ІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМ. В. ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА ПРОГРАМУВАННЯ ТА МАТЕМАТИКИ

До захисту допускається
Завідувач кафедри __ПМ_____
Лифар В.О.
«____»_____ 20__ р.

МАГІСТЕРСЬКА РОБОТА

НА ТЕМУ:

Інформаційна система цифрового проїзного квитка міського транспорту на основі мікросервісної архетектури

Освітній рівень“Магістр”
Спеціальність 126 “Інформаційні системи та технології”

Науковий керівник роботи:	_____ (підпис)	О.І. Захожай (ініціали, прізвище)
Студент:	_____ (підпис)	Д.О. Савченко (ініціали, прізвище)
Рецензент:	_____ (підпис)	С.О. Митрохін (ініціали, прізвище)
Група:		ІСТ-20дм

Сєвєродонецьк 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет 122 Інформаційних технологій та електроніки

Кафедра Програмування та математики

Освітній рівень Магістр

Спеціальність 126 “Інформаційні системи та технології”

(шифр і назва)

ЗАТВЕРДЖУЮ:

Завідувач кафедри ПМ

_____ В.О.Лифар

«_____» _____20____р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Савченко Денису Олександровичу

(прізвище, ім’я, по батькові)

1. Тема роботи Інформаційна система цифрового проїзного квитка міського транспорту на основі мікросервісної архетектури

керівник проекту (роботи) Захожай Олег Ігорович, д.т.н., доц.

(прізвище, м.’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «30» 11 2021 р. № 182/15.16

2. Строк подання студентом роботи 20.12.2021

3. Вихідні дані до роботи Матеріали науково-дослідної практики, науково-методична література; дані інтернет-мережі.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Валідація електронного проїзного, система покупки електронного проїзного, масштабована архітектура системи, висновки

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) Електронні плакати

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Розробка технічного завдання	8.11.2021-13.11.2021	
2	Аналіз літератури з досліджуваної проблеми	14.11.2021-19.11.2021	
3	Аналіз технічних засобів	20.11.2021-24.11.2021	
4	Аналіз існуючих систем	25.11.2021-30.11.2021	
4	Реалізація інформаційної системи	1.12.2021-11.12.2021	
5	Оформлення пояснювальної записки та презентації	12.12.2021-19.12.2021	

Студент _____ Савченко Д.О.
(підпис) (прізвище та ініціали)

Науковий керівник _____ Захожай О.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Савченко Д.О. Інформаційна система цифрового проїзного квитка міського транспорту на основі мікросервісної архетектури.

Метою роботи є аналіз та проектування системи диджиталізації оплати проїзду у громадському транспортному засобі.

Об’єкт дослідження - методи та інструментальні засоби побудови інформаційних систем обліку цифрових проїздних квітків для міського транспорту.

Робота присвячена складанню концепту системи цифрової оплати проїзду у громадському транспорті. Дана система буде єдиним продуктом, який легко масштабується та розгортається для нових населених пунктів.

Було проведено дослідження методів і засобів обліку цифрових проїздних квитків на міському транспорті та розроблене рішення, що дозволяє побудувати інформаційну систему обліку на основі мікросервісної архітектури.

ABSTRACT

Savchenko DO Information system of digital ticket of public transport on the basis of microservice architecture.

The purpose of the work is to analyze and design a system of digitalization of fare in a public vehicle.

The object of the study are ready-made hardware and software solutions in different cities of Ukraine, to compile a complete picture of what the research system consists of and what business processes take place in it.

The work is devoted to the concept of the system, digital fare payment in public transport. This system will be the only product that is easily scalable and deployable for new settlements.

An analysis was carried out and a solution was schematically presented, on the basis of which it is possible to develop a technical task for further implementation.

ЗМІСТ

ВСТУП

Останні кілька років Україна стрімкими кроками рухається до глобальної диждіталізації повсякденних процесів. Заміна основних документів на цифрові, діджиталіцазія багатьох бюроктатичних процесів.

Процес оплати за проїзд також не стоїть на місці. Якщо раніше, необхідно було купувати квиток, або платити готівкою за проїзд, то зараз, у Києві, у багатьох громадських транспортних засобах діє цифровий проїзний, який дозволяє оплачувати проїзд прямо зі смартфона, або спеціальної NFC картки, якщо у громадянина немає телефону, або на нього неможливо встановити програму цифрового проїзного.

Мета цієї курсової роботи - вивчити систему цифрової оплати за проїзд та запропонувати єдине програмне забезпечення, яке дозволить інтегрувати систему цифрової оплати за проїзд, навіть у найменших населених пунктах України.

Актуальність завдання полягає у тому, щоб у майбутньому перевести якнайбільше населених пунктів України на систему цифрового проїзного. Особливо важливо це зараз, у період пандемії, коли особливо ризиковано контактувати з речами інших людей (у разі з готівкою).

Розробити подібну систему в рамках одного міста – вкрай дорогий проект. Основні витрати полягають саме у створенні програмного забезпечення.

У цій роботі буде розглянуто процес створення єдиного програмного забезпечення, що масштабується, яке дозволить значно знизити підсумкову вартість переходу населеного пункту на систему цифрових проїзних.

1 АНАЛІЗ МЕТОДІВ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ СТВОРЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ОБЛІКУ ЦИФРОВИХ ПРОЇЗДНИХ КВІТКІВ

1.1 Монолітна архітектура

Термін моноліт описував об’єкти, виготовлені з одного великого шматка матеріалу. У розробці програмного забезпечення монолітна архітектура — це коли вся програма має єдину кодову базу і складається з однієї виконуваної програми. Щоб зробити все більш керованим, інженери поділяють програму за бізнесом або технічними характеристиками на менші окремі модулі. На відміну від сервісно-орієнтованої архітектури, модулі спілкуються один з одним безпосередньо за допомогою виклику функцій.

Хоча монолітний додаток може швидко перетворитися на велику миску спагетті-коду, сама архітектура прагне до простоти. Програмісти імпортують функції з різних модулів, і все. Усе виконується в одному процесі програми — без затримок, збоїв, затримок, різних версій модулів, і більшість коду синхронна. Розгортання також зводиться до оновлення програми та перезапуску.

Викликати функцію в пам’яті набагато швидше, ніж здійснювати мережевий виклик і чекати результатів. Дані в оперативній пам’яті доступні для всієї програми, і не потрібні переходи в мережу. У той час як програмі з архітектурою мікросервісів, можливо, доведеться зробити 20 різних викликів API до різних служб, які, у свою чергу, виконують більше викликів API і запитів до бази даних, моноліту потрібно зробити лише пару запитів до бази даних, щоб виконати те саме завдання.

Все знаходиться в одній кодовій базі і збирається як єдине ціле. Таким чином, розробники можуть знайти і зрозуміти всі місця, де використовується

певний фрагмент коду. Під час рефакторингу або видалення невикористаного коду вони можуть бути впевнені, що не пошкодять щось, що, можливо, залежить від цього. Їм також не потрібно мати справу з кількома версіями одного спільного коду, що розгортаються одночасно.

Хоча моноліт можна масштабувати по горизонталі, зазвичай це не так просто. Він може мати деякий стан в RAM або покладатися на ексклюзивний доступ до бази даних. Навіть якщо вони розроблені належним чином, усі модулі працюватимуть на кожному сервері. Не можна просто масштабувати частину програми, яка потребує більше ресурсів.

Коли програма аварійно завершує роботу, все зупиняється. Помилка в окремому модулі може вивести з ладу всю систему. Коли одна частина програми з'їдає всі ресурси, страждають і всі інші її аспекти. Досить важко створити високодоступний, масштабований, відмовостійкий моноліт.

Підтримувати чисту кодову базу, коли проект зростає, стає важко. Розробники завжди прагнуть зробити глобальну змінну, синглтон або зробити частину внутрішніх елементів загальнодоступною. Це занадто легко, коли перед вами є весь код.

З монолітом йде прив’язка до однієї платформи і, можливо, мови програмування. Деякі платформи, такі як .Net і Java, дозволяють майже повний обмін кодами між модулями різними мовами. Непросто впровадити нові технології, які краще підходять для майбутніх проблем.

Коли кілька команд працюють разом над проектом, спільного використання коду стає проблемою. Інженерам часто доводиться вирішувати конфлікти злиття і перевіряти, чи не порушили вони нічого зі своїм кодом у коді, написаному іншими людьми. Розгортання має бути ретельно сплановано, і вся команда повинна чекати, поки вони побачать, як їх код працює у виробництві.

1.2 Мікросервісна архітектура

Програми, створені на основі сервісно-орієнтованої архітектури (SOA), складаються з багатьох слабо пов’язаних служб, які спілкуються один з одним через мережу. Мікросервісна архітектура — це варіант SOA з крихітними сервісами, що використовують полегшені протоколи. Кожен мікросервіс є незалежним і володіє виділеними базами даних та іншими ресурсами.

Можна масштабувати кожну службу окремо відповідно до необхідного використання ресурсів і поточного навантаження. У той час як одна служба може працювати на сотнях екземплярів, інша може працювати в простою лише на одній машині. Найкращі інженери навіть спроектують послуги відповідно до їх запланованого навантаження. Служба звітності, яку використовує лише команда продукту, може використовувати прості реляційні бази даних і запити до них. На відміну від цього, частина програми, яка використовується для користувача, буде використовувати швидкі бази даних No-SQL і деяке кешування.

Завдяки гарному дизайну немає необхідності розгортати всю програму відразу. Кожну службу можна оновлювати окремо, не впливаючи на інші служби, що робить можливим безперервну доставку та скорочує весь цикл розробки та розгортання програми.

Якщо одна із служб не працює, це не впливає на всю програму. Так, функціональні можливості, за які він відповідає, будуть недоступні і можуть вплинути на підключені служби, але більшість розширеного додатка зазвичай продовжує працювати.

На відміну від моноліту, окремі служби можуть використовувати різні технології або навіть мови програмування. Вони навіть можуть працювати на різних операційних системах. Можливості безмежні, доки кожен сервіс може розуміти один одного.

Кожен окремий мікросервіс зазвичай має лише кілька тисяч рядків коду. Таким чином, легко прочитати та зрозуміти все. Налагодження служби нескладне. Модульні тести також є більш простими, оскільки код слабо пов’язаний. Процес адаптації для нових розробників є стислим і займає години, а не дні чи тижні.

Через слабке з’єднання та незалежність сервісів легко працювати з ними паралельно. Інженери можуть тимчасово створювати макети для необхідних API і повертатися до реальних служб, коли вони будуть готові.

Служби досить автономні, щоб повторно використовувати їх навіть у іншій програмі. Наприклад, декілька програм можуть спільно використовувати кошик для покупок, а служби аутентифікації часто дуже схожі.

Додати елемент до інвентарю в monolith було так само просто, як оновити кілька записів бази даних за одну транзакцію. У світі мікросервісів це вже не так просто. Служба інвентаризації повинна перевірити інвентар і, ймовірно, зарезервувати їх. Служба карток покупок повинна додати товар на картку. Служба відповідного продукту має перевіряти наявність відповідних пропозицій, а кілька аналітичних служб мають реагувати на дії користувача.

Під час налагодження однієї взаємодії в програмі мікросервісу розробники повинні запускати декілька служб локально або підключатися до них віддалено. Відстеження даних і зіставлення журналів різних служб, щоб зрозуміти, чому щось сталося, стає досить складним завданням. Інструментів, які допоможуть у цьому, також не так багато.

Кожен може запустити монолітну програму, просто виконавши файл точки входу. На відміну від цього, мікросервіси завжди вимагають певної конфігурації, щоб знати один про одного та знайти правильні служби залежностей. Є масштабованість, відмовостійкість, балансування навантаження тощо. Знадобиться DevOps навіть для помірно складних програм, щоб отримати всі переваги цієї архітектури.

У моноліті більшість коду виконується в одному процесі. У мікросервісних програмах все виконується асинхронно на різних машинах. Ці розподілені системи мають більшу затримку, оскільки їм потрібно набагато більше мережевих переходів для виконання того ж завдання. Більше того, при використанні черги подій, щоб спростити речі, події можуть накопичуватися під великим навантаженням, різко знижуючи продуктивність системи.

З монолітом у мережі доступний лише публічний API. Завдяки мікросервісам багато служб також відкривають приватні кінцеві точки для міжсервісного зв'язку. Їх потрібно ретельно оберігати від зовнішнього світу.

Використання кращих технологій для конкретних завдань — це добре, але це різко підвищує бар'єр для нових розробників. Завдяки різноманітним бібліотекам, фреймворкам і навіть мовам програмування вам буде важко приєднатися до нових розробників або навіть знадобляться окремі незмінні команди для кількох частин програми. Не всі розробники javascript також знають Python або C++ і навпаки. [1]

1.3 Брокери повідомлень

1.3.1 RabbitMQ

При створенні веб-додатків існує багато різних інструментів, які розробники можуть використовувати для оптимізації продуктивності. Однією з важливих груп інструментів є брокери повідомлень, такі як RabbitMQ.

На відміну від стандартних обчислювальних моделей, розподілена обчислювальна модель — це модель, де кілька комп'ютерів працюють в мережі, що дозволяє їм взаємодіяти один з одним, а також спільно використовувати завдання та компоненти. Машини в цій мережі спілкуються шляхом передачі повідомлень, і одним із способів передачі цих комунікацій

між машинами є черги повідомлень. Ці черги використовують так званий брокер повідомлень, щоб швидко й легко керувати попитом між комп'ютерами для максимальної ефективності.

Простіше кажучи, брокер повідомлень «сидить» між машинами в потоці процесу розподіленої обчислювальної системи. Замість того, щоб кожна машина передала повідомлення безпосередньо один одному, повідомлення спочатку надсилаються посереднику повідомлень, наприклад RabbitMQ, який потім впорядковує повідомлення в оптимізовану чергу. Ці повідомлення потім передаються на відповідні машини-отримувачі, коли ці машини будуть готові обробити повідомлення.

У цьому контексті повідомлення може бути командою для обробки замовлення, виконання визначеного завдання, запитів на витяг, зроблених до бази даних, або простого надсилання електронного листа.

У цій мережі машина, яка надсилає вихідне повідомлення, називається «виробником», а машина, яка отримує повідомлення, — «споживачем». Біт посередині — це «брокер», машина, яка об'єднує «виробника» і «споживача». Весь цей процес обміну повідомленнями контролюється протоколом Advanced Message Queuing Protocol (AMQP). AMQP — це набір стандартів, який охоплює весь процес обміну повідомленнями та забезпечує взаємосумісність між системами посередників повідомлень, які сприяє RabbitMQ; на відміну від «безпосередніх» моделей, які надсилають запити безпосередньо з машини на веб-сервер.

Як було зазначено вище, RabbitMQ (Черга повідомлень кролика) є лише одним із прикладів моделі брокера повідомлень. Сервер RabbitMQ знаходиться між машинами і керує чергою. Завдання додаються до черги брокером повідомлень, і споживачі мають підключитися до Rabbit і вказати, що вони готові отримувати та виконувати завдання з певної черги.

Розподілені обчислювальні системи та інші моделі посередників повідомлень неймовірно корисні самі по собі, забезпечуючи численні

переваги в порівнянні з іншими установками без брокерів. Нижче наведено деякі з основних переваг сервера RabbitMQ та інших моделей брокерів повідомлень.

