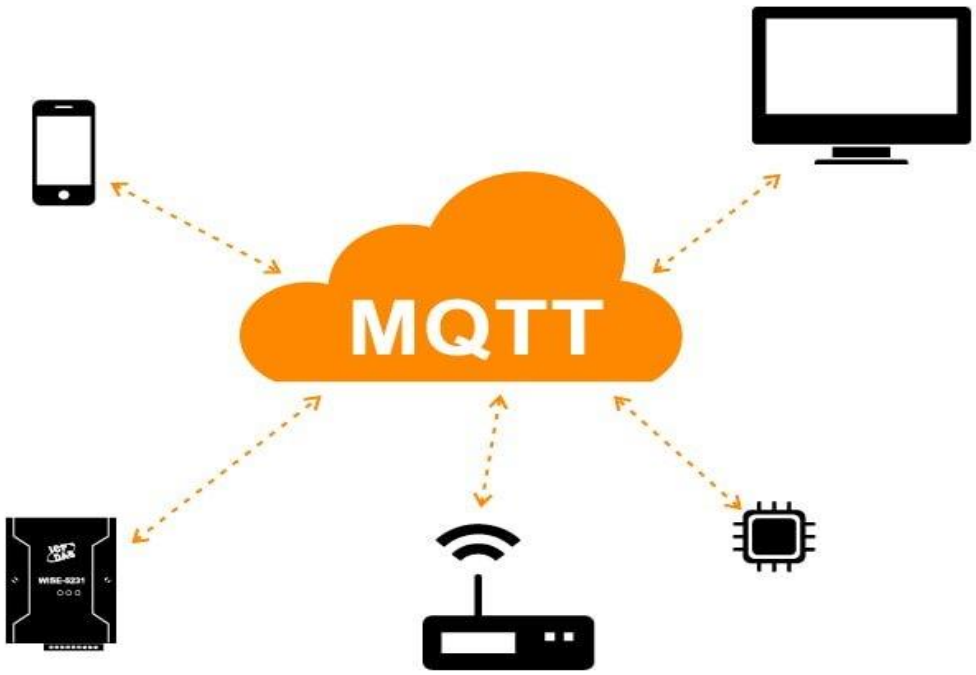

Рисунок 1.5 – Пристрої використання MQTT

Цей протокол є кращим варіантом для всіх пристроїв, заснованих на IoT, що також здатен забезпечити достатню кількість функцій маршрутизації інформації для дешевих, маловитратних і невеликих пристроїв з низьким енергоспоживанням пам'яті за допомогою мережі з низькою і вразливою пропускною здатністю.

5. CoAP

CoAP або обмежений прикладний протокол, в основному розробляється для обмежених інтелектуальних гаджетів. Дизайн CoAP призначений для використання його серед пристроїв, які мають однакову обмежену спільноту. Вона включає в себе загальні вузли та пристрої в мережі та різні пристрої, які з'єднані в Інтернеті. Системи CoAP IoT, засновані на протоколах HTTP, можуть надзвичайно добре поєднуватися з мережевими протоколами CoAP IoT. Він використовує протокол UDP для реалізації полегшених даних. Так само, як і HTTP, він також використовує архітектуру RESTful. Він також використовується всередині мобільних телефонів та інших соціальних спільнот, які є базовими програмами. CoAP допомагає позбутися двозначності за допомогою методів HTTP.

Характеристики протоколу CoAP представлено на рисунку 1.6.

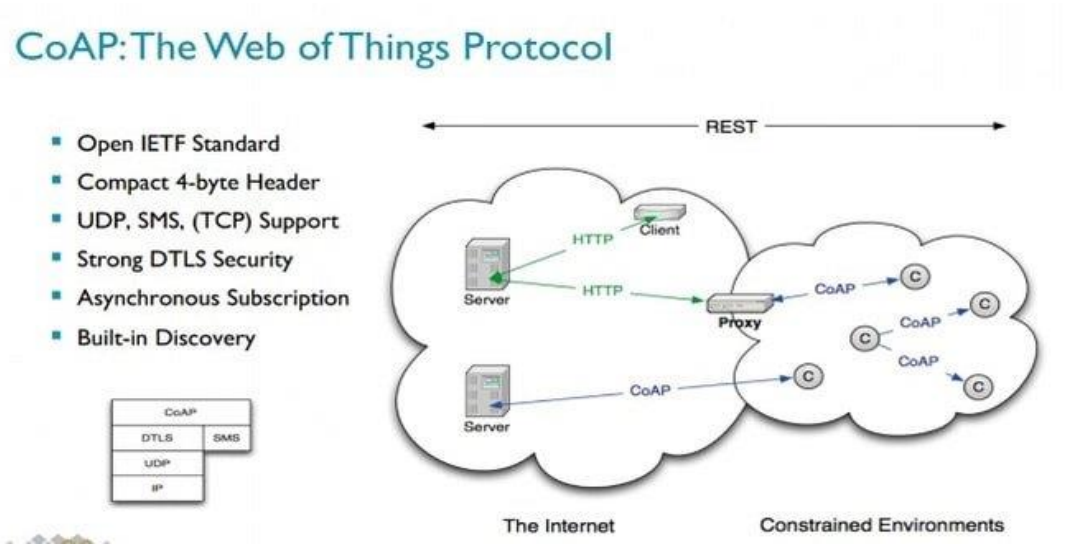

Рисунок 1.6 – Характеристика CoAP

6. NFC

NFC з протоколів IoT використовує переваги безпечного двостороннього зв'язку. Зв'язок NFC або Near Field дозволяє клієнтам підключатися до електронних пристроїв, використовувати цифровий контент і здійснювати безконтактні платіжні транзакції. Основна робота NFC полягає в розширенні

технології "безконтактних" карт. Він працює в межах 4 см (між пристроями), дозволяючи пристроям обмінюватися інформацією. Приклади використання NFC представлено на рисунку 1.7.

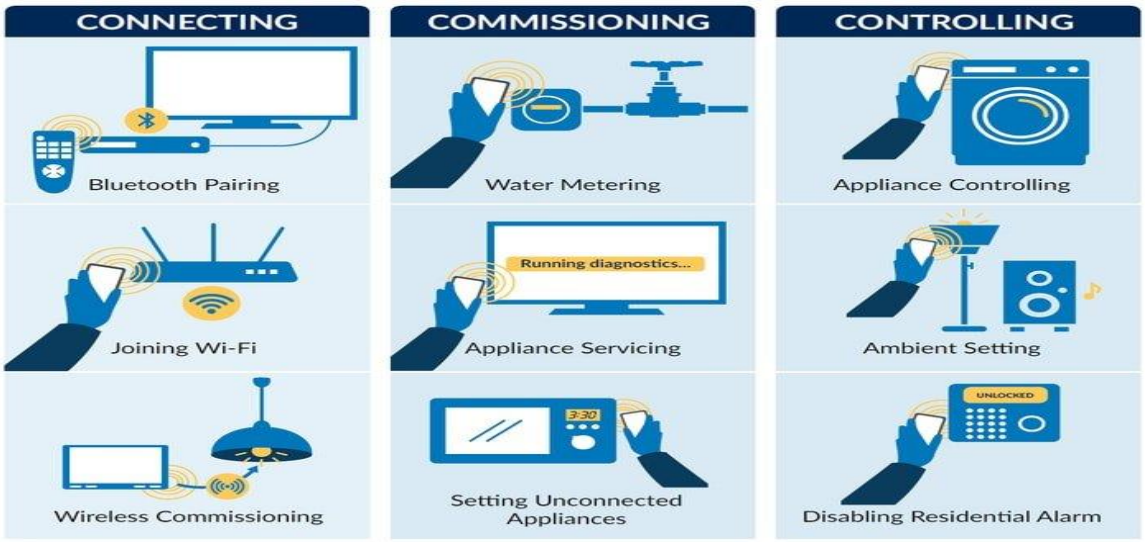

Рисунок 1.7 – Приклади використання NFC

7. Sigfox

Sigfox відомий як одна з найкращих альтернативних технологій, які несуть в собі атрибути як стільникового зв'язку, так і Wi-Fi. Як сьогодні протокол Інтернету речей розроблений і призначений для M2M додатків, він може тільки відправляти дані на низькому рівні. За допомогою UNB або ультра вузького діапазону Sigfox може підтримувати швидкість передачі низьких даних від 10 до 1000 біт в секунду. Він споживає тільки 50 мікроватт потужності. Схема роботи Sigfox представлена на рисунку 1.8.

Рисунок 1.8 – Характеристика Sigfox

1.5 Постановка мети та задач

Метою дипломної роботи є розробити веб-сайт “Система управління службами міста Smartcity”.

Задачі дослідження:

1) створити наступні сервіси для серверної та клієнтської частин аплікації:

- “Менеджмент користувачів”, що дозволяє адміністратору редагувати власні користувацькі дані, переглядати список зареєстрованих користувачів системи, деактивувати та активувати нового користувача, та при необхідності змінювати його набір ролей;

- “Менеджмент задач системи”, що дозволяє редагувати опис доступних задач в системі, який потім відображається користувачам в з відповідною роллю до певної організації;

- “Коментарі”, що дозволяє коментувати ті чи інші завдання відповідно до потреб;

- “Бюджет” та “Транзакції”, що дозволяють проводити операції з бюджетом організацій та переглядати історію платежів, при необхідності відміняти деякі з них.

2) виконати тестування системи “Розумне місто”;

3) створити бізнес-план розробки програмного продукту з наведеними маркетинговими дослідженнями ринку збуту та показниками економічної ефективності реалізованих в роботі інженерно-технічних та інших заходів.

Об’єктом дослідження є інформаційна модель системи “Smartcity”: складові – технології – безпека.

Предметом дослідження є інформаційні технології керування організаціями як складовими розумного міста і забезпечення безпеки при взаємодії між компонентами системи.

Методи дослідження: системний аналіз; методи проектування програмного забезпечення; методи шифрування та автентифікації даних; методи тестування розробленого програмного забезпечення.

1.6 Висновки до розділу

Розглянуто сутність поняття “Smart City” у контексті створення інформаційно-програмного забезпечення захищеного обміну даними в рамках вибраних комунікаційних технологій та запропоновано багаторівневу модель безпечного функціонування розумного міста. Проаналізовано основні складові розумного міста за багаторівневою моделлю.

2 ОБҐРУНТУВАННЯ ВИБОРУ МОДЕЛЕЙ ТА СТВОРЕННЯ ТАБЛИЦЬ БАЗИ ДАНИХ

2.1 Вибір методики і способу зберігання даних

Дані - це один з найцінніших ресурсів, якими володіє сучасний бізнес. Реляційні системи управління базами даних (СУБД) дуже добре підходять для зберігання і запиту структурованих реляційних даних, хоча деякі з них також підтримують зберігання неструктурованих даних і декількох типів сховищ. Вони зазвичай характеризуються як системи ЦС, які тому жертвують допуском до поділу і часто реалізуються як один сервер, що вимагає дуже дорогого і надійного комп'ютерного обладнання для масштабування. Призначення системи керування базою даних представлено на рисунку 2.1.

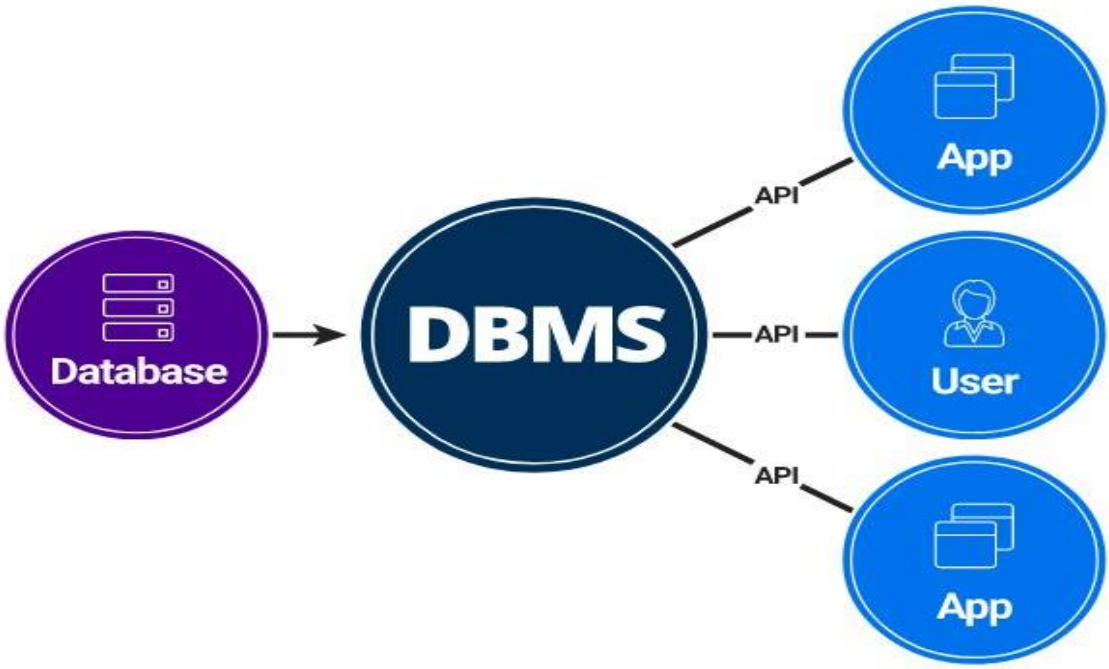

Рисунок 2.1 – Складові системи DBMS

Реляційні дані означають, що дані, що зберігаються в різних частинах (тобто таблицях) бази даних, часто пов'язані один з одним і зазвичай знаходяться

у відносинах "один до багатьох" або "багато до багатьох". Кожна таблиця (або відношення) складається з рядків (записів) і стовпців (полів або атрибутів) з унікальним ідентифікатором (ключем) для кожного рядка.

Як уже згадувалося раніше, дані зазвичай представляються в програмних додатках у вигляді доменних об'єктів. Ці об'єкти даних (або сутності) не представлені і не сформовані таким же чином в програмному забезпеченні, як в реляційній базі даних, хоча загальний шаблон полягає в поданні одного доменного об'єкта (або сутності) у вигляді запису рядка в одній конкретній таблиці сутностей (наприклад, активного запису).

Дані часто потребують зберігання та доступу через кілька таблиць через їх реляційну природу, і це створює невідповідність між додатками та поданнями даних бази даних, яку зазвичай називають невідповідністю об'єктно-реляційного імпедансу.

Ця невідповідність виникає більш конкретно через необхідність зіставлення програмних об'єктів з таблицями бази даних, створеними на основі реляційних схем. Щоб усунути цю невідповідність, були створені різні архітектурні шаблони і програмні додатки для зіставлення програмних об'єктів з таблицями бази даних і навпаки. Це називається об'єктно-реляційним відображенням (ORM).

СУБД, як правило, дуже зосереджені на забезпеченні сильної узгодженості та кислотних транзакцій (хоча і не завжди), величезної функціональності, стабільності та сильних гарантіях надійності та надійності. Вони також дуже добре підходять для складних запитів і додатків аналітики не в реальному часі.

Потенційні недоліки включають відсутність гнучкості моделювання даних, масштабованість, доступність, пропускну здатність, продуктивність і жорсткість схеми.

Жорсткість схеми відноситься до труднощів зміни як схем рівня бази даних, так і схем моделі даних (наприклад, таблиці) після їх використання у виробництві. Ці схеми рідко бувають правильно і повністю визначені заздалегідь, і часто вимагають змін з плином часу. Ці зміни можуть призвести до

дуже трудомістких і складних міграцій даних, а також до великого ризику помилок і, отже, до підвищення вимог до якості і тестування.

Нарешті, системи СУБД можуть бути досить складними для встановлення, управління, використання всіх переваг (наприклад, складних запитів, збережених процедур, тригерів тощо) та оптимізації. В результаті над цими системами традиційно працювали вузькоспеціалізовані люди з такими ролями, як Адміністратор баз даних (DBA), і в меншій мірі інженери-програмісти.

Одним з найкращих рішень для зберігання даних виявилась СУБД MySQL. MySQL - одна з найпопулярніших баз даних у світі (рис. 2.2). Вона є відкритим вихідним кодом, надійним, сумісним з усіма основними хостинг - провайдерами, економічно ефективним і простим в управлінні. Багато організацій використовують безпеку даних і потужну транзакційну підтримку, пропоновану MySQL, для забезпечення безпеки онлайн-транзакцій і поліпшення взаємодії з клієнтами. Однак підприємства, що використовують MySQL, стикаються з низкою проблем, коли їх Додатки відчувають експоненціальне зростання і їм потрібен додатковий масштаб.

