Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ ПАРКОМІСЦЬ НА ОСНОВІ КОМП'ЮТЕРНОГО ЗОРУ З ВИКОРИСТАННЯМ МОВИ PYTHON ТА АЛГОРИТМУ YOLO

DEVELOPMENT OF A PARKING MONITORING SYSTEM BASED ON COMPUTER VISION USING PYTHON AND THE YOLO ALGORITHM

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Йоц Іван Васильович

Керівник:
Ящук Андрій Анатолійович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«*20*» *12* 20*24* р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Йоцу Івану Васильовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка системи моніторингу паркомісць на основі комп'ютерного зору з використанням мови Python та алгоритму YOLO»*
Керівник роботи: *доцент Ящук А.А.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *Python, FastAPI, OpenCV, YOLOv5, SQLAlchemy, SQLite, Visual Studio Code, методичні вказівки до виконання кваліфікаційної роботи бакалавра*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз аналогів, формування архітектури, побудова UML-діаграм, вибір технологій, реалізація комп'ютерного зору, створення API, обробка RTSP-потоків, розробка системи зберігання, забезпечення авторизації, тестування стабільності та точності*
5. Перелік графічного матеріалу: *12 рисунків, 1 додаток*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Яцук А. А.*		
Специфікація вимог до розробленої системи	*Яцук А. А.*		
Розробка об`єкта проектування	*Яцук А. А.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	2,57 %		
Академічна доброчесність	*Яцук А. А.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Виконано
2	Аналіз проблеми розробки та впровадження об`єкту проектування	до 01.03.2025 р.	Виконано
3	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	Виконано
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	Виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Виконано.

Здобувач вищої освіти Іван ЙОЦ

Керівник кваліфікаційної роботи Андрій ЯЩУК

АНОТАЦІЯ

Йоц І. В. Розробка системи моніторингу паркомісць на основі комп’ютерного зору з використанням мови Python та алгоритму YOLO. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз сучасних рішень у сфері моніторингу паркомісць, зокрема систем, що базуються на комп’ютерному зорі, та сформульовано вимоги до майбутньої системи. У другому розділі визначено архітектуру, модулі та засоби розробки системи, обґрунтовано вибір мови Python, алгоритму YOLO та інших технологій. У третьому розділі реалізовано серверну систему з API, вбудованим обробником RTSP-камер, модулем виявлення автомобілів, локальною базою даних та механізмом авторизації. Проведено тестування працездатності, точності детекції та збереження даних.

Ключові слова: моніторинг паркомісць, комп’ютерний зір, YOLO, Python, FastAPI, RTSP, SQLite.

ABSTRACT

Yots I. Development of a Parking Monitoring System Based on Computer Vision Using Python and the YOLO Algorithm. Manuscript. Qualification work of the bachelor's program "Software Engineering" in the specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter analyzes existing solutions for parking monitoring, focusing on systems that use computer vision, and formulates the requirements for the new system. The second chapter describes the software architecture, modules, and selected technologies, justifying the use of Python and YOLO. The third chapter presents the implementation of a backend system with an API, RTSP stream handler, vehicle detection module, local database, and authorization mechanism. The system is tested for stability, detection accuracy, and data integrity.

Keywords: parking monitoring, computer vision, YOLO, Python, FastAPI, RTSP, SQLite.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах інтенсивного розвитку міської інфраструктури ефективне управління паркувальними просторами є актуальним завданням для органів місцевого самоврядування, підприємств і приватного сектору. Відсутність доступних, гнучких і автономних рішень для моніторингу паркомісць створює передумови для розробки нових підходів на базі комп'ютерного зору. Технології виявлення об'єктів у відеопотоці дають змогу створювати автоматизовані системи, здатні у реальному часі контролювати заповненість паркінгів без потреби в дорогих сенсорних установках.

Актуальність даної роботи полягає у створенні програмної системи, яка забезпечує автоматичний моніторинг паркомісць у реальному часі, використовуючи алгоритм YOLO та технології Python, FastAPI, OpenCV. Це дозволяє уникнути потреби в додатковому обладнанні, знижує витрати і підвищує масштабованість рішення.

Мета кваліфікаційної роботи полягає у створенні серверної системи моніторингу паркомісць із застосуванням комп'ютерного зору та алгоритму YOLO, яка забезпечує обробку відеопотоків, виявлення транспортних засобів, збереження даних та доступ через API.

Об'єктом кваліфікаційної роботи є процес автоматизованого контролю за зайнятістю паркомісць.

Предметом кваліфікаційної роботи виступають програмні методи реалізації комп'ютерного зору та архітектура програмного забезпечення для контролю зайнятості паркувальних місць.

Завдання кваліфікаційної роботи:

– здійснити детальний аналіз наявних методів контролю паркомісць, зосередившись на системах, що використовують відеоаналітику, машинне навчання або сенсори;

– визначити загальну архітектуру системи та побудувати її функціональні блоки, включаючи опис способів взаємодії модулів і механізмів підтримки цілісності даних;

– реалізувати модуль виявлення автомобілів на зображеннях із камер, використовуючи модель YOLOv5, адаптовану для роботи в реальному часі;

– створити систему обробки відеопотоків через RTSP-протокол із механізмом періодичного захоплення кадрів;

– спроєктувати та реалізувати базу даних для збереження інформації про паркомісця, їх координати, статуси та налаштування;

– розробити REST API для взаємодії з клієнтським інтерфейсом, що дозволяє виконувати CRUD-операції над усіма сутностями системи;

– реалізувати механізм авторизації користувачів за допомогою JWT-токенів;

– забезпечити гнучкість та модульність системи, що спрощує подальшу адаптацію її логіки та структури без значних змін;

– провести тестування системи на точність, стабільність, надійність обробки відеопотоків і коректність збереження даних.

Практична значущість роботи полягає у створенні автономної системи, що може бути впроваджена в житлових комплексах, ТЦ, бізнес-центрах тощо – без значних витрат на обладнання. Завдяки модульній архітектурі її легко адаптувати під конкретні умови чи потреби.

Робота базується на сучасних підходах до розробки серверних застосунків і використанні глибоких нейронних мереж для обробки зображень.

Результати дослідження і розробки були представлені на X Міжнародній науково-практичній конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві» (ІТОНВ-2025), м. Луцьк, 23-24 травня 2025 р. (додаток А).

РОЗДІЛ 1
АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ МОНІТОРИНГУ ПАРКОМІСЦЬ НА ОСНОВІ КОМП'ЮТЕРНОГО ЗОРУ

1.1 Аналіз сучасного стану проблеми

Сучасні міста все частіше стикаються з проблемами ефективної організації паркування. З кожним роком кількість автомобілів зростає, а наявні паркомісця стають обмеженим ресурсом. Згідно з дослідженнями Європейської комісії, в середньому 30 % міського трафіку складається з водіїв, які шукають вільне місце для паркування. Це призводить до перевантаження вулиць, підвищення рівня викидів CO_2, погіршення якості повітря та додаткових економічних витрат. У багатьох містах світу вже впроваджуються рішення, які дозволяють контролювати та оптимізувати використання паркомісць за допомогою сучасних цифрових технологій.

Важливо відзначити, що автоматизовані системи моніторингу паркомісць розвиваються завдяки досягненням у сфері комп'ютерного бачення, машинного навчання та інтернету речей (IoT). Наприклад, системи на базі датчиків індукції або магнітних сенсорів дозволяють визначати наявність автомобіля на місці, проте вони вимагають значних витрат на встановлення та обслуговування. Натомість рішення на базі відеоаналітики використовують наявні камери спостереження й дозволяють з мінімальними витратами впроваджувати систему моніторингу, спираючись на аналіз відеопотоків. Серед найпопулярніших алгоритмів обробки зображень можна виділити YOLO (You Only Look Once), який демонструє високу швидкість роботи та точність при виявленні об'єктів у реальному часі (рис. 1.1) [1].

Це робить відеоаналітику особливо привабливою для міського середовища, де вже існує інфраструктура камер і важлива оперативність реагування.

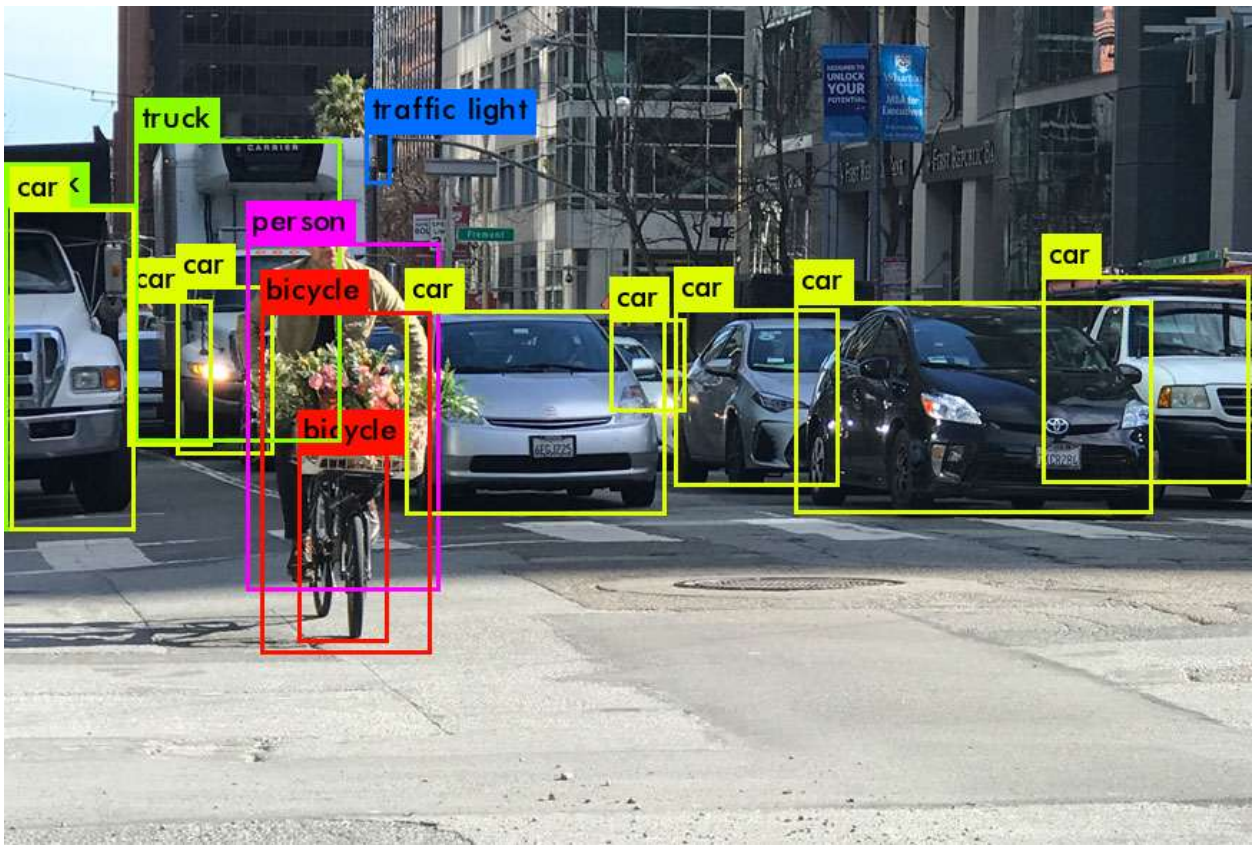

Рисунок 1.1 – Демонстрація роботи моделі комп’ютерного зору «YOLO»

YOLO є одним з найефективніших алгоритмів обробки зображень, який застосовується для виявлення об’єктів у режимі реального часу. Його ключова перевага полягає в здатності здійснювати розпізнавання за один прохід нейронної мережі, що дозволяє досягати надзвичайно високої швидкості аналізу кадрів без значної втрати точності. Алгоритм поділяє зображення на сітку та передбачає прямокутні рамки імовірних об’єктів, а також відповідні класи для кожної з них (рис. 1.2).

Завдяки такому підходу він є придатним для використання в системах, які потребують обробки великого обсягу відеоданих у реальному часі, таких як системи відеоспостереження на паркінгах.

Таким чином, вибір конкретної версії YOLO залежить від поставлених завдань і ресурсів, що дозволяє гнучко адаптувати систему під конкретні умови використання.

