КПІ ім. Ігоря Сікорського

Матеріали Першої Всеукраїнської науково-практичної конференції молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2021)

2021

22-26 листопада
Україна, Київ

Оргкомітет конференції

Оргкомітет конференції

Голова організаційного комітету: Е.В. Жаріков – в.о. зав. кафедри ІПІ, д.т.н., професор.

Члени організаційного комітету:

П.І.П.	**Посада**
Е.В.Жаріков	В.о. зав. кафедри ІПІ
О.А. Павлов	Професор кафедри ІПІ
І.В. Стеценко	Професор кафедри ІПІ
І.П. Муха	Доцент кафедри ІПІ
Ю.О. Олійник	Доцент кафедри ІПІ
К.І. Ліщук	Доцент кафедри ІПІ
І.В.Баклан	Доцент кафедри ІПІ

Члени програмного комітету:

П.І.П.	**Посада**
В.В.Гнатушенко	Професор НМетАУ
С.А.Бабічєв	Професор ХДУ
В.І.Литвиненко	Професор ХНТУ
Г.В.Рудакова	Професор ХНТУ
О.В. Гавриленко	Доцент кафедри ІСТ
О.Д.Фіногенов	Доцент кафедри ІПІ
О.І.Лісовиченко	Доцент кафедри ІПІ
Т.А.Ліхоузова	Доцент кафедри ІПІ

Перша Всеукраїнська науково-практична конференція молодих вчених та студентів «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2021). Секція кафедри інформатики та програмної інженерії. Матеріали конференції. – Київ. – 2021. 22–26 листопада 2021р. – 198 с.

У збірник включені тези доповідей, які були представлені на конференції «Інженерія програмного забезпечення і передові інформаційні технології» (SoftTech-2021) в секції інформатики та програмної інженерії. В доповідях розглянуті наукові та методичні питання щодо сучасних аспектів інформатики та обчислювальної техніки.

Редакційна колегія:

Баклан І.В. доцент, к.т.н, доцент кафедри ІПІ НТУУ «КПІ ім. Ігоря Сікорського»,
Муравйова І. М., інженер І категорії кафедри ІПІ НТУУ «КПІ ім. Ігоря Сікорського»

Дизайн титульної сторінки: провідний інженер Майєр З.О. кафедри ІПІ НТУУ «КПІ ім. Ігоря Сікорського»

Зміст

УДК 004.273

МАЛЯРЧУК Р.В.
МУХА І.П.

АРХІТЕКТУРА БАГАТОКЛАСТЕРНОЇ СИСТЕМИ НА БАЗІ МІКРОСЕРВІСУ СИНХРОНІЗАЦІЇ

В статті проведено огляд типової архітектури багатокластерної системи, побудованої згідно моделі GitOps, яка використовує передові принципи DevOps. Показано, що використання management-кластеру не забезпечує достатній рівень відмовостійкості системи та безпеки облікових даних для доступу до workload-кластерів, а його обслуговування потребує значних витрат.

Запропоновано удосконалену архітектуру багатокластерної системи за рахунок використання мікросервісу синхронізації стану workload-кластерів з системою керування версіями Git, що зменшує складність та вартість підтримки багатокластерної системи, підвищує її відмовостійкість та захист облікових даних для доступу до кластерів, пришвидшує доставку програмного продукту.

КЛЮЧОВІ СЛОВА: Архітектура, кластерні системи, GitOps, IaS, система керування версіями, багатокластерні системи, мікросервіс.

The article provides an overview of the typical architecture of a multi-cluster system based on the GitOps model, which uses the advanced principles of DevOps. It is shown that the use of a management cluster does not provide a sufficient level of system resiliency and accounting security to access workload clusters, and its maintenance requires significant costs.

An advanced architecture of multicluster systems is proposed to use a synchronization microservice to enable clusters to work with the Git version control system, which reduces the complexity and cost of support multicluser system, a system of recovery, and protection of credentials for access to clusters speed up the software product.

KEYWORDS: Architecture, cluster systems, GitOps, IaS, version control system, multicluster systems, microservice.