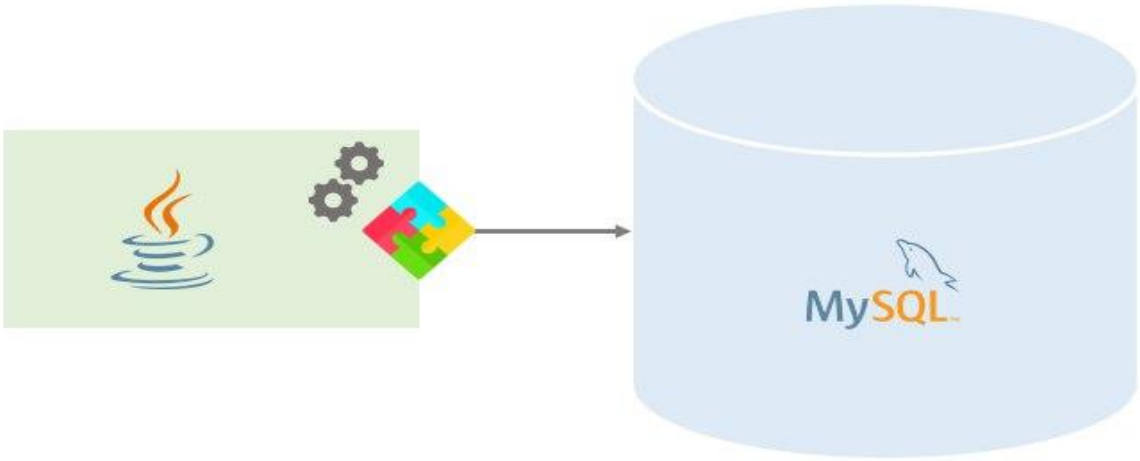

Рисунок 2.2 – Java та MySQL - поширена комбінація

Поряд з розумінням того, чому MySQL є ідеальним рішенням для швидкозростаючих середовищ, не менш важливо розуміти проблеми, які можуть

завдати шкоди бізнес-операціям(транзакціям). Ось 5 основних причин для використання MySQL у проектах типу Smart City:

Безпечні Грошові Операції

Транзакції MySQL працюють як єдине ціле, що означає, що до тих пір, поки кожен окремий операційний етап не буде успішно завершений, транзакція не буде очищена. Таким чином, якщо операція завершується невдачею на будь-якому етапі, то вся транзакція, що відбувається в цій групі, завершується невдачею. Етапи транзакції описано на рисунку 2.3.

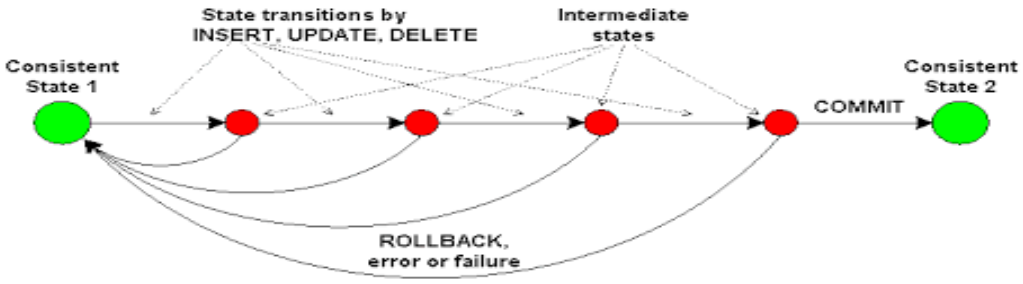

Рисунок 2.3 – Процес створення транзакції

MySQL гарантує, що фінансові транзакції мають цілісність даних, тому клієнти можуть здійснювати безтурботні транзакції онлайн. Гроші не списуються до тих пір, поки весь процес не буде завершений, і в разі невдачі кожен процес повертається на попередню стадію.

Масштабованість

MySQL поставляється з перевагою неперевершеної гнучкості, яка полегшує ефективне управління глибоко впровадженими додатками, навіть у гігантських центрах обробки даних, які складають величезну кількість критично важливої інформації. Це дозволяє повністю налаштувати його для задоволення

унікальних вимог компаній електронної комерції з набагато меншим обсягом займаної площі. MySQL забезпечує максимальну гнучкість платформи для підприємств, які потребують додаткових функцій і функцій для своїх серверів баз даних.

Доступність

Послідовна доступність - це стійка особливість MySQL - підприємства, які розгортають його, можуть насолоджуватися цілодобовим часом безвідмовної роботи. MySQL поставляється з широким спектром кластерних серверів і конфігурацій реплікації master - slave, які забезпечують миттєву відпрацювання відмови для безперебійного доступу. Незалежно від того, чи використовуєте ви Веб-сайт електронної комерції або високошвидкісну систему обробки, MySQL призначений для обробки мільйонів запитів і тисяч транзакцій, забезпечуючи при цьому унікальні кеші пам'яті, повнотекстові індекси і оптимальну швидкість.

Гарантована надійність

Захист конфіденційної ділової інформації є головною турботою кожного підприємства. MySQL забезпечує безпеку даних з винятковими функціями захисту даних. Потужне шифрування даних запобігає несанкціонованому перегляду даних, а підтримка SSH та SSL забезпечує безпечніші з'єднання. Він також має потужний механізм, який обмежує доступ до сервера для авторизованих користувачів і має можливість блокувати користувачів навіть на рівні людина-машина. Нарешті, функція резервного копіювання даних полегшує відновлення в певний момент часу.

Можливість Швидкого Запуску

MySQL працює виключно швидко, незалежно від базової платформи. Він має можливості самостійного управління, такі як автоматичний перезапуск, розширення простору і автоматична зміна конфігурації для зручності управління. Він також поставляється з повним набором інструментів міграції та повністю завантаженим графічним пакетом керування. MySQL забезпечує моніторинг продуктивності в режимі реального часу для своєчасного усунення операційних проблем з однієї робочої станції.

З усіх цих причин було прийнято рішення використовувати MySQL для миттєвої розробки та запуску додатків. Багато галузей користуються економічною ефективністю, ефективністю та надійністю MySQL для надання бездоганних послуг та збільшення своїх доходів.

2.2 Загальна архітектура аплікації

Архітектура програмного забезпечення відноситься до високорівневих структур програмної системи. Найбільш успішними є дворівнева і трирівнева архітектурні структури.

2.2.1 Дво- та трирівнева архітектурні підходи

Дворівнева система заснована на архітектурі клієнт-сервер. Дворівнева архітектура подібна клієнт-серверному додатку. Прямий зв'язок відбувається між клієнтом і сервером. Немає ніякої проміжної ланки між клієнтом і сервером. Через щільне зчеплення 2-рівневий додаток буде працювати швидше. На наведеному нижче рисунку показана архітектура дворівневої системи. Тут відбувається прямий зв'язок між клієнтом і сервером, немає проміжного рівня. Дворівнева архітектура ділиться на дві частини:

– клієнтський додаток (рівень клієнта);

– база даних (рівень даних).

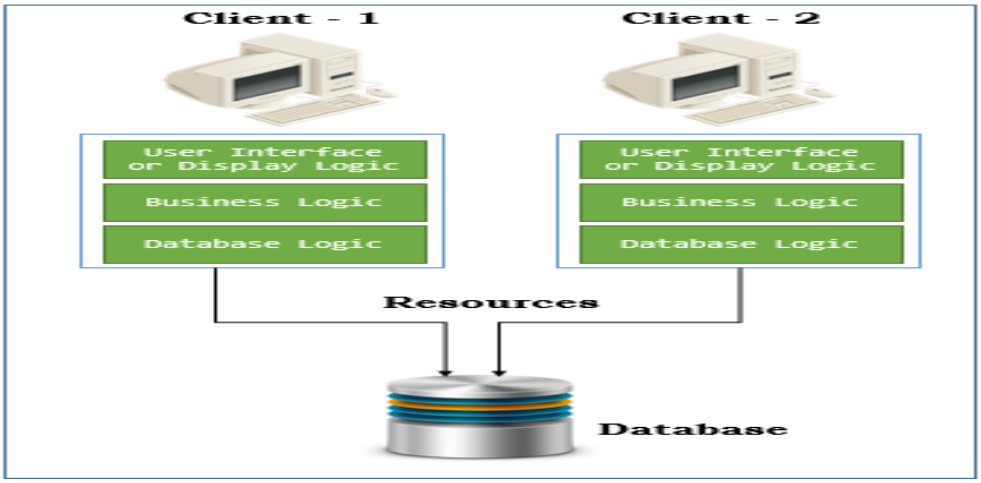

Рисунок 2.4 – Приклад розмежування шарів архітектури

На стороні клієнтського додатка написаний код для збереження даних на сервері баз даних. Клієнт відправляє запит на сервер, і він обробляє запит і відправляє назад з даними. Основна проблема дворівневої архітектури полягає в тому, що сервер не може відповісти на кілька запитів одночасно, в результаті чого виникає проблема цілісності даних. Коли розробники не дисципліновані, логіка відображення, бізнес-логіка і логіка бази даних заплутуються і / або дублюються в системі 2-го рівня клієнт-сервер.

Головними перевагами даного підходу є простота в обслуговуванні та підтримці продукту, швидкість комунікації між сервером та клієнтом. Але і недоліків також достатньо: низька продуктивність при великій кількості клієнтів, економічна невигода. Тому варто переглянути більш серйозний підхід у вигляді 3-х рівневої структури.

Для даної аплікації було прийнято рішення використовувати багаторівневу архітектуру. 3-х рівнева архітектура - це тип архітектури програмного забезпечення, який складається з трьох" рівнів "або" шарів " логічних обчислень. Вони часто використовуються в додатках як особливий тип клієнт-серверної системи. 3-рівневі архітектури забезпечують безліч переваг для виробничих середовищ і середовищ розробки, модулюючи користувальницький інтерфейс, бізнес-логіку і рівні зберігання даних. Це дає більшу гнучкість командам

розробників, дозволяючи їм оновлювати певну частину програми незалежно від інших частин. Ця додаткова гнучкість може поліпшити загальний час виходу на ринок і скоротити час циклу розробки, надаючи командам розробників можливість замінювати або модернізувати незалежні рівні, не зачіпаючи інші частини системи. Приклад 3-ьох рівневої архітектура аплікації представлена на рисунку 2.5.

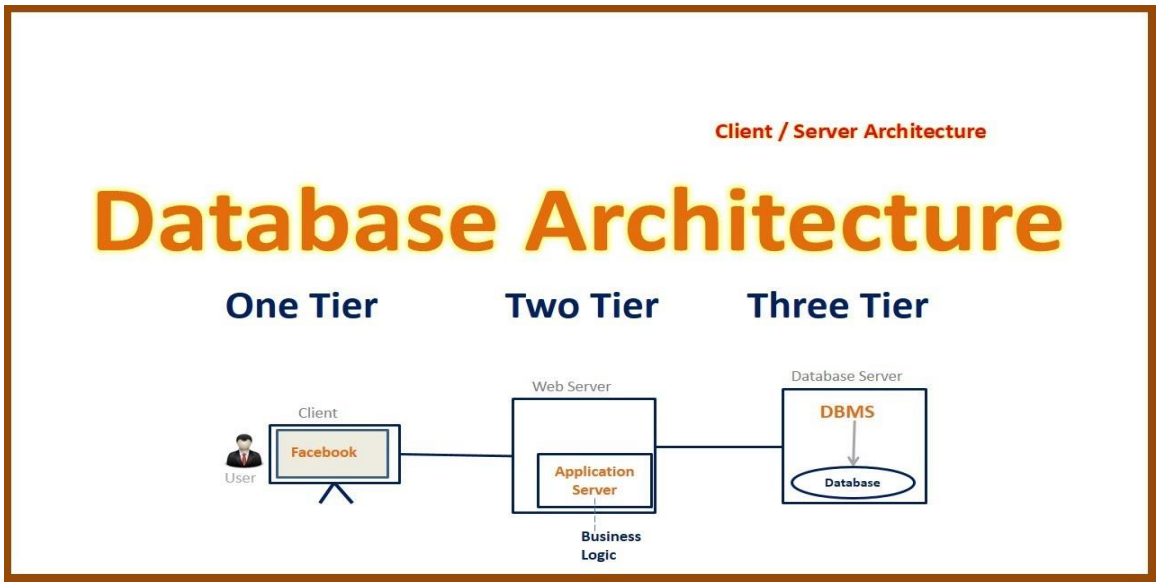

Рисунок 2.5 – Приклад архітектури аплікації з використанням бази даних

Наприклад, користувальницький інтерфейс веб-програми може бути перероблений або модернізований без шкоди для функціональної бізнес-логіки і логіки доступу до даних. Ця архітектурна система часто ідеально підходить для вбудовування та інтеграції стороннього програмного забезпечення в існуючий додаток. Ця гнучкість інтеграції також робить його ідеальним для вбудовування аналітичного програмного забезпечення в уже існуючі програми і часто використовується постачальниками вбудованої аналітики з цієї причини. 3-рівневі архітектури часто використовуються в хмарних або локальних додатках, а також в додатках software-as-a-service (SaaS). Приклад різних рівнів аплікації та приклади мов на яких вони написано зображено на рисунку 2.6.

Рисунок 2.6 – Рівні аплікації

Архітектура складається з 3 конкретно визначених рівнів:

– рівень презентації є найближчим до юзера шаром в 3-рівневій системі і складається з призначеного для нього інтерфейсом. Цей користувальницький інтерфейс часто є графічним, доступним через веб-браузер або веб-додаток і відображає контент і інформацію, корисну для кінцевого користувача. Цей рівень часто будується на основі веб-технологій, таких як HTML5, JavaScript, CSS або інших популярних фреймворків веб-розробки, і взаємодіє з іншими шарами через виклики API;

– серверний рівень аплікації містить функціональну бізнес-логіку, яка керує основними можливостями програми. Він часто пишеться на Java, Net, C#, Python, C++ тощо;

– рівень доступу до даних складається з бази даних/системи зберігання даних і рівня доступу до даних. Прикладами таких систем є MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB та ін.

Переваг такого підходу до створення аплікації значно більше. Почати варто з високої продуктивності такої системи. Розділяючи різні шари, створюється можливість масштабувати кожен з них незалежно в залежності від потреби в будь-який момент часу. Наприклад, якщо ви отримуєте багато веб-запитів, але не так багато запитів, які впливають на рівень програми, ви можете масштабувати свої веб-сервери, не торкаючись серверів додатків. Аналогічно, якщо ви отримуєте багато великих запитів додатків тільки від декількох веб-

користувачів, ви можете масштабувати свої додатки і шари даних, щоб задовольнити ці запити, не торкаючись ваших веб-серверів. Це дозволяє балансувати навантаження на кожен шар незалежно, покращуючи загальну продуктивність з мінімальними ресурсами. Крім того, незалежність, створювана в результаті модульної обробки різних рівнів, дає вам безліч варіантів розгортання. Наприклад, ви можете вибрати розміщення ваших веб-серверів у публічній або приватній хмарі, тоді як ваші програми та шари даних можуть розміщуватися на місці. Крім того, ви можете розмістити свої шари додатків і даних в хмарі, в той час як Ваші веб-сервери можуть бути розміщені локально або в будь-якій їх комбінації.

Маючи розрізнені шари, можна підвищити надійність і доступність, розмістивши різні частини програми на різних серверах і використовувати кешовані результати. На рисунку 2.7 представлена архітектура аплікації Розумне місто.

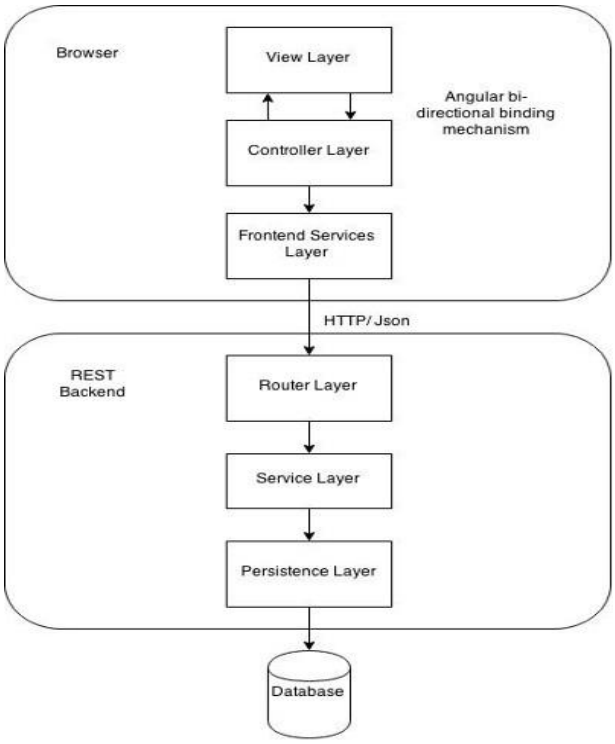

Рисунок 2.7 – Архітектура аплікації Smart city

2.2.2 Принципи якісно побудованої аплікації

При побудові комплексних систем з призначенням масштабу управління організаціями рівня великих міст надзвичайно важливе значення представляють правильно вибрані принципи та практики побудови якісних архітектур програмного забезпечення та їх слідування під час розробки та вирішення проблем, які виникають при розробці функціоналу системи.

