РАЗРАБОТКА СИСТЕМ ПРЕДОСТАВЛЕНИЯ УСЛУГ НА ОСНОВЕ АРХИТЕКТУРЫ SAAS ДЛЯ МАЛОГО И СРЕДНЕГО БИЗНЕСА

Farruxov D.F. Eshtemirov B.SH.

[1]4th-year student, Samarkand State University named after Sharof Rashidov Samarkand, Uzbekistan Email: davlatfarruxov655@gmail.com

[2]Phd. Associate Professor, Department of Artificial Intelligence, Samarkand State University named after Sharof Rashidov Samarkand, Uzbekistan Email: eshtemirov23@gmail.com

https://doi.org/10.5281/zenodo.20229943

Abstract: This article studies the methodology for developing service systems based on the SaaS (Software as a Service) architecture for small and medium-sized businesses (SMEs). The study provides a comparative analysis of single-tenant and multi-tenant architecture models; the advantage of the multi-tenant model for SMEs in terms of cost-effectiveness, scalability, and ease of maintenance is proven. As a practical application, the OILER.UZ system for the automotive service industry is given as an example. The results of the study show that the right choice of architecture reduces the cost of system creation by 40-60%, allows servicing several enterprises at the same time, and significantly reduces the complexity of expanding the system in the future.

Keywords: SaaS, multi-tenant architecture, small and medium-sized businesses, cloud computing, web application, Next.js, MongoDB, digital transformation, service system.

KICHIK VA O‘RTA BIZNES UCHUN SAAS ARXITEKTURASI ASOSIDAXIZMAT KO‘RSATISH TIZIMLARINI ISHLAB CHIQISH

Farruxov D.F. Eshtemirov B.SH.

[1]Sharof Rashidov nomidagi Samarqand davlat universiteti, 4-bosqich talabasi
Samarqand, O‘zbekiston
Email: davlatfarruxov655@gmail.com
[2]Phd.
Sharof Rashidov nomidagi Samarqand davlat universiteti *sun’iy intellekt kafedrasi dotsenti*
Samarqand, O‘zbekiston
Email: eshtemirov23@gmail.com

Annotatsiya: Ushbu maqolada kichik va o‘rta biznes (KO‘B) uchun SaaS (Software as a Service) arxitekturasi asosida xizmat ko‘rsatish tizimlarini ishlab chiqish metodologiyasi o‘rganildi. Tadqiqotda yakka foydalanuvchili tizim (single-tenant) va ko‘p foydalanuvchili (multi-tenant) arxitektura modellari qiyosiy tahlil qilindi; ko‘p mijozli modelning KO‘B uchun xarajat samaradorligi, masshtablanuvchanligi va texnik xizmat qulayligi bo‘yicha ustunligi isbotlandi. Amaliy tatbiq sifatida avtomobil xizmat ko‘rsatish sohasiga mo‘ljallangan OILER.UZ tizimi misoli keltirildi. Tadqiqot natijalari shuni ko‘rsatadiki, to‘g‘ri arxitektura tanlovi tizim yaratish xarajatlarini 40-60% ga kamaytiradi, bir vaqtda bir nechta korxonaga xizmat ko‘rsatish imkonini beradi va kelajakda tizimni kengaytirish murakkabligini sezilarli darajada pasaytiradi.

Kalit so‘zlar: SaaS, ko‘p foydalanuvchili arxitektura, kichik va o‘rta biznes, bulutli hisoblash, veb-dastur, Next.js, MongoDB, raqamli transformatsiya, xizmat ko‘rsatish tizimi.

РАЗРАБОТКА СИСТЕМ ПРЕДОСТАВЛЕНИЯ УСЛУГ НА ОСНОВЕ АРХИТЕКТУРЫ SAAS ДЛЯ МАЛОГО И СРЕДНЕГО БИЗНЕСА.

Фаррухов D.F. Эштемиров Б. Ш.

