Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Програмна система для автоматизованого керування паркінгом транспортних засобів
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.190137.01.13.ПЗ

Виконав студент IV курсу група ІПЗ-19-1 _____ В. М. Мариняк
Підпис Ініціали, прізвище

Керівник канд. техн. наук, доцент _____ О. М.Яшина
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ Ю. В. Форкун
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

8 червня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
01 03 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мариняку Владиславу Миколайовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програмна система для автоматизованого керування паркінгом транспортних засобів

Керівник проекту (роботи) Яшина Оксана Миколаївна, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2022 р. № 5
2. Строк подання студентом проекту (роботи) на кафедру 01.06.2023 р.
3. Вихідні дані до проекту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація та тестування програмної системи
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) 3 креслення
1. ER діаграма
2. DFD-діаграма
3. Діаграма варіантів використання

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Форкун Ю. В., доцент кафедри ІПЗ	31.05.23	06.06.23
Антиплагіат	Гурман І. В., доцент кафедри ІПЗ	7.06.23	7.06.23

7. Дата видачі завдання « 01 » березня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1 Ознайомлення з тематикою кваліфікаційної роботи (КвР), визначення та узгодження індивідуальних тем КвР	01.12 – 30.12.2022	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2023	
3 Проектування програмного забезпечення	01.02 – 28.02.2023	
4 Програмна реалізація	01.03 – 10.04.2023	
5 Тестування програмного забезпечення	11.04 – 30.04.2023	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог стандартів	01.05 – 25.05.2023	
7 Попередній захист КвР	Травень 2023 (згідно графіка)	
8 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки	26.05 – 1.06.2023	
9 Підготовка до захисту та захист КвР	з 01.06.2023	

Студент _____ В. М. Мариняк
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ О.М. Яшина
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмна система для автоматизованого керування паркінгом транспортних засобів».

Автор роботи: Мариняк В. М.

Керівник роботи: к.т.н., доцент Яшина Оксана Миколаївна.

Пояснювальна записка: 101 с., 15 рис., 2 табл., 2 дод., 40 джерел.

Графічна частина: 3 креслення.

ВЕБ-ДОДАТОК, ВЕБ-РЕСУРС, СИСТЕМА КЕРУВАННЯ, ЕЛЕКТРОННИЙ СЕРВІС ПОСЛУГ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ПРОГРАМНА СИСТЕМА.

Метою роботи є розробка автоматизованої системи керування паркінгом транспортних засобів.

Об'єкт дослідження – процес розробки автоматизованої системи керування паркінгом.

Предмет дослідження – методи та засоби, що забезпечують систему керування паркінгом.

У кваліфікаційній роботі було проаналізовано предметну область із подальшим здійсненням програмної реалізації програмного продукту.

6 червня 2023 р.
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.190137.01.13.ПЗ	Пояснювальна записка	101		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ.190137.01.13.Е8	DFD-діаграма	1		
5	А3	КвРІПЗ.190137.01.13.Е8	Діаграма варіантів	1		
			використання			
6	А3	КвРІПЗ.190137.01.13.Е8	ER діаграма	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.190137.01.13.ВД		
Виконав		Мариняк В.М.		6.06	Програмна система для автоматизованого керування паркінгом транспортних засобів	Літ.	Арк. 1 / Акрушів 101
Керівник		Яшина О.М.		5.06			
Н. Контр.		Форкун Ю.В.		6.06	Відомість документів	ХНУ, ІПЗ-19-1	
Зав. Каф.		Бедратюк Л.П.		6.06			

ЗМІСТ

ВСТУП

Наразі багато громадських місць, таких як торгові центри, лікарні, офіси, кінотеатри, ринки тощо, стикаються зі зростаючими проблемами, пов'язаними з паркуванням транспортних засобів. Відповідні зони, де можна поставити транспортні засоби, як правило, погано облаштовані, не мають чітких підказок щодо пошуку безкоштовних або платних місць для паркування, незручні місця для паркування та ненадійні системи моніторингу паркування. Крім того, надання цього виду послуг вимагає багато ручної роботи та інвестицій, якщо бізнес не інтегрує сучасні рішення у свої внутрішні процеси.

Із зростанням кількості власників транспортних засобів виникає нагальна потреба у створенні та управлінні якомога більшою кількістю зон для паркування транспортних засобів. Паркування є основною частиною транспортної проблеми, і ця частка швидко зростає, оскільки кількість місць для паркування обмежена. Пошук місця для паркування є рутинним і часто неприємним завданням для багатьох людей по всьому світу. Щодня на це витрачається близько мільйона барелів нафти. Згідно зі звітом, оптимізоване розумне паркування може заощадити 2 мільйони галонів пального до 2030 року і приблизно 3 мільйони галонів до 2050 року.

Автоматизовані системи паркування мають такі переваги, як власне програмне забезпечення, зчитування номерних знаків транспортних засобів, запис і зберігання відео, а також сучасні технології, що дозволяють контролювати стан паркувального простору. Автоматичні системи паркування - це найдоступніший і найзручніший варіант для використання на підприємствах з високим трафіком. Завдяки спеціальному програмному забезпеченню доступні інтегровані рішення для широкого спектру об'єктів, включаючи аеропорти, залізничні вокзали, торгові центри, розважальні центри, бізнес-центри та спортивні комплекси. Автоматизовані системи паркування можуть майже повністю замінити участь працівників у

паркувальних операціях.

Впровадження автоматизованого паркування в багатьох випадках дозволяє повністю позбутися витрат на операторів, які на не автоматизованих паркінгах вручну управляють і регулюють процесами на паркування. Велике значення має й конкурентна перевага, наявність можливості вільно запаркуватися у безпосередній близькості від мети поїздки стає значною перевагою. Особливо в той час року коли клімат не сприяє пішим прогулянкам. Дана система також сприяє ефективності паркування. Не автоматизовані паркування заповнюють таксисти, офісні працівники із найближчих бізнес-центрів, мешканці довколишніх будинків тощо. Це заважає заповненню паркінгу цільовими відвідувачами. Така система позбавляє від несумлінності персоналу (банальної крадіжки). За різними даними на платних, але не автоматизованих парковках, за участю операторів у процесах оплати, власники паркінгів не отримують до 35% виручки. У кризових умовах зазвичай зростає кількість злочинів. Важливим завданням з яким вам допоможе впоратися система - це запобігання викраденню транспортних засобів. Досвід західних країн в автоматизації паркінгів давно доводить їхню ефективність та рентабельність. Вони міцно зайняли свою нішу в міській інфраструктурі, тому на сьогодні, жоден європейський паркінг не обходиться без систем повної або часткової автоматизації внутрішніх процесів.

Метою роботи є розробка автоматизованої системи керування паркінгом транспортних засобів.

Об’єкт дослідження – процес розробки автоматизованої системи керування паркінгом.

Предмет дослідження – методи та засоби, що забезпечують систему керування паркінгом.

Виходячи із мети, об’єкту, предмету можна виділити такі завдання, щоб досягти позитивного результату під час розробки:

– здійснити детальний та коректний аналіз предметної області;

– проаналізувати сучасний стан наявного програмного забезпечення даної предметної області;

– визначити вимоги до програмної системи;

– здійснити проектування програмної системи із визначенням базових рішень, що будуть реалізовані в процесі розробки;

– здійснити розробку функціональної схеми автоматизованої системи паркінгу транспортних засобів;

– здійснити проектування моделі із розробкою структури інформаційної системи smart-паркінгу;

– здійснити розробку програмного продукту, що має дружнім, інтуїтивно зрозумілий для користувач інтерактивним інтерфейс;

– провести тестування та впровадження програмного продукту.

Отже, тематика даного дослідження є актуальною та корисною в сучасному світі, де транспортний засіб є у кожного другого жителя мегаполісу.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Підприємства витрачають багато зусиль, щоб залучити клієнтів та утримати їх. Якщо підприємці покладаються на місцевий бізнес, вони з більшою ймовірністю інвестуватимуть у власні веб-сайти, зовнішню рекламу та торгові виставки. Такі заходи організовуються для залучення нових клієнтів, які будуть підтримувати бренд. Однак власники бізнесу можуть нехтувати одним з найважливіших аспектів побудови взаємовигідних відносин, а саме облаштування зручного та автоматизованого паркінгу, що знаходиться біля будівлі підприємства чи організації.

Паркінг – це перший контакт клієнта з бізнесом. Якщо він буде незадоволений якістю послуги, то велика ймовірність, що просто не повернеться на певне підприємство чи установу (тут більше мова йде про розважальні заклади та торгові центри), а поїде туди, де більш зручна парковка. Уявімо, що потенційний клієнт шукає в Інтернеті продукт, який пропонує компанія. Власник докладе всіх зусиль, щоб продукт з'явився в топі у результатах пошуку та клієнт перейшов за посиланням на сайт. Власник також дбає проте, щоб зібрати позитивні відгуки про вже надані послуги, таким чином підвищуючи зацікавленість клієнта в продукті. Нарешті, купони, надіслані поштою, використовуються для того, щоб переконати їх відвідати бізнес.

Однак, коли клієнти приїжджали на територію компанії, вони стикалися з низкою проблем: касири довго не могли зареєструвати їхні транспортні засоби на паперових бланках, неправильно вносили дані, зазначені в квитанціях, і не могли чітко пояснити, де розташовані місця для паркування. Після всіх гарних вражень і переконань, які були прищеплені, вони пішли нанівець. Клієнт мав гроші і був готовий купувати, але передумав мати справу з таким бізнесом.

Саме для того, щоб уникнути подібних ситуацій, і існують такі рішення, як системи управління паркуванням транспортних засобів.

Близько 40%водіїв підтверджують, що вони уникали відвідування певного бізнесу через проблеми з паркуванням. Якщо бізнес не вирішить цю проблему, то з часом він неминуче втратить клієнтів.

Управління паркуванням ще тісніше пов'язане з інтересами підприємства в цілому, коли паркувальні місця монетизовані, тобто їх використання є окремою послугою, а не додатковою. Там, де на місцях, відведених під паркомісця, встановлені спеціальні лічильники, або де з авто власників стягується лише плата за в'їзд, керівництво не має можливості безпосередньо контролювати реальний процес, що відбувається на парковці. У такому випадку відвідувачі можуть користуватися паркувальними місцями безкоштовно, що призводить до втрати прибутку для бізнесу.

Агресивна поведінка на дорозі виникає, коли водії зляться, поспішають або втрачають контроль над своїми емоціями та поведінкою за кермом. Люди витрачають приблизно 17 годин на рік на пошуки місця для паркування і відчувають розчарування. Поведінка таких людей у дорожньому русі стає менш передбачуваною, що збільшує ймовірність аварій. Але це стосується не лише доріг загального користування.

Дослідження та аналітика вказують на такі цифри:

– 61 % водіїв кажуть, що вони нервують під час пошуку місця для паркування свого транспортного засобу [7];

– 23 % водіїв визнають, що проблеми з паркуванням спровокували їх на агресивну поведінку під час руху по дорозі;

– 33 % людей кажуть, що їхня агресивна поведінка за кермом проявлялася в активній жестикуляції руками [7];

– 47 % визнають, що кричали на інших учасників дорожнього руху;

– 24 % опитуваних підтвердили, що використовували свій транспортний засіб для блокування руху інших водіїв [7].

Усі перераховані вище фактори можуть вплинути на загальне враження про бізнес. Людина, яка приходить на автостоянку в напруженому емоційному стані, зайшовши в корпоративну будівлю, переносить свої емоції на інших, сприймаючи кожен рух і зусилля співробітника через призму агресії.

Більшість власників погоджуються, що вони хочуть мати кращі ділові стосунки з клієнтами, отримувати більше прибутку та загалом підвищити безпеку, але вони не хочуть платити комусь одному за моніторинг їх системи паркування. Розвиток нових тенденцій у сфері управління паркуванням передбачає участь широкого спектру технологій на основі програмного забезпечення та розширених можливостей спеціального технічного обладнання.

Можна вказати досить велику кількість причин, чому автоматизована система керування паркуванням має вирішальне значення для успіху бізнесу, незалежно від галузі його функціонування. Нижче наведено лише деякі з багатьох типів підприємств, які можуть отримати вигоду від використання програмного забезпечення для керування паркуванням:

– роздрібна торгівля – можливість надати клієнтам найкращий партнерський досвід з моменту входу на паркінг; коли клієнти відвідують магазин, вони вже мають позитивне ставлення;

– оренда нерухомості – забезпечує цілодобовий доступ до системи управління паркуванням, що дозволяє оперативно реагувати на зміни в умовах вашого партнерства та контролювати дотримання орендарями встановлених правил;

– готельне господарство та громадське харчування – відповідно до вимог клієнта для формування відчуття безпеки та зручності та створення приємного враження;

– заклади охорони здоров’я – для спрощення моніторингу паркувальних місць, уникнення стресових ситуацій для пацієнтів і відвідувачів, які займаються власними медичними проблемами, резервування місць для персоналу та на екстрений випадок;

– кампуси – університети стикаються з унікальними проблемами, пов’язаними з моніторингом великої кількості паркувальних місць, розташованих на великих територіях поблизу навчальних корпусів; адаптація системи до ваших потреб дозволить вам оптимізувати необхідні процеси моніторингу;

– муніципальні комплекси – місто робить ставку на монетизацію прибутку від паркувальних майданчиків, але її реалізація не виділяє належної кількості ресурсів; використання технологій для управління паркувальними місцями та паркуванням загалом збільшить муніципальні активи, зробивши їх безпечнішими та ефективнішими.

Отже, як бачимо дана предметна область є досить обширною, має свою специфіку, межі використання та необхідність на ринку.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Для якісного проектування та реалізації вказаної програмної системи необхідно проаналізувати, що вже розроблено на сьогоднішній день, які існують системи, визначити їх переваги та недоліки, а також вказати функціональні особливості.

АСУП СЕА (рисунок 1.1) підходить для контролю всіх типів автоматичних та напівавтоматичних паркінгів. ПЗ встановлюється на сервері паркувального комплексу і є багаторівневим клієнт-серверним додатком, що працює під контролем ОС Windows. Управління паркувальним обладнанням відбувається через локальну мережу [8].

Для забезпечення коректної роботи паркінгу необхідно внести в систему інформацію про паркувальні зони, тарифи, термінали (в'їзні, виїзні, оплати, контроль петель, табло), камери, карти доступу та ін. [8].

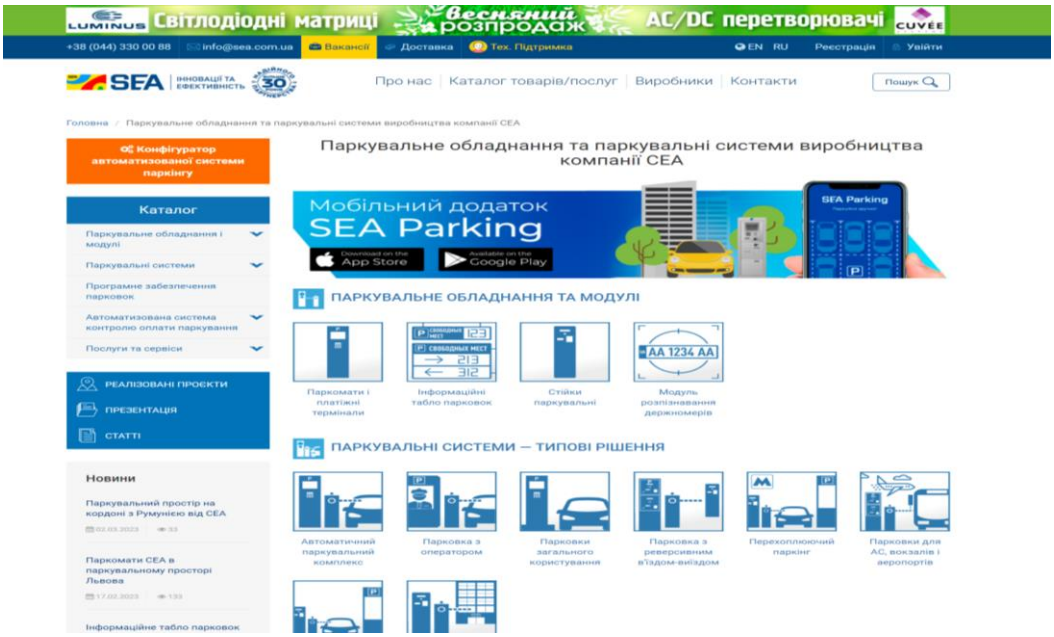


Рисунок 1.1 – Програмна система паркування SEA-Parking [8]

Далі наведено функціонал, який забезпечує роботу даної програмної системи автоматизованим керуванням паркінгом транспортних засобів.

Облікові записи користувачів

– адміністратор – повний доступ до всього функціоналу;

– менеджер – перегляд звітів, робота з касою, виконання моніторингу паркінгу (завантаженість зон, статус обладнання, події);

– оператор – робота з касою, перегляд даних про стан парковки.

Налаштування тарифів. ПЗ системи управління паркуванням дозволяє визначати і встановлювати лінійні і нелінійні (за часом доби, максимальною дією) тарифні плани для паркінгів [8].

Лінійна тарифікація, яка передбачає:

– найменування тарифу – довільний текст, відображається в інтерфейсі касира;

– тариф за кількість хвилин – дискретність тарифікації;

– вартість;

– паркувальний збір – фіксована сума, яка може стягуватися за послугу паркування;

– безкоштовний оплачуваний час – застосовується стандартна вартість хвилин в разі перевищення зазначеного часу перебування клієнта на паркінгу;

– безкоштовний безоплатний час;

– код тарифу – використовується для реєстрації продажів у фіскальному реєстраторі (для оплати штрафів виділений спеціальний код);

– зона застосування.

Прогресивна тарифікація, яка передбачає:

– найменування тарифу;

– безкоштовний час, який оплачується;

– безкоштовний час, який не оплачується;

– код тарифу;

– максимальний час;

– тариф, застосовуваний після максимального часу, – для оплати надлишкових або всіх хвилин користування парковкою;

– зона використання.

Додаткові опції:

– час початку дії – щодо календарної доби;

– тариф за кількість хвилин;

– вартість парковки.

Тарифікація для багатозонної парковки проводиться індивідуально для кожної зони: можна вказувати по декілька тарифних планів і налаштовувати будь-які з уже розроблених. Додатково передбачені тарифи для проїзних зон (на виїзді з яких встановлені переїзні термінали).

Функція автоматичного виїзду за розкладом. Для виїзних терміналів в інтерфейсі моніторингу стану обладнання доступний модуль налаштування автозапуску шлагбаума: дозволяє включити або відключити цю функцію, налаштувати розклад роботи за днями тижня (також і окремо) і часом доби.

Звітність. Передбачені наступні типи звітів:

– паркувальні сесії;

– дані платного паркінгу (разові відвідувачі);

– дані платного паркінгу (абонементи);

– дані безкоштовного паркінгу;

– статистика.

Підсистема автоматичного розпізнавання номерних знаків.

Модуль SEA ANPR [8] входить до складу системи автоматизованого управління паркінгом, взаємодіє з мережевими камерами та іншими джерелами відеопотоку. Програмний додаток встановлюється на додатковому сервері розпізнавання номерів або центральному сервері паркувального комплексу.

Якщо на парковці використовується модуль автоматичного розпізнавання державних номерних знаків автомобілів, то оператору стає доступним довідник зафіксованих транспортних засобів.

Ще одним підвидом програмної системи автоматизованого керування паркування транспортних засобів є АСКОП SEA (рисунок 1.2).

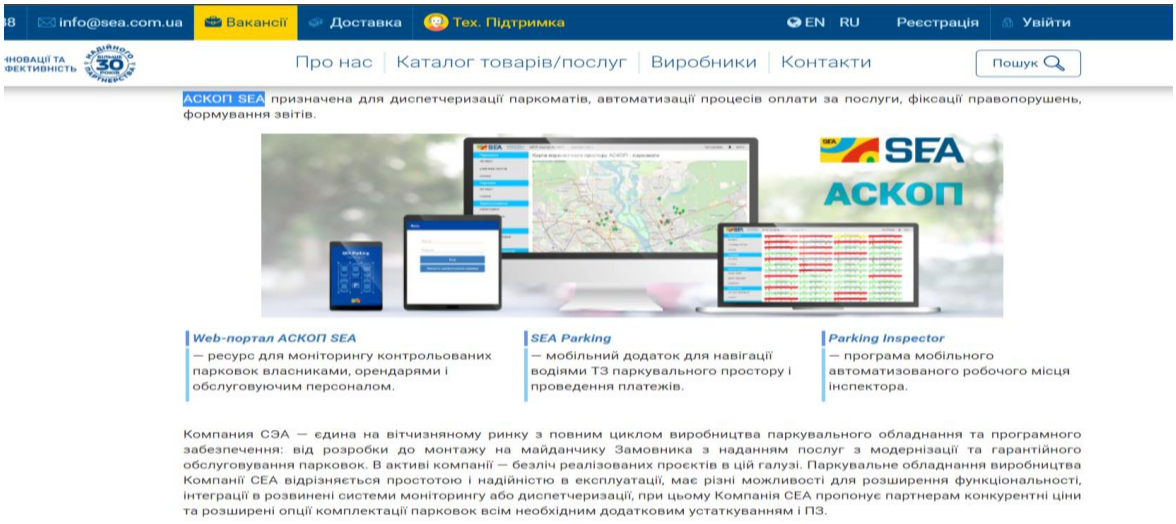


Рисунок 1.2 - АСКОП SEA [8]

Наступним для аналізу можна запропонувати SERVIO Parking (рисунок 1.3.)

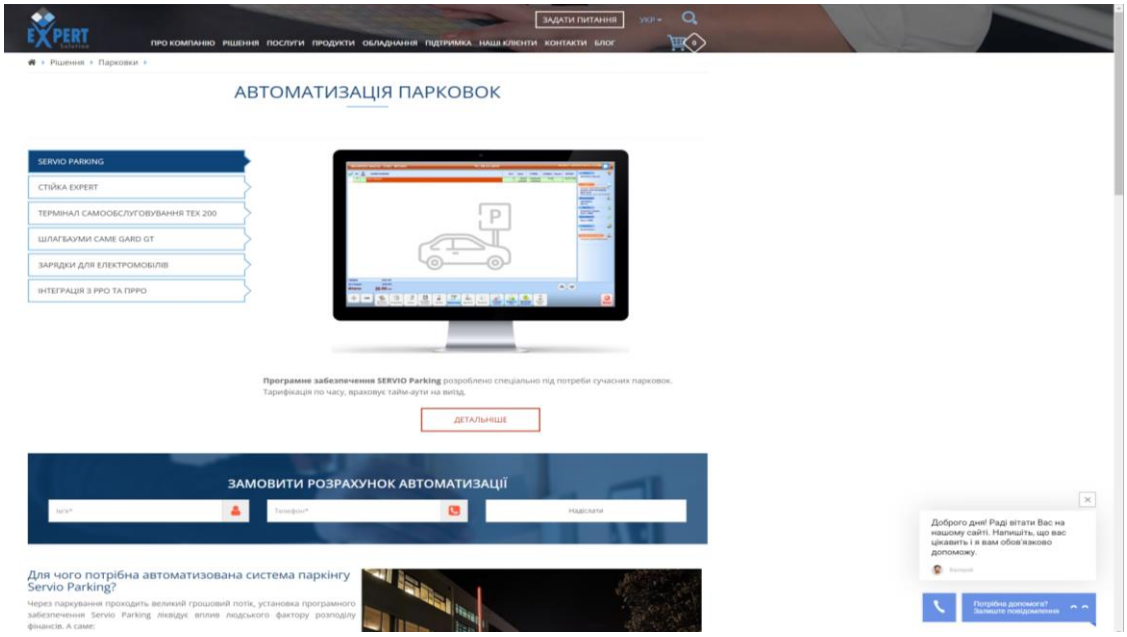


Рисунок 1.3 – SERVIO Parking [9]

Програмне забезпечення SERVIO Parking [9] - комплексне рішення для сучасних паркінгів, в яких оплата за використання паркувальних місць тарифікується за часом.

Програмне забезпечення SERVIO Parking розроблено під потреби сучасних паркінгів. Найчастіше використовується на парковках, де оплата за використання паркувальних місць тарифікується за часом (наприклад, торгові центри, житлові комплекси, готелі, ресторани, супермаркети, бізнес-центри, заміські комплекси, житлові комплекси).

Програмно-технічний комплекс дозволяє побудувати систему контролю та управління різного рівня складності. Servio Parking дозволяє монетизувати зону парковки, організувати контроль відвідування, проаналізувати активність та підвищити безпеку відвідувачів.

Програмне забезпечення підходить для контролю всіх типів автоматичних та напівавтоматичних паркінгів. Програма управління Servio Parking встановлюється на сервері паркувального комплексу. Управління паркувальним обладнанням відбувається через локальну мережу.

Функціональні можливості:

– робота з талонами, RFID картками (клієнтські, службові, абонементи);

– приймання оплати готівкою, банківською карткою, внутрішньою платіжною карткою;

– робота з РРО або пРРО;

– гнучка тарифікація по часу;

– доступні тайм-аути для виїзду;

– може працювати з пластиковими картами або паперовими квитками;

– налаштування часових рамок проїзду-виїзду;

– налаштування безкоштовного часу;

– інтеграція з готельними та ресторанними системами Servio;

– налаштування штрафів;

– ідентифікація транспортних засобів, що в'їжджають та виїжджають, через технологію розпізнавання номерів;

– керування обладнанням;

ПЗ SERVIO Parking керує таким обладнанням:

– термінал самообслуговування / платіжний термінал;

– паркувальна стійка для в'їзду;

– паркувальна стійка для виїзду;

– шлагбаум;

– індукційний контролер;

– інформаційне табло;

– інтеграція з електрозарядками GreenFuel.

Переваги використання програмної системи SERVIO Parking:

– повноцінне використання всіх можливостей програми обліку автомобілів на стоянці;

– автоматичний розрахунок вартості стоянки, виходячи із загального часу та тарифу;

– простий та зручний облік клієнтів, ведення клієнтської бази;

– створення гнучкої системи тарифів для будь-яких типів користувачів;

– ведення протоколу фінансових надходжень через платіжні термінали;

– створення необхідної кількості віддалених операторських місць;

– генерація звітів з будь-яких подій та операцій із заданою періодичністю.

1.3 Визначення вимог до програмної системи

Метою даної кваліфікаційної роботи є створення такої програмної системи, щоб за її допомогою можна було автоматизувати роботу парковки транспортних засобів.

Автомобільне паркування - платне або безкоштовне - стане значно комфортніше для користувача і ефективніше для її власника завдяки установці системи автоматизації паркування. Якщо паркування платне, то оплата паркування здійснюватиметься відповідно до встановленого тарифу у касирів-операторів або в автоматичних касових терміналах. Контроль в'їзду-виїзду та розрахунок вартості візьме на себе автоматика.

Система автоматизації безкоштовного паркування підрахує кількість вільних місць на кожному поверсі та відобразить це на спеціальних табло.

Вимоги - це набір прецедентів, тобто. модель функціонування системи та її оточення.

У споживачів та кінцевих користувачів є свої завдання (які в контексті UP називають потребами), вирішення яких має забезпечити комп'ютерна система.

Прецедент - це набір сценаріїв використання, в якому кожен екземпляр сценарію є послідовністю дій, що виконуються системою для досягнення відчутного для конкретного виконавця результату.

Сценарій (scenario) – це спеціальна послідовність дій чи взаємодій між виконавцями та системою. Його іноді називають екземпляром прецеденту (use case instance). Це один конкретний сценарій використання системи чи один прохід прецеденту.

Прецеденти програмної системи наведено нижче.

Початкове налаштування системи адміністратором:

– налаштування серверної частини системи;

– налаштування БД та дзеркалення клієнтської бази;

– налаштування системи тарифікації клієнтів;

Налаштування клієнтської частини системи;

– Налаштування інтерфейсу користувача;

Безпосереднє використання системи;

– Реєстрація нового клієнта та видача абонементу;

– Поповнення рахунку клієнта грошима;

– Зміна тарифного плану;

– Закриття рахунку клієнта;

– Разове використання паркування за допомогою жетону;

– Використання паркування за допомогою абонементу;

1. Початкове налаштування системи адміністратором

Адміністратор настроює параметри бази даних на сервері. Робота відбувається безпосередньо на серверному обладнанні або за допомогою спеціального програмного забезпечення для віддаленої роботи з будь-якої точки світу.