1. Вступ

Життєвий цикл розробки програмного забезпечення (ПЗ) є застосуванням інженерних практик, спрямованих на підтримку працездатності, якості та надійності ПЗ [1].

Вагоме місце серед них займає DevOps (Development & Operations) – сукупність практик, що поєднують процес розробки ПЗ (Development) та його експлуатації (Operation), а також відповідні інструменти автоматизації для підвищення ефективності цих процесів за рахунок безперервної інтеграції і розгортання (Continuous integration & Continuous delivery, CI/CD) та активної взаємодії профільних фахівців.

Однією із таких практик є GitOps, яка здійснює підтримку CD [2]. GitOps дозволяє використовувати найкращі практики в галузі розробки ПЗ, такі як контроль версій, спільна робота, відповідність вимогам, CI/CD, для вирішення задач автоматизації управління конфігураціями інфраструктури та додатків.

Грунтується GitOps на моделі IaS (Infrastructure as a Code), відповідно якої опис і управління інфраструктурою проєкту здійснюється через конфігураційні файли, тобто програмно, а не через ручне редагування конфігурацій на серверах чи інтерактивну взаємодію [3]. Це дає змогу легко робити копії елементів інфраструктури або відновлювати її стан. Популярними інструментами, які дозволяють реалізувати IaS є Terraform, CloudFormation, файли-маніфести Kubernetes.

Однак, на відміну від IaS, GitOps передбачає версійність як програмного, так і конфігураційного коду шляхом розміщення коду інфраструктури у репозиторіях системи керування версіями, зазвичай Git, разом із ПЗ.

Git-репозиторій є єдиним джерелом істини

в GitOps. Єдине джерело істини – це практика структурування вихідого коду і пов'язаних даних таким чином, що кожен елемент даних редагується лише в одному місці (git-репозиторії).

Всі зміни коду застосунку та інфраструктури здійснюються через Git та механізм Pull Request (PR), що дозволяє забезпечити взаємодію усіх розробників, які повинні схвалити запропоновані зміни, у процесі ревю (review) та тестування (testing) [4]. Це дозволить убезпечити систему від випадкових помилок, які можуть повністю її зламати.

Таким чином, Git і поділ доступу в Git стає єдиним місцем (єдиним джерелом істини) не тільки для коду та інфраструктури, але і для політики доступу до різних частин проєкту.

Ще однією відмінністю GitOps-підходу є автоматизація процесу управління інфраструктурою на основі CI/CD. Система сама визначає та усуває різницю між тим, що перебуває в Git та тим, що реально крутиться на серверах. Тобто відбувається процес синхронізації git-репозиторія із станом Kubernetes-кластеру, який представляє собою сукупність вузлів, які запускають контейнеризовані програми.

Для підтримки процесу управління інфраструктурою зазвичай запускають спеціальну програму-агент, яка працює прямо в оточенні системи та змінює її зсередини. Агент є відповідальним за те, щоб здійснювати оновлення даних з системи контролю версій та при наявних змінах оновлювати стан Kubernetes- кластеру.

2. Багатокластерна система в контексті GitOps

Якщо розглядати архітектуру багатокластерної системи в контексті GitOps, то, як було зазначено вище, для роботи такого підходу необхідний gitops-агент, який буде керувати станом кожного з кластерів. Архітектура типової багатокластерної системи зображена на рисунку 1.

Компонентами даної архітектури є один management-кластер (кластер управління) та довільна кількість workload-кластерів (кластерів робочого навантаження) [5].

Management-кластер служить для зберігання облікових даних для кожного із workload-кластерів та підтримка стабільного з'єднання з кожним із цих кластерів.

Основними задачами для workload-кластеру є:

– надання API для management-кластеру, щоб він міг здійснювати керування,

– підтримка gitops-агента, який буде здійснювати оновлення стану на основі інструкцій, отриманих від management-кластеру,

– підтримка власного репозиторія, в якому буде зберігатися поточна конфігурація.

