ISSN 2707-1898 (print)
Український журнал інформаційних технологій
Ukrainian Journal of Information Technology
http://science.lpnu.ua/uk/ujit
https://doi.org/10.23939/ujit2024.02.090
Article received 18.05.2024 p.
Article accepted 19.11.2024 p.
UDC 004.77

Correspondence author
A. Ya. Pryhoda
a.pryhoda@knute.edu.ua

А. Я. Пригода
Державний торговельно-економічний університет, Київ, Україна

МІГРАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З МОНОЛІТНОЇ АРХІТЕКТУРИ НА АРХІТЕКТУРУ МІКРОСЕРВІСІВ ЯК СПОСІБ ЗАХИСТУ CRM-СИСТЕМ

Висвітлено дослідження, зосереджене на захисті CRM-систем в умовах переходу програмного забезпечення, яке становить основну частину CRM-систем, із монолітної архітектури на архітектуру на основі мікросервісів. У статті досліджено стратегію міграції з використанням шаблону Strangler Fig, яка полегшує поступове впровадження мікросервісів, зберігаючи сумісність із наявним монолітом. Ключовим аспектом стратегії міграції є вибір структури, такої як Framework for Microservices Migrations (FMM), яка стандартизує методи розроблення та допомагає у декомпозиції монолітів на керовані компоненти. Зв'язок між мікросервісами та монолітом полегшується через REST API, що забезпечує безперебійну взаємодію. Інтеграція мікросервісів досягається за допомогою впровадження DTO (Data Transfer Objects – об'єктів передавання даних) і шлюзів API, що забезпечує плавний обмін даними між компонентами. У статті підкреслено важливість керування залежностями та конфігурації в мікросервісах, наголошено на необхідності інкапсуляції та автономності. Крім того, у здійсненому дослідженні розглянуто використання шаблонів Curcuit Breaker для ефективного оброблення збоїв і підтримки стійкості системи під час процесу міграції. Мікросервіси потребують надійних методів автентифікації, таких як криптографічні ключі API та токени. Security Assertion Markup Language (SAML) або OpenID з OAuth 2.0 – рішення, запропоновані для підвищення безпеки даних. Централізована архітектура для політик авторизації має вирішальне значення для регулювання доступу до всіх мікросервісів, тоді як стандартизована автентифікація на основі маркерів забезпечує взаємодію незалежно від платформи. Однак децентралізоване застосування авторизації в кожній точці входу мікросервісу може призвести до непослідовних політик безпеки. Крім того, для доступу до API, що обробляють конфіденційні дані, потрібні маркери автентифікації із цифровими підписами або з надійних джерел. Політики контролю доступу повинні бути ретельно визначені та запроваджені, враховуючи термін дії маркера та обмеження обсягу. Анотація підкреслює важливість централізованого керування доступом і проблеми, пов'язані з анонімними з'єднаннями в обхід шлюзів API.

Ключові слова: процес міграції, монолітна архітектура, мікросервіси, CRM-системи, керування залежностями, оброблення конфігурації, стійкість системи, інкапсуляція.

Вступ / Introduction

Сьогодні керівники бізнесу, компанії повинні усвідомлювати виклики, які зумовлює все інтенсивніша конкуренція. У таких умовах одним із найважливіших факторів для багатьох компаній є лояльність клієнтів, якої можна досягти за допомогою концепції управління взаємовідносинами з клієнтами. Завдяки управлінню взаємовідносинами з клієнтами з'являється можливість зібрати повну інформацію про клієнтів, від центрального реєстру адрес до складних знань про клієнта, доступних кожному працівнику.

Проте, з погляду програмування, традиційні системи CRM стикаються із проблемами щодо безпеки та конфіденційності даних. Це пов'язано з тим, із урахуванням поточної практики, під час розроблення нового програмного забезпечення прийнято починати процес зі створення монолітної архітектури. Однак використання монолітної архітектури має низку недоліків, оскільки функціонал програмного забезпечення істотно розширюється із плином часу. Ці недоліки особливо очевидно проявляються у складності управління, проблемі обслуговування та нездатності монолітних додатків впоратися із ситуаціями, коли кількість користувачів перевищує потужність сервера. Для вирішення цих проблем було розроблено стиль архітектури мікросервісів. Мікросервіси незалежно масштабуються, встановлюються незалежно та їх легше підтримувати.

Однак міграція монолітної структури в мікросервісну пов'язана з деякими складностями. Багато факторів, наприклад вибір стратегії міграції та підхід до ідентифікації послуг, відіграють у цьому випадку важливу роль.

Об'єкт дослідження – системи CRM.

Предмет дослідження – мікросервісна архітектура систем CRM як їх подальший захист.