– Налаштування серверної частини системи;

– Налаштування БД та дзеркальної клієнтської бази;

Адміністратор використовує стандартний інтерфейс Windows для налаштування параметрів бази даних, включаючи резервне копіювання та дзеркало системи на зовнішньому носії за допомогою стандартного інтерфейсу MS Windows Server та MSSQL Server.

1.1.2 Налаштування системи тарифікації клієнтів.

Попередньо узгодивши питання цінової політики з керівництвом підприємства, адміністратор встановлює чинні тарифні плани на паркування.

Для цього він використовує спеціальне клієнт-серверне програмне забезпечення, розроблене компанією-виробником АІС.

1.2 Налаштування клієнтської частини системи

1.2.1 Налаштування інтерфейсу користувача

Адміністратор налаштовує інтерфейс терміналів оплати та наводить його на дружню форму.

2. Безпосереднє використання системи

2.1 Реєстрація нового клієнта та видача абонементу

Використовуючи сенсорну панель терміналу оплати, потенційний клієнт може зареєструватися в системі та отримати абонемент згідно з обраним тарифним планом.

2.2 Поповнення рахунку клієнта грошима

Використовуючи сенсорну панель терміналу оплати, клієнт за умови використання абонементу може поповнити свій віртуальний рахунок готівкою.

2.3 Зміна тарифного плану

Використовуючи сенсорну панель терміналу оплати, клієнт за умови використання абонементу може змінити особистий тарифний план на годинний, денний, триденний, тижневий, двотижневий або місячний.

2.4 Закриття рахунку клієнта

Використовуючи сенсорну панель терміналу оплати, клієнт за умови використання абонементу може закрити свій рахунок, якщо в ньому більше немає необхідності.

– Розгорнутий прецедент;

– Разове використання паркування за допомогою жетону;

– Основний виконавець: Клієнт;

Зацікавлені особи та їх вимоги:

Клієнт. Хоче поставити автомобіль на паркування, сплатити паркування, отримати чек про оплату.

Власник програмної системи. Хоче отримати кошти з клієнта.

Державні податкові служби. Бажають отримувати податок від кожного продажу.

Передумови. Клієнт натиснув кнопку на терміналі при в'їзді і отримав пластиковий жетон, в якому записано час в'їзду, номер стійки та інша інформація. Клієнт рухається за допомогою автомобіля.

Клієнт знаходиться усередині території паркування. Шлагбаум закрито. За жетоном здійснено оплату.

Основний успішний сценарій (або основний процес):

1. Клієнт натискає кнопку видачі жетону на терміналі;

2. Термінал видає клієнту жетон, на якому міститься інформація про час в'їзду, номер стійки та ідентифікаційний номер клієнта;

3. Шлагбаум відкривається;

4.Клієнт в'їжджає на територію паркування;

5. Шлагбаум закривається;

6. Клієнт здійснює оплату жетону в терміналі оплати.

Розширення (або альтернативні потоки)

1. Відсутній папір для друку фіскального чека у терміналі оплати

1.1 Система сповіщає клієнта про це за допомогою спливаючого повідомлення та пропонує записати номер транзакції вручну.

2. На парковці відсутні паркувальні місця

2.1 Система сповіщає про це клієнта за допомогою терміналу повідомлення, що випливає на екрані. На екрані висвічується час, що залишився до звільнення першого місця для паркування.

Спеціальні вимоги.

Потенційна можливість розширення фізичної паркувальної площі.

Підтримка 5 різних мов інтерфейсу, включаючи, українську, англійську, німецьку, французьку та іспанську.

Ударостійкий екран на терміналі при в'їзді.

Перезавантаження програми клієнта у разі збою або відсутності відгуку протягом 30 секунд.

Стійкість терміналів користувача до збоїв, викликаним використанням алгоритму, що виходить за рамки основного алгоритму (захист від непередбачуваних дій клієнтів).

1.4 Розробка технічного завдання програмної системи

1. Вступ

Основна мета цього документу полягає у визначенні вимог та попередньому перегляді деяких елементів керування паркінгом web-сервісу Smart Park. Розробка web-сервісу виконується для внутрішньої системи керування паркінгом, що надає послуги тимчасового та постійного зберігання різногабаритних транспортних засобів. Потенційними користувачами сервісу будуть касири та менеджери паркінгу.

2. Найменування й область застосування

Цей документ визначає вимоги до автоматизованої програмної системи керування паркінгом.

Сервіс дозволить:

– Вести облік запаркованих транспортних засобів

– Змінювати стан та умови паркування для конкретного транспортного засобу

– Стягувати плату за послуги згідно терміну та характеристик транспортного засобу

– Моніторити стан та грошовий баланс паркінгу

– Формувати фінансові документи та звіти

3. Підстава для розробки

Розробка програмної системи ведеться на підставі вимог, представлених у вигляді завдання до кваліфікаційної роботи бакалавра. Організацією, що

затверджує даний документ є Хмельницький національний університет, а представником виступає студент-розробник. Датою затвердження вважати перший робочий день другого тижня навчального семестру, тобто 1 березня 2023 року.

4. Призначення розробки

4.1.Функціональне призначення

Програмна система надає можливість ефективно та надійно керувати внутрішніми процесами паркінгу. До них відносяться: відображення актуальної інформації про наповненість паркомісць транспортними засобами відповідних категорій, ведення детального обліку фінансових операцій, оптимізація процесів створення, редагування, та заповнення шаблонів звітності для клієнтів та менеджерів паркінгу, виконання фінансових операції по оплаті наданих послуг та здійснення автоматичних списань коштів з балансів відповідних транспортних засобів згідно встановлених керівництвом паркінгу умов.

4.2.Експлуатаційне призначення

Даний ПС є повноцінною системою керування паркінгом, що дозволить оптимізувати процедуру реєстрації та забезпечити умови зберігання різногабаритних транспортних засобів відповідно до потреб клієнтів. Впровадження сервісу в систему менеджменту гарантуватиме підвищення рівня надійності в проведенні фінансових операцій, веденні документації та обробці даних, що дозволить скоротити витрати завдяки автоматизації процесів та зменшенню кількості необхідного обслуговуючого персоналу.

5. Технічні вимоги до програми або програмного виробу

5.1.Вимоги до функціональних характеристик

Сервіс складається з двох основних компонент: клієнт та сервер. Між ними повинна бути реалізована взаємодія за допомогою HTTP-запитів.

5.2.Вимоги до надійності

5.2.1. Вимоги до забезпечення функціонування програми

Користувачу, який працює з сервісом за допомогою web-браузера, повинен бути наданий безлімітний доступ до ПС, який розміщений по

визначеній URL адресі. ПС повинна забезпечувати обробку критичних ситуацій для забезпечення безперебійного доступу.

5.2.2. Час відновлення після виникнення критичної ситуації

У випадку відмови роботи серверної частини й подальшій відсутності доступу до програмної системи, час відновлення не повинен перевищувати 6 годин.

5.2.3. Виникнення критичної ситуації через некоректну взаємодію

Після запуску програми на сервері, виникнення критичної ситуації під час роботи програмної системи в наслідок дій користувача повинно бути унеможливлено.

5.3.Вимоги до експлуатації

5.3.1. Вимоги до виду обслуговування

Обслуговування не є необхідним.

5.3.2. Вимого до чисельності та кваліфікації персоналу

Для керування системою достатньо одного спеціаліста, в обов’язки якого входитиме обслуговування серверу.

5.4.Вимоги до складу й параметрів технічних засобів

Версія ОС	Windows 7 і вище	MacOS 10.12 і вище
Процесор	Pentium 4 або новіший з підтримкою SSE2	Intel x86, Apple Silicon або новіший
ОЗП	512 MB для 32-bit, 2 GB для 64-bit	512 MB
Дисковий пр.	200 MB	

5.5.Вимоги інформаційної та програмної сумісності

Для розробки програмної системи повинен використовуватися ASP.NET Core MVC.

5.6.Вимоги до транспортування та зберігання

Вихідний код програмної системи повинен зберігатися в публічному репозиторії на GitHub. Вимоги до програмної документації. Ведення документації повинно виконуватися згідно ISO/IEC/IEEE-29148-2018.

1.5 Висновки до 1-го розділу. Постановка задачі.

Отже, в результаті виконання 1-го розділу було здійснено аналіз предметної області. Виявлено, що створення програмної системи для автоматизації роботи паркінгу для транспортних засобів є актуальним завданням, оскільки із зростанням кількості власників транспортних засобів виникає нагальна потреба у створенні та управлінні якомога більшою кількістю зон для паркування транспортних засобів. Паркування є основною частиною транспортної проблеми, і ця частка швидко зростає, оскільки кількість місць для паркування обмежена.

Метою роботи є розробка автоматизованої системи керування паркінгом транспортних засобів.

Об’єкт дослідження – процес розробки автоматизованої системи керування паркінгом.

Предмет дослідження – методи та засоби, що забезпечують систему керування паркінгом.

Виходячи із мети, об’єкту, предмету можна виділити такі завдання, щоб досягти позитивного результату під час розробки:

– здійснити детальний та коректний аналіз предметної області та проаналізувати сучасний стан наявного програмного забезпечення даної предметної області;

– визначити вимоги до програмної системи;

– здійснити проектування програмної системи із визначенням базових рішень, що будуть реалізовані в процесі розробки;

– здійснити розробку функціональної схеми автоматизованої системи паркінгу транспортних засобів;

– здійснити розробку програмного продукту, що має дружнім, інтуїтивно зрозумілий для користувач інтерактивним інтерфейс;

– провести тестування та впровадження програмного продукту.

2 ПРОЕКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Проектування архітектури та функціональної структури веб-ресурсу

Проектування архітектури програмної системи є критичним етапом у розробці будь-якої програми або програмної системи. Ось декілька причин, чому воно є важливим:

1. Розуміння вимог: правильне проектування архітектури допомагає розібратися у вимогах до системи. Воно дозволяє визначити, які компоненти потрібні, як вони взаємодіють між собою та які будуть вимоги до продуктивності, безпеки та інших аспектів системи.

2. Масштабованість: проектування архітектури дозволяє створити систему, яка легко масштабується. Це особливо важливо, якщо система передбачає зростання обсягу даних, користувачів або функцій. Добре спроектована архітектура дозволить легко розширювати систему, додавати нові функції та забезпечувати продуктивність.

3. Повторне використання: проектування архітектури допомагає створити модульну систему, де окремі компоненти можна повторно використовувати. Це покращує продуктивність розробки, спрощує тестування та забезпечує більшу стабільність системи.

4. Продуктивність та ефективність: правильне проектування архітектури дозволяє оптимізувати використання ресурсів системи, таких як процесор, пам'ять та мережа. Це може значно покращити продуктивність та ефективність системи, зменшуючи витрати на обладнання та операційні витрати.

5. Забезпечення якості: проектування архітектури допомагає забезпечити якість програмної системи. Це означає розробку системи з врахуванням принципів чистого коду, модульності, легкості тестування та інших аспектів, що сприяють стабільності та підтримці системи.

Узагалі, проектування архітектури допомагає зрозуміти складність системи, знижує ризики неправильної реалізації та сприяє створенню стійкої, масштабованої та ефективної програмної системи.

Трьохрівнева архітектура (рисунок 2.1) є широко використовуваним шаблоном проектування для бізнес-застосунків, який забезпечує ефективне розподілення відповідальностей між різними рівнями системи без надмірних ускладнень. Вона не обмежується конкретною реалізацією користувацького інтерфейсу.

На першому рівні, який називається "логіка представлення", визначається спосіб взаємодії користувача з програмним забезпеченням. Це може бути командний рядок, графічний інтерфейс з багатим функціоналом або веб-інтерфейс на основі HTML. Головним завданням цього рівня є відображення інформації для користувача та інтерпретація його команд для роботи з доменною логікою та джерелом даних.

Другий рівень, відомий як "логіка джерела даних", відповідає за взаємодію з іншими системами або базами даних, що виконують завдання в межах програмного застосунку. Це можуть бути системи моніторингу транзакцій, системи передачі повідомлень тощо. У більшості корпоративних застосунків головною частиною цього рівня є база даних, яка забезпечує зберігання постійних даних.

Останній рівень, відомий як "логіка домену" або "бізнес-логіка", виконує завдання, пов'язані з діловими процесами та логікою домену, з яким працює програмний застосунок. Вона включає обчислення на основі вхідних даних та даних зберігання, перевірку валідності даних, отриманих з рівня представлення, і визначення логіки взаємодії з джерелом даних в залежності від команд, отриманих з рівня представлення.

Така архітектура дозволяє забезпечити чітке розділення відповідальності між різними рівнями системи, сприяє підтримці, тестуванню та розширенню коду, а також забезпечує гнучкість і масштабованість програмного застосунку.

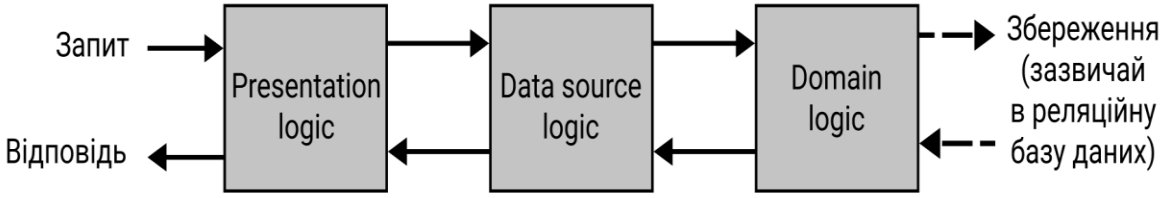


Рисунок 2.1 – Схематичний рисунок трьохрівневої архітектури

Принцип роботи трьохрівневої архітектури базується на розділенні функцій програмної системи на три логічні рівні: логіка представлення (Presentation layer), логіка домену (Domain layer) і логіка джерела даних (Data source layer). Кожен з цих рівнів виконує свої завдання і має свої відповідальності.

1. Логіка представлення (Presentation layer): Цей рівень відповідає за взаємодію з користувачем і представлення інформації. Він включає в себе різноманітні інтерфейси, такі як графічний інтерфейс користувача, веб-інтерфейс або API. Логіка представлення забезпечує відображення даних користувачу, обробку його введення та передачу відповідних команд до логіки домену.

2. Логіка домену (Domain layer): Цей рівень включає бізнес-логіку програмної системи і представляє основну функціональність додатку. Він обробляє всі обчислення, перевірки даних, бізнес-правила та ділові процеси, пов'язані з доменною областю. Логіка домену не залежить від конкретного інтерфейсу користувача або джерела даних, що дозволяє їй бути незалежною та повторно використовуватись.

3. Логіка джерела даних (Data source layer): Цей рівень відповідає за доступ до джерел даних, таких як бази даних, зовнішні сервіси або інші джерела інформації. Він забезпечує збереження і отримання даних, виконання запитів та маніпулювання даними відповідно до вимог логіки домену. Логіка джерела даних ізолює роботу з конкретними джерелами даних від інших частин системи, що сприяє зручності зміни або розширення джерел даних.

Принцип роботи трьохрівневої архітектури полягає в тому, що кожен з цих рівнів має свою відповідальність і може працювати незалежно від інших. Взаємодія між рівнями здійснюється через чітко визначені інтерфейси, що дозволяє підтримувати модульність, розширюваність і тестованість системи. Крім того, така архітектура спрощує розподілену розробку та підтримку системи, оскільки кожен рівень може бути реалізований окремо командою розробників.

2.2 Визначення основних модулів програмної системи

Для повноцінної роботи розроблюваного програмної системи необхідно визначити модулі, з яких буде складатись цей програмний продукт. На рисунку 2.2 подано опис фізичної структури програмної системи.

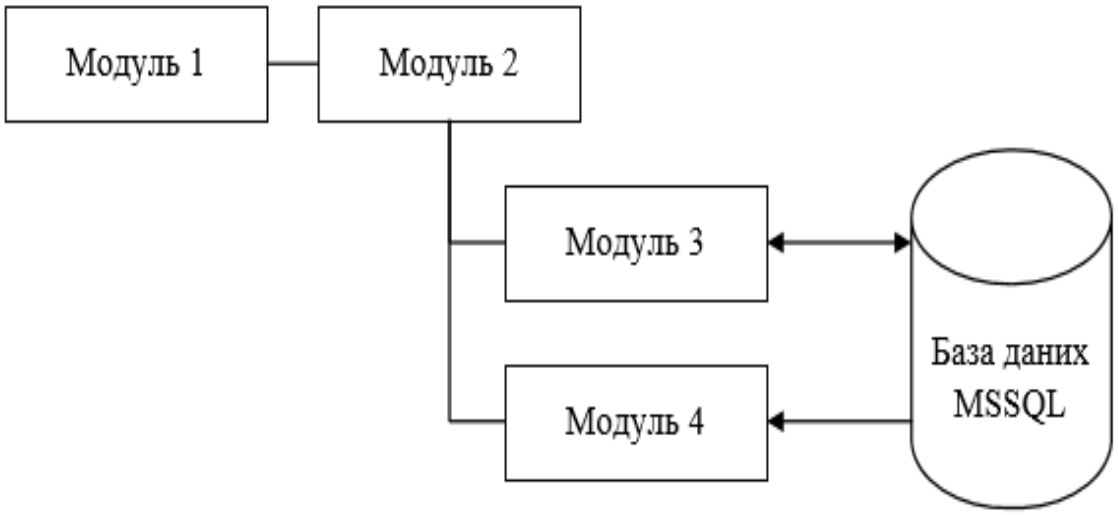


Рисунок 2.2 - Функціональна схема системи паркінгу

У таблиці 2.1 представлено опис модулів структури програмної системи, визначеної на основі рисунку 2.2.

Таблиця **Помилка! У документі відсутній текст указаного стилю.**.1 – Призначення програмних модулів

№ з/п	Позначення	Призначення
1	Модуль 1	Модуль авторизації.
2	Модуль 2	Головний модуль, що забезпечує перехід між функціональними блоками.
3	Модуль 3	Модуль для роботи з транспортними засобами.
4	Модуль 4	Модуль для роботи з даними.

Специфікація процесів відповідно до посади:

– касир – на початку робочої зміни авторизується в системі за допомогою логіну та пароля. Під час прибуття транспортного засобу, касир виконує його реєстрацію та поповнює баланс транспортного засобу. Під час відпуску, касир знімає з обліку транспортний засіб, ввівши значення номерного знаку. При кожній успішній фінансовій операції касир надає клієнту роздрукований чек. Після завершення робочої зміни, касир повинен виконати деавторизацію [2].

– менеджер – дана програмна система є допоміжним менеджерським інструментом та використовується лише для зберігання даних, зібраних під час роботи паркінгу. Для надання доступу до даних, менеджер повинен виконати авторизацію в системі. В своїй роботі менеджер використовує дані про грошові баланси паркінгу (загальний, поточний та посесійний), дані про фінансові транзакції (загальні, поточні та посесійні), дані про наповненість паркінгу та дані про транспортні засоби, що знаходяться на території паркінгу. Менеджер має змогу вивести опрацьовану інформацію в електронному форматі, чи роздрукувати її. Деавторизація менеджера відбувається після певного терміну бездіяльності в системі, чи за його бажанням [2].

Касир:

– система авторизації;

– реєстрація транспортного засобу;

– поповнення балансу транспортного засобу;

– відпуск транспортного засобу;

– друк чеку відповідно до проведеної операції [2].

Менеджер:

– система авторизації;

– збір даних по категоріям:

- баланс паркінгу (загальний, поточний, посесійний);
- фінансові транзакції (загальні, поточні та посесійні);
- стан паркінгу (наповненість, поточні умови експлуатації);
- транспортні засоби, що знаходиться на території паркінгу;

– сортування відібраних даних за властивостями;

– експорт відібраних даних в формати електронних документів такі як: .docx, .xlsx, .pdf, .png, .jpeg, .svg та ін.;

– друк відібраних даних [2].

Для візуалізації інформаційних потоків у системі широко використовуваним способом є діаграма потоків даних (Data flow diagram). Вона графічно відображає вимоги до реалізації рішення, потоки вхідної та вихідної інформації, зміни цієї інформації та місця її зберігання. Діаграма потоків даних може слугувати засобом комунікації між розробниками та учасниками системи і є основою для внесення змін у процес розробки.

Існують два типи нотацій для діаграм потоків даних: Йордана і Коада, та Гейна і Сарсона, які використовують різні візуальні представлення для процесів, сховищ даних, потоків даних та зовнішніх об'єктів. Основна відмінність між ними полягає у вигляді процесів. У першому випадку, вони зображуються у формі кола, у другому - у формі квадрата з закругленими кутами [2].

Відповідно до нотації, діаграма використовує наступні символи:

– Процеси (перетворюють вхідний потік даних у вихідний).

– Сховища даних (репозиторії даних в системі, також відомі як файли).

– Потоки даних (магістралі, по яких рухаються пакети інформації).

– Зовнішні сутності (об'єкти, що знаходяться поза системою і взаємодіють з нею, вони є джерелами та отримувачами даних системи).

На рисунку 2.3 наведена діаграма потоків даних у нотації Гейна і Сарсона, яка є більш поширеною для візуалізації інформаційних систем. У даній діаграмі клієнт є джерелом даних для системи, надаючи інформацію про властивості транспортного засобу та свої персональні дані. Це дозволяє виконати процес реєстрації, який полягає у внесенні відповідних записів до бази даних у вигляді моделей [2].

У подальшому обслуговуванні клієнта всі дані, необхідні для виконання операцій, будуть отримуватися зі сховища за унікальним ідентифікатором, який може бути, наприклад, брендованою карткою паркінгу або роздрукованим талоном [2].

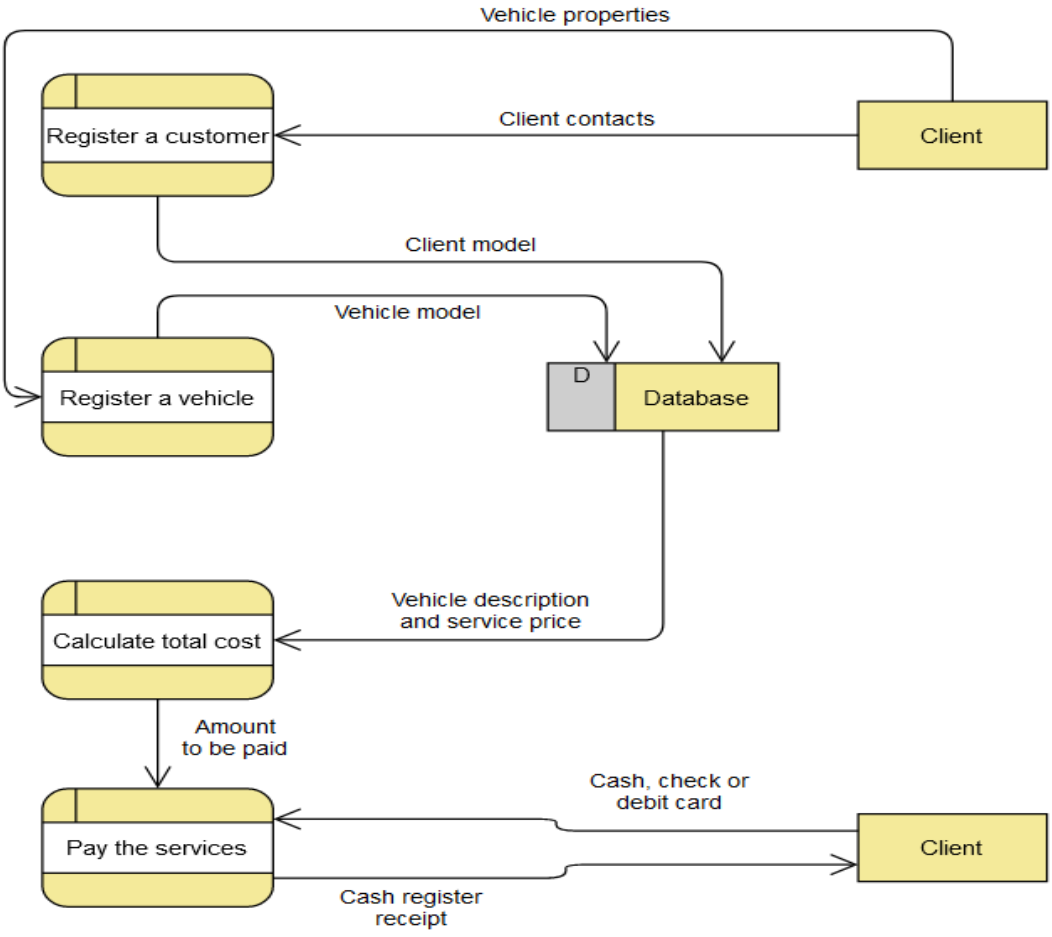


Рисунок **Помилка! У документі відсутній текст указаного стилю.**.3 – DFD для оформлення послуги «Розміщення транспортного засобу на території паркінгу»

Діаграма переходів станів (State-transition diagram - STD) відображає всі можливі стани, які може мати об'єкт, а також події, які спричиняють зміну стану об'єкта (переходи), умови, які повинні бути виконані перед переходом (guards), і дії, які виконуються під час існування об'єкта (дії). Ці діаграми корисні для визначення поведінки окремих об'єктів у різних сценаріях використання, які впливають на ці об'єкти.

На рисунку 2.4 показана послідовність станів, які відповідають умовам системи. Після отримання даних система виконує процес реєстрації, що призводить до надання права доступу до місця паркування. Надання послуги може бути скасовано в будь-який момент. У такому випадку клієнт отримає повернення коштів. Коли оплачений час перебування буде вичерпано, клієнт може продовжити або припинити використання місця паркування. Після завершення процесу відпуску транспортного засобу видається чек про надану послугу [2].

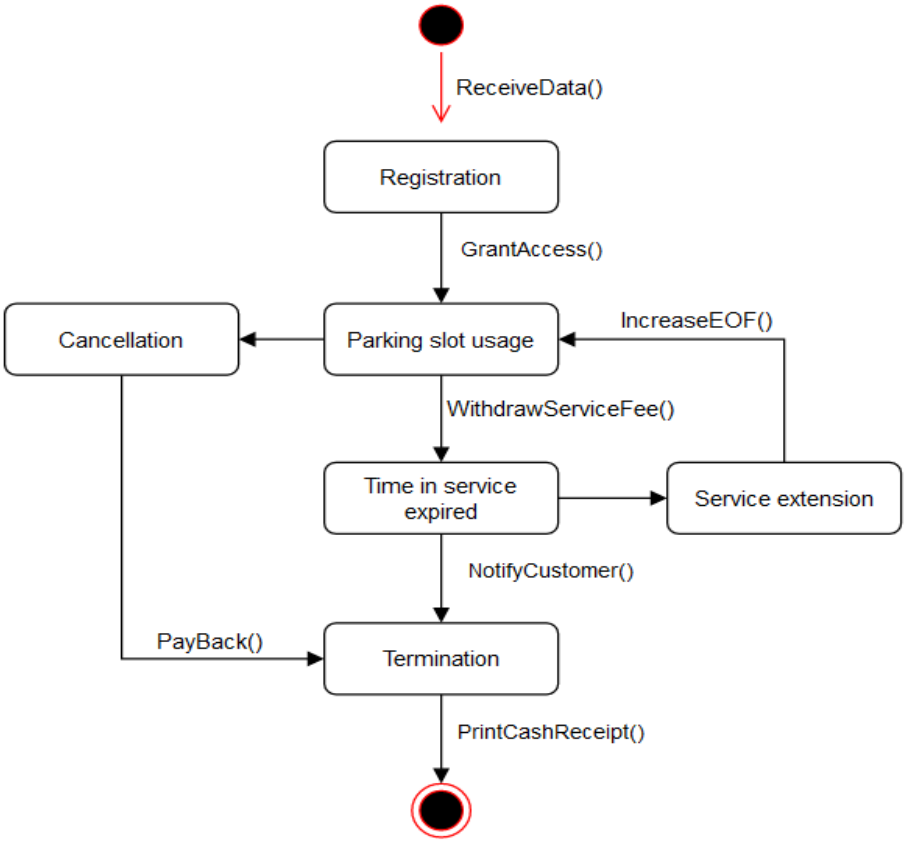


Рисунок **Помилка! У документі відсутній текст указаного стилю.**.4 – Діаграма переходів станів (STD)

Діаграма варіантів використання (Use case diagram) – це динамічна діаграма або діаграма поведінки в UML. Вони моделюють функціональність системи за допомогою діючих осіб і варіантів використання., що являються набором дій, послуг і функцій, які повинна виконувати система. Діючи особи, в цьому випадку, це люди або організації, які виконують певні ролі в системі [2].