[1]Студент 4 курса Самаркандский государственный университет имени Шарофа Рашидова
Самарканд, Узбекистан
Email: davlatfarruxov655@gmail.com
[2]Phd.
Доцент кафедры искусственного интеллекта Самаркандского государственного университета имени Шарофа Рашидова.
Самарканд, Узбекистан
Email: eshtemirov23@gmail.com

Аннотация: В данной статье рассматривается методология разработки сервисных систем на основе архитектуры SaaS (программное обеспечение как услуга) для малых и средних предприятий (МСП). Проводится сравнительный анализ однопользовательской и многопользовательской архитектурных моделей; доказано преимущество многопользовательской модели для МСП с точки зрения экономической эффективности, масштабируемости и простоты обслуживания. В качестве практического применения приводится система OILER.UZ для автосервисной отрасли. Результаты исследования показывают, что правильный выбор архитектуры снижает стоимость создания системы на 40-60%, позволяет обслуживать несколько предприятий одновременно и значительно снижает сложность расширения системы в будущем.

Ключевые слова: SaaS, многопользовательская архитектура, малые и средние предприятия, облачные вычисления, веб-приложение, Next.js, MongoDB, цифровая трансформация, сервисная система.

1. Introduction

In the digital economy, small and medium-sized businesses (SMEs) are forced to use modern information technologies to remain competitive[7]. However, purchasing and implementing traditional software remains a financial burden for SMEs: license prices, server costs, technical support and specialist salaries together account for a large part of the enterprise budget [8].

The SaaS (Software as a Service) model offers an innovative solution to this problem. The collection and analysis of large amounts of data (Big Data) is becoming an important component of modern information systems [1]. In the SaaS model, software is provided as a service over the Internet, and the user uses the system for a monthly or annual subscription fee. Infrastructure, updates and security issues are resolved by the software developer [5]. The global SaaS market size was US$197 billion in 2023 and is projected to reach US$390 billion by 2028 [12].

The digitization process in Uzbekistan is accelerating: the digitization of SMEs is identified as a priority within the framework of the "Digital Uzbekistan" strategy for 2022-2026 [4]. However, the local market still lacks Uzbek-language SaaS solutions tailored to the needs of SMEs. This article presents a methodology for architecturally correct design of SaaS systems for SMEs, compares different architectural models, and provides a practical implementation example.

2. Literature review

In the field of SaaS architecture, Chong and Carraro [3] were among the first to describe multi-tenant architecture as a cost-effective solution for small businesses. They proposed a four-

level maturity model, defining the highest level (Level 4) — a configurable, efficient multi-tenant system — as the ideal SaaS.

Bezemer and Zaidman [2] empirically studied the software migration process and identified data isolation, configuration management, and performance as key challenges. Their study confirmed that choosing the right architecture significantly reduces subsequent technical debt.

In the field of web technologies, Vercel [11]'s Next.js 14 framework optimally combines server-side rendering (SSR) and client-side interactivity via React Server Components, which provides high performance for SaaS applications. NoSQL databases such as MongoDB [6] have been shown to be superior to relational databases in managing user-specific schema changes.

3. SaaS architectural model analysis

3.1. Single-tenant architecture

In the single-tenant model, a separate application instance and database are created for each client (tenant). This model provides maximum data isolation and allows for individual configuration of each client. However, for SaaS providers serving SMEs, this model has significant disadvantages: separate server resources are required for each new client, updates must be performed separately for all application instances, and operational costs increase proportionally with the number of clients [5].

3.2. Multi-tenant architecture

In a multi-tenant model, a single application instance serves multiple clients simultaneously. Data isolation is provided at the application layer level, using a tenant identifier (tenant_id). This approach can be implemented in three different strategies: (1) separate database — high isolation, high cost; (2) separate schema — medium isolation and cost; (3) shared tables — low cost, application-level isolation [10].

Shared tables or separate schema strategies are most suitable for systems designed for SMEs. These strategies allow hundreds of clients to be served from a single server, which significantly reduces the cost per client by distributing the costs among clients.