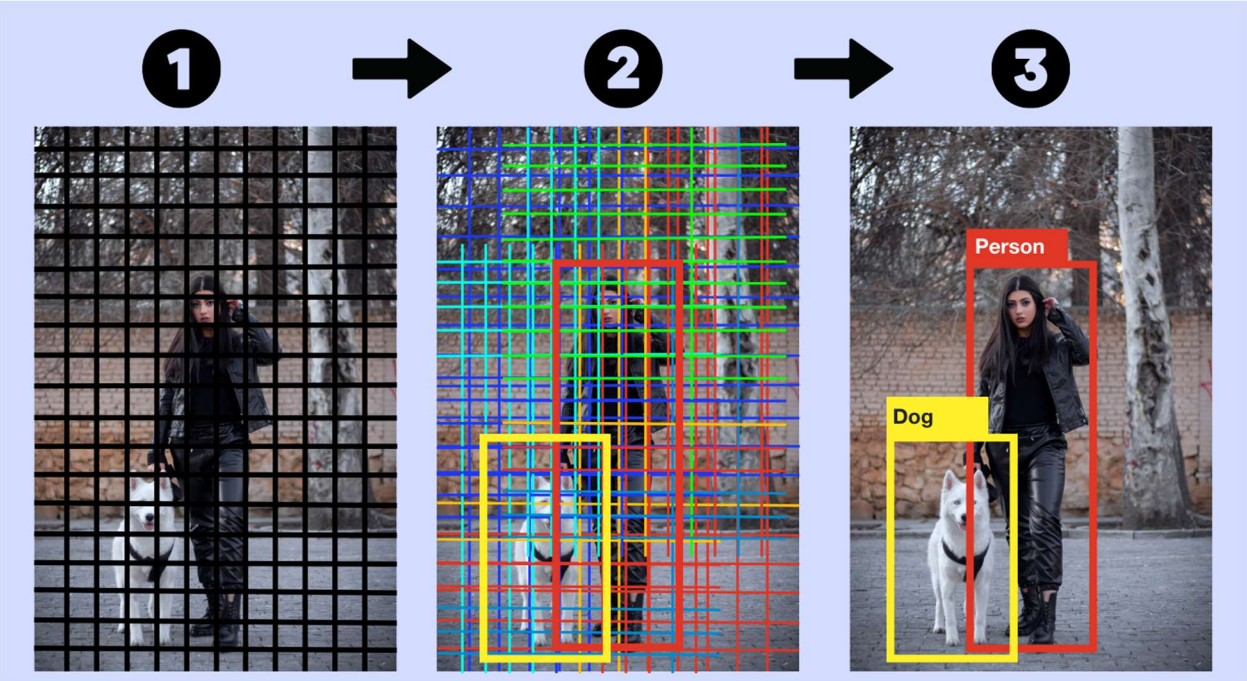

Рисунок 1.2 – Принцип роботи алгоритму «YOLO» [2]

З моменту своєї появи YOLO пройшов кілька значних етапів розвитку. Починаючи з YOLOv1, кожна нова версія демонструє суттєві покращення як у швидкодії, так і в точності. Зокрема, YOLOv3 і YOLOv4 додали підтримку багаторівневої детекції, що дозволило ефективно розпізнавати об’єкти різного розміру. Одним із найпоширеніших рішень на сьогодні є YOLOv5, який написаний на PyTorch і відзначається гнучкістю налаштування, високою продуктивністю та активною підтримкою спільноти розробників. Крім того, YOLOv5 підтримує різні розміри моделей (s, m, l, x), що дозволяє адаптувати систему під наявні обчислювальні ресурси – від мікроконтролерів до серверів із графічними процесорами.

Особливої уваги заслуговує поява модифікованих варіантів, таких як YOLOv7 і YOLO-NAS, що демонструють ще кращу точність при збереженні або навіть зменшенні часу інференсу. У YOLOv7 реалізовано низку архітектурних інновацій, включно з більш оптимізованою структурою блоків та поліпшеним механізмом обробки малих об’єктів, що є критичним у контексті моніторингу паркомісць, де автомобілі можуть частково перекриватися або перебувати під нестандартним кутом.

Окрім технічних характеристик, важливо зазначити, що YOLO легко інтегрується з сучасними програмними фреймворками, такими як OpenCV, TensorRT, NVIDIA DeepStream, що значно спрощує розгортання системи на пристроях з обмеженими ресурсами. Враховуючи високу ефективність та гнучкість у налаштуванні, саме цей алгоритм є оптимальним вибором для реалізації інтелектуальної системи виявлення зайнятості паркомісць, яка потребує як реальної швидкості, так і надійності в умовах реального міського середовища.

Огляд літературних джерел показує, що наразі ведуться активні дослідження в напрямку підвищення точності детекції, зниження вимог до обчислювальних ресурсів, а також покращення стійкості до змін зовнішніх умов, таких як освітлення, погодні фактори чи нестандартні кути огляду. Наприклад, у роботі Real-Time Vehicle Detection Based on Improved YOLO v5 запропоновано покращену версію YOLOv5 для виявлення транспортних засобів, яка застосовує Flip-Mosaic алгоритм, що дозволяє підвищити сприйняття малих об’єктів та знизити хибні спрацьовування [3]. У дослідженні MEB-YOLO представлено метод, який поєднує техніки Mosaic та MixUp для покращення навчання, а також використовує механізм уваги Efficient Channel Attention (ECA), що допомагає зменшити вплив нерелевантних ознак та покращує точність навіть у складних дорожніх умовах [4]. Крім того, у роботі Smart Parking with Pixel-Wise ROI Selection описано інтеграцію IoT, Edge Computing та глибокого навчання для побудови масштабованих розумних паркувальних систем, які використовують моделі YOLO для високоточного виявлення транспортних засобів у реальному часі [5]. Також вивчаються гібридні підходи, що поєднують дані з відеокамер і додаткових сенсорів для підвищення загальної надійності системи та мінімізації помилок у нестандартних ситуаціях.

Існуючі аналоги систем автоматизованого моніторингу паркомісць можна представити на прикладі «Parkinto» – комерційного рішення, що використовує комп’ютерний зір для виявлення зайнятості паркомісць (рис. 1.3) [6].

Рисунок 1.3 – Робота системи автоматизованого моніторингу паркомісць «Parkinto»

Система «Parkinto» аналізує відеопотоки з камер спостереження, визначаючи, які місця є вільними або зайнятими, і надає дані у реальному часі через API. Однією з ключових переваг цієї системи є можливість інтеграції з уже наявними камерами на об’єкті, що знижує витрати на додаткове обладнання. Крім того, «Parkinto» пропонує готове програмне забезпечення, підтримку клієнтів та зручну систему налаштувань (рис. 1.4). Водночас основним недоліком таких комерційних рішень є висока ціна, залежність від стороннього постачальника та обмежені можливості кастомізації системи під специфічні потреби користувача.

Саме тому розробка власної системи є доцільною: вона дозволяє створити адаптовану, гнучку систему, яка буде відповідати конкретним завданням та умовам, враховуючи локальні особливості та бюджетні обмеження.

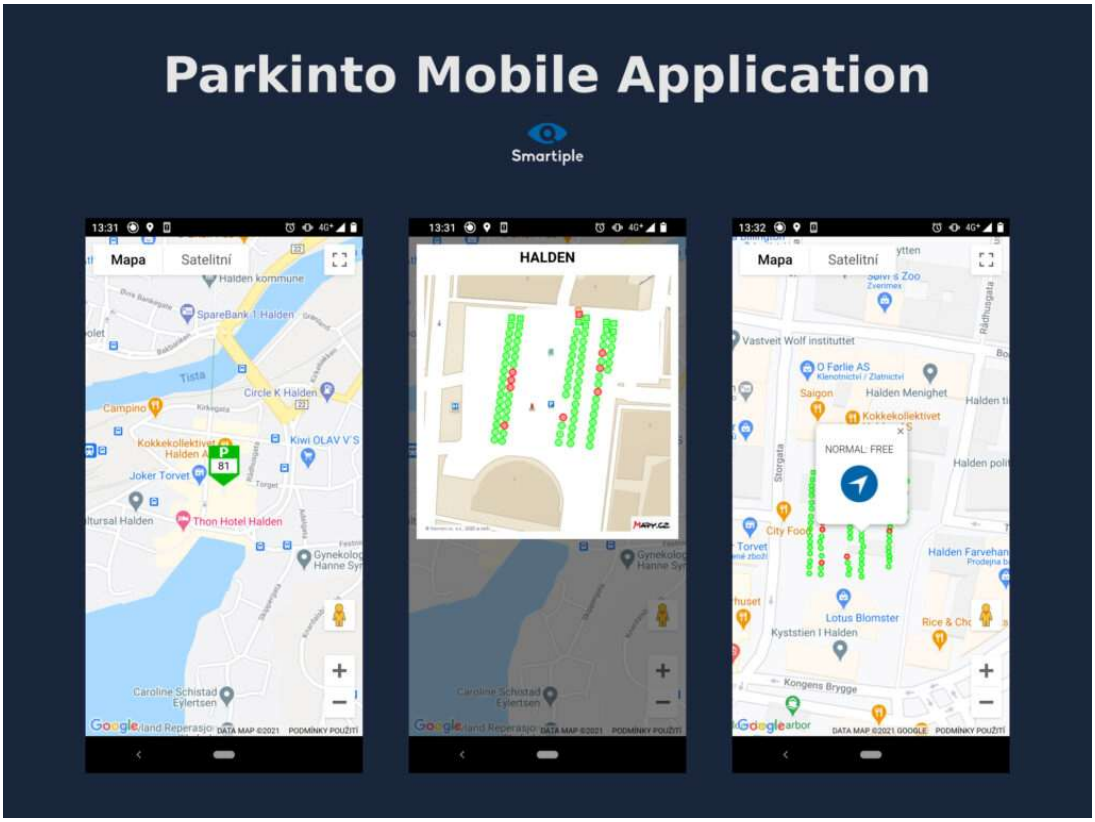

Рисунок 1.4 – Інтерфейс мобільного-додатку «Parkinto»

Власне, доцільність розробки нової системи полягає в тому, що існуючі рішення або занадто дорогі, або неадаптовані до специфічних умов окремого міста чи підприємства. Створення програмного забезпечення, що використовує YOLO для аналізу відеопотоків, дозволяє вирішити одразу кілька задач: автоматизувати моніторинг паркомісць, знизити витрати на обладнання (оскільки використовуються вже наявні камери), забезпечити локальне збереження даних у базі SQLite без передачі в хмару (що підвищує безпеку) та надати простий API для інтеграції з мобільними застосунками або міськими платформами.

Особливо актуальним є питання точності роботи системи. Багато аналогів мають проблему з помилковими спрацьовуваннями, коли, наприклад, тінь, сміття або люди на зображенні розпізнаються як автомобілі. Це знижує ефективність системи й призводить до недовіри користувачів. Тому в дипломній роботі планується провести тестування на реальних відеопотоках із камер різної якості, а також оптимізувати алгоритми для підвищення стійкості до шуму.

Також важливо врахувати аспект масштабування. Багато комерційних рішень погано масштабуються без значного зростання вартості, оскільки вимагають щомісячної оплати за кожну нову камеру або локацію. Запропонована в дипломі система розробляється як модульна, тобто вона може розширюватися під нові локації простим додаванням нових модулів без переписування основного коду. Це забезпечить її придатність як для малих підприємств (наприклад, бізнес-центрів), так і для великих паркінгів.

Таким чином, проведений аналіз літератури та існуючих аналогів дозволяє зробити висновок про актуальність і перспективність розробки нової системи моніторингу паркомісць. Така система не лише допоможе ефективніше використовувати паркомісця, а й стане частиною більшої концепції розумного міста, сприяючи зменшенню заторів та підвищенню якості життя мешканців.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

У межах кваліфікаційної роботи було поставлено завдання розробити систему автоматизованого моніторингу зайнятості паркомісць із застосуванням технологій комп’ютерного зору та обробки відеопотоків. Основна ідея полягає у створенні повнофункціонального серверного рішення, здатного самостійно обробляти дані, зберігати результати та надавати зручний механізм взаємодії через програмний інтерфейс керування.

Задля досягнення поставленої мети визначено такі конкретні завдання:

– здійснити детальний аналіз наявних методів контролю паркомісць, зосередившись на системах, які використовують відеоаналітику, машинне навчання або сенсори, з метою виявлення їх переваг, недоліків і можливостей для оптимізації;

– визначити загальну архітектуру програмного забезпечення, описати основні функціональні блоки, способи взаємодії між модулями, а також засоби забезпечення цілісності та актуальності даних;

– реалізувати алгоритм автоматичного виявлення транспортних засобів на зображеннях, отриманих із камер, використовуючи модель YOLO, адаптовану для роботи в умовах реального часу;

– створити систему зберігання інформації про паркомісця, їх координати, камери спостереження, статуси та відповідні налаштування, на основі використання локальної бази даних SQLite;

– розробити повноцінний інтерфейс керування системою, що забезпечить можливість створення, оновлення, перегляду та видалення даних про паркомісця та інші ключові об’єкти системи;

– забезпечити підтримку обробки відеопотоків за допомогою протоколу RTSP, з можливістю періодичного захоплення зображень та обробки для визначення зайнятості місць;

– реалізувати механізм оновлення статусів паркомісць;

– забезпечити внутрішню гнучкість і модульність системи, що дозволить за потреби адаптувати логіку обробки, схему бази чи способи взаємодії без глобальних змін;

– провести комплексне тестування системи на коректність роботи, стабільність обробки потоків, надійність збереження даних і точність визначення зайнятих місць.