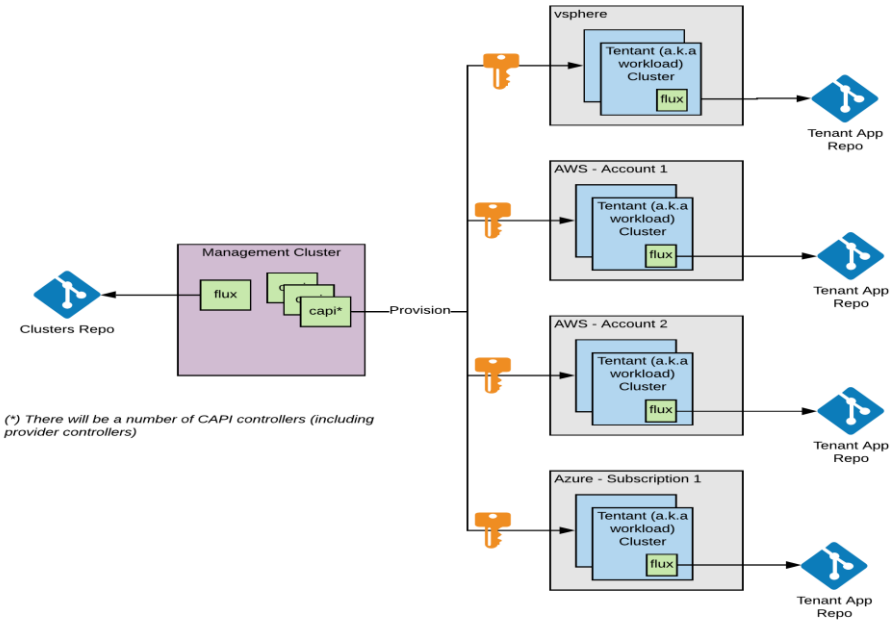

Рис. 1. Типова архітектура багатокластерної системи [5]

Оскільки management-кластер зберігає облікові дані для кожного з workload-кластерів, він є вразливим місцем системи. Якщо зловмисник отримає доступ до management-кластеру, найбільш ймовірно, що він зможе отримати доступ і до відповідних workload- кластерів.

Також, певною вразливістю є те, що кожний із workload-кластерів керується одним і тим же management-кластером. У разі відмови management-кластеру відбудеться відмова усієї системи в цілому.

Окрім того, значним недоліком даної архітектури є необхідність встановлення та запуску на кожному з workload-кластерів gitops-агента, що призводить до збільшення витрат на обслуговування системи. Певних затрат вимагає і підтримка працездатності самого management-кластеру.

3. Архітектура багатокластерної системи на базі мікросервісу синхронізації

Враховуючи наявні недоліки типової архітектури багатокластерної системи, пропонується реалізувати архітектуру даної системи на основі мікросервісу синхронізації, що не вимагає жодної модифікації самих кластерів (рис. 2).

Основними компонентами запропонованої архітектури, які відповідають за бізнес-логіку, є два мікросервіси: мікросервіс синхронізації та мікросервіс роботи з Kubernetes.

Основна задача мікросервісу синхронізації – отримувати зміни вихідного коду з системи керуваннями версіями (Git) та у разі їх наявності сформувати повідомлення на шину даних для того, щоб передати відповідне повідомлення в мікросервіс роботи з Kubernetes.