При розробленні архітектури програмного забезпечення інформаційної системи «розумне місто» було використано наступні кращі практики розроблення ПЗ, які дозволяють створювати кращу архітектуру:

The Single Responsibility Principle

A module should be responsible to one, and only one, actor.

Часто її трактували наступним чином: Модуль повинен мати тільки один обов'язок. І це головна помилка при знайомстві з принципами. Все дещо хитріше.

На кожному проекті люди грають різні ролі: аналітик, дизайнер, адміністратор баз даних. Природно, одна людина може грати відразу кілька ролей. У цьому принципі мова йде про те, що зміни в модулі може запитувати одна і тільки одна роль. Наприклад, є модуль, що реалізує якусь бізнес-логіку, запросити зміни в цьому модулі може тільки аналітик, але ніяк не адміністратор баз даних або дизайнер.

OCP: The Open Closed Principle

A software artifact should be open for extension but closed for modification.

Щоб розширити поведінку, потрібно скористатися динамічним поліморфізмом. Наприклад, наша програма повинна надсилати сповіщення. Використовуючи dependency inversion, наш модуль оголошує тільки інтерфейс відправки повідомлень, але не реалізацію. Таким чином, логіка нашого застосування міститься в одному dll файлі, а клас відправки повідомлень, що реалізує інтерфейс - в іншому. Таким чином, ми можемо без зміни

(перекомпіляції) модуля з логікою використовувати різні способи відправки повідомлень. Цей принцип тісно пов'язаний з LSP і DIP.

ISP: The Interface Segregation Principle

Make fine grained interfaces that are client specific.

Під інтерфейсом тут розуміється саме Java, C # інтерфейс. Розділення інтерфейсу полегшує використання та тестування модулів.

DIP: The Dependency Inversion Principle

Depend on abstractions, not on concretions.

Модулі верхніх рівнів не повинні залежати від модулів нижніх рівнів. Обидва типи модулів повинні залежати від абстракцій.

Абстракції не повинні залежати від деталей. Деталі повинні залежати від абстракцій.

Що таке модулі верхніх рівнів? Як визначити цей рівень? Як виявилося, все дуже просто. Чим ближче модуль до введення / виведення, тим нижче рівень модуля. Тобто модулі, що працюють з BD, інтерфейсом користувача, низького рівня. А модулі, що реалізують бізнес-логіку-високого рівня.

Що таке залежність модулів? Це посилання на модуль у вихідному коді, тобто import, require і т.д. за допомогою динамічного поліморфізму в runtime можна звернути цю залежність.

Є модуль Logic, що реалізує логіку, який повинен відсилати повідомлення. У цьому ж пакеті оголошується інтерфейс ISender, який використовується Logic. Рівнем нижче, в іншому пакеті оголошується ConcreteSender, що реалізує ISender. Виходить, що в момент компіляції Logic не залежить від ConcreteSender. У runtime, наприклад, через конструктор в Logic встановлюється екземпляр ConcreteSender [11].

Окремо варто відзначити часте питання " навіщо плодити абстракції, якщо ми не збираємося замінювати базу даних?". Логіка тут наступна. На старті проекту, ми знаємо, що будемо використовувати реляційну базу даних, і це точно буде MySQL. Ми навіть не плануємо її змінювати в майбутньому. Але ми хочемо відкласти прийняття рішень про те, яка буде схема таблиць, які будуть індекси, і

т.д. до моменту, поки це не стане проблемою. Також ми можемо раніше налагодити логіку нашого застосування, реалізувати інтерфейс, зібрати зворотний зв'язок від замовника, і мінімізувати наступні зміни, адже багато реалізовано тільки у вигляді заглушок.

2.2.3 Формат передачі даних

Лідируючими форматами передачі даних є XML та JSON.

XML - це мова розмітки, створена консорціумом World Wide Web Consortium (W3C) для визначення синтаксису кодування документів, які можуть читати як люди, так і машини. Він робить це за допомогою тегів, які визначають структуру документа, а також те, як документ повинен зберігатися і транспортуватися. Ймовірно, найпростіше порівняти його з іншою мовою розмітки, мовою гіпертекстової розмітки (HTML), використовуваним для кодування веб-сторінок. HTML використовує заздалегідь визначений набір символів розмітки (коротких кодів), які описують формат вмісту веб-сторінки [12]. Приклад XML документу продемонстровано на рисунку 2.8.

```
<?xml version="1.0" encoding="iso-8859-8" standalone="yes" ?>
<CURRENCIES>
  <LAST_UPDATE>2004-07-29</LAST_UPDATE>
  <CURRENCY>
    <NAME>dollar</NAME>
    <UNIT>1</UNIT>
    <CURRENCYCODE>USD</CURRENCYCODE>
    <COUNTRY>USA</COUNTRY>
    <RATE>4.527</RATE>
    <CHANGE>0.044</CHANGE>
  </CURRENCY>
  <CURRENCY>
    <NAME>euro</NAME>
    <UNIT>1</UNIT>
    <CURRENCYCODE>EUR</CURRENCYCODE>
    <COUNTRY>European Monetary Union</COUNTRY>
    <RATE>5.4417</RATE>
    <CHANGE>-0.013</CHANGE>
  </CURRENCY>
</CURRENCIES>
```

Рисунок 2.8 – Приклад XML формату

Головними перевагами по відношенню до інших форматів є:

– XML не залежить від платформи і мови програмування, тому він може бути використаний в будь-якій системі і підтримує зміну технології, коли це відбувається;

– XML підтримує unicode. Unicode - це міжнародний стандарт кодування, призначений для використання з різними мовами і сценаріями, відповідно до якого кожній букві, цифрі або символу присвоюється унікальне Числове значення, яке застосовується на різних платформах і програмах. Ця функція дозволяє XML передавати будь-яку інформацію, написану на будь-якій людській мові;

– дані, що зберігаються і транспортуються за допомогою XML, можуть бути змінені в будь-який момент часу, не впливаючи на представлення даних. Як правило, для представлення даних використовується інша мова розмітки, такий як HTML, HTML отримує дані з XML і відображає їх на GUI (графічний користувальницький інтерфейс), як тільки дані оновлюються в XML, вони відображаються в HTML без внесення будь-яких змін в HTML GUI;

– XML дозволяє проводити валідацію з використанням DTD і схеми. Ця перевірка гарантує, що XML - документ не містить будь-яких синтаксичних помилок;

– XML спрощує обмін даними між різними системами завдяки своїй незалежній від платформи природі. XML - дані не вимагають ніякого перетворення при передачі між різними системами.

Але і недоліки є досить суттєві:

– синтаксис XML є детальним і надмірним у порівнянні з іншими текстовими форматами передачі даних, такими як JSON;

– надмірність синтаксису XML призводить до більш високої вартості зберігання і транспортування при великому обсязі даних;

– XML-документ менш читабельний порівняно з іншими текстовими форматами передачі даних, такими як JSON;

– XML не підтримує масив;

– розміри XML-файлу зазвичай дуже великі через його багатослівного характеру, він повністю залежить від того, хто його пише.

Більш сучасним вважається формат JSON.

JavaScript Object Notation(JSON) - це стандартний текстовий формат для представлення структурованих даних, заснований на синтаксисі об'єктів JavaScript. Він зазвичай використовується для передачі даних у веб-додатках (наприклад, відправка деяких даних з сервера на клієнт, так що вони можуть бути відображені на веб-сторінці, або навпаки). З появою сайтів з підтримкою AJAX стає все більш важливим, щоб сайти могли завантажувати дані швидко і асинхронно або у фоновому режимі, не затримуючи рендеринг сторінок. Перемикання вмісту певного елемента в наших макетах без необхідності оновлення сторінки додає " вау " фактор в наші програми, не кажучи вже про додаткову зручність для наших користувачів. Через популярність і простоту соціальних мереж багато сайтів покладаються на контент, наданий такими сайтами, як Twitter, Flickr та іншими [13].

Перевагами JSON є:

– JSON синтаксис дуже простий у використанні. Ми повинні використовувати тільки синтаксис, який забезпечує нам легкий розбір даних і більш швидке виконання даних. Оскільки його синтаксис дуже малий і легкий, саме тому він виконує відповідь швидше.

– парсинг на стороні сервера є важливою частиною, яку хочуть розробники, якщо парсинг буде швидким на стороні сервера, то тільки користувач може отримати швидку відповідь своєї відповіді, тому в цьому випадку парсинг на стороні сервера JSON є сильною стороною, яка вказує нам на використання JSON на стороні сервера.

– має широкий спектр підтримуваних браузерів сумісності з операційними системами, так що додатки, зроблені з кодуванням JSON не вимагає особливих зусиль, щоб зробити його все сумісним з браузером. Під час розробки розробник думає для різних різних браузерів, але JSON надає цю функціональність.

– інструмент для обміну даними - JSON є найкращим інструментом для обміну даними будь-якого розміру, навіть аудіо, відео і т.д. це відбувається тому, що JSON зберігає дані в масивах, тому передача даних стає простіше. З цієї причини JSON є чудовим форматом файлів для веб-API і для веб-розробки.

Приклад XML документу продемонстровано на рисунку 2.9.

Тому було прийнято рішення використовувати саме такий формат обміном даних.

```
{
  "orderID": 12345,
  "shopperName": "Ivan Ivanov",
  "shopperEmail": "ivanov@example.com",
  "contents": [
    {
      "productID": 34,
      "productName": "Super product",
      "quantity": 1
    },
    {
      "productID": 56,
      "productName": "Wonderful product",
      "quantity": 3
    }
  ],
  "orderCompleted": true
}
```

Рисунок 2.9 – Приклад JSON формату

2.3 UML - методологія розроблення програмного забезпечення

Для того щоб в майбутньому продемонструвати діаграму взаємодії користувачів з системою ознайомимось з поняттям UML. Приклад представлено на рисунку 2.10.

UML, скорочено від Unified Modeling Language, - це стандартизована мова моделювання, що складається з інтегрованого набору діаграм, розроблених для допомоги розробникам систем і програмного забезпечення у визначенні,

візуалізації, побудові та документуванні артефактів програмних систем, а також для бізнес-моделювання та інших не програмних систем. UML являє собою сукупність кращих інженерних практик, які довели свою успішність при моделюванні великих і складних систем. UML - це дуже важлива частина розробки об'єктно-орієнтованого програмного забезпечення та процесу розробки програмного забезпечення. В UML використовуються, в основному, графічних позначень, щоб висловити розробки програмних проектів. Складні програми вимагають спільної роботи і планування з боку декількох команд і, отже, вимагають чіткого і лаконічного способу спілкування між ними. Бізнесмени не розуміють коду. Таким чином, UML стає необхідним для зв'язку з несуттєвими вимогами, функціональними можливостями і процесами системи [14].

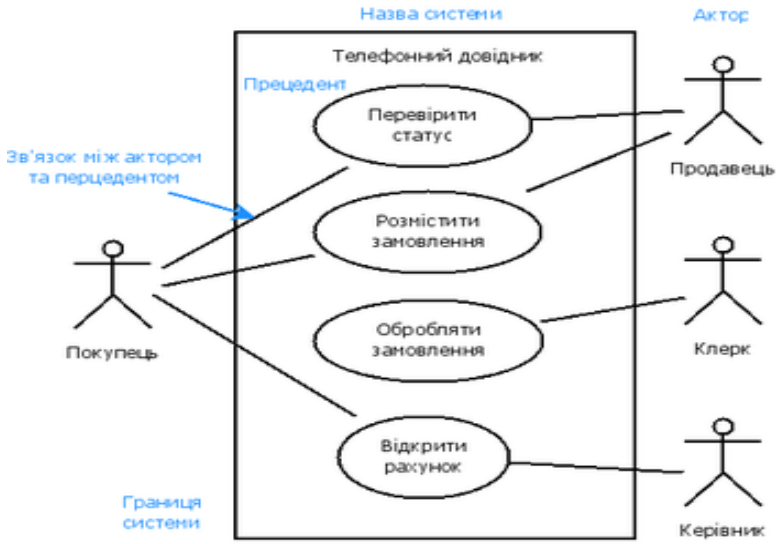

Рисунок 2.10 – Приклад UML діаграми

Коли команди можуть візуалізувати процеси, взаємодії користувачів і статичну структуру системи, вони економлять багато часу. UML пов'язаний з об'єктно-орієнтованим проектуванням і аналізом. UML використовує елементи і формує зв'язки між ними для формування діаграм. Використання UML допомагає проектним командам спілкуватися, досліджувати потенційні проекти та перевіряти архітектурний дизайн програмного забезпечення. Найважливішою складовою UML є графічна нотація (рис. 2.11), що дозволяє специфікувати,

проектувати, візуалізувати і документувати артефакти об’єктно-орієнтованого програмного забезпечення.

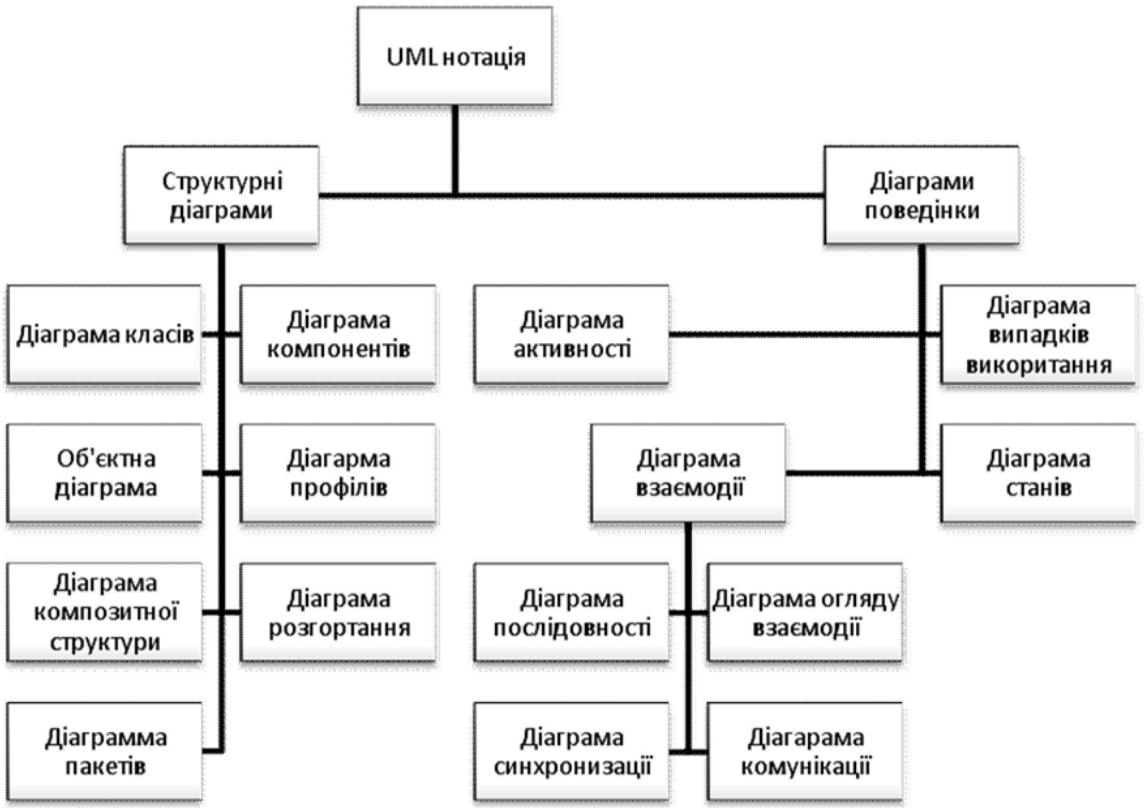

Рисунок 2.11 - Структура графічної нотації мови UML

2.4 Опис моделей користувачів. Опис ролей. Створення таблиць

Основним елементом БД є таблиця. В реляційній БД, поняття «таблиця» є нестроге і зазвичай означає не «відношення» як абстрактне поняття, а візуальне представлення відношення на папері чи екрані.