Діаграми даного типу використовуються для візуалізації функціональних вимог до системи, які будуть трансформовані в дизайні рішення та пріоритети процесу розробки. Вони також допомагають визначити будь-які внутрішні та зовнішні фактори, які можуть вплинути на систему й повинні бути взяті до уваги.

В програмній системі передбачено рольові моделі. На рисунку 2.5 подано три дійові особи: менеджер, касир та клієнт. Касир, так само як і клієнт, бере участь в операціях реєстрації, поповненні балансу, відпуску транспортного засобу. Обидва працівники (касир та менеджер) повинні обов’язково перед початком роботи виконати аутентифікацію в системі. Менеджер має змогу працювати лише з даними, які збираються та зберігаються в процесі надання послуг [2].

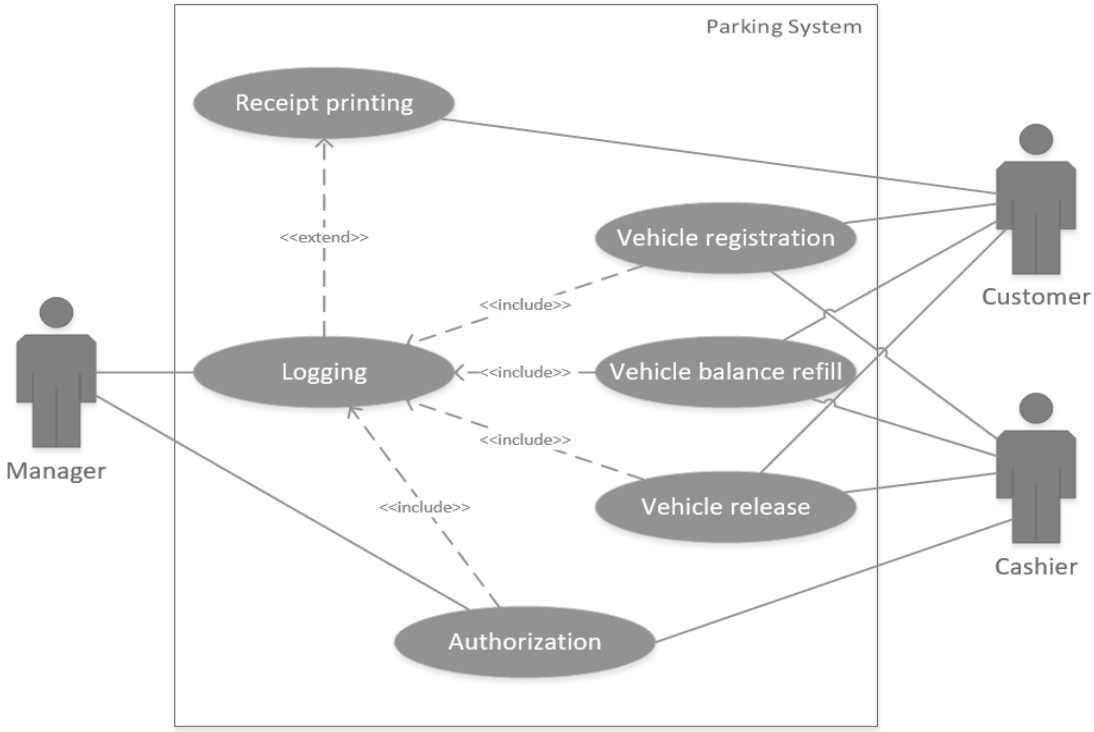


Рисунок **Помилка! У документі відсутній текст указаного стилю.**.5 – Діаграма варіантів використання

Методологія IDEF0 (Integration Definition for Process Modeling) - це public-domain методологія, яка використовується для моделювання та графічного відображення бізнес-процесів. Вона дозволяє відображати логічні зв'язки між діями системи, не звертаючи увагу на послідовність їх виконання у відношенні до часу. Об'єкти на діаграмі зображуються у вигляді ієрархічного дерева. Цей підхід до побудови спрощує розуміння предметної області і дозволяє легко вдосконалювати її реалізацію.

На рисунку 2.6 представлено найбільш вживаний сценарій надання послуги.

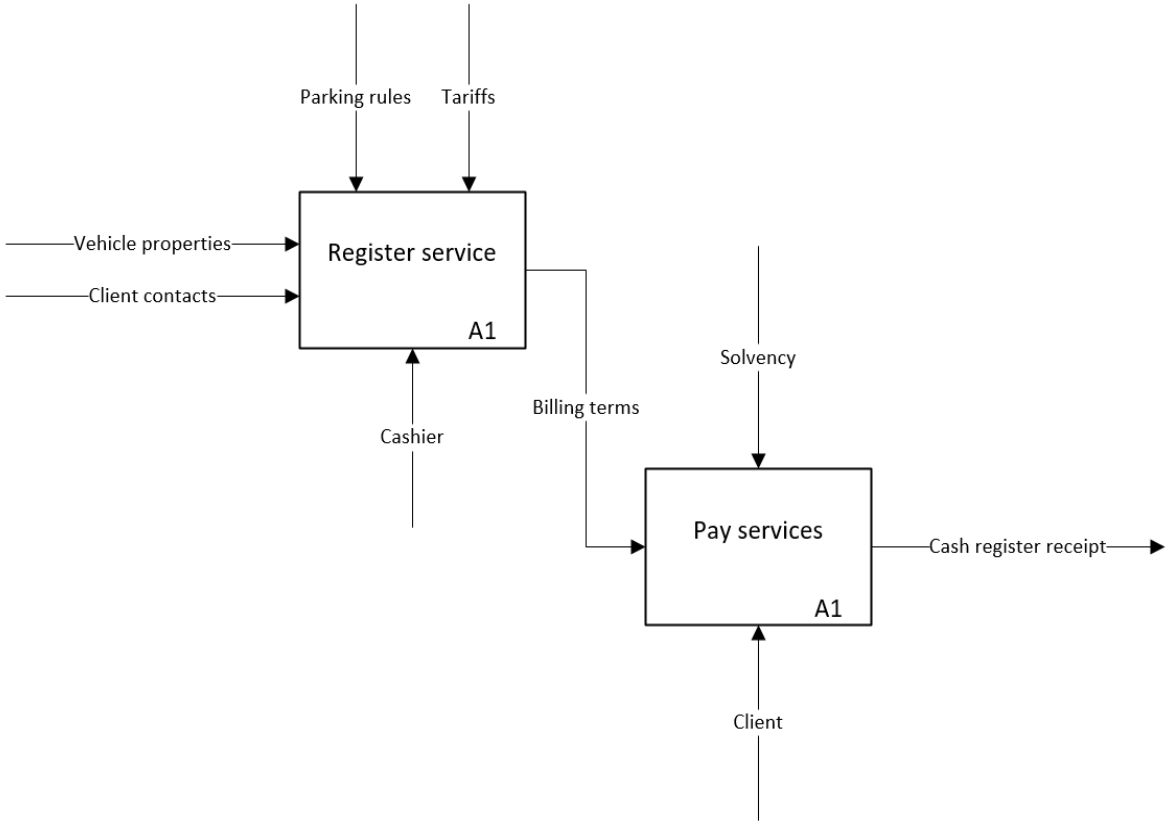


Рисунок **Помилка! У документі відсутній текст указаного стилю.**.6 – Опис надання послуги паркування за допомогою IDEF0 – (функціональна діаграма (методика SADT))

Клієнт паркінгу під'їжджає на транспортному засобі до місця реєстрації. Касир вводить номерний знак транспортного засобу, обирає тип транспортного засобу та суму для поповнення балансу транспортного засобу. В разі успішного виконання операції, друкується чек в якому вказуються інформація про умови зберігання даного клієнтського транспортного засобу (термін перебування, оплачена сума послуги згідно тарифу, номер місця для паркування, надана інформація про транспортного засобу, дата та час виконання оплати).

2.3 Проектування логічної моделі бази даних

Модель представляє собою набір даних і правил, що використовуються для роботи з даними у програмному продукті, додатку або ресурсі. Вся

структура додатка моделюється за допомогою даних, які обробляються згідно з встановленими правилами. Таким чином, всі типи даних повинні підпорядковуватись певним правилам (наприклад, дата не може бути у майбутньому, електронна пошта або номер телефону мають бути у певному форматі, ім'я не може бути довшим за певну кількість символів тощо).

Для розробки логічної моделі бази даних для інтернет-ресурсу з торгівлею одягом була обрана реляційна модель бази даних. Реляційна база даних базується на реляційній моделі даних, де зв'язки (реляції) визначаються між сутностями. Ця модель даних включає сутності та зв'язки між ними, атрибути з унікальними ключами для кожної окремої сутності.

Реляційна база даних зазвичай складається з таблиць, які, у свою чергу, складаються з стовпців і рядків. Кожен стовпець має властивості, такі як назва і тип даних.

Основні особливості реляційних баз даних:

– модель є логічною, що означає, що відносини між даними є логічними (абстрактними) структурами, а не фізичними (збереженими).

– уся інформація бази даних представлена одним і єдиним способом - явним визначенням значень атрибутів;

– реляційна модель бази даних є логічною, що означає, що відношення між даними є абстрактними структурами, а не фізичними;

– інформація бази даних представлена єдиним способом - через явне визначення значень атрибутів;

– відсутність покажчиків або адрес, які зв'язують значення між собою.

Реляційна алгебра дозволяє реалізувати декларативне програмування та декларативний опис обмежень цілісності, додатково до навігаційного програмування та процедурної перевірки умов.

Основні властивості таблиць у реляційних базах даних такі:

– Кожна таблиця має унікальне ім'я.

– Рядки в таблиці не можуть бути однаковими.

– Кожен стовпець має унікальне ім'я.

– Порядок рядків у таблиці не має значення, якщо не задано сортування.

– Структуру таблиці можна описати наступним чином:

– Для кожного стовпця вказується його ім'я та тип даних.

У реляційних базах даних використовуються зв'язки між таблицями. Для встановлення зв'язків необхідно мати наступні елементи:

Первинний ключ, який є набором атрибутів, що однозначно ідентифікують унікальність рядка, зазвичай це одне поле.

Ідентифікатор (ID), що автоматично збільшується при додаванні запису.

Зовнішній ключ, який є набором атрибутів таблиці, що однозначно ідентифікують унікальність рядка в іншій таблиці.

Зв'язок між первинним ключем і зовнішнім ключем.

У реляційних базах даних можуть існувати такі типи зв'язків між сутностями [2, 5]:

– один до одного;

– один до багатьох;

– багато до багатьох.

1.Одному рядку таблиці відповідає один рядок в іншій таблиці.

2.Одному рядку таблиці відповідає один чи декілька рядків в іншій таблиці, але одному рядку в другій таблиці відповідає тільки один рядок в першій таблиці.

Одному рядку в першій таблиці відповідає один чи декілька рядків в другій таблиці, і навпаки. У такому випадку створюється додаткова таблиця зі своїм первинним ключем та двома зовнішніми ключами до першої та другої таблиці для збереження цього зв'язку.

Для більш детального та якіснішого розуміння логіки бази даних потрібно спроектувати ER-діаграму. Діаграма «сутність – зв’язок» (Entity Relationship diagram – ERD) відображає взаємозв'язки між різними сутностями (наприклад, клієнтами та товарами) в базі даних. Вона складається з трьох ключових елементів, які відображають загальну графічну структуру:

Сутності (entities) - це таблиці в базі даних, які узагальнюють класифікацію понять, місць, подій, предметів, об'єктів тощо. На діаграмі вони зображуються у вигляді прямокутників з полями для назв.

Атрибути (attributes) - це факти або властивості, які описують кожну таблицю окремо. Вони також представлені як іменники і, зазвичай, становлять стовпці в таблицях. На діаграмі вони розміщуються всередині прямокутників у вигляді записів.

Зв'язки (relationships) - використовуються для позначення асоціацій між сутностями. Вони зображаються лініями з прямокутними засічками на кінцях. Назви зв'язків є дієсловами.

ER-діаграма є потужним інструментом для моделювання даних, який не залежить від конкретного програмного забезпечення. Вона є основою для різних моделей даних, таких як мережеві, об'єктні, ієрархічні тощо.

Згідно з рисунком 2.4, видно, що основними сутностями бази даних є "Послуга" (Service), яку надає паркінг, і "Працівник" (Employee), який виконує обслуговування клієнтів і моніторинг. Працівник може виконувати роль менеджера або касира, тому між сутностями "Працівник" (Employee) і "Посада" (Position) встановлено зв'язок "багато до багатьох", який реалізовується через проміжну таблицю "Посада працівника" (EmployeePosition) згідно з процесом нормалізації бази даних.

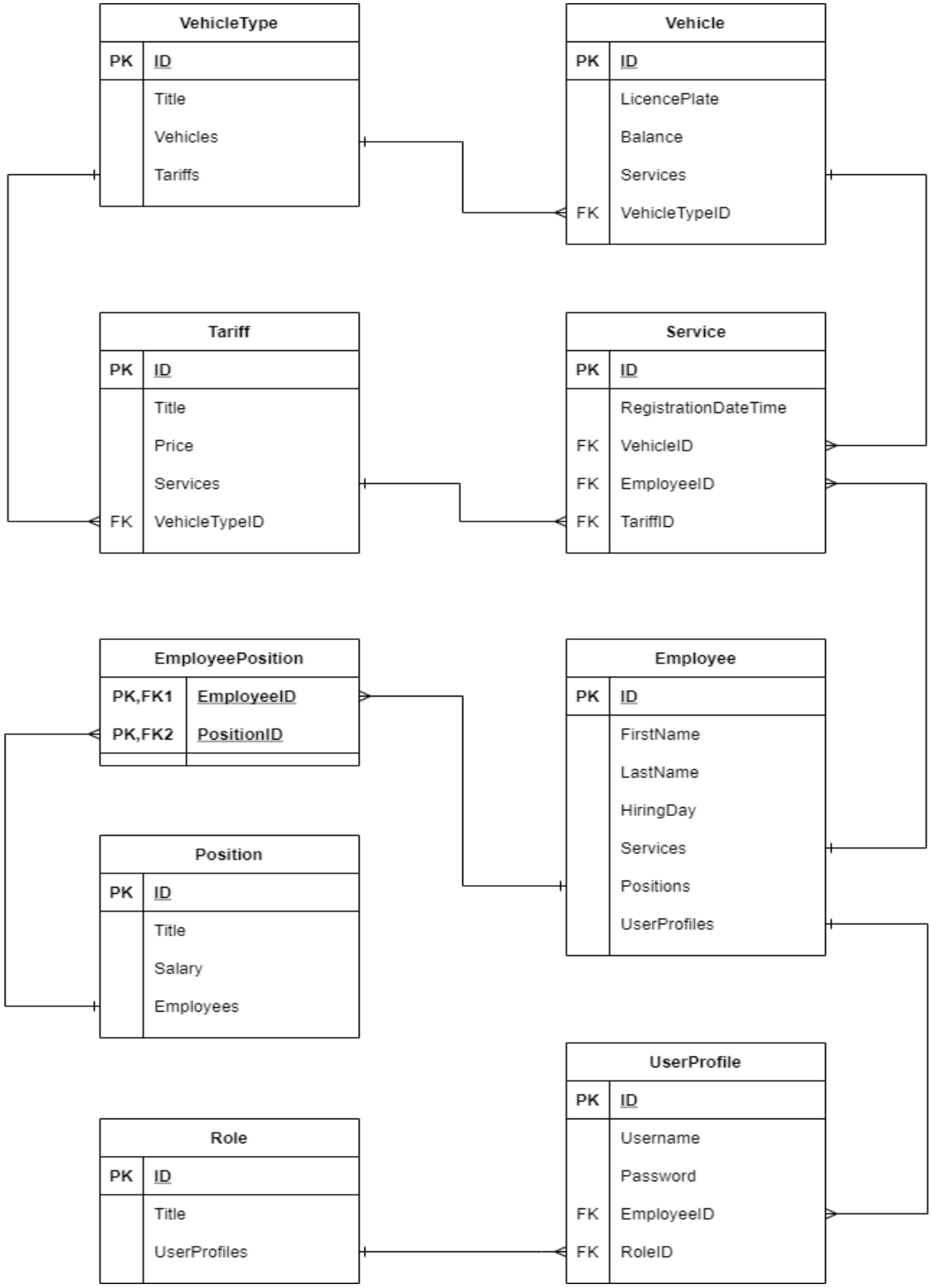


Рисунок 2.4 - ER-діаграма бази даних

2.4 Проектування інтерфейсу

Програмна система перш за все повинна бути зручною та у використанні не викликати труднощів, враховуючи можливі способи ідентифікації користувачів. Тому, виходячи з розробленої специфікації програмної архітектури модуля було сформовано методику організації використання програмної системи кінцевим користувачем в залежності від типу ідентифікації. У рамках розробленої архітектури модуля проектування інтерфейсу користувачеві пропонується самому скласти опис своєї професійної діяльності на природному мовою. Ця методика лежить у рамках концепції управління структурою інтерфейсу користувача з урахуванням варіативності вимог предметної області, яка застосовується в категорії конфігурованих систем, призначених для користувачів, щоб дати їм можливість самостійно змінювати систему залежно від потреб предметної області. Залежно від вимог, що змінюються, предметної області може змінюватися і структура виконуваної професійної діяльності, а саме - склад та послідовність виконуваних Користувачем дій. У разі таких змін користувальницький інтерфейс також повинен бути адаптований до умов, що змінилися, щоб користувачеві було зручно виконувати свої професійні завдання.

Кроки методики організації використання програмної системи кінцевим користувачем:

1. Завантаження документів та визначення інформаційних частин документів;

2. Опис процесу діяльності у предметній галузі самим користувачем у вигляді чотирирівневого дерева, що відповідає концепції декомпозиції мети на дії, завдання, операції та кроки;

3. Для кожного кроку діяльності користувача відбувається визначення даних, що використовуються у процесі його виконання;

4. Процес перетворення інформаційних частин документів на прив'язки до дій у набір форм та елементів інтерфейсу, за допомогою моделі дій користувача

у вигляді механізмів та методики вибору елементів та формування структури інтерфейсу;

5. Наповнення структури інтерфейсу елементами.

Залежно від конфігурації, система керування паркуванням має кілька типів ідентифікації користувачів.

В'їзний квиток із штрих-кодом. Це найпростіший спосіб ідентифікації. Паркувальна стійка біля входу обладнана принтером для друку квитків і відвідувачі отримують паперовий квиток при вході. Штрих-код заноситься до бази даних і прив'язується до конкретного автомобіля. При вильоті цей квиток зчитується спеціальним сканером штрих-коду - або касиром або терміналом самообслуговування.

Безконтактні пластикові карти. Паркувальні місця оснащені пристроєм видачі та зчитування карток для видачі та збору карток у відвідувачів. Ця карта може бути використана для різних знижок у комплексі.

Номерні знаки. Номери автомобіля використовуються як ідентифікаційний знак транспортного засобу. При під'їзді до шлагбауму номерний знак фіксується та зв'язується з автомобілем у базі даних - при виїзді зі шлагбауму номерний знак знову фіксується та система розраховує ціну. Все аналогічно звичайному квитку, але на пунктах реєстрації та виїзду (або на платіжних терміналах) встановлено камери, які фіксують номерний знак. Якщо записати номерний знак неможливо, клієнт автоматично видає талон зі штрих-кодом. Автоматизація паркування – основні елементи системи.

Автоматична система паркування працює наступним чином: людина натискає кнопку для в'їзду на паркування та отримує квиток зі штрих-кодом. Коли водій хоче виїхати з території паркування, він повинен підійти до паркувального терміналу, розташованого у спеціально відведеному місці, та скористатися своїм вхідним квитком.

Термінал оснащений інформаційною панеллю, яка інформує відвідувача про суму платежу, за яким тарифом він був сформований та всю необхідну інформацію. Термінал також оснащений усім необхідним для оплати купюрами,

монетами, банківськими картками, і залежно від конфігурації функції терміналу можуть бути розширені.

Після оплати водій отримує квитанцію та виїзний квиток, яким він може залишити паркування на виїзній стійці. Система автоматично реєструє всі входи, виходи та генерує фінансові звіти.

2.5 Аналіз та вибір технологій і методів реалізації веб-ресурсу

Аутентифікація на основі cookies (cookie-based authentication) є популярним методом ідентифікації користувача. Проте, все більше уваги приділяється аутентифікації на основі токенів (token-based authentication), особливо для односторінкових додатків (Single Page Application).

Під час процесу аутентифікації користувача, йому надається токен після введення логіна (ім'я користувача) та пароля. Цей токен містить інформацію, необхідну для аутентифікації та авторизації користувача. Токен зберігається у вигляді cookie, яке додається до кожного запиту користувача. Генерація та перевірка цього cookie виконується за допомогою Middleware автентифікації на основі cookies (Cookie Authentication Middleware), яке вбудоване у потік обробки запитів та відповідей додатку. Middleware серіалізує дані користувача в зашифроване cookie. Під час наступних запитів, Middleware перевіряє валідність cookie, створює нові дані про користувача та присвоює їх властивості User контексту HTTP (HttpContext) [2].

Після успішної аутентифікації користувачу виділяється токен, який неможливо підробити (antiforgery token). Цей токен містить інформацію про користувача у вигляді оголошених значень (claims), таких як ім'я, електронна адреса, вік тощо. Також можуть використовуватись токени посилань (reference token), які вказують на стан користувача та підтримуються додатком.

Коли користувач намагається отримати доступ до ресурсу, який вимагає аутентифікації, додаток відправляє токен з додатковим заголовком

автентифікації - Bearer token. Це зроблено для того, щоб додаток не залежав від стану. Токен передається при кожному наступному запиті для перевірки на серверній стороні. Він не є зашифрованим, але є закодованим. На сервері токен розкодовується для отримання доступу. Для надсилання токена при подальших запитах він зберігається в локальному сховищі браузера.

Для написання програми було обрано мову програмування C#. На даний момент вона є однією з найпопулярніших мов програмування, яка використовується для розробки різних видів програм, такі як мобільні додатки, ігри, корпоративне програмне забезпечення та ін. Це мова загального застосування, яка призначена для розробки програмних продуктів на платформі Microsoft, яка працює на основі .NET-фреймворка під Windows [2].

.NET – це безкоштовне, багатоплатформне та відкрите середовище для розробників, яка дозволяє створювати додатки різних типів. З допомогою .NET можна використовувати декілька мов програмування (серед яких C#, Visual Basic, F# та ін.), редакторів і бібліотек, які спрощують процес написання програм. Незалежно від того, чи працює розробник на C #, F # або Visual Basic, його код буде працювати на будь-якій сумісній операційній системі [2].

Web API представляє спосіб побудови додатку ASP.NET, який спеціально розроблений для роботи в стилі REST (Representation State Transfer або передача стану представлення). REST-архітектура передбачає застосування наступних методів або типів запитів HTTP для взаємодії з сервером: GET, POST, PUT, DELETE. Створення проекту ASP.NET Core WebAPI в середовищі Visual Studio включає такі ж самі етапи, що і MVC, за винятком того, що обраний підтип програми буде обраний як Web API [2].

Dependency injection або впровадження залежностей – механізм, що дозволяє забезпечити слабку зв’язаність компонентів сервісу. Такі компоненти зв’язані між собою за допомогою абстракцій, наприклад через інтерфейси або абстрактні класи. При цьому відповідальність за створення відповідних залежностей покладається на зовнішній, спеціально призначений для цього, загальний механізм вбудовування. Часто, в ролі таких механізмів виступають

IoC-контейнери. Такі контейнери служать своєрідними фабриками, які встановлюють залежність між абстракціями та конкретними реалізаціями і, як правило, керують створенням цих об’єктів з попередньо зазначеними термінами виконання. Такий підхід дозволяє збільшити гнучкість системи, полегшити підтримку, розширення та заміну компонентів системи [2].

У попередніх версіях ASP.NET, для використання механізму інверсії керування (IoC) необхідно було використовувати зовнішні контейнери IoC, які надавалися сторонніми розробниками, такими як Ninject, Unity, Autofac і т.д. Однак, починаючи з версії ASP.NET 5, у фреймворку є вбудований контейнер IoC, який представлений інтерфейсом IServiceProvider. Конфігурування цього контейнера виконується в класі Startup.

Entity Framework Core - це спеціалізована об'єктно-орієнтована технологія, заснована на .NET Core, для роботи з даними і ORM-фреймворком для технології ADO.NET. З відмінності від традиційних технологій роботи з базами даних, Entity Framework надає більш високий рівень абстракції, що дозволяє працювати з даними незалежно від типу сховища, що значно підвищує можливості повторного використання коду [2].

На фізичному рівні ми працюємо з таблицями, первинними та зовнішніми ключами, індексами і т.д., але на абстрактному рівні, який надає Entity Framework, ми працюємо вже з об'єктами, що відображають модель даних, і маємо можливість чітко визначити зв'язки між цими сутностями.

Одне з центральних понять в концепції Entity Framework - це поняття "сутності". Сутність представляє набір даних, пов'язаних з певним об'єктом. Тому цей інструмент дозволяє працювати не з таблицями та рядками, а з об'єктами. Кожна сутність має один або кілька параметрів (атрибутів), що відрізняють її від інших і унікально її визначають. Ці параметри називаються первинними ключами [2].

Сутності можуть бути пов'язані асоціативними зв'язками, такими як "один до багатьох", "один до одного" і "багато до багатьох", що імітують властивості реальних баз даних та їх зв'язків за допомогою зовнішніх ключів.

Однією з переваг Entity Framework є використання запитів LINQ для вибірки даних. За допомогою LINQ ми можемо отримувати не тільки об'єкти сутностей з бази даних, але й об'єкти, пов'язані різними асоціативними зв'язками. Іншим важливим поняттям є модель EDM, яка відповідає класам сутностей і реальним таблицям у базі даних.

Entity Framework пропонує три можливих підходи для взаємодії з базою даних:

– Database First: Entity Framework автоматично генерує набір класів, які відображають модель даних наявної бази даних.

– Model First: Розробник створює модель даних, на основі якої Entity Framework генерує реальну базу даних на сервері.

– Code First: Розробник створює класи моделі даних, які будуть зберігатися в базі даних. Після цього Entity Framework, на основі створених класів моделі, генерує базу даних з відповідними таблицями.

Angular - це відкритий фреймворк, розроблений Google, призначений для розробки односторінкових додатків (Single Page Application). Основна мета Angular - розширення браузерних додатків на основі шаблону модель-представлення-контролер (MVC), а також спрощення їх тестування та розробки. Його основні переваги полягають в простоті написання коду, масштабованості та підтримці роботи з Rest API [2].

Додаток Angular побудований у вигляді дерева компонентів. Ідея використання незалежних функціональних блоків HTML з власними стилями подібна до концепції веб-компонентів (web-components). Розмітка компонентів Angular розміщується в Shadow DOM - внутрішньому DOM кожного складного елемента, що відображається як єдине ціле. Компоненти представляють собою класи стандарту мови JavaScript EcmaScript 6 з анотаціями. Для їх використання не потрібен спеціальний синтаксис, як, наприклад, у версії 1.x для директив.

2.6 Висновки до 2-го розділу

Отже, у другому розділі було здійснено проектування основних модулів програмної системи, а саме проектування архітектури, логічної схеми бази даних, інтерфейсу. Було встановлено, що буде використовуватись трьохрівнева архітектура, що є широко використовуваним шаблоном проектування для бізнес-застосунків, який забезпечує ефективне розподілення відповідальностей між різними рівнями системи без надмірних ускладнень. Як уже зазначалось, вона не обмежується конкретною реалізацією користувацького інтерфейсу.

Для більш детального та якіснішого розуміння логіки бази даних було спроектовано ER-діаграму. Діаграма «сутність – зв’язок» відображає взаємозв'язки між різними сутностями (наприклад, клієнтами та товарами) в базі даних та складається з трьох ключових елементів, які відображають загальну графічну структуру.