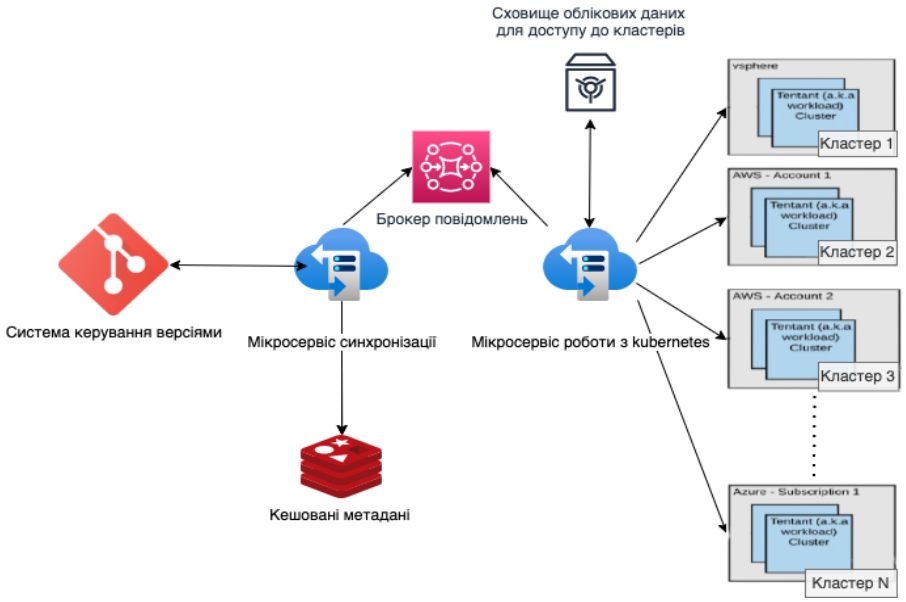

Рис. 2. Архітектура багатокластерної системи з безперервним розгортанням на основі мікросервісу синхронізації

Мікросервіс роботи з Kubernetes служить для оновлення стану кожного з Kubernetes-кластерів. Зв'язок мікросервісу синхронізації та мікросервісу роботи з Kubernetes відбувається через брокер повідомлень.

Основною перевагою запропонованого архітектурного рішення є відсутність management-кластеру для керування workload-кластерами, що зменшує витрати на обслуговування системи. Окрім того, забезпечується вищий рівень безпеки, оскільки облікові дані, необхідні для авторизації до workload-кластерів, зберігаються на спеціалізованому сервері - так званому сховищі облікових даних.

Також важливою перевагою даного рішення є те, що на workload-кластерах не потрібно встановлювати додаткове ПЗ, на відміну від існуючого підходу, який вимагає встановлення gitops-агенту.

Розроблена мікросервісна архітектура є асинхронною, що дозволить уникнути блокування на мікросервісах.

Висновки

Проведено аналіз типової архітектури багатокластерної системи, який дозволив виявити такі її недоліки, як недостатній рівень безпеки облікових даних для доступу до кластерів, низька відмовостійкість, необхідність встановлення та запуску на кожному workload-кластері gitops-агента, що збільшує витрати на обслуговування системи.

Запропоновано удосконалену архітектуру багатокластерної системи на основі використання мікросервісу синхронізації стану workload-кластерів з системою керування версіями Git, що зменшує складність та вартість підтримки системи за рахунок відсутності необхідності адміністрування кожного з кластерів окремо, підвищує захист облікових даних для доступу до кластерів за рахунок використання сховища облікових даних, пришвидшує доставку програмного продукту за рахунок простоти процесу доставки змін та вбудованого кешування.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Jevtic G. What is SDLC? Phases of Software Development, Models, & Best Practices [Електронний ресурс] / Goran Jevtic – Режим доступу до ресурсу: https://phoenixnap.com/blog/software-development-life-cycle
2. Cook, R. (2019). GitOps, an Elegant Tool for Hybrid Cloud Kubernetes.
3. Morris, K. (2016). Infrastructure as code: managing servers in the cloud. " O'Reilly Media, Inc.".
4. Limoncelli, T. A. (2018). GitOps: a path to more self-service IT. Communications of the ACM, 61(9), 38-42.
5. Case R., GitOps and Cluster API: Multi-cluster Manager [Електронний ресурс] / Richard Case – Режим доступу до ресурсу: https://www.weave.works/blog/gitops-and-cluster-api-master-of-masters

УДК 004.852

ХАЛУС О.А.
КОРОЛЬОВА Л.В.

АЛГОРИТМ НАВЧАННЯ ДІАЛОГОВОЇ СИСТЕМИ З ВИКОРИСТАННЯМ ОСТОВИХ ДЕРЕВ

В статті розглянуто алгоритм навчання діалогової системи який включає в себе: підготовку даних для моделі, навчання моделі, та використання навченої моделі для генерації результатів. Представлений опис самої моделі, а саме представлення даних, метрики для навчання та параметри моделі. Також надано порівняння метрик для оцінки результатів генерації діалогу.