Згідно поставленого завдання спроектована програмна система пропонує функціонал для трьох ролей – звичайного користувача, адміністратора системи, супервайзора. Кожен з них відрізняється своїми привілеями та обмеженнями. UML діаграма на рисунку 2.12 допоможе нам зрозуміти які дії кожен з користувачів буде в змозі виконати та якими привілеями мусить володіти.

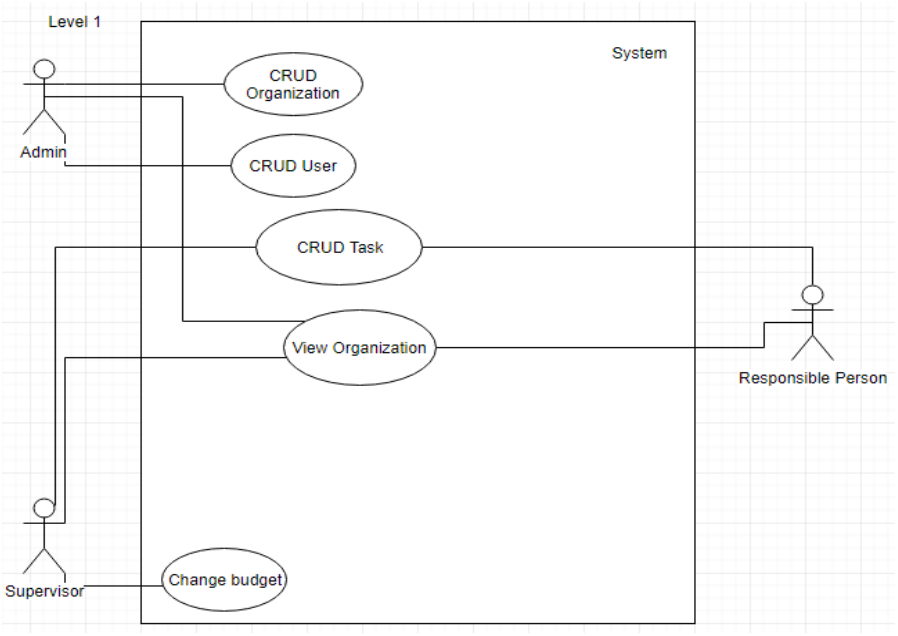

Рисунок 2.12 UML діаграма користувачів аплікації

З цієї діаграми випливає список функції ролей користувачів системи:

1) ADMIN:

– може додавати організації;

– може змінювати назву організації;

– може видаляти організації;

– може створювати облікові записи іншим користувачам;

– може редагувати облікові записи користувачів;

– може видаляти облікові записи;

– може надавати ролі Supervisor або Admin іншим користувачам;

– може призначати користувача як відповідального(Responsible person) за деяку службу.

2) SUPERVISOR:

– може створювати задачі для різних служб;

– може редагувати вже існуючі задачі, зокрема міняти бюджет і термін виконання.(Можливо і міняти і суть задачі);

– бачить задачі всіх служб та їх стани;

– може змінювати бюджет.

3) RESPONSIBLE PERSON:

– змінює стани задач (очікує виконання, виконується, виконано) (в межах служби, за яку він відповідальний);

– може переглядати задачі служби;

– може додавати задачі (в межах служби, за яку він відповідальний).

На рисунку 2.13 зображено таблицю "USERS" (користувач), що містить усі поля таблиці. Така кількість полів дозволяє зберігати інформацію, що однозначно визначає потрібну інформацію про користувача.

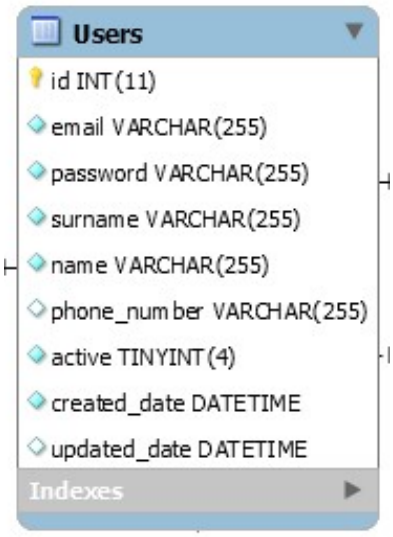

Рисунок 2.13 – Таблиця користувачі

Інформація по стовпцям таблиці:

– ID – оригінальний номер запису в таблиці, який однозначно ідентифікує певного юзера. Всі таблиці в БД мають такий стовпець, несучи однакове ідейне навантаження. У всіх випадках полю призначене властивість первинного ключа. Даючи кожній базі даних стовпець "id", ми замінюємо крихітну частину операційних накладних витрат на майбутнє, в якому нам буде легше вносити зміни, які, як ми знали, ніколи не знадобляться;

– EMAIL – електронна пошта;

– PASSWORD – зашифрований пароль користувача;

– NAME, SURNAME та PHONE_NUMBER – прізвище, ім’я та мобільний номер телефону для ідентифікації;

– ACTIVE – зберігає інформацію про те чи користувач досі активний(працює);

– CREATED_DATE та UPDATED_DATE – потрібні для визначення дати реєстрації та останньої зміни особистої інформації в системі.

Також деякі поля цієї таблиці виступають як вторинні ключі для інших другорядних моделей таких як “USERS_ORGANIZATIONS”, “SEENCOMMENTS” та “USERS_ROLES”.

Таблиця **USERS_ORGANIZATIONS** (рис. 2.14) являє собою пару юзер - організація, тобто конкретну сутність яка означає що користувач входить до певної організації.

Рисунок 2.14 – Таблиця організації та користувачі

Таблиця **USERS_ROLES** ,що представлена на рисунку 2.15, дуже схожа на попередню, різниця лише в тому що тут сутність є парою для юзер - роль.

Рисунок 2.15 – Таблиця ролей користувачів

Таблиця **SEENCOMMENTS** визначає пару користувач - прочитане повідомлення. Це потрібно для того щоб зберігати інформацію про те чи прочитав деякий юзер конкретне повідомлення. Представлена на рисунку 2.16.

Рисунок 2.16 – Таблиця переглянутих коментарів

2.5 Опис моделей організацій

Таблиця "Organizations" (рис. 2.17) – одна з основних таблиць, містить дані про ключову одиницю інформації системи, організації, які будуть керуватись відповідальними користувачами. Містить такі поля:

– ID – унікальний ідентифікатор організації;

– NAME – назва організації;

– ADDRESS – адреса організації;

– CREATED_DATE – дата створення;

– UPDATED_DATE – дата останнього редагування.

Таблиця представлена на наступному рисунку.

Рисунок 2.17 – Таблиця організації

2.6 Опис моделей завдань та коментарів

Кожна організація буде містити певний список завдань до виконання, кожне завдання буде мати певний статус, людину яка відповідає за нього та коментарі, певну можливу дискусію, уточнення між керуючою та виконавчою особою. Для того щоб зберігати інформацію про завдання та коментарі відповідальних осіб до нього, було створено такі таблиці як “Tasks” та “Comments”. Таблиця «Завдання» (рис. 2.18) містить такі відомості:

– ID – унікальний ідентифікатор завдання;

– TITLE – заголовок до завдання;

– DESCRIPTION – опис завдання;

– DEADLINE_DATE – дата завершення;

– TASK_STATUS – стан завдання;

– BUDGET – кошти, які виділили під завдання;

– APPROVED_BUDGET – остаточно затверджені кошти на завдання;

– CREATED_DATE – дата створення;

– UPDATED_DATE – дата останнього редагування;

– TASK_STATUS – вторинний ключ таблиці юзер-організація, що означає причетність завдання до конкретної організації.

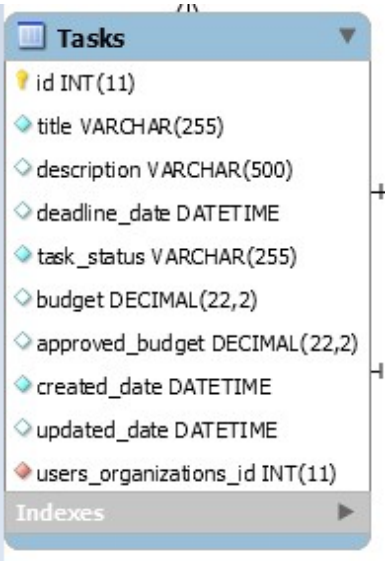

Рисунок 2.18 – Таблиця завдань

Таблиця «Коментарі» (рис. 2.19) містить таку інформацію:

– ID – унікальний ідентифікатор коментаря;

– DESCRIPTION – текст коментаря;

– CREATED_DATE – дата створення;

– UPDATED_DATE – дата останнього редагування;

– USER_ID – вторинний ключ, визначає причетність конкретного користувача до коментаря;

– TASK_ID – також вторинний ключ, визначає причетність коментаря до конкретного завдання.

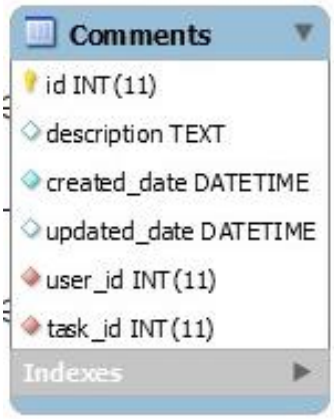

Рисунок 2.19 – Таблиця коментарі

2.7 Опис моделей бюджету та транзакцій. Опис взаємодії

Так як місто має певний бюджет, винесемо його як окрему незалежну сутність. Тож таблиця «Budget» (рис. 2.20) буде містити лише 1 поле – кошти міста.

Рисунок 2.20 – Таблиця бюджет

З бюджету міста будуть виділятись кошти на фінансування певних організацій. Сама транзакція, в контексті даного проекту, є контрактом між містом та організацією. Для того щоб зберігати інформацію про ці події, створено таблицю «Transactions» (рис. 2.21) яка буде містити собі інформацію про транзакції – переведення коштів певній організації. Таблиця містить такі поля:

– ID – унікальний ідентифікатор транзакції;

– TASK_ID – вторинний ключ, визначає причетність транзакції до конкретного завдання організації;

– CURRENT_BUDGET – стан бюджету(кількість грошей) на момент створення транзакції;

– TRANSACTION_BUDGET – конкретна сума грошей на завдання;

– CREATED_DATE – дата проведення транзакції.

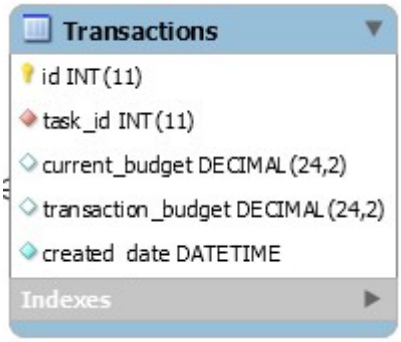

Рисунок 2.21 – Таблиця транзакцій

3 РОЗРОБКА ТА ТЕСТУВАННЯ АПЛІКАЦІЇ

3.1 Розробка серверної частини

Для стабільної та швидкої роботи сервера було вирішено скористатися фреймворком Spring. Spring Framework надає комплексну модель програмування і конфігурації для сучасних корпоративних додатків на базі Java - на будь-якій платформі розгортання.

Ключовим елементом Spring є інфраструктурна підтримка на прикладному рівні: Spring фокусується на "сантехніці" корпоративних додатків, щоб команди могли зосередитися на бізнес - логіці прикладного рівня без непотрібних зв'язків з конкретними середовищами розгортання.

Основна перевага Spring - можливість розробки програми як набору слабозв’язаних компонентів. Чим менше компоненти програми знають один про одного, тим простіше розробляти новий і підтримувати існуючий функціонал програми. Класичний приклад - управління транзакціями. Spring дозволяє керувати транзакціями абсолютно незалежно від основної логіки взаємодії з БД. Зміна цієї логіки не порушить транзакційність, так само як зміна логіки управління транзакціями не зламає логіку програми. Компоненти можна додавати і видаляти (майже) незалежно один від одного. В принципі, додаток можна розробити таким чином, що воно навіть не буде знати, що управляється Spring'ом. Також Spring помітно спрощує модульне тестування (модульного тестування): в компонент, розроблений для роботи в IoC контейнері дуже легко вводити «замокані» залежності і перевірити роботу тільки цього компонента. І тому маємо такі переваги:

– забезпечення слабкої зв'язаності компонентів;

– спрощення ініціалізації і налаштування компонентів;

– спрощення модульного тестування;

– спрощення розробки та підтримки програми в цілому.

Тож, так як Spring був розроблений як альтернатива EJB прямо з самого початку, і пропонує багато можливостей, що полегшують розробку, використання цього фреймворку значно збільшить якість розробки.

3.1.1 Розробка процесу автентифікації

Вхід до аплікації починається з процесів автентифікації та авторизації. Для забезпечення авторизації та входу в систему «Розумне місто» використано технології генерації токенів для користувачів JWT (рис. 3.1) та фреймворк Spring Security.

Дуже важливо розуміти, що використання JWT не приховує і не маскує дані автоматично. Причина, чому JWT використовуються - це перевірка, що відправлені дані були дійсно відправлені авторизованим джерелом. Дані всередині JWT закодовані і підписані, це не одне і теж, що зашифровані. Мета кодування даних - перетворення структури. Підписані дані дозволяють одержувачу даних перевірити автентифікацію джерела даних. Таким чином закодування і підпис даних не захищає їх. З іншого боку, головна мета шифрування - це захист даних від неавторизованого доступу.

Замість запуску процесу автентифікації шляхом перенаправлення на сторінку входу, коли клієнт запитує захищений ресурс, сервер REST автентифікує всі запити, використовуючи дані, доступні в самому запиті, в даному випадку токен JWT. Якщо така аутентифікація не вдається, перенаправлення не має сенсу. REST API просто надсилає HTTP - код 401 (несанкціонований запит), і клієнти повинні знати, що робити; наприклад, браузер покаже динамічний div, щоб дозволити користувачеві ввести ім'я користувача та пароль. З іншого боку, після успішної аутентифікації на класичних багатосторінкових веб-сайтах користувач перенаправляється за

допомогою HTTP-коду 301 (переміщається постійно), як правило, на домашню сторінку або, що ще краще, на сторінку, яку користувач спочатку запросив.

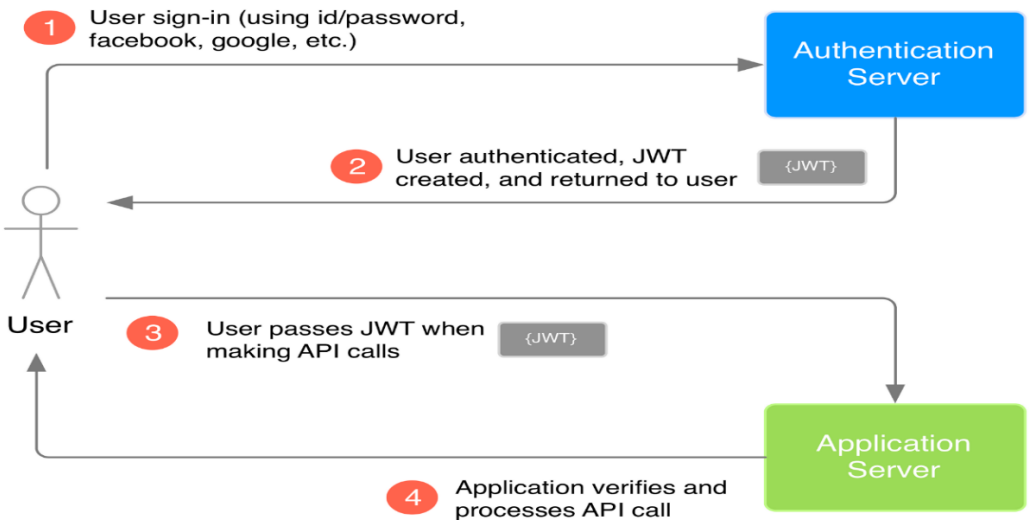

Рисунок 3.1 – Приклад використання JWT

Замість цього ми просто продовжимо виконання запиту, як якщо б ресурс взагалі не був захищений, повернемо http - код 200 (OK) і очікуване тіло відповіді. В системі «Розумне місто» використано симетричний алгоритм SHA-256 для забезпечення хешування JWT токена.

Сторінка автентифікації в систему з отриманим від сервера JWT токеном представлена на рисунку 3.2.

