УДК 004.4’24 http://doi.org/10.15407/pp2023.02.091

Б.В. Бодак, А.Ю. Дорошенко

АВТОМАТИЗАЦІЯ В СИСТЕМІ ЕЛЕКТРОННИХ ЗАКУПІВЕЛЬ З МОДУЛЕМ АУКЦІОНУ

Розроблено комплексну систему автоматизації електронних закупівель з інтегрованим модулем аукціону. Комплекс побудовано на базі мікросервісної архітектури з використанням новітніх технологій для сервера та клієнта. Взаємодія між модулями системи та синхронізація даних реалізована у реальному часі через протоколи Web Socket. Для збереження даних використовується розподілена база даних з кешуванням для прискорення часу доступу до даних. Авторизацію користувачів та підсистем між собою створено на основі останнього стандарту OAuth 2.0 та додаткового використання нової власної реалізації алгоритму безпечної аутентифікації відкритого клієнта. Програмний комплекс скомпоновано у форматі Docker-контейнера, готового до публікації у хмарних сервісах або на фізичних серверах будь-якої операційної системи без зміни програмного коду. Створено набір засобів тестування, а саме модульних тестів та інтеграційних тестів для кожної підсистеми. Моніторинг комплексу автоматизації і його модулів реалізовано через стек технологій для запису та візуалізації стану, помилок, навантаження у системі. Вдосконалено автоматизовані процеси побудови та публікації системи електронних закупівель з модулем аукціону та системою моніторингу.

Ключові слова: електронні закупівлі, аукціон, автоматизація закупівель, модульна архітектура, мікросервіси, REST API, Blazor, Web Sockets, Identity Service, OAuth 2.0, Docker, ELK.

Актуальність комплексу електронних закупівель

Проведення закупівель за допомогою електронних торговельних майданчиків не є новим для України. В епоху швидкого розвитку інформаційних систем та технологій усе більше процесів піддаються автоматизації та спрощенню, що в цілому підвищує їхню ефективність. Не є винятком процес тендерних закупівель, який, на жаль, вважається найбільш заангажованим та корумпованим у бізнесі. Одним із кроків на шляху до вирішення цієї проблеми може стати проведення закупівель через електронні системи автоматизації.

Подібні системи, орієнтовані на прозорість, продуктивність, зручність та доступність, уже набули популярності у сфері закупівель для державного сектору. Найбільш розповсюдженою системою для здійснення державних електронних закупівель є «Prozorro» [1], яка розпочала свою роботу у лютому 2015 р. завдяки цій системі, вдалося значно зменшити часові та матеріальні витрати, систематизувати документообіг у електронному форматі, а також підвищити прозорість державних торгів.

Спираючись на досвід системи «Prozorro» [2] і подібних [3, 4], та показники економії коштів у використанні електронних систем закупівель [5] було ухвалено рішення реалізувати комплексну систему автоматизації електронних закупівель для бізнесу. Комплекс призначено для використання приватними підприємствами, фізичними та юридичними особами, зацікавленими у проведенні відкритих торгів або постачанні певних товарів/послуг для інших підприємств.

1. Архітектура комплексу

Комплексна система автоматизації електронних закупівель складається з таких модулів: центральна база даних (ЦБД), мікросервіси з REST API інтерфейсом, модуль аукціону, сервіс аутентифікації та авторизації, веб-клієнт. Основні компоненти системи та зв’язки між ними представлено на Рис. 1.

База даних побудована на технології Microsoft SQL Server та включає основні сутності системи, такі як закупівлі, лоти, пропозиції, контракти, аукціони, етапи та інші. Цілісність БД забезпечується через первинні та зовнішні ключі, а використання індексів для кожної сутності пришвидшує виконання запитів до таблиць зі значною кількістю даних. Діаграма основних сутностей БД зображена на Рис. 2.