Також після детального аналізу інструментарію для розробки було встановлено, що таким буде Entity Framework Core - спеціалізована об'єктно-орієнтована технологія, заснована на .NET Core, для роботи з даними і ORM-фреймворком для технології ADO.NET. На відміну від традиційних технологій роботи з базами даних, Entity Framework надає більш високий рівень абстракції, що дозволяє працювати з даними незалежно від типу сховища, що значно підвищує можливості повторного використання коду.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Реалізація модулів програмної системи

На основі раніше визначених сценаріїв користувача, а також функціональних вимог була розроблена User flow діаграма, що наочно демонструє можливості користувача в системі та послідовність виклику сторінок представлення. Згідно діаграми, представленої на рисунку 3.1, були розроблені детальні макети програмної системи, що забезпечують користувача всіма необхідними функціями для керування паркінгом [2].

User flow діаграма – це візуальне представлення послідовності дій, які користувач виконує для досягнення своєї цілі, що може охоплювати лише певну функцію або цілком весь продукт. Вона допомагає створювати якісний користувацький досвід (user experience) для кінцевого користувача та більш ефективно задовольняти його потреби [2].

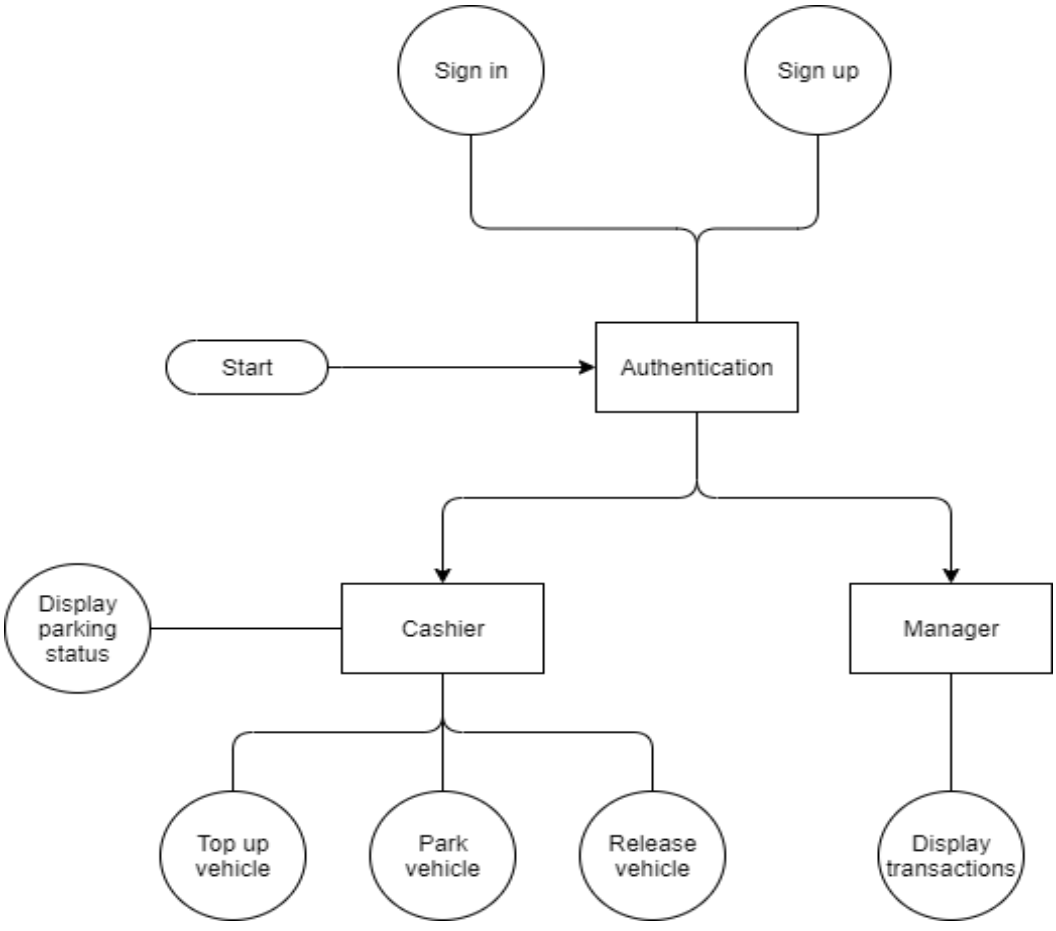


Рисунок 3.1 - User flow діаграма

Проект включає шість модулів. Завдяки архітектурному рішенню програми, яке було описано у попередніх розділах, визначення модулів збігається з визначенням класів. Це означає, що ключові компоненти проекту виконують роль раніше розроблених бібліотек та проектів.

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
            .AddJwtBearer(options => {
                options.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuer = true,
                    ValidateAudience = true,
                    ValidateLifetime = true,
                    ValidIssuer = Configuration["Jwt:Issuer"],
                    ValidAudience = Configuration["Jwt:Audience"],
                    IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
                };
            });

        services.AddCors(options =>
            options.AddDefaultPolicy(builder => builder.AllowAnyOrigin()));

        services.AddParkingContext(Configuration);
        services.RegisterCustomServices(Configuration);
        services.AddAutoMapper();

        services.AddMvc();
        services.AddControllers();
    }
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseCors(builder => builder
            .WithOrigins(Configuration["Cors:Origin"])
            .AllowAnyMethod()
            .AllowAnyHeader());

        app.UseAuthentication();
        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
```

Кожен з цих модулів містить набір властивостей та методів, необхідних для вирішення конкретних завдань. Таким чином, кожен модуль є самостійним об'єктом у загальній структурі рішення. На основі встановлених завдань було розроблено функціонал модулів. Нижче наведені приклади методів, які здійснюють обмежені операції у вузько спрямованому контексті [2].

У класі Startup проекту WebAPI виконується конфігурація контролерів, аутентифікації, CORS, контексту бази даних та реєстрація сервісів, розміщених в бібліотеці бізнес-логіки [2].

```
[Route("api/[controller]")]
[ApiController]
public class AuthController : ControllerBase
{
    private readonly IAuthService authService;

    public AuthController(IAuthService authService)
    {
        this.authService = authService;
    }

    [AllowAnonymous]
    [HttpPost("log-in")]
    public async Task<ActionResult<string>> LogIn(UserLogInDto userLogin)
    {
        var token = await authService.ObtainJwTokenAsync(userLogin);
        return string.IsNullOrEmpty(token) ? NotFound("User not found") : Ok(new { token });
    }
}

public interface IAuthService
{
    Task<string> ObtainJwTokenAsync(UserLogInDto userLogin);
}

public class AuthService : IAuthService
{
    private readonly ParkingContext context;
    private readonly IConfiguration configuration;
    private readonly IMapper mapper;
    public AuthService(ParkingContext context, IConfiguration configuration, IMapper mapper)
    {
        this.context = context;
        this.configuration = configuration;
        this.mapper = mapper;
    }

    public async Task<string> ObtainJwTokenAsync(UserLogInDto userLogin)
    {
        var user = await Authenticate(userLogin);
        return user == default ? string.Empty : Generate(user);
    }
}
```

Кожен контролер WebAPI отримує через конструктор класу екземпляр відповідного сервісу за допомогою IoC-контейнера, що відповідає визначеному інтерфейсу. Такий підхід реалізації за допомогою абстракцій дозволяє гнучко розробляти систему знижуючи зв’язаність об’єктів між собою.

Комунікація між сервером та клієнтом базується на використанні об'єктів передачі даних (DTO). DTO (Data Transfer Object) - це об'єкт, що використовується для упаковки та передачі даних з однієї підсистеми до іншої. Кожен DTO зберігається в загальній бібліотеці, до якої мають доступ всі проекти. Це дозволяє обмінюватися лише необхідною інформацією, що зменшує навантаження на трафік сервера та час, потрібний для обробки даних. Опис DTO має наступний формат [2]:

```
public class UserProfileDto
{
    public string Username { get; set; }
    public string Password { get; set; }
    public string Role { get; set; }
    public string Surname { get; set; }
    public string GivenName { get; set; }
}
```

Для перетворення об'єкту передачі даних на серверну модель використовується функціонал бібліотеки AutoMapper. Ця бібліотека дозволяє гнучко здійснювати перетворення екземплярів класів, шляхом конфігурації їхніх властивостей в класах, що успадковуються від класу Profile. Завдяки цьому підходу можна швидко перетворювати об'єкти на необхідні типи, викликаючи всього лише один метод [2].

```
public class UserProfileMapProfile : Profile
{
    public UserProfileMapProfile()
    {
        CreateMap<UserProfile, UserProfileDto>()
            .ForMember(dest => dest.Role, opt => opt.MapFrom(scr => scr.Role.Title))
            .ForMember(dest => dest.Surname, opt => opt.MapFrom(scr =>
scr.Employee.LastName))
            .ForMember(dest => dest.GivenName, opt => opt.MapFrom(scr =>
scr.Employee.FirstName));
    }
}

private async Task<UserProfileDto> Authenticate(UserLogInDto userLogin)
{
    var currentUser = await context.UserProfiles.SingleOrDefaultAsync(x => /* ... */);
    return mapper.Map<UserProfileDto>(currentUser ?? default);
}
```

Робота з базою даних MS SQL Server, виконувалась за допомогою Entity Framework Core. Було створено контекст бази даних на основі попередньо визначеної ER-діаграми, а також були налаштовані зв'язки між сутностями за допомогою Fluent API. Для наповнення бази даних демонстраційними даними (seeding) використовувалась бібліотека Bogus.

```
public class ParkingContext : DbContext
{
    public ParkingContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<Employee> Employees { get; set; }
    public DbSet<Position> Positions { get; set; }
    public DbSet<Role> Roles { get; set; }
    public DbSet<Service> Services { get; set; }
    public DbSet<Tariff> Tariffs { get; set; }
    public DbSet<UserProfile> UserProfiles { get; set; }
    public DbSet<Vehicle> Vehicles { get; set; }
    public DbSet<VehicleType> VehicleTypes { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Configure();
        modelBuilder.Seed();
    }
}

public class UserProfileConfiguration : IEntityTypeConfiguration<UserProfile>
{
    public void Configure(EntityTypeBuilder<UserProfile> builder)
    {
        builder.HasOne(x => x.Employee)
                .WithMany(x => x.UserProfiles)
                .HasForeignKey(x => x.EmployeeId);

        builder.HasOne(x => x.Role)
                .WithMany(x => x.UserProfiles)
                .HasForeignKey(x => x.RoleId);
    }
}

private static ICollection<Employee> Employees => GenerateEmployees();

public static void Seed(this ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Employee>().HasData(Employees);
}

private static List<Employee> GenerateEmployees()
{
    return new Faker<Employee>()
        .UseSeed(2937)
        .RuleFor(x => x.Id, f => ++f.IndexVariable)
        .RuleFor(x => x.FirstName, f => f.Name.FirstName())
        .RuleFor(x => x.LastName, f => f.Name.LastName())
        .RuleFor(x => x.HiringDay, f => f.Date.Between(new DateTime(2021, 1, 1), new
DateTime(2021, 6, 30)))
        .Generate(count: 5);
}
```

Code First є одним з підходів до створення бази даних за допомогою Entity Framework Core, де база даних створюється на основі моделей класів. Замість вручну визначеного схеми бази даних або використання вже існуючої бази даних, Code First дозволяє розробникам спрощено визначити структуру бази даних без прямого втручання в SQL скрипти або діаграми бази даних.

Один з ключових елементів Code First - використання Fluent API, який дозволяє налаштовувати деталі створення бази даних через розширені можливості конфігурації. Fluent API надає розробникам зручний і гнучкий спосіб визначення правил та налаштувань бази даних без залежності від атрибутів або конвенцій.

За допомогою Fluent API, розробники можуть визначати різні аспекти бази даних, такі як первинні ключі, зовнішні ключі, обмеження, типи даних і багато іншого. Це дозволяє зберігати моделі домену (наприклад, класи Vehicle або Employee) незалежними від специфіки бази даних, що полегшує розробку та підтримку коду.

Code First підхід з використанням Fluent API дозволяє розробникам використовувати мову програмування C# для визначення структури бази даних, зберігаючи її в синхронізованому стані з моделями домену. При виконанні міграцій, Entity Framework Core використовує цю інформацію для створення або оновлення бази даних відповідно до описаної моделі.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Vehicle>(entity =>
    {
        entity.HasKey(e => e.Id);

        entity.Property(e => e.LicensePlate).IsRequired();

        entity.Property(e => e.Balance).HasColumnType("decimal(18,
2)").HasDefaultValue(0);

        entity.HasOne(e => e.VehicleType)
            .WithMany()
            .HasForeignKey(e => e.VehicleTypeId);
    });
}
```

У вказаному прикладі використовується Fluent API для визначення моделі Vehicle в контексті бази даних з використанням Entity Framework Core. В методі OnModelCreating визначається конфігурація цієї сутності.

Спочатку використовується метод HasKey, який вказує, що властивість Id є первинним ключем сутності Vehicle. Далі за допомогою методу Property встановлюється конфігурація для властивостей LicensePlate та Balance. Метод IsRequired використовується для позначення поля LicensePlate як обов'язкового.

Для властивості Balance використовується метод HasColumnType, щоб вказати тип колонки в базі даних (у даному випадку – decimal (18, 2)). Також за допомогою методу HasDefaultValue встановлюється значення за замовчуванням для властивості Balance, яке становить 0.

Для встановлення зв'язку між Vehicle та VehicleTyp, використовується метод HasOne. Він вказує, що властивість VehicleType є навігаційною властивістю до зв'язаного об'єкту VehicleType. За допомогою методу WithMany позначається відсутність навігаційної властивості у сутності VehicleType. Метод HasForeignKey вказує, що зовнішній ключ VehicleTypeId використовується для зв'язку з VehicleType.

Цей підхід дозволяє гнучко налаштовувати модель Vehicl` за допомогою Fluent API, замість атрибутів або конвенцій за замовчуванням.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Employee>(entity =>
    {
        entity.HasKey(e => e.Id);

        entity.Property(e => e.FirstName).IsRequired();

        entity.Property(e => e.LastName).IsRequired();

        entity.Property(e => e.HiringDay).IsRequired();
    });
}
```

У даному прикладі створюється модель Employee з використанням Entity Framework Core. Ця модель має такі атрибути: Id, FirstName, LastName та HiringDay.

Клас Employee включає в себе властивості для зберігання інформації про співробітника. Властивість Id використовується як первинний ключ для сутності Employee. Атрибути FirstName, LastName та HiringDay використовуються для зберігання імені, прізвища та дати прийому на роботу співробітника відповідно.

Для визначення конфігурації моделі Employee використовується Fluent API в методі OnModelCreating. Виклик методу HasKey вказує на те, що властивість Id є первинним ключем для цієї сутності. Методи IsRequired використовуються для позначення обов'язковості полів FirstName, LastName та HiringDay, тобто вони не можуть мати значення null у базі даних.

Цей підхід дозволяє гнучко визначати модель Employee з використанням Fluent API, додаючи атрибути та налаштовуючи обмеження для кожної властивості.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Service>(entity =>
    {
        entity.HasKey(e => e.Id);

        entity.Property(e => e.RegistrationDateTime).IsRequired();

        entity.HasOne(e => e.Vehicle)
            .WithMany()
            .HasForeignKey(e => e.VehicleId);

        entity.HasOne(e => e.Employee)
            .WithMany()
            .HasForeignKey(e => e.EmployeeId);
    });
}
```

Наведений код демонструє модель Service, яка об'єднує сутності Employee та Vehicle за допомогою зовнішніх ключів VehicleId та EmployeeId. Модель також має атрибут RegistrationDateTime та первинний ключ `Id`.

Клас Service включає в себе властивості для зберігання інформації про сервісне обслуговування. Властивість Id використовується як первинний ключ для сутності Service. Атрибут RegistrationDateTime використовується для зберігання дати та часу реєстрації сервісного обслуговування.

Зв'язок з Employee встановлюється за допомогою властивості EmployeeId та зовнішнього ключа EmployeeId, який посилається на первинний ключ Id сутності Employee. Аналогічно, зв'язок з Vehicle встановлюється за допомогою властивості VehicleId та зовнішнього ключа VehicleId, який посилається на первинний ключ Id сутності Vehicle.

У методі OnModelCreating використовується Fluent API для визначення конфігурації моделі Service. Виклик методу HasKey позначає властивість `Id` як первинний ключ для сутності Service. Метод `IsRequired` використовується для позначення поля RegistrationDateTime як обов'язкового.

Метод HasOne використовується для встановлення зв'язку між Service та Vehicle або Employee. Метод WithMany вказує, що в сутності Vehicle або Employee відсутня навігаційна властивість для зв'язку з Service. Метод HasForeignKey використовується для вказівки зовнішнього ключа VehicleId або EmployeeId для зв'язку з відповідними сутностями.

3.2 Реалізація інтерфейсу програмної системи

На етапі формування моделі структури інтерфейсу програмної системи було розглянуто модель структури інтерфейсу для окремої дії. Кожна окрема дія – це акт взаємодії користувача з системою з метою здійснення за допомогою її певної професійної дії. Склад елементів інтерфейсу, згідно з наведеною класифікацією, повинен відповідати набору даних, які будуть оброблені в рамках цього окремого дії. Крім того, в моделі повинні бути присутні елементи,

що відбивають логіку виконання діяльності користувачем. Відмінність такої моделі інтерфейсу від існуючої моделі віконного інтерфейсу полягає в тому, що наповнення форм елементами, організація взаємозв'язку елементів та самих форм буде підпорядкована логіці виконання професійних дій користувачем, на відміну інших аспектів організації наповнення та взаємозв'язку, таких, як логіка роботи додатки, логіка обробки даних, визначена розробником, та інше.

Говорячи загалом про процес виконання професійної дії, потрібно помітити, що користувач повинен пам'ятати весь процес, включаючи послідовність виконання кроків, набір даних, які необхідно підготувати та обробити, результат, який має вийти. У ході автоматизації частину цих маніпуляцій було покладено на систему, яка найчастіше успішно справляється з обробкою даних та їх перетворенням у результат, при цьому процес, що виконується користувачем видозмінився. Користувач, як і раніше, повинен виконувати весь процес цілком, проте оскільки частково дія стала виконуватися системою, загальна картина може бути спотворена користувача, оскільки може не знати, як і в якій послідовності системою ведеться обробка даних, необхідні для певного кроку, особливо при першому знайомство із системою. У зв'язку з цим пропонується загальний процес виконання дії, який залишився лише «в голові» користувача концептуально перенести в систему та відобразити в інтерфейсі, що дасть користувачеві цілісну картину його професійної діяльності та допоможе краще орієнтуватися у системі.

В даний час наявність декількох модальностей у користувальницькому інтерфейсі дедалі більше впроваджується у системи для забезпечення ефективної комунікації між людиною та комп'ютером і для забезпечення адаптивності до умов, що змінюються. Ефективність комунікації досягається за рахунок того, що інформація може бути передана за допомогою кількох каналів комунікації в залежності від побажання користувача та умов виконання завдання. При використанні такої полімодальної взаємодії людини з автоматизованою системою користувач може оперативніше виконувати завдання, адаптуватися до ведення діалогу з системою та керувати нею. Крім

того, для ефективного взаємодії користувача із системою необхідно забезпечити можливість відображення в інтерфейсі не тільки необхідних для роботи даних, а також логіки професійної діяльності, що супроводжується інформаційною системою.

У процесі людино-машинної взаємодії можна виділити два типи взаємодії:

- ведення діалогу та управління системою;

- введення даних та маніпулювання. При цьому в кожному типі взаємодії можуть бути використані різні модальності, що є способами передачі інформації від людини до системи та отримання інформації від системи до людини. Всі елементи інтерфейсу користувача можуть бути систематизовані, як відносяться до однієї з наступних чотирьох категорій:

1. Елементи введення інформації (Input Controls) - дозволяють користувачам вводити інформацію в систему.

2. Елементи здійснення навігації (Navigation Components) – допомагають користувачам переміщатися у рамках інтерфейсу.

3. Інформаційні елементи (Informational Components) – призначені для показу інформації користувачам.

4. Елементи-контейнери (Containers) – призначені для угруповання пов'язаного за змістом контенту (взаємопов'язаних елементів інтерфейсу інших категорій). Оскільки проводиться дослідження побудови полімодальних інтерфейсів, що забезпечують комунікацію з користувачем кількома каналами сприйняття інформації, необхідно виділити канали комунікації, які будуть використані в інформаційній системі. Було виділено чотири канали комунікації – текстовий, мовний, акустичний та візуальний. Кожному каналу комунікації відповідають певні засоби передачі інформації, яким у кожному типі дій користувача ставиться у відповідність тип дії, що здійснюється елементами інтерфейсу, оскільки саме елементи інтерфейсу дозволяють користувачам робити ті чи інші дії в системі. І, нарешті, залежно від типу дії відбувається перехід безпосередньо до типових елементів інтерфейсу. Деякі типові елементи інтерфейсу можуть відповідати кільком каналам комунікації та кільком

модальностям, тому можна бачити дублювання елементів. Аналогічна ситуація спостерігається і з типами дій користувача та типами дій елементів інтерфейсу.

Для того, щоб програмною системою було зручно користуватись необхідно, щоб усі елементи користувацького інтерфейсу взаємодіяли між собою та працювали злагоджено.

На рисунку 3.2 зображено сторінку, де користувач може ввести свої облікові дані, щоб отримати доступ до захищених ресурсів сайту. Для входу необхідно ввести ім'я користувача в системі (Username) та пароль. Якщо введено некоректний пароль, система повідомить про відповідну помилку. Якщо користувач ще не зареєстрований, він може самостійно приєднатися до сайту, але з обмеженими можливостями, або звернутися до адміністратора системи, який має доступ до повного функціоналу для створення нового облікового запису користувача. Крім того, користувач може перейти на домашню сторінку програмної системи.

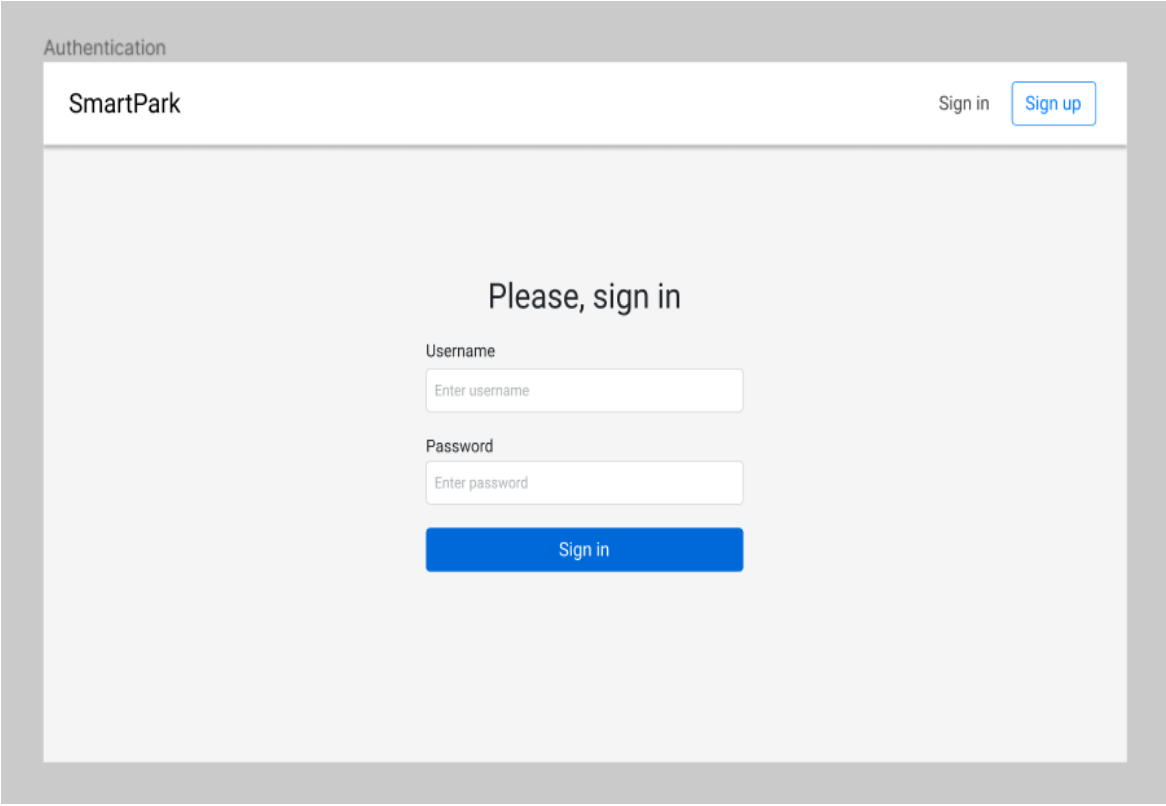


Рисунок **Помилка! У документі відсутній текст указаного стилю.**.1 – Макет сторінки аутентифікації користувача

На рисунку 3.3 зображено сторінку касира за допомогою якої він може надавати послуги паркування, а саме: додання транспортного засобу, що прибув, поповнення його балансу та відпуск. Касир також може виконувати моніторинг стану програмної системи.

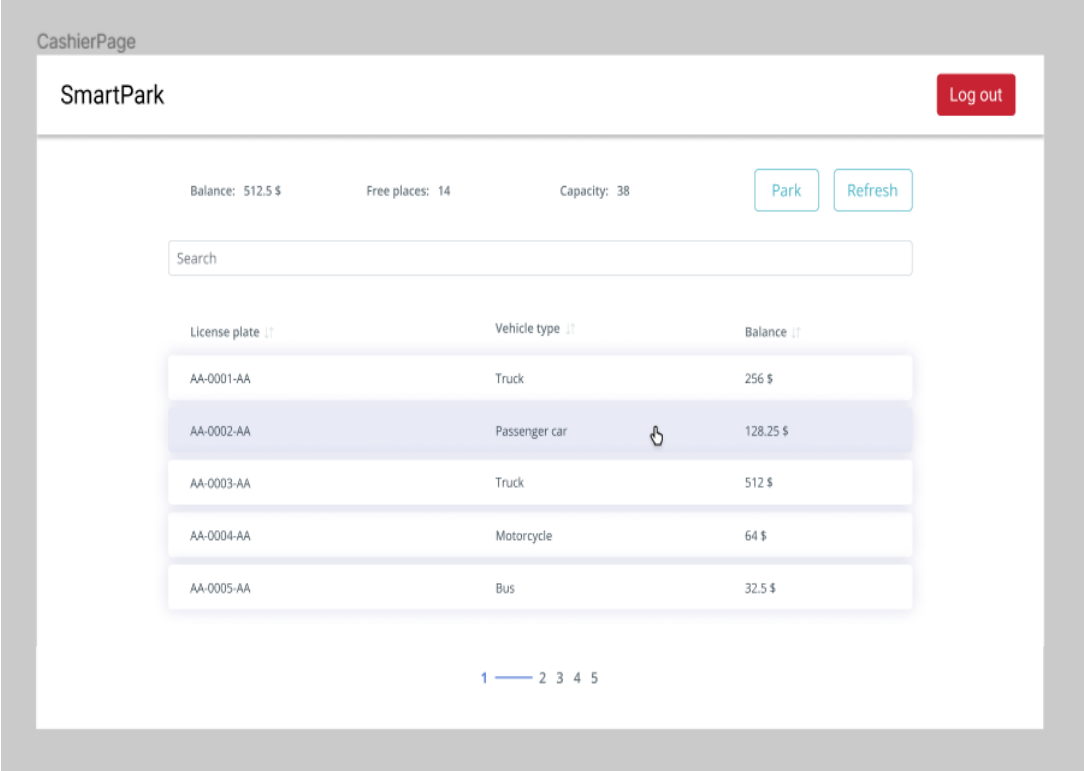


Рисунок **Помилка! У документі відсутній текст указаного стилю.**.2 – Макет сторінки касира

На рисунку 3.4. продемонстровано вигляд сторінки касира при виконання певної дії.

Як видно даний інтерфейс передбачає ідентифікацію за певним типом. Також передбачено пошук за типом транспортного засобу, номерними знаками, способом та типом паркування, датою та часом паркування.

При введені пошукового запиту на екран виводиться діалогове вікно, що передбачає пошук за ключовими позиціями, такими як номерний знак, марка транспортного засобу, час паркування, дата паркування, період, протягом якого

транспортний засіб перебував на стоянці, сума, що була оплачена за користування платним паркомісцем.