The article considers the algorithm of learning a dialog system which includes: preparation of data for the model, learning the model, and the usage of the trained model to generate results. A description of the model itself is presented, namely data representation, training metrics and model parameters. A comparison of metrics for evaluating the results of dialogue generation is also provided.

1. Вступ

В статті розглянуто алгоритм навчання діалогової системи який включає в себе: підготовку даних для моделі, навчання моделі, та використання навченої моделі для генерації результатів. Представлений опис самої моделі, а саме представлення даних, метрики для навчання та параметри моделі. Також надано порівняння метрик для оцінки результатів генерації діалогу.

2. Підготовка даних

Для навчання підходять тексти різноманітної тематики. Якщо в подальшому модель необхідно буде використовувати для якоїсь конкретної специфічної сфери, то слід підготувати тексти з цієї сфери. Проте щоб

регулювати, додаючи додаткові умови або редагуючи вже існуючі. Окрім інтуїтивно зрозумілого задання правил визначення рецептів на основі наявних інгредієнтів, використання експертної системи також передбачає і отримання усіх унікальних функціональностей, що притаманні лиш цій системі, включаючи: моделювання рішення людиною-експертом, представлення ходу думок, що привели до прийняття того чи іншого рішення, тощо.

Список літератури

1. Jackson P. Introduction To Expert Systems – 1998. – Vol.1.1– P. 542
2. Riley G. CLIPS Reference Manual // Basic Programming Guide – 2017 – Vol.1. – P. 3-18
3. Giarratano J., Riley G. Principles and Programming // Expert Systems – 2005 – Vol.1. – P. 288

УДК 004.02

ШВЕЦЬ Е.Я.,
МУХА І.П.

АВТОМАТИЗАЦІЯ МУЛЬТИПЛАТФОРМЕННОЇ РЕАЛІЗАЦІЇ СЕРВІСІВ РОБОТИ З БАЗОЮ ДАНИХ

В даній статті проведено аналіз існуючого процесу створення мультиплатформенного мобільного застосунку та запропоновано метод автоматизованого створення програмних компонетів, що реалізують сервіси роботи з БД, який передбачає транспіляцію вхідних описів сутностей бази даних та налаштувань (DAO, CRUD, DI), представлених на спеціально розробленій предметно-орієнтованій мові, у SQL- та Kotlin-компоненти загального коду мультиплатформенного мобільного проєкту.
Ключові слова: Первинний мультиплатформенний мобільний проєкт, Kotlin Multiplatfrom Mobile, автоматизація, мультиплатформенний мобільний застосунок.

This article analyzes the existing process of creating a multi-platform mobile application and proposes a method of automated creation of software components that implement database services, which involves translating input descriptions of database entities and settings (DAO, CRUD, DI), presented on a specially designed subject. oriented language, in SQL and Kotlin components of the common code of a multiplatform mobile project.
Keywords: Primary multiplatform mobile project, Kotlin Multiplatfrom Mobile, automation, multiplatform mobile application.

1. Вступ

Останнім часом все більшої популярності у світі мобільної розробки набуває мультиплатформенна розробка мобільних додатків під ОС Android та iOS. Існує ряд технологій, які дозволяють це зробити, зокрема, Xamarin, React Native, Flutter, Kotlin Multiplatform Mobile (KMM) [1]. Найбільш перспективною серед них є KMM, яка має помітні переваги у порівнянні із іншими альтернативними технологіями.

Оскільки версії одного і того ж застосунку для Android та iOS часто мають багато спільного, наприклад, бізнес-логіку застосунку (зокрема, код, який відповідає за автентифікацію, керування даними, аналітику), то для різних платформ цілком природним є використання загального коду, який реалізує відповідний функціонал.

KMM SDK якраз і дозволяє оптимізувати розробку мультиплатформенного мобільного застосунку шляхом одноразового написання блоку універсального коду, а потім спільного використання цього коду на різних платформах. Платформний же код використовується тільки там, де це необхідно — для реалізації інтерфейсу або під час роботи з платформо-залежними API.