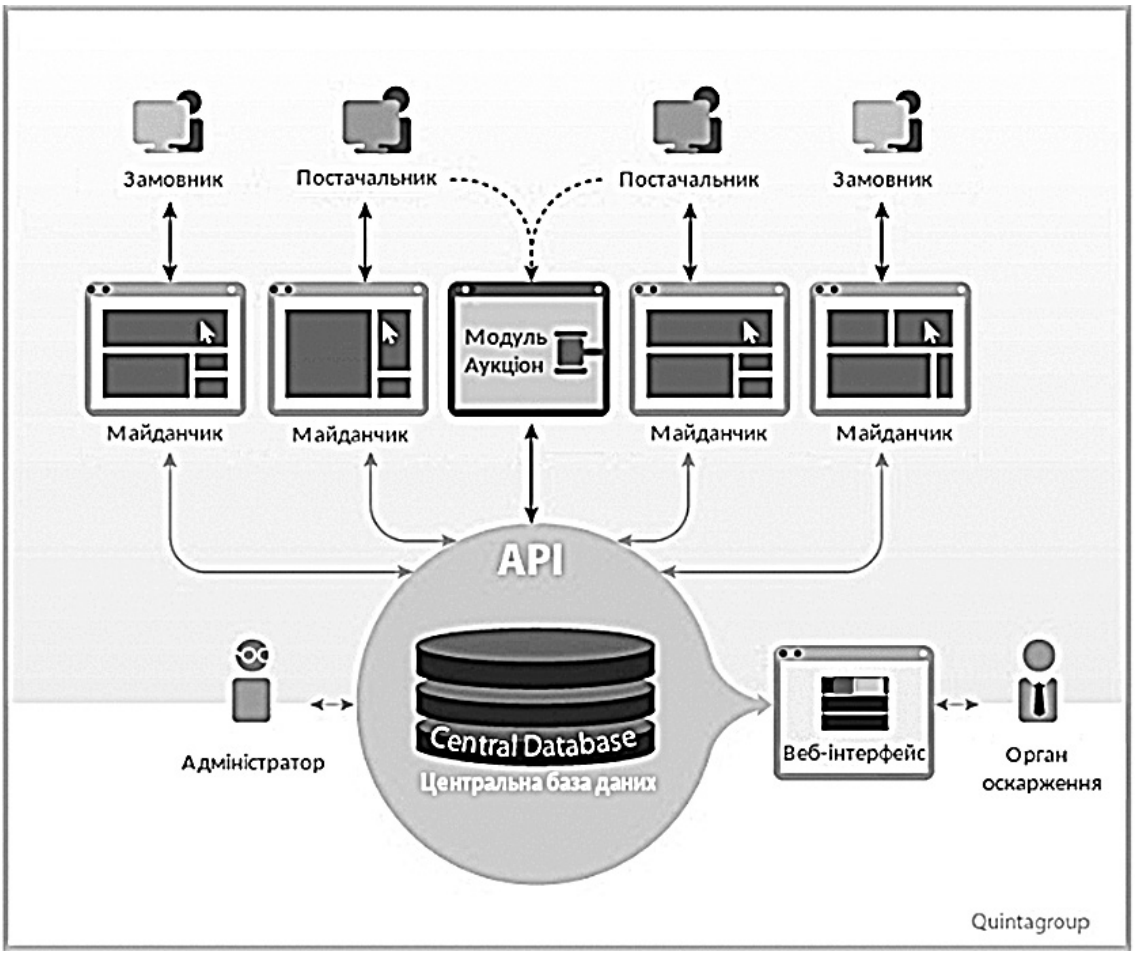

Рис. 1. Діаграма модулів комплексу електронних закупівель

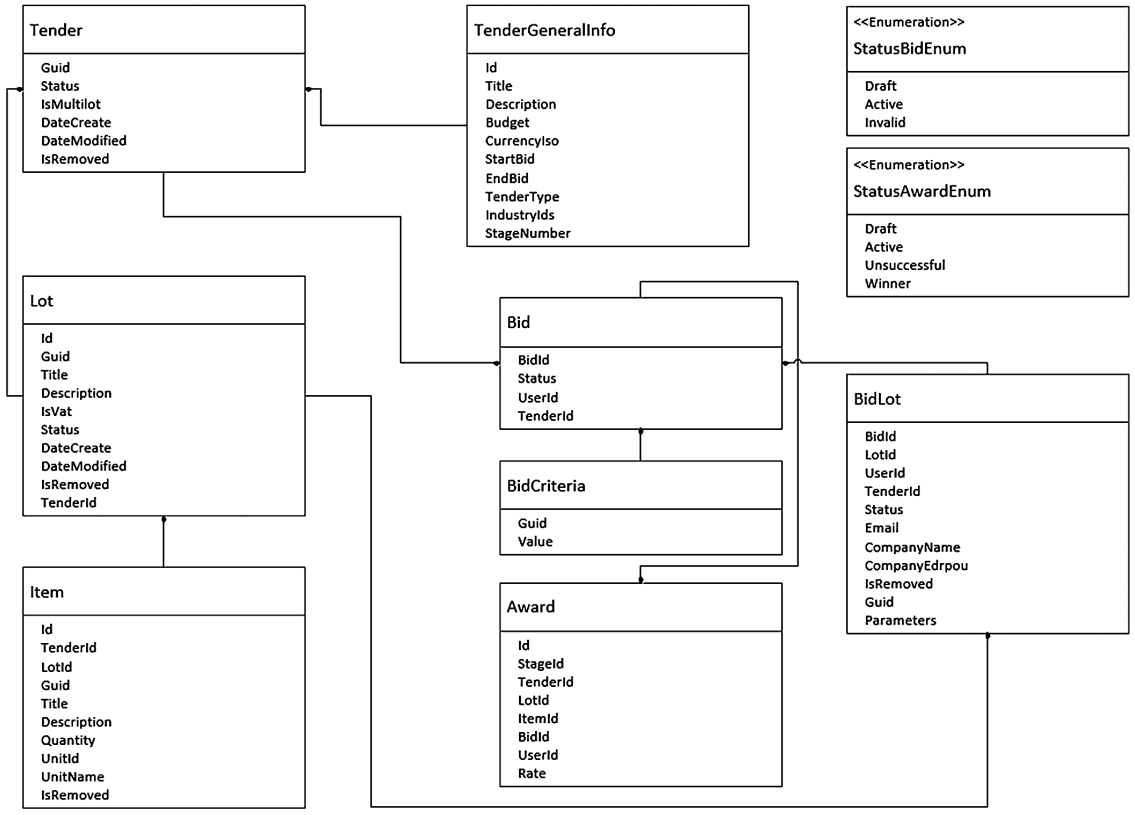

Рис. 2. Діаграма бази даних

БД містить збережені процедури для роботи із закупівлями, аукціонами, контрактами, та іншими таблицями. Використання збережених процедур дозволило у подальшому спростити рівень доступу до даних у мікросервісах і реалізувати потенціал MS SQL Server для обробки складних запитів. Приклад збереженої процедури Tender_GetById для отримання закупівлі представлено нижче:

```
CREATE PROCEDURE [dbo].[Tender_
GetById]
        (
        @Id BIGINT
)

        AS
        BEGIN

        SET NOCOUNT ON;

        SELECT T.[Id]
                ,[dbo].[Stage_Get](@Id,0)
AS ‘StageNumber’
                        ,T.[Guid]
                        ,T.[TenderId]
                        ,T.[Status]
                        ,T.[Title]
                        ,T.[Description]
                        ,T.[Budget]
                        ,T.[CurrencyIso]
                        ,T.[StartBid]
                        ,T.[EndBid]
                        ,T.[InfoCompany]
                        ,T.[ContactPerson]
                        ,T.[TenderType]
                        ,T.[AdditionalAttributes]
                        ,T.[ListFeatures]
                        ,T.[ListDocuments]
                        ,T.[ListCriteriaGroups]
                        ,T.[DateCreate]
                        ,T.[DateModified]
                        ,T.[IsRemoved]
                        ,T.[PlatformId]
                        ,(SELECT [dbo].
[ShortLot_GetByTenderId](@Id)) AS
ListShortLotsOrItems
        FROM [dbo].[Tenders] AS T
        WHERE T.[Id] = @Id AND
ISNULL(T.[IsRemoved], 0) = 0
END
```

Приведена вище процедура вибирає дані одразу з декількох таблиць та видає у результаті повноцінну модель закупівлі, готову до використання у мікросервісі.

