КОМП'ЮТЕРНІ НАУКИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

УДК 004.422 **DOI: 10.30977/BUL.2219-5548.2020.89.0.7**

АРХІТЕКТУРА ДОДАТКУ ДЛЯ ПОШУКУ МІСЦЬ ДЛЯ ПАРКУВАННЯ У МІСТІ

Лантрат О. В., Сахно Є. В., Шапошнікова О. П.
Харківський національний автомобільно-дорожній університет

Анотація. *Метою роботи є розроблення архітектури додатку, який можна використовуватися як WEB- або мобільний додаток залежно від потреб користувача для пошуку та користування вільним паркувальним місцем. Для досягнення мети, дослідивши результати аналізу наявних архітектур, було обґрунтовано вибір мікросервісної архітектури та розроблено модель архітектурного вирішення цього питання. Розроблена система підтримки зовнішніх конфігурацій, налаштовано систему пошуку та взаємодії сервісів, систему контейнеризації та швидкого розгортання додатків.*

Ключові слова: *архітектура, мікросервіс, патерн, фреймворк, контейнеризація, розгортання.*

Вступ

У зв'язку з різким збільшенням кількості автомобільного транспорту у великому місті для власників автотранспорту стало дуже проблематично знайти вільне місце для паркування. Розроблення якісного додатку, який зможе надати інформацію про доступні паркувальні місця, місце розташування майданчика, інфраструктурні характеристики місць для паркування, є актуальним для вирішення цієї проблеми.

Розроблення додатків, як і будь-якого програмного забезпечення (ПЗ) є досить складним процесом, де проектування надійної архітектури відіграє вирішальну роль.

Архітектура ПЗ складається із важливих проектних рішень щодо структур програми та взаємодії між ними. Проектні рішення забезпечують певний набір властивостей, який має підтримувати програма, що розробляється. Ці рішення надають концептуальну основу для розроблення ПЗ, її підтримки та обслуговування [1].

Від правильного вибору архітектури ПЗ залежить простота та ефективність процесу розроблення та супроводу програми, тому що в процесі проектування архітектури визначаються внутрішні властивості ПЗ та здійснюється деталізація його зовнішніх властивостей на основі сформульованих бізнес-вимог.

Аналіз публікацій

У технології програмування немає чіткого визначення архітектури ПЗ [2–9]. Значна частина визначень містить одне або декілька положень. Тому очевидно, що поняття архітектури ПЗ багатозначне. Суть програмної архітектури полягає у структуруванні системи, яка містить програмні компоненти, видимі зовні властивості цих компонентів та відношення між ними.

Архітектура – це процес перетворення таких характеристик ПЗ, як гнучкість, масштабування, можливість реалізації, багатократність використання та безпека в структуроване рішення, яке відповідає як технічним, так і бізнес-вимогам.

Для побудови архітектури розглядалось два основні підходи: монолітна та мікросервісна архітектура. Монолітна архітектура передбачає компонування функціоналу в одному додатку та має певні переваги та недоліки. Мікросервісна архітектура – це підхід до розроблення додатків, в якому великий додаток будується як набір модульних служб, тобто набір вільно пов'язаних модулів або компонентів. Кожен модуль підтримує конкретну бізнес-мету і використовує простий, чітко визначений інтерфейс для спілкування з іншими наборами послуг [10–12].

Відповідно до вимог проекту була вибрана мікросервісна архітектура.

Мета і постановка завдання

Розроблення додатку для пошуку паркувальних місць є досить актуальною та підтверджується великою потребою цього сервісу у великих містах особливо зі збільшенням трафіку, який пропорційно збільшується зі зростанням кількості населення великих міст. Додаток надаватиме можливість швидкого та зручного пошуку парковочного місця у місті Харків за допомогою геолокації.

Метою роботи є розроблення архітектури додатку, що зможе використовуватися як WEB- або мобільний додаток залежно від потреб користувача для пошуку та користування вільним паркувальним місцем.

Вибір інструментів для розроблення додатку

Розроблення додатку для пошуку та користування паркувальними місцями здійснено за допомогою інтегрованого середовища розробки Intelij IDEA.

Для реалізації додатку на основі мікросрвісної архітектури було вибрано велику кількість інструментів, які пов'язані з цією архітектурою, тобто побудовані на основі паттернів, дійсних для додатків цієї структури. Основну частину цього інструментарію склали додатки Netflix OSS та їхніх продуктів співпраці з Spring Cloud.