По-перше, RabbitMQ дозволяє кільком різними мовами програмування та платформам працювати разом. Завдяки RabbitMQ виробник може написати повідомлення на C#, але споживач обробить його на Java. Ці дві машини, по суті, намагаються розмовляти один з одним різними мовами, які вони обидві не розуміють, але вони можуть робити це без проблем, спілкуючись через багатомовний сервер посередника RabbitMQ як проксі-сервер. Повідомлення можуть бути написані та отримані, наприклад, на C#, Java, PHP або Perl, але зазвичай брокер передає їх у форматі XML/JSON/plaintext.

Саме по собі відокремлення виробника від його споживачів має багато переваг. Виробнику повідомлення не потрібно нічого знати про споживача, а це означає, що йому не потрібно роздуватися, навчаючись підключатися до нього чи спілкуватися з ним. Виробник просто надсилає повідомлення брокеру повідомлень, у цьому випадку RabbitMQ, а споживач отримує його. Оскільки вони роз’єднані, для виробника не має значення, чи переходить споживач на новий сервер або його замінить інший процес, він може продовжувати безперебійно спілкуватися через сервер RabbitMQ.

Якщо споживач недоступний з будь-якої причини (наприклад, для обслуговування сервера), повідомлення, надіслані від виробника, можна просто утримувати в черзі, доки споживач знову не стане доступним, після чого він зможе швидко обробити їх усі. Це має додаткову перевагу, що дозволяє виробнику продовжувати власну роботу, поки споживач не буде резервним копіюванням і не буде готовий обробляти повідомлення від брокера. Без черги повідомлень виробнику довелося б чекати в простою, поки споживач не буде готовий, а потім відправити всі свої запити за один раз. Отже, замість того, щоб перевантажувати споживача робочим

навантаженням, з яким він не може впоратися, RabbitMQ дозволяє споживачу забрати роботу, коли він буде готовий її обробити.

Аналогічно, якщо є занадто багато повідомлень, які споживач може обробити, поки вони активні, вони можуть просто залишитися в черзі для використання пізніше. Крім того, можна додати більше споживачів для обробки повідомлень, що надходять із черги. Це покращить час обробки та дозволить розробникам побачити, які машини в мережі виконують багато роботи, а коли споживачі не встигають за виробниками. Потім сервер RabbitMQ може бути використаний для збільшення масштабу цих часто використовуваних частин. [2]

1.3.2 Apache Kafka

Програмна платформа з відкритим кодом, розроблена LinkedIn для обробки даних у реальному часі, називається Kafka. Він публікує і підписується на потік записів, а також використовується для відмовостійкого зберігання. Програми призначені для обробки записів часу та використання. Розділи журналів різних серверів реплікуються в Kafka. Він зберігає, зчитує та аналізує потокові дані, де розробники та користувачі вносять оновлення коду. Kafka використовується для обміну повідомленнями, відстеження активності веб-сайту, агрегації журналів і журналів фіксації. Kafka можна використовувати як базу даних, але вона не має моделі даних чи індексів.

Його зростання стрімко. Йому віддають перевагу більше ніж одна третина списку Fortune 500 по всьому світу. Цей розподіл поділяють компанії туристичного бізнесу, телекомунікаційні гіганти, банки та деякі інші. LinkedIn, Microsoft і Netflix обробляють повідомлення з чотирма комами в день за допомогою Kafka (майже дорівнює 1 000 000 000 000). Він використовується для потоків даних у реальному часі, збору великих даних або аналізу в реальному часі (або обох). Kafka використовується з мікросервісами в пам’яті для забезпечення довговічності, і його можна

використовувати для передачі подій у CEP (складні системи потокової передачі подій) і системи автоматизації в стилі IoT/IFTTT.

Керуючись простотою, це був би правильний спосіб визначити продуктивність. Легко зрозуміти, як Kafka працює з такою легкістю з його налаштуванням та використанням. Ця підвищена продуктивність у поведінці присвячена його стабільності, забезпеченню надійної довговічності, з його гнучкою вбудованою можливістю публікації, підписки або обслуговування черги. Це важливо мати, якщо потрібно мати справу з N – кількістю груп клієнтів, якщо вам потрібно показати надійну реплікацію на ринку, щоб забезпечити своїм клієнтам послідовний підхід (тобто розділ теми Kafka). Вирішальна поведінка Kafka, яка виділяє його серед конкурентів, — це його сумісність із системами з потоками даних — його процес дає змогу цим системам об’єднувати, трансформувати та завантажувати інші сховища для зручності роботи. Як все працює для Кафки на рівні ОС:

- Він покладається на ядра ОС для більш швидкого переміщення даних і працює за принципом нульової копії.
- Він дозволяє групувати записи даних у фрагменти, які можна побачити з файлової системи (він же журнал тем Kafka) для споживачів.
- Можливість пакетної обробки даних забезпечує ефективне стиснення даних із зменшенням затримки введення-виводу.
- Він має можливість горизонтального масштабування за допомогою шардинга. Він може розділити журнал заголовків на сотні розділів до тисяч. Це дозволяє йому легко справлятися з величезним робочим навантаженням.

З метою відстеження даних та керування ними відповідно до потреб бізнесу, Kafka є перевагою в усьому світі. Це дає можливість передавати дані в режимі реального часу за допомогою аналітики в режимі реального часу.

Він швидкий, масштабований і довговічний і розроблений як відмовостійкість. В Інтернеті є кілька варіантів використання, де можна

зрозуміти, чому JMS, RabbitMQ і AMQP навіть не вважаються підходящими, оскільки потрібно працювати з величезними обсягами та швидко реагувати.

Він має високу пропускну здатність, надійне налаштування з характеристиками реплікації, що робить його кращим вибором для роботи з датчиками Інтернету речей. Сумісність є ще однією причиною для його використання і зробила його прийнятним у всьому світі. Його можна легко налаштувати для роботи з програмою, наведеною нижче. Ця комбінація є життєво важливою для багатьох компаній для розвитку бізнесу та виживання (оскільки це економить час і гроші).

Висока пропускна здатність. Він може легко обробляти великий обсяг даних, коли генерування з високою швидкістю є винятковою перевагою на користь Кафки. У цій програмі бракує величезного обладнання для підтримки пропускної здатності повідомлень із частотою тисяч повідомлень в секунду.

Низька затримка. Низька затримка для обробки цього великого обсягу генерації повідомлень.

Відмовостійкість. Ця функція зручна; він має властиву можливість обмежуватися вузлом, вбудованим у кластер.

Міцний. Він дуже довговічний у своїй роботі, тому багато МНК вважають за краще використовувати Kafka. Говорячи про довговічність операцій, повідомлення не можуть бути втрачені в довгостроковій перспективі.

Наразі Kafka став де-факто стандартом для аналізу даних у реальному часі з найвищою точністю в мікросекундах. Кілька великих компаній щодня використовують дані; для цього їм потрібні професіонали, щоб використовувати ці величезні масиви даних. З Кафкою можна бути впевненим, що керуватимете своєю кар’єрою в аналітиці BigData. [3]

Kafka — це хороша система зберігання записів/повідомлень. Kafka діє як високошвидкісна файлова система для зберігання та реплікації журналів

фіксації. Ці характеристики роблять Kafka корисним для всіх видів додатків. Записи, записані на теми Kafka, зберігаються на диску та реплікуються на інші сервери для відмовостійкості. Оскільки сучасні накопичувачі швидкі та досить великі, це добре підходить і дуже корисно. Продюсери Kafka можуть чекати підтвердження, тому повідомлення є довговічними, оскільки виробник записує не завершено, поки повідомлення не повториться. Структура диска Кафки добре масштабується. Сучасні дискові накопичувачі мають дуже високу пропускну здатність при записі великими потоковими пакетами. Крім того, клієнти/споживачі Kafka можуть контролювати позицію читання (зміщення), що дозволяє використовувати такі випадки, як повторне відтворення журналу, якщо була критична помилка (виправити помилку та повторне відтворення).

Зв’язок Kafka від клієнтів і серверів використовує дротовий протокол через TCP, який перевірено і задокументовано. Kafka обіцяє підтримувати зворотну сумісність зі старими клієнтами, і багато мов підтримуються. Є клієнти на C#, Java, C, Python, Ruby та багатьох інших мовах. Екосистема Kafka також забезпечує REST-проксі, що дозволяє легко інтегрувати через HTTP і JSON, що робить інтеграцію ще простішою. Kafka також підтримує схеми Avro через реєстр Confluent Schema Registry для Kafka. Avro і реєстр схем дозволяють створювати і читати клієнти складних записів багатьма мовами програмування, а також дозволяють розвивати записи. Кафка справді поліглот.

Kafka найчастіше використовується для потокової передачі даних в інші системи в режимі реального часу. Kafka — це середній рівень для роз’єднання конвеєрів даних у реальному часі. Ядро Kafka не підходить для прямих обчислень, таких як агрегації даних або CEP. Kafka Streaming, який є частиною екосистеми Kafka, надає можливість проводити аналітику в режимі реального часу. Kafka можна використовувати для живлення систем швидкої смуги (систем реального часу та операційних даних), таких як

Storm, Flink, Spark Streaming, а також ваші послуги та системи CEP. Kafka також використовується для потокової передачі даних для пакетного аналізу даних. Кафка годує Hadoop. Він передає дані на платформу BigData або в RDBMS, Cassandra, Spark або навіть S3 для подальшого аналізу даних. Ці сховища даних часто підтримують аналіз даних, звітність, аналіз даних, аудит відповідності та резервне копіювання.

Kafka в значній мірі покладається на ядро ОС для швидкого переміщення даних. Він спирається на принципи Zero Copy . Kafka дозволяє групувати записи даних у фрагменти. Ці пакети даних можна побачити від кінця до кінця від виробника до файлової системи (Kafka Topic Log) до споживача. Пакетування дозволяє ефективніше стискати дані та зменшує затримку введення-виводу. Kafka записує в журнал незмінних фіксацій на диск послідовно; таким чином, уникає випадкового доступу до диска, повільного пошуку диска. Kafka забезпечує горизонтальне масштабування за допомогою сегментування. Він розбиває журнал теми на сотні потенційно тисяч розділів на тисячі серверів. Цей шардінг дозволяє Kafka впоратися з великим навантаженням.

Kafka в значній мірі покладається на ядро Кафка має операційну простоту. Kafka потрібно налаштувати та використовувати, і легко зрозуміти, як працює Kafka. Однак головна причина, чому Кафка дуже популярний, — це її чудова продуктивність. Він також має інші характеристики, але також мають інші системи обміну повідомленнями. Kafka має чудову продуктивність і стабільність, забезпечує надійну довговічність, має гнучку передплату/чергу на публікацію, яка добре масштабується з N-кількістю груп споживачів, має надійну реплікацію, надає виробникам настроювані гарантії узгодженості та забезпечує збереження порядку на рівень фрагмента (розділ теми Kafka). Крім того, Kafka добре працює з системами, які мають потоки даних для обробки, і дозволяє цим системам об’єднувати, трансформувати та завантажувати в інші сховища.

Але жодна з цих характеристик не мала б значення, якби Кафка був повільним. [4]

1.4 MongoDB та PostgreSQL

1.4.1 MongoDB

MongoDB — це документно-орієнтована база даних NoSQL . Від реляційних баз даних він відрізняється своєю гнучкістю та продуктивністю.

MongoDB — це документно-орієнтована база даних NoSQL, яка з’явилася в середині 2000-х років. Використовується для зберігання великих обсягів даних. На відміну від традиційної реляційної бази даних SQL , MongoDB не покладається на таблиці та стовпці. Дані зберігаються як колекції та документи.

Документи – це пари значення/ключ, які служать основною одиницею даних. Колекції містять набори документів і функцій. Вони є еквівалентом таблиць у класичних реляційних базах даних.

Кожна база даних MongoDB містить колекції, самі містять документи. Кожен документ відрізняється і може мати різну кількість полів. Розмір і зміст кожного документа також відрізняються.

Структура документа відповідає розробникам шляху будувати свої класи і об'єкти на мові програмування, якиа використовується. Загалом, класи не є рядками і стовпцями, а мають чітку структуру, що складається з пар значення/ключ. Документи не мають попередньо визначеної схеми, і поля можна додавати за бажанням. Модель даних, доступна в MongoDB, полегшує представлення ієрархічних відносин або інших складних структур.

Ще однією важливою особливістю MongoDB є еластичність його середовища . Багато компаній мають кластери з понад 100 вузлів для баз даних, що містять мільйони документів.

Архітектура MongoDB базується на кількох основних компонентах. По-перше, «_id» є обов'язковим полем для кожного документа. Він представляє унікальне значення і може розглядатися як основний ключ документа для його ідентифікації в колекції.

Документ є еквівалентом запису в традиційній базі даних . Він складається з полів імені та значення. Кожне поле є асоціацією між іменем і значенням і подібне до стовпця в реляційній базі даних.

Колекція — це група документів MongoDB і відповідає таблиці, створеної за допомогою будь-якої іншої RDMS, як-от Oracle або MS SQL, у реляційній базі даних. Він не має попередньо визначеної структури.

База даних — це контейнер колекцій , так само як RDMS — це контейнер таблиць для реляційних баз даних. Кожен з них має власний набір файлів у файловій системі. Сервер MongoDB може зберігати кілька баз даних.

Нарешті, JSON (JavaScript Object Notation) — це простий текстовий формат для вираження структурованих даних. Він підтримується багатьма мовами програмування.

MongoDB має кілька основних переваг. По-перше, ця документоорієнтована база даних NoSQL дуже гнучка і адаптована до конкретних випадків використання підприємства.

Спеціальні запити дозволяють знайти конкретні поля в документах. Також можна створювати індекси для підвищення ефективності пошуку. Будь-яке поле можна індексувати.

Ще однією перевагою є можливість створювати «набори реплік», що складаються з двох або більше екземплярів MongoDB. Кожен учасник може діяти як вторинна або основна репліка в будь-який час.

Основною реплікою є головний сервер, який взаємодіє з клієнтом і виконує всі операції читання та запису. Вторинні репліки зберігають копію даних. Таким чином, у разі відмови основної репліки перемикання на

вторинну здійснюється автоматично. Ця система гарантує високу доступність.

Нарешті, концепція шардинга дозволяє горизонтальне масштабування шляхом розподілу даних між кількома екземплярами MongoDB. База даних може працювати на кількох серверах, і це дозволяє балансувати навантаження або дублювати дані, щоб підтримувати працездатність системи в разі збою обладнання.

Через ці численні переваги MongoDB зараз є широко використовуваним інструментом у сфері розробки даних. Це необхідне рішення для розробників даних.

Існує кілька основних відмінностей між MongoDB і RDBMS (система управління реляційною базою даних). Як згадувалося раніше, дані зберігаються не в таблицях, а в колекціях документів. Ці документи замінюють рядки СУБД. Вони містять поля пар значення/ключ, які самі замінюють стовпці.

Крім того, цілісність даних не є обмеженням для MongoDB. Дані не потрібно «нормалізувати» перед використанням, як у СУБД. Це справжня перевага, оскільки обмеження нормалізації може погіршити продуктивність у міру зростання бази даних.

На відміну від баз даних SQL, MongoDB не передбачає жодних обмежень щодо структури документа. Дані не мають заздалегідь продуманої схеми, і саме ця гнучкість робить MongoDB настільки потужною та ефективною.

Моделювання даних і структура документа повинні відповідати лише потребам користувача. Важливо враховувати потреби програми, а отже, які дані та типи даних будуть потрібні.