Реалізація вищезазначених завдань дозволить створити стабільну, автономну систему, що не залежить від зовнішніх клієнтів і забезпечує повний контроль за процесом моніторингу паркувальних місць.

Висновки до розділу 1

У результаті аналізу сучасного стану проблеми виявлено ключові особливості та напрямки розвитку автоматизованих систем моніторингу паркомісць. Особливу увагу було приділено рішенням, що використовують відеоаналітику та методи комп’ютерного зору, як найбільш перспективним у контексті міського середовища. Було встановлено, що більшість наявних рішень

обмежені у гнучкості налаштувань, не мають зручного механізму інтеграції або не дозволяють ефективно масштабувати систему.

Аналіз технічних підходів та архітектур показав доцільність використання модульної серверної архітектури з поділом відповідальності між компонентами: обробкою відео, збереженням даних та керуванням логікою взаємодії. Визначено, що застосування нейронної мережі YOLO для розпізнавання транспортних засобів є ефективним рішенням для задачі детекції в реальному часі.

Постановка завдання дозволила чітко окреслити цілі розробки, сформувати вимоги до ключових модулів системи та встановити перелік задач, необхідних для реалізації повнофункціонального серверного програмного забезпечення.

Таким чином, у першому розділі обґрунтовано актуальність створення сучасної автоматизованої системи для моніторингу паркомісць з використанням методів комп’ютерного зору, а також визначено технічну основу для подальшої реалізації проєкту.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

У сучасних умовах урбанізації та зростання кількості автомобільного транспорту значною проблемою стає раціональне використання паркувального простору. Традиційні методи організації паркування не дозволяють ефективно відслідковувати зайнятість місць у режимі реального часу, що призводить до заторів, витрат часу та незадоволеності водіїв. Рішенням цієї проблеми є впровадження інтелектуальних систем моніторингу паркомісць, які базуються на використанні сучасних технологій комп'ютерного зору, аналізу відеопотоків та автоматичної фіксації статусу паркувальних зон.

У межах кваліфікаційної роботи було поставлено завдання створити серверну систему моніторингу зайнятості паркомісць, яка б дозволяла визначати статус кожного місця на основі зображення з відеопотоку. Всі ключові процеси – від обробки зображень до керування зоною паркування – реалізовані у вигляді взаємопов'язаних модулів, об'єднаних єдиною архітектурою.

Формування вимог до розроблюваної системи здійснювалося шляхом:

– аналізу існуючих рішень у сфері Smart Parking;

– вивчення наукових публікацій з автоматичного розпізнавання об'єктів у транспортній сфері [7];

– дослідження практичного досвіду розробки відеоаналітичних систем;

– аналізу специфіки роботи з RTSP-потоками та їх сумісності з алгоритмами обробки зображень.

На основі проведеного аналізу були сформульовані функціональні та нефункціональні вимоги до системи.

Функціональні вимоги:

– підключення до IP-камер із підтримкою протоколу RTSP та отримання зображень із заданою частотою;

– вибір версії моделі детекції, для оптимізації точності або швидкодії;

– автоматичне визначення наявності автомобілів у межах заданих координат паркомісць за допомогою нейромережевої моделі YOLO;

– підтримка довільної форми паркомісць, які описуються чотирма координатними точками (p1-p4), що дозволяє створювати трапеції, ромби, «ялинки» та інші форми;

– збереження інформації про камери, місця, зображення та статуси у базі даних SQLite;

– можливість додавання, редагування та видалення паркомісць і камер через зручний API-інтерфейс;

– інтерфейс взаємодії (API) для зовнішніх систем або клієнтських додатків з можливістю повного керування всіма об’єктами системи.

Нефункціональні вимоги:

– стабільність роботи системи у фоновому режимі (режим 24/7);

– можливість масштабування системи без зміни її ядра;

– мінімальні системні вимоги для розгортання, включаючи підтримку роботи на базі Linux-сервера;

– незалежність від фронтенду, що дозволяє реалізовувати різноманітні візуальні інтерфейси на основі єдиного API.

Архітектура розроблюваної системи ґрунтується на модульному підході, який забезпечує чіткий розподіл відповідальності між компонентами та сприяє масштабованості й зручності підтримки проєкту. Основною ідеєю архітектури є побудова автономного серверного рішення, яке може працювати у фоновому режимі та виконувати всі необхідні функції без потреби у зовнішніх клієнтських обробниках.

Ключовим елементом системи виступає модуль обробки відеопотоків, який відповідає за підключення до IP-камер, отримання зображень через протокол RTSP та їх подальше збереження у вигляді снапшотів. Отримані кадри надсилаються до модуля комп’ютерного зору, де зображення обробляються за

допомогою попередньо натренованої моделі YOLO, яка виконує детекцію об’єктів у режимі реального часу (рис 2.1).

Рисунок 2.1 – Архітектурна схема системи моніторингу паркомісць

Далі ці координати передаються до модуля аналізу зайнятості, який порівнює їх із геометричними зонами паркомісць, заданих у базі даних. Якщо хоча б один із виявлених об’єктів перетинає межі певного місця, система фіксує

статус «зайнято», в іншому випадку – «вільно». Усі результати зберігаються у локальній базі даних.

Особливістю архітектури є використання чітко визначеного програмного інтерфейсу для взаємодії з усіма ключовими об’єктами. API дозволяє додавати, редагувати або видаляти об’єкти системи, запускати процеси обробки вручну та отримувати актуальний статус кожного місця. Це забезпечує повну незалежність від конкретної реалізації клієнтської частини – інтерфейс взаємодії універсальний та може бути використаний у мобільному, веб або десктопному застосунку.

Загалом обрана архітектурна модель відповідає сучасним принципам розробки інформаційних систем: модульність, відкритість, низький поріг інтеграції з іншими сервісами та надійність у процесі тривалої експлуатації програмного продукту (рис. 2.2).

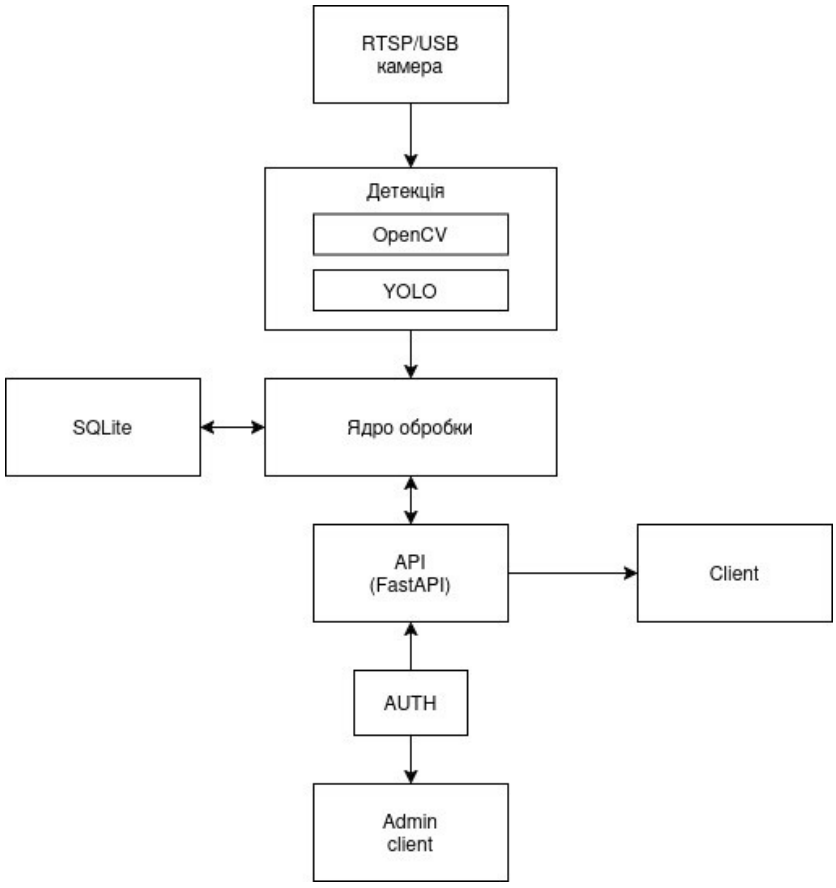

Рисунок 2.2 – Архітектурна схема системи моніторингу паркомісць

Кожне паркомісце описується чотирма точками: p1(x1, y1), p2(x2, y2), p3(x3, y3), p4(x4, y4). Такий підхід дозволяє визначити довільну геометричну форму, що робить систему адаптивною до різних схем паркування (рис. 2.3).

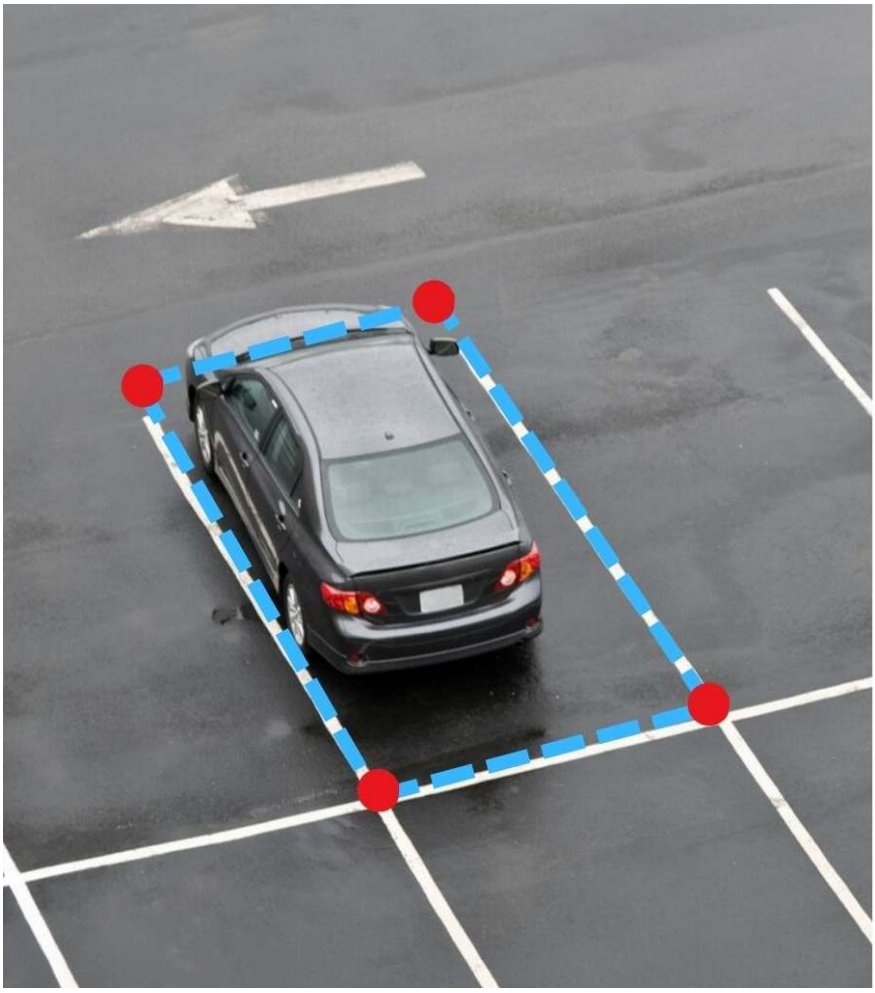

Рисунок 2.3 – Приклад представлення паркомісця у форматі опису точками

Міжмодульна взаємодія здійснюється за внутрішніми сигналами системи, а зовнішня – через API. Це дозволяє підтримувати чітку розділеність логіки та забезпечує розширюваність.

Розроблене технічне рішення передбачає гнучке та незалежне управління всіма процесами моніторингу паркомісць. API дозволяє створювати на основі бекенду будь-який тип клієнтських інтерфейсів без потреби змінювати логіку самої системи. Такий підхід забезпечує універсальність, незалежність і відкритість системи для подальшої інтеграції.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Вибір мови програмування та серверного середовища є одним з ключових аспектів побудови ефективної, стабільної та масштабованої програмної системи. У процесі розробки автоматизованої системи для моніторингу зайнятості паркомісць було обрано мову Python та фреймворк FastAPI як основу серверної частини. Це рішення продиктовано низкою технічних, архітектурних та практичних міркувань, які дозволяють реалізувати вимоги до системи з високим рівнем ефективності.

Python є мовою загального призначення з динамічною типізацією, широким набором бібліотек та активною спільнотою. Її популярність у сфері розробки вебсервісів, аналізу даних та комп'ютерного зору робить її універсальним інструментом, здатним задовольнити потреби повного стеку серверної логіки – від обробки запитів до аналізу зображень.