Загальна структура додатку із позначеннями використаних інструментів наведена на рисунку 1.

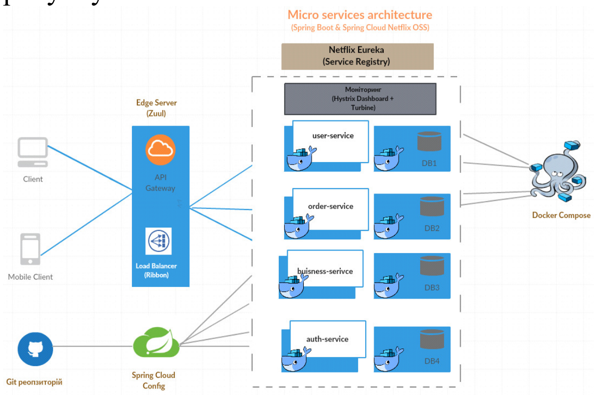

Рис. 1. Реалізація мікросервісної архітектури додатку

Для проектування архітектури за основу були взяті принципи, які є вирішальними для мікросервісів:

– еластичність відносно падінь сервісів. Кожен сервіс повинен мати мінімальний вплив на всю систему у випадку збою одного з них;

– покращене масштабування. Оскільки сервіси не залежать один від одного, то на розгортання одного або декількох з них необхідно набагато менше часу та затрат з боку розробників;

– незалежність сервісів один від одного дозволяє створювати сервіс для специфічних бізнес-задач.

Сервер конфігурацій

Кожен додаток має конфігураційні властивості. У більшості випадків їх заносять до відповідного файлу та розгортаються з упакованою програмою. У разі виникнення потреби в зміні властивості програма має бути перезапущена. Використання централізованого місця зберігання властивостей (в проекті цим місцем є GitLab) має такі переваги:

– конфігурація не запаковується і розгортається з кодом програми, що дозволяє змінювати або відхиляти конфігурацію без відновлення або розгортання програми;

– мікросервіси, які мають спільну конфігурацію, не повинні дублювати схожі параметри для декількох сервісів, тому можуть бути перенесені до загальної конфігурації та розподілити ці властивості;

– важливі деталі конфігурації можуть бути зашифровані і підтримуватися окремо від коду програми. Нешифровані значення можуть бути доступними для застосування на вимогу, а не вимагати, щоб програма несла код, що розшифровує інформацію.

Таким чином, за допомогою Spring Cloud Config в додатку була реалізована можливість зберігання конфігурацій на Git- репозиторії, налаштування відповідного мапінгу, уникнення необхідності перезавантаження сервісу за потреби зміни тих чи інших властивостей додатку.

На рис. 2 подано архітектуру взаємодії між сервером з конфігураціями та сервісами, які використовують ці конфігурації.

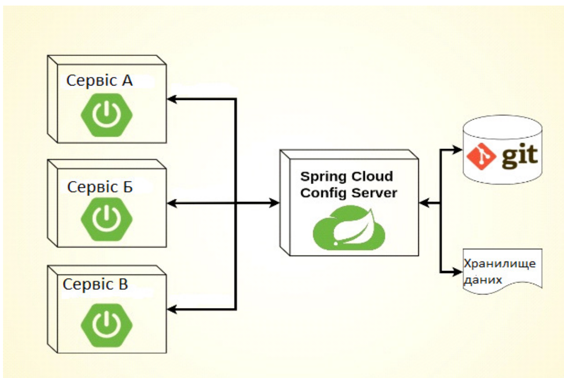

Рис. 2. Архітектура взаємодії сервера конфігурацій з іншими сервісами

У випадку зміни конфігурації на сервері (Git) відправляється повідомлення до сервера конфігурацій (Spring Cloud Config) [13], з якого потім відправляється повідомлення до кожного сервісу, який відстежує зміни пов'язаних з ним властивостей.

Оскільки як паттерн сервісу конфігурацій було вибрано Spring Cloud Config, цей модуль підключається до проекту як зовнішня залежність [14] за допомогою системи автоматичного збирання Gradle. Тобто, Gradle підключає цю бібліотеку до проекту.

Виявлення служб є одним із ключових принципів архітектури на основі мікросервісів. Спроба вручну налаштувати кожного клієнта або форму угоди може не мати бажаного результату. Eureka – це сервер і клієнт служби виявлення Netflix [15]. Eureka діє як центральний реєстр всіх служб у застосуванні мікросервісу і може розглядатися як мікросервіс, задача якого полягає в тому, щоб допомогти іншим службам виявити один одного.

