http://www.moderntechno.de/index.php/meit/article/view/meit22-01-038
DOI: 10.30890/2567-5273.2022-22-01-038

**УДК 004.4**

# WEB APPLICATIONS TECHNOLOGICAL STACK
## ТЕХНОЛОГІЧНИЙ СТЕК РОБОТИ ВЕБ-ДОДАТКІВ

**Mychuda L.Z./ Мичуда Л.З.**
*d.t.s., prof. / д.т.н., проф.*
*ORCID: 0000-0001-8266-1782*
**Korobeinikova T.I. / Коробейнікова Т.І.**
*c.t.s., as.prof. / к.т.н., доц.*
*ORCID: 0000-0003-2487-8742*
*National University "Lviv Polytechnic", Lviv, S. Bandery St., 12, 79013*
*Національний Університет «Львівська політехніка», Львів, С. Бандери, 12, 79013*
**Nepyivoda M.V. / Непийвода М. В.**
*c.t.s., as.prof. / к.т.н., доц.*
*викладач спецдисциплін / teacher of special disciplines*
*ORCID: 0000-0002-9383-7752*
**Rekalo O. V. / Рекало О. В.**
*викладач спецдисциплін / teacher of special disciplines*
*ORCID: 0000-0003-3264-6263*
*Vinnytsia Technical Vocational College, Vinnytsia, Khmelnytskyi Shosse St., 91/2, 21021*
*Вінницький технічний фаховий коледж, Вінниця, Хмельницьке шосе, 91/2, 21021*

***Анотація.*** *В цій роботі розглядаються механізми і алгоритми роботи серверних додатків, зокрема технологій, на базі яких розробляються самі додатки, технології найбільш популярних баз даних, технології збирання та розгортання додатків і варіанти їх розміщення в робочому середовищі.*

***Ключові слова:*** *веб-додаток, трирівнева архітектура, технологічний стек роботи веб-додатків, рівень презентації, рівень додатку, рівень зберігання, SPA, CDN, монолітна архітектура, мікросервіс, база даних, інфраструктура веб-додатків, SaaS.*

**Вступ**. В обчислювальних системах веб-додаток – це клієнт-серверна комп'ютерна програма, з якою клієнт (включно із інтерфейсом користувача та логікою на стороні клієнта) працює засобами веб-браузерів [1, 2]. Загальний перелік веб-програм містить такі, як веб-пошта, роздрібні продажі в Інтернеті, онлайн-аукціони тощо [3, 4]. Програми зазвичай розбиваються на логічні фрагменти, які називаються «рівнями», де кожен рівень має свою роль. Традиційно, прикладні програми складаються з одного рівня (рівня клієнтської машини), але веб-додатки природньо прогресують до *n*-рівневого підходу.

Існує багато варіантів структури веб-додатків і найбільш поширеною є трирівнева. Для таких додатків, кінцевим завданням яких є властивості відмовостійкості та автомасштабування [5, 6], пропонується такий технологічний стек роботи веб-додатків: *презентації*, *додатку* і *зберігання*.

Веб-браузер – перший рівень (рівень презентації); рівень, що використовує деякі технології динамічного веб-контенту (наприклад ASP, CGI, ColdFusion, Dart, JSP/Java, Node.js, PHP, Python або Ruby on Rails) – це другий рівень (рівень додатку – тобто, логіка програми); а база даних – це є третій рівень (рівень зберігання). Веб-браузер надсилає запити до другого рівня, що обслуговує його в той час, коли виконує запити і викликає оновлення щодо бази даних (БД) для того, аби згенерувати інтерфейс користувача.

**Взаємодія складових у трирівневій архітектурі.**

Трирівнева архітектура (рис.1) може бути деталізована і ускладнена під час розроблення більш складних додатків, і тоді може бути доречним використання *n*-ярусного підходу, де найбільшою перевагою стане розділення бізнес-логіки на рівні додатка на більш дрібні моделі. Додатковою перевагою може стати додавання рівня інтеграції, що відокремлює рівень зберігання даних від решти рівнів із забезпеченням простого за реалізацією інтерфейсу для доступу до даних. Скажімо, клієнтські дані будуть доступні за допомогою виклику функції «list_clients()» замість того, щоб зробити SQL-запит безпосередньо до таблиці клієнтів БД. Це дозволяє замінити основну БД без її модифікації на інші рівні.