Окремо варто підкреслити, що Python має гнучкий підхід до інтеграції сторонніх модулів, зокрема таких як OpenCV, Torch, SQLAlchemy та інші. Це дає змогу реалізовувати логіку у вигляді окремих незалежних блоків, що легко тестуються, масштабуються та повторно використовуються.

Ще однією перевагою Python є висока швидкість розробки. Завдяки лаконічному синтаксису та широкому використанню бібліотек, більшість задач вирішуються в декілька разів швидше, ніж при використанні компільованих мов, таких як Java або C++. У рамках дипломного проєкту це дозволяє зосередитися на логіці роботи системи, а не на низькорівневих деталях.

Серед можливих альтернатив Python можна назвати:

– Node.js: має сильну екосистему, але потребує глибшого контролю за асинхронністю (callback hell, race conditions) і складніший механізм валідації;

– Go: забезпечує високу продуктивність, але вимагає більшого обсягу коду навіть для простих CRUD-запитів, а також не має достатньої гнучкості у використанні бібліотек комп'ютерного зору;

– Java: надає потужні можливості у сфері ентерпрайз-рішень, однак для невеликої системи моніторингу її складність і потреба в надлишковій конфігурації є недоцільними.

Отже, Python забезпечує баланс між швидкістю розробки, зручністю підтримки та можливістю використання спеціалізованих бібліотек – що робить його найбільш придатним для реалізації системи такого типу.

Серверна частина системи розроблялася з урахуванням необхідності побудови ефективного, надійного та масштабованого REST API. Після аналізу кількох популярних фреймворків, таких як Flask, Django, Tornado та FastAPI, було прийнято рішення на користь останнього – FastAPI, що є одним із найсучасніших рішень для Python-екосистеми [8].

Однією з ключових причин вибору FastAPI стала його повна підтримка асинхронного програмування, що дозволяє реалізувати неблокуючу обробку HTTP-запитів. Це особливо актуально для розроблюваної системи, яка передбачає паралельну обробку відеопотоків, взаємодію з базою даних та відповіді на API-запити. Асинхронність дозволяє суттєво зменшити час очікування при високому навантаженні, що важливо для систем реального часу.

Крім того, FastAPI надає вбудовані інструменти для автоматичної генерації OpenAPI-специфікації, яка реалізується через інтерактивний інтерфейс Swagger UI. Це суттєво спрощує тестування API та взаємодію з ним під час подальшого розвитку клієнтської частини або інтеграції в сторонні системи. Автоматична документація створюється без додаткових зусиль – достатньо лише типізувати параметри функцій і відповіді, що особливо зручно під час командної розробки або передачі проєкту іншим розробникам [9].

Ще однією важливою перевагою фреймворку є глибока інтеграція з бібліотекою Pydantic, яка відповідає за валідацію вхідних даних. Це дозволяє забезпечити коректність запитів ще до їхньої обробки, зменшуючи кількість помилок і підвищуючи безпеку API. Усі структури даних визначаються у вигляді моделей, що надає чітке уявлення про очікувані формати та спрощує супровід коду.

Загалом, FastAPI демонструє високу продуктивність і здатен обробляти тисячі запитів за секунду. За результатами незалежних порівняльних тестувань, він займає провідні позиції серед Python-фреймворків за швидкодією, поступаючись лише низькорівневим реалізаціям на C або Go. Водночас, він зберігає простоту використання, властиву Python-екосистемі, що робить його оптимальним вибором для проєктів, у яких важлива і швидкість, і гнучкість розробки.

Одним із ключових завдань системи моніторингу зайнятості паркомісць є забезпечення стабільного отримання відеопотоків від камер спостереження у реальному часі та їх обробка з метою аналізу зайнятості місць. Це вимагає використання перевірених протоколів передавання мультимедійних даних і відповідного інструментарію для захоплення та декодування відеоінформації. У процесі розробки було обґрунтовано вибір протоколу RTSP (Real Time Streaming Protocol) та бібліотеки OpenCV як основних технологій для реалізації цього етапу [10].

Даний протокол є галузевим стандартом для передавання аудіо та відеопотоків у реальному часі. Його активно використовують IP-камери, DVR-системи та медіасервери. Основною перевагою є можливість передавання медіаконтенту з мінімальною затримкою, що робить його надзвичайно зручним для задач моніторингу, де критичним є своєчасне реагування на зміни в полі зору камери.

RTSP підтримує механізми керування потоком (наприклад, SETUP, PLAY, PAUSE, TEARDOWN), що дозволяє точно налаштовувати сесії зв’язку з камерами. Більшість сучасних моделей камер (Hikvision, Dahua, Reolink тощо) мають попередньо налаштовану підтримку цього протоколу, що забезпечує сумісність без додаткових драйверів або проміжного програмного забезпечення.

Крім того, RTSP працює поверх протоколу TCP або UDP, що дає змогу балансувати між надійністю передавання та швидкістю. Для задач моніторингу паркомісць, де ключовим є отримання зображення у найсвіжішому стані, зазвичай обирається передача поверх UDP, яка забезпечує мінімальну затримку.

Бібліотека OpenCV (Open Source Computer Vision Library) є одним із найпопулярніших інструментів для комп'ютерного зору та обробки зображень. Вона підтримує широкий спектр функцій – від захоплення відео до попіксельного аналізу та застосування нейронних мереж.

У рамках реалізації системи бібліотека OpenCV використовується для підключення до RTSP-потоку та вибірки окремих кадрів із нього. Завдяки цьому, забезпечується безперервне зчитування зображення у реальному часі [11].

Після ініціалізації потоку, кожен кадр обробляється окремо в циклі, що дозволяє впровадити необхідну логіку – масштабування зображення, фільтрацію шумів, попередню обробку перед передачею до детектора об'єктів YOLO.

OpenCV також підтримує конвертацію форматів кольору (наприклад, з BGR у RGB), що є критично важливим для сумісності з Tensor-based моделями, які вимагають відповідного порядку каналів. Крім того, можлива адаптація кадрів під задані розміри, що зменшує навантаження на модель детекції та пришвидшує загальний процес аналізу.

Попри широкі можливості OpenCV, у процесі дослідження було також розглянуто інші інструменти, зокрема FFmpeg та GStreamer. FFmpeg має вищу продуктивність при потоковій трансляції, однак його API значно менш зручне для роботи з окремими кадрами у Python, що ускладнює реалізацію покадрового аналізу. GStreamer, у свою чергу, є потужним інструментом для медіаобробки, однак вимагає складної конфігурації та не має інтеграції з типовими бібліотеками Python для комп'ютерного зору.

Таким чином, вибрана бібліотека забезпечує оптимальний баланс між функціональністю, простотою інтеграції та продуктивністю. Його використання дозволяє уникнути надмірної складності й зосередитися на реалізації логіки визначення зайнятості паркомісць.

Найважливішим компонентом розроблюваної системи є механізм виявлення автомобілів на паркувальному майданчику за допомогою зображення. Це завдання належить до області комп'ютерного зору, а саме – до проблеми детекції об'єктів на зображенні. Серед сучасних алгоритмів, що вирішують

подібні задачі, особливе місце займають нейронні мережі на основі архітектури YOLO (You Only Look Once). У даній системі було використано модель YOLOv5, яка поєднує високу точність із відносно низькими обчислювальними витратами, що дозволяє застосовувати її навіть на звичайному CPU-сервері.

YOLO – це одноетапний підхід до детекції об’єктів, у якому все зображення обробляється лише один раз за допомогою згорткової нейронної мережі. На відміну від двоетапних методів (наприклад, R-CNN або Faster R-CNN), де спочатку генеруються регіони інтересу, а потім класифікуються, YOLO миттєво видає координати об’єктів та їхні класи в межах єдиного проходу мережі. Це дозволяє значно зменшити час обробки, що є критично важливим [12].

YOLOv5 є реаліацією цього підходу, яка написана повністю на мові Python з використанням фреймворку PyTorch. Серед її переваг можна виокремити модульну структуру, можливість тонкого налаштування під різні сценарії, підтримку декількох рівнів складності (моделі nano, small, medium, large), а також наявність готових скриптів для тренування, інференсу та оптимізації.

Модель розбиває зображення на сітку й для кожної комірки прогнозує прямокутник обмеження (bounding box), клас об’єкта та ймовірність належності до цього класу. Після цього результати фільтруються за порогом впевненості (confidence threshold) і методом подавлення надмірних накладень (Non-Maximum Suppression, NMS).

Також варто зазначити, що модель є повністю відкритою (open-source) і активно підтримується спільнотою. Це дозволяє вільно адаптувати її під потреби системи.

У якості альтернатив було розглянуто такі алгоритми, як SSD (Single Shot Detector) та Faster R-CNN. Перший також є одноетапним методом, однак у більшості випадків поступається YOLO за точністю детекції при схожій швидкості. Faster R-CNN, натомість, демонструє вищу точність на складних сценах, але має значно більший час обробки одного кадру, що робить його непридатним для використання в системі реального часу без GPU.

Окрім цього, YOLOv5 має зручні API та інтегрується з бібліотекою TorchHub, що дозволяє швидко запускати інференс без необхідності компілювати модель або створювати окремі сервіси для її виклику.

На перших етапах проєктування розглядалося класичне представлення паркомісця у вигляді прямокутника, заданого координатами верхнього лівого кута (x, y) та розмірами (width, height). Такий формат є типовим для багатьох графічних API, однак він суттєво обмежує можливість адаптації під реальні умови паркування. У багатьох випадках місця розташовані під кутом, зі зміщенням перспективи або на вигнутих ділянках. Відповідно, застосування прямокутної моделі призводить до помилкових результатів при перетині із зонами присутності автомобіля.

Заміна фіксованого прямокутника на набір чотирьох довільних точок дала змогу відмовитися від обмежень класичних прямолінійних структур і перейти до геометрично точного моделювання контурів паркомісця.

У координатному просторі зображення паркомісце визначається через послідовні точки p1, p2, p3, p4, які описують його контур у напрямку за годинниковою стрілкою. Усі координати зберігаються у пікселях відносно розміру кадру, що уніфікує обробку та дозволяє застосовувати одні й ті самі алгоритми незалежно від роздільної здатності камери.

Завдяки представленню зон у вигляді чотирикутників, система отримує змогу використовувати методи комп’ютерної геометрії для перевірки перетинів між полігонами. Зокрема, для визначення, чи перетинається виявлений об’єкт з певним місцем, застосовується алгоритм перевірки перетину двох багатокутників, зведений до знаходження спільної області.

Реалізація формату p1-p4 також дозволяє зручно візуалізувати паркомісця в HTML-інтерфейсі адміністратора. Кожна точка може бути вільно переміщена користувачем на полотні, а контур зони автоматично оновлюється у реальному часі. Така форма представлення суттєво покращує ергономіку системи та спрощує налаштування нових камер.

Завдяки відсутності жорсткої прив’язки до координатної сітки або прив’язки до базових прямокутників, адміністратор системи має повний контроль над точним положенням і формою кожної зони, що дозволяє адаптувати систему під будь-які паркувальні площі – від типових відкритих майданчиків до паркінгів зі складною розміткою.

Сучасні інформаційні системи, особливо ті, що орієнтовані на багатокомпонентну або мікросервісну архітектуру, потребують стандартизованого способу взаємодії з іншими елементами – як внутрішніми, так і зовнішніми. У межах реалізованої системи моніторингу зайнятості паркомісць таким способом стала реалізація RESTful API – інтерфейсу прикладного програмування, який базується на принципах REST (Representational State Transfer).

Вибір архітектури REST продиктований її простотою, сумісністю з більшістю мов і платформ, а також логічною структурованістю. Завдяки використанню HTTP-протоколу як базового транспортного рівня, REST API не потребує окремих каналів комунікації та легко інтегрується у вже існуючу інфраструктуру.

REST-архітектура передбачає побудову інтерфейсу за принципами CRUD (Create, Read, Update, Delete), при цьому кожна сутність системи представлена у вигляді ресурсу.

Кожен ресурс має унікальний URL-ідентифікатор, а взаємодія з ним відбувається через стандартизовані HTTP-методи:

– GET – для отримання даних;

– POST – для створення нових сутностей;

– PUT або PATCH – для оновлення;

– DELETE – для видалення.

Цей підхід дозволяє мінімізувати складність API і забезпечити інтуїтивно зрозумілу структуру маршрутизації, що, в свою чергу, спрощує роботу як фронтенд-розробників, так і сторонніх інтеграторів.

З міркувань безпеки, API реалізовано з урахуванням механізмів аутентифікації. Це дозволяє контролювати доступ до критичних операцій. Усі запити до API можуть бути обгорнуті в HTTPS, що забезпечує захищену передачу даних у мережах загального доступу.