На рис. 2 подано як сервіси реєструють себе на сервері Eureka та зв'язуються один з одним, використовуючи унікальні імена для кожного сервісу, які зберігається та відстежуються на Eureka сервері. Коли новий сервіс розгортається, він відправляється до Eurka-сервера та реєструє себе там. Під час запиту користувачем Сервісу Б потрібно скористатися функціоналом, який надає Сервіс А. Оскільки існує можливість динамічного розгортання сервісів, додаток не повинен орієнтуватися на властивості, які мають особливість змінюватись разом зі зміною їхніх мережевих властивостей, тому використовуються імена сервісів, а не їхні порти.

Використання Eureka в проекті дозволило уникнути використання прямої специфікації серверів, на яких розгортаються екземпляри додатків, та реєструвати їх за допомогою унікального імені. Ім'я сервісу має визначну роль в Cloud-середовищі, бо саме воно дозволяє комунікацію між сервісами.

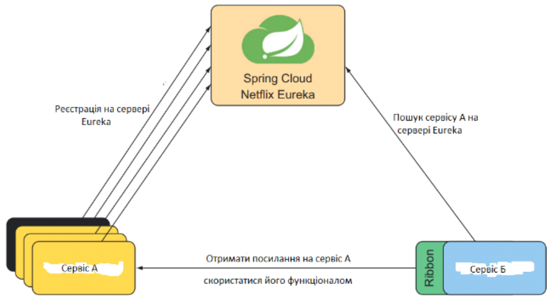

Рис. 3. Реєстрація сервісів Eureka та їхній зв'язок один з одним

Єдиною точкою входу до додатку є API-шлюз. Це служба, яка є точкою входу до програми з мережі інтернет, що відповідає за маршрутизацію запитів, склад API та інші функції, наприклад автентифікацію. Усі запити API від зовнішніх клієнтів спочатку переходять на шлюз API, який маршрутизує деякі запити до відповідної служби. Шлюз API [16] обробляє інші запити, що використовують шаблон композиції API і викликають кілька служб, і агрегування результатів.

Під час розроблення він використовується для підтримки захищеності сервісів. На відміну від монолітної архітектури, яка може бути побудована з підтримкою сесії, що зберігається на сервері і це стає проблемою у разі розгортання нових екземплярів, мікросервіси не мають зберігати такий стан. [17].

На рис. 4 подано діаграму діяльності для процесу автентифікації. Через фільтри проксі- серверу визначається, що користувач не має JWT-токена, і запит за допомогою Eureka та Ribbon, вбудованого розподільника навантажень (Load balancer) відправляють до сервісу автентифікації.

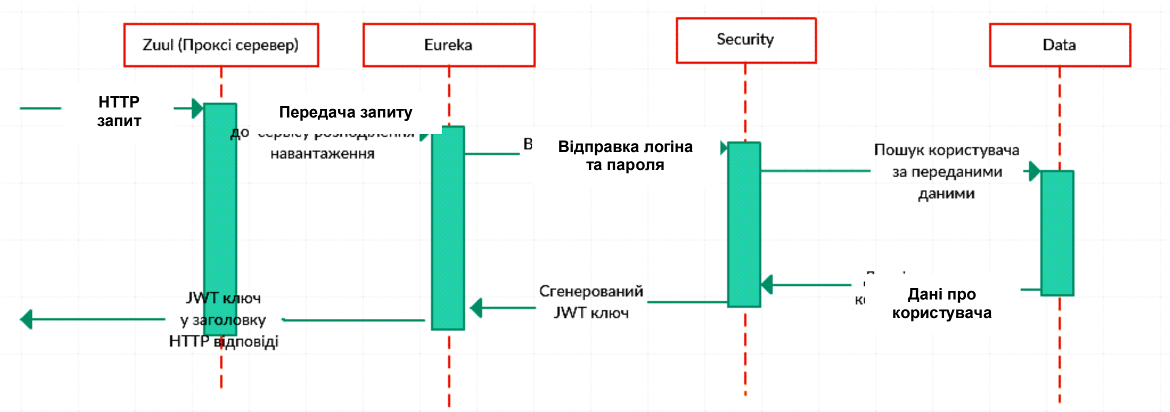

Рис. 4. Діаграма діяльності для процесу автентифікації

За допомогою Ribbon відбувається балансування навантаження. Нині алгоритмом балансування є Round Robin. Згодом перевіряються дані користувача на коректність і в разі відповідності генерується токен та повертається до проксі-сервісу, де він може бути внесений до заголовка HTTP або до Cookie.