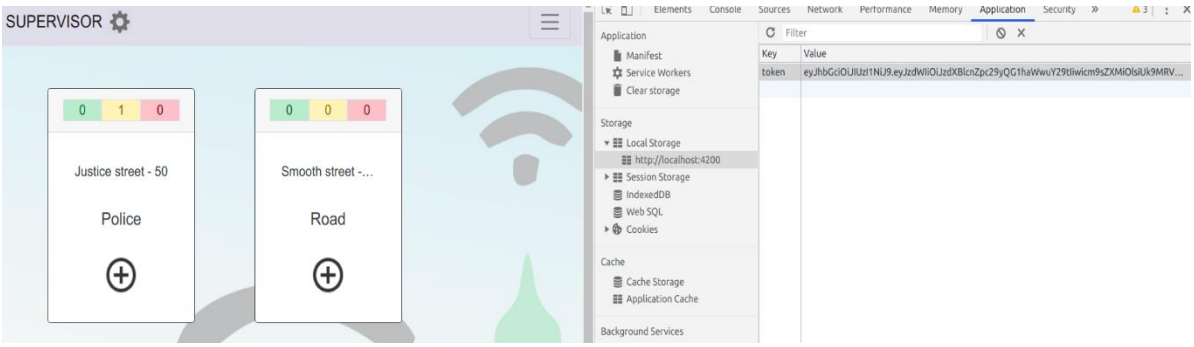

Рисунок 3.2 – Збережений JWT токен

Реалізація JWT автентифікації проведена засобами мови Java. Створено три класи: JWT Token Filter, JWT Token Provider, JWT TokenConfigurer. Вони проілюстровані в наступних рисунках.

```
public class JwtConfigurer extends SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {

    private JwtTokenProvider jwtTokenProvider;

    public JwtConfigurer(JwtTokenProvider jwtTokenProvider) { this.jwtTokenProvider = jwtTokenProvider; }

    @Override
    public void configure(HttpSecurity http) {
        JwtTokenFilter customFilter = new JwtTokenFilter(jwtTokenProvider);
        http.addFilterBefore(customFilter, UsernamePasswordAuthenticationFilter.class);
    }
}
```

Рисунок 3.3 – Клас конфігурації JWT

```
public void doFilter(ServletRequest req, ServletResponse res, FilterChain filterChain)
        throws IOException, ServletException {

    try {
        HttpServletRequest httpServletRequest = (HttpServletRequest) req;
        String token = jwtTokenProvider.resolveToken(httpServletRequest);

        if (token != null && jwtTokenProvider.validateToken(token)) {
            Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
            if (authentication == null) {
                authentication = jwtTokenProvider.getAuthentication(token);
                SecurityContextHolder.getContext().setAuthentication(authentication);
            }
        }
        filterChain.doFilter(req, res);

    } catch (InvalidJwtAuthenticationException exception) {
        ((HttpServletResponse) res).setStatus(HttpServletResponse.SC_UNAUTHORIZED);

        ObjectMapper objectMapper = new ObjectMapper();
        String message = objectMapper.writeValueAsString(ExceptionResponse.builder().url(((HttpServletRequest) req).getRequestURI())
                .message("Token is invalid or expired!").build());
        res.getWriter().write(message);
```

Рисунок 3.4 – Метод фільтрації запиту

```
public String createToken(String username, List<String> roles) {

    Claims claims = Jwts.claims().setSubject(username);
    claims.put("roles", roles);

    Date now = new Date();
    Date validity = new Date(now.getTime() + Long.parseLong(validityInMilliseconds));

    return Jwts.builder()
            .setClaims(claims)
            .setIssuedAt(now)
            .setExpiration(validity)
            .signWith(SignatureAlgorithm.HS256, secretKey)//
            .compact();
}
```

Рисунок 3.5 – Метод генерації токену

```
public boolean validateToken(String token) {
    try {
        Jws<Claims> claims = Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token);

        if (claims.getBody().getExpiration().before(new Date())) {
            return false;
        }

        return true;
    } catch (JwtException | IllegalArgumentException e) {
        throw new InvalidJwtAuthenticationException("Expired or invalid JWT token");
    }
}
```

Рисунок 3.6 – Метод валідації JWT

3.1.2 Розробка DAO та сервісів

Шаблон об'єкта доступу до даних (DAO) - це структурний шаблон, який дозволяє нам ізолювати рівень програми/бізнесу від рівня персистентності (зазвичай це реляційна база даних, але це може бути будь-який інший механізм персистентності) за допомогою абстрактного API.

Функціональність цього API полягає в тому, щоб приховати від програми всі складнощі, пов'язані з виконанням операцій CRUD в базовому механізмі

зберігання. Це дозволяє обом рівням розвиватися окремо, нічого не знаючи один про одного. На рисунку 3.7 продемонстрована архітектура аплікації з використанням DAO рівня.

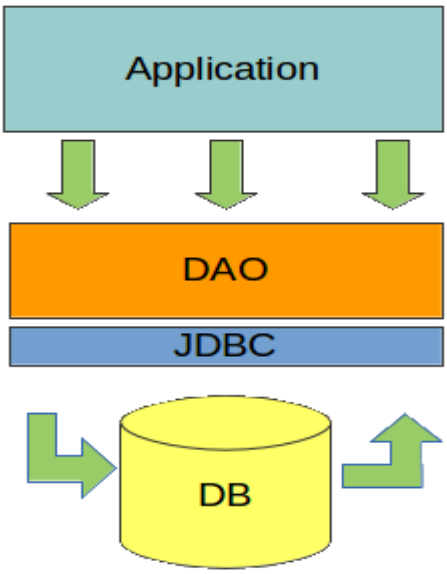

Рисунок 3.7 – Архітектура аплікації з DAO рівнем

Тож в аплікації «Розумне місто» кожна сутність має свій клас, що описує доступ до бази даних. Це такі сутності як:

- Users – містить дані про користувачів системи;
- UsersOrganizations – відображає зв’язок багато до багатьох між таблицями користувачів та організацій;
- Organizations – містить дані про ключову одиницю інформації системи, організації, які будуть керуватись відповідальними користувачами;
- User roles та Roles – ролі в системі;
- Budget – бюджет системи;
- Comments та SeenComments – інформація про коментарі до певних завдань;
- Task – таблиця з інформацією про завдання до конкретної організації;
- Transactions – таблиця з інформацією про переведення коштів на рахунок організації.

У дод. А наведена реалізація транзакції та організації рівня DAO.

Доступом до бази даних, виконанням запитів займається JdbcTemplate. Клас JdbcTemplate є центральним класом у базовому пакеті JDBC. Він обробляє створення і вивільнення ресурсів, що допомагає уникнути поширених помилок, таких як забування закрити з'єднання з базою. Він виконує основні завдання основного робочого процесу JDBC, такі як створення та виконання інструкцій, залишаючи код програми для надання SQL та отримання результатів. Клас JdbcTemplate виконує SQL-запити, інструкції оновлення та виклики збережених процедур, виконує ітерацію по результуючим наборам і витяг повертається значень параметрів. Він також ловить непередбачувані помилки під час роботи з базою і переводить їх у загальну, більш інформативну ієрархію винятків, визначену в пакеті DAO фреймворку Spring.

Під час використання даного підходу роботи з базою даних (JdbcTemplate), потрібно було лише реалізувати інтерфейси зворотного виклику, надаючи їм чітко визначений контракт. Інтерфейс зворотного виклику PreparedStatementCreator створює підготовлений оператор з урахуванням з'єднання, наданого цим класом, надаючи SQL і будь-які необхідні параметри. Те ж саме стосується інтерфейсу CallableStatementCreator, який створює оператори, що викликаються. Інтерфейс RowCallbackHandler витягує значення з кожного рядка результуючого набору.

JdbcTemplate було використано в реалізації DAO за допомогою прямого створення налаштованого екземпляра в контейнері Spring IoC і передано в якості Spring Bean.

Але безпосередньо використання рівню DAO (або рівень репозиторію) для отримання даних на користувальницький інтерфейс не прийнято і вважається поганим тоном, для цього були придумані сервіси. Service - це Java клас, який вміщує в собі основну (бізнес-логіку) роботи з певною сутністю. В основному сервіс використовує готові DAO / Repositories або ж інші сервіси, для того щоб надати кінцеві дані для інтерфейсу користувача. У системі «Розумне місто» було створено наступні сервіси:

– “Менеджмент користувачів” дозволяє адміністратору редагувати власні користувацькі дані, переглядати список зареєстрованих користувачів системи, деактивувати та активувати нового користувача, та при необхідності змінювати його набір ролей;

– “Менеджмент задач системи” дозволяє редагувати опис доступних задач в системі, який потім відображається користувачам в з відповідною роллю до певної організації;

– “Коментарі” дозволяє коментувати ті чи інші завдання відповідно до потреб;

– “Бюджет” та “Транзакції” дозволяють проводити операції з бюджетом організацій та переглядати історію платежів, при необхідності відміняти деякі з них.

На рівні сервісів використовується спеціальний мапер(конвертер), що трансформує об’єкти з бази даних до JSON формату. Приклад такої сутності продемонстровано на рисунку 3.8.


```
{
   "id":1,
   "name":"Police",
   "address":"Justice street - 50",
   "responsiblePersons":[
      {
         "id":3,
         "name":"RESPONSIBLE",
         "surname":"PERSON",
         "email":"responsible_person@mail.com",
         "phoneNumber":"0666666666",
         "active":true,
         "createdDate":"2020-04-19T12:15:40.000",
         "updatedDate":"2020-04-19T12:30:17.000"
      }
   ],
   "createdDate":"2020-04-19T12:21:43.000",
   "updatedDate":"2020-04-19T12:21:43.000"
}
```


Рисунок 3.8 – Приклад запиту організації

3.1.3 Розробка контролерів

Розробка та тестування контролерів проводились відповідно до вимог системи «Розумне місто». Це означає, що було створено обмеження по ролям на всі шляхи доступу до інформації. Ці обмеження описані в класі MethodSecurityConfig.

```
@Configuration("securityConfiguration")
@PropertySource(value = "classpath:methodSecurity.properties")
public class MethodSecurityConfig {

    @Value("#{'${userController.deleteUser}'.split(',')}")
    private List<String> userControllerDeleteUserAllowedRoles;

    @Value("#{'ROLE_ADMIN'.split(',')}")
    private List<String> userControllerActivateUserAllowedRoles;

    @Value("#{'ROLE_ADMIN'.split(',')}")
    private List<String> userControllerSetRolesByUserIdAllowedRoles;
```

Рисунок 3.9 – Клас MethodSecurityConfig

Як можна помітити зверху, є анотація PropertySource, це означає що якісь властивості будуть взяті з файлів конфігурації аплікації. Конкретно в файлі methodSecurity.properties описано яку роль має мати юзер щоб отримати доступ до шляху пов’язаного з контролерами. На наступному рисунку продемонстровано контролер організацій.

```
#OrganizationController
organizationController.create=ROLE_ADMIN
organizationController.update=ROLE_ADMIN
organizationController.delete=ROLE_ADMIN
organizationController.addUserToOrganization=ROLE_ADMIN
organizationController.removeUserFromOrganization=ROLE_ADMIN
```

Рисунок 3.10 – Список ролей та ендпоінтів контролеру організації

Як можна побачити, лише користувач з роллю адміністратор може видаляти, активовувати та змінювати ролі інших користувачів. Такі самі налаштування існують для абсолютно всіх роутів аплікації.

Для полегшення розробки контролерів було використано Swagger Documentation. Swagger - це набір правил (іншими словами, специфікація) для формату, що описує API REST. Перш за все, оскільки Swagger використовує спільну мову, яку може зрозуміти кожен, він легко зрозумілий як розробникам, так і не розробникам.

Таким чином, розробники програмного забезпечення, менеджери продуктів і проектів, бізнес-аналітики і навіть потенційні клієнти можуть отримати доступ до дизайну API.

Крім того, оскільки Swagger легко регулюється, його можна успішно використовувати для тестування API та виправлення помилок. Ще один важливий момент полягає в тому, що одна і та ж документація може використовуватися для прискорення різних процесів, що залежать від API [15]. Конкретно в даній аплікації було використано засоби Swagger UI та Swagger Inspector:

− Swagger UI - дозволяє самостійно створювати документацію для різних платформ. Swagger UI - це повністю настроюється інструмент, який може бути розміщений в будь-якому середовищі. Великим плюсом є те, що він дозволяє розробникам економити багато часу на документацію API;

− Swagger Inspector - це інструмент для тестування та автоматичної генерації документації OpenAPI для будь-якого API. Swagger Inspector дозволяє легко перевіряти і тестувати API-інтерфейси без будь-яких обмежень на те, що тестується. Тести автоматично зберігаються в хмарі з простим доступом.

На рисунку 3.11 можна побачити повний список контролерів.

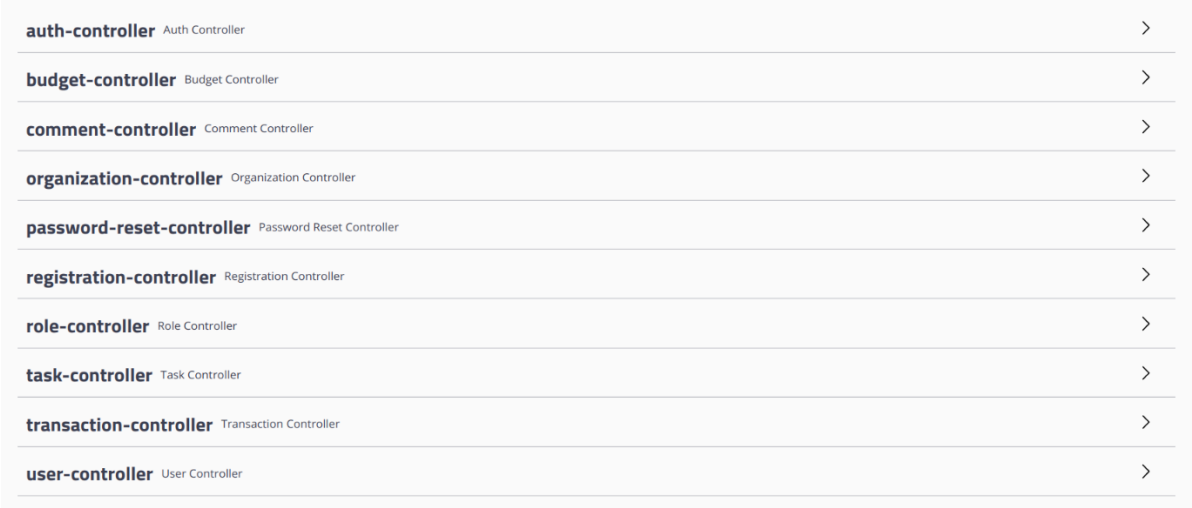

Рисунок 3.11 – Список всіх контролерів аплікації

На наступному рисунку продемонстровано список всіх роутів контролера Організації. Для того що відтворити процес автентифікації використовуючи Swagger, було вирішено скористатись Bearer токеном. Це проста строка, яка діє як аутентифікація запиту API, відправленого в заголовку HTTP "Authorization". Рядок не має сенсу для клієнтів, які використовують його, і може бути різної довжини.

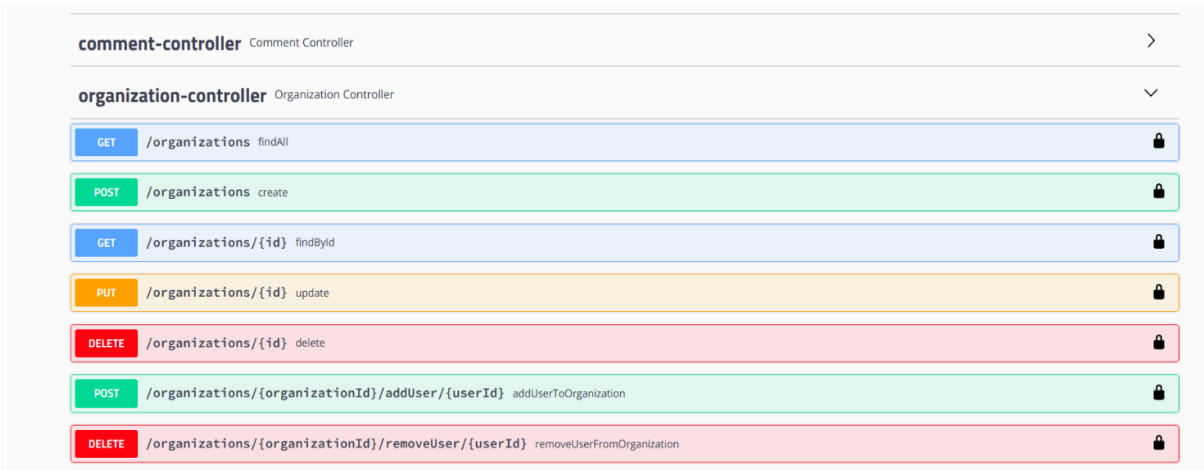

Рисунок 3.12 – Список ендпоінтів контролеру організацій

Як ми можемо побачити, всі вони потребують авторизації(JWT токену) для доступу.