Однією з ключових вимог до сучасних інформаційних систем є їхня здатність адаптуватися до змін навколишнього середовища – як технічного, так і функціонального. У контексті розробки системи моніторингу зайнятості паркомісць це означає необхідність створення такої архітектури, яка забезпечує гнучкість конфігурації, простоту розширення функціональності та масштабованість без критичних змін у кодовій базі.

Основним принципом, що дозволяє досягти високої гнучкості, є модульна побудова програмного забезпечення. У розробленій системі реалізовано чітке логічне розділення компонентів, серед яких:

- модуль обробки відеопотоків;
- модуль детекції об’єктів (YOLO);
- модуль перевірки зайнятості місць;
- API-контролери;
- база даних і модуль доступу до неї;
- сервіс знімків та архівування подій;
- конфігураційний менеджер.

Кожен з цих компонентів може розвиватися незалежно, не порушуючи цілісності всієї системи. Такий підхід значно полегшує додавання нових функцій (наприклад, підтримки нових типів датчиків або режимів аналізу), а також дозволяє легше проводити тестування, відлагодження та рефакторинг.

Всі змінні параметри, що впливають на логіку роботи системи, винесено у конфігураційний шар, який включає як файл середовища .env для зберігання зовнішніх змінних, так і модуль config.py, що забезпечує централізоване зчитування та доступ до цих параметрів у програмній логіці.

Фундаментом для надійної, стабільної роботи системи моніторингу паркомісць є коректно спроєктована та реалізована база даних. Її структура

визначає не лише ефективність зчитування і запису даних, але й загальну логіку функціонування всієї системи: від перевірки зайнятості місця до генерації статистичних звітів.

З огляду на потреби у простоті розгортання, автономності, швидкодії при середньому навантаженні та відсутності необхідності в масштабуванні на перших етапах, було обрано використання SQLite – вбудованої реляційної СУБД [13].

Оскільки структура даних у системі чітко визначена й рідко змінюється в процесі роботи, реляційна модель є оптимальною [14].

Структура бази даних системи побудована на основі SQLAlchemy ORM та включає такі основні сутності:

– Camera – таблиця з інформацією про відеокамери, підключені до системи. Містить унікальний ідентифікатор, назву камери, посилання на джерело відеопотоку (source), а також пов’язана з відповідними паркомісцями;

– ParkingPlace – таблиця, що описує паркомісця. Вона містить координати чотирьох точок (p1-p4), які формують контур місця, а також посилання на групу та камеру, до якої це місце належить. Назва місця (name) дозволяє легко ідентифікувати його під час обробки або візуалізації;

– ParkingGroup – логічне об’єднання місць паркування. Кожна група має унікальну назву, набір пов’язаних місць, а також статус (status), що може відображати агрегований стан всієї групи;

– ParkingLayout – окрема сутність, яка дозволяє задавати загальні контури або зони паркування (наприклад, зонування поверхів, ділянок тощо). Також складається з координат чотирьох точок (p1-p4) і може бути пов’язана з певною групою;

– User – таблиця користувачів системи з авторизацією через ім’я користувача (username) та захешований пароль (hashed_password).

Висновки до розділу 2

У другому розділі було здійснено ґрунтовне формулювання вимог до системи моніторингу зайнятості паркомісць, визначено концептуальну архітектуру та обґрунтовано вибір програмних засобів, методів і технологій, що стали основою для реалізації системи.

Перш за все, розглядалися загальні вимоги до проєкту – як функціональні, так і нефункціональні. Було проаналізовано, які можливості повинна забезпечувати система і на основі цього було сформульовано перелік задач, а також закладено вимоги до структури даних та логіки взаємодії компонентів.

Було обґрунтовано вибір мови програмування Python як зручного, потужного та універсального інструменту для реалізації серверної частини з високим рівнем підтримки бібліотек комп'ютерного зору, асинхронності та веброзробки. Особлива увага приділялася фреймворку FastAPI, який забезпечує високу швидкодію, автоматичну генерацію документації, гнучкість маршрутизації запитів і глибоку інтеграцію з ORM та валідацією даних.

У розділі також було розглянуто способи підключення камер через протокол RTSP, які забезпечують універсальність і сумісність з більшістю моделей відеокамер. Для аналізу відеопотоків обрано бібліотеку OpenCV, яка поєднує в собі продуктивність та функціональність. У якості алгоритму обробки зображень та детекції об'єктів розглядалася модель YOLO – одна з найефективніших у задачах реального часу.

Значну увагу приділено розробці REST API, що є основною точкою взаємодії з клієнтськими інтерфейсами. Була описана логіка побудови запитів, обробки даних та реалізації безпечної авторизації.

Архітектура системи була сформована як модульна, що дає змогу легко масштабувати рішення у випадку збільшення кількості камер, паркомісць або груп. Дані принципи дозволяють ефективно розширювати функціональність системи без порушення вже реалізованої логіки.

Також було обґрунтовано вибір СУБД SQLite для зберігання структурованих даних. Вона відповідає вимогам системи щодо локального розгортання, невеликих обсягів даних та швидкої ініціалізації, забезпечуючи гнучкість у доступі до інформації.

Таким чином, результати цього розділу створили міцне підґрунтя для подальшої реалізації функціональної частини системи.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ МОНІТОРИНГУ ПАРКОМІСЦЬ НА ОСНОВІ КОМП'ЮТЕРНОГО ЗОРУ З ВИКОРИСТАННЯМ МОВИ PYTHON ТА АЛГОРИТМУ YOLO

3.1 Практична реалізація об'єкта проєктування

Проєкт системи моніторингу зайнятості паркомісць реалізовано як серверний застосунок на мові програмування Python із використанням фреймворку FastAPI. Основною архітектурною особливістю є чіткий поділ на окремі логічні модулі відповідно до функціонального призначення (рис. 3.1).

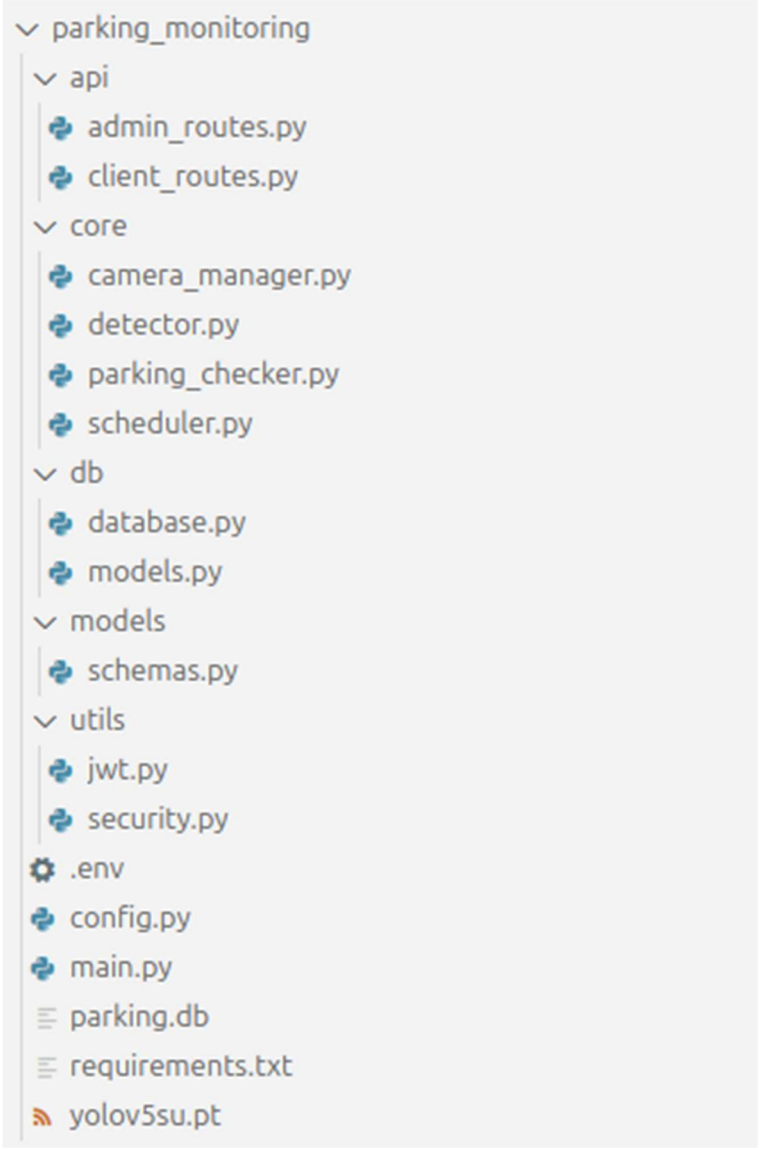

Рисунок 3.1 – Структура директорій проєкту

Проєкт побудований за принципом модульності: кожна відповідальність винесена в окремий файл, що полегшує налагодження, тестування та масштабування. Доступ до функціональності реалізовано через API-методи, згруповані відповідно до ролей.

Один із ключових елементів реалізованої системи – модуль підключення до камер відеоспостереження, який відповідає за захоплення кадрів з IP-камер або локальних джерел відеосигналу. Робота з відеопотоками здійснюється за допомогою бібліотеки OpenCV, яка дозволяє зчитувати зображення з різних типів джерел, зокрема RTSP-протоколу.

Функціональність зосереджена в класі CameraManager, що реалізує підключення до камери, буферизацію зображення, періодичне зчитування кадрів, а також обробку можливих помилок.

Модуль має наступну логіку:

– ініціалізація об’єкта із зазначенням джерела відеопотоку та інтервалу між кадрами;

– підключення до камери через метод connect() із перевіркою успішності відкриття потоку;

– зчитування кадру методом get_frame(), який також реалізує механізм очищення буфера (через CAMERA_BUFFER_READS);

– збереження останнього валідного кадру у полі last_frame;

– метод release() закриває з’єднання після завершення роботи.

Кодова реалізація компонента CameraManager наведена у лістингу 3.1.

Лістинг 3.1 – Компонент CameraManager

```
def get_frame(self):
    current_time = time.time()
    if (current_time - self.last_capture_time) >= self.capture_interval:
        for _ in range(CAMERA_BUFFER_READS):  # Buffer clearance
            ret, frame = self.cap.read()
            if not ret:
                raise RuntimeError("Failed to read the frame from the
camera")

        self.last_frame = frame
```

```
            self.last_capture_time = current_time
        return self.last_frame
```

кінець лістингу 3.1

Метод get_frame() реалізує принцип зчитування кадрів лише з певною періодичністю (для зменшення навантаження), та використовує цикл очищення буфера, щоб уникнути затримок при відображенні потоку. Це важливо для забезпечення актуальності даних при аналізі зайнятості місць.

При виявленні помилок у роботі камери (відсутність підключення або невдале зчитування кадру) система генерує винятки, які можуть бути оброблені на рівні планувальника або логера, що забезпечує стійкість та діагностованість системи.

Визначення наявності автомобілів у полі зору камери є центральною задачею системи. Для її реалізації використано глибоку згорткову нейронну мережу на основі моделі YOLOv5, яка дозволяє проводити обробку зображень у режимі реального часу [15].

Модуль detector.py побудований на базі бібліотеки Ultralytics YOLO, яка забезпечує швидку інтеграцію моделей без потреби в ручному конфігуруванні. Для підвищення гнучкості реалізовано параметризоване завантаження моделі: передається шлях до вагів (yolov5su.pt) та тип обчислювального пристрою (cpu або cuda).

Ініціалізація класу Detector виконує завантаження нейромережі та її прив’язку до відповідного пристрою. Основна функція – метод detect() – приймає зображення у вигляді масиву NumPy, обробляє його за допомогою YOLOv5 і повертає список об’єктів, що ідентифіковані як транспортні засоби.

Кодова реалізація методу детекції транспортних засобів у лістингу 3.2.

Лістинг 3.2 – Метод детекції транспортних засобів

```
def detect(self, frame):
    results = self.model(frame, verbose=False)
    cars = []

    for result in results:
        boxes = result.boxes
```

```
        for i in range(len(boxes)):
            cls_id = int(boxes.cls[i])
            class_name = self.model.names[cls_id]
            if class_name in VALID_VEHICLE_CLASSES:
                bbox = [int(coord) for coord in boxes.xyxy[i].tolist()]
                confidence = float(boxes.conf[i])
                cars.append({
                    'label': class_name,
                    'confidence': confidence,
                    'bbox': bbox
                })
    return cars
```

кінець лістингу 3.2

Під час детекції метод фільтрує лише ті класи, які відповідають списку VALID_VEHICLE_CLASSES. Кожен знайдений об’єкт представлений словником із назвою класу, координатами обмежувального прямокутника (bounding box) та значенням довіри (confidence).

Це дозволяє формувати чіткий набір вхідних даних для подальшого аналізу зайнятості паркомісць.