На рис. 5 наведена діаграма послідовності, яка відображає процес запиту до будь-якого ресурсу додатку. Запит надходить до шлюзу з відкритим вихідним кодом Zuul проксі-сервісу, де перевіряється на наявність токена. У разі його наявності за допомогою Eureka вибирається сервіс, до якого необхідно відправити запит. [18].

User-service, отримавши запит, перенаправляє його до сервісу безпеки з метою перевірки доступу цього користувача для специфікованого ресурсу. У разі коректної ролі User-service повертає тіло відповіді до користувача. Для контейнеризації сервісів на ресурсі, на якому буде розгортатись додаток, має підтримуватись віртуалізація та встановлений інструмент, який надає зручний інтерфейс для роботи з контейнерами Docker.

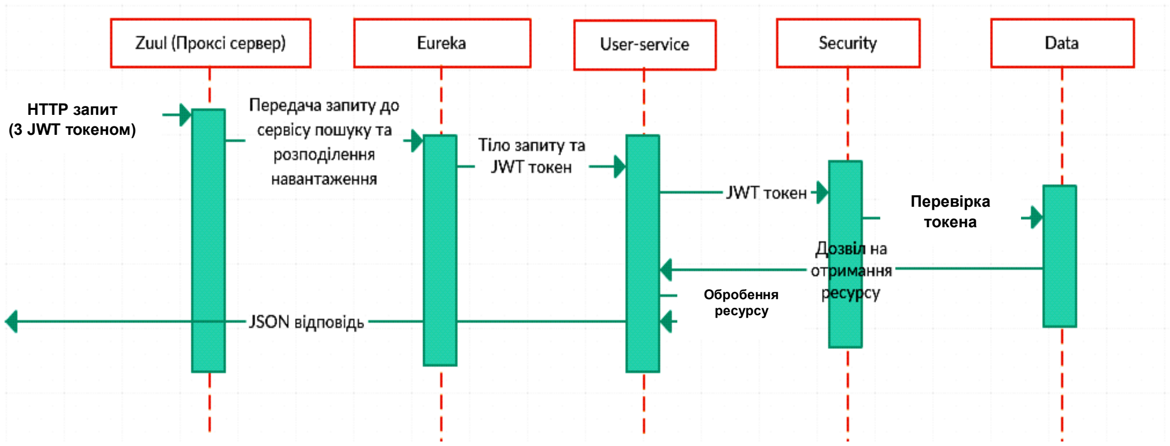

Рис. 5. Діаграма послідовності, яка відображає процес запиту до ресурсу додатку

Невід'ємною частиною використання мікросервісів є їхня контейнеризація з подальшим розгортанням, для цього було вибрано найбільш популярний на сьогодні інструмент Docker.

Щоб створити образи для кожного сервісу, у роботі було використано Docker – плагін, який використовує його API для роботи з Docker – потоком-демоном [19].

У проекті використано Spring Boot Application Plugin. Підключення здійснюється за допомогою змін в build.gradle-файлі.

Після підключення плагін описує процес створення зображення сервісу:

Плагін дає можливість використання деяких готових завдань для Gradle:

- dockerCreateDockerfile – створює Docker-зображення для зазначеного сервісу,
- dockerBuildImage – розгортає Docker- зображення,
- dockerPushImage – надсилає створене зображення до сховища.

Таким чином, створено файл компонування для Docker, який дозволяє локально розгорнути всі сервіси додатку з можливістю перевести всі контейнери до Docker-сервісів та розгорнути їх у кластері.

Структура бази даних додатку

Відповідно до вимог була розроблена структура бази даних для додатку та здійснена її декомпозиція, яка дозволяє розглянути відповідні частини для кожного сервісу (рис. 6).

Для цього було вибрано два типи бази даних для використання їх в тих чи інших умовах. Також було використано ORM- фреймворк Hibernate для об'єктно-реляційного відображення, тобто для роботи з Java-об'єктами, та інструмент Liquibase для контролю за змінами в базі даних.