Якщо очікувати багато запитів, важливо використовувати індекси в моделі даних, щоб підвищити ефективність запитів. Нарешті, якщо дані часто додаються, оновлюються та видаляються, для підвищення загальної

ефективності середовища слід використовувати індекси та систему розподілу.[5]

1.4.2 PostgreSQL

PostgreSQL є, мабуть, найдосконалішою базою даних на ринку реляційних баз даних з відкритим кодом. Вперше він був випущений в 1989 році, і з тих пір було багато покращень. За даними db-engines, це четверта найбільш використовувана база даних на момент написання.

Фізична структура PostgreSQL дуже проста. Він складається із спільної пам’яті та кількох фонових процесів і файлів даних.

Спільна пам'ять відноситься до пам'яті, зарезервованої для кешування бази даних і кешування журналу транзакцій. Найважливішими елементами спільної пам'яті є загальний буфер і буфери WAL.

Буфер WAL — це буфер, який тимчасово зберігає зміни в базі даних. Вміст, що зберігається в буфері WAL, записується у файл WAL у заздалегідь визначений момент часу. З точки зору резервного копіювання та відновлення, буфери WAL і файли WAL дуже важливі.

PostgreSQL має чотири типи процесів.

- Postmaster (Daemon) Process
- Фоновий процес
- Бекенд процес
- Клієнтський процес

Процес Postmaster — це перший процес, який запускається під час запуску PostgreSQL. Під час запуску виконує відновлення, ініціалізує спільну пам’ять і запускає фонові процеси. Він також створює серверний процес, коли є запит на підключення від клієнтського процесу.

Максимальна кількість серверних процесів встановлюється параметром max_connections, а значення за замовчуванням — 100. Бекенд-

процес виконує запит запиту процесу користувача, а потім передає результат. Деякі структури пам'яті необхідні для виконання запиту, яке називається локальною пам'яттю.

Клієнтський процес відноситься до фонового процесу, який призначається для кожного з'єднання бекенд-користувача. Зазвичай процес postmaster розділяє дочірній процес, призначений для обслуговування з'єднання користувача.

Елементи, пов’язані з базою даних:

- PostgreSQL складається з кількох баз даних. Це називається кластером бази даних.
- Коли виконується initdb (), створюються бази даних template0 , template1 і postgres.
- Бази даних template0 і template1 є шаблонними базами даних для створення баз даних користувачів і містять таблиці системного каталогу.
- Список таблиць у базах даних template0 і template1 однаковий відразу після initdb (). Проте база даних template1 може створювати об’єкти, які потрібні користувачеві.
- База даних користувачів створюється шляхом клонування бази даних template1.
- Елементи, пов’язані з табличним простором:
- Табличні простори pg_default і pg_global створюються відразу після initdb().
- Якщо не пропишеться табличний простір під час створення таблиці, він зберігається в табличному просторі pg_dafault.
- Таблиці, керовані на рівні кластера бази даних, зберігаються в табличному просторі pg_global.

- Фізичним розташуванням табличного простору pg_default є $PGDATA\base.
- Фізичним розташуванням табличного простору pg_global є $PGDATA\global.
- Один табличний простір може використовуватися кількома базами даних. У цей час у каталозі табличного простору створюється специфічний для бази даних підкаталог.
- Створення табличного простору користувача створює символічне посилання на табличний простір користувача в каталозі $PGDATA\tblspc.
- Предмети, пов’язані з таблицею:
- У кожній таблиці є три файли.
- Одним з них є файл для зберігання даних таблиці. Ім'я файлу є OID таблиці.
- Один із них - це файл для керування вільним простором таблиці. Ім'я файлу OID_fsm.
- Один - це файл для керування видимістю блоку таблиці. Ім'я файлу OID_vm.
- Індекс не має файлу _vm. Тобто OID і OID_fsm складаються з двох файлів. [6]

Роль PostgreSQL на цьому ринку - це система управління постоб'єктними реляційними базами даних (PORDBMS) з відкритим вихідним кодом. PostgreSQL доступний на ринку більше 25 років. За цей час він перетворився на зрілу пропозицію послуг передачі даних. Ці сервіси створюють тисячі розробників по всьому світу. На даний момент достатньо сказати, що ви тут працюєте не на порожньому місці.

Як і після об'єктного відношення за годинниковим часом. Це стосується розширених функцій зберігання та інтерпретації PostgreSQL,

таких як підтримка JSON і XML, альтернативних механізмів зберігання даних, моделей реплікації та інструментів керування підприємством. Протягом останнього десятиліття глобальна спільнота розробників PostgreSQL зосередилася на наборі інструментів, що оточують PostgreSQL, щоб зробити його екосистемою баз даних світового класу.

АБО означає об'єктно-реляційний. Очевидним посиланням є той факт, що сутності в базі даних (відношення — таблиці, уявлення, функції тощо) пов’язані один з одним за допомогою посилань на дані, які залежать від дизайну. На додаток до цього, самі об’єкти можуть використовуватися як базові класи в сенсі об’єктно-орієнтованого програмування. Тобто таблиця може бути визначена за допомогою визначення іншої таблиці або функція може прийняти визначення таблиці як вхідні дані та надати інше визначення таблиці як вихід. Ця досить унікальна реалізація передбачає використання вбудованих типів даних як основи для ваших власних типів даних, а також визначень обмежень.

DB означає базу даних. Це ядро системи, яке керує рівнем зберігання та пошуку даних. Це написано на C, і це сліпуче швидко.

MS розшифровується як система управління. Служба PostgreSQL є дещо невірною назвою. Насправді це кілька служб, які працюють разом для виконання зберігання, пошуку, керування користувачами, кешування, тимчасового зберігання та кількох інших завдань. Він може розростатися до кількох сотень або навіть тисяч процесів Linux, які потребують координації центральної служби. Сучасне підприємство зберігає дані в багатьох різних системах і в багатьох різних форматах. Ці системи включають сховища даних, звітність, оперативне зберігання даних, системи з єдиним джерелом істини, системи навантаження трансформації екстракту (ETL), системи підтримки додатків та багато іншого.

Сучасне підприємство зберігає дані в багатьох різних системах і в багатьох різних форматах. Ці системи включають сховища даних, звітність,

оперативне зберігання даних, системи з єдиним джерелом істини, системи навантаження трансформації екстракту (ETL), системи підтримки додатків та багато іншого.

PostgreSQL — це модель вертикальної масштабованості. Одночасно може бути тільки одна система, яка отримує зміни до даних. Це обмеження визначає максимальну кількість транзакцій для програми, яку дозволить PostgreSQL. [7]

1.4.3 MongoDB vs PostgreSQL

MongoDB є провідною базою даних документів . Він побудований на розподіленій архітектурі з масштабуванням і став комплексною хмарною платформою для керування та доставки даних до програм. MongoDB обробляє транзакційні, операційні та аналітичні навантаження в масштабі. Якщо турбує час виходу на ринок, продуктивність розробників, підтримка DevOps і гнучких методологій, а також створення матеріалів, які масштабуються без операційної гімнастики, MongoDB — це шлях.

PostgreSQL — це надійна база даних SQL корпоративного рівня з відкритим кодом, яка розширює свої можливості протягом 30 років. Все, що ви коли-небудь хотіли б від реляційної бази даних, є в PostgreSQL, який спирається на архітектуру масштабування. Якщо турбує сумісність, обслуговування тисяч запитів із сотень таблиць, використання наявних навичок SQL та досягнення межі SQL, PostgreSQL виконає чудову роботу.

Обидві бази даних чудові. Якщо необхідно розподілену базу даних для сучасних транзакційних та аналітичних додатків, які працюють зі швидко мінливими багатоструктурованими даними, то MongoDB — це шлях. Якщо база даних SQL відповідає потребам, то Postgres — чудовий вибір. Правильна відповідь для потреб, звичайно, заснована на тому, що намагаєтеся зробити. Але часто на початку проекту розробки керівники проекту часто добре розуміють варіанти використання, але насправді не

мають чіткості щодо конкретних функцій програми, які знадобляться їхньому бізнесу та користувачам. Вони повинні зробити ставку на те, що найкраще підходить.

Будь-яке серйозне обговорення баз даних буде включати принаймні мимохідне посилання на теорему CAP. Теорема CAP стверджує, що мережева система загальних даних може мати щонайбільше дві з наступних трьох гарантій: узгодженість (у будь-який момент кожен вузол бачить одні й ті самі дані, а дані оновлені) доступність (кожний запит правильний сповіщається кожним вузлом), допуск на розділ (система функціонує незалежно від розділення мережі). Ця теорема також використовується для вільної класифікації баз даних на три сімейства: CA, CP і AP. Для представлення систем баз даних, які відповідають двом із цих критеріїв. Системи CA явно неможливі, оскільки введення навіть одного мережевого клієнта робить систему розділеною. Будь-який постачальник, який продає систему ЦС, відверто бреше. Системи CP будуть, за наявності перегородки, вибрати обслуговувати лише узгоджені дані. Якщо узгодженість даних не може бути забезпечена, вони відмовляться обслуговувати запит. Системи AP, за наявності розділу, продовжуватимуть відповідати, але не дадуть гарантій узгодженості відповіді.

Транзакція бази даних — це представлення окремої одиниці роботи з базою даних, яка має бути оброблена як така.

ACID означає: atomic (Усе в транзакції є частиною одного «атома». Або все вдається, або все невдало.) послідовний (як початковий, так і кінцевий стан транзакції є дійсним і не порушує жодних правил бази даних) ізольований(Транзакції ізольовані одна від одної і не можуть заважати одна одній) довговічні (Усі завершені транзакції зберігаються постійно). Узгодженість у CAP насправді є набагато більшою гарантією, і її зазвичай називають лінеаризувальністю.

Вікіпедія визначає нормалізацію як «процес організації атрибутів і таблиць реляційної бази даних для мінімізації надмірності даних». Існує багато стратегій нормалізації, але база даних зазвичай вважається нормалізованою, якщо вона знаходиться в третій нормальній формі (3NF). Це означає наступне:

1. Кожен атрибут містить лише атомні значення. Це явно виключає складні структури JSON або масиви, що зберігаються на листовому вузлі.

2. Жодні дані не представлені надмірно на основі будь-яких неунікальних підмножин. Іншими словами, для кожного унікального набору записів (це називається ключем-кандидатом) жоден інший атрибут не залежить від будь-якої підмножини ключа-кандидата.

3. Жодні дані не залежать ні від чого, крім ключа.

Система управління реляційною базою даних (RDBMS) — це база даних, заснована на реляційній моделі. Він призначений для зосередження на зв’язках між даними і зберігає дані в таблицях рядків стовпців.

Postgres — це СУБД з транзакціями, сумісними з ACID, з повною можливістю серіалізації транзакцій, якщо рівень транзакції встановлений на Serializable.

MongoDB — це звір із мультисистемою зберігання даних. Механізм зберігання за замовчуванням — MMAP, хоча WiredTiger — нещодавно випущений механізм зберігання, який нібито усуває деякі з більш системних недоліків. Однак, як свідчить велика кількість помилок, пов’язаних як із втратою даних, так і з витоком пам’яті, він явно ще не готовий до прайм-тайму. Таким чином, хоча вдосконалення, передбачені WiredTiger, будуть викликані та проаналізовані, де це можливо, претендентом у лівому куті є MMAP. MongoDB — це СУБД без схем, орієнтована на документи з JSON-подібними об’єктами, що формують модель. Він не підтримує транзакції з коробки. Це не точка доступу, тому що це система з одним провідним, і все зчитується до основного вузла (хоча це можна покращити за допомогою

наборів реплік і автоматичного перемикання збоїв, за допомогою якого новий основний вибирається на мережевому розділі).

Транзакції та довговічність у Postgres

Postgres не вимагає блокування читання (за винятком випадків, коли встановлено рівень транзакції Serializable), оскільки кожна транзакція має знімок бази даних. Тому непослідовне читання (також відоме як брудне читання) неможливе. Postgres має 3 рівні ізоляції транзакцій.

Перше - це Read committed. Це значення за замовчуванням. Це означає, що кожен запит у транзакції бачить лише дані, які вже були передані в базу даних на момент початку цього запиту. Жодні зміни, внесені під час виконання запиту, не відображатимуться, однак одночасні зміни, записані під час між запитами іншими транзакціями , потраплять в транзакцію.

Другий є Repeatable Read. Цей рівень транзакції гарантує, що жодні зміни, внесені паралельними транзакціями під час виконання транзакції, не будуть видимі. Фактично зміни транзакцій застосовуються до моментального знімка незалежно від інших транзакцій. Якщо транзакція зазнає невдачі через іншу транзакцію, яка має змінені дані, транзакція завершиться невдачею з винятком паралельності, і програмі, доведеться повторити транзакцію.

Фінал це Serializable. Цей рівень транзакції гарантує, що будь-які транзакції, що виконуються одночасно, матимуть точно такий же ефект, як якщо б вони виконувались одна за одною. Хоча на перший погляд це може здатися дуже схожим на те, Repeatable Read тут є чудове пояснення різниці.

Коротким (практичним) прикладом є банк, який дозволяє перерахувати будь-який з рахунків, якщо кумулятивна сума на всіх рахунках перевищує мінус. Зловмисник може спробувати використати Repeatable Read, одночасно знімаючи великі суми з усіх рахунків. Оскільки Repeatable Read

отримує моментальний знімок, кожна окрема транзакція буде успішною, що призведе до великих чистих збитків для банку. Тут варто звернути увагу, що, хоча було зроблено багато роботи для оптимізації вищих рівнів транзакцій, все ще правда, що вищі рівні транзакцій супроводжуються деякими накладними витратами на продуктивність. Варто чи ні вища узгодженість вартості продуктивності, це те, що слід оцінювати в кожному конкретному випадку, використовуючи контрольні показники ваших даних та операцій.

Транзакції та довговічність у MongoDb

MongoDB не підтримує транзакції з коробки. Однак це дозволяє конфігурувати його Write Concern, що, хоча й не пов’язане з транзакціями, все ж говорить про довговічність операцій запису. За замовчуванням є Acknowledged, що гарантує, що операція запису досягла бази даних, але не гарантує, що дані дійсно були записані на диск. Інші варіанти:

1. Journaled: запит на запис був записаний у журнал MongoDB (черга операцій, яка ще не зберігалася на власному диску).
2. Majority: Запис поширився на більшість вузлів і був визнаний ними.

Хоча Majority забезпечує узгодженість за відсутності мережевих розділів, розділення мережі може призвести до суперечливих даних та/або втрати даних навіть у межах одного документа. Правда також, що MongoDB дійсно надає $isolated оператор, який, забезпечуючи узгодженість шляхом блокування запису, також запобігає будь-якому шардінгу і фактично не гарантує атомарність, оскільки помилка під час операції запису не відкочує всю «транзакцію». MongoDB також дає вільні рекомендації щодо впровадження двофазної фіксації.

Денормалізовані дані в Postgres

Postgres підтримує 4 типи стовпців для зберігання денормализованих даних.

- hstore: це для зберігання пар ключ-значення. Здебільшого він доступний лише для застарілого використання.
- json: тут зберігаються json blob, перетворені в рядок. Він виконує деякі перевірки, щоб забезпечити добре сформований json, і надає оператори зручності для доступу, але це все. Насправді він не забезпечує індексування. Це також значною мірою лише для застарілого використання.
- jsonb: Це зберігає json як двійковий і відображає його як json, а не як одне текстове значення, і дозволяє індексувати в довільні атрибути для швидкого пошуку. Однак для запису все ще потрібно повне оновлення, хоча в 9.5 з’являться функції, які дозволять легше оновлювати вкладені шляхи.
- array: Тут зберігається масив іншого типу даних (текст, число, будь-що).