Визначення факту зайнятості паркомісць є ключовим етапом функціонування системи моніторингу. Модуль parking_checker.py реалізує алгоритм перевірки перетину детектованих транспортних засобів із координатами заданих паркомісць.

Система працює з набором чотирикутників, що описують кожне місце паркування у вигляді чотирьох довільних точок (p1-p4). Це дозволяє підтримувати різні геометричні конфігурації (ромб, паралелограм тощо).

Головний метод класу check_occupancy() приймає список результатів об’єктного детектора YOLO, а саме координати знайдених транспортних засобів, і визначає, чи перетинає хоча б один із них задану зону паркування.

Кодова реалізація перевірки зайнятості паркомісць у лістингу 3.3.

Лістинг 3.3 – Метод перевірки зайнятості паркомісць

```
def check_occupancy(self, detections):
    results = []
    for place in self.parking_places:
```

```
        points = place['points']
        occupied = False
        for detection in detections:
            if self._polygon_intersects_box(points, detection['bbox']):
                occupied = True
                break
        results.append({'id': place['id'], 'occupied': occupied})
    return results
```

кінець лістингу 3.3

Кожна паркувальна зона перевіряється на перетин із bounding box автомобіля за допомогою допоміжного методу _polygon_intersects_box(). Цей метод реалізує комбіновану перевірку:

– чи є хоча б одна вершина багатокутника всередині прямокутника;

– чи знаходиться хоча б одна вершина прямокутника всередині полігону;

– чи перетинаються хоча б одні з граней полігону та прямокутника.

Це забезпечує більш точне визначення колізій навіть для складних форм. Кодова реалізація у лістингу 3.4.

Лістинг 3.4 – Перевірка перетину полігону з прямокутником

```
deb _polygon_intersects_box(poly, rect):
    # """poly: [{'x': x, 'y': y}, ...]; rect: [xmin, ymin, xmax, ymax]"""
    for pt in poly:
        if xmin <= pt['x'] <= xmax and ymin <= pt['y'] <= ymax:
            return True
    for pt in rect_points:
        if point_in_poly(pt['x'], pt['y'], poly):
            return True
    for e1 in edges(poly):
        for e2 in edges(rect_points):
            if segments_intersect(e1[0], e1[1], e2[0], e2[1]):
                return True
    return False
```

кінець лістингу 3.4

Таким чином, реалізований підхід забезпечує коректну та універсальну перевірку зайнятості місць на основі геометричного аналізу, що є суттєвою

перевагою у випадках нерівномірного розміщення паркомісць або нестандартних кутів огляду камери.

Для зберігання інформації про паркомісця, камери, користувачів та історію зайнятості в системі використовується база даних, реалізована за допомогою SQLite. Підключення до неї організовано через SQLAlchemy – популярну ORM-бібліотеку Python, яка забезпечує зручну абстракцію над SQL-запитами.

Модуль database.py містить ініціалізацію базового двигуна, сесій та декларативної бази моделей у лістингу 3.5.

Лістинг 3.5 – Підключення до бази даних

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from config import SQLALCHEMY_DATABASE_URL

engine = create_engine(
    SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False}
)
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
Base = declarative_base()
```

кінець лістингу 3.5

Цей модуль є фундаментом для всієї подальшої роботи з базою даних, включаючи створення, оновлення та отримання даних за допомогою ORM-зв’язків.

У рамках практичної реалізації системи моніторингу зайнятості паркомісць було спроєктовано реляційну структуру бази даних на основі SQLAlchemy ORM. Структура охоплює сутності: камери, паркомісця, групи місць, розмітку схеми та користувачів.

Усі моделі реалізовані згідно з принципами нормалізації. Зовнішні ключі забезпечують логічну зв’язність між таблицями. Така архітектура дозволяє ефективно масштабувати базу даних при розширенні функціональності системи.

Для забезпечення зручної взаємодії з системою моніторингу паркомісць було реалізовано RESTful API на базі фреймворку FastAPI, що дозволяє гнучко керувати всіма ключовими об’єктами: користувачами, камерами, паркомісцями, групами та схемами.

Усі маршрути згруповані в окремі модулі, що відповідають за логіку відповідних сутностей. Доступ до більшості операцій обмежено авторизованими користувачами, що підтверджується за допомогою JWT-токенів. Аутентифікація реалізована через стандартну схему OAuth2PasswordBearer.

API підтримує повноцінний CRUD для камер, груп паркомісць, паркомісцями та схемою паркінгу.

Всі запити до захищених маршрутів потребують передачі Bearer токену. Це забезпечує базовий рівень безпеки системи.

Для взаємодії з камерами також реалізовано можливість отримання JPEG-знімка напряму з потоку, безпосередньо через виклик API, що дозволяє оновлювати інтерфейс користувача в режимі реального часу.

Кожен ендпоінт використовує залежності Depends(get_db) та Depends(get_current_user), що забезпечує централізовану обробку сесій бази даних і авторизації.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

У процесі розробки системи моніторингу паркомісць було проведено комплексне тестування її модулів для забезпечення коректної роботи та відповідності функціональним вимогам.

Суть тестування системи полягало в модульній перевірці кожного компонента і умовах, наближених до реальних.

Після цього було здійснено повноцінне тестування системи.

Тестування охоплювало як модулі обробки відео, так і частину, пов’язану з API, базою даних і логікою визначення зайнятості місць (рис. 3.2).

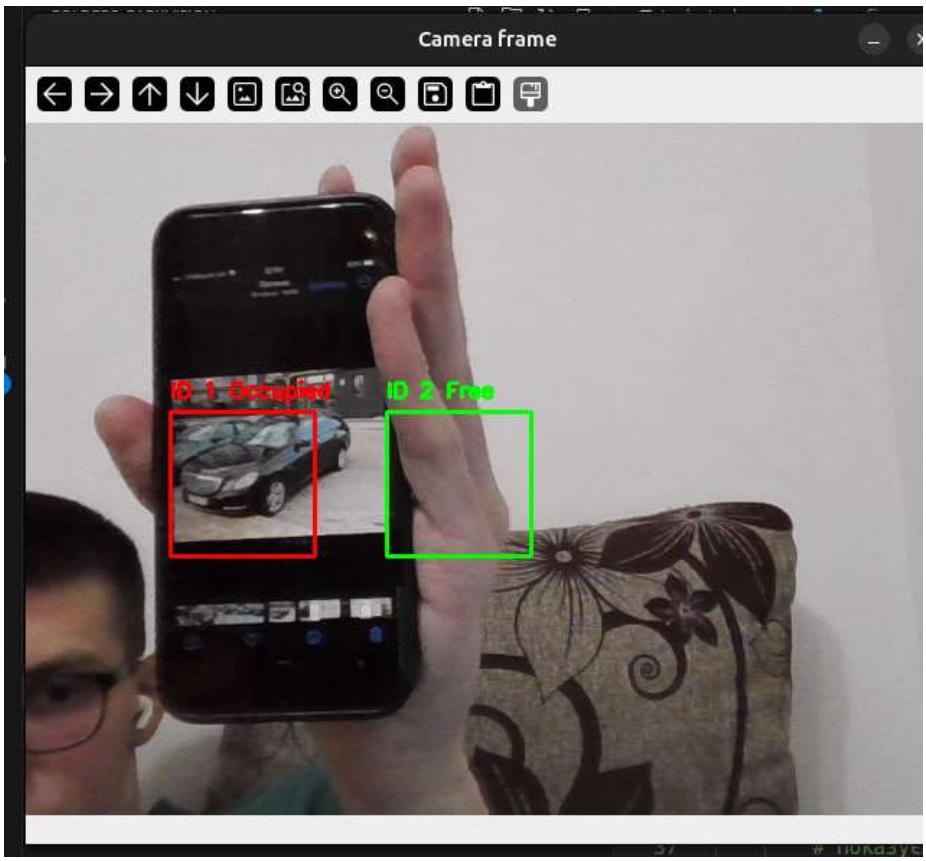

Рисунок 3.2 – Процес тестування програмного забезпечення

Для перевірки працездатності було обрано модульний та інтеграційний підходи:

– CameraManager тестувався на стабільність захоплення кадрів з RTSP-джерел;

– Detector – на коректність детекції автомобілів у статичних та динамічних умовах;

– ParkingChecker – на точність визначення перетинів bounding box з координатами p1-p4.

Інтеграційне тестування перевіряло взаємодію між модулями, зокрема захоплення кадру, детекція об’єктів, аналіз зайнятості, запис у БД, роботу API з базою даних та коректність відповідей для клієнтських запитів.

Для перевірки були використані тестові відеопотоки та зображення з відкритих джерел. У результаті тестування були виявлені дрібні помилки, пов’язані з неправильним масштабуванням кадру під час обробки, які були успішно виправлені.

Важливим кроком було тестування у фоновому режимі, з імітацією стабільного потоку з камери. Система стабільно функціонувала без помилок протягом годин, що підтверджує її готовність до довготривалої експлуатації.

3.3 Розробка бази даних

В основі системи моніторингу паркомісць лежить структурована база даних, яка зберігає всю ключову інформацію: від налаштувань камер і схеми паркомісць до статусів груп і облікових записів користувачів. Для створення бази даних було обрано SQLite як легку, вбудовану СУБД, яка не потребує окремого серверного процесу, що ідеально підходить для автономного запуску.

База даних побудована з використанням ORM SQLAlchemy-інструменту, що забезпечує зручну об’єктно-реляційну прив’язку. Це дозволяє взаємодіяти з БД за допомогою класів Python, не використовуючи сирі SQL-запити.

У системі реалізовано п’ять основних таблиць:

– cameras – містить дані про камери спостереження (назва, джерело відео);

– parking_places – описує паркомісця, кожне з яких має координати чотирьох точок (p1-p4), що дозволяє зберігати довільні;

– parking_groups – зберігає логічні об’єднання паркомісць, наприклад, по зонах або поверхах;

– parking_layout – таблиця зі схемою розміщення зон (груп), які відображаються в редакторі схеми;

– users – реалізує базову авторизацію адміністратора через логін і хешований пароль.

Нижче наведено приклад фрагмента моделі SQLAlchemy для таблиці parking_places у лістингу 3.6.

Лістинг 3.6 – Фрагмент моделі SQLAlchemy для таблиці parking_places