Даний функціонал є досить простим та зручним і не потребує якогось спеціального навчання.

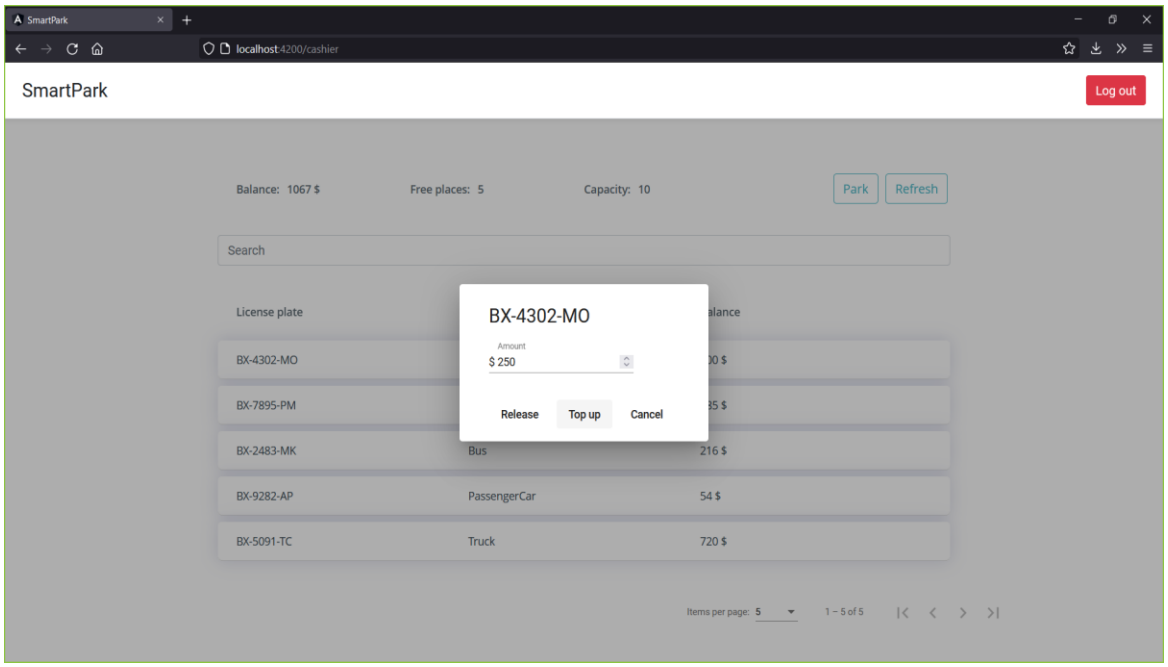


Рисунок 3.4 – Вигляд сторінки касира при виконанні дії

На рисунку 3.5 зображено вигляд сторінки адміністратора програмної системи. Даний користувач може здійснювати створення ролей, редагувати та вносити зміни. Також є продуманий функціонал для відображення звітів за певний період за марками транспортних засобів, номерними знаками, сумами та періодами паркування.

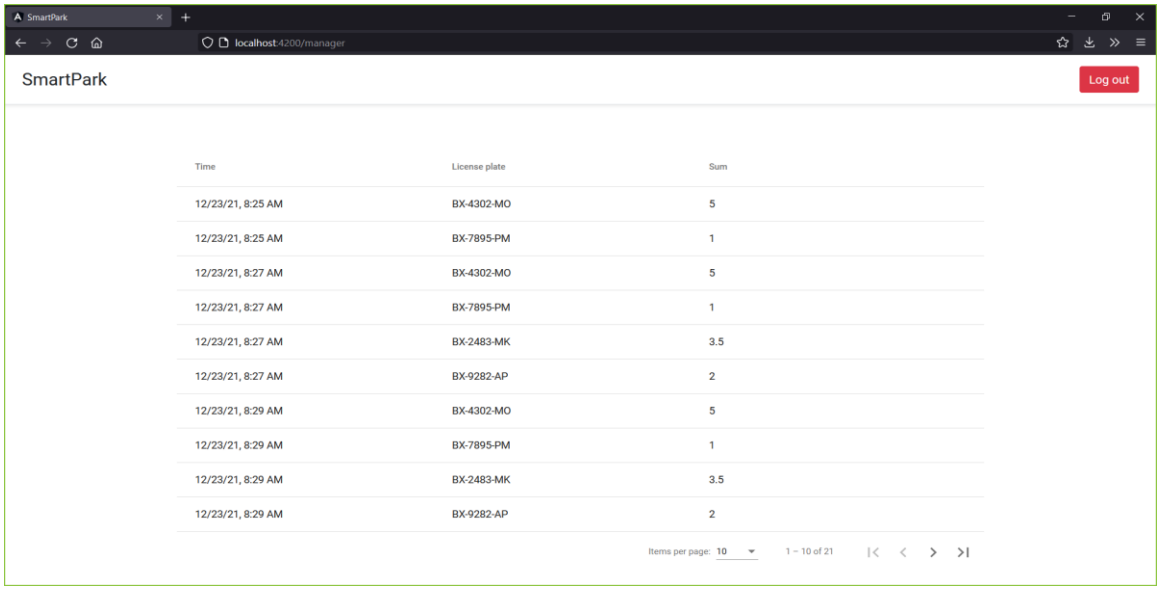


Рисунок 3.5 – Вигляд сторінки адміністратора

3.3 Тестування та аналіз програмної системи

Загалом існує дуже багато видів, типів та підтипів тестування програмного забезпечення. Класифікація їх досить широка та багатокомпонентна.

Модульне тестування використовується для тестування будь-якого одного логічно виділеного та ізольованого елемента системи (окремі методи класу або проста функція, subprograms, subroutines, класи чи процедури) у коді. Очевидно, що це тестування методом білої скриньки і найчастіше воно проводиться самими розробниками.

Метою тестування модуля є не демонстрація правильного функціонування модуля, а демонстрація наявності помилки у модулі, а також у визначенні ступеня готовності системи до переходу на наступний рівень розробки та тестування. На рівні модульного тестування найпростіше виявити дефекти, пов'язані з алгоритмічними помилками та помилками кодування алгоритмів, типу роботи з умовами та лічильниками циклів, а також з використанням локальних змінних та ресурсів. Помилки, пов'язані з неправильним трактуванням даних, некоректною реалізацією інтерфейсів, сумісністю, продуктивністю тощо. зазвичай пропускаються лише на рівні модульного

тестування і виявляються більш пізніх стадіях тестування. Ізоляція тестованого блоку досягається за допомогою заглушок (stubs), манекенів (dummies) та макетів (mockups).

Компонентне тестування - тип тестування, при якому тестування виконується для кожного окремого компонента окремо, без інтеграції з іншими компонентами. Його також називають модульним тестуванням (Module testing), якщо його з погляду архітектури. Як правило, будь-яке програмне забезпечення загалом складається з кількох компонентів. Тестування на рівні компонентів (Component Level testing) має справу із тестуванням цих компонентів індивідуально. Це один із найчастіших типів тестування чорної скриньки, який проводиться командою QA. Для кожного з цих компонентів буде визначено сценарій тестування, який потім буде наведено до Test case високого рівня -> детальним Test case низького рівня з попередніми умовами.

Виходячи з глибини рівнів тестування, компонентне тестування можна класифікувати як:

Тестування компонентів у малому (CTIS - Component testing In Small): тестування компонентів може проводитися з або без ізоляції інших компонентів у програмному забезпеченні або додатку. Якщо це виконується з ізоляцією іншого компонента, це називається CTIS;

Тестування компонентів загалом (CTIL - Component testing In Large) - тестування компонентів, виконане без ізоляції інших компонентів у програмному забезпеченні або додатку;

Для тестування розроблюваного програмного продукту використання модульне тестування. Модульне тестування, також відоме як юніт-тестування (unit testing), є процесом у програмуванні, що дозволяє перевірити поведінку та результат окремих модулів у вихідному коді програми. Основна ідея полягає у написанні тестів для кожного складного методу. Цей підхід дозволяє швидко перевірити, чи призвела остання зміна коду до регресії, тобто до появи помилок у вже протестованих частинах програми, а також полегшує виявлення і виправлення цих помилок у виконанні визначеної логіки програми.

Модульне тестування складається з трьох етапів:

– ініціалізація невеликого фрагмента програми, яку потрібно протестувати;

– виклик методу, що застосовується як стимул до системи, що тестується;

– спостереження за поведінкою модуля, що перевіряється. Якщо поведінка, що спостерігається, відповідає очікуванням користувача, то модульний тест проходить. Цей покроковий процес також називається AAA (Arrange, Act, Assert).

Тест вважається по-справжньому модульним, якщо має наступні властивості:

Сфокусований - перевіряє лише одне твердження.

Цінний - відображає актуальну вимогу.

Незалежний - від інших тестів чи оточення, на якому виконується.

Швидкий - виконується досить швидко, щоб запускатися при кожній зміні коду.

Зрозумілий - дотримується структура, конвенція іменування, має невеликий обсяг.

Підтримуваний - змінюється з часом.

Модульний тест буває двох видів: з урахуванням стану, і навіть з урахуванням взаємодії. Якщо ви відстежуєте отриманий стан, це модульне тестування на основі стану.

Якщо тестування відбувається з використанням певних методів - це модульне тестування з урахуванням взаємодії. Користувачі часто плутають модульне та інтеграційне тестування. Тому досить важливим є розуміння даної різниці.

Метою модульного тестування є перевірка поведінки кожної частини програмного забезпечення незалежно від інших частин. Модульні тести мають більш вузьку сферу застосування, дозволяють охопити всі випадки та

гарантувати, що кожна окремо взята ділянка працює бездоганно. Таке тестування досить легко продати.

З іншого боку, інтеграційне тестування підтверджує, що різні частини системи нормально працюють спільно у реальному середовищі. Це високорівневе тестування, яке перевіряє складні сценарії. Зазвичай, потрібні зовнішні ресурси, такі як веб-сервери та бази даних.

Інтеграційне тестування застосовують при взаємодії між різними компонентами за умов максимально близьких до реального середовища (за допомогою додаткових інструментів).

Коли ці два види тестування об'єднуються, то отримується високий рівень впевненості в тому, що вся система працює належним чином. Однак завжди потрібно пам'ятати про те, який тест ми реалізуємо: модульний чи інтеграційний.

Трапляються випадки, коли модульні тести вимагають наявності зовнішніх ресурсів, таких як веб-сервери або база даних. Найчастіше причина криється в поганому дизайні модульного тесту.

Модульні тести застосовуються для перевірки різних аспектів програми, не витрачаючи багато часу та зусиль з боку розробників.

Такі тести працюють, навіть якщо в системі, що тестується, є помилка. На хороші модульні тести не впливатимуть зовнішні чинники, такі як довкілля чи порядок виконання. Якщо це станеться, то це безперечно недолік дизайну.

Якісний модульний тест зрозумілий та демонструє поведінковий аспект програми. За допомогою тестових прикладів легко зрозуміти, який сценарій був протестований. Якщо тест не проходить, виправити помилки можна швидко і без налагодження коду.

Модульні тести написані таким чином, що їх можна використовувати багаторазово.

Unit-тести і система, що тестується, не повинні звертатися до мережевих ресурсів, файлових систем і баз даних, щоб виключити загальний вплив

зовнішніх факторів, що робить такий тест дійсно модульним, а не інтеграційним. Вони прості у написанні, і їх легко підтримувати.

У платформі .NET Core підтримуються різні фреймворки для тестування. Проте xUnit виявився найбільш популярним завдяки своїй простоті, виразності та розширюваності. Цей проект підтримує .NET Foundation і є частиною останніх версій .NET Core. Це означає, що для використання xUnit не потрібно встановлювати додаткові інструменти, окрім .NET Core SDK.

У ході тестування бізнес-логіки проекту були написані модульні тести для всіх функцій, які надає сервіс паркування. Кожен з цих тестів був успішно виконаний, що дозволяє бути впевненим у правильному виконанні алгоритмів.

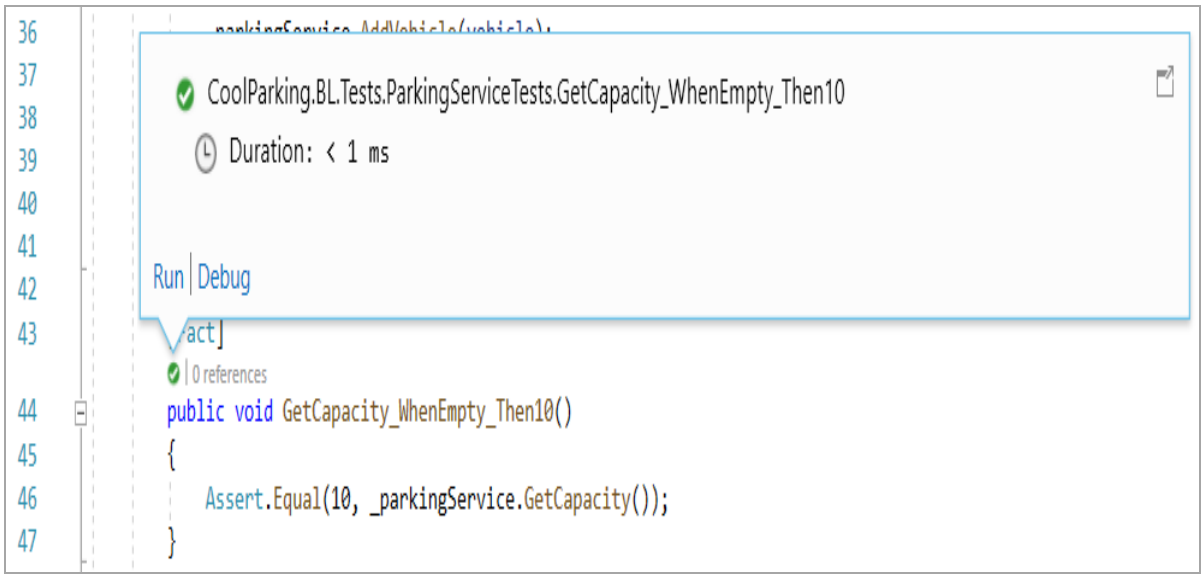


Рисунок 3.4 – Перевірка заповненості паркінгу

Рисунок 3.5 показує перевірку кількості створених екземплярів класу ParkingService, який працює за шаблоном Singleton, шляхом додавання створеного транспортного засобу до сховища з використанням однієї змінної того ж класу.

На рисунку 3.4 зображено тестування методу отримання ємності паркінгу, коли він не має запаркованих транспортних засобів. Цей тест перевіряє загальні налаштування паркінгу, зокрема, чи відповідає вказана у файлі конфігурації ємність тій, що реалізується сервісом.

Рисунок 3.5 – Перевірка властивості екземпляра класу ParkingService

Рисунок 3.6 демонструє юніт-тест, який перевіряє можливість поповнення балансу транспортного засобу, що не знаходиться на парковці. У разі такої ситуації сервіс повинен викинути помилку виконання, щоб про це було сповіщено.

Рисунок 3.6 – Здіснення перевірки поповнення балансу

На рисунку 3.7 показано юніт-тест, який перевіряє правильність виконання логіки списання коштів з балансу кожного запаркованого транспортного засобу та їх перенесення на загальний баланс паркінгу. Для цього було створено два екземпляри класу Vehicle з визначеними бюджетами та

різними типами транспортних засобів (кожен тип має свій тариф), і були спровоковані дві події, що сигналізують про необхідність списання коштів.

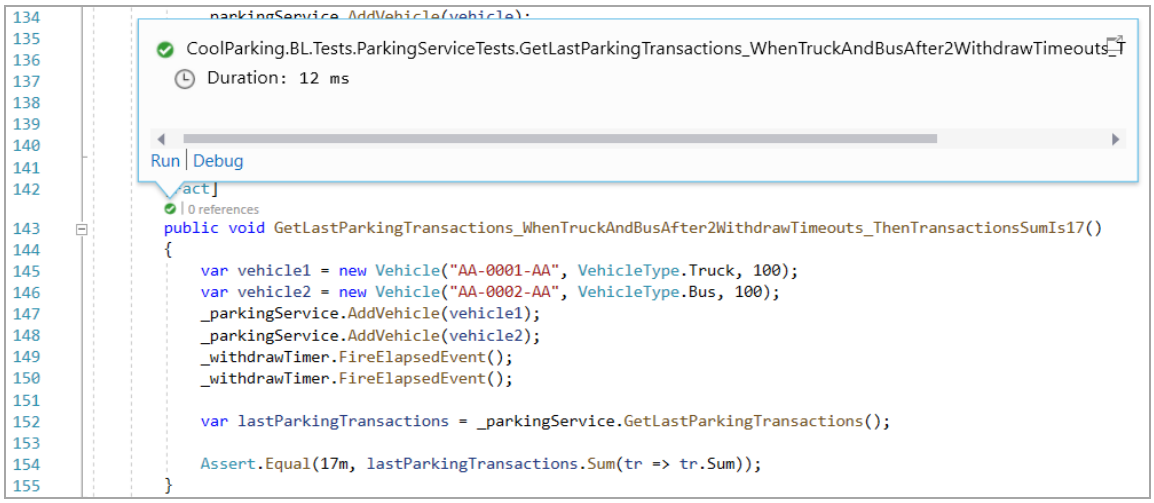


Рисунок 3.7 – Перевірка списання коштів

На рисунку 3.8 показано виконання перевірки методу додавання транспортного засобу на паркінг. Для цього був створений екземпляр класу Vehicle, який додається в систему за допомогою сервісу.

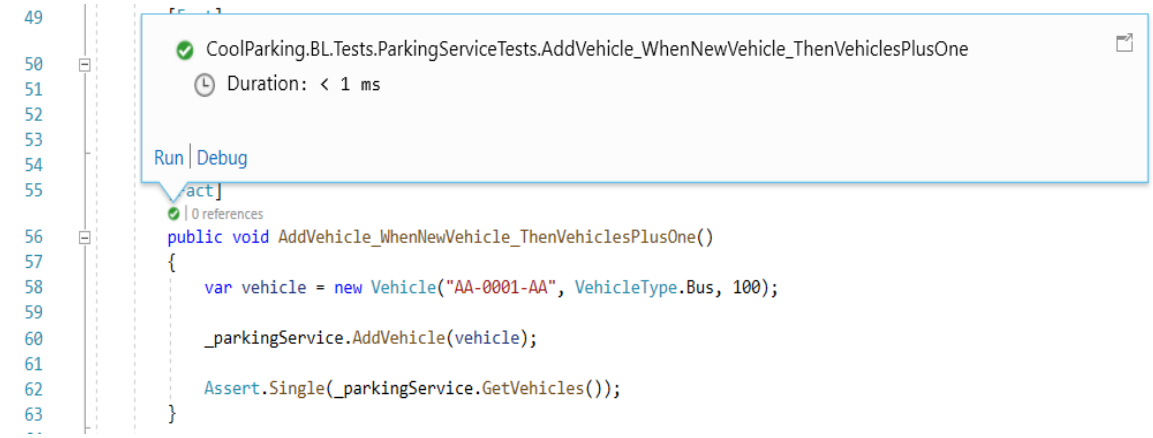


Рисунок 3.8 – Здійснення перевірки сервісу логування

Отже, модульне тестування є досить доцільним та показало, що система працює злагоджено та без перебоїв.

Висновки до 3-го розділу

У даному розділу було здійснено реалізацію програмної системи автоматизованого керування паркінгом транспортних засобів.

Загалом проект складається із шести модулів. Завдяки архітектурному рішенню програми, яке було описано у попередніх розділах, визначення модулів збігається з визначенням класів. Це означає, що ключові компоненти проекту виконують роль раніше розроблених бібліотек та проектів.

Також було розроблено та реалізовано інтерфейс програмної ситеми. Дана частина є досить простою у експлуатації та не потребує якогось додаткового навчання.

Під час етапу тестування було створено низку юніт-тестів, які повністю охопили функціонал, реалізований у сервісі паркування та логування.

ВИСНОВКИ

Отже, на основі проведеної роботи та здійсненого дослідження можна зробити такі висновки.

У першому розділі для вирішення завдань, було проведено детальне дослідження предметної області і використано такі технології: ASP.NET Core WebAPI, EntityFramework Core, MS SQL Server, Angular і JWT. Також подано опис наявних рішень, сформульоване технічне завдання проекту, визначені ключові проблеми та наведений перелік технологій, які були використані для досягнення поставлених завдань. Також було розглянуто можливості їх реалізації.

У другому розділі була описана трьохрівнева архітектура, яка була застосована в проекті. Були створені та обґрунтовані діаграми, що описують внутрішню структуру системи та рух даних, а також надано опис ролей користувачів. Крім того, було представлено огляд використовуваних технологій та інструментів розробки.

У третьому розділі була визначена структура програми, що базується на технології ASP.NET Core WebAPI. Були встановлені зв'язки між моделями та описано їх повний функціонал, який був розподілений по бібліотеках. Також були виділені певні фрагменти програмного коду, які потребували докладнішого пояснення їх реалізації. Були визначені мінімальні вимоги до технічних засобів.

Розробка програмної системи для паркінгу може бути доцільною з кількох причин:

Автоматизація процесів: програмна система дозволяє повністю автоматизувати процеси управління паркінгом, включаючи реєстрацію в'їзду і виїзду, розрахунок оплати, контроль доступу та інші аспекти. Це спрощує роботу персоналу та забезпечує ефективне функціонування паркінгу.

Зручність для користувачів: програмна система може надати зручні інструменти для користувачів, такі як онлайн-бронювання місць, безконтактна оплата, нагадування про термін паркування тощо. Це робить процес паркування зручнішим та швидшим для водіїв.

Оптимізація використання простору: програмна система може допомогти оптимізувати використання паркінгових місць, розподіляючи їх ефективно залежно від потреб і доступності. Це дозволяє забезпечити оптимальне використання простору і запобігти зайнятості місць без потреби.

Збільшення ефективності та контролю: програмна система дозволяє збільшити ефективність управління паркінгом шляхом автоматизації багатьох процесів, зменшення людського фактора та поліпшення контролю за станом паркінгу. Це може включати моніторинг за заповненістю, збір статистики, контроль доступу та інші функції.

Покращення безпеки: програмна система може сприяти покращенню безпеки паркінгу шляхом впровадження системи відеоспостереження, контролю доступу, автоматичного розпізнавання номерних знаків та інших технологій. Це забезпечує захист автомобілів та зменшує ризик виникнення конфліктів.

Враховуючи ці переваги, розробка програмної системи для паркінгу може бути доцільною для покращення ефективності, зручності та безпеки управління паркінгом.

Розроблений програмний продукт повністю автоматизує процес управління паркінгом, що було головною метою проекту. Він буде корисним в різних сферах підприємницької діяльності та дозволить вирішити раніше виявлені проблеми в організації роботи парковки.

У майбутньому, програмний продукт можна поліпшити шляхом розширення функціональності для відображення та виведення статистичних даних для фінансового обліку, удосконалення та розширення реалізації процесів паркування, додавання нових можливостей до графічного інтерфейсу користувача, оптимізації коду і поліпшення архітектури програми за допомогою шаблонів проектування та нових можливостей мови програмування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Інструкція з охорони праці для працівника офісу [Електронний ресурс]. URL: https://pro-op.com.ua/article/871-nstruktsya-z-ohoroni-prats-dlya-pratsvnika-ofsu (дата звернення: 12.02.2023).
2. Курсовий проект "Автоматизована система керування паркінгом" з дисципліни "Програмування Інтернет" студента Мариняка В.М. / керівник Яшина О.М. / ХНУ: 2021, 45 с.
3. Програмне забезпечення парковок [Електронний ресурс]. URL: https://www.sea.com.ua/ua/parkovochnoe_oborudovanie/programmnoe-obespecenie-parkovok/ (дата звернення: 19.02.2023).
4. How to test a website. Geteasyqa [Електронний ресурс]. URL: https://geteasyqa.com/qa/test-website/ (дата звернення: 17.04.2023).
5. Наскрізна практична підготовка: програма та методичні вказівки щодо її організації та виконання студентами спеціальності 121 «Інженерія програмного забезпечення» освітнього ступеня «бакалавр» / Г. І. Радельчук, Л. П. Бедратюк, – Хмельницький : ХНУ, 2021. – 40 с.
6. Системи навігації паркінгу [Електронний ресурс]. URL: https://line-llc.com/Avtomatuchna-parkovka/sustema-dinamichnoy-navogaciu-dlya-krutogo-parkingy (дата звернення: 20.02.2023).
7. Програмне забезпечення парковок. [Електронний ресурс]. URL: https://www.sea.com.ua/ua/parkovochnoe_oborudovanie/programmnoe-obespecenie-parkovok/ (дата звернення 17.02.2023).
8. Автоматизація роботи парковок. [Електронний ресурс]. URL: https://expertsolution.com.ua/uk/ (дата звернення 17.02.2023).
9. Як тестувати веб-сайт: основні етапи і поради / Brainlab [Електронний ресурс]. URL: https://brainlab.com.ua/uk/blog-uk/yak-testuvati-veb-sayt-osnovn-etapi-poradi (дата звернення: 25.03.2023).

10. Безпека життя і діяльності людини [Електронний ресурс]. URL: https://ela.kpi.ua/bitstream/123456789/48045/1/p.149-153.pdf (дата звернення: 17.02.2023).

11. Техніка безпеки [Електронний ресурс]. URL: https://www.victorija.ua/blanki-ta-formi-dokumentiv/instruktsiya-z-ohorony-pratsi-ta-pozhezhnoyi-bezpeky.html (дата звернення: 17.02.2023).

12. Організація процесу навчання у вищій школі [Електронний ресурс]. URL: https://pidru4niki.com/88905/pedagogika/organizatsiya_protsesu_navchannya_vishchiy_shkoli (дата звернення: 17.02.2023).

13. Web Application Testing: 8 Step Guide to Website Testing. Guru99 [Електронний ресурс]. URL: https://www.guru99.com/web-application-testing.html (дата звернення: 15.04.2023).

14. Art-lemon [Електронний ресурс]. URL: https://art-lemon.com/site-test (дата звернення: 15.05.2022).

15. Захарова О. В., Захарова Е. Г., Резниченко В. А. "Каталог наукових електронних бібліотек в Інтернет" // ІПС НАН України. – К., 2012. – 76 с.

16. Alessandro Del Sole. Visual Studio Code Distilled: Evolved Code Editing for Windows, macOS, and Linux / Alessandro Del Sole. – Cremona, Italy : Apress, 2019. – 221 p. https://doi.org/10.1007/978-1-4842-4224-7 https://www.pdfdrive.com/visual-studio-code-distilled-evolved-codeediting-for-windows-macos-and-linux-e185943962.html

17. David Flanagan. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. —7th Edition — O'Reilly Media, 2020. —706 p.l.

18. Волохін О. М. "Каталогізація цифрових ресурсів Інтернет: Дублінське ядро метаданих: посібник" /О.М. Волохін. – Кіровоград, 2013. – 70 с.

19. Сучасні інформаційні технології і популярні системи / О. Григоровська, С. Панасоовович // Зап. Чернівці. наук. б-ки ім. В. Стефаника. – 2014. – Вип. 13. – С. 519–522.

20. ДСТУ 8302:2015. "Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання". – Уведено вперше; чинний від 2016–07–01. – Київ : ДП «УкрНДНЦ», 2016. – 17 с.

21. Adobe [Електронний ресурс]. URL: https://helpx.adobe.com/ua/illustrator/using/design-website-layout.html (дата звернення: 27.04.2023).

22. Теорія та практика побудови баз даних. 8-е вид. Д. Кренке. – Вид.: Суми, 2017. – 800 с.

23. Модульне тестування. QaLight [ЕлектрЗнний ресурс]. URL: https://qalight.ua/baza-znaniy/modulne-testuvannya/ (дата звернення: 15.04.2023).

24. Уніфікована мова програмування UML. Портал знань [Електронний ресурс]. URL: http://www.znannya.org/?view=uml (дата звернення: 15.04.2023).

25. NOSQL – переваги та недоліки нереляційних баз даних. QAinfo [Електронний ресурс]. URL: https://qainfo.com.ua/qa-blog/46-nosql-perevagy-ta-nedoliky-nerelyatsijnyh-baz-danyh (дата звернення: 15.04.2023).