Мета роботи – дослідження методології міграції програмного забезпечення задля реалізації захисту систем управління взаємовідносинами з клієнтами (Customer Relationship Management – CRM).

Для досягнення зазначеної мети визначено такі основні *завдання дослідження*:

- поступова міграція системи CRM з монолітної на мікросервісну архітектуру з частковим зв'язком з монолітом;
- використання шаблонів Curcuit Breaker для оброблення збоїв і підтримки стійкості системи під час та після процесу міграції;
- використання протоколів автентифікації OpenID у поєднанні з OAuth 2.0.

Матеріали та методи дослідження. Аналіз: у статті детально проаналізовано, як різні механізми автентифікації та авторизації можуть бути реалізовані в системах CRM в умовах архітектури мікросервісів, перевірено ефективність різних стратегій, таких як ключі API, маркери та централізовані політики доступу.

Аналіз останніх досліджень і публікацій. У науково-дослідницькому просторі сьогодення з'являються роботи, які стосуються винаходу та аналізу методології розроблення алгоритмів захисту систем управління взаємовідносинами з клієнтами (Customer Relationship Management – CRM) за допомогою використання архітектури мікросервісів.

У ході роботи [1] здійснено дослідження впливу мікросервісів і контейнеризації на ефективність ресурсів, масштабованість і загальну продуктивність систем SAAS CRM. Експериментальні результати продемонстрували істотне підвищення ефективності використання ресурсів, оскільки CRM-системи на основі контейнерних мікросервісів демонстрували нижчий рівень використання ЦП і пам'яті порівняно із монолітними аналогами. Цю підвищену ефективність використання ресурсів можна пояснити притаманною мікросервісам модульністю та детальною масштабованістю, що дає змогу краще розподіляти ресурси та керувати ними. Крім того, такі платформи контейнеризації, як Docker, забезпечують легку віртуалізацію та ізоляцію ресурсів, що додатково сприяє оптимізованому використанню ресурсів. Масштабованість, ще один важливий аспект систем CRM, також забезпечила помітні покращення в експериментальній установці. Системи CRM на основі мікросервісів продемонстрували кращу горизонтальну масштабованість, що дало їм змогу справлятися зі збільшеним навантаженням, просто додавши більше екземплярів необхідних служб. Це різко контрастує із монолітними програмами, які часто потребують повномасштабної реплікації всієї програми, щоб впоратися зі збільшенням навантаження. Крім того, платформи оркестрування контейнерів, такі як Kubernetes, спрощують розгортання та керування контейнерними програмами, забезпечуючи плавне автоматичне масштабування мікросервісів на основі попиту в реальному часі.

У документі [2] запропоновано рішення для налаштування для мультитенантного SaaS із використанням мікросервісів. Було запроваджено неінтрузивний підхід, який може бути практичнішим для промисловості. Запропоновано узагальнену еталонну архітектуру, яка дає змогу налаштовувати SaaS із кількома клієнтами за допомогою мікросервісів.

Крім того, варто виокремити праці таких науковців: Шарда Кумарі [1], Сонг Хуей, Нгуен Фу, Човель Франк [2], Нанхе Проф, Нанхе Мс [3], Маліма Боніфас, Мбого Кріспін [4], Резер Марсель [5], Адріель Кевін, Сударман Мухаммад, Сміт Берон, Мустікасарі Фараніта [6], Вем Лінус, Мшелмбула Джой, Очігбо Абель, Агвом-Панле Рут [7], Цзінцзін Сюе [8], Двіведі Рам Кумар, Лохмор Шейлі, Діксіт Ринку Шарма, Сахіба Зайнаб, Наік Сатьяпра, Тахердуст Хамед [10], Льою Гжегож, Боченек Магдалена [11], Наїм Арші [12], Раджагукгук Вілсон, Самосір Омас, Раджагукгук Джосія, Раджагукгук, Хасіана [13], Гболагаде Адекола, Адейемі Олажумоке [14] Яшодха Др, Латітха Др [15], Пріор Даніель [16], Манахемен Проді [16] та інших.

Проте, беручи до уваги зазначену вище наукову документацію, питання, пов'язане з методологією розроблення алгоритмів захисту систем управління взаємовідносинами з клієнтами (Customer Relationship Management – CRM) за допомогою використання архітектури мікросервісів, все ще залишається недостатньо дослідженим та потребує подальшого опрацювання.