Переходи між станами закупівлі у період аукціону реалізовано на рівні БД через Hangfire, що являє собою надбудову над стандартними тригерами БД та забезпечує синхронізацію даних, що вкрай важливо у аукціоні. Модуль аукціону доповнює тригери на рівні БД через фреймворк Stateless, реалізовуючи скінчений автомат станів для переходу між етапами закупівлі – TenderStateMachine.

Мікросервіси використовують micro-ORM Dapper та мають доступ до бази даних через репозиторій DatabaseRepository, який уможливлює зчитування, оновлення, вставки, видалення даних, а також виклик збережених процедур. Даний компонент являє собою абстракцію бази даних та працює із сервером БД, на якому розгорнута база даних MS SQL. У порівнянні з розповсюдженим Entity Framework, Dapper має обмежені можливості, проте значно вищу швидкість обробки запитів [6], що є ключовим у контексті застосування в розробленому комплексі електронних закупівель.

Серверна частина комплексу написана на C# .NET Core API та структурно складається з набору мікросервісів, які обмінюються між собою даними у форматі JSON об’єктів через REST протокол.

Для обміну даними у реальному часі на критичних етапах закупівлі, а також для оповіщення клієнта про зміни (наприклад, статусу закупівлі) використовується протокол Web Sockets [7]. Даний протокол дозволяє встановити зв’язок між сервісом та клієнтом (або між двома сервісами) для обміну даними у реальному часі з мінімальними затримками. Програмна реалізація протоколу Web Sockets на рівні мікросервісів створена з використанням інструменту SignalR [8] від Microsoft. Приклад налаштування SignalR у модулі аукціону та сповіщення про зміну статусу закупівлі наведено нижче:

```
private readonly IHubContext<TenderHub,
ITenderHubClient> _tenderHubContext;
await_tenderHubContext.
```

```
Clients.Group(TenderHub.
GetTenderGroupName(tenderId)).
StatusChanged(status);
```

Кожен із мікросервісів містить сторінку з документацією, детальним описом API методів та параметрів, згенеровану автоматично інструментом для документації – Swagger [9]. Документація методів та моделей написана у форматі XML коментарів у коді сервісів. Приклад документації методів наведено нижче у розділі.