Використання KMM хоча і прискорює процес створення мультиплатформенних мобільних застосунків за рахунок спільного

використання загального коду, однак написання цього коду теж потребує значних витрат часу.

При цьому бізнес-логіка предметної області передбачає реалізацію певних етапів, характерних для будь-якого мульти-платформенного мобільного застосунку. Автоматизувавши процес створення таких уніфікованих фрагментів коду, можна значно прискорити як процес створення загального коду, так і загалом процес створення мультиплатформенного мобільного застосунку.

2. Наявний підхід до створення мультиплатформенного мобільного застосунку з використанням технології КММ

Наявний підхід до розробки мультиплатформенного мобільного застосунку з використанням технології КММ передбачає винесення типових для декількох платформ дій, як правило, бізнес-логіки, у єдиний універсальний блок коду – загальний код, який в подальшому спільно використовуватиметься в одних частинах програми, залишаючи інші частини (наприклад, UI-шар) повністю нативними. Загальний код реалізується на мові Kotlin і міститься у загальному модулі (shared-модулі) КММ-проєкту.

Спільна бізнес-логіка мультиплатформенного мобільного застосунку може включати сервіси для роботи з БД, сервіси для роботи з мережею, моделі даних. Також до неї можуть входити архітектурні компоненти програми, які безпосередньо не включають UI, але з ним взаємодіють (ViewModel, Presenter тощо).

Більшість мобільних застосунків локально зберігають дані на клієнтських пристроях в БД SQLite [2] - спеціалізованій, оптимізованій під мобільні платформи програмній бібліотеці, яка реалізує систему управління реляційною БД [3]. Тому реалізацію сервісів роботи з БД можна вважати типовою функціональністю при розробці мультиплатформенних мобільних застосунків.

Створення відповідної БД потребує виконання рутинних дій по реалізації сутностей предметної області, стандартних операцій обробки таблиць БД - CRUD-операцій (Create Read, Update, Delete) [4], інтерфейсу взаємодії застосунку з БД - DAO-класів (Data Access Object), класу управління БД. Сутності БД і CRUD-операції реалізуються на мові SQL, DAO-класи, клас управління БД - на мові Kotlin.

Окрім того, в рамках shared-модуля можна реалізувати механізм впровадження залежностей (Dependency Injection, DI). Це дасть змогу створювати і обробляти залежності (об'єкти інших класів, які можна використовувати як сервіс) поза залежними класами і, як наслідок, спростить модифікацію класів.

З розвитком КММ стали з'являтися мультиплатформенні фреймворки, зокрема, SQLDelight – фреймворк для роботи з SQLite, Koin [5] – DI-фреймворк для впровадження залежностей. Однак їх функціональність не дозволяє в повній мірі автоматизувати ні процес створення shared-модуля, ні окремих його складових.

Зокрема, фреймворк SQLDelight [6] підтримує лише генерацію класів даних (data-класів) для сутностей БД на мові програмування Kotlin [7] і автоматично реалізує взаємодію мобільного застосунку з БД.

Левова ж частина роботи по створенню загального коду (shared-модуля) все ще виконується в ручному режимі.

3. Первинний мультиплатформенний мобільний проєкт

Оскільки, типовим функціоналом при розробці мобільних застосунків є реалізація сервісів роботи з БД, актуальною задачею є автоматизація саме цих процесів. При цьому інші сервіси передбачається реалізовувати в ручному режимі.

З урахуванням зазначеного, програмні компоненти загального коду, які є реалізацією сервісів роботи з БД, назвемо базовим кодом, а частину shared-модуля, що містить базовий код - первинним мультиплатформенним мобільним проєктом

Фактично, процес створення shared-модуля мультиплатформенного мобільного застосунку пропонується здійснювати у два етапи: генерація базового коду (первинного проєкту), подальша реалізація структурних

компонент вручну (в середовищі Android Studio).

4. Метод автоматизованого створення первинного мультиплатформенного мобільного проєкту

Як зазначалося, реалізація сервісів роботи з БД передбачає опис сутностей БД та налаштувань (DAO, CRUD, DI) на мові Kotlin та SQL.