Денормалізовані дані в Mongo

Mongo оптимізовано для зберігання Json. Mongo зберігає свої дані у двійковому форматі під назвою BSON, який є (приблизно) просто двійковим представленням надмножини JSON. Причиною грубого квантора є відсутність числового типу, а причиною надмножини є підтримка прямих двійкових даних. [8]

1.5 Система моніторингу – Zabbix

Zabbix – це інструмент моніторингу, розроблений латвійським Zabbix SIA. Він використовується в багатьох країнах і в різних галузях промисловості. У Японії також діють Zabbix Japan LLC та японська спільнота Zabbix. Використання цієї системи моніторингу значно поширилося в цій країні.

Zabbix має такі функції:

- Він розроблений та надається як програмне забезпечення з відкритим кодом. Він пропонує кілька методів моніторингу.
    - Зовнішній безагентний моніторинг, наприклад моніторинг PING, SNMP, TCP і SSH
    - Внутрішній моніторинг агентів, що підтримує різноманітні ОС
    - Збір розрахункових та агрегованих даних
    - Моніторинг віртуальних машин
- Автоматичний моніторинг
    - Автоматичне додавання та видалення елементів моніторингу
    - Моніторинг віртуальних машин і контейнерів
    - Автоматичне додавання моніторингу за допомогою агента моніторингу
- Дуже гнучка конфігурація визначення виявлення проблеми
    - Збір матриці, автоматичне визначення статусу проблеми
    - Встановлення рівнів серйозності
    - Прогноз тенденції
- Моніторинг кількох хостів за допомогою інформаційної панелі
    - Розширений відображення графіка
- Розширені дії під час проблеми
    - Ескалація повідомлень
    - Автоматичне відновлення при проблемах

- Розподілений моніторинг
    - Моніторинг даних кількох тисяч елементів моніторингу
    - Брандмауер і моніторинг за межами DMZ
- Безпечний моніторинг
    - Шифрування шляху передачі даних моніторингу
    - Розділення контролю привілеїв користувачів
    - Аутентифікація користувача за допомогою OpenLDAP і ActiveDirectory

Zabbix складається з таких компонентів:

- Сервер Zabbix. Надає центральні функції моніторингу Zabbix. Виконує моніторинг. Зберігає конфігурацію моніторингу та дані моніторингу в базах даних.
- Веб-сервер Zabbix Веб - інтерфейс для налаштування та відображення даних моніторингу Zabbix.
- Проксі-сервери Zabbix. Сервери поширюються в місцях, не прозорих у мережі. Здійснювати моніторинг замість сервера Zabbix та обмінюватися конфігурацією моніторингу та даними моніторингу із сервером Zabbix.
- Агент Zabbix, який діє на відстежувану мету. Відправляє дані моніторингу цілі на сервер Zabbix.

Сервер Zabbix і веб-сервер Zabbix можуть бути розгорнуті в одному місці. Однак, оскільки їм просто потрібно використовувати ту саму БД, не має значення, чи використовується інший сервер Zabbix. Проксі-сервер Zabbix використовується для моніторингу через брандмауер. Він також використовується для розподілу навантаження моніторингу сервера Zabbix. Це необов'язковий компонент.

Моніторинг комп’ютерної діяльності складається із зовнішнього моніторингу цілей і внутрішнього моніторингу цілей. Зовнішній моніторинг

означає послуги моніторингу ззовні. Це схоже на тестування чорного ящика, де контрольована ціль розглядається як скринька, вміст якої неможливо побачити. Моніторинг знаходиться поблизу поля зору користувача. Під час моніторингу веб-служби елементи моніторингу включають те, чи може користувач отримати доступ до Інтернету чи ввійти, а також чи відображаються веб-сторінки, як очікувалося. З іншого боку, внутрішній моніторинг можна назвати тестуванням білої скриньки. Ціль моніторингу розглядається як поле, вміст якого є видимим. Під час моніторингу веб-служби елементи моніторингу включають, чи працюють необхідні служби, чи працює мережа, чи велике навантаження на центральний процесор, пам’ять, сховище та мережу, а також чи був несанкціонований доступ ззовні. Для зовнішнього моніторингу Zabbix може контролювати не тільки ICMP (моніторинг ping), а й статус входу на основі веб-сценарію. Завдяки внутрішньому моніторингу, який виконується за допомогою програми Zabbix Agent, яка запускається всередині машини, також можна відстежувати стан служб і процесів, а також завантаження машини. Для мережевого обладнання, на якому не можна запустити Zabbix Agent, можна контролювати обладнання за допомогою SNMP (простого протоколу керування мережею). Внутрішній моніторинг і зовнішній моніторинг є однаково важливими та важливими для безперервності обслуговування. Zabbix може виконувати обидва типи моніторингу. Це інтегрований інструмент моніторингу, здатний керувати історією та конфігурацією.

Зовнішній моніторинг контролює об’єкт моніторингу як чорний ящик. Загалом Zabbix відстежує такі елементи:

- Служба моніторингу Monitors чи необхідні послуги, такі як HTTP, SMTP і SSH працюють.
- Моніторинг TCP/UDP Відстежує, чи відкритий сервісний порт TCP/UDP для вказаної IP-адреси.

- ICMP (моніторинг ping) Відстежує, чи функціонує мережа з певною IP-адресою за допомогою ICMP (Internet Control Message Protocol).
- Моніторинг SNMP Відстежує мережеве обладнання, обладнання для живлення та серверне обладнання за допомогою протоколу SNMP (простого протоколу керування мережею).

SNMP насправді не є зовнішнім моніторингом; це моніторинг шляхом зв'язку з внутрішнім модулем SNMP. Однак він часто використовується для моніторингу мережевого обладнання (коммутатори L2, комутатори L3, балансери навантаження). Тому його часто розглядають як зовнішній моніторинг. Zabbix вважає SNMP зовнішнім моніторингом, оскільки Zabbix Agent не використовується для моніторингу.

Зовнішній моніторинг є найважливішим для територій поблизу кінцевого користувача. Воно повинно здійснюватися із зовнішньої по відношенню до служби мережі. Можливо, розглянемо розділення постачальника послуг мережі та постачальника хмарних послуг.

Крім частини, близької до кінцевого користувача, необхідно також контролювати мережу в рамках служби та сервісні з'єднання (наприклад, з'єднання БД) між серверами. Зокрема, якщо вплив зниженої бази даних великий, вам також слід підготувати таблиці для служб моніторингу та контролювати, чи можна виконувати читання/запис для певних таблиць.

Внутрішній моніторинг контролює елемент моніторингу як білий ящик. Для внутрішнього моніторингу користувачеві необхідно встановити Zabbix Agent на цільовий сервер моніторингу та дозволити зв’язок із сервером Zabbix, який виконує центральні процеси моніторингу. Зазвичай з'єднання сервера з агентом встановлюється для отримання інформації моніторингу. [9]

1.6 DOCKER ТА KUBERNETES

Оскільки компанії переміщують свою інфраструктуру та архітектуру, щоб відображати епоху, керовану даними, засновану на хмарі, спостерігається зростання тематики хмарних обчислень, контейнеризації та оркестрації контейнерів. Розповідаючи про ці тенденції, важко проігнорувати деякі з великих імен, як-от Kubernetes і Docker, які за браком кращого слова зробили революцію в тому, як розробляється та розгортається програмне забезпечення в масштабі. Docker, платформа контейнеризації, і Kubernetes, платформа оркестрування контейнерів.

1.6.1 ЩО ТАКЕ DOCKER

Docker робить дуже легким і простим для різних людей, які працюють над проектом, запуск своєї програми в одному середовищі без будь-яких залежностей або проблем з ОС, оскільки Docker надає власну ОС.

До Docker - розробник надсилає код тестувальнику, але він не запускається в системі тестувальника через різні проблеми із залежностями, проте він чудово працює на стороні розробника.

Після Docker - оскільки тестувальник і розробник тепер мають одну й ту саму систему, яка працює на контейнері Docker, вони обидва можуть запускати програму в середовищі Docker без необхідності стикатися з проблемами залежностей, як раніше.

Docker — це платформа контейнеризації, яка упаковує програму та всі її залежності разом у вигляді контейнера докерів. Це набір продуктів платформи як послуги, розроблених для вирішення багатьох проблем, створених тенденцією DevOps, що розвивається.

Docker полегшує створення, розгортання та запуск програм за допомогою контейнерів. Контейнери — це те, що робить Docker таким привабливим для сучасного розробника. Вони створюють абстракцію на

рівні програми, яка упаковує програму та залежності з усім необхідним для виконання, включаючи: операційну систему, код програми, середовище виконання, системні інструменти, системні бібліотеки тощо. Контейнери займають менше місця, ніж віртуальні машини (образи контейнерів зазвичай мають розмір десятків МБ), можуть обробляти більше додатків і потребують менше віртуальних машин і операційних систем.

1.6.2 Що таке Kubernetes

Kubernetes — це потужний інструмент керування контейнерами, який автоматизує розгортання та керування контейнерами. Kubernetes (k8) — це наступна велика хвиля хмарних обчислень. Коли справа доходить до використання контейнерів у виробництві, з часом можна отримати десятки, навіть тисячі контейнерів. Ці контейнери потрібно розгортати, керувати, підключати й оновлювати; якщо робити це вручну, знадобиться ціла команда, яка займається цим.

Недостатньо запускати контейнери; необхідно вміти:

- Інтегрувати та оркеструвати ці модульні частини .
- Збільшувати та зменшувати масштаб відповідно до попиту .
- Зробіть їх відмовостійкими.
- Забезпечити комунікацію в кластері. [10]

Спочатку розроблений Google, Kubernetes є відкритим вихідним кодом з ontainer оркестровка платформою, призначена для автоматизації розгортання, масштабування і управління контейнерних додатків. Фактично, Kubernetes зарекомендував себе як стандарт дефакто для оркестровки контейнерів і є флагманським проектом Cloud Native Computing Foundation (CNCF), який підтримується такими ключовими гравцями, як Google, AWS, Microsoft, IBM, Intel, Cisco і Red Hat.

Оскільки все більше і більше організацій переходять до мікросервісних і хмарних архітектур, які використовують контейнери, вони шукають

надійні, перевірені платформи. Практикуючі переходять на Kubernetes з чотирьох основних причин.

Kubernetes допомагає рухатися швидше. Справді, Kubernetes дозволяє надавати платформу як послугу самообслуговування (PaaS), яка створює рівень апаратної абстракції для команд розробників. Команди розробників можуть швидко та ефективно запитувати необхідні ресурси. Якщо їм потрібно більше ресурсів для обробки додаткового навантаження, вони можуть отримати їх так само швидко, оскільки всі ресурси надходять з інфраструктури, спільної для всіх команд. Більше не потрібно заповнювати форми для запиту на нові машини для запуску програми!

Kubernetes є економічно ефективним. Kubernetes і контейнери дозволяють набагато краще використовувати ресурси, ніж гіпервізори та віртуальні машини. Оскільки контейнери настільки легкі, для їх роботи потрібно менше ресурсів ЦП і пам’яті.

Kubernetes не залежить від хмар. Kubernetes працює на Amazon Web Services (AWS), Microsoft Azure і Google Cloud Platform (GCP), і також можна запускати його на місці. Можна переміщувати робочі навантаження без необхідності перепроектувати свої програми або повністю переосмислити інфраструктуру, що дає змогу стандартизувати платформу й уникати прив’язки до постачальника.

Хмарні провайдери керуватимуть Kubernetes. Як зазначалося раніше, Kubernetes наразі є чітким стандартом для інструментів оркестрування контейнерів. Не дивно, що великі постачальники хмарних послуг пропонують безліч пропозицій Kubernetes як послуги. Amazon EKS, Google Cloud Kubernetes Engine, Azure Kubernetes Service (AKS), Red Hat OpenShift та IBM Cloud Kubernetes Service — усі вони забезпечують повне керування платформою Kubernetes.

Центральним компонентом Kubernetes є кластер. Кластер складається з багатьох віртуальних або фізичних машин, кожна з яких виконує спеціалізовану функцію або як головне, або як вузол. Кожен вузол містить групи з одного або кількох контейнерів і головний вузол спілкується з вузлами про те, коли створювати або знищувати контейнери. У той же час він повідомляє вузлам, як перенаправляти трафік на основі нових вирівнювань контейнерів. [11]

1.6.3 Docker і Kubernetes: краще разом

Обидві технології розроблені для спільної роботи, і коли вони працюють, це мрія DevOps. Як згадувалося раніше, недостатньо просто запускати контейнери у виробництві, їх потрібно регулювати, і Kubernetes пропонує кілька чудових функцій, які роблять роботу з контейнерами ще простішою. Kubernetes пропонує такі речі, як автоматичне масштабування, перевірки справності та балансування навантаження, які є вирішальними для керування життєвим циклом контейнера.

Обидві технології залишаються тут і стають одними з найбільш затребуваних технологій на ринку. Оволодіння ним зараз допоможе створювати краще програмне забезпечення, підвищити рівень кар’єри та виділитися з натовпу.

2 МЕТОДИ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ПОБУДОВИ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ ПРОЇЗДНИХ КВИТКІВ

2.1 Обґрунтування вибору середовища програмної реалізації

У даній роботі для створення логіки мікросервісів було обрано мову PHP, середовище розробки PhpStorm та фреймворк Laravel як найкращий PHP-фреймворк на сьогодні.

2.1.1 Мова програмування PHP

PHP — це мова сценаріїв з відкритим вихідним кодом, для всіх цілей яка широко використовується. Спочатку вона була розроблена для використання в розробці веб-сайтів. Насправді, PHP почав своє життя як персональна домашня сторінка, інструмент для особистих сторінок, розроблений Рамусом Лердорфом для допомоги користувачам із завданнями веб-сторінок. PHP виявився настільки корисним і популярним, що швидко виріс у повноцінну мову з усіма можливостями сьогодення; отримав назву PHP Hypertext Preprocessor, щоб представити його розширені можливості: обробляти веб-сторінки перед їх відображенням.

Популярність PHP продовжує швидко зростати завдяки його численним перевагам:

- Це швидко: на веб-сайтах, і оскільки він вбудований в HTML-код, час на обробку та завантаження веб-сторінки короткий
- Він простий у використанні: синтаксис простий і легкий для розуміння та використання навіть для тих, хто не програмує. Для використання на веб-сайтах PHP-код розроблено для легкого включення до файлу HTML

- Він універсальний: PHP працює на різних операційних системах: Windows, Linux, Mac OS і більшість різновидів UNIX

PHP — це мова загального призначення, яку можна використовувати для написання скриптів для загальних цілей. Скрипти – це комп'ютерні файли, які містять інструкції мовою PHP, які наказують комп'ютеру виконувати дії, наприклад показувати Hello на екрані або зберігати певні дані в базі даних. Більшість сценаріїв містять ряд інструкцій, які можуть виконувати завдання від розробки веб-сторінок до навігації по файловій системі. Оскільки PHP почав своє життя в Інтернеті, він має багато функцій, придатних для використання в сценаріях, які розробляють динамічні веб-сторінки.

На даний момент PHP використовується частіше на веб-сторінках, але його використання для інших цілей зростає. PHP дуже популярний для веб-сайтів. За даними веб-сайту PHP (www.php.net/usage.php), понад 11 мільйонів доменів використовують PHP. Yahoo!, ймовірно, найбільш відвідуваний веб-сайт у світі нещодавно вирішив змінити свою власну мову на PHP.