26. Тестування UX-прототипів як потрібна ланка розробки продукту. Інтерактивні та статичні прототипи. URL: https://rustrackers.ru/uk/setting-up-software/testirovanie-ux-prototipov-kak-neobhodimoe-zveno-razrabotki/ (дата звернення: 12.03.2023).

27. Роберт М. С. Чистий код: створення та рефакторинг за допомогою Agile / Мартін Сесіл Роберт. – Харків: Ранок, 2020. – 448 с. – ISBN 978-617-09-5285-1.

28. Роберт М. С. Чиста архітектура: мистецтво розроблення програмного забезпечення / Мартін Сесіл Роберт. – Харків: Ранок, 2019. – 368 с. – ISBN 978-617-09-5286-8.

29. Joseph A. C# 8.0 in a Nutshell / A. Joseph, J. Eric. – Sebastopol: O'Reilly, 2020. – 1102 с. – ISBN 978-1-492-05113-8.

30. Fowler M. Patterns of Enterprise Application Architecture / Martin Fowler., 2003. – 560 с.

31. Freeman A. Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages / Adam Freeman. – Berkeley, CA: Apress, 2020. – 1080 с.

32. Ward B. SQL Server 2019 Revealed. Including Big Data Clusters and Machine Learning / Bob Ward. – Berkeley, CA: Apress, 2019. – 422 с. – ISBN 978-1-4842-5418-9.

33. Fowler M. Refactoring: Improving the Design of Existing Code / Martin Fowler., 1999. – 431 с. – ISBN 978-0201485677.

34. ASP.NET Core fundamentals [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-5.0&tabs=windows.

35. Framework Design Guidelines [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/.

36. How to get Entity Framework Core [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.learnentityframeworkcore.com/efcore/how-to-get.

37. Querying Data [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/ef/core/querying/.

38. Vernon V. Implementing Domain-Driven Design / Vaughn Vernon., 2013. – 656 с. – ISBN 978-0321834577.

39. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software / Eric Evans., 2003. – 560 с. – ISBN 978-0321125217.

40. Seemann M. Advanced Unit Testing [Електронний ресурс] / Mark Seemann / – 2013. – Режим доступу до ресурсу: https://www.pluralsight.com/courses/advanced-unit-testing

ДОДАТОК А
(обов’язковий)

КОД (ЛІСТИНГ)

Startup.cs:

```
using CoolParking.WebAPI.Extensions;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;
using System.Text;

namespace CoolParking.WebAPI
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the
container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
                .AddJwtBearer(options => {
                    options.TokenValidationParameters = new TokenValidationParameters
                    {
                        ValidateIssuer = true,
                        ValidateAudience = true,
                        ValidateLifetime = true,
                        ValidateIssuerSigningKey = true,
                        ValidIssuer = Configuration["Jwt:Issuer"],
                        ValidAudience = Configuration["Jwt:Audience"],
                        IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
                    };
                });

            services.AddCors(options =>
                options.AddDefaultPolicy(builder => builder.AllowAnyOrigin()));

            services.AddParkingContext(Configuration);
            services.RegisterCustomServices(Configuration);

            services.AddAutoMapper();

            services.AddMvc();
            services.AddControllers();

            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title = "CoolParking.WebAPI", Version =
"v1" });
            });
        }

        // This method gets called by the runtime. Use this method to configure the HTTP
request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
```

```
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseSwagger();
                app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json",
"WebApplication1 v1"));
            }

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseCors(builder => builder
                .WithOrigins(Configuration["Cors:Origin"])
                .AllowAnyMethod()
                .AllowAnyHeader());

            app.UseAuthentication();
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}
```

ParkingSessionService.cs:

```
using CoolParking.BL.Models;
using CoolParking.BL.Interfaces;
using CoolParking.WebAPI.Interfaces;
using System.Collections.Generic;
using System.Linq;
using System;
using CoolParking.WebAPI.Models;

namespace CoolParking.WebAPI.Services
{
    public class ParkingSessionService : IParkingSessionService
    {
        private IParkingService parkingService;

        public ParkingSessionService(IParkingService parkingService)
        {
            this.parkingService = parkingService;
        }

        public void DeleteVehicle(string id)
        {
            parkingService.RemoveVehicle(id);
        }

        public string GetAllTransactions()
        {
            return parkingService.ReadFromLog();
        }

        public decimal GetBalance()
        {
            return parkingService.GetBalance();
        }

        public int GetCapacity()
        {
```

```
            return parkingService.GetCapacity();
        }

        public int GetFreePlaces()
        {
            return parkingService.GetFreePlaces();
        }

        public TransactionInfo[] GetLastTransactions()
        {
            return parkingService.GetLastParkingTransactions();
        }

        public Vehicle GetVehicle(string id)
        {
            var vehicle = parkingService.GetVehicles().FirstOrDefault(x => x.Id == id) ??
                throw new ArgumentNullException("Vehicle with the specified ID is not
parked");

            return vehicle;
        }

        public List<Vehicle> GetVehicles()
        {
            return new List<Vehicle>(parkingService.GetVehicles());
        }

        public Vehicle PostVehicle(VehicleData vehicleData)
        {
            var vehicle = new Vehicle(vehicleData.Id, (VehicleType)vehicleData.VehicleType,
(decimal)vehicleData.Balance);
            parkingService.AddVehicle(vehicle);
            return vehicle;
        }

        public Vehicle TopUpVehicle(string id, decimal sum)
        {
            parkingService.TopUpVehicle(id, sum);
            return GetVehicle(id);
        }
    }
}
```

ParkingController.cs

```
using CoolParking.Shared.DTOs.Parking;
using CoolParking.WebAPI.Interfaces;
using Microsoft.AspNetCore.Mvc;

namespace CoolParking.WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ParkingController : ControllerBase
    {
        private readonly IParkingSessionService parkingSessionService;

        public ParkingController(IParkingSessionService parkingSessionService)
        {
            this.parkingSessionService = parkingSessionService;
        }

        // GET api/parking/balance
        [HttpGet("[action]")]
        public ActionResult<BalanceDto> Balance()
        {
            return new BalanceDto(parkingSessionService.GetBalance());
```

```
        }

        // GET api/parking/capacity
        [HttpGet("[action]")]
        public ActionResult<CapacityDto> Capacity()
        {
            return new CapacityDto(parkingSessionService.GetCapacity());
        }

        // GET api/parking/freePlaces
        [HttpGet("[action]")]
        public ActionResult<FreePlacesDto> FreePlaces()
        {
            return new FreePlacesDto(parkingSessionService.GetFreePlaces());
        }
    }
}
```

VehicleController.cs

```
using AutoMapper;
using CoolParking.BL.Models;
using CoolParking.Shared.DTOs.Parking;
using CoolParking.WebAPI.Interfaces;
using CoolParking.WebAPI.Models;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;

namespace CoolParking.WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class VehiclesController : ControllerBase
    {
        private readonly IParkingSessionService parkingSessionService;
        private readonly IMapper mapper;

        public VehiclesController(IParkingSessionService parkingSessionService, IMapper
mapper)
        {
            this.parkingSessionService = parkingSessionService;
            this.mapper = mapper;
        }

        // GET api/vehicles
        [HttpGet]
        public ActionResult<List<VehicleDto>> GetVehicles()
        {
            return mapper.Map<List<VehicleDto>>(parkingSessionService.GetVehicles());
        }

        // GET api/vehicles/id
        [HttpGet("{id}")]
        public ActionResult<VehicleDto> GetVehicle(string id)
        {
            if (!Vehicle.IsValidRegistrationPlateNumber(id))
            {
                return BadRequest();
            }

            try
            {
                return mapper.Map<VehicleDto>(parkingSessionService.GetVehicle(id));
            }
            catch (ArgumentNullException)
            {
```

```
                return NotFound();
            }
        }

        // POST api/vehicles
        [HttpPost]
        public ActionResult<VehicleDto> PostVehicle(VehicleData vehicle)
        {
            try
            {
                var response = parkingSessionService.PostVehicle(vehicle);
                return CreatedAtAction(nameof(GetVehicle), new { id = vehicle.Id },
mapper.Map<VehicleDto>(response));
            }
            catch (ArgumentException)
            {
                return BadRequest();
            }
        }

        // DELETE api/vehicles/id
        [HttpDelete("{id}")]
        public IActionResult DeleteVehicle(string id)
        {
            if (!Vehicle.IsValidRegistrationPlateNumber(id))
            {
                return BadRequest();
            }

            try
            {
                parkingSessionService.DeleteVehicle(id);
                return NoContent();
            }
            catch (ArgumentException)
            {
                return NotFound();
            }
            catch (InvalidOperationException)
            {
                return Conflict();
            }
        }
    }
}
```

AuthController.cs

```
using CoolParking.BL.Interfaces;
using CoolParking.Shared.DTOs.UserProfile;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace CoolParking.WebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class AuthController : ControllerBase
    {
        private readonly IAuthService authService;

        public AuthController(IAuthService authService)
        {
            this.authService = authService;
        }
```

```
        [AllowAnonymous]
        [HttpPost("log-in")]
        public async Task<ActionResult<string>> LogIn(UserLogInDto userLogin)
        {
            var token = await authService.ObtainJwTokenAsync(userLogin);
            return string.IsNullOrEmpty(token) ? NotFound("User not found") : Ok(new { token
});
        }
    }
}
```

ParkingContext.cs

```
using CoolParking.DAL.Entities;
using CoolParking.DAL.Extensions;
using Microsoft.EntityFrameworkCore;

namespace CoolParking.DAL
{
    public class ParkingContext : DbContext
    {
        public ParkingContext(DbContextOptions options) : base(options)
        {
        }

        public DbSet<Employee> Employees { get; set; }
        public DbSet<Position> Positions { get; set; }
        public DbSet<Role> Roles { get; set; }
        public DbSet<Service> Services { get; set; }
        public DbSet<Tariff> Tariffs { get; set; }
        public DbSet<UserProfile> UserProfiles { get; set; }
        public DbSet<Vehicle> Vehicles { get; set; }
        public DbSet<VehicleType> VehicleTypes { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Configure();
            modelBuilder.Seed();
        }
    }
}
```

AuthService.cs

```
using AutoMapper;
using CoolParking.BL.Interfaces;
using CoolParking.DAL;
using CoolParking.Shared.DTOs.UserProfile;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;
using System;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;

namespace CoolParking.BL.Services
{
    public class AuthService : IAuthService
    {
        private readonly ParkingContext context;
        private readonly IConfiguration configuration;
        private readonly IMapper mapper;

        public AuthService(ParkingContext context, IConfiguration configuration, IMapper
mapper)
        {
```

```
            this.context = context;
            this.configuration = configuration;
            this.mapper = mapper;
        }

        public async Task<string> ObtainJwTokenAsync(UserLogInDto userLogin)
        {
            var user = await Authenticate(userLogin);
            return user == default ? string.Empty : Generate(user);
        }

        private async Task<UserProfileDto> Authenticate(UserLogInDto userLogin)
        {
            var currentUser = await context.UserProfiles
                .Include(x => x.Employee)
                .Include(x => x.Role)
                .SingleOrDefaultAsync(x =>
                x.Username == userLogin.Username &&
                x.Password == userLogin.Password);

            return mapper.Map<UserProfileDto>(currentUser ?? default);
        }

        private string Generate(UserProfileDto user)
        {
            var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["Jwt:Key"]));
            var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);

            var claims = new[]
            {
                new Claim(ClaimTypes.NameIdentifier, user.Username),
                new Claim(ClaimTypes.GivenName, user.GivenName),
                new Claim(ClaimTypes.Surname, user.Surname),
                new Claim(ClaimTypes.Role, user.Role)
            };

            var token = new JwtSecurityToken(configuration["Jwt:Issuer"],
                configuration["Jwt:Audience"],
                claims,
                DateTime.Now,
                DateTime.Now.AddMinutes(30),
                credentials
            );

            return new JwtSecurityTokenHandler().WriteToken(token);
        }
    }
}
```

ParkingService.cs

```
using AutoMapper;
using CoolParking.BL.Interfaces;
using CoolParking.BL.Models;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Text;
using System.Timers;

namespace CoolParking.BL.Services
{
    public class ParkingService : IParkingService, IDisposable
    {
        private readonly Parking parking;
```

```
        private readonly ITimerService withdrawTimer;
        private readonly ITimerService logTimer;
        private readonly ILogService logService;
        private readonly IMapper mapper;
        private readonly Dictionary<VehicleType, decimal> tariffs;
        private List<TransactionInfo> BufferedParkingTransactions { get; } = new
List<TransactionInfo>();

        public ParkingService(ITimerService withdrawTimer, ITimerService logTimer,
ILogService logService, IMapper mapper = null)
        {
            parking = Parking.Instance;

            this.withdrawTimer = withdrawTimer;
            this.logTimer = logTimer;

            this.withdrawTimer.Elapsed += WithdrawTimer_Elapsed;
            this.logTimer.Elapsed += LogTimer_Elapsed;

            this.logService = logService;
            this.mapper = mapper;
            tariffs = new Dictionary<VehicleType, decimal>()
            {
                { VehicleType.PassengerCar, Settings.PassengerCarTariff },
                { VehicleType.Truck, Settings.TruckTariff },
                { VehicleType.Bus, Settings.BusTariff },
                { VehicleType.Motorcycle, Settings.MotorcycleTariff }
            };
        }

        private void WithdrawTimer_Elapsed(object sender, ElapsedEventArgs e)
        {
            foreach (var vehicle in parking.Vehicles)
            {
                ExecuteTransaction(vehicle, GetAmount(vehicle));
            }
        }

        private void ExecuteTransaction(Vehicle vehicle, decimal amount)
        {
            vehicle.Balance -= amount;
            parking.Balance += amount;
            lock (BufferedParkingTransactions)
            {
                BufferedParkingTransactions.Add(new TransactionInfo(vehicle.Id, amount));
            }
        }

        private decimal GetAmount(Vehicle vehicle)
        {
            decimal initialBalance = vehicle.Balance;
            decimal tariff = tariffs[vehicle.VehicleType];
            decimal fine = Settings.FineCoefficient;

            if (vehicle.Balance < tariff)
            {
                if (initialBalance <= 0)
                {
                    return tariff * fine;
                }
                else
                {
                    return (tariff - initialBalance) * fine + initialBalance;
                }
            }
            else
            {
```

```
                return tariff;
            }
        }

        private void LogTimer_Elapsed(object sender, ElapsedEventArgs e)
        {
            var sb = new StringBuilder();
            TransactionInfo[] buffer;

            lock (BufferedParkingTransactions)
            {
                buffer = BufferedParkingTransactions.ToArray();
                BufferedParkingTransactions.Clear();
            }

            foreach (var transaction in buffer)
            {
                sb.Append($"[{transaction.Time}] ID: {transaction.Id} Amount:
{transaction.Sum}\n");
            }

            logService.Write(sb.ToString().TrimEnd('\n'));
        }

        public void AddVehicle(Vehicle vehicle)
        {
            if (isParked(vehicle))
            {
                throw new ArgumentException("Vehicle with the specified ID is already
parked");
            }

            if (parking.Vehicles.Count < Settings.Capacity)
            {
                parking.Vehicles.Add(vehicle);
            }
            else
            {
                throw new InvalidOperationException("Parking is completely full");
            }
        }

        private bool isParked(Vehicle vehicle) => GetVehicle(vehicle.Id) is not null;

        public void Dispose()
        {
            withdrawTimer.Elapsed -= WithdrawTimer_Elapsed;
            logTimer.Elapsed -= LogTimer_Elapsed;

            parking.Vehicles.Clear();
            parking.Balance = 0.0M;
        }

        public decimal GetBalance() => parking.Balance;

        public int GetCapacity() => Settings.Capacity;

        public int GetFreePlaces() =>
            Settings.Capacity - parking.Vehicles.Count;

        public TransactionInfo[] GetLastParkingTransactions() =>
            BufferedParkingTransactions.ToArray();

        public ReadOnlyCollection<Vehicle> GetVehicles() =>
            new ReadOnlyCollection<Vehicle>(parking.Vehicles);

        public string ReadFromLog() => logService.Read();
```

```
        public void RemoveVehicle(string id)
        {
            var vehicle = GetVehicle(id) ??
                throw new ArgumentException("Vehicle with the specified ID is not parked");

            if (vehicle.Balance < 0.0M)
            {
                throw new InvalidOperationException("Vehicle has a negative balance");
            }
            else
            {
                parking.Vehicles.Remove(vehicle);
            }
        }

        public void TopUpVehicle(string id, decimal sum)
        {
            var vehicle = GetVehicle(id) ??
                throw new ArgumentException("Vehicle with the specified ID is not parked");

            if (sum > 0.0M)
            {
                vehicle.Balance += sum;
            }
            else
            {
                throw new ArgumentException("The top-up amount must be greater than zero");
            }
        }

        public Vehicle GetVehicle(string id) =>
            parking.Vehicles.Find(i => i.Id == id);
    }
}
```

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { CashierComponent } from './components/cashier/cashier.component';
import { ManagerComponent } from './components/manager/manager.component';
import { SigninComponent } from './components/signin/signin.component';
import { SignupComponent } from './components/signup/signup.component';
import { UserProfileComponent } from './components/user-profile/user-
profile.component';
import { AuthGuard } from './guards/auth.guard';
import { CashierGuard } from './guards/cashier.guard';
import { ManagerGuard } from './guards/manager.guard';

const routes: Routes = [
  { path: '', redirectTo: '/log-in', pathMatch: 'full' },
  { path: 'log-in', component: SigninComponent },
  { path: 'sign-up', component: SignupComponent },
  { path: 'user-profile', component: UserProfileComponent, canActivate: [AuthGuard]
},
  { path: 'cashier', component: CashierComponent, canActivate: [CashierGuard] },
  { path: 'manager', component: ManagerComponent, canActivate: [ManagerGuard] }
];

@NgModule({
```

```
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

app-component.html

```
<div
  class="d-flex flex-column flex-md-row align-items-center p-3 px-md-4 mb-3 bg-
white border-bottom shadow-sm fixed-top">
  <h1 class="my-0 mr-md-auto font-weight-normal">SmartPark</h1>
  <nav class="my-2 my-md-0 mr-md-3">
    <a *ngIf="!this.authService.isLoggedIn" class="p-2 text-dark"
routerLinkActive="active" routerLink="/log-in">Sign
      in</a>
  </nav>
  <a *ngIf="!this.authService.isLoggedIn" class="btn btn-outline-primary"
routerLinkActive="active"
    routerLinkActive="active" routerLink="/sign-up">Sign up</a>
  <button (click)="logout()" *ngIf="this.authService.isLoggedIn" type="button"
class="btn btn-danger">Log out</button>
</div>

<router-outlet></router-outlet>
```

cashier.component.html

```
<div class="container">
    <div class="row">
        <div class="inner-main">
            <mat-table class="marg-auto" mat-table [dataSource]="dataSource">

                <ng-container matColumnDef="id">
                    <mat-header-cell *matHeaderCellDef> <span>Balance:</span>
{{balance}} $</mat-header-cell>
                </ng-container>

                <ng-container matColumnDef="freePlaces">
                    <mat-header-cell *matHeaderCellDef> <span>Free places:</span>
{{freePlaces}} </mat-header-cell>
                </ng-container>

                <ng-container matColumnDef="capacity">
                    <mat-header-cell *matHeaderCellDef> <span>Capacity:</span>
{{capacity}} </mat-header-cell>
                </ng-container>

                <ng-container matColumnDef="button">
                    <mat-header-cell *matHeaderCellDef>
                        <button (click)="parkOpenDialog()" type="button" class="btn
btn-outline-info btn-park">Park
                        </button>
```

```
                        <button (click)="refresh()" type="button" class="btn btn-
outline-info btn-refresh">Refresh
                        </button>
                    </mat-header-cell>
                </ng-container>

                <mat-header-row *matHeaderRowDef="['id', 'freePlaces', 'capacity',
'button']">
                </mat-header-row>

            </mat-table>

            <div class="search-container marg-auto">
                <div class="md-form active-cyan-2 mb-3">
                    <input class="form-control" type="text"
(keyup)="applyFilter($event)" placeholder="Search"
                        aria-label="Search">
                </div>
            </div>

            <mat-table class="marg-auto mat-table-mb" mat-table
[dataSource]="dataSource" matSort>

                <ng-container matColumnDef="id">
                    <mat-header-cell *matHeaderCellDef mat-sort-header> License
plate </mat-header-cell>
                    <mat-cell *matCellDef="let vehicle"> {{vehicle.id}} </mat-cell>
                </ng-container>

                <ng-container matColumnDef="vehicleType">
                    <mat-header-cell *matHeaderCellDef mat-sort-header> Vehicle
type </mat-header-cell>
                    <mat-cell *matCellDef="let vehicle"> {{vehicle.vehicleType}}
</mat-cell>
                </ng-container>

                <ng-container matColumnDef="balance">
                    <mat-header-cell *matHeaderCellDef mat-sort-header> Balance
</mat-header-cell>
                    <mat-cell *matCellDef="let vehicle"> {{vehicle.balance}} $
</mat-cell>
                </ng-container>

                <mat-header-row *matHeaderRowDef="['id', 'vehicleType',
'balance']">
                </mat-header-row>
                <mat-row *matRowDef="let row; columns: ['id', 'vehicleType',
'balance']" (click)="openDialog(row)">
                </mat-row>

            </mat-table>
```

```
            <mat-paginator [pageSizeOptions]="[5, 10, 20]"
showFirstLastButtons></mat-paginator>
        </div>
    </div>
</div>
```

cashier.component.css

```
.space {
    padding-bottom: 20px;
}
.w100{
    width: 100%;
}

mat-table {
    width: 85%;
    background-color: transparent;
}

.mat-table-mb {
    height: 50vh;
}

span {
    margin-right: 10px;
}

.search-bar {
    width: 74%;
    box-sizing: border-box;
    height: 100%;
    border: none;
    background-color: initial;
    outline: none;
}

.searchBox{
    height: 40px;
    height: 40px;
    width: 58%;
    background-color: rgba(91, 127, 220, 0.1);;
    color: black;
}

.mt40{
    margin-top: 40px;
}

.marg-auto{
    margin: auto;
}
```

```
mat-row{
    box-shadow: 0px 0px 20px rgba(26, 35, 126, 0.15);
    margin-top: 8px;
    background-color: #ffffff;
    border-radius: 5px;
    border-top: none;
}

mat-cell{
    height: 48px;
    font-weight: 400;
}

mat-header-cell,mat-cell{
    font-family: 'Open Sans';
    color: #384853;
    font-size: 14px;
    line-height: 19.07px;
}

mat-header-row{
    border-bottom: transparent;
}

mat-header-cell{
    font-weight: 600;
    font-size: 14px;
    opacity: 80%;
}

.btn-refresh {
    margin-left: 10px;
}

.btn-park {
    margin-left: 100px;
}

.mat-row:hover {
    background-color: #E8EAF6;
    cursor: pointer;
  }

.active-cyan-2 input.form-control[type=text]:focus:not([readonly]) {
    border-bottom: 1px solid #4dd0e1;
    box-shadow: 0 1px 0 0 #4dd0e1;
}

.active-cyan input.form-control[type=text] {
    border-bottom: 1px solid #4dd0e1;
    box-shadow: 0 1px 0 0 #4dd0e1;
}
```

```
.search-container {
    margin-top: 30px;
    margin-bottom: 30px;
    width: 85%;
}
```

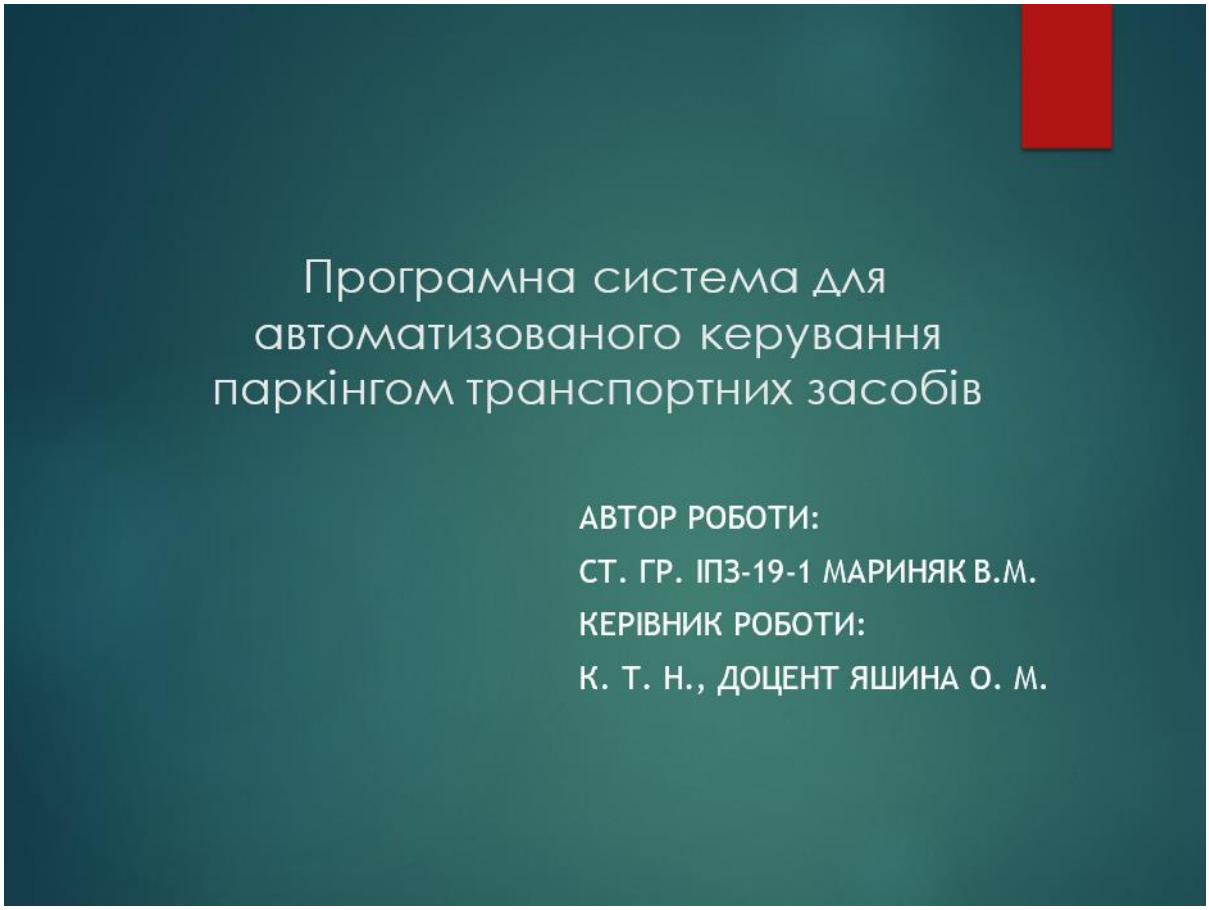
cashier.component.ts

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { MatDialog } from '@angular/material/dialog';
import { ParkingService } from 'src/app/services/parking.service';
import { Balance } from 'src/app/shared/balance';
import { Capacity } from 'src/app/shared/capacity';
import { FreePlaces } from 'src/app/shared/free-places';
import { Vehicle } from 'src/app/shared/vehicle';
import { MatPaginator } from '@angular/material/paginator';
import { MatSort } from '@angular/material/sort';
import { MatTableDataSource } from '@angular/material/table';
import { TopUpVehicleComponent } from '../dialogs/top-up-vehicle/top-up-
vehicle.component';
import { Payment } from 'src/app/shared/payment';
import { ParkVehicleComponent } from '../dialogs/park-vehicle/park-
vehicle.component';

export interface TopUpVehicleDialogData {
  balance: number;
  id: string;
}

export interface ParkVehicleDto {
  id: string;
  vehicleType: number;
  balance: number;
}

@Component({
  selector: 'app-cashier',
  templateUrl: './cashier.component.html',
  styleUrls: ['./cashier.component.css']
})
export class CashierComponent implements OnInit {

  animal: string;
  name: string = 'Pes';

  vehicle: ParkVehicleDto = {} as ParkVehicleDto;

