УДК 004.273

Зосімович Д. М., магістрант, група ІПЗм-20-1
Науковий керівник: Морозов А. В., канд. техн. наук, доцент
Державний університет «Житомирська політехніка»

АНАЛІЗ СПЕЦИФІКИ РЕАЛІЗАЦІЇ АРХІТЕКТУР ПРОГРАМНИХ СИСТЕМ

Проблема збільшення складності робіт з розробки програмних продуктів і, як наслідок, коштів на їх розробку зі збільшенням кількості функціоналу, який вже реалізовано, є відомою, очікуваною та прог-нозованою на сьогоднішній день.

Найбільш розповсюдженими на сьогоднішній день є наступні системні архітектури:

- мікро-сервісна;
- сервісно-орієнтована;
- безсерверна;
- монолітна.

На прикладі процесу розробки системи розкладу для Державного університету «Житомирська політехніка» розглянемо реалізацію останніх 3 з них.

Під час першої фази розробки система мала монолітну архітектуру. На віртуальній машині було розміщено копію бази університету (MySql) та серверний додаток. Мобільний додаток виконував HTTP запити для отримання розкладу у форматі JSON та подальшого його відображення користувачеві.

З часом виникла необхідність розробки функціоналу для управління розкладом в базі. Вважатимемо її фазою 2.

На той момент було розв’язано наступні фундаментальні проблеми:

- чи продовжувати розробку в рамках моноліту?
- які технології використати для реалізації фази 2?
- як реалізувати СОА в контексті моноліту з мінімальними затратами часу?

Виконувати міграцію системи з віртуальної машини у хмарне середовище AzureCloud або AmazonWebServices було очевидно дорого, тим паче машина вже знаходилася в інфраструктурі хмарного стартапу Scaleway. Було прийнято рішення розмістити додаток панелі адміністрування на машині поряд з монолітом.

Однією з переваг СОА є можливість масштабування окремих сервісів. Умови використання додатку було проаналізовано на момент проектування фази 1. У результаті було виявлено, що система не потребує спеціальної реалізації можливостей горизонтального масштабування в майбутньому, що дозволило уникнути проектування відповідних засад.

Через деякий час після завершення фази 2 в системі було виявлено декілька суттєвих проблем, основною з яких була проблема її подальшої підтримки та розробки. Через те, що система складається з декількох компонентів, необхідно підтримувати та дороблювати кожний. Ресурси для виконання таких дій з боку університету були і є досить обмеженими, а отже постало питання доцільності і можливості використання подібного програмного рішення.

Проаналізувавши існуючий ринок безсерверних сервісів, а також переваги, які вони надають, було прийнято одноосібне рішення мігрувати існуючу систему управління розкладом на безсерверну архітектуру. Розробку в рамках даної міграції вважатимемо фазою 3. Поста-чальником послуг було обрано GoogleFirebase.

Використання GoogleFirebase змушувало вдатися до міграції іс-нуючої MySql бази даних. Під час першої спроби базу було мігровано зі збереженням існуючої третьої форми нормалізації. Даний підхід виходив за рамки безкоштовної квоти, яку надавав сервіс на виконання запитів читання, проте квота на сховище була використана лише на 30%. Наявність вільного місця дозволила частково пожертвувати ним задля вирішення проблеми надмірної кількості запитів шляхом денормалізації бази даних способом шардованого розбиття даних.

Упровадження зазначеного підходу к контексті нетабличної NO-SQL бази GoogleFirebase мало власну інтерпретацію. Так, аналіз ва-ріантів використання додатку дозволив дефініювати формат ключів партицій для множини занять – «сутність/ідентифікатор», наприклад «group/24».

Користуючись перевагами, які надають NO-SQL рішення для збереження даних, а саме можливістю збереження множини даних в рамках одного запису, кожна партиція мала всі необхідні дані для задоволення варіанту використання «пошук та перегляд розкладу за групою», а отже 1 перегляд розкладу для 1 групи виконував лише 1 запит на читання рядка, при цьому використання квоти сховища виросло до 70%.

Провівши аналіз архітектур програмного забезпечення, визначимо, що конкретна реалізація кожного з їх видів є досить масштабним та кропітким процесом, який вимагає ретельного планування та попе-реднього аналізу. Архітектурні помилки та фактори, що не були взяті до уваги, можуть призвести до нівелювання попереднього прогресу або значної переробки існуючої системи. Це призводить до значного збільшення загальної вартості розроблюваної системи.