Результати дослідження та їх обговорення / Research results and their discussion

Поступова міграція монолітної архітектури в мікросервісну відбувається за допомогою шаблону Strangler Fig. Це дає змогу додавати нові функції до програми та вилучати наявні функції з моноліту як мікросервіси, тоді як моноліт, що залишився, може працювати паралельно із мікросервісами. Це означає, що для цілей цієї роботи можна використовувати єдиний репозиторій. Шаблон спільної бази даних використовується для міграції бази даних. Розділення таблиць, які використовують декілька мікросервісів, та їх інтеграція в інші таблиці мікросервісів не є варіантом для застосунку, орієнтованого на обслуговування клієнтів та продажі, оскільки це призведе до надто складної міграції даних для наявних клієнтів, які хочуть перейти на рішення мікросервісів. Тому важливо або дублювати таблиці бази даних, або використовувати шаблон спільної бази даних. Зв'язок між мікросервісами та між монолітом і мікросервісами відбуватиметься через REST API. Він використовує стандартні методи HTTP та формати даних, такі як JSON або XML, щоб полегшити зв'язок між клієнтами та серверами.

Оскільки міграція ґрунтується на шаблоні Strangler Fig, кінцеві пристрої спілкуються із рештою моноліту. Цей моноліт містить інтерфейси для вже вилучених мікросервісів, які викликають відповідний REST API мікросервісу. Відповідно до шаблону Strangler, модуль у розташуванні вилученої служби реструктурується так, що він викликає REST API видобутого сервісу. Шлюз API діє як точка входу для кінцевих пристроїв і пересилає запити до відповідних сервісів. Щоб шлюз API міг адресувати правильні сервіси, реалізується реєстр послуг, у якому можуть реєструватися активні мікросервіси.

Щоб забезпечити відсутність залежності від інших сервісів або моноліту, кожен мікросервіс керує власни-

ми бібліотеками, щоб запобігти каскаду збоїв. Щоб переконатися, що всі мікросервіси реалізують правильні основи кожного мікросервісу, створено шасі мікросервісу та шаблон мікросервісу. Для параметрів, які можуть впливати на кілька мікросервісів або які не повинні бути жорстко закодовані, створюється зовнішній файл конфігурації, з якого мікросервіси можуть зчитувати свої параметри.

Сервіс Crm.PdfGeneration видобувається за допомогою шаблону Strangler Fig. У тому місці, де цей модуль містився у моноліті, модуль доповнюється модулем, який пересилає запит до сервісу. Створюється новий проєкт "ASP.NET Core Web API" у рішенні-репозиторії CRM. Потім у проєкт копіюються класи, які утворюватимуть мікросервіс. У цьому випадку класи IPdfService.cs і PdfService.cs копіюються з Crm.Library. Оскільки для цих класів характерні залежності від Crm.Library, а для тих залежностей – інші залежності, залежності також копіюються у проєкт Crm.PdfGeneration, оскільки кожен мікросервіс повинен сам керувати своїми бібліотеками та залежностями. Зовнішні залежності встановлюються через NuGet Package Manager. Щоб правильно ініціалізувати проєкт, на додаток до Program.cs також створюється клас Startup.cs. У класі Startup.cs, серед іншого, зчитуються HttpClient і контролер API, як показано у вихідному коді в табл. 1

Табл. 1. Ініціалізація файлів конфігурації, PdfService та контролерів у класі Startup мікросервісу PdfGeneration / Initialization of configuration files, PdfService and controllers in the Startup class of the PdfGeneration microservice

```
44 public void ConfigureServices(IServiceCollection services)
45 {
46 serviceSettings =
configuration.GetSection(nameof(ServiceSettings)).Get<ServiceSettings>();
47 communicationSettings =
configuration.GetSection(nameof(CommunicationSettings))
.Get<CommunicationSettings>();
48 services.AddSingleton<IPdfService, PdfService>();
49 services.AddControllers();
50 services.AddTransient<PdfService>();
51 }
```

Щоб гарантувати, що всі необхідні значення доставляються разом зі зв'язком, створюються класи DTO, які містять необхідну інформацію. PdfDTO використовується для передавання згенерованого PDF-файлу назад абоненту як масив байтів. Решта DTO містять параметри, важливі для різних викликів методів PdfService.cs. Потім створюється клас PdfController.cs для зв'язку через REST API. У цьому класі метод Get генерується для кожного публічного методу класу PdfService.cs. Кожен із цих методів get отримує спеціальний DTO як параметр, що передається через тіло запиту виклику REST API. Спочатку виклик API керувався за допомогою параметрів /Html2Pdf/{html}/{headerMargin}/{footerMargin}. Пізніше почало здійснюватися передавання параметрів через тіло запиту. Метод Get викликає відповідний метод PdfService із параметрами DTO. Один із цих методів Get показано у вихідному коді в табл. 2.

PDFService, поданий у табл. 4, отримує значення від контролера, рядок 18. Генерація виконується CefToPdfConverter у рядках 25–29. Нарешті PDF-файл повертається як масив байтів, рядок 30, або, у випадку вилучення часу очікування, повертається нульове значення, рядок 35.