Розділ «Місце для паркування» не тільки фіксує всю важливу інформацію про парковку, але й спрощує спосіб керування найменшою одиницею парковки (слотом). Деякі стовпці таблиці були додані для того, щоб зробити резервування та операції паркування більш ефективними в таких розділах:

1) parking_lot – зберігає основну інформацію про місця для паркування;

2) block – майданчик для паркування можна поділити на один або декілька блоків. У цій таблиці зберігається інформація про кожен з них;

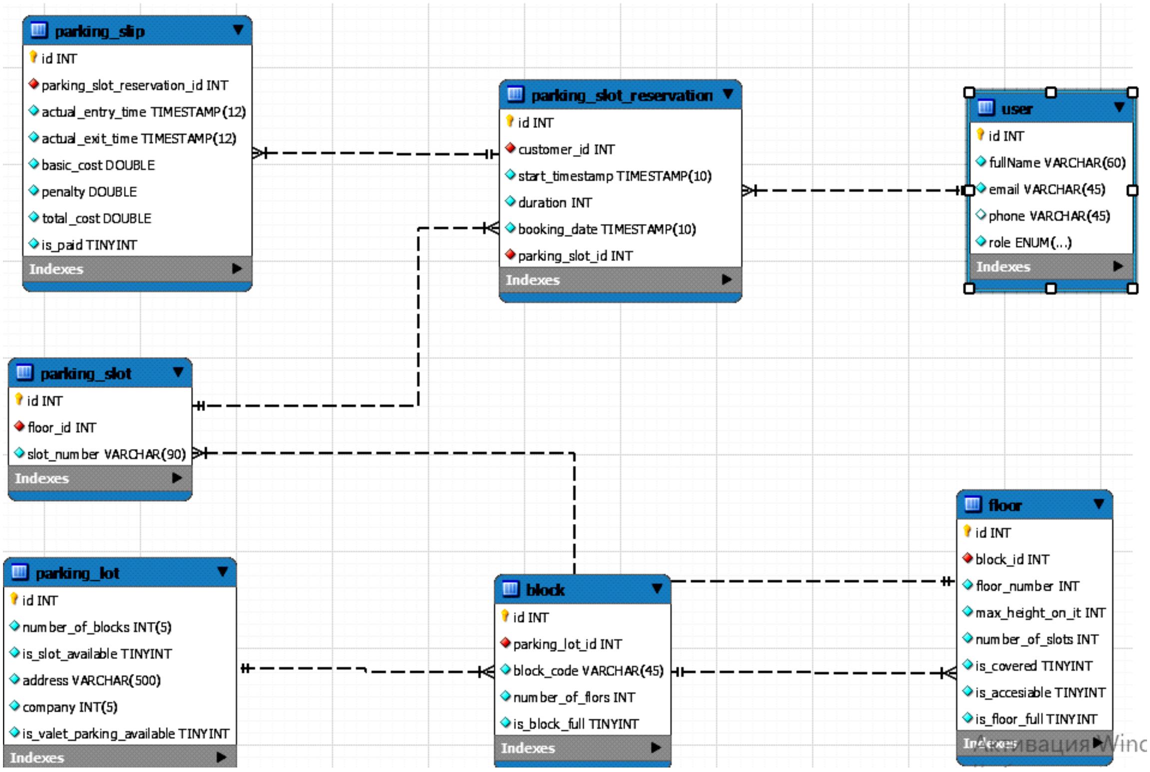

Рис. 6. Схема бази даних додатку

3) floor – у багаторівневих паркінгах блоки можуть мати більш ніж один поверх. На цю таблицю також можна зробити посилання за допомогою блоків наземного рівня;

4) parking_slot – ця таблиця зберігає всю інформацію про місця для паркування на автостоянці.

Розділ «Користувачі» призначено для зберігання основних даних про користувачів, їх ролі та найбільш важливі данні для ефективного користування додатком;

5) user – таблиця для зберігання контактних даних та ролі користувача цієї системи;

6) бронювання (parking_slot_reservation) – зберігає деталі бронювання. Цей розділ створений для деталізації процесу резервування місця для паркування. Під час оформлення замовлення клієнт зазвичай надає такі деталі, як очікувана дата та час прибуття, час, на який бажано зарезервувати слот, та схожі дії;

7) parking_slip – зберігає інформацію про час входу та виходу клієнта, а також про будь-які відповідні збори. Ця таблиця створена для таких майданчиків для паркування, що дозволяють кілька записів і виходів під одним резервуванням.

Висновки

Під час роботи було проаналізовано наявні архітектурні рішення, їхні переваги, недоліки та вибрано мікросервісну архітектуру, що є найбільш прийнятною для розроблення додатку для пошуку місць для паркування у місті Харків.