Автоматизувати процес отримання відповідних програмних компонент пропонується шляхом транспіляції вхідних описів (сутностей БД та налаштувань), представлених на спеціально розробленій предметно-орієнтованій мові, у коди на мові Kotlin та SQL.

Загалом, метод автоматизованого створення первинного мультиплатформенного мобільного проєкту можна представити наступною послідовністю кроків:

Крок 1. Опис вхідних даних на предметно-орієнтованій мові (сутностей БД та налаштувань DAO, CRUD, DI).

Крок 2. Лексичний аналіз вхідного тексту. Формування списку зчитаних токенів на основі правил визначення їх типів.

Крок 3. Синтаксичний аналіз списку зчитаних токенів і формування дерева (AST) на основі правил граматики проблемно-орієнтованої мови. Парсинг виконується з використанням методу низхідного синтаксичного LL-аналізу.

Крок 4. Прямий обхід (NLR) синтаксичного дерева та збереження даних (сутностей БД та налаштувань DAO, CRUD, DI) у тимчасові об'єкти, реалізовані у вигляді класів.

Крок 5. Валідація тимчасових об'єктів на основі попередньо сформованих вимог до валідації (методів класу Validator).

Крок 6. Генерація SQL- та Kotlin-коду шляхом вставки даних з тимчасових об'єктів у спеціально створені шаблони коду.

Схему транспіляції вхідних описів сутностей БД та налаштувань в базовий загальний код первинного мультиплатформенного мобільного застосунку можна представити наступним чином:

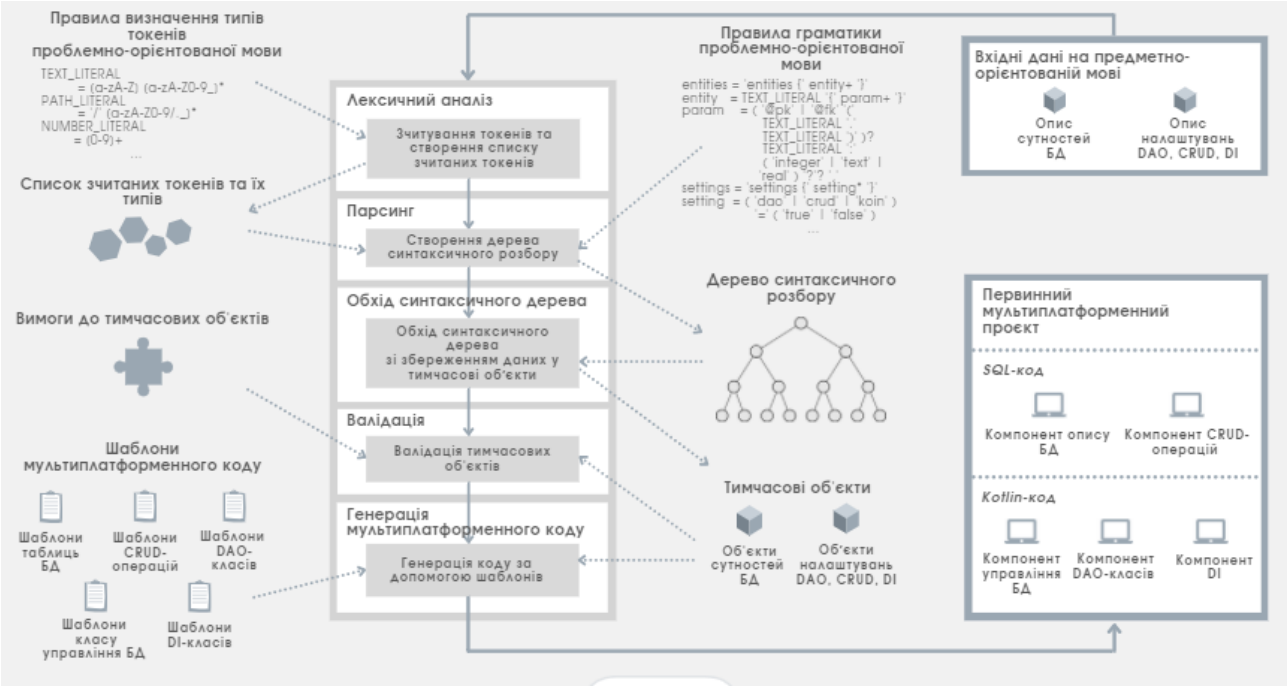

Рис.1. Схема транспіляції вхідного опису у загальний код

Запропонований метод пришвидшує як процес створення базового загального коду, так і загалом процес створення мультиплатформенного мобільного застосунку.