Аутентифікація та авторизація користувачів реалізована на основі Microsoft Identity Service [10] відповідно до стандарту OAuth 2.0. Система підтримує формат Single Sign On (SSO), тобто користувач має авторизуватися лише один раз для використання усіх можливих підсистем комплексу електронних закупівель. Відповідно до вимог [11], у системі існує дві основні ролі – замовник та постачальник. Процес реєстрації користувачів, доступний функціонал та повноваженння різняться відповідно до ролі. Додатково введена роль адміністратора, котрій доступний повний функціонал системи. Авторизація здійснюється на основі JSON Web Token (JWT) [12], який містить повноваження та іншу інформацію про користувача (Claims). Валідація токену проводиться на сервері при кожному запиті. Приклад JWT постачальника виглядає так:

```
{
«sub»: «37483811»,
«name»: «Bohdan Bodak»,
«aud»: «https://bbtender.azurewebsites.net»,
«iat»: 1685490321,
«exp»: 1717026321,
«sub»: «bohdan.bodak@outlook.com»,
«GivenName»: «Bohdan»,
«Surname»: «Bodak»,
«Email»: «bohdan.bodak@outlook.com»,
«Role»: [
«Supplier»,
«Project Administrator»
]
}
```

За аутентифікацію, авторизацію, реєстрацію та роботу з користувачами в цілому у системі відповідає мікросервіс AuthService, що є обгорткою над Microsoft Identity Service та надає REST API для зручного використання клієнтом чи іншими сервісами. Інтерфейс API сервісу авторизації зображено на Рис. 3.

Клієнтський додаток написаний на Blazor Web Assembly [13] додатково використовує власну реалізацію алгоритму Proof Key for Code Exchange (PKCE) [14] для безпечної авторизації публічного клієнту. Детальніше про клієнта та його авторизацію написано у розділі 2.

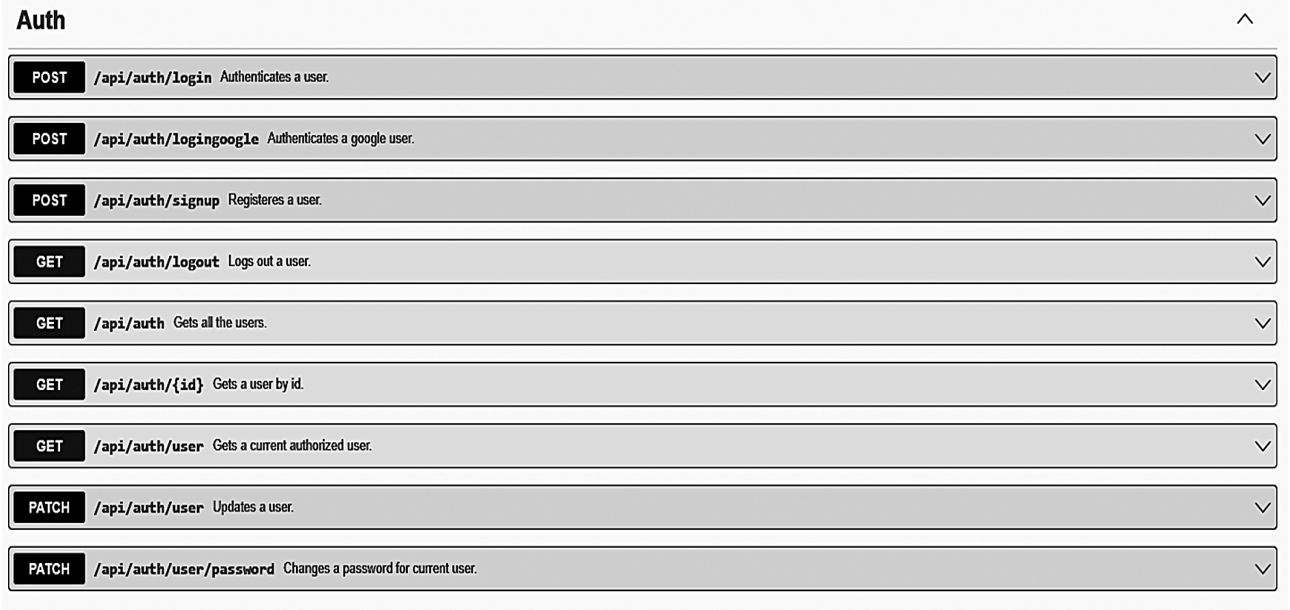

Рис. 3. API сервісу авторизації

2. Клієнт Blazor Web Assembly

Клієнт комплексної системи електронних закупівель представляє собою модульну-компонентну архітектуру та використовує технологію одно сторінкових додатків Blazor Web Assembly. Окрім існуючих переваг одно сторінкових вебсайтів, дана технологія дозволяє розробникам створювати код популярною строго типізованою мовою C#. На відміну від JavaScript, C# надає розробникам клієнтських додатків доступ до більшості можливостей фреймворку .NET: класів, інтерфейсів, Dependency Injection, спрощує процес розробки, полегшує налагодження, а також пошук помилок.