Розроблено систему підтримки зовнішніх конфігурацій, налаштовано систему пошуку та взаємодії сервісів, а також систему контейнеризації та швидкого розгортання додатків, реалізовано сервіс автентифікації користувача.

Використовуючи мову моделювання UML, було розроблено модель архітектурного рішення.

Відповідно до вимог [20]. була розроблена структура бази даних для додатку та розроблена її декомпозиція.

Література

1. Бендик Н. В., Петрова С. А. Системная архитектура информационных систем: Учебное пособие. Иркутск: Изд-во Иркутского ГАУ, 2016. 92 с.
2. Электронный ресурс (Архитектура ПО: разница между архитектурой и проектированием) – режим доступа: https://medium.com/nuances-of-programming/архитектура-по-разница-между-архитектурой-и-проктированием-204f2e7aeff
3. Электронный ресурс (Кратко о типах архитектур программного обеспечения, и какую из них выбрали мы для IaaS-провайдера) - режим

доступа: https://habr.com/ru/company/1cloud/blog/424911/
4. Фаулер М. Архитектура корпоративных программных приложений / пер. с англ. Москва: Вильямс, 2006. 544 с.
5. Руководство Microsoft по проектированию архитектуры приложений [Электронный ресурс]. URL: http:// apparchguide.ms/Book (дата обращения: 17.11.2019).
6. Мацяшек Л. А., Лионг Б. Л. Практическая программная инженерия на основе учебного примера / пер. с англ. Москва: «БИНОМ. Лаборатория знаний», 2009. 956 с.
7. Назаров С. В. Архитектура и проектирование программных систем: монография. Москва: Инфра-М, 2016. 374 с
8. Фаулер М. Шаблоны корпоративных приложений. Москва: Издат. дом «Вильямс», 2011. 544 с.
9. Караванов А. В., Иванов Н. Д. Архитектура программного обеспечения для высоконадежных систем. Космические аппараты и технологии. 2018. Т. 2. № 2. С. 100–104
10. Шитько А. М. Проектирование микросервисной архитектуры программного обеспечения. Труды БГТУ. Сер. 3: Физико-математические науки и информатика. 2017. №9 (200). URL: https://cyberleninka.ru/article/n/ proektirovanie-mikroservisnoy-arhitektury-programmnogo-obespecheniya (Дата звернення 19.11.2019).
11. Электронный ресурс (Когда оправдано использование микросервисной архитектуры) - http://hawkhouse.ru/blog/kogda-opravdano-ispolzovanie-mikroservisnoj-arhitektury/
12. Электронный ресурс (Load Balancing) – Режим доступа: https://microservices.io/patterns/server-side-discovery.html
13. Электронный ресурс (Pattern: Externalized configuration) Режим доступа: https://microservices.io/patterns/externalized-configuration.html
14. Електронний ресурс (Declaring dependencies) Режим доступу: https://docs.gradle.org/current/userguide/declarin g dependencies.html
15. Електронний ресурс (Service Discovery: Eureka Clients) – режим доступу: https://cloud.spring.io/spring-cloud-netflix/multi/multi__service_discovery_eureka_cl ients.html
16. Електронний ресурс (API Gateway / Backends for Frontends) - Режим доступу: https://microservices.io/patterns/apigateway.html
17. Електронний архів (Wikipedia. Hypertext Transfer Protocol) - Режим доступу: https://en.wikipedia.org/wiki/Hypertext_Transfer Protocol
18. Електронный ресурс (Ограничение скорости в Spring Cloud Netflix Zuul) - Режим доступа: https://www.codeflow.site/ru/article/spring-cloud-zuul-rate-limit
19. Електронный ресурс (Docker, часть 2 – работа с контейнерами) - Режим доступа: (https://itproffi.ru/docker-chast-2-rabota-s-kontejnerami/)
20. Лантрат О.В., Сахно Є.В., Шапошнікова О.П. Розроблення мобільного додатку «Міські парковки» / Вісник ХНАДУ, вип. 87, 2019, С. 59-66.