Спочатку веб-сторінки були статичними: вони лише представляли документи. Користувачі відвідували веб-сайти, щоб прочитати інформацію. Документи були пов'язані, щоб користувачі могли легко знайти потрібну інформацію, але веб-сторінки не змінилися. Кожен користувач, який зайшов на сайт, бачив точно те саме.

Незабаром веб-розробники захотіли зробити більше. Вони хотіли взаємодіяти з відвідувачами, збирати інформацію від користувачів і надавати веб-сторінки, які були б адаптовані для кожної людини. Було розроблено кілька мов, які можна використовувати для створення динамічних веб-сайтів. PHP є однією з найуспішніших мов; Він швидко розвивався, щоб з кожним днем ставати все кориснішим, що швидко збільшило його популярність.

Коли сценарій виконується на сервері, PHP може динамічно створювати HTML-код, який генерує веб-сторінка, що дозволяє окремим користувачам переглядати власні веб-сторінки. Відвідувачі веб-сайту бачать вихідні результати сценаріїв, але це не самі сценарії.

Здатність PHP взаємодіяти з базами даних є особливо надійною. PHP підтримує практично будь-яку базу даних, про яку ви чули та знаєте. PHP обробляє з'єднання та зв'язок між базою даних, тому вам не потрібно знати технічні деталі того, як підключитися до бази даних або як обмінюватися повідомленнями з нею.

PHP добре працює з веб-сайтом, керованим базою даних. Сценарій PHP на веб-сайті може зберігати дані та витягувати дані з будь-якої бази даних, яка підтримується. PHP також взаємодіє з базами даних, які підтримуються за межами веб-середовища. Використання баз даних є однією з найкращих функцій PHP.

PHP може взаємодіяти зі своїми файловими системами: каталогами та файлами, які знаходяться на жорсткому диску або на іншому ПК, доступному через мережу. PHP може записувати у файл у своїй файловій системі. Можна створювати каталоги, копіювати файли, перейменовувати їх, видаляти, змінювати їх атрибути та виконувати багато інших завдань файлової системи. PHP дозволяє виконувати практично будь-які завдання, пов'язані з вашою файловою системою.

Багатьом веб-сайтам потрібно безпосередньо взаємодіяти з файловою системою. Наприклад, веб-додаток може зберігати тимчасову інформацію у файлі замість бази даних, або йому може знадобитися читати інформацію з файлу. [12]

2.1.2 Середовище розробки PhpStorm

PhpStorm — це PHP IDE. Він забезпечує запобігання помилок на льоту, автозаповнення та рефакторинг коду, налагодження нульової конфігурації та

розширений редактор HTML, CSS та JavaScript. PhpStorm також надає потужні вбудовані інструменти для налагодження, тестування та профілювання програм.

PhpStorm надає багатий та інтелектуальний редактор коду для PHP з підсвічуванням синтаксису, розширеною конфігурацією форматування коду, перевіркою помилок на льоту та інтелектуальним завершенням коду.

PhpStorm Редактор PhpStorm поважає PHPDoc у вашому коді та надає відповідні пропозиції щодо завершення коду на основі анотацій @property, @method та @var. Коли ви редагуєте PHPDoc для свого коду, імена та типи змінних автоматично заповнюються з відповідних блоків коду. Рефакторинг коду PHP також розглядає PHPDocs, щоб підтримувати їх в актуальному стані.

Ретельний аналіз вихідного коду дозволяє PhpStorm забезпечувати витончене завершення коду навіть для некоментованого коду, наприклад:

- Тип повернення функції вираховується з її тіла та операторів return
- Типи властивостей класу (і оголошення) витягуються з коду конструктора
- Подання структури файлів і ієрархії класів, методів і викликів дозволяють швидше переглядати код і навігацію.

Спеціальні конфігурації Run/Debug дозволяють у будь-який час виконувати необхідний набір тестів. Тести виконуються в спеціальному інтерфейсі Test Runner, відображаючи огляд результатів і детальну статистику для всього набору та кожного окремого тесту. У разі збою тесту можна миттєво перейти від стека до рядка коду, де сталася помилка. [13]

2.1.3 Фреймворк Laravel

Laravel — це добре відомий безкоштовний веб-фреймворк PHP з відкритим вихідним кодом, спочатку задуманий і створений Тейлором

Отвеллом. З самого початку програмне забезпечення було призначене для розробки веб-додатків, зокрема для тих, які слідують архітектурному шаблону MVC (або шаблону модель-перегляд-контролер) на основі Symfony (іншого фреймворка веб-додатків). Laravel має чимало чудових функцій, таких як модульна система пакування, а також Composer, спеціальний менеджер залежностей мови PHP. Він також дозволяє отримати доступ до реляційних баз даних різними способами та використовує функції, які допомагають під час обслуговування та розгортання додатків. Простіше кажучи, Laravel — це веб-фреймворк, який допомагає створювати нові програми для Інтернету, включаючи такі речі, як API або веб-сервіси. Це допомагає створювати компоненти, які можна повторно використовувати, і виконувати щоденні завдання з більшою легкістю для веб-програм, що спрощує їх використання та обслуговування. При цьому Laravel — це всесвітньо популярне програмне забезпечення для PHP, яке використовується як для малих, так і для великих проектів. Професійні розробники зазвичай хвалять Laravel по всьому світу за його надійну продуктивність, чудові функції та масштабованість. Як було сказано раніше, програмне забезпечення слід добре відомій структурі MVC, що, як правило, значно полегшує початок створення, а потім підтримку функціональності веб-додатків. Він також надає чудові вбудовані функції, такі як аутентифікація, сеанси, маршрутизація та пошта. Усі ці зручні характеристики та функції роблять Laravel досить практичною частиною програмного забезпечення, яке розробники програмного забезпечення повинні опанувати та використовувати. Його вихідний код розміщено на GitHub і ліцензований відповідно до умов ліцензії MIT.

Laravel було створено з потреби розширити межі того, на що були здатні старі частини програмного забезпечення фреймворку, і полегшити щоденне завдання. Ці функції можна розглядати як основні моменти Laravel, і вони не класифікуються в певному порядку, пов’язаному з пакетами. Таким

чином, Laravel завжди зосереджувався на функціях, які зроблять розробку простішою, додатки — надійнішими та складнішими, але водночас простими для навігації.

Модульність – це ступінь, до якої компоненти програми можна об’єднати, розділити, а потім знову об’єднати. Розділивши компоненти на різні модулі, які працюють разом, щоб зробити веб-додаток повністю функціональним, Laravel був розроблений з урахуванням модульності, оскільки саме програмне забезпечення являє собою велику колекцію різних компонентів. Модульність дозволяє легко створювати і розробляти великомасштабні програми з кращою обробкою помилок і виявленням помилок. Laravel дає розробникам простий і легкий спосіб слідувати інструкціям, щоб розпочати роботу та виконати свої проекти.

Управління залежностями – деякі розробники вважають це флагманською функцією програмного забезпечення. Це означає використання різних пакетів або іноді навіть пакетів сторонніх розробників у Laravel з повною сумісністю та функціональністю з легкістю. Наприклад, під час прив’язування стороннього пакета до програми Laravel немає потреби прив’язувати його вручну, оскільки це можна зробити з класу постачальника послуг спеціального пакета Laravel. Проблеми залежностей Laravel вирішуються Composer, менеджером залежностей PHP, який робить використання різних пакетів більш ефективним і менш трудомістким.

Аутентифікація – ця функція стала невід’ємною частиною будь-якого серйозного веб-додатка. У минулому написання аутентифікації в таких фреймворках, як CodeIgniter, було тривалим процесом. Laravel надає цю функцію «з коробки», а все, що розробник повинен зробити, — це запустити просту команду, щоб створити функціональну, безперебійну систему аутентифікації. Крім того, деякі параметри дозволяють розробникам створювати свою автентифікацію.

Кешування. Зберігання даних, які можна легко і швидко отримати, також було основою веб-розробки, і Laravel також має цю зручну функцію. Це дійсно практично, оскільки дозволяє веб-додатку (наприклад, веб-сайту) працювати краще, зменшуючи час обробки інформації під час доступу до програми.

Налагодження та тестування – Laravel має вбудований PHPUnit, який тестуватиме програми. Оскільки сам Laravel був розроблений з урахуванням тестування, функції налагодження та тестування інтегровані.

Локалізація – Laravel полегшує створення багатомовних додатків, оскільки це можна зробити одночасно під час розробки самого веб-додатка. Це може зробити його ідеальним для багатонаціональних веб-проектів або двомовних веб-сайтів.

Маршрутизація – маршрутизацію в Laravel можна використовувати для групування маршрутів, іменування маршрутів, прив’язування до них даних моделі та застосування до них фільтрів.

Шаблонний механізм або BLADE – цей механізм надає величезну кількість допомоги для форматування даних. Він також має реалізоване спадкування шаблонів, що дозволяє розробникам створювати складні, але навігаційні макети. Крім того, Blade не обмежує розробників використовувати звичайний PHP-код у представленнях, які всі компілюються у звичайний код і кешуються. Вони залишаються там, поки не будуть змінені, а це означає, що движок не додає накладних витрат на веб-додатки.

Конструктор запитів до бази даних – Laravel дає розробникам зручний спосіб створювати запити до бази даних з великою кількістю опцій і фільтрів для керування даними та навіть легкого виконання складних запитів.

Eloquent ORM – Eloquent ORM від Laravel забезпечує чудову підтримку більшості механізмів баз даних і ідеально працює з SQLite та MySQL, а також надійною документацією для всіх функцій Eloquent.

Безпека – більшість розробників похвалили Laravel за його здатність створювати безпечні веб-додатки. Для початку є система аутентифікації фреймворків, а також красномовний ORM використовує зв’язування PDO, яке служить захистом від ін’єкцій SQL. Laravel також пропонує вбудовану підтримку для захисту коду від атак XSS. Фреймворк також пропонує різні пакети безпеки для подальшого підвищення безпеки веб-додатків. Такі пакети, як Laravel Security Component, Laravel-ACL, усі служать меті підвищення безпеки додатків.

Envoy – за допомогою цієї функції загальні завдання можна запускати з віддалених серверів із самого додатка, що дозволяє розробникам налаштовувати майбутні завдання для розгортання.

Система міграції – Laravel також має надійну систему міграції для створення структур баз даних. Ці структури можуть бути створені простою мовою PHP замість того, щоб покладатися на SQL. Він дає змогу користувачеві створювати таблиці, бази даних та індекси за допомогою міграцій. Це може стати в нагоді, коли потрібно змінити таблиці, оскільки розробники можуть просто запустити нову міграцію замість того, щоб знову створювати таблицю.

Файлова система. Завдяки підтримці кількох файлових систем, Laravel дозволяє користувачам використовувати локальні файлові системи або хмарне сховище, як-от Amazon S3, з додатковою можливістю вибору між цими системами та їх легкого вибору. [14]

2.2 Опис роботи існуючих систем

Система складається з програмного забезпечення та апаратної частини - валідатора цифрових проїзних. У наступних розділах буде описано всю систему, на прикладі міста Київ.

2.2.1 Валідатор цифрових проїзних

Валідатор в громадському транспорті може бути встановлено в автобусах для багатоцільових функцій, таких як підрахунок витрат на проїзд, вирахування вартості проїзду в автобусі, а також відображення плати за карту та залишку плати. Дисплей, жорстке декодування QR-коду або камера OCR забезпечує надшвидке розпізнавання, NFC-оплата. Функція позиціонування GPS допомагає відстежувати транспортні засоби в реальному часі.

Валідатор підтримує 2 види проїзного квитка - одноразовий QR код та NFC чіп.

Валідатор відправляє запит на сервер для верифікації проїзного, що запитується. Якщо перевірка успішна – поїздка підтверджується та поточний проїзний прив'язується до цього транспортного засобу.

Також, валідатор взаємодіє з влаштуванням перевірки активованих проїзних, з якими працюють контролери.

Розглянемо валідатор, встановлений у транспортних засобах Києва.

Рисунок 2.1 – Приклад валідатора з NFC та сканером QR-коду

Живлення здійснюється завдяки підключенню до центральної системи електропостачання транспортного засобу.

Завдання валідатора - прочитати QR код проїзного, або NFC чіпа, надіслати запит на сервер з отриманими даними від пасажира, повернути відповідь і переключитися в режим зчитування.

Вартість такого пристрою варіюється від 200 до 450 доларів.

2.2.2 Валідатор для перевірки з боку контролерів

Кондукторів замінили контролерами. Вони здійснюють перевірки на різних станціях. Не всі пасажири є законослухняними та оплачують проїзд. У контролерів є спеціальний пристрій, за допомогою якого перевіряють відскановані проїзні. Для зв'язку транспортного засобу з проїзними за

допомогою NFC ці пристрої сполучають з валідаторами транспортного засобу.

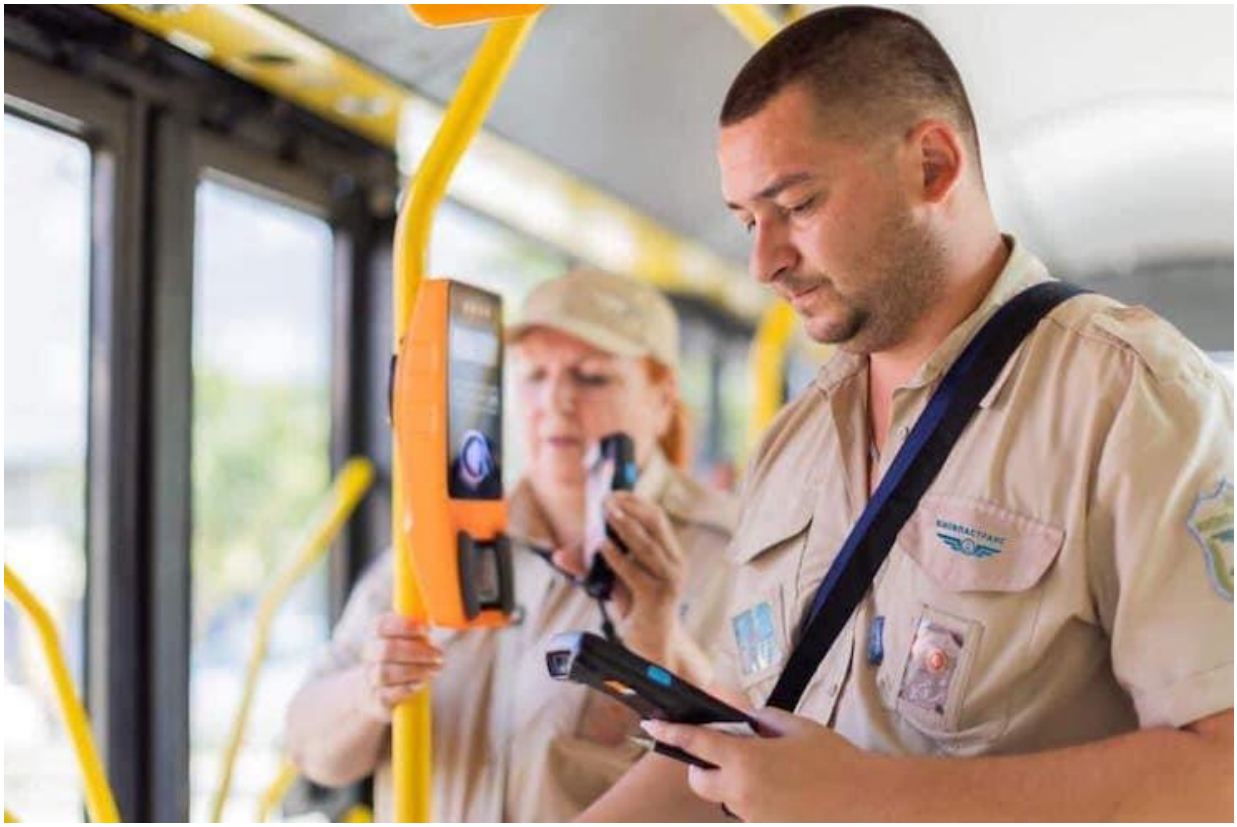


Рисунок 2.2 – Контролери Київпастранс

Після синхронізації пристрою контролер приступає до перевірки проїзних у пасажирів. При взаємодії пристрою з проїзним відбувається запит до системи, де йде пошук недавньої оплати, пов'язаної з цим транспортним засобом.