Для зменшення часу завантаження клієнтського додатку у браузері, був використаний підхід розподілення окремих частин клієнта на модулі. У комплексній системі електронних закупівель було виділено 3 модуля з такими основними компонентами:

- Модуль замовника: створення закупівлі, лотів, позицій, вибір пропозиції, підписання договору.
- Модуль постачальника: подача пропозиції, підписання договору.
- Спільний модуль / модуль неавторизованого користувача: перегляд закупівель та їхніх подробиць, сторінка авторизації та реєстрації.

Завантаження цих модулів відбувається за потребою, через так званий “Lazy Loading” [15]. За замовчуванням завантажується спільний модуль з основним функціоналом: перегляд закупівель та деталей. Коли користувач, скажімо, у ролі замовника, звертається до специфічної сторінки з іншого модулю, наприклад, до сторінки створення закупівлі, то цей модуль буде завантажено. Такий підхід дозволяє значно прискорити завантаження головних сторінок додатку та створити комфортний інтерфейс з мінімальною затримкою для користувачів.

Аутентифікація користувачів у клієнті відповідає стандарту OAUTH 2.0 для відкритих клієнтів. Використовується власна бібліотека для аутентифікації та авторизації за алгоритмом PKCE [16], базова реалізація якого передбачає, що клієнт надає серверу авторизації доказ того, що код авторизації належить саме йому, щоб сервер видав клієнтові код доступу. Для генерації коду доступу та коду підтвердження використовується PkceGenarator із власної бібліотеки.

```
PkceGenerator pkceGenerator = new ();
string codeVerifier = pkceGenerator.
CodeVerifier;
string codeChallenge = pkceGenerator.
CodeChallenge;
```

Було вдосконалено метод збереження коду підтвердження, замінивши сховище браузера на HttpOnly Cookie, що у свою чергу підвищило стійкість процесу авторизації до перехоплення кодів. Логіка процесу аутентифікації та авторизації зосереджена у двох класах: CustomAuthenticationStateProvider та SignInManager. Перший має посилання на другий і наслідує та реалізовує стандартний AuthenticationStateProvider із фреймворку Blazor, котрий має два методи: GetClaimsPrincipalAsync (для отримання поточного користувача) та Notify (для сповіщення змін у стані авторизації). Другий надає повноцінний інтерфейс для роботи з авторизацією: Login, Logout, GetClaimsPrincipalAsync.

Авторизація користувачів для певних сторінок задано через стандартний атрибут @attribute [Authorize(Role = “Supplier”)] (для постачальника або Role = “Buyer” для замовника). Деякі сторінки системи, такі як список закупівель не потребують авторизації та позначаються атрибутом @attribute [AllowAnonymous], надаючи доступ до них без авторизації.

Для умовного показу контенту для авторизованих користувачів на сторінках використовується тег <Authorized>. Приклад умовного показу кнопки подачі пропозиції для авторизованого постачальника:

```
<AuthorizeView>
<h1>Деталі закупівлі</h1>
…
<Authorized Role=”Supplier”>
<MudButton Variant=»Variant.Filled»
Color=»Color.Primary»>
```

```
Подати пропозицію
</MudButton>
</Authorized>
</AuthorizeView>
```

Клієнт використовує бібліотеку компонентів MudBlazor [17], яка набуває популярності серед розробників Blazor, та має широкий набір різних компонентів і стилів для побудови сучасного, швидкого та привабливого інтерфейсу користувача. Приклад сторінки з інформацією про закупівлю зображено на Рис. 4.

3. Публікація комплексної системи через Docker контейнер та моніторинг ELK

Для підтримки постійного впровадження змін у системі та стабільності використовуються 3 середовища: Dev (для розробників), Stage (копія реальної системи) та Prod (активна версія реальної системи). Усі середовища незалежні один від одного та побудовані на базі операційної системи Linux. Публікація комплексу включає у себе 2 етапи: побудова Docker-контейнеру [18] з вихідного коду (так званого артефакту) та розгортання контейнеру в обраному середовищі.

Процес побудови артефакту описано через Azure Build Pipeline та включає в себе завантаження коду з активної гілки репозиторію, виконання команд з побудови проекту, запуск автоматичних тестів та пакування Docker контейнеру. Таким чином, якість системи постійно контролюється через автоматичні тести. Новий артефакт створюється після будь-яких змін в основній гілці репозиторію, а також один раз на день (nightly build).