Bearer токен - це набагато простіший спосіб виконання запитів API, оскільки вони не вимагають криптографічного шифрування. Компроміс полягає в тому, що всі запити API повинні бути зроблені через HTTPS - з'єднання, оскільки запит містить маркер відкритого тексту, який може бути використаний ким завгодно, якщо він був перехоплений. Перевага полягає в тому, що він не вимагає складних бібліотек для виконання запитів і набагато простіше для реалізації як клієнтами, так і серверами.

Тож скористаємось контролером автентифікації (рис. 3.13).

Рисунок 3.13 – Ендпоінт логіну в Swagger

Як бачимо треба ввести логін та пароль для доступу до ресурсів. Після успішної авторизації буде згенеровано JWT токен і ми зможемо спробувати використати інші контролери. Приклад відповіді серверу продемонстровано на рисунку 3.14.

Рисунок 3.14 – Респонс на запит логіну

Так виглядає успішна авторизація, нам повернувся token. Після цього просто додамо цю строку з префіксом «Bearer» в заголовок запиту і спробуємо

дістати список всіх організацій (лише маючи права адміністратора або supervisor’a). Продемонстровано на наступному рисунку.

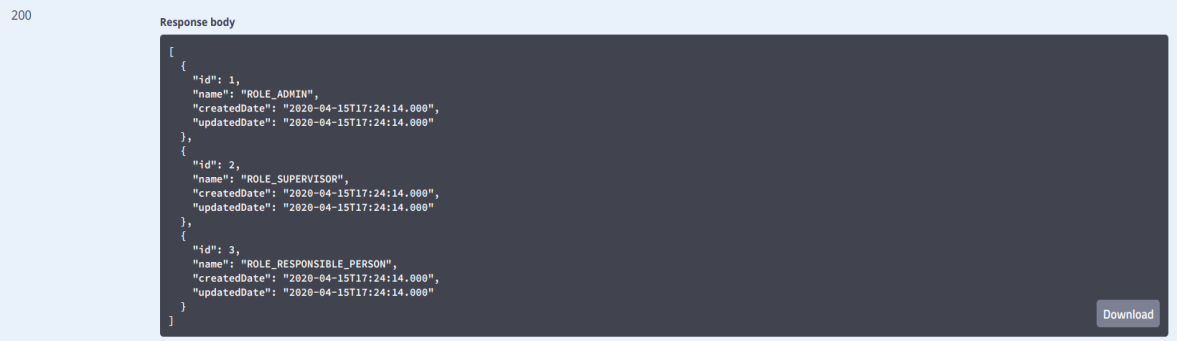

Рисунок 3.15 – Респонс на запит всіх організацій

Таким чином Swagger допомагає легко тестувати та вдосконалювати розробку контролерів.

3.2 Розробка клієнтської частини

Для розробки UI(клієнтської) частини було обрано Angular, що частіше всього використовується для побудови SPAs (односторінкових додатків). Ось короткий огляд відмінних рис Angular:

– REST Easy. RESTful аплікації швидко стають стандартом для обміну даними між сервером і клієнтом. В одному рядку JavaScript можна швидко спілкуватися з сервером і отримувати дані, необхідні для взаємодії з веб-сторінками. Angular перетворює це в простий об'єкт JavaScript, як моделі, дотримуючись шаблону MVVM (Model View View-Model);

– прив'язка даних і впровадження залежностей. Все в шаблоні MVVM передається автоматично через користувальницький інтерфейс, коли що-небудь змінюється. Це усуває необхідність в оболонках, геттерах / сетерах або оголошеннях класів. Angular обробляє все це, тому можна описувати дані так

само просто, як за допомогою примітивів JavaScript, таких як масиви, або так само складно, як за допомогою користувацьких типів. Оскільки все відбувається автоматично, можна запросити свої залежності в якості параметрів в сервісних функціях Angular, а не один гігантський виклик main() для виконання коду;

– розширює HTML-код. Більшість сайтів, створених сьогодні, являють собою гігантську серію тегів з невеликою семантичною ясністю. В такому випадку потрібно було б створювати великі та вичерпні класи CSS, щоб висловити намір кожного об'єкта в DOM. За допомогою Angular можна керувати своїм HTML як XML, що дає нескінченні можливості для тегів і атрибутів. Angular виконує це за допомогою свого HTML - компілятора і використання директив для запуску поведінки, заснованого на недавно створеному синтаксисі;

– тестування на високому рівні. Як вже говорилося вище, AngularJS не вимагає ніяких додаткових фреймворків або плагінів, включаючи тестування. Якщо ви знайомі з такими засобами як QUnit, Mocha або Jasmine, то вам не складе труднощів вивчити API модульного тестування Angular і Scenary Runner, які допоможуть виконати тести в максимально наближеному до реального стану виробничого додатки стані. Це фундаментальні принципи, які спрямовують Angular до створення ефективної, керованої продуктивністю і підтримуваної інтерфейсної кодової бази. Сервер має справу тільки з наданням даних з бази даних через REST API, тому він також працює швидко, тому що він не перевантажений завданням рендеринга сторінок.

Це полегшує завдання бекенд - розробника, а фронтенд - розробник отримує більше контролю над веб-сайтом. Завдяки цьому додаток може бути побудовано за менший час.

3.2.1 Розробка сервісів

Сервіс в Angular - це широкий шар аплікації, що охоплює купу значень та функцій необхідних додатку. Служба - це, як правило, клас з вузькою, чітко визначеною метою.

Angular розрізняє компоненти від сервісів для підвищення модульності і можливості повторного використання. Відокремлюючи функціональні можливості компонента, пов'язані з поданням, від інших видів обробки, можна зробити свої класи компонентів економічними і ефективними.

Angular дійсно допомагає дотримуватися цих принципів, дозволяючи легко розкласти логіку програми на служби і зробити ці служби доступними для компонентів за допомогою впровадження залежностей.

Так в системі «Розумне місто» було створено таку саму кількість сервісів що має REST частина, вони спілкуються між собою за допомогою токену.

3.2.2 Розробка компонентів

Компоненти - це «будівельний блок» інтерфейсу користувача Angular. Angular по суті - це дерево Angular компонентів. Angular компонент - це підмножина директив. На відміну від директив, компоненти завжди мають шаблон, і тільки один компонент може бути створений для кожного елемента в шаблоні.

В ідеалі завдання компонента полягає в тому, щоб включити користувальницький інтерфейс і нічого більше. Компонент повинен представляти властивості та методи прив'язки даних, щоб бути посередником між поданням (візуалізованим шаблоном) та логікою програми (яка часто включає в себе деяке поняття моделі).

В системі «Розумне місто» компонент делегує завдання сервісам, такі як витяг даних з сервера, перевірка введення даних користувачем або ведення журналу безпосередньо в консоль.

3.2.3 Розробка інтерфейсу користувача

Розробка інтерфейсу відіграє напевно найважливішу частину в аплікації, тому що потрібно зручно вмістити існуючий функціонал для користувача. Для цього краще розглядати кожен компонент як сутність і мати певний перелік дій які можливо проводити над нею. Наприклад, організація:

− може бути створена / видалена / редагована тільки адміністратором;

− має завдання від супервайзера які потрібно виконати до певної дати;

− має відповідальну особу яка відповідає за виконання покладених задач;

− має проблеми, вирішення яких потребує деяких затрат, отже відповідальна особа може надіслати запит на вирішення цієї проблеми супервайзеру.

Коментар:

− може бути створеним / видаленим / редагованим;

− доступний під конкретним завданням доступна для перегляду та коментування лише для Responsible Person прикріпленої до цього завдання та Supervisor’a.

Завдання:

− завдання може бути відміненим в процесі розгляду;

− можливість заборонити подальше коментування під конкретним завданням;

− може бути заплановане на майбутнє;

− може бути заморожене Superviserom;

– прогрес виконання може змінювати відповідальналюдина тільки з підтвердження супервайзером;

– може сповіщувати супервайзора та відповідальну особу про момент завершення, приближення дати завершення;

– завдання може змінювати відповідальна людина до моменту підтвердження його (завдання) супервайзером. Після пітвердження задвання зміна відбудеться лише зі згоди супервайзора;

– завдання може мати коментарі від супервайзора і відповідальної особи;

– завдання може мати стан (В процесі підтвердження, В процесі виконання, Виконане, Заморожене, В очікувані виділення бюджету).

Перше що бачить користувач це - інтерфейс входу до системи. Виглядає він наступним чином (рис 3.16). Він являє собою авторизацію, дозволяючи створити користувача, а також нагадати пароль виславши його на пошту.

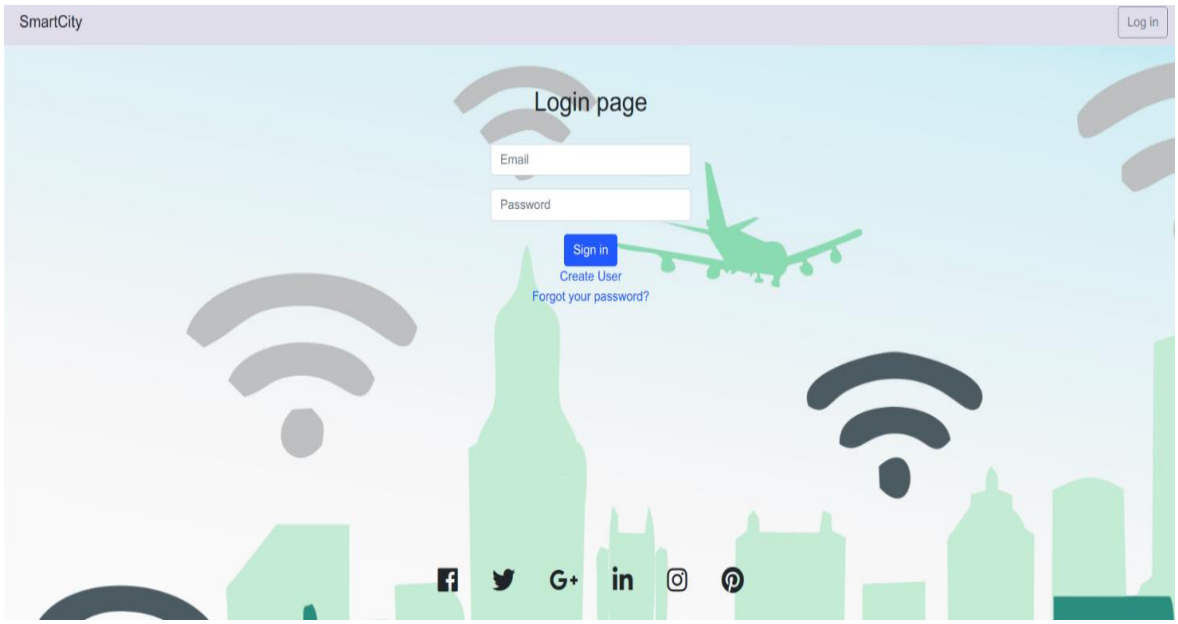

Рисунок 3.16 – Сторінка авторизації

Інтерфейс реєстрації виглядає таким чином (рис. 3.17). В ньому присутня валідація номеру телефона (по кількості та знакам), також електронної пошти і паролю.

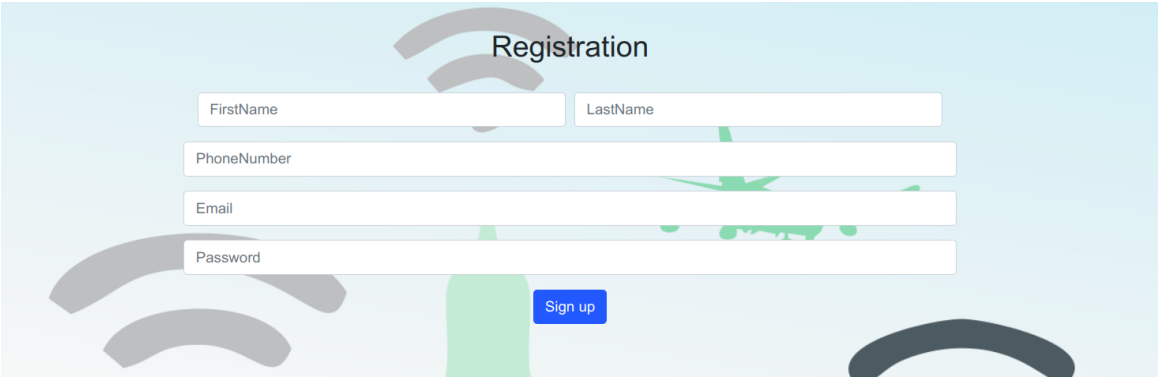

Рисунок 3.17 – Сторінка реєстрації користувача

Адміністраторська частина роботи системи «Розумне місто» дозволяє управляти відповідальними особами як користувачами організацій та супервайзорами які відповідальні за ведення бюджету системи. Вигляд управління користувачами представлено на рисунку 3.18.

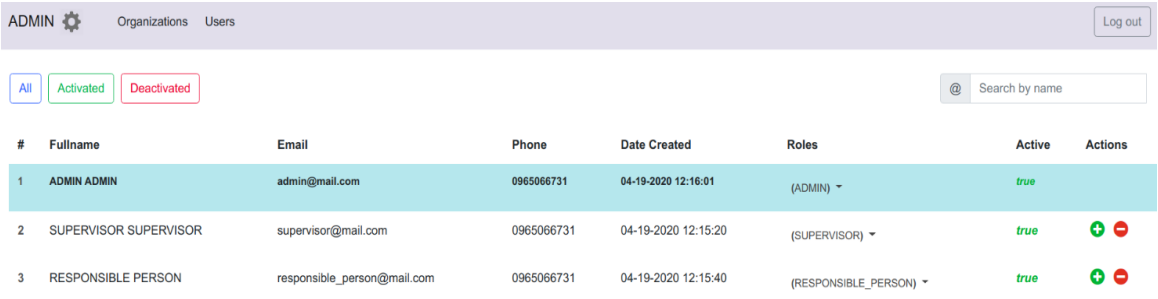

Рисунок 3.18 - Адміністративна частина управління користувачами

Редагування профілю представляє собою можливість змінити лише ім’я, прізвище та номер мобільного телефону. Це продемонстровано на наступному рисунку.

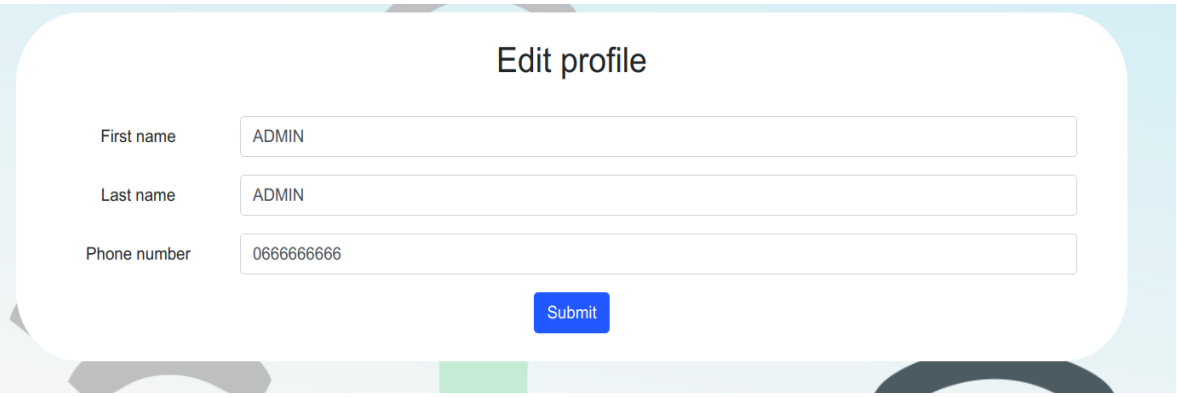

Рисунок 3.19 – Сторінка редагування профілю

Організація системи «Розумне місто» виглядає як віджет що містить найважливішу інформацію:

- кількість завдань які перебувають в різних станах;
- назва та адреса організації.

Таких віджетів може бути безліч, все залежить від кількості служб міста.

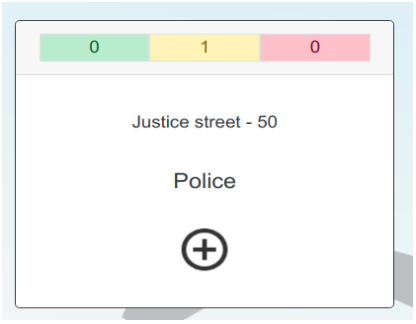

Рисунок 3.20 – Віджет організації

Створювати завдання для організації можуть як супервайзор так і відповідальна особа. Інтерфейс цього функціоналу виглядає наступним чином (рис. 3.21). Він складається з :

– заголовку завдання;

– повного опису;

– кошти які потрібні для того щоб виконати це завдання на думку тієї особи що створює це завдання (далі воно буде або узгоджене з супервайзором або змінено бюджет завдання);

– відповідальна особа з цієї організації;

– дата до якої треба виконати завдання.