Дворівнева архітектура передбачає веб-додаток як дворівневу архітектуру із «розумним» клієнтом, який виконує всю роботу і запитує «простий» сервер, або ж «простий» клієнт, який опирається на «розумний» сервер. Тоді клієнт обробляє рівень презентації, сервер буде працює з БД (рівень зберігання), а бізнес-логіка (рівень додатку) буде частиною одного з них або обох. Такий підхід дійсно збільшує масштабованість додатків та відокремлює рівень презентації від рівня зберігання (БД) і не дозволяє здійснювати справжню спеціалізацію шарів, тому більшість програм в перспективі переросте цю модель.

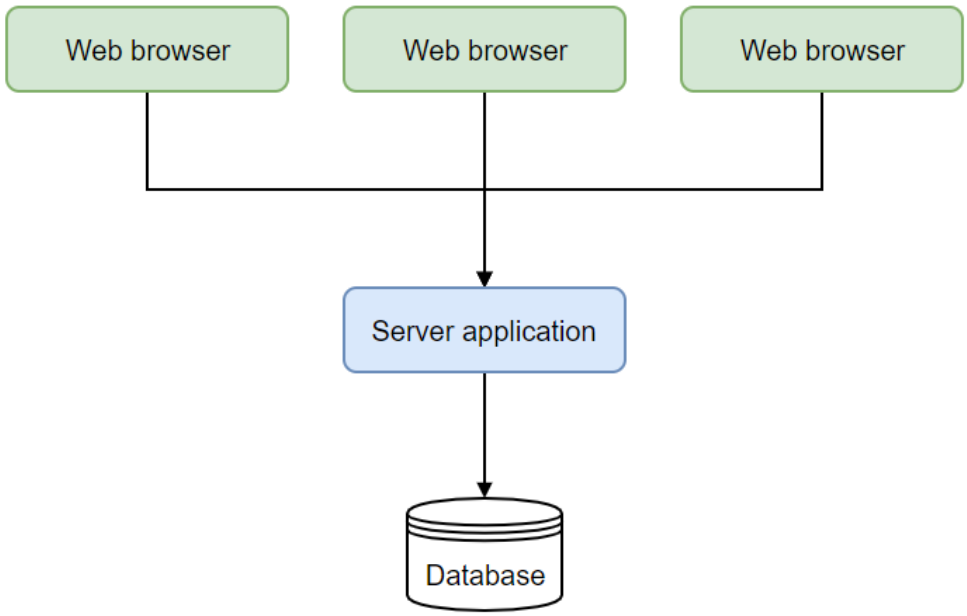


**Рисунок 1 – Трирівнева архітектура, взаємодія рівнів**

*Авторська розробка*

Для реалізації комунікації між першим та другим рівнями веб-додатку існує поняття клієнт-серверної моделі взаємодії. Клієнтом зазвичай є веб-браузер (в класичних веб-додатках), але також клієнтами можуть бути і інші додатки, в тому числі серверні. Наприклад – під час розробки додатків за сервісно-орієнтованою архітектурою, сервіс може бути одночасно і клієнтом і сервером, залежно від поточної задачі (рис. 2).