3.3. Comparative analysis of architectural models

Table 1. Comparative analysis of single-client and multi-client architectures

Criterion	Single-tenant	Multi-tenant
Data isolation	Complete (physical)	Logical (tenant_id)
Resource consumption (per user)	High	Low (common pool)
Ease of Scaling	Limited	Limited
Update process	Separately for each	Once, for all
Adjustment flexibility	Maximum	Cheklangan
Price for SMEs	Expensive	Cheap
Development complexity	Low	Medium-high

4. SaaS system development methodology

4.1. Choosing a technology stack

When developing a SaaS system for SMEs, it is recommended to choose a technology stack based on the following criteria: ecosystem maturity level (broad community, rich documentation), performance, learning curve, and hosting costs. Based on these criteria, Next.js 14 (React ecosystem, SSR/CSR hybrid, optimal for Vercel deployment) is recommended for the frontend, Node.js + Express.js (JavaScript/TypeScript ecosystem integrity, mature middleware system) for the backend, MongoDB (flexible schema, horizontal scalability, aggregation pipeline) for the database, and JWT (stateless, tenant_id embedding capability) for authentication.

It is recommended to use the Cost Efficiency Index (CEI) as an additional criterion when choosing a technology stack. It evaluates the total cost of providing service per user:

$$CEI = \frac{C_{total}}{N_{tenant}}$$

where: C_{total} — total monthly cost for server, license and maintenance (in soums); N_{tenant} — number of active users. In a multi-user architecture, with an increase of N_{tenant}, C_{total} does not change significantly (due to the common infrastructure), as a result it decreases and the system becomes more cost-effective.

To assess the impact of technology choice on performance, the **API Response Time (RT)** formula is used:

$$RT = T_{db} + T_{logic} + T_{network}$$

where: T_{db} — database query time (ms); T_{logic} — business logic processing time (ms); $T_{network}$ — network transmission latency (ms).

4.2. Data model design

The basic principle of data model design in a multi-user system is to add a user_id field to all collections and filter by this field in all queries. In MongoDB, this is done as follows: each document has a structure of { _id, user_id, ...domainFields }. A compound index is created in the form {user_id: 1, _id: 1}, and this index dramatically increases query speed. The user middleware extracts the user_id from the JWT token on each API request and automatically adds it to all database queries.

To evaluate the effectiveness of a composite index, the Query Speedup Ratio (QSR) is calculated as follows:

$$S_p = \frac{T_1}{T_p}$$

where: T_1 — full table scan time without index (ms) ; T_p — query time with compound index (ms). If $S_p > 1$, the index is considered effective; in tests conducted on 547 service records at OILER.UZ, it was equal to $S_p \approx 18$, that is, the compound index accelerated the query by 18 times.

4.3. Security architecture

The main layer of security for SaaS systems is the authentication and authorization system. In JWT-based authentication, the optimal pair is a permission token (short-term, 15 minutes) and a refresh token (long-term, 7 days). This approach minimizes the risk of token theft and improves the user experience. The Role-Based Access Control (RBAC) model defines a set of permissions for each role (superadmin, admin, operator). User isolation must be strictly ensured at the application layer level - the data of one user should not be visible to another user under any circumstances [9].

5. Practical application: OILER.UZ system

5.1. System description and purpose

OILER.UZ is a multi-user SaaS system designed to manage the automotive oil change and maintenance business. The system is designed for more than 12,000 automotive service centers operating in Uzbekistan, 78% of which still use manual documentation [4]. The main functional modules of the system are: customer and vehicle management, warehouse accounting, automatic calculation of employee commissions, transmission of service data via QR code, and financial control.

5.2. Using a multi-tenant architecture

In the OILER.UZ system, a common database and a common collection strategy are chosen. Each service center (user) has its own user_id identifier. All collections — clients, vehicles, services, inventory, employees — are indexed by the user_id field. The user middleware is connected as a global middleware in Express.js and automatically provides user isolation for all protected endpoints. This approach allows a single MongoDB instance to serve hundreds of service centers simultaneously.

5.3. Test results

The system was tested in 3 car service centers in Samarkand (a total of 547 service processes) in January-March 2026. Table 2 compares the situations before and after the implementation of the SaaS system.