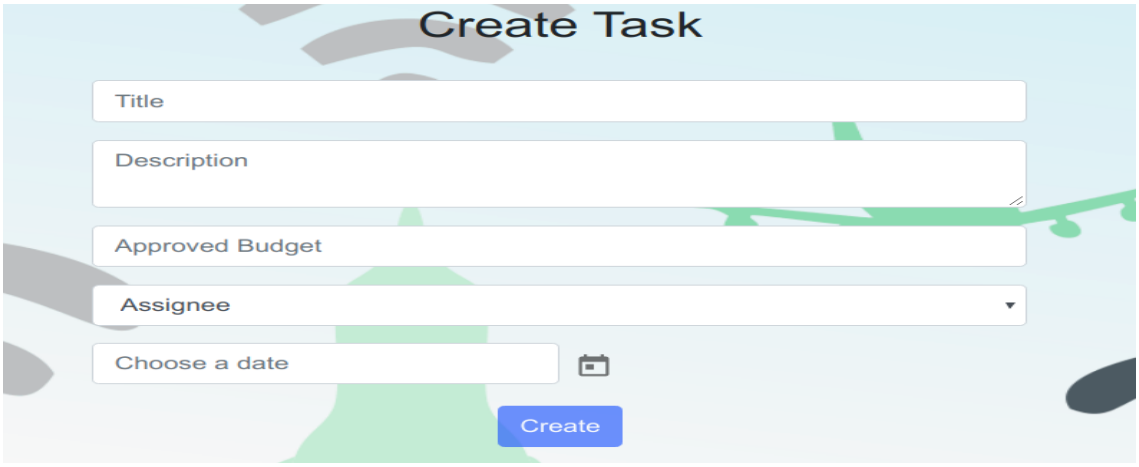

Рисунок 3.21 – Інтерфейс створення завдання

Супервайзор може редагувати завдання, змінювати бюджет якщо це потрібно. Тоді буде потрібно створити додаткову транзакцію з бюджету міста до організації. Інтерфейс редагування завдання подано на рисунку 3.22.

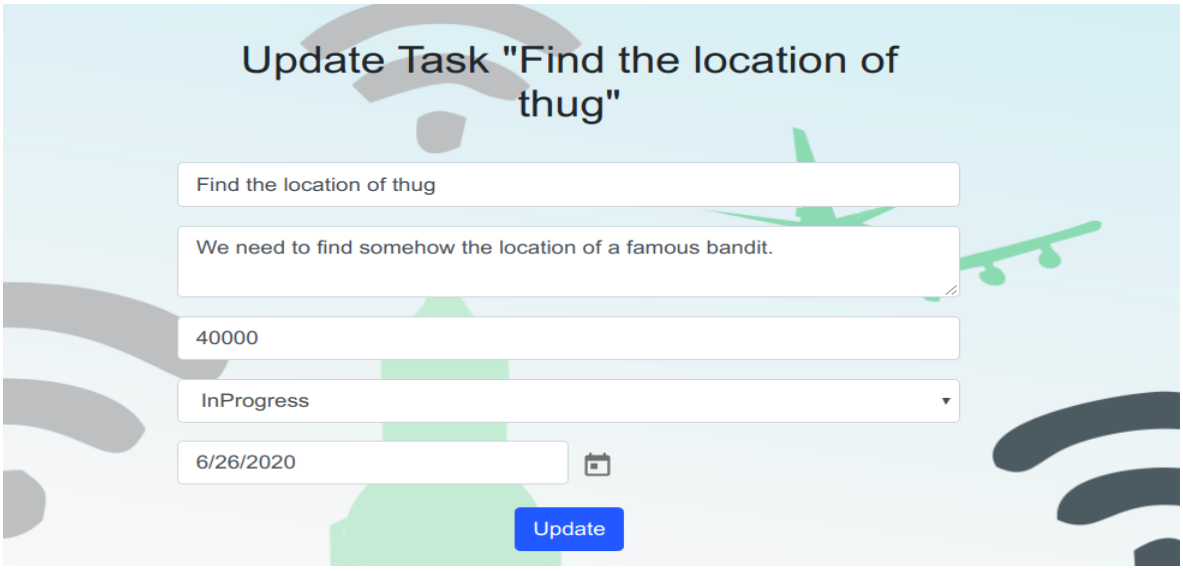

Рисунок 3.22 – Інтерфейс редагування завдання

Кожна організація має свій список завдань. На наступному рисунку представлено організацію що відображає поліцію та має поки що лише 1 завдання. Тут можна прослідкувати його статус, кошти які було виділено на нього, чи воно було узгоджено з обох сторін (супервайзора та виконавчої особи), продивитись транзакції, коментарі або видалити.

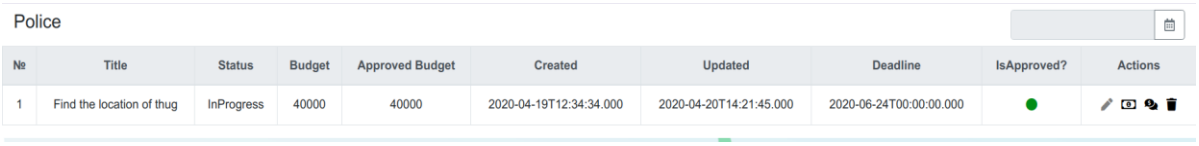

Рисунок 3.23 – Інтерфейс списку завдань організації

Як було згадано вище у завданні є секція коментарів, вона виглядає наступним чином (рис. 3.24). Коментарі можна сортувати по даті та шукати по ключовим словам.

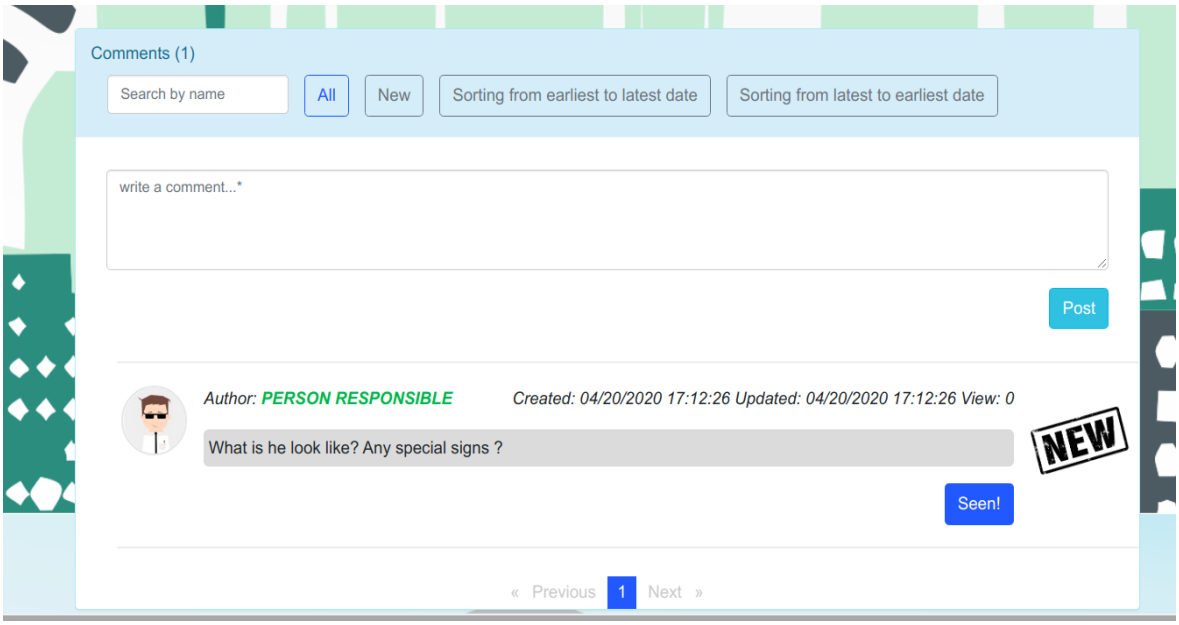

Рисунок 3.24 – Інтерфейс секції коментарів до завдання

А список транзакцій організації виглядає наступним чином (рис. 3.25). Тут є інформація про кількість коштів які було передано, стан бюджету до операції.

Рисунок 3.25 – Інтерфейс транзакцій конкретного завдання

3.3 Тестування

Для забезпечення стійкості та впевненості в правильності написання коду та його функціонуванні в розробку системи «Розумне місто» включено інтеграційні та модульні види тестувань.

Використано наступні інструменти для тестування програмного забезпечення системи «Розумне місто»:

1. Mockito Framework – засіб створення тестованих класів;
2. Junit Framework – фреймворк для тестування;
3. AssertJ – бібліотека для тестування функціоналу.

Для автоматичного запуску тестів при зміні вихідного коду програми іншими розробниками виконано інтеграцію з системою CI – Travis CI. Система автоматично запускає тести вихідного коду системи «Розумне місто» при кожній зміні коду і затверджує його роботу, якщо тести пройшли успішно. Приклад інтеграційного тесту представлено на рисунку 3.26.

```
@Test
void findById_failFlow() throws Exception {
    Mockito.when(userService.findById(fakeId))
            .thenThrow(notFoundException);

    mockMvc.perform(get( urlTemplate: "/users/" + fakeId)
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isNotFound())
            .andExpect(jsonPath( expression: "url").value( expectedValue: "/users/" + fakeId))
            .andExpect(jsonPath( expression: "message").value(notFoundException.getLocalizedMessage()));
}
```

Рисунок 3.26 – Приклад тесту

Розроблений програмний продукт системи «Розумне місто» покрито тестами та перевірено кожну деталь розроблюваного функціоналу, реалізацію логіки та керування модулями системи, авторизації виконання транзакційних дій користувачів, процесу створення, редагування та видалення користувачів системи. Загальний відсоток покриття тестами представлено на рисунку 3.27.

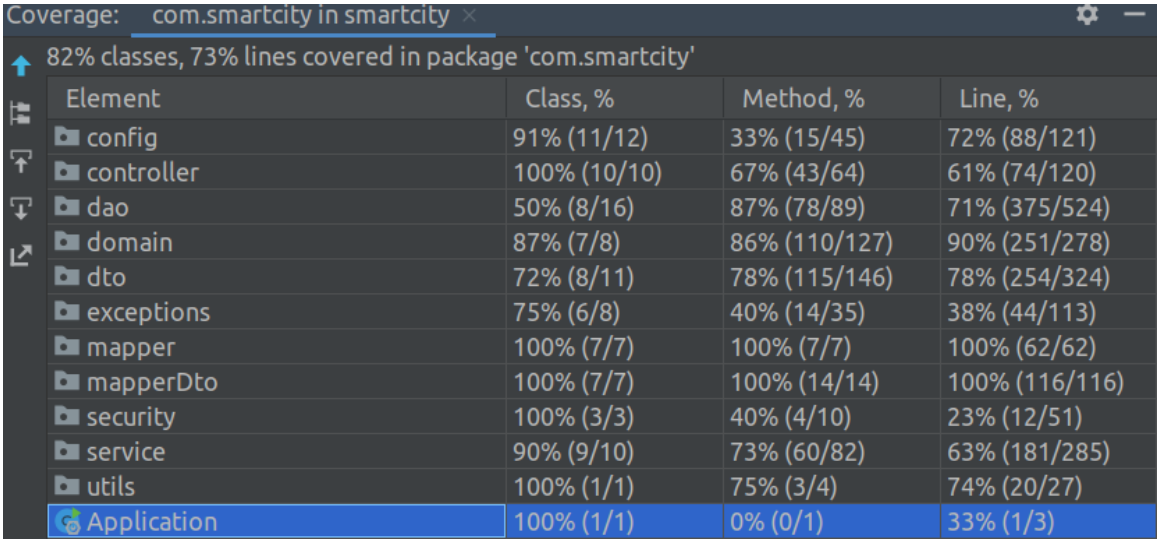

Рисунок 3.27 – Покриття аплікації тестами

4 ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ПРОЕКТУ

4.1 Бізнес-план розробки програмного забезпечення

Коротке найменування: Розробка веб-сайту «Система управління службами міста - SmartCity».

Повне найменування: Розробка веб-сайту «Система управління службами міста - SmartCity» для різних міст, з можливістю створення завдань для конкретних служб міста.

4.1.1 Резюме

Видом послуг є надання послуг в інформатизації та створенні відповідного сайту системи управління службами конкретного міста.

КВЕД 62.01 Комп'ютерне програмування

КВЕД 62.02 Консультування з питань інформатизації

Цільова аудиторія продукту – керманичі міст, зацікавлені в розвитку безпеки та прагматичності свого міста. Для реалізації проекту слід залучити інвестиції в розмірі 1 500 000.00 грн (Один мільйон п’ятсот тисяч гривень, 00 копійок). Джерелами фінансування проекту будуть адміністрації міст та можливі інвестиції.

За перший рік не плануються продажі. Вони стануть можливими лише на 2 рік після повної реалізації продукту першого міста.

Для створення продукту потрібні фронтенд команда в кількості 2 чоловік та бекенд команда в кількості 4 чоловік. Менеджер проекту в кількості 1 людина. Також бізнес аналітик та команда промоутерів в кількості 5 людей.

Спочатку місто має замовити продукт, та надалі оплачувати підтримку системи. Тому оплата планується частинами, різниця в оплаті буде лише в

кількості служб міста. Середня ціна реалізації системи під конкретне місто буде становити близько мільйону гривень, підтримка системи оцінюється в 100 000 грн за місяць.

– середня ціна системи під одне місто – 1 мільйон гривень (час реалізації близько року);

– загальний прибуток (за місяць) без податків 100 000,00 грн. ;

– рентабельність: 1.9 .

4.1.2 Маркетинг

Для того щоб задовольнити клієнта потрібна зручна система. Немало часу має буде присвячено розробці дизайну зручного інтерфейсу. Також, швидкодія та надійність є важливими пунктами. Команда бекенд має докласти зусилля щоб код працював швидко та відлагоджено, що немало важливо є також читабельність такого коду. Тому потрібні фахівці високого класу.

Аналогів послуги немає в країні.

Головним недоліком послуги є велика ціна та незацікавленість міст в отриманні доступу до даної системи.

Альтернативних рішень аплікації немає, тому встановлювати ціну у порівнянні з ними - неможливо.

Вартість послуги вираховується на основі кількості організацій та співробітників відповідно. Вартість однієї служби в середньому буде становити 250 000 грн (Двісті п’ятдесят тисяч гривень 00 копійок). Тому мінімумом для міста буде приблизно 4 служби.

Просуванням системи буде займатись команда із промоутерів. Будуть створені реклами на сайтах, проінформовані голови міст. Також планується створення реклами для кожної зі служб міста.

4.1.3 Обґрунтування необхідних фінансових вкладень

Багато спеціалістів працюють дистанційно. Тому офіс не відіграє ролі.

Режим роботи - повний робочий день. Перші фінансові вкладення потрібні будуть на оплату фахівців та пошук кадрів.

4.1.4 Нормативно-правові нюанси

Юридичний статус - Фізична особа-підприємець 3 групи.

Переваги ФОП:

– низький розмір штрафів;

– низьке держмито;

– свобода в розподілі доходів;

– спрощена ліквідація;

– звільнення від сплати майнового податку на використовуване майно;

– можливість працювати без відкриття банківського рахунку;

– можливість вести діяльність без атрибутів юрособи - печатки, штампа, статуту;

– робота по схемі єдиного податку.

4.1.5 Складання фінансового бюджету

Джерелом фінансування будуть інвестиції(1 500 000 грн) або кредитні кошти.

Витрати на старт проекту близько 1 000 000 грн.

Перший прибуток буде лише приблизно через рік, після створення першого екземпляру системи для міста.

Системою оподаткування є фізична особа-підприємець 3 групи. Єдиний Податок — від доходу 3% або 5% ЄСВ «за себе» — 1039,06 грн/міс. Термін сплати податків: ФОП групи 3 — протягом 10 к. дн., що настають за граничним строком подання декларації (п. 295.3 ПКУ) .

4.1.6 Оцінка можливих ризиків проекту

Існують деякі ризики провалу проекту в плані незацікавленості в послузі. В даному випадку буде виділено більше коштів на рекламу системи. В плані покриття збитків можливе підвищення цін на послугу.