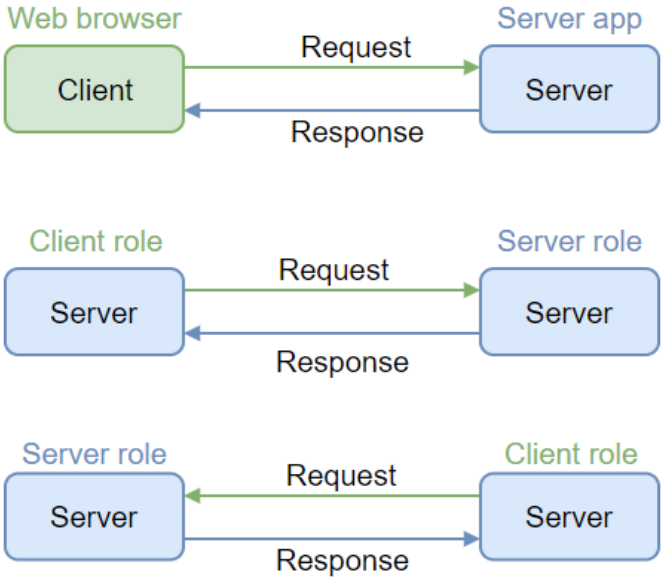


**Рисунок 2 – Ролі додатків в клієнт-серверній моделі**

*Авторська розробка*

**Клієнтський додаток (рівень презентації).**

Зазвичай для розробки рівня презентації використовують технологію SPA (Single page application). Односторінковий додаток (SPA) – це веб-додаток або веб-сайт, який взаємодіє з користувачем, динамічно переписуючи поточну сторінку, а не завантажуючи нові сторінки з сервера. Цей підхід дозволяє уникнути переривання користувацького досвіду між послідовними сторінками, роблячи програму більш схожою на настільну програму. У SPA весь необхідний код – HTML, JavaScript, і CSS – витягується з одного завантаження сторінки, або відповідні ресурси динамічно завантажуються і додаються на сторінку за необхідності, як правило, у відповідь на дії користувача. Сторінка не перезавантажується в будь-якій точці процесу, а також не здійснює передачу керування на іншу сторінку. Взаємодія з SPA часто передбачає динамічну комунікацію з веб-сервером у фоні. Якщо традиційні клієнтські додатки оперують лише HTML сторінками, завантажуючи їх з серверу, то SPA використовують технологію AJAX для отримання даних порційно, завантаживши HTML один раз (рис 3).

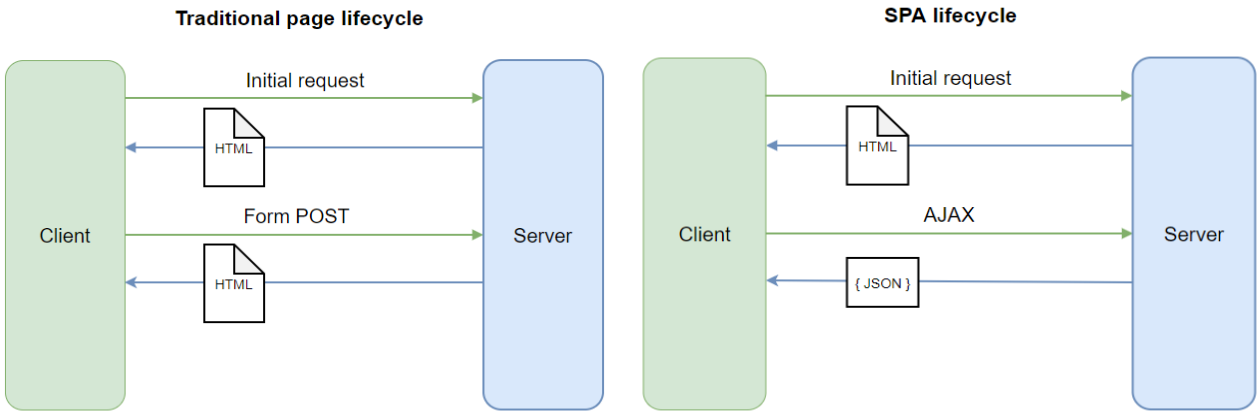


**Рисунок 3 – Порівняння алгоритмів роботи традиційних клієнтських додатків та додатків SPA**

*Авторська розробка*

Точкою ініціалізації SPA є завантаження клієнтського додатку (html, css, js), це може робити сервер (який надалі прийматиме запити від SPA) або розподілена мережа доставки контенту CDN (Content delivery network). В першому випадку навантаження лягає на сервер, що може негативно відобразитися на використанні його ресурсів, в другому – на один із серверів провайдеру CDN, який знаходиться територіально найближче до користувача, що істотно зменшує мережеві затримки, тому використання CDN є більш прийнятним для доставлення статичних файлів.

**Серверний додаток (рівень додатку).**

Серверний додаток це такий, що побудований за моделлю Request-Response (запит-відповідь) і запускається з серверного середовища та із використанням серверних технологій. Серверний додаток не має свого GUI, єдині варіанти взаємодії з ним – через консольні команди (якщо додаток надає таку можливість), та безпосередньо через веб-сервер (Web server) або сервер додатків (Application server). Основними технологіями, на базі яких можлива розробка повноцінних серверних додатків є: ASP.NET, Java Enterprise Edition, PHP, Python, Ruby, NodeJs, Golang тощо. Під час розробки архітектури таких додатків є два основних підходи: монолітна та сервіс-орієнтована архітектура.