References

1. Benddik N. V., Petrova S. A. Sistemnaya arhitektura informatsionnyh system: uchebnoe posobie. Irkutsk: Szd-vo Irkutskogo GAU, 2016. 92 s.
2. Elektronnyi resurs (Arckitektura PO: raznitca mezdu arhitekturoyi I proektirovaniem) – rezim dostupa: https://medium.com/nuances-of-programming/архитектура-по-разница-между-архитектурой-и-проктированием-204f2e7aeff
3. Elektronnyi resurs (Kratko o tipach arhitektur programmnogo obespecheniay. I kakuiu iz nikh my vybrali dlia IaaS-provaidera) rezim dostupa: https://habr.com/ru/company/1cloud/blog/424911
4. Fowler M. Arhitektura korporativnyh programmnyh prilozhenij [Architecture of corporate software applications]. Moscow, Williams, 2006, 544 p. (In Russian).
5. Microsoft's guide to designing an application architecture. Available at: http://apparchguide.ms/Book (accessed 17.11.2019).
6. Matsyachek L. A., Lyong B. L. Prakticheskaya programmnaya inzheneriya na osnove uchebnogo primera [Practical software engineering on the basis of a case study]. Moscow, «BINOM. Laboratory of Knowledge», 2009, 956 p. (In Russian).
7. Nazarov S. V. Arkhitektura i proektirovanie programmnykh sistem [Arkhitecture and design of software systems]. Moscow, Infra-M, 2016. 374 p. (In Russian)
8. Fowler M. Shablony korporativnyh prilozhenij [Corporate Application Templates]. Moscow, Publishing. house «Williams», 2011. 544 p. (In Russian)
9. Karavanov A. V., Ivanov N. D. Arkhitektura programmnogo obespecheniia dlia vysokonadeznyh system. Kosmicheskie apparaty I tehnologii. 2018. T. 2. № 2. S. 100–104
10. Shytko A. M. Proektirovanie mikroservisnoyi arkhitektury programmnogo obespecheniia. Trudy BGTU. Seriya 3. Fisziko-matematicheskie nauki I informatika. 2017. № 9 (200) URL: https://cyberleninka.ru/article/n/proektirovanie-mikroservisnoy-arhitektury-programmnogo-obespecheniya (Data szvernennia 19.11.2019).
11. Elektronnyi resurs (Kogda opravdano ispolszovanie mikroservisnoiy arkhitektury) - rezim dostupu: http://hawkhouse.ru/blog/kogda-opravdano-ispolzovanie-mikroservisnoj-arhitektury/

12. Elektronnyi resurs (Load Balancing) – rezim dostupu: https://microservices.io/patterns/server-side-discovery.html
13. Elektronnyi resurs (Pattern: Externalized configuration) – rezim dostupu: https://microservices.io/patterns/externalized-configuration.html
14. Elektronnyi resurs (Declaring dependencies) – rezim dostupu: https://docs.gradle.org/current/userguide/declaring_dependencies.html
15. Elektronnyi resurs (Service Discovery: Eureka Clients) – rezim dostupu: https://cloud.spring.io/spring-cloud-netflix/multi/multi__service_discovery_eureka_clients.html
16. Elektronnyi resurs (API Gateway / Backends for Frontends) – rezim dostupu: https://microservices.io/patterns/apigateway.html
17. Elektronnyi resurs (Wikipedia. Hypertext Transfer Protocol) - rezim dostupu: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
18. Elektronnyi resurs (Ogranichenie skorosti v Spring Cloud Netflix Zuul) - rezim dostupu: https://www.codeflow.site/ru/article/spring-cloud-zuul-rate-limit
19. Elektronnyi resurs (Docker, chast 2 – rabota s konteiynerami) - rezim dostupu: (https://itproffi.ru/docker-chast-2-rabota-s-kontejnerami/)
20. Lantrat O.V., Sakhno Y.V., Shaposhnikova O.P. Rozroblennya mobilnogo dodatku «Miski parkovky» / Visnyk KHNADU, vyp. 87, 2019, S. 59-66.

Лантрат Олег Володимирович, магістрант Харківського автомобільно-дорожнього університету, тел. +38-057-707-37-43,
e-mail: jonwick1791@gmail.com, Харківський національний автомобільно-дорожній університет, 61002, Україна, м. Харків, вул. Ярослава Мудрого, 25;
Сахно Євгенія Вячеславівна, магістрант Харківського автомобільно-дорожнього університету, тел. +38-057-707-37-43,
e-mail: evgeniasakhno12@gmail.com, Харківський національний автомобільно-дорожній університет, 61002, Україна, м. Харків, вул. Ярослава Мудрого, 25;
Шапошнікова Олена Павлівна, к.т.н., доцент кафедри комп'ютерних технологій і мехатроніки, тел. +38-057-707-37-43,
e-mail: shaposhnikovaep@gmail.com"com, Харківський національний автомобільно-дорожній університет, 61002, Україна, м. Харків, вул. Ярослава Мудрого, 25.