  @ViewChild(MatPaginator) paginator!: MatPaginator;
  @ViewChild(MatSort) sort!: MatSort;

  balance: number = 0;
  freePlaces: number = 0;
```

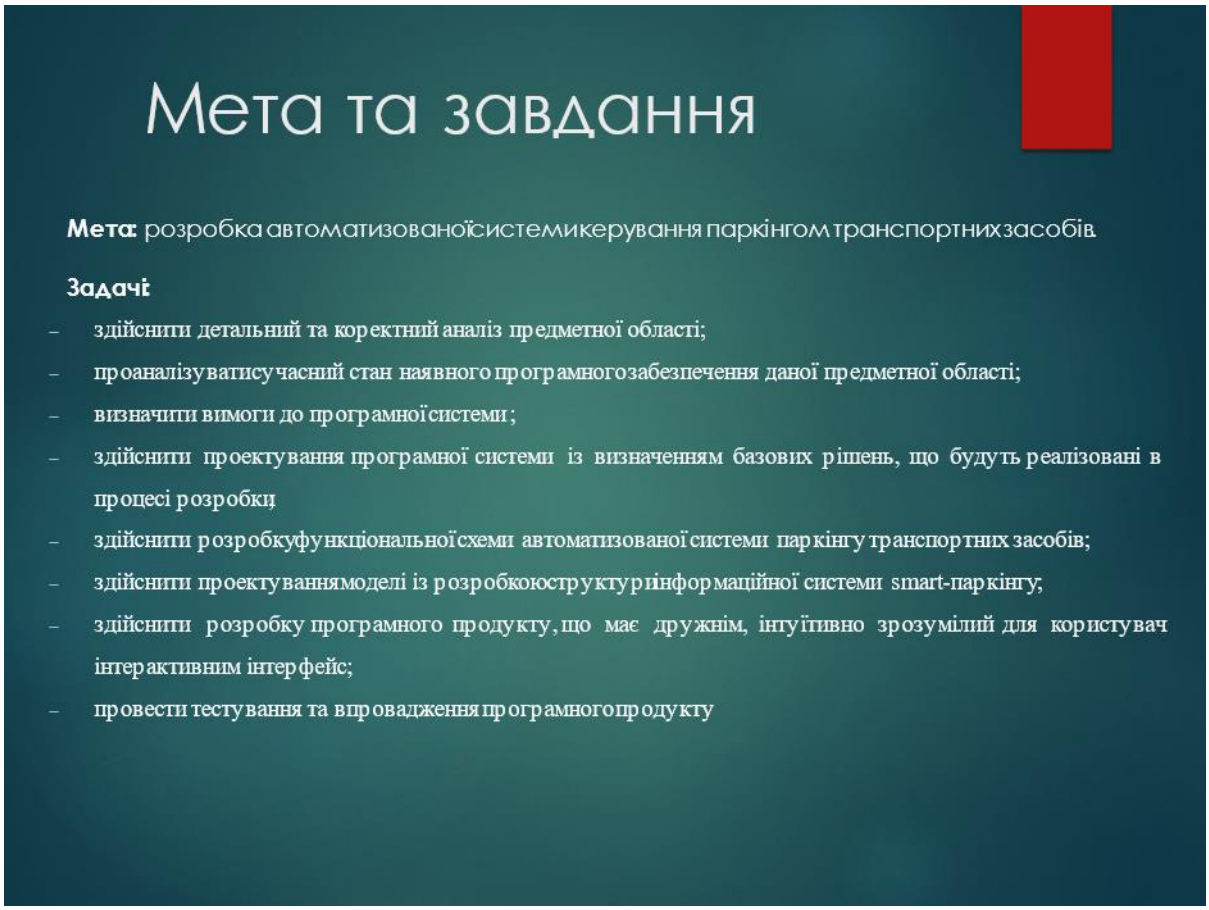
```
  capacity: number = 0;

  vehicles: Vehicle[] = [];

  constructor(
    public parkingService: ParkingService,
    public dialog: MatDialog
  ) { }

  openDialog(vehicle: Vehicle): void {
    const dialogRef = this.dialog.open(TopUpVehicleComponent, {
      width: '330px',
      data: { id: vehicle.id, balance: this.animal }
    });

    dialogRef.afterClosed()
      .subscribe((result: TopUpVehicleDialogData) => {
        if (result) {
          this.parkingService.topUpVehicle({ id: result.id, sum: result.balance }
as Payment)
            .subscribe(() => { this.refresh() });
        }
        this.refresh();
      });
  }

  parkOpenDialog(): void {
    const dialogRef = this.dialog.open(ParkVehicleComponent, {
      width: '250px',
      data: this.vehicle
    });

    dialogRef.afterClosed()
      .subscribe((result: ParkVehicleDto) => {
        if (result) {
          this.parkingService.parkVehicle(this.vehicle)
            .subscribe(() => { this.refresh(); this.vehicle = {} as ParkVehicleDto
});
        }
      });
  }

  releaseVehicle(vehicle: Vehicle): void {
    console.log(vehicle.id)
  }

  public dataSource: MatTableDataSource<Vehicle> = new MatTableDataSource();