Табл. 2. Метод Get для виклику методу Html2Pdf мікросервісу PdfGeneration / Get method to call the Html2Pdf method of the PdfGeneration microservice

```
6 namespace Crm.PdfGeneration.Controllers
7 {
8 [ApiController]
9 [Route("api/Pdf")]
10 public class PdfController : ControllerBase
11 {
12 private readonly IPdfService pdfService;
13 public PdfController(IPdfService pdfService)
14 {
15 this.pdfService = pdfService;
16 }
17 [HttpGet("Html2Pdf/{dto}")]
18 public PdfDTO GetHtml2Pdf([FromBody] SendPdfDTO dto)
19 {
20 var pdf = this.pdfService.Html2Pdf(dto.html, dto.headerMargin,
dto.footerMargin);
21 var responseDTO = new PdfDTO(Guid.NewGuid(), pdf);
22 return responseDTO;
```

Табл. 3. Клас PdfModule.cs мікросервісу PdfGeneration / The PdfModule.cs class of the PdfGeneration microservice

```
35 namespace Crm.PdfGeneration.AutoFac
36 {
37 public class PdfModule : Module
38 {
39 protected override void Load(ContainerBuilder builder)
40 {
41 base.Load(builder);
42 builder.Register(c => c.Resolve<ISiteService>().CurrentSite).As<Site>();
43 builder.RegisterType<CefToPdfConverter>().As<ICefToPdfConverter>();
44 builder.Register(
45 c =>
46 {
47 var appSettingsProvider = c.Resolve<IAppSettingsProvider>();
48 var cefToPdfPath =
appSettingsProvider.GetValue(ServiceSettings.CefToPdfPath);
49 var fileName = Path.Combine(cefToPdfPath, "CefToPdf.exe");
50 return (IDeployment)new StaticDeployment(cefToPdfPath);
51 }).As<IDeployment>();
52 }
53 }
54 }
```

Табл. 4. Метод Html2Pdf служби PdfService у мікросервісі PdfGeneration /
The Html2Pdf method of the PdfService service in the PdfGeneration microservice

```
1 namespace Crm.PdfGeneration.Services
2 {
3 public class PdfService : IPdfService
4 {
5 protected readonly ILog logger;
6 protected readonly ICefToPdfConverter converter;
7 private Site site;
8 public PdfService(Site site, ILog logger, ICefToPdfConverter converter)
9 {
10 this.site = site;
11 this.logger = logger;
12 this.converter = converter;
13 }
14 public virtual byte[] Html2Pdf(string html, double headerMargin = 0,
double footerMargin = 0)
15 {
16 var result = converter.HtmlToPdf(html, new CefToPdfSettings
17 {
18 MarginTop = headerMargin,
19 MarginBottom = footerMargin
20 });
21 return result;
22 }
23 }
24 }
```

Для виклику мікросервісу PdfGeneration зв'язок реалізується у проєкті Crm.Library. Оскільки мікросервіс PdfGeneration працює на іншій IP-адресі з іншим портом, клас PdfServiceClient.cs, показаний у табл. 5, реалізовано у Crm.Library, який регулює виклик зовнішнього REST API. SendPdfDTO генерується із переданих параметрів, рядок 23. Потім DTO серіалізується як об'єкт JSON, рядок 24. Потім генерується URI для виклику API, рядок 25. Для цього використовується BaseAddress у випадку локального виконання з визначеним портом https://localhost:7114, зчитаним із файлу конфігурації та пов'язаним зі шляхом методу API Get. Потім генерується запит із тілом виклику, рядки 26-31, і надсилається рядок 32. Нарешті, повернутий PdfDTO витягується та повертається, рядки 33 та 34. Цей клас PdfServiceClient.cs було ініціалізовано в класі Startup.cs.

Щоб запити, надіслані до оригінального класу PdfService.cs проєкту Crm.Library, пересилалися до мікросервісу PdfGeneration, тіла методів оригінального класу адаптовано так, щоб вони викликали відповідний метод класу PdfServiceClient.cs, який є запитом перенаправлення до відповідного API REST мікросервісу PdfGeneration. У табл. 6 показано, як змінюється оригінальне тіло методу. Тепер викликається метод GetPdfAsync pdfServiceClient і передаються отримані параметри, рядок 14.