Архитектура приложения для поиска парковки в городе

Аннотация. *Целью работы является разработка архитектуры приложения, которое может использоваться как WEB- или мобильное приложение, в зависимости от потребностей пользователя, а также поможет пользователям найти и воспользоваться свободным парковочным местом. Для достижения цели, используя результаты анализа существующих архитектур, был обоснован выбор микросервисной архитектуры и разработана модель архитектурного решения. Разработана система поддержки внешних конфигураций, настроена система поиска и взаимодействия сервисов, система контейнеризации и быстрого развертывания приложений.*

Ключевые слова: *архитектура, микросервис, паттерн, фреймворк, контейнеризация, разворачивание.*

Лантрат Олег Владимирович, магистрант Харьковського автомобильно-дорожного университета, тел. +38-057-707-37-43,
e-mail: jonwick1791@gmail.com, Харьковский национальный автомобильно-дорожный университет, 61002, Украина, г. Харьков, ул. Ярослава Мудрого, 25;
Сахно Евгения Вячеславовна, магистрант Харьковського автомобильно-дорожного университета, тел. +38-057-707-37-43,
e-mail: evgeniasakhno12@gmail.com, Харьковский национальный автомобильно-дорожный университет, 61002, Украина, г. Харьков, ул. Ярослава Мудрого, 25;
Шапошникова Елена Павловна, к.т.н., доцент кафедры компьютерных технологий и мехатроники, тел. +38-057-707-37-43,
e-mail: shaposhnikovaep@gmail.com, Харьковский национальный автомобильно-дорожный университет, 61002, Украина, г. Харьков, ул. Ярослава Мудрого, 25.

Architecture of the application for searching parking in the city

Problem. *Application development is a rather complicated process, as designing a reliable architecture plays a crucial role. The simplicity and efficiency of the process of developing and maintaining the program depends on how correctly the software architecture is selected because in the process of designing the architecture, the internal properties of the software are determined and details of its external properties are performed based on the formulated business requirements.* ***Purpose***. *The development of an application for finding parking spaces is very relevant and is confirmed by the great need for this service in large cities, especially with an increase in traffic, which proportionally increases with the population of large cities. The aim of the work is to develop an application architecture that can be used as a web or mobile application depending on the needs of the user. The application will help users find and use the free parking space.* ***Methodology.*** *For development of the system, the following tools were used: Spring Core, Spring Cloud, Netflix OSS, Hibernate, Liquibase, Mongo DB, MySQL, and Docker. During the design process, a unified modeling language UML was used.* ***Results***. *To solve this problem, we analyzed the existing architectural solutions, their* *advantages, disadvantages, and selected microservice architecture as the most suitable for developing an application for finding parking in the city of Kharkiv. An external configuration support system has been designed. The system for search, interaction of services, the system of containerization and rapid deployment of applications were configured. A user authentication service has been implemented, an architectural solution model has been developed, and the database structure for the application has been decomposed.* ***Scientific novelty.*** *Based on the results obtained in the research process architectural decisions were made to develop an application that is designed to be used in Kharkiv.* ***Practical significance.*** *The whole range of architectural decisions will be used as the basis for the further development of the application, which will be able to provide a user with information about available parking places, their locations, infrastructure characteristics, etc. in the Kharkiv city.*

Keywords: *architecture, microservice, pattern, framework, containerization, deployment.*

Lantrat Oleg, master Kharkiv National Automobile and Highway University, tel. +38-057-707-37-43, e-mail: jonwick1791@gmail.com,
25, Kharkiv National Automobile and Highway University, Yaroslava Mudrogo str., Kharkiv, 61002, Ukraine;

Sakhno Yevheniia, master Kharkiv National Automobile and Highway University,
tel. +38-057-707-37-43,
evgeniasakhno12@gmail.com,
25, Kharkiv National Automobile and Highway University, Yaroslava Mudrogo str., Kharkiv, 61002, Ukraine;

Shaposhnikova Elena, candidate of technical sciences, Department of Computer Technology and Mechatronics, tel. +38-057-707-37-43,
e-mail: shaposhnikovaep@gmail.com, Kharkiv National Automobile and Highway University,
25, Yaroslava Mudrogo str., Kharkiv, 61002, Ukraine.