*Монолітна програма* є самодостатньою і незалежною від інших обчислювальних додатків (рис. 4). Філософія архітектури полягає в тому, що додаток відповідає не тільки за конкретне завдання, але може виконувати всі необхідні кроки для виконання певної функції до кінця, і є «комбайном даних», а не частинами більшої системи додатків, які працюють разом. Монолітні веб-додатки масштабуються цілісно і це може привести до неефективності, так як одна частина такого додатку може бути більш навантажена за іншу, і комплексне масштабування коштуватиме набагато більше, ніж потрібно в конкретній ситуації. Перевагами ж такого підходу є тісне узгодження компонентів – немає необхідності їх окремої інтеграції, немає мережевих затримок на виконання підпрограм, тощо.

Мікросервіси – це метод розробки програмного забезпечення – варіант архітектурного стилю *сервісно-орієнтованої архітектури* (SOA), що можна розглядати як мережа слабкозв’язаних мікрододатків [7-9]. У архітектурі мікросервісів кожен сервіс вузькоспеціалізований, а протоколи – легкі (рис. 4). Перевага декомпозиції програми на різні менші служби полягає в тому, що забезпечує модульність. Це полегшує розуміння, розробку, тестування та підвищує стійкість до ерозії архітектури. Така архітектура паралелізує розробку, дозволяючи малим автономним командам розробляти, розгортати і масштабувати свої відповідні послуги незалежно. Вона також дозволяє архітектурі окремого сервісу проходити через постійний рефакторинг. Мікросервісні архітектури забезпечують безперервну доставку і розгортання (CI/CD).

У визначенні архітектури мікросервісу важливим є з'ясувати, наскільки великим має бути індивідуальний мікросервіс. Тут правильна відповідь залежить від ділового та організаційного контексту (часто організації формують «підрозділи» по 6-8 розробників). Але ключове рішення залежить від

того, наскільки «чистими» можуть бути межі сервісу. На протилежній стороні спектру вважається поганою практикою зробити службу занадто малою, оскільки тоді витрати на виконання та складність роботи можуть перекрити переваги підходу. Коли сервіси стають занадто дрібними, необхідно розглянути альтернативні підходи, наприклад, упакувати функцію як бібліотеку або помістити цю функцію в інші мікросервіси.