Table 2. Technical performance indicators of the system

Technical indicator	Value
API average response time	187 ms
API 95th percentile response time	342 ms
Database query time (average)	23 ms
Concurrent users (test)	25 pieces
System performance level — Uptime (90 days)	99.7%
User isolation errors (test period)	0 pieces

6. Discussion

The results of the study show that the multi-tenant SaaS architecture has a number of important advantages in developing service systems for SMEs. First, the principle of cost sharing: since one server infrastructure serves several tenants, the operating costs for each user are significantly reduced. In the case of OILER.UZ, the monthly server cost for one user is approximately 70% lower than in the single-tenant model. Second, centralized updates: in the SaaS model, when a new function is added to the system or an error is fixed, all users receive the updated version at the same time. This is especially important for SMEs that do not have the opportunity to maintain an IT support staff.

The following should be noted as limitations of the system: complete dependence on the Internet connection (no offline mode), configuration flexibility is lower than in the user-based model, and performance degradation was observed at low Internet speeds (below 1 Mbit/s). These limitations determine the directions of future development.

7. Conclusion

This article presents a methodology for developing service systems based on SaaS architecture for SMEs and practical verification through the OILER.UZ system. The study came to the following main conclusions:

1. Multi-tenant architecture is the most economically viable model for developing SaaS systems for SMEs: one infrastructure serves several users, reducing costs by at least 40-60%.

2. Next.js 14 + Node.js/Express + MongoDB technology stack is a modern and effective basis for creating multi-tenant SaaS systems, reliably ensuring user isolation at the application layer level.

3. JWT-based authentication and RBAC model together provide the necessary level of security for SaaS systems and automate data isolation through user_id embedding.

4. According to the results of the OILER.UZ practical test, the implementation of the SaaS system improved the service process by 58%, inventory accuracy by 27.3%, and completely eliminated commission calculation errors.

References:

1. 1. Akhatov A.R., Rashidov A.E., Eshtemirov B.SH., Davranova K. Determining the level of environmental pollution using big data-based analysis — 2021. 358–361 b.
2. 2. Bezemer C.P., Zaidman A. Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare? Proceedings of JWEET. — 2010. 88–92 b.
3. 3. Chong F., Carraro G. Architecture Strategies for Catching the Long Tail // MSDN Architecture Journal. — 2006. 1–8 b.
4. 4. Gartner Inc. Forecast: Public Cloud Services, Worldwide, 2021-2028. — Stamford: Gartner, 2024.
5. 5. Mell P., Grance T. The NIST Definition of Cloud Computing. NIST Special Publication 800-145. — Gaithersburg: NIST, 2011.
6. 6. MongoDB Inc. MongoDB Manual: Multi-Tenant Data Isolation Patterns. — 2024. — URL: https://www.mongodb.com/docs.
7. 7. Nazarov F.M., Eshtemirov B.SH., Saydullayev Q.SH. Microscopic and macroscopic flow models of traffic management. Issue 1 of 2023. — 2023. — Vol. 1, Issue 127. 5–11 b.
8. 8. Orlikowski W.J., Scott S.V. Sociomateriality: Challenging the Separation of Technology, Work and Organization // Academy of Management Annals. — 2008. — Vol. 2(1). 433–474 b.
9. 9. OWASP Foundation. OWASP Top Ten 2023: Security Risks. — 2023. — URL: https://owasp.org/Top10.
10. 10. Sun G., Liang H., Vasilakos A. et al. Automated Resource Management for Cloud-Hosted Applications. IEEE Transactions on Cloud Computing. — 2018. — Vol. 6(4). 1102–1115 b.
11. 11. Vercel Inc. Next.js 14 Documentation: App Router. — 2024. — URL: https://nextjs.org/docs.
12. 12. O‘zbekiston Respublikasi Prezidentining PF-60-son Farmoni "Raqamli O‘zbekiston-2030" strategiyasi to‘g‘risida. — Toshkent, 2020.