Створений Docker контейнер публікується у Dev середовищі для перевірки. Публікація на Stage та Prod виконується вручну після ретельного тестування та додаткових перевірок. Для публікації використовується Azure Release Pipeline.

Окрему увагу було приділено моніторингу комплексної системи. Для моніторингу системи за такими параметрами як швидкість виконання запитів, кількість активних користувачів, критичні та не критичні помилки, завантаженість, загальний стан системи та інших використовувався стек технологій Elasticsearch, Logstash та Kibana (ELK) [19].

Стек ELK – це найбільш популярна у світі платформа для аналізу та обробки лог-файлів із відкритим кодом. Даний набір технологій прийшов на заміну багатьом власним реалізаціям та швидко став стандартом індустрії. Серед користувачів стеку ELK можна зазначити такі популярні платформи як Netflix, LinkedIn, StackOverflow та інші. З розвитком мікросервісів та серверних даних у цілому, логфайли стають усе більш важливими, не тільки для пошуку потенційних проблем, а й для моніторингу системи та оптимізації продуктивності.

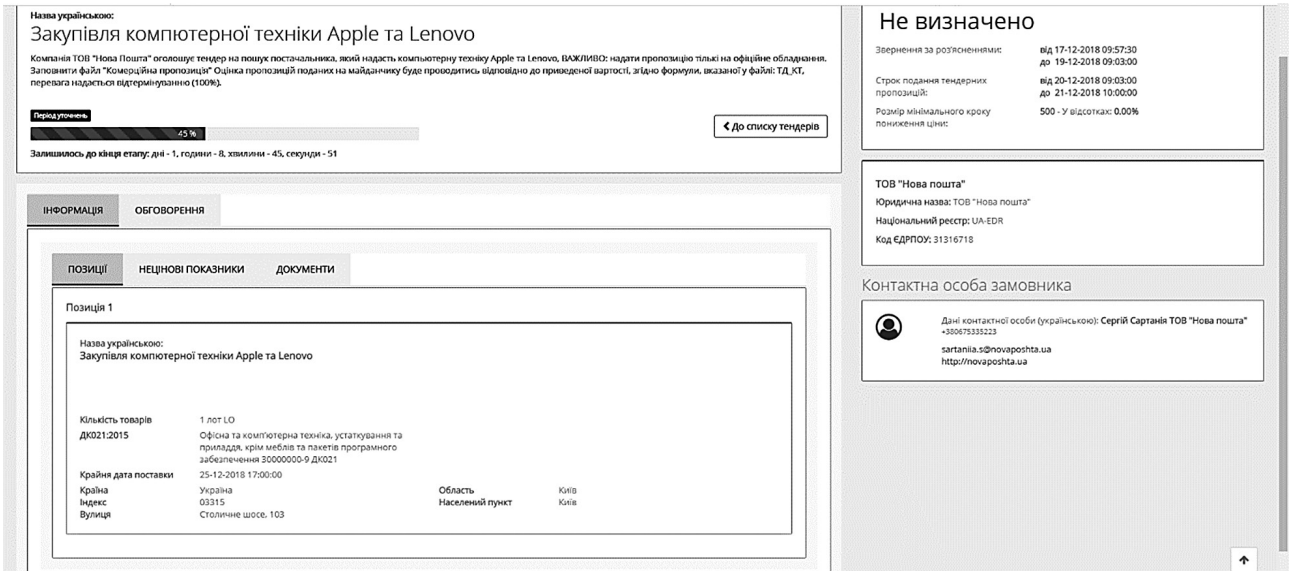

Рис. 4. Сторінка інформації про закупівлю

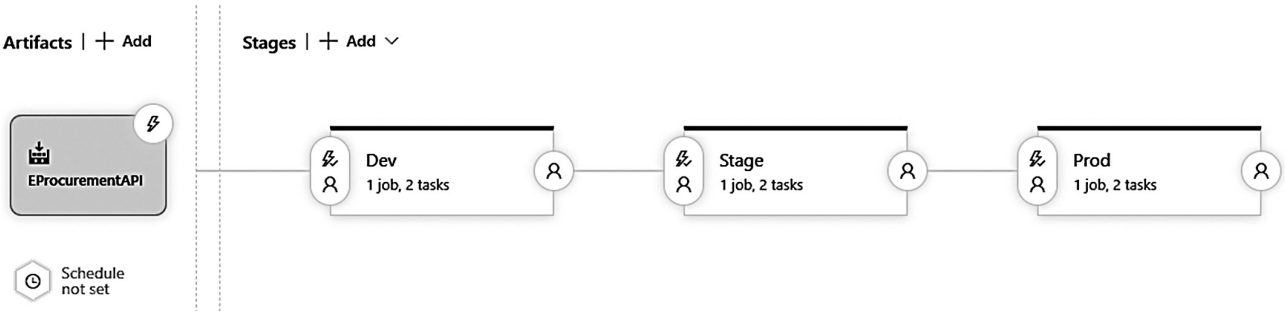

Рис. 5. Публікація системи через Azure

Основні компоненти стеку ELK:

• Elasticsearch: зберігає та індексує дані оброблені Logstash.