2.2.3 Купівля проїзного

Термінали «Київ Цифровий» - дозволяють купити разовий QR код, а також поповнити транспортну картку.

Рисунок 2.3 – Термінали «Київ Цифровий»

Каси метрополітену - дозволяють купити разовий QR квиток та оформити транспортну картку, яку можна буде поповнити у будь-який момент через термінал або програму на телефоні.

Рисунок 2.4 – Каси метрополітену

Додаток «Київ Цифровий» - дозволяє купувати разові QR квитки та поповнювати транспортну картку. Містить всю необхідну інформацію для перевірки контролером.

Рисунок 2.5 – Додаток «Київ Цифровий»

Існує достатньо способів купівлі проїзного. Не дивлячись на діджиталізацію, навіть літня людина, яка не має телефону, може запросто купити проїзну.

2.2.4 Висновок про існуючі системи

Система цифрового проїзного є не лише у Києві. У кожного міста своя реалізація цієї системи, зі своїми плюсами та недоліками.

Це максимально зручний задум автоматизації рутинного процесу оплати проїзду. Немає контакту з готівкою у транспорті. Оптимізовані витрати щодо кондукторів, частина яких скорочується, а частина перекваліфікуються на контролерів.

У всього є свій темний бік. Вартість реалізації цієї системи дуже висока і дозволити її може далеко не кожний населений пункт. Головна стаття витрат полягає саме у програмній реалізації та підтримці проекту.

Якщо є можливість реалізувати єдиний програмний продукт та автоматизувати процес диджиталізації проїзду в будь-якому населеному пункті – можна значно знизити витрати на розробку програмного забезпечення та його підтримки, оскільки буде єдина версія продукту, яка працює для кількох населених пунктів.

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ОБЛІКУ ПРОЇЗДНИХ КИТКІВ НА ОСНОВІ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

У результаті має вийти легко масштабована система з єдиною версією програмного продукту. Для кожного населеного пункту буде розгорнутий сервер із клоном програмного забезпечення та специфічною конфігурацією.

3.1 Валідатори у транспорті

Встановлені в салоні валідатори для транспорту об'єднані в локальну мережу з бортовим комп'ютером транспортного засобу. Отримана інформація від усіх приладів аналізується процесором, шифрується і передається спеціальним радіоканалом в процесинговий центр.

Запит, що надійшов обробляється програмним забезпеченням і повертає відповідь, яка виводиться на екрані валідатора.

Розробляти власний валідатор немає сенсу, оскільки ринок надає масу готових апаратів вітчизняного та зарубіжного виробництва.

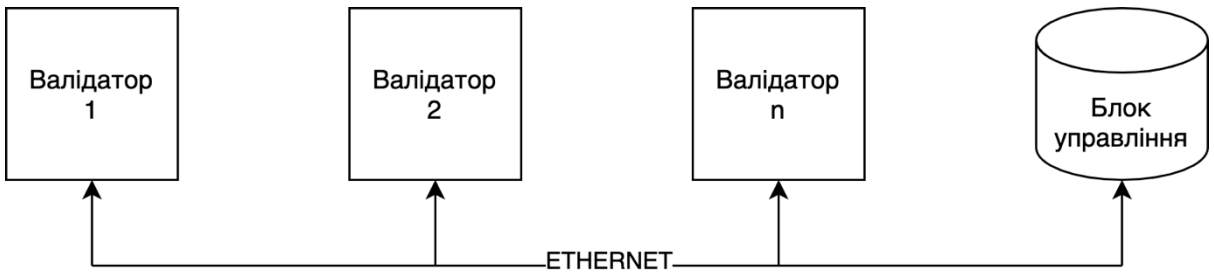


Рисунок 3.1 – Схема валідаторів у транспортному засобі

3.2 Валідатор кондукторів

Цей тип валідаторів є мобільним. Як було написано вище, вони застосовуються для перевірки пасажирів щодо оплати проїзду.

Цей пристрій підключається через валідтор, який перебуває у транспорті. Після успішного сполучення пристроїв - валідтори у транспорті блокуються, не дозволяючи сплатити проїзд у момент перевірки контролерами, щоб уникнути обману з боку пасажира.

Даний пристрій має сканер QR кодів та NFC зчитувач. Це дозволяє валідувати всі типи проїзних. Пристрій взаємодіє зі стаціонарним валідатором транспорту, що дозволяє надсилати та отримувати дані через єдину локальну мережу з одним модемом.

При скануванні проїзного пристрій надсилає запит до валідатора, який у свою чергу робить запит на сервер для отримання інформації про поточний проїзний. Відповідь програмного забезпечення передається назад у мобільний валідатор та виводиться на екрані.

Розробляти власний мобільний валідатор немає сенсу, оскільки ринок також надає масу готових апаратів вітчизняного та зарубіжного виробництва.

3.3 Купівля проїзного

На початковому етапі реалізації проекту достатньо буде розробити єдиний мобільний додаток із вибором населеного пункту, при реєстрації. Що стосується купівлі паперових QR кодів – реалізувати продаж у касах авто та залізничних вокзалів.

Продаж багаторазових проїзних з NFC чіпом не входить до мінімальної реалізації проекту і буде здійснено лише за умови зацікавленості населення до проекту.

Розробляти мобільний додаток під кожен населений пункт – неправильне рішення. Це значно вплине на вартість реалізації проекту і ускладнить завантаження на майданчики для завантаження програми мобільними пристроями, так як з'явиться безліч клонів однієї й тієї ж програми.

Пропонується реалізувати єдиний мобільний додаток. При реєстрації та налаштуваннях можна буде обрати свій населений пункт. Населений пункт у додатку впливатиме на вартість проїзду, яку вказав уряд. Придбаний QR-код можна буде застосувати тільки у вибраному населеному пункті.

Інтерфейс міститиме мінімальний набір функцій:

- Реєстрація.
- Авторизація.
- Налаштування.
- Купівля проїзного.
- Виведення вибраного проїзного на весь екран для зчитування валідатором.

Реєстрація та авторизація буде за номером мобільного телефону через смс-код. Це проста, але надійна система.

Купівля проїзного здійснюватиметься через вбудований платіжний шлюз, Google pay, Apple pay.

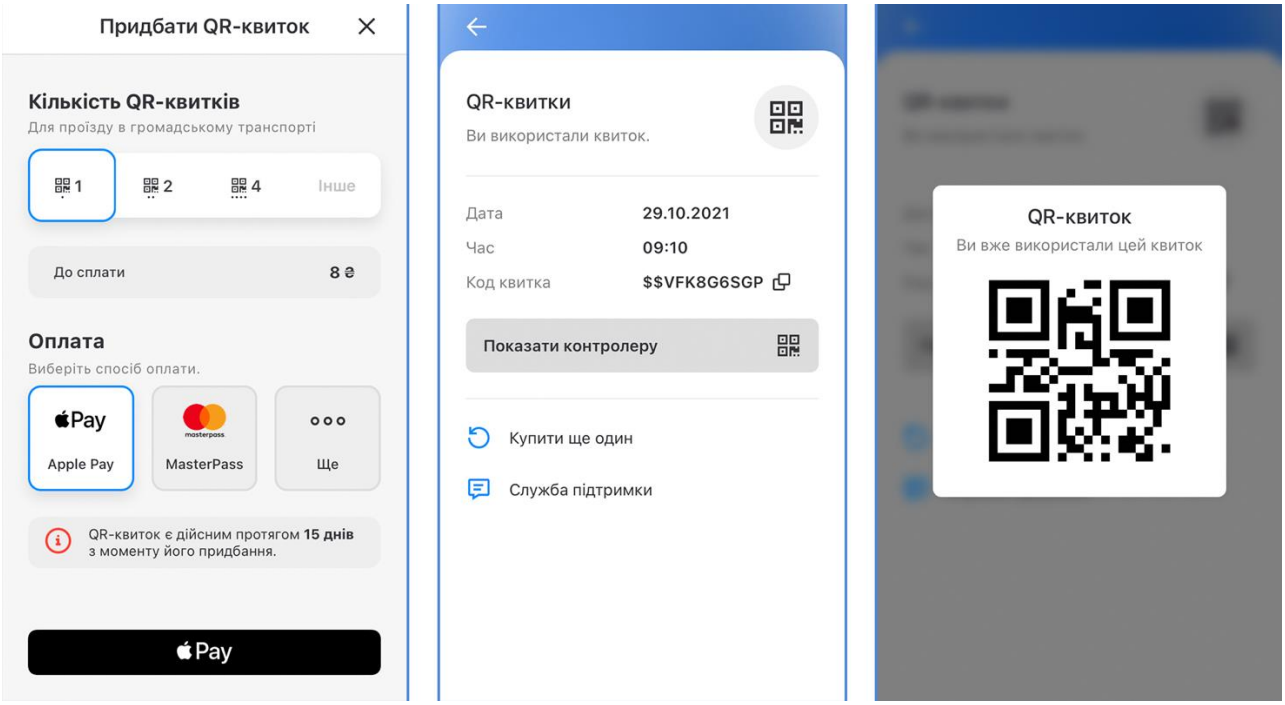


Рисунок 3.2 – Приклад мобільної програми цифрового проїзного

Для друку паперових QR кодів у касах буде використаний термопринтер, підключений до штатного комп'ютера працівника каси. Управління отриманням коду відбуватиметься через веб-додаток, під обліковим записом працівника.

Розробляти десктопний додаток під кожну операційну мережу, не має сенсу через великі витрати. Сучасні браузери відмінно працюють з PDF документами і дають необхідний функціонал швидкого друку на вибраному принтері.

Рисунок 3.3 – Приклад термопринтера

3.4 Розробка мобільного додатку

Буде розроблено дві мобільні програми - для IOS та Android. Вся основна логіка буде на окремому мікросервісі, з яким мобільний додаток буде взаємодіяти через API.

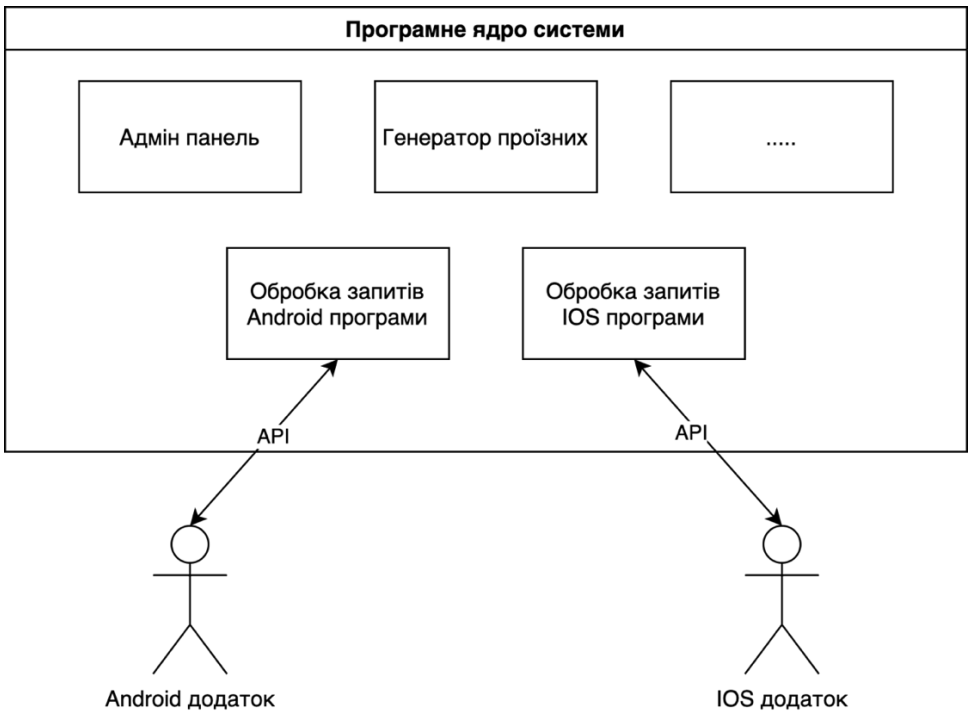


Рисунок 3.4 – Діаграма архітектури роботи мобільних додатків

3.5 Розробка веб-додатків для працівників каси

Буде розроблено веб-додаток. Розробляти настільний додаток немає сенсу, оскільки сучасні браузери дозволяють покрити всі потреби для продажу проїзних. А саме відобразити створений PDF із QR кодом та запустити його друк на підключеному принтері.

Вся бізнес логіка буде знову ж таки на сторонньому мікросервісі, що дозволить знизити витрати на розробку та уникнути дублювання функціоналу в системі.

Програма буде реалізована мовою PHP, фреймворк Laravel. Це найкраще рішення на сьогодні для цього завдання.

3.6 Розробка програмного ядра

Як СУБД буде використано PostgreSQL. Це надійна об'єктно-реляційна система управління базами даних.

Оскільки дані в цьому проекті пов'язані, використовувати noSQL рішення не має сенсу.

Для забезпечення відмовостійкості буде кілька ідентичних БД, саме – для читання та запису.

Такий підхід розподілить навантаження і зробить систему значно надійнішою. При відмові однієї з БД будуть доступні інші. Найкраще використовувати сервери в різних датацентрах, щоб унеможливити надзвичайні пригоди через постачальника послуг. Наприклад, пожежа, затоплення, вихід із ладу серверів через коротке замикання тощо.

У результаті буде одна майстер-база яка буде доступна як на читання так і на запис, а також n ідентичних баз даних, доступних тільки на читання.

На рисунку 2.10 зображено примітивну діаграму розподілу навантаження на кілька баз даних, за умови однієї активної ноди веб-додатку.