ВИСНОВКИ

В даній дипломній роботі було розглянуто проблему розробки та експлуатації систем «Розумне місто», їх основні складові та властивості впровадження в життя та організацію роботи сучасного міста. Описано особливості впровадження даних систем на основі досвіду зарубіжних міст.

Проаналізовано засоби захищеного підключення по мережі Інтернет. Виявлено надійність та простоту використання технології JWT яка з'явилась досить недавно і швидко набула популярності у використанні.

Доступ до системи «Розумне місто» здійснюється через мережу Інтернет, тому проведено детальний аналіз протоколів захисту для забезпечення захищеного підключення та безпечної передачі даних користувачів.

Для забезпечення підключення користувачів було використано протокол HTTPS, який базується на використанні алгоритму обміну сертифікатів з відкритим ключем TSL.

Запропоноване рішення ідеально підходить для систем з великим навантаженням підключень на одиницю часу, оскільки веб-сервер автоматично розподіляє навантаження та запити і підключення поміж різною кількістю піднятих реплікації додатку в реальному часі, а якщо раптом одна із них перестає відповідати на запити користувачів, або припиняє свою роботу через ряд інших причин, веб-сервер автоматично виключає її зі списку доступних для роботи з користувачами до тих пір поки репліка не відновить свою роботу.

Систему «Розумне місто» було розподілено на такі компоненти: база даних MySQL, клієнтська частина написана засобами розробки веб з використанням фреймворку Angular 8. Серверна частина засобами мови Java, фреймворку Spring. Викладено вимоги до системи в технічному завданні та визначено функціонал ролей користувачів системи відповідно до кожного функціонального модуля, керованої частини системи «Розумне місто». Представлено діаграму архітектури та взаємодії користувачів, їх функції та операції над роботою в системі «Розумне місто».

Базуючись на цих даних програмно реалізовано систему організації роботи міських установ та автентифікації користувачів за допомогою технології JWT. Система «Розумне місто» включала в себе розробку клієнтської частини, проектування бази даних MySQL, серверну частину написану мовою Java. Вихідний код серверної частини покрито інтеграційними та модульними тестами з використанням фреймворків Junit, Mockito, AssertJ. Для забезпечення продуктивної та безпечної роботи над проектом використано інструмент автоматичного тестування Travis CI, який демонструє надійність написаного коду та гарантує його стабільність при добавленні нового функціоналу в систему, з можливістю відслідковування проблемного коду. Результатом роботи є система «Розумне місто» з підтримкою роботи в режимі високої доступності, безпечним доступом до даних, готова для використання в сучасному середовищі для організації роботи міських установ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

REFERENCES

1. Anthony M. Townsend Smart Cities: Big Data, Civic Hackers, and the Quest for a New Utopia. London : Norton, 2013. 324 p.
2. John Rossman The Amazon Way on IoT: 10 Principles for Every Leader from the World's Leading Internet of Things Strategies. Seattle : Clyde Hill Publishing, 2016. 158 p.
3. Jennifer Clark Uneven Innovation: The Work of Smart Cities. New York : Columbia University Press, 2020. 311 p.
4. I. Lazarowych and J. Nikolaychuk, "Theory and methods of digital streams randomization in telecommunicational systems," Modern Problems of Radio Engineering, Telecommunications and Computer Science (IEEE Cat. No.02EX542), Lviv-Slavsko, Ukraine, 2002, pp. 265-, doi: 10.1109/TCSET.2002.1015956.
5. Oliver Gassmann, Jonas Böhm, Maximilian Palmié Smart Cities: Introducing Digital Innovation to Cities. Bingley : Emerald Group Publishing, 2019. 368 p.
6. Germaine Halegoua Smart Cities (MIT Press Essential Knowledge series). Massachusetts : MIT Press, 2020. 248 p.
7. Los Angeles Smart City: Data And Sustainability. URL: https://mobility.here.com/learn/smart-city-initiatives/los-angeles-smart-city-data-and-sustainability (дата звернення: 15.02.2020).
8. What is Smart Parking? URL: https://www.parkeagle.com/2018/05/12/what-is-smart-parking/ (дата звернення: 15.02.2020).
9. Smart Waste Management Systems. URL: https://www.postscapes.com/smart-trash/ (дата звернення: 15.02.2020).
10. ZigBee alliance. URL: https://zigbeealliance.org/ (дата звернення: 06.03.2020).

11. S.O.L.I.D: The First 5 Principles of Object Oriented Design. URL: https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design (дата звернення: 15.03.2020).
12. What is XML? URL: https://whatis.techtarget.com/definition/XML-Extensible-Markup-Language (дата звернення: 15.03.2020).
13. JSON: What It Is, How It Works, & How to Use It. URL: https://www.copterlabs.com/json-what-it-is-how-it-works-how-to-use-it/ (дата звернення: 12.04.2020).
14. All You Need to Know About UML Diagrams: Types and 5+ Examples. URL: https://tallyfy.com/uml-diagram/ (дата звернення: 12.04.2020).
15. What Is Swagger? URL: https://swagger.io/docs/specification/2-0/what-is-swagger/ (дата звернення: 12.04.2020).
16. CIT 2:2018. Стандарт кафедри інформаційних технологій. Дипломний проект. Вимоги до змісту та оформлення [Чинний від 2018-09-03]. Вид. офіц. Івано-Франківськ, 2018. 43 с.

ДОДАТОК А

Реалізація Transaction та Organization методів рівня DAO

```
@Repository
public class TransactionDaoImpl implements TransactionDao {
private static final Logger logger =
LoggerFactory.getLogger(TransactionDaoImpl.class);
private JdbcTemplate template;
private TransactionMapper mapper;

@Autowired
public TransactionDaoImpl(DataSource source, TransactionMapper mapper) {
this.mapper = mapper;
template = new JdbcTemplate(source);
}

public Transaction create(Transaction transaction) {
try {
LocalDateTime currDate = LocalDateTime.now();
GeneratedKeyHolder holder = new GeneratedKeyHolder();
transaction.setCreatedDate(currDate);
transaction.setUpdatedDate(currDate);
template.update(con -> createStatement(transaction, con), holder);
transaction.setId(Optional.ofNullable(holder.getKey()).map(Number::longVa
lue)
.orElseThrow(() -> new DbOperationException("Create transaction Dao
method error: " +
"Autogenerated key is null")));
return transaction;
} catch (Exception e) {
logger.error("Can't create transaction:{}. Error:{}", transaction,
e.getMessage());
throw new DbOperationException("Can't create transaction by id=" +
transaction.getId() +
"Transaction:" + transaction);
}
}
```

```
public Transaction findById(Long id) {
try {
return template.queryForObject(Queries.SQL_TRANSACTION_GET, mapper, id);
} catch (EmptyResultDataAccessException erd) {
throw loggedNotFoundException(id);
} catch (Exception e) {
logger.error("Can't get transaction by id={}. Error: ", id, e);
throw new DbOperationException("Can't get transaction with id=" + id);
}
}

public Transaction update(Transaction transaction) {
int rowsAffected;
try {
LocalDateTime updatedDate = LocalDateTime.now();
transaction.setUpdatedDate(updatedDate);
rowsAffected = template.update(Queries.SQL_TRANSACTION_UPDATE,
transaction.getTaskId(),
transaction.getCurrentBudget(),
transaction.getTransactionBudget(),
updatedDate,
transaction.getId());
} catch (Exception e) {
logger.error("Update transaction error:" + transaction + " " +
e.getMessage());
throw new DbOperationException("Update transaction error.Transaction" +
transaction);
}
if (rowsAffected < 1) {
throw loggedNotFoundException(transaction.getId());
}
return transaction;
}

public boolean delete(Long id) {
int rowsAffected;
```

```
try {
rowsAffected = template.update(Queries.SQL_TRANSACTION_DELETE, id);
} catch (Exception e) {
logger.error("Delete transaction by id = {} error: {}", id,
e.getMessage());
throw new DbOperationException("Delete transaction error");
}
if (rowsAffected < 1) {
throw loggedNotFoundException(id);
} else return true;
}

@Override
public List<Transaction> findByTaskId(Long id) {
List<Transaction> list;
try {
list = template.query(Queries.SQL_TRANSACTION_GET_BY_TASK_ID, mapper,
id);
} catch (Exception e) {
logger.error("Get transaction (task id = {}) exception. Message: {}",
id, e.getMessage());
throw new DbOperationException("Get transaction exception");
}
return list;
}

@Override
public List<Transaction> findByDate(Long id, LocalDateTime from,
LocalDateTime to) {
List<Transaction> list;
try {
list = template.query(Queries.SQL_TRANSACTION_GET_BY_DATE, mapper, from,
to, id);
} catch (Exception e) {
logger.error("Get transaction (task id = {},date from = {},date to = {})
exception. Message: {}",
id, from, to, e.getMessage());
```

```
throw new DbOperationException("Get transaction exception");
}
return list;
}

private PreparedStatement createStatement(Transaction transaction,
Connection con) throws SQLException {
PreparedStatement ps = con.prepareStatement(
Queries.SQL_TRANSACTION_CREATE, Statement.RETURN_GENERATED_KEYS);
ps.setLong(1, transaction.getTaskId());
ps.setLong(2, transaction.getCurrentBudget());
ps.setLong(3, transaction.getTransactionBudget());
ps.setObject(4, transaction.getCreatedDate());
ps.setObject(5, transaction.getUpdatedDate());
return ps;
}

private NotFoundException loggedNotFoundException(Long id) {
NotFoundException notFoundException = new NotFoundException("Transaction
not found.Id = " + id);
logger.error("Runtime exception. Transaction by id = {} not found.
Message: {}",
id, notFoundException.getMessage());
return notFoundException;
}

class Queries {
static final String SQL_TRANSACTION_CREATE = "INSERT INTO
Transactions(task_id,current_budget,transaction_budget," +
"created_date,updated_date) values(?,?,?,?,?)";
static final String SQL_TRANSACTION_GET = "SELECT * FROM Transactions
where id = ?";
static final String SQL_TRANSACTION_UPDATE = "UPDATE Transactions set
task_id = ? , current_budget = ? ," +
"transaction_budget = ? , updated_date = ? where id = ?";
static final String SQL_TRANSACTION_DELETE = "DELETE FROM Transactions
where id = ?";
```

```
static final String SQL_TRANSACTION_GET_BY_TASK_ID = "SELECT * FROM
Transactions where task_id = ?";

static final String SQL_TRANSACTION_GET_BY_DATE = "Select * from
Transactions where created_date between ? and ? " +

"and task_id = ? order by created_date;";

}

}


@Repository

public class OrganizationDaoImpl implements OrganizationDao {


private static final Logger logger =
LoggerFactory.getLogger(OrganizationDaoImpl.class);

private JdbcTemplate jdbcTemplate;

private OrganizationMapper mapper;


@Autowired

public OrganizationDaoImpl(DataSource source, OrganizationMapper mapper)
{

jdbcTemplate = new JdbcTemplate(source);

this.mapper = mapper;

}


@Override

public Organization create(Organization organization) {

try {

GeneratedKeyHolder holder = new GeneratedKeyHolder();

LocalDateTime currDate = LocalDateTime.now();

organization.setCreatedDate(currDate);

organization.setUpdatedDate(currDate);

jdbcTemplate.update(connection -> createStatement(organization,
connection), holder);

organization.setId(Optional.ofNullable(holder.getKey()).map(Number::longV
alue)

.orElseThrow(() -> new DbOperationException("Create organization error:
AutoGeneratedKey = null")));

return organization;

} catch (Exception e) {
```

```
logger.error("Can't create organization:{}. Error:{}", organization,
e.getMessage());
throw new DbOperationException("Can't create organization: " +
organization +
"Create organization Dao method error:" + e);
}
}

@Override
public Organization findById(Long id) {
try {
return jdbcTemplate.queryForObject(Queries.SQL_ORGANIZATION_GET,
mapper, id);
} catch (EmptyResultDataAccessException erda) {
throw loggedNotFoundException(id);
} catch (Exception e) {
logger.error("Can't get organization by id={}. Error: {}", id,
e.getMessage());
throw new DbOperationException("Can't get organization by id = " + id +
"Get organization Dao method error:" + e);
}
}

@Override
public Organization update(Organization organization) {
int rowsAffected;
organization.setUpdatedDate(LocalDateTime.now());
try {
rowsAffected = jdbcTemplate.update(Queries.SQL_ORGANIZATION_UPDATE,
organization.getName(),
organization.getAddress(),
organization.getUpdatedDate(),
organization.getId());
} catch (Exception e) {
logger.error("Can't update organization:{}. Error:{}", organization,
e.getMessage());
```

```
throw new DbOperationException("Can't update organization: " +
organization +
"Update organization Dao method error:" + e);
}
if (rowsAffected < 1) {
throw loggedNotFoundException(organization.getId());
} else return organization;
}

@Override
public boolean delete(Long id) {
int rowsAffected;
try {
rowsAffected = jdbcTemplate.update(Queries.SQL_ORGANIZATION_DELETE, id);
} catch (Exception e) {
logger.error("Can't delete organization by id={}. Error: {}", id,
e.getMessage());
throw new DbOperationException("Can't delete organization by id = " + id
+
"Delete organization Dao method error:" + e);
}
if (rowsAffected < 1) {
throw loggedNotFoundException(id);
} else return true;
}

@Override
public List<Organization> findAll() {
try {
return this.jdbcTemplate.query(Queries.SQL_ORGANIZATION_GET_ALL, mapper);
} catch (Exception e) {
logger.error("Can't get all organizations. Error:{}", e.getMessage());
throw new DbOperationException("Can't get all organizations. GetAll
organization Dao method error:" + e);
}
}
```

```
@Override
public boolean addUserToOrganization(Organization organization, User
user) {
try {
LocalDateTime currDate = LocalDateTime.now();
jdbcTemplate.update(Queries.SQL_ORGANIZATION_ADD_USER_TO_ORGANIZATION,
user.getId(), organization.getId(),
currDate,
currDate);
return true;
} catch (Exception e) {
logger.error("Can't add user:{} to organization:{}. Error:{}", user,
organization, e.getMessage());
throw new DbOperationException("Can't add user to organization." +
" AddUserToOrganization organization Dao method error:" + e);
}
}

@Override
public boolean removeUserFromOrganization(Organization organization, User
user) {
int rowsAffected;
try {
rowsAffected =
jdbcTemplate.update(Queries.SQL_ORGANIZATION_REMOVE_USER_FROM_ORGANIZATIO
N,
organization.getId(), user.getId());
} catch (Exception e) {
logger.error("Can't delete user from organization. User:{}.
Organization{}. Error: {}",
user, organization, e.getMessage());
throw new DbOperationException("Can't delete user from organization." +
" User = " + user + "Organization = " + organization +
"Delete organization Dao method error:" + e);
}
if (rowsAffected < 1) {
logger.error("Can't find user_organization by organizationId = {} and
userId = {}",
```

```
organization.getId(), user.getId());

throw new NotFoundException("Can't find user_organization by
organizationId = "

+ organization.getId() + " and userId = " + user.getId());

} else return true;


}


private PreparedStatement createStatement(Organization organization,
Connection connection) throws SQLException {

PreparedStatement ps =
connection.prepareStatement(Queries.SQL_ORGANIZATION_CREATE,

Statement.RETURN_GENERATED_KEYS);

ps.setString(1, organization.getName());

ps.setString(2, organization.getAddress());

ps.setObject(3, organization.getCreatedDate());

ps.setObject(4, organization.getUpdatedDate());

return ps;

}


private NotFoundException loggedNotFoundException(Long id) {

NotFoundException notFoundException = new NotFoundException("Organization
not found.Id = " + id);

logger.error("Runtime exception. Organization by id = {} not found.
Message: {}",

id, notFoundException.getMessage());

return notFoundException;

}


class Queries {

static final String SQL_ORGANIZATION_CREATE = "INSERT INTO
Organizations(name, address, created_date," +

" updated_date) values(?,?,?,?)";

static final String SQL_ORGANIZATION_GET = "SELECT * FROM Organizations
where id = ?";

static final String SQL_ORGANIZATION_UPDATE = "UPDATE Organizations set
name = ?, address = ?," +

" updated_date = ? where id = ?";
```

```
static final String SQL_ORGANIZATION_DELETE = "DELETE FROM Organizations
where id = ?";

static final String SQL_ORGANIZATION_GET_ALL = "Select * from
Organizations";

static final String SQL_ORGANIZATION_ADD_USER_TO_ORGANIZATION = "INSERT
INTO Users_organizations" +

" (user_id, organization_id, created_date, updated_date) VALUES (?, ?, ?,
?)";

static final String SQL_ORGANIZATION_REMOVE_USER_FROM_ORGANIZATION =
"DELETE FROM Users_organizations " +

"WHERE organization_id=? AND user_id=?";}}
```