Табл. 5. Виклик API для методу Html2Pdf мікросервісу PdfGeneration у класі PdfServiceClient.cs проєкту Crm.Library / API call for the Html2Pdf method of the PdfGeneration microservice in the PdfServiceClient.cs class of the Crm.Library project

```
12 namespace Crm.Library.Services
13 {
14 public class PdfServiceClient
15 {
16 private readonly HttpClient httpClient;
17 public PdfServiceClient(HttpClient httpClient)
18 {
19 this.httpClient = httpClient;
20 }
21 public async Task<PdfDTO> GetPdfAsync(string html, double headerMargin,
double footerMargin)
22 {
23 var dto = new SendPdfDTO(html, headerMargin, footerMargin);
24 var json = JsonConvert.SerializeObject(dto);
25 var uri = new Uri(httpClient.BaseAddress.ToString() +
$"api/Pdf/Html2Pdf/dto");
26 var request = new HttpRequestMessage
27 {
28 Method = HttpMethod.Get,
29 RequestUri = uri,
30 Content = new StringContent(json, Encoding.UTF8, "application/json")
31 };
32 var response = await httpClient.SendAsync(request);
33 var pdf = await response.Content.ReadFromJsonAsync<PdfDTO>();
34 return pdf;
35 }
36 }
37 }
```

Табл. 6. Адаптація тіл методів класу PdfService.cs в Crm.Library / Adaptation of PdfService.cs class method bodies in Crm.Library

```
1 namespace Crm.Library.Services
2 {
3 public class PdfService : IPdfService
4 {
5 protected readonly ILog logger;
6 private readonly PdfServiceClient pdfServiceClient;
7 public PdfService(ILog logger, PdfServiceClient pdfServiceClient)
8 {
9 this.logger = logger;
10 this.pdfServiceClient = pdfServiceClient;
11 }
12 public virtual byte[] Html2Pdf(string html, double headerMargin = 0,
double footerMargin = 0)
13 {
14 var result = Task.Run(async () => await pdfServiceClient.GetPdfAsync(html,
headerMargin,
footerMargin));
15 var res = result.Result.content;
16 return res;
17 }
18 }
19 }
```

Оскільки реалізація вирішила використати асинхронний виклик API, потрібно дочекатися відповіді, інакше процес продовжуватиме виконуватися без відповіді та відповідь від мікросервісів не дасть результатів, рядок 14. Перш ніж відповідь можна буде повернути, PDF необхідно видобути у вигляді масиву байтів, рядок 15.

Реалізація класів та модулів моноліту, зв'язаних із класом PdfService.cs моноліту, не повинна змінюватись, а функціональні можливості передаються мікросервісу. Оскільки зв'язок із мікросервісом PdfGeneration і назад до моноліту регулюється через DTO, відповідні DTO також були створені в проєкті Crm.Library. Ці DTO ідентичні DTO із мікросервісу.

Табл. 7. Прототип Circuit Breaker для PdfServiceClient, реалізований у класі запуску моноліту / Circuit Breaker prototype for PdfServiceClient implemented in the monolith's startup class

```
99 services.AddHttpClient<PdfServiceClient>(client =>
100 {
101 client.BaseAddress = new
Uri(configuration.GetValue<string>("PdfServiceAddress"));
102 })
103 .AddTransientHttpErrorPolicy(builder =>
builder.Or<TimeoutRejectedException>().WaitAndRetryAsync(
104 5,
105 retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt))
106 + TimeSpan.FromMilliseconds(jitterer.Next(0, 1000)),
107 onRetry: (outcome, timeSpan, retryAttempt) =>
108 {
109 var serviceProvider = services.BuildServiceProvider();
110 serviceProvider.GetService<ILogger<PdfServiceClient>>()?
111 .LogWarning($"Delaying for {timeSpan.TotalSeconds} seconds, then making
retry {retryAttempt}");
112 }
113 ))
114 .AddTransientHttpErrorPolicy(builder =>
builder.Or<TimeoutRejectedException>().CircuitBreakerAsync(
115 3,
116 TimeSpan.FromSeconds(15),
117 onBreak: (outcome, timeSpan) =>
118 {
119 var serviceProvider = services.BuildServiceProvider();
120 serviceProvider.GetService<ILogger<PdfServiceClient>>()?
121 .LogWarning($"Onening the circuit for {timeSpan.TotalSeconds} seconds.");
122 },
123 onReset: () =>
124 {
125 var serviceProvider = services.BuildServiceProvider();
126 serviceProvider.GetService<ILogger<PdfServiceClient>>()?
127 .LogWarning($"Closing the circuit.");
128 }
129 ))
130 .AddPolicyHandler(Policy.TimeoutAsync<HttpResponseMessage>(1));
```

Для виклику REST API мікросервісу PdfGeneration у класі Startup.cs моноліту реалізовано прототип Circuit Breaker, як показано у табл. 7. Загалом моноліт повторно застосовує мікросервіс п'ять разів, коли запит не вдається, рядок 104, щоразу з різним часом очікування до наступної спроби, рядки 105 і 106. Після трьох спроб розмикається Circuit Breaker і чекає 15 секунд, поки не відімкнеться, а потім знову закривається, рядки 115–128.