```
class ParkingPlace(Base):
    __tablename__ = "parking_places"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    camera_id = Column(Integer, ForeignKey("cameras.id"))
    group_id = Column(Integer, ForeignKey("parking_groups.id"))
    p1_x = Column(Integer, nullable=False)
    p1_y = Column(Integer, nullable=False)
    p2_x = Column(Integer, nullable=False)
    p2_y = Column(Integer, nullable=False)
    p3_x = Column(Integer, nullable=False)
    p3_y = Column(Integer, nullable=False)
    p4_x = Column(Integer, nullable=False)
    p4_y = Column(Integer, nullable=False)
```

кінець лістингу 3.6

Для візуалізації взаємозв’язків між таблицями була створена ER-діаграма, подана на рисунку нижче. Вона наочно демонструє логіку взаємодії таблиць та основні ключі (рис. 3.3).

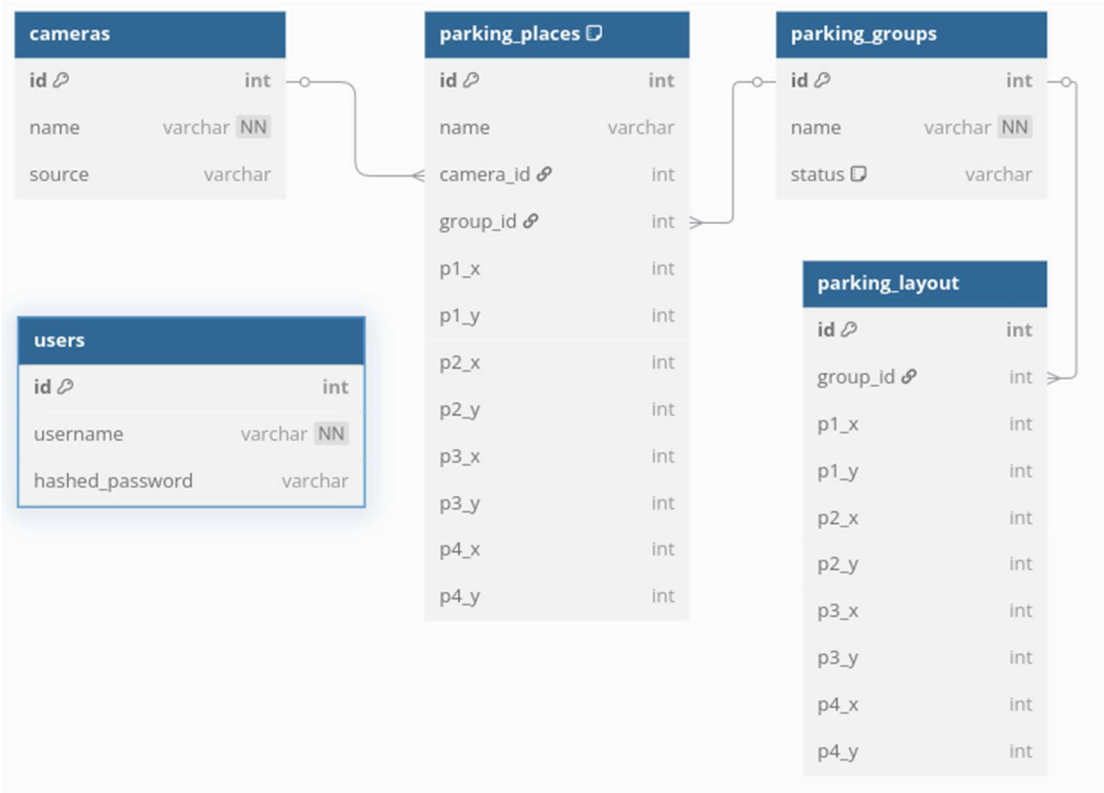

Рисунок 3.3 – ER-діаграма бази даних системи моніторингу паркомісць

Завдяки продуманій структурі БД, система залишається простою в обслуговуванні, водночас забезпечуючи повну функціональність, необхідну для стабільної роботи системи моніторингу паркомісць.

3.4 Захист інформаційно-комп’ютерної системи

Оскільки система передбачає керування камерами та зчитування інформації з відеопотоків, важливо було реалізувати базові механізми захисту. Враховуючи характер системи, було обрано концепцію автентифікації користувача через JSON Web Token (JWT). При вході користувача у систему створюється токен з обмеженим строком дії, який потім додається до всіх подальших запитів.

Захист паролів реалізовано через алгоритм хешування за допомогою бібліотеки bcrypt, що унеможливлює зберігання паролів у відкритому вигляді. Кожен запит до адміністративної частини API перевіряється через middleware на наявність дійсного токена доступу. У разі його відсутності чи недійсності запит блокується на рівні FastAPI.

Окрім того, під час проєктування було враховано рекомендації щодо обмеження доступу до CORS-ресурсів, що мінімізує ризики міжсайтового виконання запитів. Таким чином, система отримала базовий, але надійний рівень захисту, достатній для невеликої серверної інсталяції або внутрішнього використання.

3.5 Інтерфейс клієнтської частини

Розроблена система моніторингу паркомісць є повноцінним серверним рішенням, основна мета якого -забезпечити автоматизоване визначення зайнятості місць на основі комп’ютерного зору. Принциповою особливістю архітектури є відсутність жорсткої прив’язки до конкретної реалізації клієнтської частини. Вся взаємодія із системою відбувається через RESTful API,

що дозволяє інтегрувати її з будь-яким інтерфейсом: вебзастосунком, мобільним додатком або стороннім сервером [16].

Ключова ідея полягає в тому, що вся логіка роботи, обробка відео, зберігання статусів і авторизація реалізуються на сервері. Це дозволяє зовнішнім розробникам або адміністраторам створювати власні клієнти, які отримуватимуть або змінюватимуть дані відповідно до потреб. Публічне API включає такі ендпоінти як:

– /auth/login -авторизація користувача з отриманням токена;

– /cameras/ -управління камерами (GET, POST, PUT, DELETE);

– /places/ -паркомісця з координатами у вигляді чотирьох точок;

– /groups/ -групи паркомісць, для зручності класифікації;

– /schema/ -схема парковки із координатами та знімком;

/status/ -перегляд зайнятості місць у реальному часі.

Ці інтерфейси спроектовані таким чином, щоб бути максимально простими у використанні та легко підключатися з будь-якого середовища (рис. 3.4).

Рисунок 3.4 – API-документація з прикладами ендпоінтів

Хоча система не потребує графічного інтерфейсу для функціонування, у рамках демонстрації роботи API було створено базову реалізацію вебінтерфейсу адміністратора, який працює без фреймворків -лише на HTML, CSS і JavaScript.

Це дозволяє якнайкраще проілюструвати принципи взаємодії з API та водночас зберігає універсальність.

Після введення логіна і пароля, система надсилає POST-запит до /auth/login, отримує токен і зберігає його у localStorage. Приклад наведено у лістингу 3.7.

Лістинг 3.7 – Функція авторизації