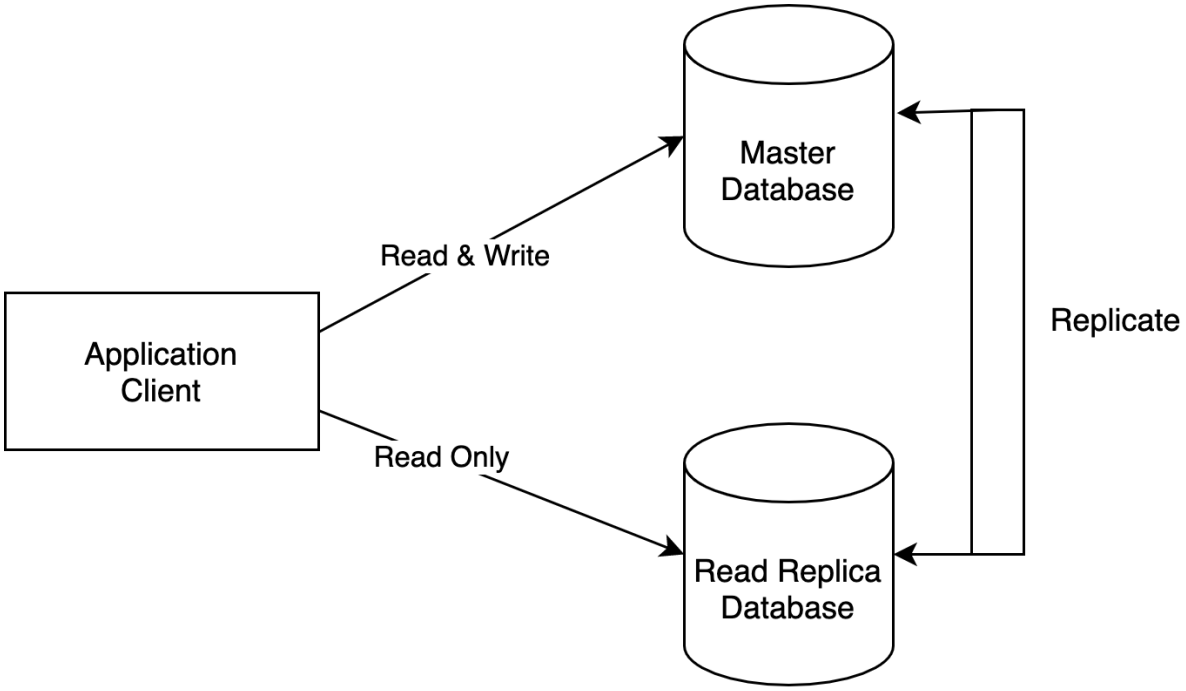


Рисунок 3.5 – Розподіл навантаження на БД

Головна програма буде реалізована на PHP, фреймворк Laravel. Цей фреймворк має дуже потужний функціонал і покриває всі вимоги до системи, що розробляється.

У реалізації MVP буде єдине ядро з усіма необхідними ендпоінтами для мобільних додатків, веб-додатки для купівлі паперового QR коду та обробки запитів валідатора. У майбутньому, якщо система виправдає себе і буде підвищений попит на диджиталізацію проїзду в різних населених пунктах – деякі модулі можна буде винести на окремі мікросервіси.

Спочатку ядро буде реалізовано в модульній архітектурі, що дозволить у майбутньому, без проблем, винести необхідний фукціонал в окремий мікросервіс.

Слабка пов'язаність – головна вимога до проектування архітектури ядра.

Незважаючи на те, що весь функціонал знаходиться в одному місці, для досягнення стійкості до відмов, буде працювати кілька копій ядра на різних серверах, в різних датацентрах. Розподілятиме навантаження

балансувальник. Для запитів буде єдина точка входу – одна URL. Після отримання запиту балансувальник навантаження надсилає запит на менш навантажений сервер і повертає відповідь клієнту.

Підсумкова діаграма серверної реалізації архітектури ядра зображена на рисунку 2.11.

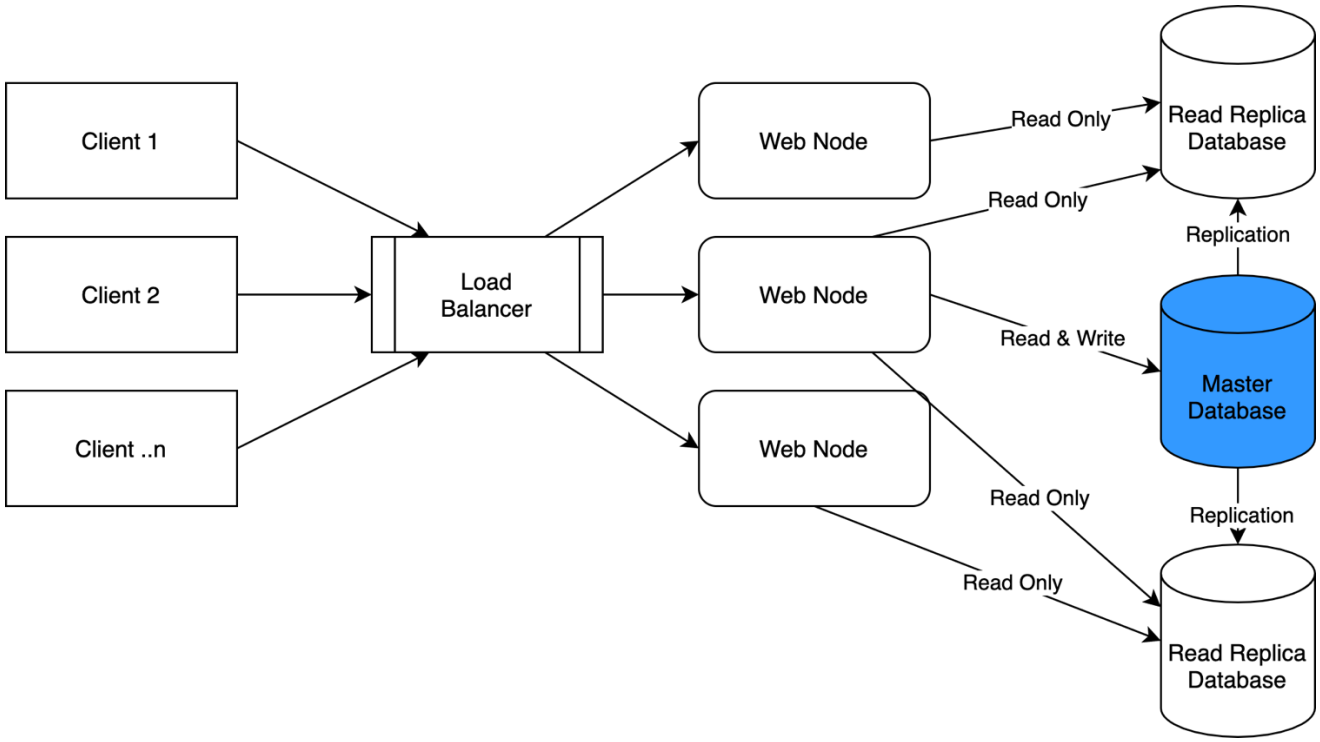


Рисунок 3.6 – Розподіл навантаження усього ядра

Архітектурно, ядро складатиметься з таких модулів:

1. Адмін-панель
2. Обробка запитів веб-додатку з продажу паперових QR кодів
3. Обробка запитів Android програми
4. Обробка запитів IOS програми
5. Обробка запитів стаціонарних валідаторів транспорту
6. Обробка запитів мобільних валідаторів контролерів
7. Генератор проїзних
8. Збір статистики з усіх модулів системи

У таблиці 2.1 описано необхідний набір функцій моніторингу та підтримки всіх модулів ядра. Зрозуміло, функціонал адмін-модуля може бути в кілька разів більшим, але лише за умови зацікавленості у проекті та достатньому фінансуванні.

Таблиця 3.1. Внутрішня структура модуля адмін-панелі.

Ролі та дозволи	Управління ролями та доступами всіх користувачів системи
Логи	Логи різного ступеня важливості з усіх програм системи
Транзакції	Історія платіжних операцій купівлі проїзних
Оплата проїзду	Історія оплати проїзду за допомогою купленого проїзного
Стан стаціонарних валідаторів	Статус роботи стаціонарних валідаторів у транспорті. Моніторинг доступності апаратної системи
Стан ядра системи	Моніторинг стану ядра. • Збір критичних програмних помилок із різних програмних модулів. • Збір інформації із системи моніторингу (кількість запитів на кожен модуль, навантаження на апаратні компопенти серверів тощо)
Управління доступом кас продажу QR кодів	Повний програмний контроль доступу до системи каси продажу QR проїзних. Статистика продажів, управління доступом до системи тощо
Керування доступом мобільного додатку	Управління мобільними користувачами. Блокування до функцій додатка користувача та глобальне відключення обробки запитів з мобільних додатків

Реєстрація у веб-додатку продажу проїзних для кас відбувається у ручному режимі з боку адміністраторів. Для входу в систему видається одноразовий токен. Після першого входу в систему необхідно зберегти в надійне місце новий токен доступу.

Будь-яка підтримка з питань системи здійснюється за гарячою лінією.

Таблиця 3.2. Внутрішня структура модуля веб-додатку

Аутентифікація	Вхід в систему. Перевірка прав доступу до системи за умови блокування точки продажу QR кодів.
Продаж QR коду	Запит на створення QR проїзного для цього населеного пункту. Виведення до друку створеного QR коду.
Аналітика	Базова аналітика продажу за різний термін часу

Таблиця 2.3. Внутрішня структура модуля Android та IOS додатку

Аутентифікація	Реєстрація, верифікація номера телефону, скидання доступу до програми, автентифікація
Основна інформація	Отримання списку невикористаних QR квитків. Історія останніх платіжних транзакцій та поїздок.
Отримання QR квитка	Отримання QR зображення для запитуваного квитка
Історія транзакцій	Повна історія платіжних транзакцій покупки QR квитків
Історія подорожей	Повна хронологія використання QR квитків з отриманням детальної інформації про кожний квиток для перевірки контролером
Налаштування	Зміна населеного пункту у разі потреби придбання квитка в іншому.
Купівля квитка	Обробка платежу та видача QR квитка у разі успішної оплати.

Таблиця 3.4. Внутрішня структура модуля стаціонарного валідатора транспорту

Аутентифікація	Аутентифікація бортового комп'ютера транспортного засобу.
Валідація	Ідентифікація отриманого проїзного.
Зміна режиму роботи	Зміна статусу роботи валідаторів у транспортному засобі, що запитується. Блокування валідаторів за умови перевірки контролерів, розблокування та перевірка доступу бортового комп'ютера до програмного ядра системи.
Валідація проїзного	Перевірка проїзного на відповідність поточному транспортному засобу. Запит здійснюється через мобільний валідатор.

Таблиця 3.5. Внутрішня структура модуля валідатора контролера

Аутентифікація	Обробка запиту на пару мобального валідтора зі стаціонарним. Пошук мобільного валідатора у єдиній базі системи.

Зміна режиму роботи	Блокування та розблокування стаціонарних валідаторів при запиті на перевірку проїзних пасажирів

Модуль генерації проїзних являє собою приватний функціонал, який можна використовувати тільки з інших модулів на підставі успішної фінансової транзакції або запиту з боку фізичної каси. Проїзний є записом у БД з безліччю атрибутів, таких як унікальний ідентифікатор згідно зі стандартом UUID, населений пункт в якому можна його використовувати, користувач який подав запит на створення проїзного, статус та безліч іншої інформації.

Після успішного створення проїзного, формується відповідь клієнту, яка складається із зашифрованого UUID ідентифікатора запису у БД. Незважаючи на те, що UUID практично неможливо підібрати, для додаткової безпеки він шифрується і лише потім, результат шифрування повертається у вигляді рядка клінету. На основі відповіді модуля на стороні клієнта створюється QR код.

Таблиця 3.6. Внутрішня структура модуля генератора проїзних

Створення	Генерація проїзного на основі даних від інших модулів. Повернення ідентифікатора проїзного у зашифрованому вигляді для подальшого створення QR коду.
Отримання інформації	Отримання детальної інформації про проїзний.

Незважаючи на простоту в базовій реалізації, у системі відбувається безліч операцій, особливо фінансових. Процес купівлі та валідації має відбуватися максимально швидко та без помилок. Навіть разові проблеми викликають масу обурень у користувачів. Для цього має бути потужна система збору даних із усіх частин проекту.

Для цих завдань необхідно розробити окремий модуль збору та струтурування даних про помилки, попередження тощо.

Таблиця 3.7. Внутрішня структура модуля збору статистики з усіх модулів системи

Збір даних із сторонніх систем	Збір та збереження важливої інформації із систем моніторингу.
Логування помилок та попереджень	Збір помилок та попереджень під час виконання коду ядра
Логування інформаційних повідомлень	Збір повідомлень інформаційного плану.
Логування помилок доступу	Збір повідомлень про помилки доступу до системи. Спроби доступу до розділів системи з обмеженими правами
Відображення статистики	Формування звітів на основі запиту клієнта

3.7 Розгортання системи для нового населеного пункту

Важливий момент полягає в тому, що вся система перебуває на стороні розробників. Коли місто укладає договір на інтеграцію системи, вся програмна реалізація розміщується на серверах виконавця.

Завдяки тому, що вся програмна реалізація працює всередині Docker – процес розгортання займає значно менше часу, на відміну від варіанта роботи на сервері, без віртуалізації.

Фінансове питання щодо плати за оренду серверів має гнучке рішення. Спочатку програмне забезпечення розгортається на мінімальній кількості серверів. Завдяки віртуалізації, можна використовувати різні дитрибутиви і набори пакетів, на одному сервері, якщо розташовано відразу кілька додатків.

Система моніторингу відображає статистику навантаження, за якою можна робити висновки щодо майбутнього масштабування системи. У перші місяці роботи це дозволить заощадити значні кошти, використовуючи раціональну кількість обчислювальних потужностей.

На початковому етапі буде єдиний програмний продукт для всіх населених пунктів.

Буде єдиний GIT репозиторій з налаштованим деплоєм на сервери кожного населеного пункту. Відмінність буде лише у змінних оточення.

Причиною для перенесення програмного забезпечення в окремий GIT репозиторій певного населеного пункту буде локальне доопрацювання функціоналу. Якщо буде замовлення впровадження специфічних функцій, властивих одному населеному пункту - таку розробку варто проводити лише в окремих репозиторіях.

ВИСНОВКИ

У магістерській роботі були проведені дослідження щодо актуальності та доцільності готового комплексу програмного та апаратного забезпечення системи цифрового проїзного у громадському транспорті.

Було проведено аналіз програмних технологій та готових апаратних рішень для реалізації системи.

Програмне ядро системи буде реалізовано як єдиний сервіс із модульною архітектурою. Такий підхід дозволить зменшити витрати на реалізацію першої версії. Якщо розроблена система буде виправдана значним попитом у населених пунктів завдяки модульній архітектурі програмного ядра, можна буде розділяти навантажені модулі на окремі мікросервіси.

У ході виконання атестаційної роботи було розроблено концепт системи. Програмна реалізація ядра системи вестиметься серед розробці PhpStorm. Основною серверною мовою буде PHP та фреймворк Laravel.

Перша стабільна версія програмного продукту (всієї системи) призначена для диджиталізації процедури оплати за проїзд у міському транспорті, що дозволить значно знизити аварійні ситуації, в які можуть потрапити люди, оплачуючи процїзд у момент пересування транспортного засобу. Диджиталізація описуваного процесу дозволить знизити ризик зараження COVID через тілесний контакт з готівкою.