• Logstash: збирає інформацію про події, метадані та лог-файли. Оброблює та трансформує дані перед відправкою до Elasticsearch.

• Kibana: інструмент для візуалізації даних, який працює разом з Elasticsearch та надає користувачам можливість аналізувати інформацію та будувати складні звіти.

Слід зазначити, що для успішного відображення даних Kibana досить навіть одного кластера Elasticsearch. Однак завантаження інформації до пошукового ядра Elasticsearch – це складне завдання. На перший погляд може здаватися, що можливо завантажувати дані до Elastisearch через простий RESTful API. Але у такому разі система не буде працювати з конкурентними запитами, де кількість запитів доходитиме до декількох сотень на секунду. В нашій системі електронних закупівель присутні як конкурентні запити, так і висока кількість запитів на секунду від різних користувачів. Для того, щоб забезпечити якісний моніторинг усіх запитів та побудувати систему, здатну розширятися у майбутньому, ми будемо використовувати технологію Logstash [20], котра збирає дані (лог-файли, події, аналітику) з різних джерел, обробляє їх і передає у Elasticsearch.

Для передачі даних у Logstash ми використовуємо Azure Event Hubs, що являє собою чергу низку повідомлень типу “fire-and-forget” та має можливість розширення до мільйонів повідомлень на секунду [21]. Оскільки Azure Events Hubs – це черга повідомлень, не має потреби фізичного зв’язку між сервером ELK та мікросервісами нашої системи, що задовольняє критерії для публікації через Docker контейнер.

Створення лог-файлів та запис подій у системі реалізовано через провайдер Serilog, що підтримує інтеграцію з Azure Event Hubs [22]. Serilog надає стандартну реалізацію .NET Core інтерфейсу ILoggerFactory, через який у системі записуються усі події, виключення, та метадані про запит. Налаштування провайдеру Serilog в одному із мікросервісів нашої системи наведено нижче:

```
services.AddMvc(opt =>
 {
 opt.Filters.Add<LogResultFilter>();
 });

Log.Logger = new LoggerConfiguration()
.WriteTo.AzureEventHub(new JsonFormatter(), eventHubConfig.GetValue<string>(«connectionString»),eventHubConfig.GetValue<string>(«entityName»))
.CreateLogger();
```

Код створює новий екземпляр класу Logger та записує дані до Azure Event Hub. Для підтримки даних у компоненті Elasticsearch використовується клас JsonFormatter. Параметри конфігурації “connectionString” та ‘entityName” задані у створеному Event Hub на порталі Azure.

Для запису інформації про кожен запит на сервері була написана реалізація інтерфейсу IAsyncResourceFilter. Даний метод викликається для кожного запиту та передає інформацію про запит через Serilog до Azure Event Hub для подальшої індексації через ELK стек та

Рис. 6. Компоненти стеку ELK

Рис. 7. Візуалізація моніторингу

відображення на сторінці моніторингу. Приклад реалізації простого моніторингу для кожного запиту наведено нижче:

```
public class LogResultFilter :
IAsyncResourceFilter
{
public async Task
OnResourceExecutionAsync(
ResourceExecutingContext context,
ResourceExecutionDelegate next)
{
await next();

var data = new
{
Action = context.ActionDescriptor.
RouteValues[«action»],
Controller = context.ActionDescriptor.
RouteValues[«controller»],
Url = context.HttpContext.Request.Path,
StatusCode = context.HttpContext.
Response.StatusCode
};

Log.Information(«Request completed: {@
data}», data);
}
}
```

У додаток до стандартної інформації про запит, переданої нами у прикладі, провайдер Serilog автоматично передає такі метадані як час виконання запиту, навантаження на сервер або кластер, кількість виділених потоків, та інше. Це робить моніторинг системи надзвичайно зручним та корисним.

Публікація ELK стеку виконується у форматі окремого Docker контейнера на виділеному сервері для використання іншими мікросервісами та модулями комплексу. Процес публікації ідентичний описаному вище у розділі, за винятком до-

даткових стандартних процесів побудови та публікації для стеку ELK.

У результаті ми розробили модуль моніторингу для нашої комплексної системи електронних закупівель, через який розробники можуть спостерігати за станом системи, переглядати детальну інформацію про кожен запит, швидкість обробки, завантаженість серверу, та важливі події у системі.