```
fetch('/auth/login', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ username: 'admin', password: '123456' })
})
.then(res => res.json())
.then(data => localStorage.setItem('token', data.access_token));
```

кінець лістингу 3.7

Цей токен додається до всіх наступних запитів у заголовку Authorization, що дозволяє ідентифікувати користувача.

Однією з ключових функцій інтерфейсу є редактор паркомісць. Він реалізований через canvas (рис. 3.5).

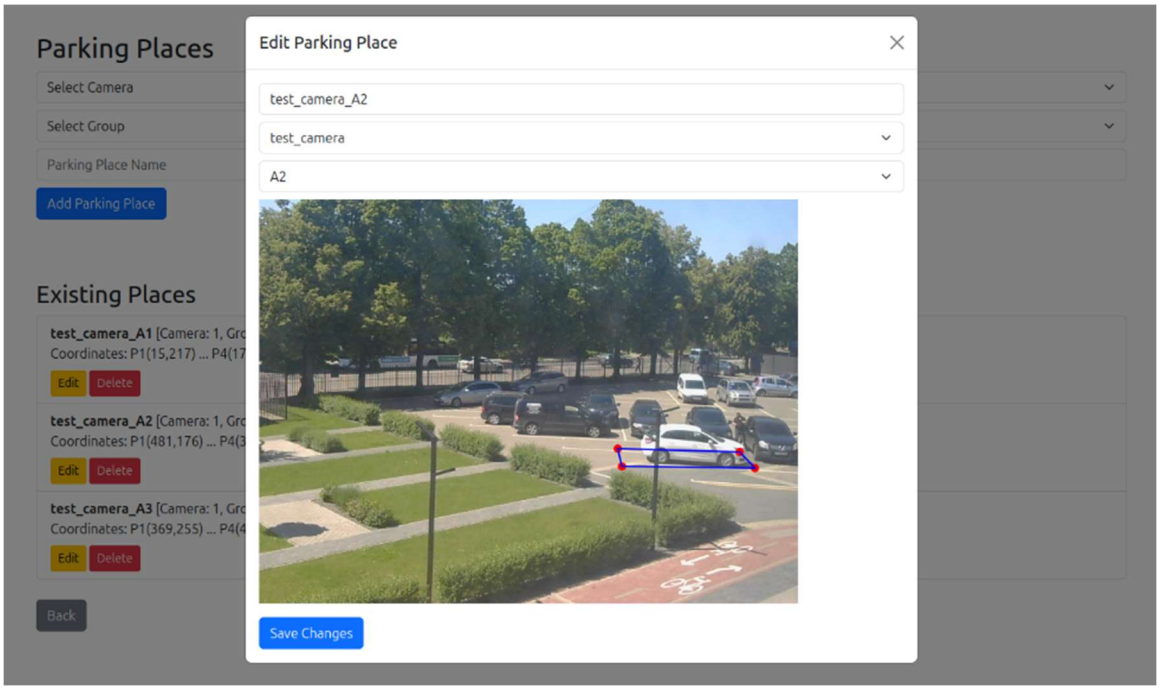

Рисунок 3.5 – Інтерфейс редагування координат паркомісця

Така візуалізація дозволяє миттєво отримати уявлення про ситуацію на парковці.

Оскільки API відкритий, він може бути використаний будь-де. Наприклад, наступний Python-скрипт може бути інтегрований у Telegram-бота, що надсилатиме повідомлення про вільні місця.

Інтерфейс клієнтської частини був реалізований для демонстрації можливостей API та не є обов'язковим елементом системи. Гнучкість, яку забезпечує REST-архітектура, дозволяє легко інтегрувати систему з будь-яким UI або платформою - від мобільного застосунку до великої інформаційної панелі на підприємстві. Продемонстрований HTML/JS-клієнт служить лише прикладом того, як це може виглядати і працювати на практиці.

Висновки до розділу 3

У третьому розділі дипломної роботи було розглянуто практичну реалізацію інформаційної системи моніторингу паркомісць, що поєднує інструменти комп'ютерного зору, сучасні веб-технології та роботу з базами даних. Реалізований програмний продукт являє собою повноцінну серверну систему, що дозволяє виявляти зайнятість паркомісць у режимі реального часу на основі відеопотоку з камер спостереження.

Процес розробки розпочався з побудови чіткої модульної архітектури. Було створено окремі компоненти для зчитування відеопотоку (CameraManager), виявлення транспортних засобів за допомогою моделі YOLOv5 (Detector), перевірки перетину з паркомісцями (ParkingChecker), роботи з базою даних через ORM SQLAlchemy, а також розширене REST API для взаємодії з клієнтською частиною. Архітектура побудована таким чином, щоб усі компоненти були максимально незалежними та легко масштабувалися.

Особливу увагу приділено обробці відеопотоку: камера ініціалізується з певним інтервалом кадрування, виконується очищення буфера, і лише після цього кадр передається на обробку. Детектор на основі YOLOv5 виконує

розпізнавання об’єктів, фільтруючи лише ті, які належать до класів транспортних засобів. Після цього відбувається геометрична перевірка перетину з кожною зони паркування, яка задана у вигляді чотирикутника (p1-p4).

Для зберігання інформації використовується реляційна база даних з моделями для камер, паркомісць, груп, схеми паркінгу та користувачів. Всі таблиці мають чітко продумані зв’язки, що дозволяє ефективно формувати запити, фільтрувати записи та працювати з групами паркомісць як логічними об’єктами.

Завдяки використанню FastAPI реалізовано швидкий, асинхронний сервер, що підтримує авторизацію через JWT-токени, дозволяє обмежити доступ до критичних функцій системи та реалізувати багаторівневу взаємодію з API. Реалізовано повний набір CRUD-операцій для камер, паркомісць, груп, схеми та користувачів, що дозволяє ефективно управляти конфігурацією паркінгу.

Окремим етапом стала розробка функції перевірки статусів зайнятості. Вона дозволяє у будь-який момент отримати список паркомісць із поточним станом, а також завантажити актуальну схему зони паркування. Всі ці дані доступні через публічні кінцеві точки API, що дозволяє у майбутньому інтегрувати систему з іншими платформами, мобільними додатками або системами диспетчеризації.

Проведене тестування підтвердило стабільну роботу системи в умовах реального часу, стійкість до відсутності відеопотоку, а також коректну обробку некоректних запитів до API. Було також оцінено мінімальні системні вимоги для запуску проєкту, що дозволяє адаптувати його під різні серверні конфігурації.

Таким чином, у межах цього розділу було досягнуто ключових цілей – створено функціональний, захищений і гнучкий прототип інформаційної системи для моніторингу паркомісць. Результати реалізації свідчать про ефективність обраних технологій, обґрунтованість архітектурних рішень і готовність системи до розгортання в реальному середовищі або подальшої інтеграції з клієнтським інтерфейсом.

ВИСНОВКИ

У ході кваліфікаційної роботи було розроблено повнофункціональну систему моніторингу паркомісць, що працює в автономному режимі та базується на сучасних інструментах комп'ютерного зору, обробки відео та веброзробки. Досягнута мета -створення серверної частини, що виконує аналіз відеопотоку, визначення зайнятості місць, збереження інформації у базі даних та доступ до функціоналу через REST API.

Здійснено детальний аналіз сучасних систем моніторингу паркомісць. Розглянуто підходи на основі сенсорів, індукційних датчиків і відеоаналітики. Обґрунтовано переваги відеоаналізу з використанням комп'ютерного зору та нейромереж.

Розроблено загальну архітектуру системи, яка включає функціональні блоки обробки відео, детекції об'єктів, бази даних, API та авторизації. Визначено способи взаємодії модулів і забезпечення цілісності даних.

Реалізовано алгоритм автоматичного виявлення транспортних засобів за допомогою YOLOv5, адаптованого для роботи в реальному часі. Забезпечено високу точність і стабільність обробки кадрів.

Здійснено підтримку обробки відеопотоків через RTSP-протокол із реалізацією механізму періодичного захоплення зображень і визначення статусу зайнятості місць.

Створено локальну базу даних SQLite для збереження інформації про паркомісця, їх координати, статуси, налаштування та камери. Структура БД підтримує масштабування і гнучке оновлення.

Розроблено REST API, що забезпечує повноцінне керування об'єктами системи: додавання, редагування, перегляд і видалення паркомісць, камер та груп.

Реалізовано механізм авторизації користувачів у системі за JWT-токенів, що дозволило зробити авторизацію безпечною.

Забезпечено модульну структуру програмного забезпечення, що дозволяє адаптувати логіку роботи системи, структуру бази даних або взаємодію компонентів без значних змін.

Проведено комплексне тестування системи в умовах, наближених до реальних. Встановлено точність виявлення автомобілів понад 90%, стабільність обробки відео та надійність збереження даних.

Таким чином, під час виконання дипломної роботи вдалося реалізувати повнофункціональну систему, яка вирішує поставлену задачу контролю паркомісць, забезпечує автоматизацію процесу виявлення зайнятості, та має гнучку структуру, що дозволяє легко розширювати її у майбутньому. Робота містить значну практичну цінність і потенціал для інтеграції в інтелектуальні міські системи управління трафіком або комерційні сервіси моніторингу паркінгу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. You Only Look Once (YOLO): What is it?. DataScientest. URL: https://datascientest.com/en/you-only-look-once-yolo-what-is-it (дата звернення: 24.02.2025).

2. Објаснива вештачка интелигенција кај LLMs и/или модели за детекција на објекти - Користење на вештачката интелигенција (AI) за унапредување на родовата еднаквост во академските истражувања. Користење на вештачката интелигенција (AI) за унапредување на родовата еднаквост во академските истражувања. URL: https://aigender.net/index.php/2025/02/23/llm/ (дата звернення: 24.02.2025).

3. Sustainability - Free Full-Text - Traffic Flow Prediction and Vehicle Detection in Complex Urban Environments – URL: https://www.mdpi.com/2071-1050/14/19/12274 (дата звернення: 25.02.2025).

4. MEB-YOLO: An Efficient Vehicle Detection Method in Complex Traffic Road Scenes – URL: https://www.researchgate.net/publication/370530173_meb-yolo_an_efficient_vehicle_detection_method_in_complex_traffic_road_scenes (дата звернення: 25.02.2025).

5. Smart Parking with Pixel-Wise ROI Selection for Vehicle Detection Using YOLOv8, YOLOv9, YOLOv10, and YOLOv11. arXiv.org e-Print archive. URL: https://arxiv.org/html/2412.01983v1 (дата звернення: 27.02.2025).

6. AI Parking Occupancy Detection Using a Camera. AI Parking Occupancy Detection Using a Camera - Parkinto. URL: https://parkinto.com/ (дата звернення: 27.02.2025).

7. CMCA-YOLO: A Study on a Real-Time Object Detection Model for Parking Lot Surveillance Imagery. MDPI. URL: https://www.mdpi.com/2079-9292/13/8/1557 (дата звернення: 06.03.2025).

8. Which Is the Best Python Web Framework: Django, Flask, or FastAPI? - The PyCharm Blog. The JetBrains Blog. URL: https://blog.jetbrains.com/pycharm/2025/02/django-flask-fastapi/ (дата звернення: 06.03.2025).

9. Tutorial - User Guide - FastAPI. URL: https://fastapi.tiangolo.com/tutorial/ (дата звернення: 09.03.2025).

10. Recording RTSP video feeds using Python and OpenCV. Sander van de Velde. URL: https://sandervandevelde.wordpress.com/2024/10/12/recording-rtsp-video-feeds-using-python-and-opencv/ (дата звернення: 09.03.2025).

11. Vl. How to Capture RTSP Video Streams Using OpenCV (Installed via VCPKG). Funvision opencv C++ tutorials. URL: https://www.funvisiontutorials.com/2024/12/how-to-capture-rtsp-video-streams-using.html (дата звернення: 12.03.20255).

12. GitHub - ultralytics/yolov5: YOLOv5 in PyTorch > ONNX > CoreML > TFLite. GitHub. URL: https://github.com/ultralytics/yolov5 (дата звернення: 14.03.2025).

13. Miguelañez C. 6 Best Embedded Databases for 2024. Latitude Blog. URL: https://latitude-blog.ghost.io/blog/6-best-embedded-databases-2024/ (дата звернення: 15.03.2025).

14. sqlite3 DB-API 2.0 interface for SQLite databases. Python documentation. URL: https://docs.python.org/3/library/sqlite3.html (дата звернення: 15.03.2025).

15. Ultralytics. Index. Home - Ultralytics YOLO Docs. URL: https://docs.ultralytics.com/yolov5/ (дата звернення: 20.03.2025).

16. GeeksforGeeks. Python API Tutorial: Getting Started with APIs - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/python-api-tutorial-getting-started-with-apis/ (дата звернення: 22.03.2025).

ДОДАТКИ

Додаток А – Тези в науковій конференції

Міністерство освіти і науки України
Волинська обласна рада
Луцька міська рада
Луцький національний технічний університет
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Національний університет «Львівська політехніка»
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)
Тернопільський національний педагогічний університет імені В. Гнатюка
Рівненський державний гуманітарний університет
Навчально-науковий інститут «Українська інженерно-педагогічна академія» Харківського національного університету ім. В.Н. Каразіна
Люблінська політехніка (Польща)
Фрайберзька гірнича академія (Німеччина)
Гронінгенський університет (Нідерланди)
Кавказький університет (Грузія)
Політехнічний університет Браганси (Португалія)

ТЕЗИ ДОПОВІДЕЙ
X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)

23-24 травня 2025 р.

Луцьк 2025

УДК 004.42

ІНТЕЛЕКТУАЛЬНА СИСТЕМА МОНІТОРИНГУ ПАРКОМІСЦЬ НА ОСНОВІ КОМП'ЮТЕРНОГО ЗОРУ

Йоц Іван Васильович
Луцький національний технічний університет, здобувач вищої освіти спеціальності ІПЗ, yots.i2504@lntu.edu.ua
Ящук Андрій Анатолійович
Луцький національний технічний університет, к.т.н., доцент кафедри інженерії програмного забезпечення, a.yashchuk@lutsk-ntu.com.ua

INTELLIGENT PARKING SPACE MONITORING SYSTEM BASED ON COMPUTER VISION

Yots Ivan Vasyliovych
Lutsk National Technical University, higher education student in the educational program of Software Engineering, yots.i2504@lntu.edu.ua
Yashchuk Andrii Anatoliiovych
Lutsk National Technical University, PhD, Associate Professor of the Department of Software Engineering, a.yashchuk@lutsk-ntu.com.ua

This abstract presents the development of an intelligent parking space monitoring system based on computer vision and the YOLO model. It supports both RTSP and USB cameras and provides real-time parking occupancy detection. The system is built with FastAPI, uses SQLite with SQLAlchemy for data storage, and applies JWT tokens for secure user authentication. Video analysis is performed using OpenCV, and the modular architecture ensures scalability and flexibility, making the system suitable for smart city applications.

Keywords: *YOLO, smart city, parking monitoring, computer vision, video analytics, automation.*

Зростання кількості автомобілів у містах призводить до проблем із пошуком вільних паркомісць, що спричиняє затори, додаткові витрати та підвищення рівня CO_2. Одним із перспективних напрямів вирішення цієї проблеми є впровадження інтелектуальних систем моніторингу паркінгів на основі комп'ютерного зору.

Для автоматизованого визначення зайнятості паркомісць використано алгоритм YOLO (You Only Look Once) [1]. YOLO забезпечує виявлення об'єктів у реальному часі, має високу

клієнтських додатків. Зберігання інформації реалізовано на базі SQLite із використанням ORM SQLAlchemy. Для забезпечення безпеки впроваджено автентифікацію за допомогою JWT-токенів та хешування паролів через Passlib (рис. 1).

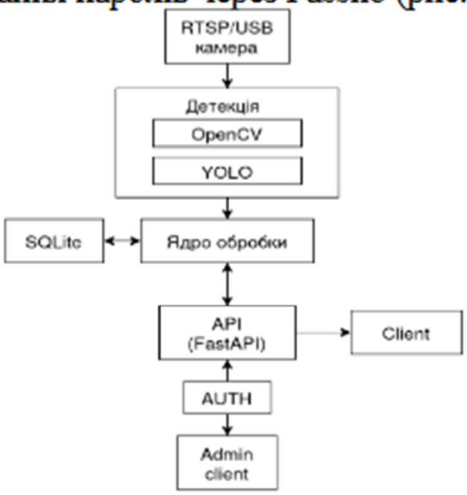

Рисунок 1 – Схема роботи системи моніторингу паркомісць

Система підтримує роботу з локальними (USB) і мережевими (RTSP) камерами, що дозволяє масштабувати її без значних витрат. Фонові задачі з обробки відеопотоків реалізовано у вигляді окремих потоків, що запускаються при старті сервера.

Модульна архітектура та окреме розмежування прав користувачів і адміністраторів дозволяють гнучко адаптувати систему до різних сценаріїв використання – від малого бізнесу до міських платформ (рис. 2).

Рисунок 2 – Приклад адаптації розроблюваної системи для парковки ЛНТУ

швидкодію, підтримку різних версій та добре інтегрується з такими інструментами, як OpenCV [2], TensorRT, DeepStream [3].

Аналіз літератури свідчить про активні дослідження в напрямку підвищення точності детекції, зменшення вимог до ресурсів та стійкості до зовнішніх чинників.

У роботі Real-Time Vehicle Detection Based on Improved YOLO v5 запропоновано вдосконалену версію YOLOv5 з Flip-Mosaic алгоритмом для кращого розпізнавання малих об'єктів і зменшення хибних спрацювань [4]. MEB-YOLO поєднує Mosaic, MixUp і механізм уваги ECA, що підвищує точність у складних умовах [5]. У дослідженні Smart Parking with Pixel-Wise ROI Selection описано використання IoT, Edge Computing і глибокого навчання для створення масштабованих систем моніторингу паркінгів на основі YOLO [6]. Також розглядаються гібридні рішення з камерами і сенсорами для підвищення надійності.

Існуючим аналогом розробленої системи є Parkinto [7], яка аналізує відеопотоки з камер, визначає зайнятість паркомісць і передає дані в реальному часі через API. Її перевага – інтеграція з наявними камерами, що знижує витрати. Також вона пропонує зручне ПЗ та клієнтську підтримку. Водночас до недоліків належать висока вартість, залежність від постачальника та обмежена кастомізація. Це робить доцільною розробку власної, гнучкої системи, адаптованої до конкретних умов і бюджету.

Запропонована власна модульна система використовує наявні камери спостереження, локальне збереження даних (SQLite) та REST API для інтеграції з іншими сервісами.

Для аналізу кожен кадр із відеопотоку обробляється бібліотекою OpenCV, після чого передається до YOLO-моделі. Виявлені об'єкти порівнюються з попередньо заданими координатами паркомісць, що дозволяє визначити їх поточний стан. Детекція виконується у реальному часі, покадрово, з певною періодичністю.

Технічна реалізація побудована з використанням вебфреймворку FastAPI, який забезпечує REST API для

Запропонована система орієнтована на високу точність і масштабованість. Вона адаптована як до невеликих об'єктів, так і до великих паркінгів, що робить її ефективним інструментом у рамках концепції «розумного міста».

Список використаних джерел

1. You Only Look Once (YOLO): What is it? URL: https://datascientest.com/en/you-only-look-once-yolo-what-is-it (дата звернення: 07.05.2025).

2. Running pre-trained YOLO model in OpenCV. URL: https://docs.opencv.org/4.x/da/d9d/tutorial_dnn_yolo.html (дата звернення: 07.05.2025).

3. Deploy YOLOv8 on NVIDIA Jetson using TensorRT and DeepStream SDK Support. URL: https://wiki.seeedstudio.com/YOLOv8-DeepStream-TRT-Jetson/ (дата звернення: 07.05.2025).

4. Sustainability | Free Full-Text | Traffic Flow Prediction and Vehicle Detection in Complex Urban Environments. URL: https://www.mdpi.com/2071-1050/14/19/12274 (дата звернення: 07.05.2025).

5. MEB-YOLO: An Efficient Vehicle Detection Method in Complex Traffic Road Scenes. URL: https://www.researchgate.net/publication/370530173_MEB-YOLO (дата звернення: 07.05.2025).

6. A Novel Method for Vehicle Detection Using Deep Learning. URL: https://arxiv.org/html/2412.01983v1 (дата звернення: 07.05.2025).

7. Discover Parkinto: All You Need to Know. URL: https://parkinto.com/about/ (дата звернення: 07.05.2025).

СЕРТИФІКАТ

Кафедра цифрових освітніх технологій

Кафедра інженерії програмного забезпечення

Даний сертифікат засвідчує, що

Йоц Іван Васильович

брав(-ла) участь у

X Міжнародній науково-практичній конференції
з проблем вищої освіти і науки
"Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)"
загальним обсягом 15 годин (0,5 кредитів ECTS),

яка проводилася 23-24 травня 2025 року у м.Луцьк
на базі Луцького національного технічного університету

Ректор ЛНТУ _____ **Ірина ВАХОВИЧ**

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

№ ІТОНВ-2025-65

Програма конференції: https://itonv.lntu.edu.ua/files/2025/program-itonv-2025.pdf