Наведено різні ключові показники ефективності (KPI) та інструменти моніторингу для оцінювання продуктивності мікросервісів і контейнеризації в CRM-системах. Використання ресурсів є критичним KPI для оцінювання ефективності мікросервісів і контейнеризації в CRM-системах. Відстежування використання ЦП, пам'яті та мережі дає змогу виявляти потенційні вузькі місця та оптимізувати розподіл ресурсів. Такі інструменти, як Prometheus і Grafana, можна використовувати для збирання, візуалізації та аналізу показників використання ресурсів, забезпечуючи детальне уявлення про продуктивність системи. Час відгуку та затримка також є важливими KPI для оцінювання продуктивності мікросервісів і контейнеризації в системах CRM. Менший час відповіді вказує на ефективнішу та масштабованішу систему, що забезпечує кращий досвід користувача. Інструменти моніторингу, такі як Jaeger і Zipkin, можна використовувати для відстеження запитів і вимірювання затримок у мікросервісах.

Обговорення результатів дослідження. Коли йдеться про безпеку даних у CRM-системах, міграція від монолітної архітектури до архітектури мікросервісів є неоднозначним рішенням. Поділ на менші незалежні сервіси, кожен із власною функціональністю, сприяє масштабованості та гнучкості, але також розширює поверхню атаки. Коли зростає кількість сервісів, які спілкуються через мережі, з'являється більше точок входу для зловмисників. У монолітній архітектурі всі компоненти зазвичай розташовані в одному середовищі, що полегшує захист каналів зв'язку та застосування заходів безпеки на рівні мережі. Однак в архітектурі мікросервісів сервіси часто спілкуються через мережі, що створює потенційну загрозу перехоплення або маніпулювання конфіденційних даних. Окрім цього, у монолітній системі дані часто тісно пов'язані та розподіляються між різними модулями чи рівнями. Це може створити проблеми для сегментації даних і контролю доступу, оскільки компроміс в одній частині системи потенційно може вплинути на безпеку всього набору даних.

З іншого боку, мікросервіси сприяють сегментації даних, інкапсулюючи їх в окремі сервіси. Це може по-

силити безпеку, обмеживши доступ до певних наборів даних на основі кордонів служби, зменшивши вплив порушення безпеки.

Оскільки мікросервіси реалізуються як API, початкова форма автентифікації для мікросервісів передбачає використання криптографічних ключів API. Маркери автентифікації, закодовані на мові розмітки Security Assertion Markup Language (SAML) або через підключення OpenID у межах OAuth 2.0, є потенціальним рішенням щодо підвищення безпеки даних. Для авторизації необхідна централізована архітектура для надання та застосування політик доступу, що регулюють доступ до всіх мікросервісів. Також потрібен стандартизований, нейтральний щодо платформи метод передавання рішень щодо авторизації через стандартизований маркер. Деякі з них можуть бути маркерами доступу OAuth 2.0, закодованими у форматі JSON, оскільки кожен із мікросервісів можна реалізувати на іншій мові чи платформі. Надання політики та обчислення рішень щодо доступу потребують використання сервера авторизації. Недолік упровадження політик контролю доступу в точці доступу кожної мікрослужби полягає в тому, що потрібні додаткові зусилля, щоб гарантувати, що загальні політики, застосовні до всіх API мікрослужб, реалізуються однаково. Будь-яка розбіжність у реалізації політики безпеки між API матиме наслідки для безпеки всієї програми на основі мікросервісів.

Коли йдеться про імплементацію концепцій захисту в автентифікаціях, для доступу до API, які мають доступ до конфіденційних даних, повинні використовуватися токени автентифікації, які або мають цифровий підпис, або підтверджені авторитетним джерелом. Крім того, для деяких сервісів можуть знадобитися одноразові або короткочасні токени, щоб обмежити шкоду, яку може завдати скомпрометований токен. Маркери автентифікації мають бути реалізовані на основі дескриптора, де спочатку посилання на маркер надсилається стороні, яка перевіряє. Кожен ключ API, який використовується в CRM-системі, повинен мати обмеження, визначені як для системи, так і для набору API, де їх можна застосовувати. Обсяг обмежень для функціональності кожного ключа API повинен відповідати рівню надійності, що надається під час перевірки ідентифікації. Коли токени автентифікації без збереження стану (наприклад, вебтокени JSON (JWT)) використовують для упровадження спільних бібліотек, пов'язаних із мікросервісом, термін дії маркера має бути якомога коротшим, оскільки він визначає тривалість сеансу, і активний сеанс не може бути відкликано, а секретний ключ маркера не повинен бути частиною коду бібліотеки. Це має бути динамічна змінна, представлена змінною середовища або вказана у файлі даних середовища.