Як подальший розвиток теми можна скласти детальне технічне завдання, в якому будуть описані всі бізнес-процеси, структура системи, мови та технології, а також мінімальні серверні вимоги для запуску такої системи в тестовому режимі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1) Безсерверні, мікросервісні чи монолітні: вибір архітектури серверної частини для запуску. Режим доступу: www. URL: https://leanylabs.com/blog/serveless-microservices-monolith/

2) RabbitMQ та брокери повідомлень. Режим доступу: www. URL: https://www.fasthosts.co.uk/blog/rabbitmq-and-message-brokers/

3) Що таке Кафка. Режим доступу: www. URL: https://www.educba.com/what-is-kafka/

4) Що таке Кафка. Режим доступу: www. URL: http://cloudurable.com/blog/what-is-kafka/index.html

5) MongoDB: все про документно-орієнтовану базу даних NoSQL. Режим доступу: www. URL: https://datascientest.com/en/mongodb-all-about-the-document-oriented-nosql-database

6) Розуміння архітектури PostgreSQL. Режим доступу: www. URL: https://severalnines.com/database-blog/understanding-postgresql-architecture

7) Кейс для PostgreSQL. Режим доступу: www. URL: https://www.instaclustr.com/blog/the-case-for-postgresql/

8) SQL проти NoSQL. Постгрес проти Монго. Режим доступу: www. URL: https://www.airpair.com/postgresql/posts/sql-vs-nosql-ko-postgres-vs-mongo

9) Вступ до моніторингу за допомогою Zabbix. Режим доступу: www. URL: https://workinjapan.today/hightech/introduction-to-monitoring-with-zabbix/

10) Початок роботи з Docker і Kubernetes: посібник для початківців. Режим доступу: www. URL: https://www.educative.io/blog/docker-kubernetes-beginners-guide

11) Що таке Kubernetes? Вступ до надзвичайно популярної платформи оркестрування контейнерів. Режим доступу: www. URL: https://newrelic.com/blog/how-to-relic/what-is-kubernetes

12) Що таке PHP? Пояснення значення мови програмування PHP. Режим доступу: www. URL: https://disenowebakus.net/en/what-is-php

13) PhpStorm. Режим доступу: www. URL: https://www.alfasoft.com/no/produkter/utvikling/jetbrains/phpstorm.html

14) Що таке Laravel. Режим доступу: www. URL: https://www.popwebdesign.net/what-is-laravel.html

ДОДАТОК А.

Електронні плакати

Міністерство освіти і науки України
Східноукраїнський національний університет ім. В. Даля
Кафедра програмування та математики

Магістерська робота

«Інформаційна система цифрового проїзного квитка міського транспорту на основі мікросервісної архетектури»

ст.гр. ІСТ-20дм Савченко Д.О.
Керівник: Захожай О.І.

Актуальність роботи

- **Метою роботи** є аналіз та проектування системи диджиталізації оплати проїзду у громадському транспортному засобі.
- **Об’єктом дослідження** є методи та інструментальні засоби побудови інформаційних систем обліку цифрових проїздних квітків для міського транспорту.
- **Робота присвячена** складанню концепту системи цифрової оплати проїзду у громадському транспорті.

Проблеми відсутності системи

- Довгий процес оплати за проїзд готівкою
- Підвищена травмонебезпека. Найчастіше за проїзд оплачуємо в момент руху транспортного засобу
- Контакт із грошима з рук у руки. Особливо небезпечно сьогодні, під час пандемії
- Відсутність автоматичних звітів про доходи та кількість пасажирів по кожному з маршрутів

Реалізовані системи

У деяких великих містах України реалізовано свою власну систему цифрової оплати проїзду. У кожного міста своє програмне та апаратне рішення. До мінусів такого підходу можна віднести:

- Великі витрати на розробку проекту які повторюються для кожного міста
- Різні розробники, підходи та підсумкова реалізація
- Кожне місто повторює єдину ідею системи

Мета

- Спроектувати універсальний програмний та апаратний комплекс, який можна буде інтегрувати у будь-який зацікавлений у диджиталізації населений пункт з мінімальними витратами

- Просте розгортання програмної частини для нового населеного пункту без модифікації коду проекту. Єдина кодова база.

Вимоги до системи

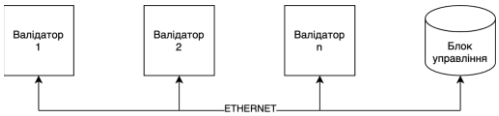
- **Універсальне програмне та апаратне рішення** для диджиталізації всіх базових процесів оплати за проїзд у громадському транспорті населеного пункту.

  - Продаж / Покупка проїзного
  - Валідація проїзного в транспортному засобі
  - Перевірка проїзних пасажирів на рахунок здійсненої оплати

Стаціонарний валідатор

Встановлені в салоні валідатори для транспорту об'єднані в локальну мережу з бортовим комп'ютером транспортного засобу.
Отримана інформація від усіх приладів аналізується процесором та надсилається запитом до програмного ядра системи.
Розробляти свій валідатор немає сенсу, оскільки є багато готових рішень, що в результаті буде значно дешевшим.

Валідатор має два інтерфейси обробки проїзного:
- NFC зчитувач
- Сканер QR коду



Мобільний валідатор

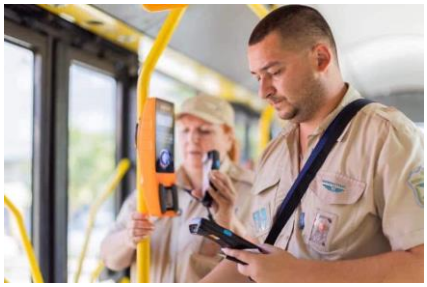
Кондукторів замінили контролерами. Вони здійснюють перевірки на різних станціях. Не всі пасажири є законослухняними та оплачують проїзд.
У контролерів є спеціальний пристрій, за допомогою якого перевіряють проїзні. Розробляти свій валідатор немає сенсу, оскільки є багато готових рішень.

За допомогою NFC ці пристрої сполучають з валідаторами транспортного засобу.

Валідатор має два інтерфейси обробки проїзного:

- NFC зчитувач
- Сканер QR коду

Після сполучення валідаторів - стаціонарні блокуються на момент перевірки контролерами.



Покупка проїзного

У базову реалізацію системи входитиме два способи купівлі проїзного:

- Через єдиний мобільний додаток
- У касі міського вокзалу

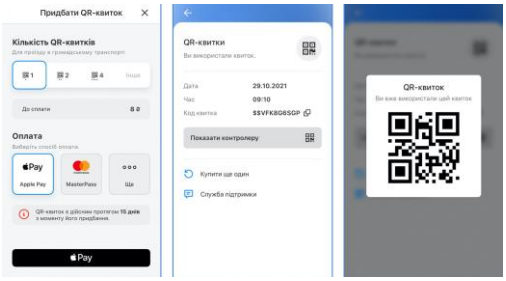
Мобільний додаток

Буде розроблено єдиний мобільний додаток для платформ Android та IOS.

При реєстрації та в налаштуваннях можна буде вибрати свій населений пункт для оновлення локальної цінової політики.

Придбаний QR-код можна буде застосувати тільки у вибраному населеному пункті.

Інтерфейс міститиме мінімальний набір функцій:
- Реєстрація.
- Авторизація.
- Налаштування.
- Купівля проїзного.
- Виведення вибраного проїзного на весь екран для зчитування валідатором.



Покупка у касі міського вокзалу

Продаж паперових QR квитків буде здійснюватися через міські **авто** та **залізничні** вокзали.

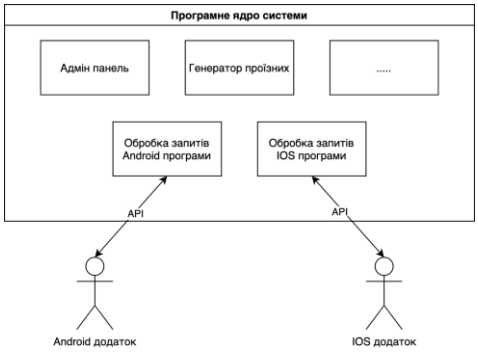
- Для друку паперових QR квитків у касах буде використаний термопринтер, підключений до штатного комп'ютера працівника каси.
- Управління отриманням коду відбуватиметься через веб-додаток, під обліковим записом працівника.

Програмне ядро системи

Спочатку програмне ядро буде реалізовано на основі модульної архітектури. Завдяки такому підходу в майбутньому можна буде виносити модулі в окремі сервіси. Такий підхід економить кошти та час на старті проекту.

Ядро системи є єдиним програмним рішенням для всіх частин проекту:

- Адмін-панель
- Обробка запитів **веб-додатку** з продажу паперових QR кодів
- Обробка запитів **Android** програми
- Обробка запитів **IOS** програми
- Обробка запитів **стаціонарних валідаторів** транспорту
- Обробка запитів **мобільних валідаторів** контролерів
- Генератор проїзних
- Збір **статистики** з усіх модулів системи



Модуль генерації проїзних

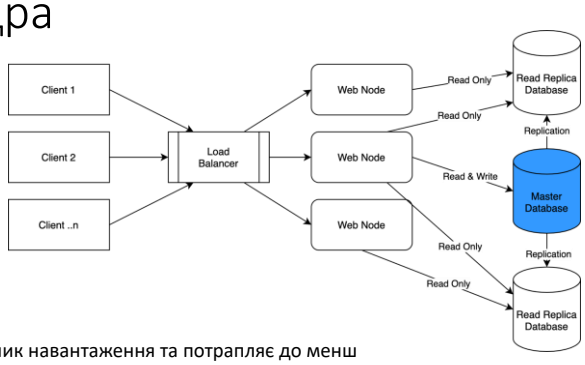
- Модуль генерації проїзних являє собою приватний функціонал, який можна використовувати тільки з інших модулів на підставі успішної фінансової транзакції або запиту з боку фізичної каси.
- Проїзний є записом у БД з безліччю атрибутів, таких як унікальний ідентифікатор згідно зі стандартом UUID, населений пункт в якому можна його використовувати, користувач який подав запит на створення проїзного, статус та безліч іншої інформації.
- Після успішного створення проїзного, формується відповідь клієнту, яка складається із зашифрованого UUID ідентифікатора запису у БД.
- На основі відповіді модуля на стороні клієнта створюється QR код.

Призначення програмного ядра

- Програмне ядро містить логіку всіх елементів системи.
- Мобільні програми та веб-додаток кас залізничних та авто вокзалів є інтерфейсами взаємодії з API програмного ядра і містять лише набір функцій для взаємодії з ядром системи.
- Валідатори також містять лише набір методів взаємодії з API ядра.

Відмовостійкість ядра

Так як ядро на даний момент це єдина програма, розподіл навантаження здійснюється наступним чином:



- Кожен запит проходить через балансувальник навантаження та потрапляє до менш навантаженого екземпляра програми, залежно від типу запиту (читання або запис)
- Розгорнуто кілька екземплярів програми на різних серверах. Більшість доступна лише на читання.
- Розгорнуто кілька екземплярів БД. Більшість доступна лише на читання.

Розгортання системи

- Вся система перебуває на стороні розробників. Коли місто укладає договір на інтеграцію системи, вся програмна реалізація розміщується на серверах виконавця.
- Завдяки тому, що вся програмна реалізація працює всередині Docker – процес розгортання займає значно менше часу, на відміну від варіанта без віртуалізації.
- Спочатку програмне забезпечення розгортається на мінімальній кількості серверів.
- Система моніторингу відображає статистику навантаження, за якою можна робити висновки щодо майбутнього масштабування системи. У перші місяці роботи це дозволить заощадити значні кошти, використовуючи раціональну кількість обчислювальних потужностей.
- Буде єдиний GIT репозиторій з налаштованим деплоєм на сервери кожного населеного пункту. Відмінність буде лише у змінних оточення (ENV).
- Якщо буде замовлення впровадження специфічних функцій, властивих одному населеному пункту - таку розробку варто проводити лише в окремих репозиторіях.

Функціонал модуля адмін-панелі

Ролі та дозволи	**Управління ролями та доступами всіх користувачів системи**
Логи	Логи різного ступеня важливості з усіх програм системи
Транзакції	Історія платіжних операцій купівлі проїзних
Оплата проїзду	Історія оплати проїзду за допомогою купленого проїзного
Стан стаціонарних валідаторів	Статус роботи стаціонарних валідаторів у транспорті. Моніторинг доступності апаратної системи
Стан ядра системи	Моніторинг стану ядра. □ Збір критичних програмних помилок із різних програмних модулів. □ Збір інформації із системи моніторингу (кількість запитів на кожен модуль, навантаження на апаратні компоненти серверів тощо)
Управління доступом кас продажу QR кодів	Повний програмний контроль доступу до системи каси продажу QR проїзних. Статистика продажів, управління доступом до системи тощо
Керування доступом мобільного додатку	Управління мобільними користувачами. Блокування до функцій додатка користувача та глобальне відключення обробки запитів з мобільних додатків

Функціонал модуля веб-додатку

Аутентифікація	**Вхід в систему. Перевірка прав доступу до системи за умови блокування точки продажу QR кодів.**
Продаж QR коду	Запит на створення QR проїзного для цього населеного пункту. Виведення до друку створеного QR коду.
Аналітика	Базова аналітика продажу за різний термін часу

Функціонал модуля Android та IOS додатку

Аутентифікація	**Реєстрація, верифікація номера телефону, скидання доступу до програми, автентифікація**
Основна інформація	Отримання списку невикористаних QR квитків. Історія останніх платіжних транзакцій та поїздок.
Отримання QR квитка	Отримання QR зображення для запитуваного квитка
Історія транзакцій	Повна історія платіжних транзакцій покупки QR квитків
Історія подорожей	Повна хронологія використання QR квитків з отриманням детальної інформації про кожний квиток для перевірки контролером
Налаштування	Зміна населеного пункту у разі потреби придбання квитка в іншому.
Купівля квитка	Обробка платежу та видача QR квитка у разі успішної оплати.

Функціонал модуля
стаціонарного валідатора транспорту

Аутентифікація	**Аутентифікація бортового комп'ютера транспортного засобу.**
Валідація	Ідентифікація отриманого проїзного.
Зміна режиму роботи	Зміна статусу роботи валідаторів у транспортному засобі, що запитується. Блокування валідаторів за умови перевірки контролерів, розблокування та перевірка доступу бортового комп'ютера до програмного ядра системи.
Валідація проїзного	Перевірка проїзного на відповідність поточному транспортному засобу. Запит здійснюється через мобільний валідатор.

Функціонал модуля валідатора контролера

Аутентифікація	**Обробка запиту на пару мобального валідтора зі стаціонарним. Пошук мобільного валідатора у єдиній базі системи.**
Зміна режиму роботи	Блокування та розблокування стаціонарних валідаторів при запиті на перевірку проїзних пасажирів

Функціонал модуля генератора проїзних

Створення	**Генерація проїзного на основі даних від інших модулів. Повернення ідентифікатора проїзного у зашифрованому вигляді для подальшого створення QR коду.**
Отримання інформації	Отримання детальної інформації про проїзний.

Функціонал модуля збору статистики з усіх модулів системи

Збір даних із сторонніх систем	**Збір та збереження важливої інформації із систем моніторингу.**
Логування помилок та попереджень	Збір помилок та попереджень під час виконання коду ядра
Логування інформаційних повідомлень	Збір повідомлень інформаційного плану.
Логування помилок доступу	Збір повідомлень про помилки доступу до системи. Спроби доступу до розділів системи з обмеженими правами
Відображення статистики	Формування звітів на основі запиту клієнта

Висновки

- Були проведені дослідження щодо актуальності та доцільності готового комплексу програмного та апаратного забезпечення системи цифрового проїзного у громадському транспорті.
- Програмне ядро системи буде реалізовано як єдиний сервіс із модульною архітектурою. Такий підхід дозволить зменшити витрати на реалізацію першої версії. Якщо розроблена система буде виправдана значним попитом у населених пунктів завдяки модульній архітектурі програмного ядра, можна буде розділяти навантажені модулі на окремі мікросервіси.
- Як подальший розвиток теми можна скласти детальне технічне завдання, в якому будуть описані всі бізнес-процеси, структура системи, мови та технології, а також мінімальні серверні вимоги для запуску такої системи в тестовому режимі.