Висновки

В даній статті проведено аналіз існуючого процесу створення мультиплатформенного мобільного застосунку. Показано, що він потребує великої кількості рутинних операцій,

зокрема при реалізації сервісів роботи з БД, які можнка частково автоматизувати.

Запропоновано метод автоматизованого створення первинного мультиплатформенного мобільного проєкту, який передбачає транспіляцію вхідних описів (сутностей БД та налаштувань DAO, CRUD, DI), представлених на спеціально розробленій предметно-орієнтованій мові, у SQL- та Kotlin-компоненти загального коду мультиплатформенного мобільного проєкту.

Список літератури

1. Kotlin Multiplatform Mobile [Електронний ресурс]. – Режим доступу: https://kotlinlang.org/docs/kmm-overview.html
2. Using SQLite Jay A. Kreibich– M.: «O'Reilly Media, Inc.», 2010
3. Learning SQL: Master SQL Fundamentals by Alan Beaulieu – M.: «O'Reilly Media», 2009
4. CRUD [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/CRUD
5. Framework Koin [Електронний ресурс]. – Режим доступу: https://insert-koin.io/
6. SQLDelight [Електронний ресурс]. – Режим доступу: https://www.kotlinresources.com/library/sqldelight/
7. Programming Kotlin: Create Elegant, Expressive, and Performant Jvm and Android Applications by Venkat Subramaniam – W.: «Dover Publications» 2019

УДК 004.912

ГОНЧАРЕНКО Є. І.

ПРАКТИЧНЕ ЗАСТОСУВАННЯ СИМЕТРИЧНИХ АЛГОРИТМІВ ШИФРУВАННЯ

Криптографія з симетричним ключем – це будь-який криптографічний алгоритм, який базується на спільному ключі, який використовується для шифрування або дешифрування тексту/шифротексту, у контракті з криптографією з асиметричним ключем, де ключі шифрування та дешифрування відрізняються.

КРИПТОГРАФІЧНІ АЛГОРИТМИ, БЕЗПЕКА, ШИФРУВАННЯ

Symmetric key cryptography is any cryptographic algorithm based on a shared key that is used to encrypt or decrypt text / ciphertext in a contract with asymmetric key cryptography where the encryption and decryption keys are different.

CRYPTOGRAPHIC ALGORITHMS, SECURITY, ENCRYPTION

1. Введення

Алгоритми з симетричним ключем (іноді їх називають алгоритмами секретного ключа) використовують один ключ як для застосування криптографічного захисту, так і для зняття або перевірки захисту (тобто один і той самий ключ використовується для криптографічної операції та її інверсії). Наприклад, ключ, який використовується для шифрування даних (тобто для застосування захисту), також використовується для розшифрування зашифрованих даних (тобто для зняття захисту). У разі шифрування вихідні дані називаються відкритим текстом, а зашифрована форма даних називається шифротекстом. Щоб дані залишалися захищеними, ключ повинен зберігатися в секреті.

2. Використання симетричних алгоритмів шифрування

При використанні алгоритму із симетричним ключем необхідно створити унікальний ключ для кожного криптографічного зв’язку і для кожної мети (наприклад, шифрування, аутентифікація цілісності даних та обгортання ключів). Технічно один і той же ключ може використовуватися для кількох цілей, коли використовується той самий алгоритм, але це зазвичай недоцільно, оскільки використання одного ключа для двох різних криптографічних процесів (наприклад, HMAC і отримання ключа з використанням однієї і тієї ж хеш-функції) може послабити