Висновки

Реалізовано комплекс автоматизації електронних закупівель з модулем аукціону на базі REST API мікросервісів, бази MS SQL, та клієнту Blazor Web Assembly. Надана можливість авторизувати користувачів системи, а саме постачальників та замовників на основі стандарту OAuth 2.0 через Microsoft Identity Server. Бібліотека для роботи з розподіленим кешем Redis використовувалась для прискорення роботи системи та кешування даних на рівні доступу до БД. Зміни етапів закупівлі було реалізовано через тригери Hangfire та кінечний автомат Stateless. Сповіщення про важливі зміни у реальному часі виконано через протокол Web Sockets та SignalR. Програмний комплекс підготовлено до публікації у Microsoft Azure, інших хмарних сервісах або на фізичних серверах Windows/Linux через Docker контейнер. Забезпечено моніторинг системи та модулів через стек Elasticsearch, Logstash та Kibana (ELK) для візуалізації стану, помилок, навантаження.

References

1. ProZorro open-source government e-procurement system [Online] – Available from: https://prozorro.gov.ua/en.
2. ProZorro results report 2022 [Online] – Available from: https://infobox.prozorro.org/articles/richniy-zvit-prozorro-za-rezultatami-2022-roku.
3. Doroshenko, A.Yu., Bodak B.V. (2019). The impact and unforeseen challenges of E-procurement systems in Canada. Winter InfoCom Advanced Solutions. pp. 9 10.
4. SmartTender for commercial customers [Online] – Available from: https://smarttender.biz/etm/pro-kompaniyu.
5. How much money does a e-procurement system save, Centre of Excellence in Procurement [Online] – Available from: https://cep.kse.ua/article/impact-of-prozorro/impact-of-prozorro.pdf
6. Dapper performance benchmark, Dapper GitHub [Online] – Available from: https://github.com/DapperLib/Dapper/blob/main/Readme.md.
7. Fette I, Melnikov A. (2011). The WebSocket protocol [Online] – Available from: https://datatracker.ietf.org/doc/html/rfc6455.
8. Overview of ASP.NET Core SignalR, Microsoft Documentation [Online] – Available from: https://learn.microsoft.com/en-us/aspnet/core/signalr/introduction.
9. About Swagger specification [Online] – Available from: https://swagger.io/docs/specification/about/.
10. Identity Server Documentation, Duende [Online] – Available from: https://docs.duendesoftware.com/identityserver/v6/overview/.
11. Doroshenko A.Yu., Bodak B.V. (2021). Designing RESTful API for the e-procurement system in private sector. Problems of programming No 1. pp. 3 15. DOI: https://doi.org/10.15407/pp2021.01.003.
12. Jones M., Bradley J., Sakimura M. (2015). JSON Web Token (JWT) [Online] – Available from: https://datatracker.ietf.org/doc/html/rfc7519.
13. ASP.NET Core Blazor, Microsoft Documentation [Online] – Available from: https://learn.microsoft.com/en-us/aspnet/core/blazor.
14. Proof Key for Code Exchange, IETF [Online] – Available from: https://datatracker.ietf.org/doc/html/rfc7636.
15. Lazy load assemblies in ASP.NET Core Blazor Web Assembly, Microsoft Documentation [Online] – Available from: https://learn.microsoft.com/en-us/aspnet/core/blazor/webassembly-lazy-load-assemblies.
16. Bodak B.V., Doroshenko A.Yu. (2022). Protecting public clients using an authorization algorithm. Problems of programming No 3-4. pp. 409 416. DOI: https://doi.org/10.15407/pp2022.03-04.409.
17. Getting started with MudBlazor [Online] – Available from: https://mudblazor.com/getting-started/installation.

18. Building a Docker image for ASP.NET Core, Microsoft Documentation [Online] – Available from: https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/docker/building-net-docker-images.
19. What is ELK Stack, Elastic [Online] – Available from: https://www.elastic.co/what-is/elk-stack.
20. Logstash introduction, Elastic [Online] – Available from: https://www.elastic.co/guide/en/logstash/current/introduction.html.
21. About Azure Event Hubs – A big data streaming platform, Microsoft Documentation [Online] – Available from: https://learn.microsoft.com/en-us/azure/event-hubs/event-hubs-about.
22. How to log events to Azure Event Hubs in Azure API Management, Microsoft [Online] – Available from: https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-log-event-hubs.

Одержано: 18.05.2023

Про авторів:

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії комп'ютерних обчислень Інституту програмних систем НАН України,
професор кафедри інформаційних систем та технологій НТУУ «КПІ імені Ігоря Сікорського».
Кількість наукових публікацій в українських виданнях – понад 200.
Кількість наукових публікацій в іноземних виданнях – понад 100.
Індекс Гірша – 6.
http://orcid.org/0000-0002-8435-1451,

Бодак Богдан Вікторович,
аспірант кафедри інформаційних систем та технологій НТУУ «КПІ імені Ігоря Сікорського».
Кількість наукових публікацій в українських виданнях – 4.
http://orcid.org/0000-0002-4854-2314.

Місце роботи:
Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.