Керування доступом повинно здійснюватися так. Політики доступу до всіх API та їхніх ресурсів мають бути визначені та надані серверу доступу. Дозвіл на виклик для певного набору адресних функцій повинен бути визначений та запроваджений на початковому шлюзі API, тоді як авторизації на вищому рівні деталізації мають бути визначені та запроваджені ближче до розташування мікросервісів. Мікросервіси повинні мати змогу кешувати дані політики; на цей кеш можна покладатися лише тоді, коли сервер доступу недоступний і його термін дії повинен закінчитися через період, відповідний для середовища / інфраструктури. Сервер доступу повинен уможливлювати підтримання детальних політик. Рішення щодо доступу від сервера доступу мають передаватися окремим мікросервісам за допомогою стандартизованих маркерів, закодованих у нейтральному для платформи форматі (наприклад, маркер OAuth 2.0, закодований у форматі JSON). Необхідно ретельно контролювати обсяг внутрішніх маркерів авторизації, доданих мікрошлюзом або точкою прийняття рішень до кожного запиту; наприклад, у запиті на транзакцію внутрішній маркер авторизації має бути обмежений за обсягом, щоб залучати лише кінцеві точки API, до яких необхідно отримати доступ для цієї транзакції. Шлюз API можна використовувати для централізації примусової автентифікації та контролю доступу для всіх наступних мікросервісів, у такому разі немає потреби забезпечувати автентифікацію та контроль доступу для кожного окремого сервісу. Будь-який компонент, відповідно розміщений у мережі, може створювати анонімні з'єднання зі службами в обхід шлюза API та його захисту.

Наукова новизна отриманих результатів дослідження – вперше було інтегровано протокол автентифікації OpenID у поєднанні OAuth 2.0 саме в системах CRM мікросервісної архітектури.

Практична значущість результатів дослідження – захист даних CRM-систем в умовах децентралізованої, тобто менш контрольованої за природою мікросервісної архітектури.

Висновки / Conclusions

Мета цієї роботи – здійснити часткову міграцію додатка із програмного забезпечення CRM-системи за допомогою FMM. Виявилося, що міграція бази даних є однією із найчастіше згадуваних проблем у разі міграції програмного забезпечення з монолітної структури в мікросервісну. Описано, як ідентифіковано послуги за допомогою вибраного методу, відфільтровано шаблони та найкращі практики, а архітектура мікросервісів згенерована із цими результатами.

Підсумуємо: виконане дослідження продемонструвало, що інтеграція мікросервісів і контейнеризації в системи CRM може сприяти істотному підвищенню ефективності використання ресурсів, масштабованості та загальної продуктивності системи. Використовуючи ці технології та дотримуючись найкращих практик щодо їх упровадження, підприємства можуть створювати системи CRM наступного покоління, які не лише відповідають дедалі вищим вимогам сучасного цифрового ландшафту, але й створюють міцну основу для майбутнього зростання та інновацій. Хоча перехід до мікросервісів і контейнеризації може спричинити труднощі та потребувати зміни практики розроблення, зважаючи на потенційні переваги в плані підвищення продуктивності, економічності та зручності обслуговування, його варто здійснити організаціям, які прагнуть залишатися конкурентоспроможними в умовах діловодства сьогодення.

References

1. Sharda, K. (2020). Next-Gen SAAS CRM: Microservices and Containerization for Resource Efficiency. *International Journal of Novel Research and Development*, (5), 35–41.
2. Song, H., Nguyen, P., & Chauvel, F. (2019). Using microservices to customize multi-tenant SaaS: From intrusive to non-intrusive.