  ngOnInit(): void {
    this.refresh();
  }
}
```

ДОДАТОК Б
(обов’язковий)

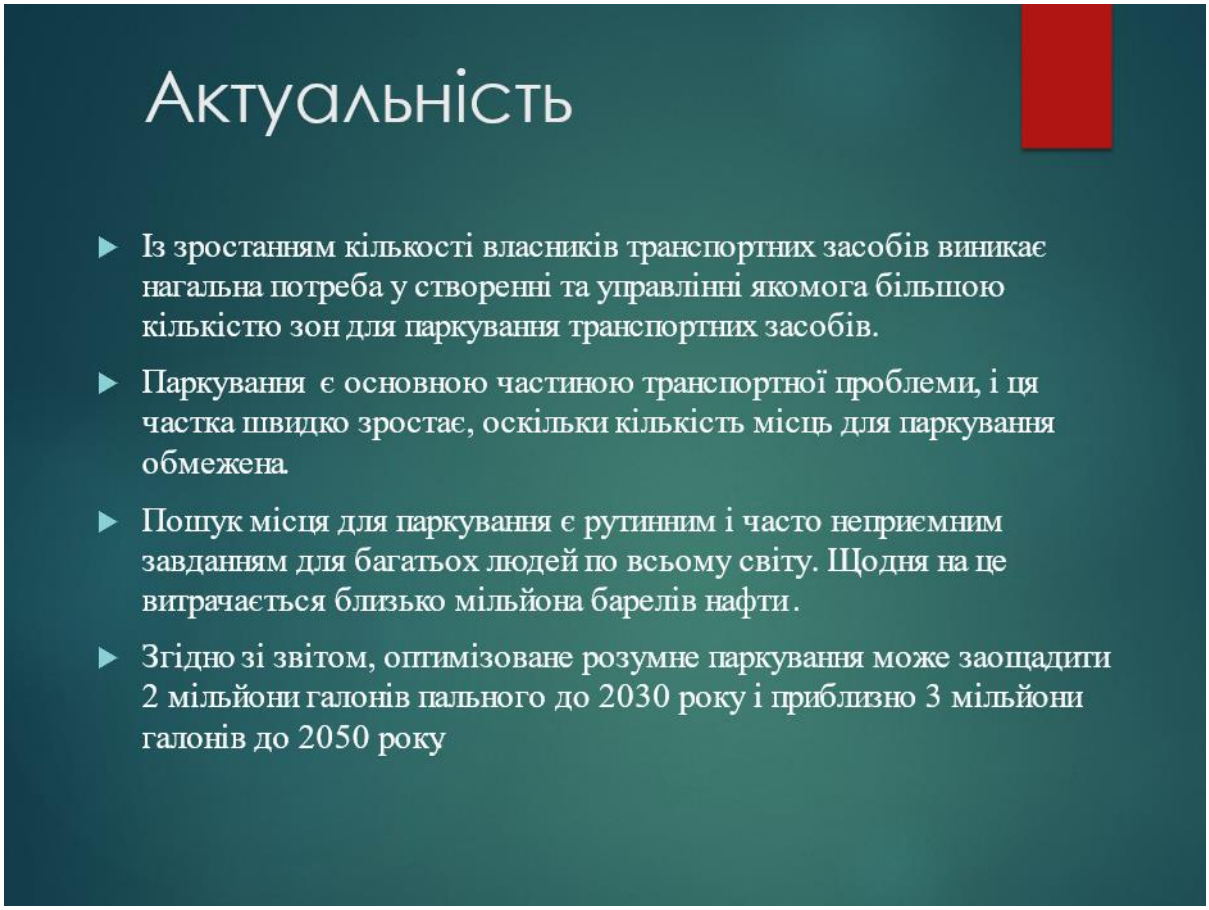
ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ



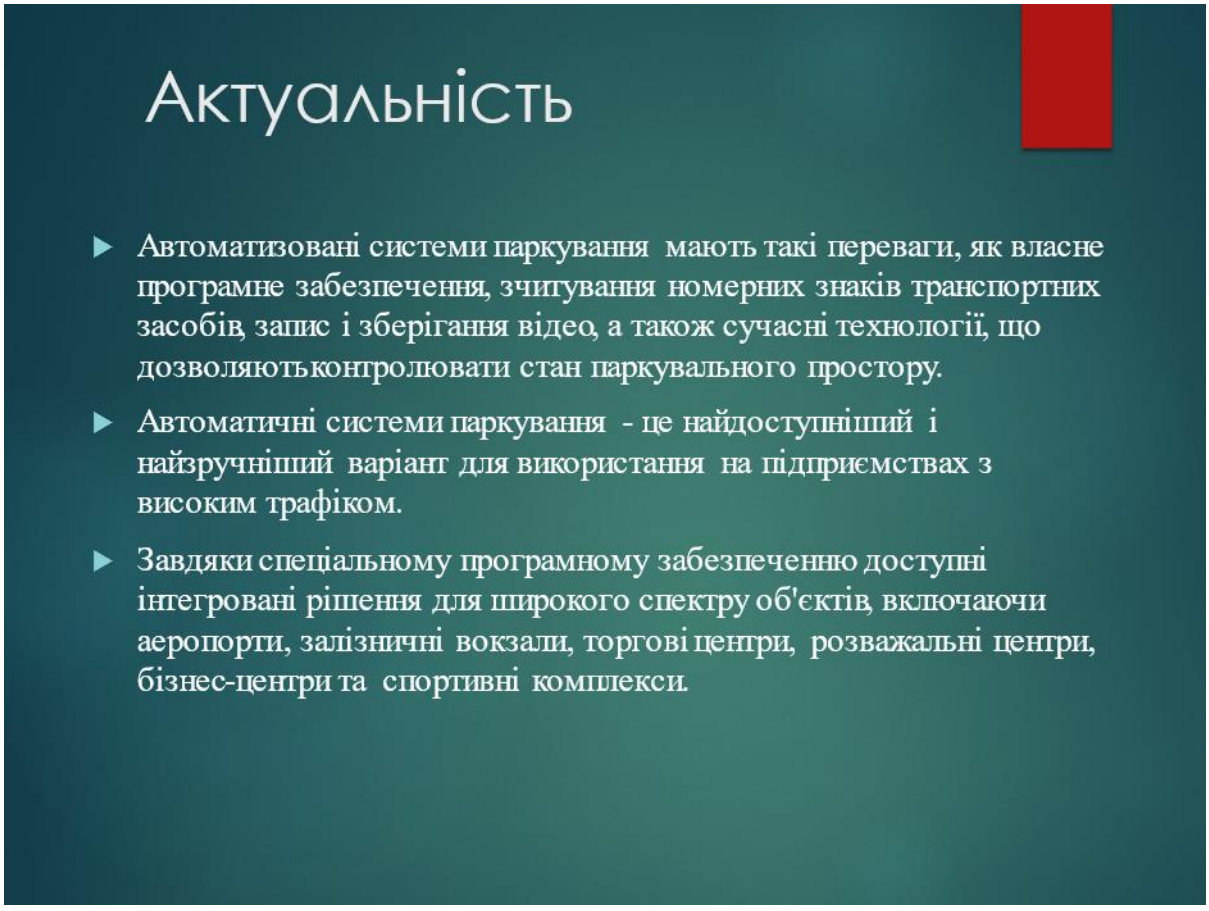
Слайд 1 – Вступ



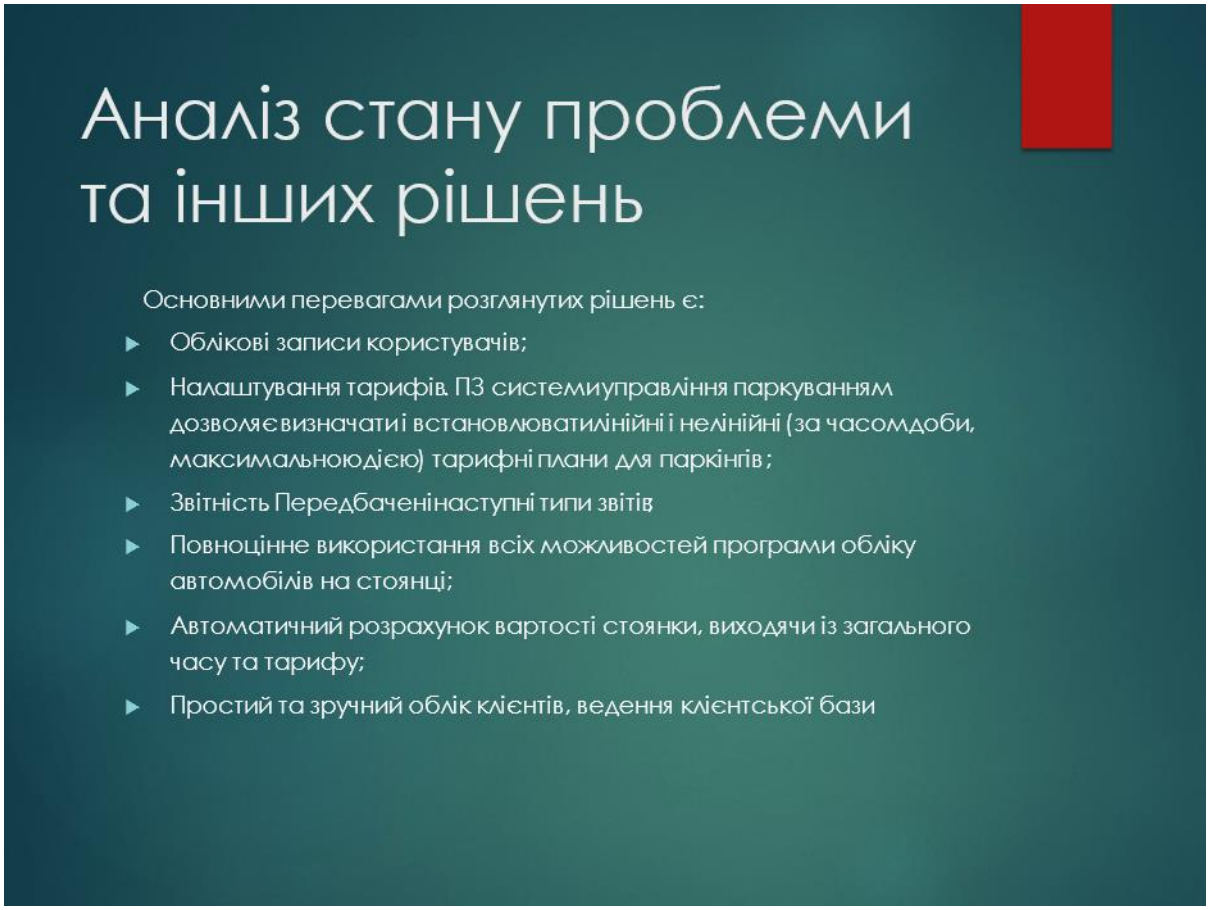
Слайд 2 – Мета та завдання



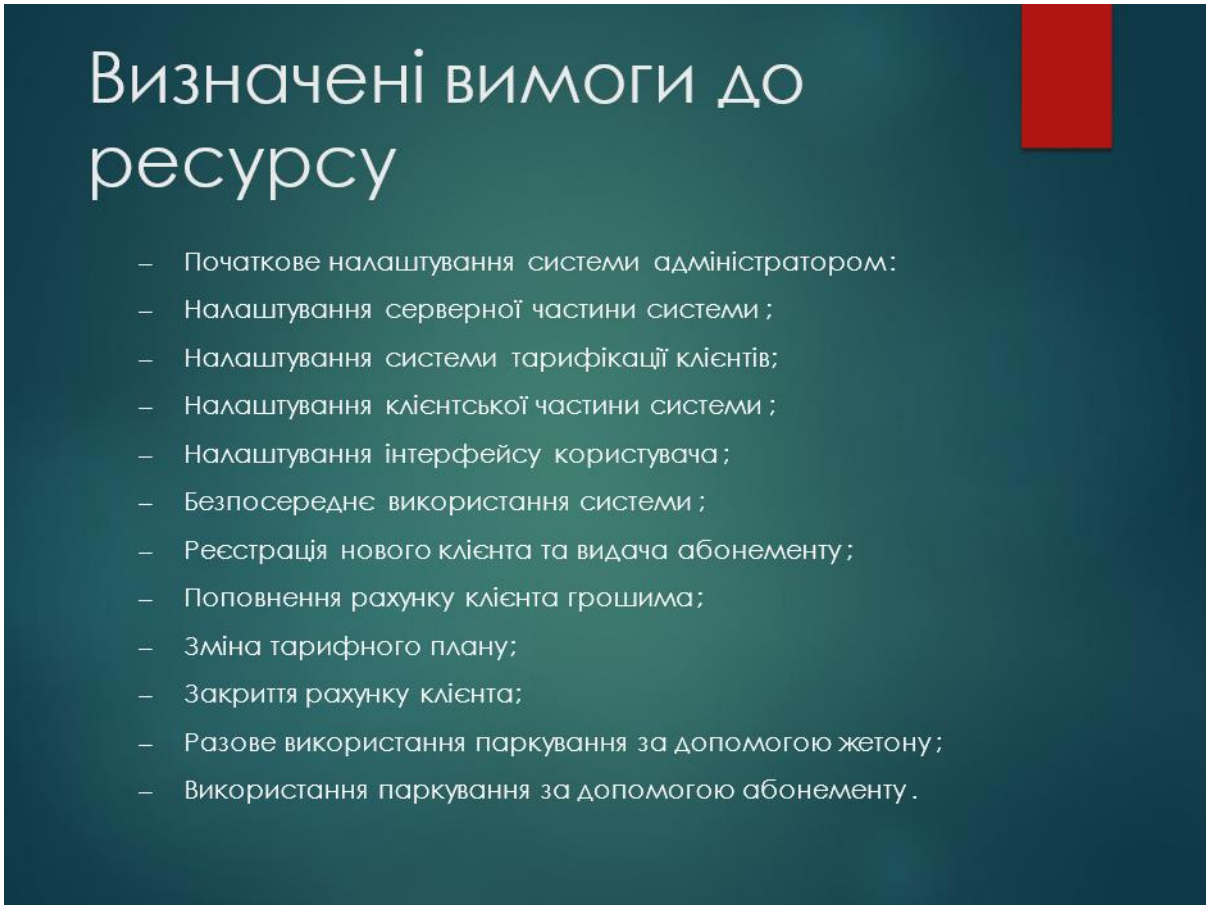
Слайд 3 – Актуальність



Слайд 4 – Актуальність

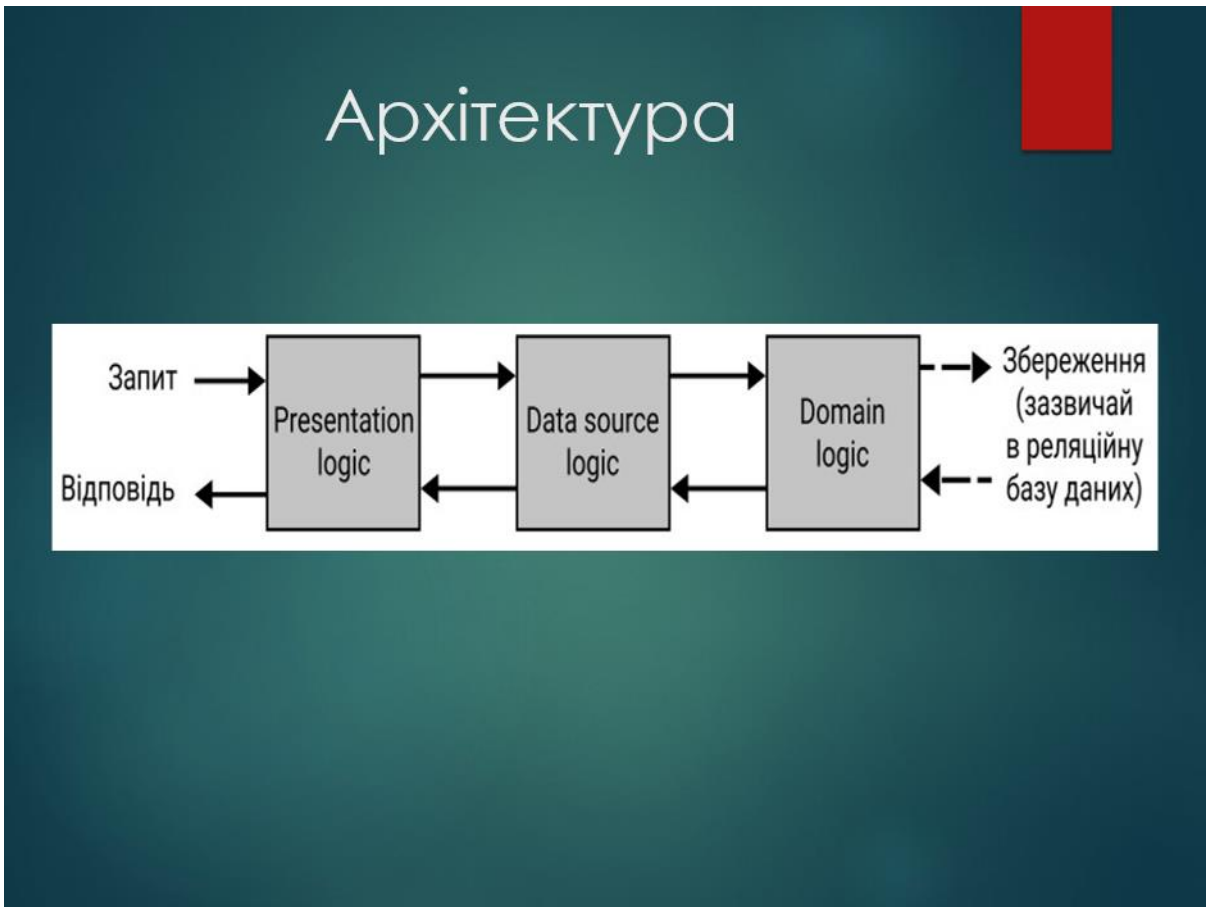


Слайд 5 – Аналіз стану проблеми та інших рішень

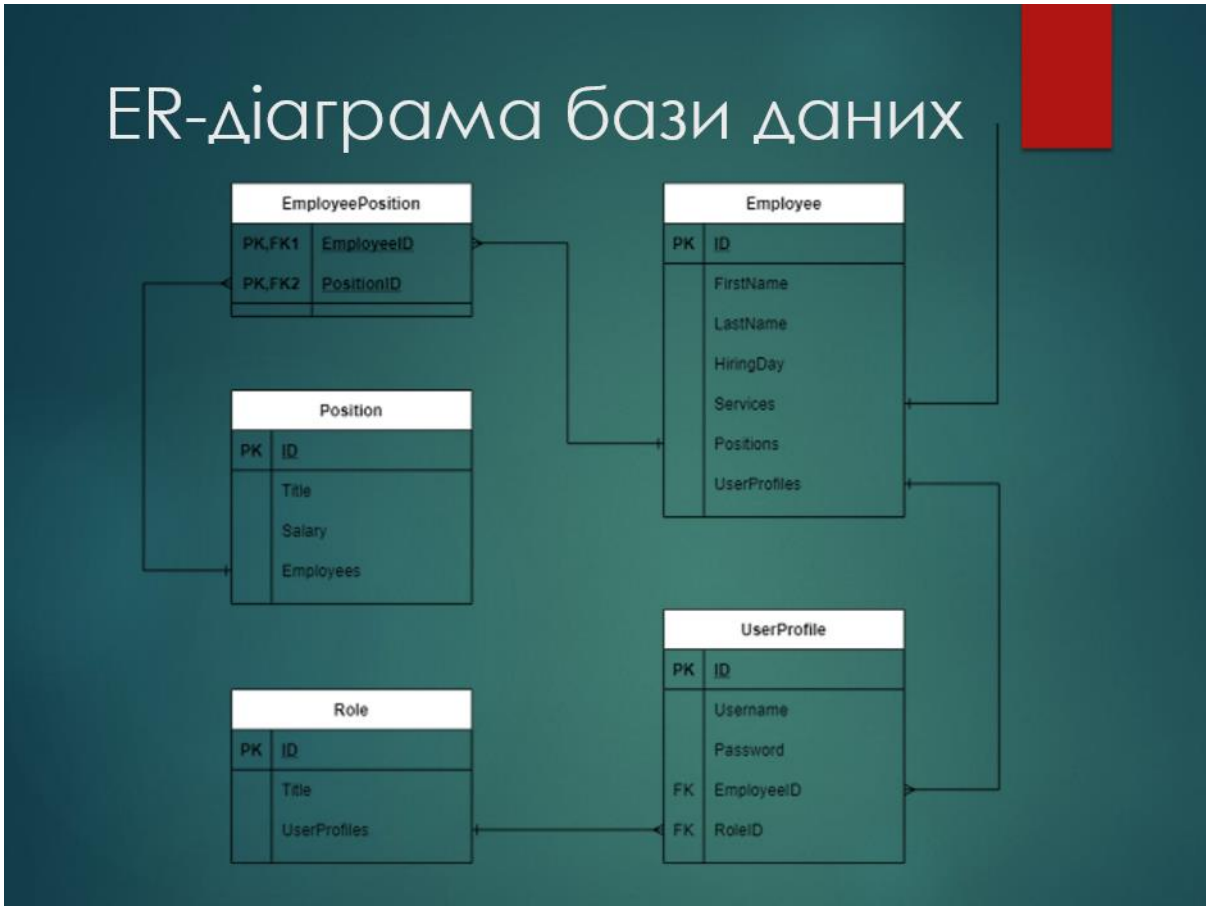


Слайд 6 – Визначені вимоги до ресурсу

Слайд 7 – Діаграма варіантів використання

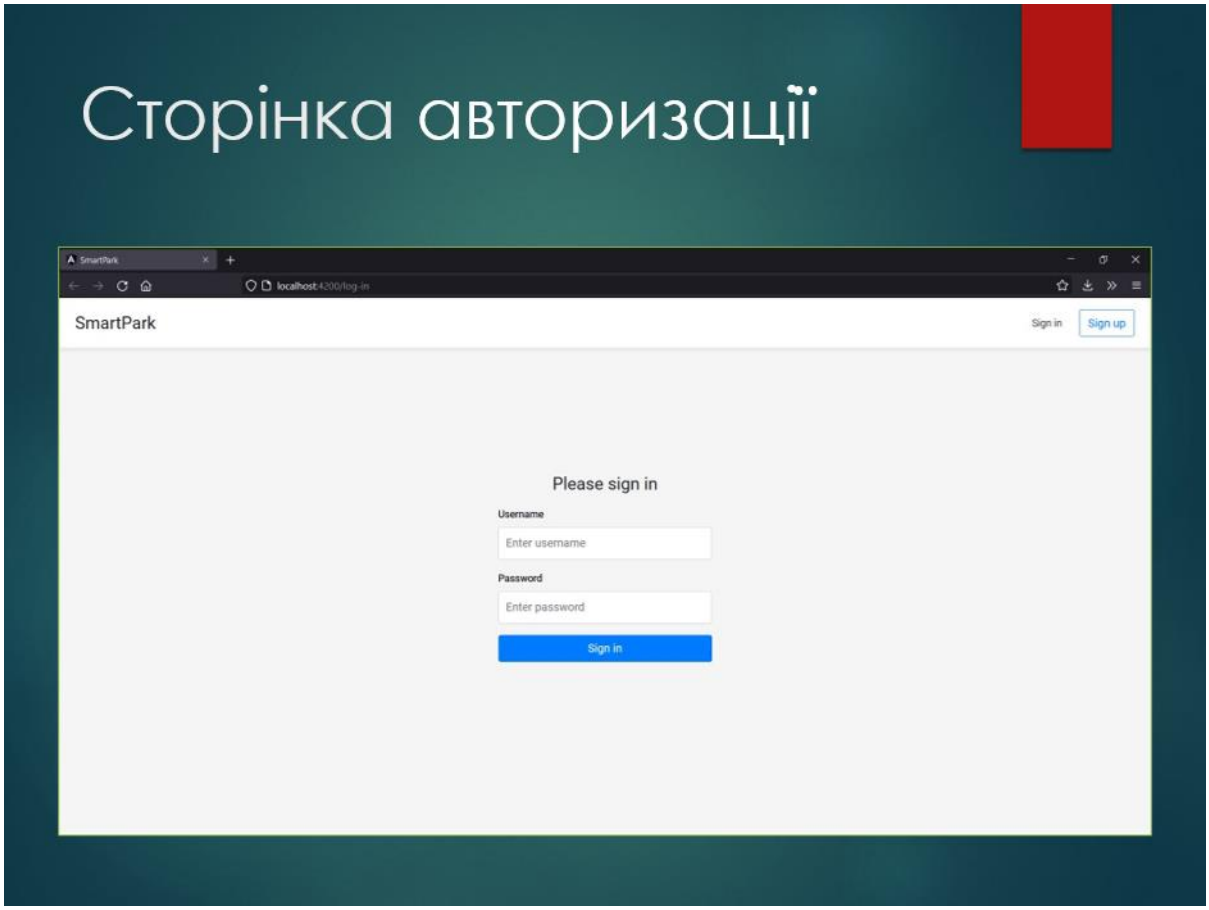


Слайд 8 – Архітектура

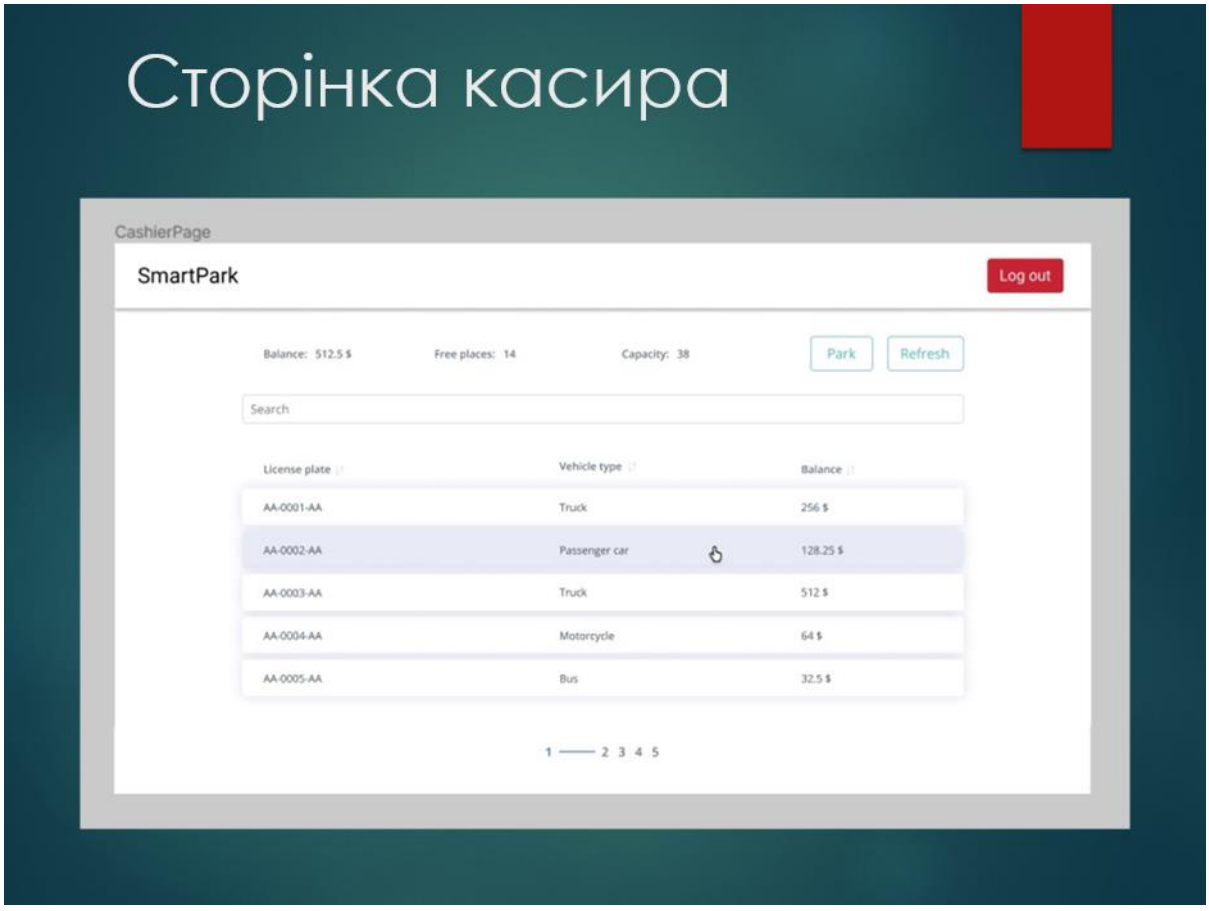


Слайд 9 – ER-діаграма бази даних

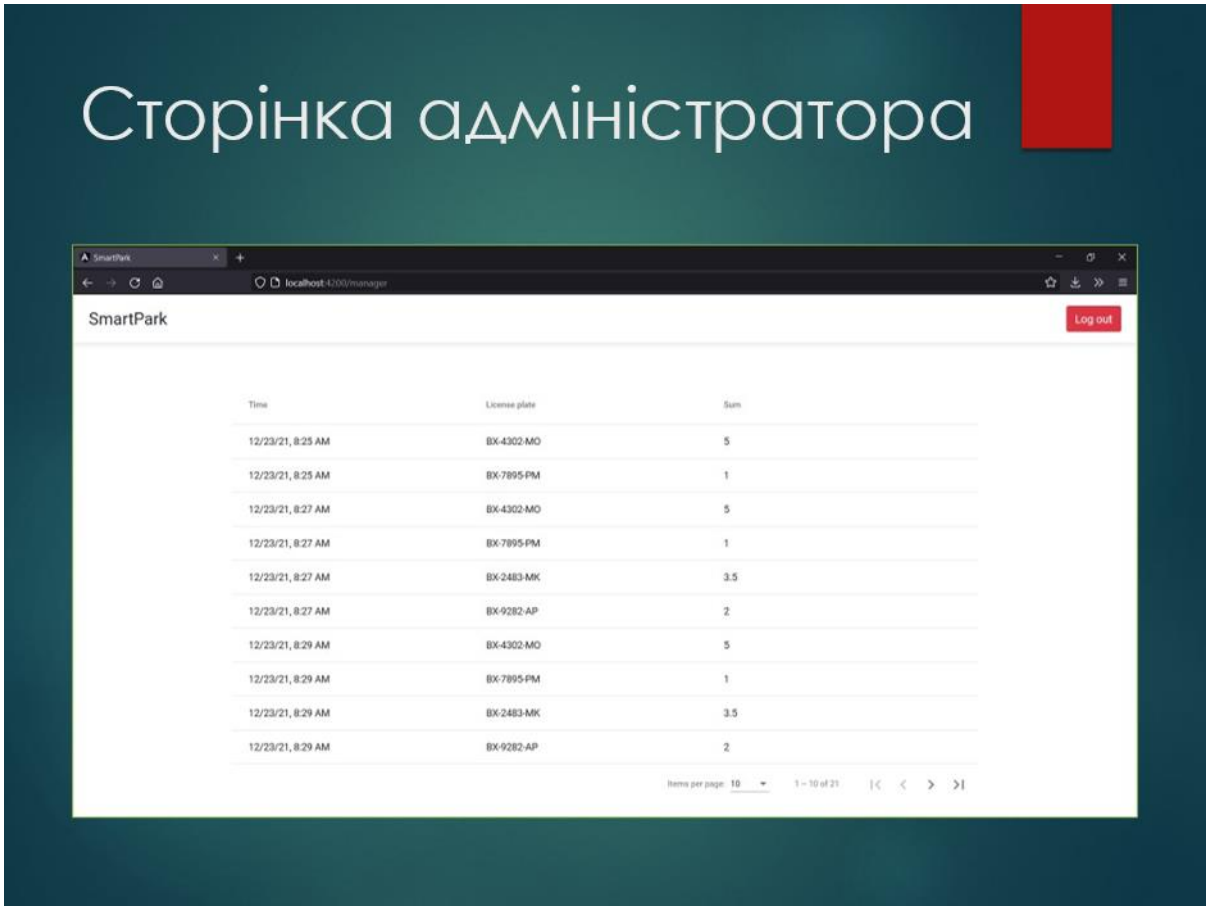
Слайд 10 – ER-діаграма бази даних



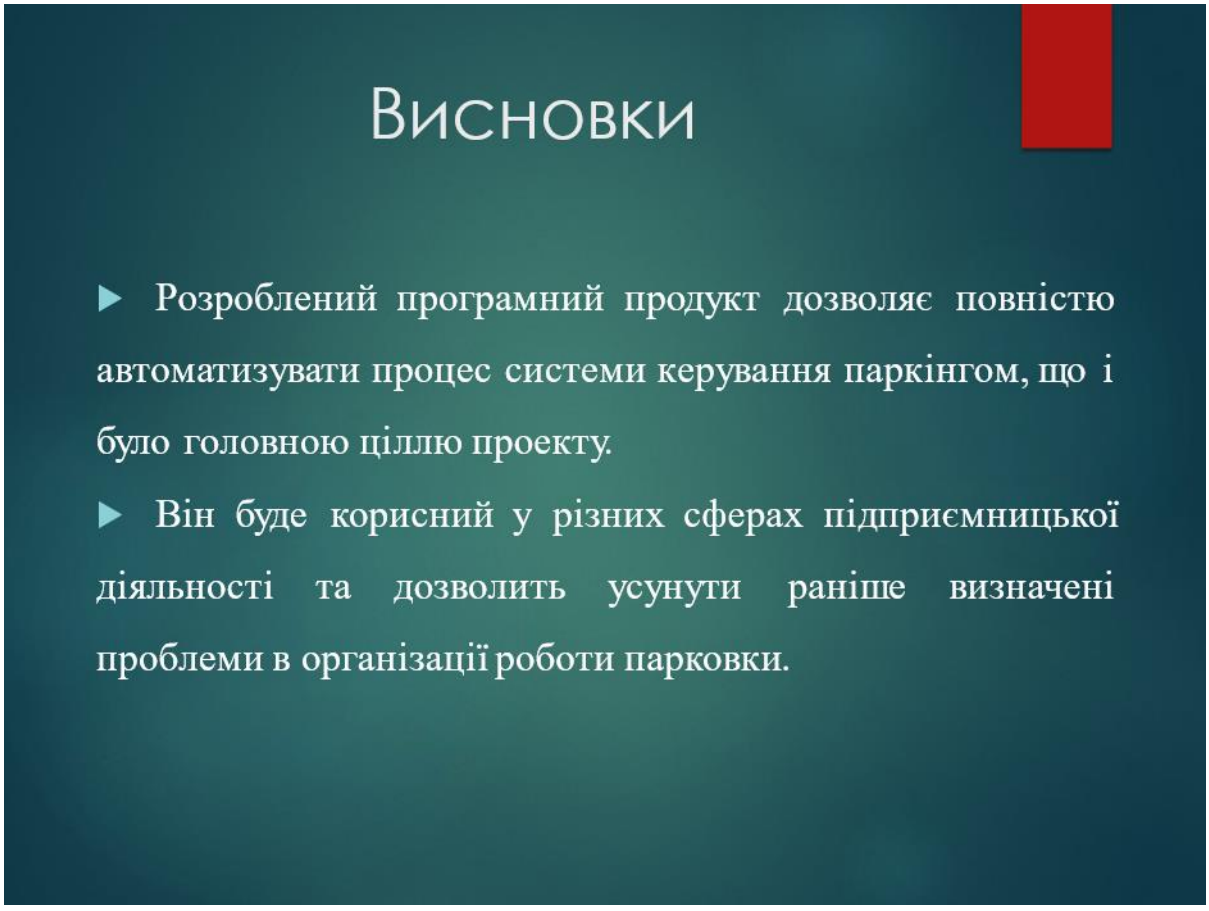
Слайд 11 – Сторінка авторизації



Слайд 12 – Сторінка касира

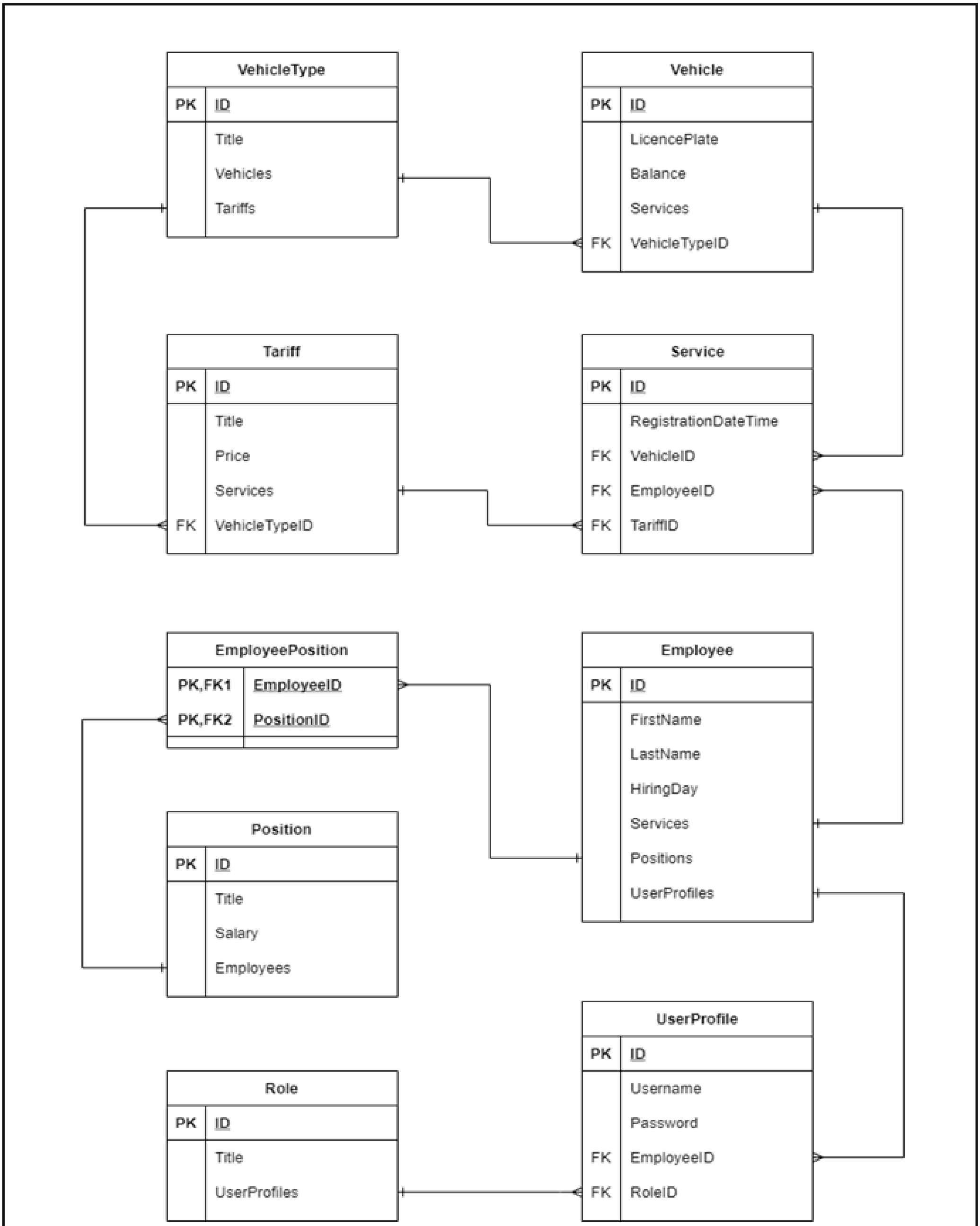


Слайд 13 – Сторінка адміністратора

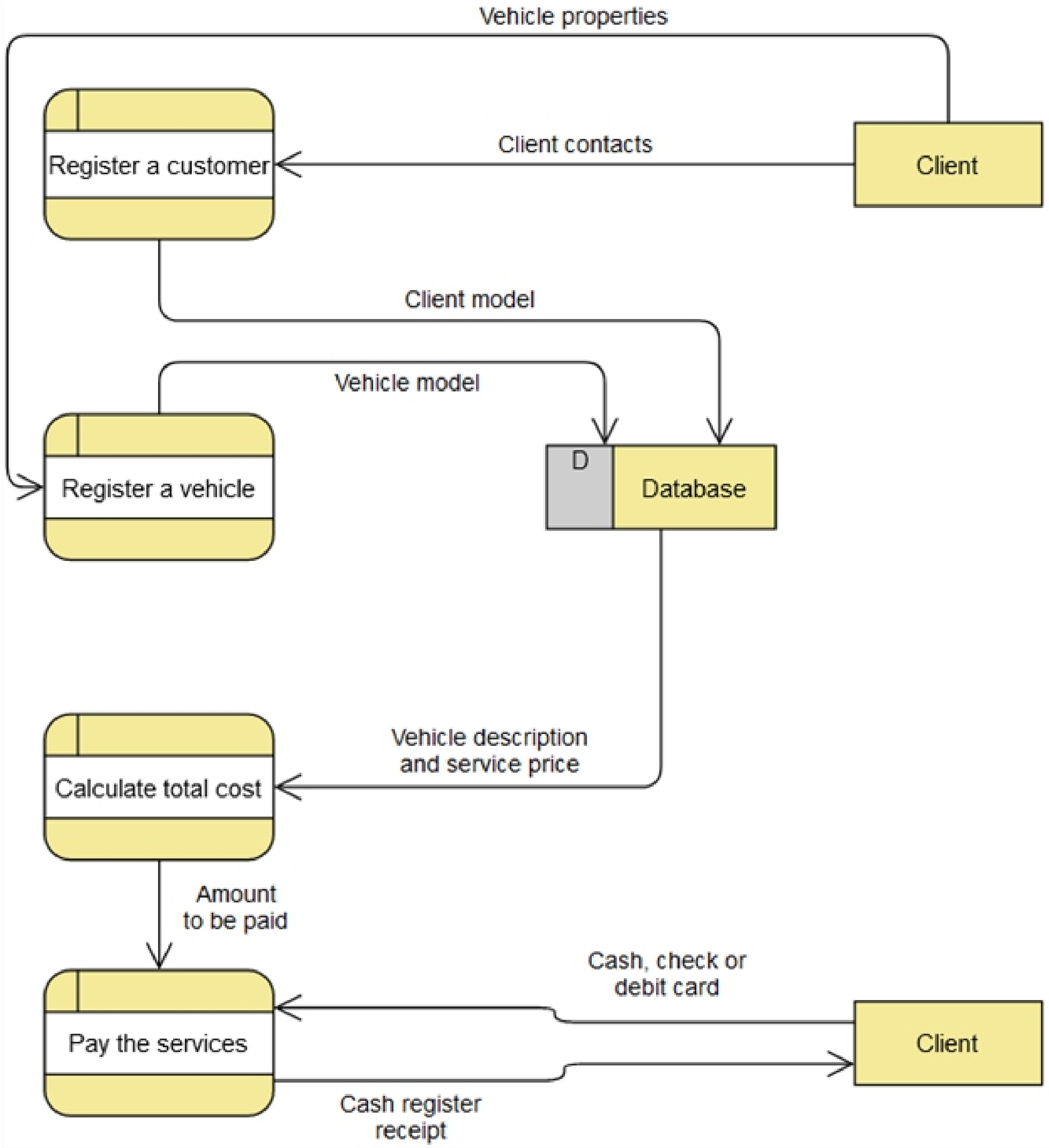


Слайд 14 – Висновки

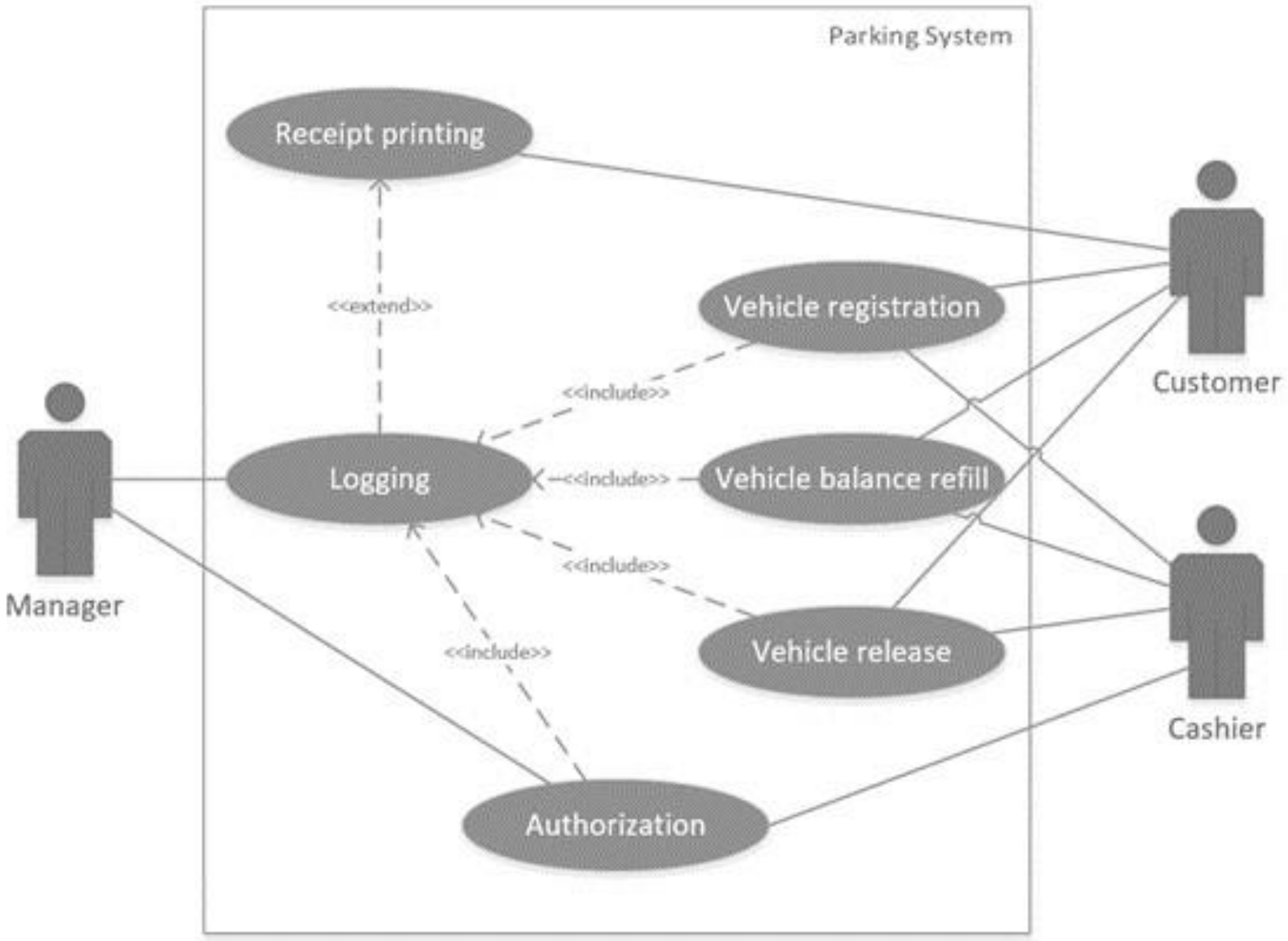
ГРАФІЧНА ЧАСТИНА



					КвРІПЗ.190137.01.13.Е8			
						Літера	Маса	Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата	ER діаграма			
Розробив		Мариняк В.М				Аркуш 1	Аркушів 3	
Керівник		Яшина О.М.						
Консульт.								
Н. Контр		Форкун Ю.В.						
Зав. каф.		Бедратюк Л.П.						



					КвРІПЗ.190137.01.13.Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	DFD-діаграма послуги «Розміщення т/з на території паркінгу»	Літера	Маса / Масштаб
Розробив		Мариняк В.М					
Керівник		Яшина О.М.				Аркуш 2	Аркушів 3
Консульт.							
Н. Контр		Форкун Ю.В.					
Зав. каф.		Бедратюк Л.П.					



					КвРІПЗ.190137.01.13.Е8			
						Літера	Маса	Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата	Діаграма варіантів використання			
Розробив		Мариняк В.М.				Аркуш 3	Аркушів 3	
Керівник		Яшина О.М.						
Консульт.								
Н. Контр.		Форкун Ю.В.						
Зав. каф.		Бедратюк Л.П.						

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Мариняка В.М
Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-19-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

6.6.2023
дата

підпис

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1015480852

Дата перевірки:
07.06.2023 12:18:22 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
07.06.2023 12:19:37 EEST

ID користувача:
100005589

Назва документа: **КвР_Мариняк_2_плагіат**

Кількість сторінок: **92** Кількість слів: **13216** Кількість символів: **105384** Розмір файлу: **3.10 MB** ID файлу: **1015136744**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

18.4%
Схожість

Найбільша схожість: **3.69%** з джерелом з Бібліотеки (ID файлу: **1015073345**)

15.9% Джерела з Інтернету	733	Сторінка 94
7.94% Джерела з Бібліотеки	158	Сторінка 101

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1
Підозріле форматування	16 сторінок

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 6.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 8%**

	Документ		Сумарний збіг по Базі Даних	
ID: 115050 Назва: БКР Програмна система для автоматизованого керування паркінгом транспортних засобів Додано в БД: 2023-06-07 Автора: Мариняк В.М. Керівники: Яшина О.М. к.т.н. доц. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	77476	1190	11502 (15%)	176 (15%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Мариняк Владислав Миколайович

Тема Програмна система для автоматизованого керування паркінгом транспортних засобів

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 ; кількість сторінок записки 101

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено і проаналізовано предметну область, визначено усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих програм на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Розглянуто інструменти для реалізації спроектованих рішень, в результаті чого створено програмне забезпечення. Також було проведено тестування програми, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експуатації.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз сучасних архітектур, розглянуто їх переваги і недоліки та визначено, що система буде відповідати монолітній архітектурі та моделі клієнт-сервер. У третьому розділі підготовлено всі залежності для написання коду та виконано практичну розробку програмних модулів і описано їх особливості, в результаті чого створено програмний продукт. Також у цьому розділі виконано модульне тестування системи та проведено його у відповідності до функціональних вимог, в результаті було підтверджено коректну роботу програми.

4. Позитивні сторони роботи Розроблений програмний продукт демонструє можливості повної автоматизації процесів паркування, включаючи реєстрацію в'їзду і виїзду, розрахунок оплати, контроль доступу та інші функції керування. Це сприяє спрощенню роботи персоналу та забезпечує ефективне функціонування паркінгу. Оптимізація використання паркінгових місць, контроль за заповненістю, збір статистики та інші функції дозволяють досягти оптимального використання простору та запобігти зайнятості місць без потреби.

5. Негативні сторони роботи Складність впровадження системи на існуючих паркінгах, особливо тих, що мають застарілу інфраструктуру або використовують інші програмні рішення, є потенційно негативною стороною роботи. Інтеграція нової системи з уже існуючими може потребувати значних зусиль та ресурсів. Крім того, введення нової технології може створювати потребу у перепідготовці персоналу та збільшенні витрат на навчання. Такі фактори можуть стати причиною затримок у впровадженні та збільшення витрат на проект.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Лисенко Сергій Миколайович, доктор технічних наук, професор кафедри комп'ютерної інженерії та інформаційних систем (КІІС) ХНУ

"6" червня 2023 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом (ами) перевірки текстів на плагіат:

Назва: «Програмна система для автоматизованого керування паркінгом транспортних засобів»

Автор: Мариняк Владислав Миколайович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Яшина Оксана Миколаївна, кандидат технічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних *бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо*, у назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 18,4% і адресується до 891 джерел, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 7 червня 2023 р.

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ЯШИНА