2019 OASICS Microservices, 1. https://doi.org/10.4230/OASIcs.Microservices.2017/2019.1
3. Nanhe, P., & Nanhe, Ms. (2024). An overview of customer relationship management. *International Journal of Advanced Research in Science, Communication and Technology*, 32–36. https://doi.org/10.48175/IJARSCT-17507
4. Malima, B., & Mbogo, C. (2024). Influence of customer relationship management on customer retention in the banking sector in Tanzania. *International Journal of Research and Innovation in Social Science*, (8), 2728–2741. https://doi.org/10.47772/IJRISS.2024.803189
5. Röser, M. (2024). Customer relationship management in new business models. https://doi.org/10.5772/intechopen.114840
6. Adriel, K., Sudarman, M., Smith, B., & Mustikasari, F. (2024). The effect of social customer relationships management on customer loyalty in Indonesia's e-commerce. *International Journal of Professional Business Review*, (9). https://doi.org/10.26668/businessreview/2024.v9i3.4319
7. Vem, L., Mshelmbula, J., Ochigbo, A., & Agwom-Panle, R. (2024). How internal customer relationship management and word of mouth affect customer loyalty. *Etikonomi*, (23), 81–92. https://doi.org/10.15408/etk.v23i1.26921
8. Jingjing, X. (2024). Social media marketing, customer relationship management. *International Journal of Research Studies in Management*, (12). https://doi.org/10.5861/ijrsm.2024.1026
9. Dwivedi, R. K., Lohmor, S., Dixit, R. S., Sahiba, Z., & Naik, S. (2024). The customer loyalty vs. customer retention: The impact of customer relationship management on customer satisfaction. *Web Intelligence*, 1–18. https://doi.org/10.3233/WEB-230098
10. Taherdoost, H. (2023). Customer relationship management. https://doi.org/10.1007/978-3-031-39626-7_10
11. Lew, G., & Bochenek, M. (2023). The concept of customer cost accounting in customer relationship management. *Humanities and Social Sciences Quarterly*, (30), 173–187. https://doi.org/10.7862/rz.2023.hss.71
12. Naim, A. (2024). Utilization of information systems to enhance customer relationship management, (28).
13. Rajagukguk, W., Samosir, O., Rajagukguk, J., & Rajagukguk, H. (2024). Service quality and supply chain value on customer loyalty: The role of customer relationship management. *Uncertain Supply Chain Management*, (12), 955–964. https://doi.org/10.5267/j.uscm.2023.12.012
14. Gbolagade, A., & Adeyemi, O. (2024). Effect of customer relationship management on customer satisfaction in the Nigeria food industry: A case study of Sweet Sensation Confectioneries.
15. Yashodha, Dr., & Lalitha, Dr. (2024). A study on customer relationship management in FMCG companies.
16. Prior, D. (2023). Customer relationship management. https://doi.org/10.1007/978-3-031-23409-5_6
17. Manajemen, P. (2023). The effect of service quality and customer relationship management (CRM) on customer loyalty. https://doi.org/10.31219/osf.io/y52r4

A. Ya. Pryhoda

State University of Trade and Economics, Kyiv, Ukraine

SOFTWARE MIGRATION FROM MONOLITHIC ARCHITECTURE TO MICROSERVICES ARCHITECTURE AS A WAY OF PROTECTING CRM SYSTEMS

The study of this work was focused on the process of protecting CRM systems in the conditions of the transition of software, which is the main part of CRM systems from a monolithic architecture to an architecture based on microservices. This article explores a migration strategy using the Strangler Fig pattern that facilitates the incremental adoption of microservices while maintaining compatibility with the existing monolith. A key aspect of a migration strategy is choosing a framework, such as the Framework for Microservices Migrations (FMM), that standardizes development practices and helps decompose monoliths into manageable components. Communication between microservices and the monolith is facilitated through a REST API that ensures seamless interaction. The integration of microservices is achieved through the implementation of DTOs (Data Transfer Objects) and API gateways, which ensures smooth data exchange between components. The article emphasizes the importance of dependency and configuration management in microservices, emphasizing the need for encapsulation and autonomy. In addition, the research examines the use of Curcuit Breaker patterns to effectively handle failures and maintain system stability during the migration process. Microservices, implemented as APIs, demand robust authentication methods such as cryptographic API keys and tokens. Security Assertion Markup Language (SAML) or OpenID with OAuth 2.0 are suggested solutions to enhance data security. Centralized architecture for authorization policies is crucial to regulate access across all microservices, while standardized token-based authentication ensures platform-neutral interoperability. However, decentralized authorization enforcement at each microservice entry point may lead to inconsistent security policies. Moreover, access to APIs handling confidential data necessitates authentication tokens with digital signatures or from trusted sources. Access control policies should be meticulously defined and enforced, with considerations for token expiration and scope limitations. The abstract underscores the significance of centralized access management and the challenges posed by anonymous connections bypassing API gateways.

Keywords: migration process, monolithic architecture, Microservices, CRM systems, dependency management, configuration handling, system resilience, encapsulation.

Інформація про авторів:

Пригода Андрій Ярославович, аспірант, асистент, кафедра інженерії програмного забезпечення та кібербезпеки.
Email: a.pryhoda@knute.edu.ua; https://orcid.org/0000-0003-3774-4583

Цитування за ДСТУ: Пригода А. Я. Міграція програмного забезпечення з монолітної архітектури на архітектуру мікросервісів як спосіб захисту CRM-систем. *Український журнал інформаційних технологій*. 2024, т. 6, № 2. С. 90–97.

Citation APA: Pryhoda, A. Ya. (2024). Software migration from monolithic architecture to microservices architecture as a way of protecting CRM systems. *Ukrainian Journal of Information Tecnology, 6*(2), 90–97. https://doi.org/10.23939/ujit2024.02.090