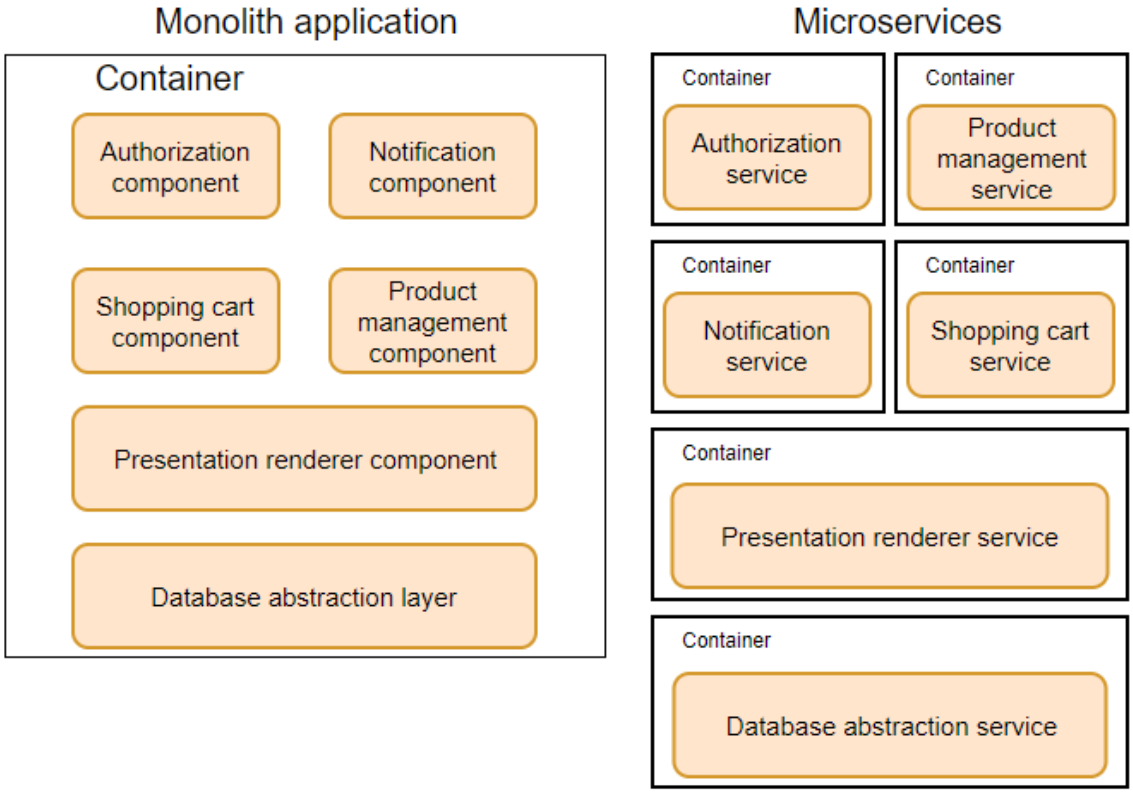


**Рисунок 4 – Порівняння монолітної та мікросервісної архітектури**
*Авторська розробка*

Комп'ютерні мікросервіси можуть бути реалізовані різними мовами програмування і використовувати різні інфраструктури. Тому найважливішим під час вибору технології є те, як мікросервіси спілкуються один з одним (синхронні, асинхронні, інтеграція інтерфейсу тощо) і протоколи, що використовуються для зв'язку (REST, обмін повідомленнями, тощо). У традиційній системі більшість технологічних варіантів, таких як мова програмування, впливають на цілі системи.

У мікросервісній архітектурі кожен екземпляр служби з'єднаний з екземпляром зворотного проксі-сервера (проксі-сервіса). Екземпляр служби та проксі-сервер спільно використовують контейнер, а контейнери керуються інструментом для оркестрування контейнерів, таким як Kubernetes. Проксі-сервіси служби несуть відповідальність за зв'язок з іншими екземплярами служби і можуть підтримувати такі можливості, як Service discovery (пошук сервісів), Load balancing (балансування навантаження), аутентифікація і авторизація, захищені комунікації та інші.

**База даних (рівень зберігання).**

Класичним рішенням для маніпуляцій із даними є реляційна БД. Програмна система, що використовується для підтримки реляційних БД, є реляційною системою управління базами даних (СУБД) і більшість із них використовують SQL для формування запитів і підтримки БД. Реляційна модель організовує дані в одній або декількох таблицях (або «реляціях» БД) стовпців і рядків з унікальним ключем, що ідентифікує кожен рядок (рядки є записами, колонки – атрибутами). Як правило, кожна таблиця/реляція є одним типом сутності. Рядки є екземплярами цього типу об'єктної сутності, а стовпці містять значення, що належать цьому екземпляру.

Кожен рядок таблиці має свій унікальний ключ. Рядки в таблиці можуть бути пов'язані з рядками в інших таблицях шляхом додавання стовпця для унікального ключа пов'язаного рядка (такі стовпці є зовнішніми ключами). Відношення даних довільної складності можуть бути представлені простим набором понять. Реляції є логічним зв'язком між різними таблицями, встановленими на основі взаємодії цих таблиць (рис. 5).

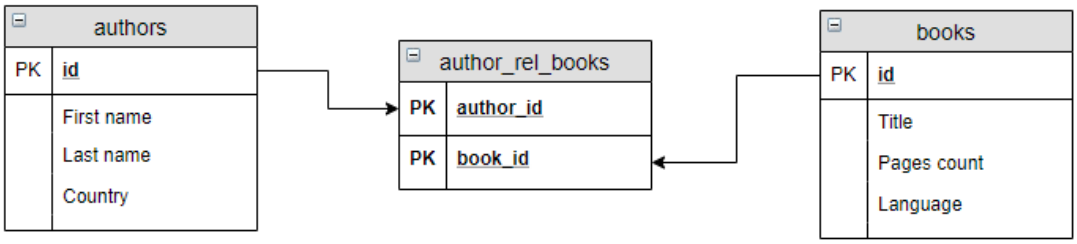


**Рисунок 5 – Схема-приклад зв'язку між таблицями в реляційній БД**

*Авторська розробка*

Існує ряд категорій БД, що лежить в проміжку між «плоскими» файлами (які не відносяться до NoSQL) і новими БД на основі графа (вважаються навіть більш реляційними, ніж стандартні реляційні БД). БД плоских файлів складається з однієї таблиці даних, яка не має взаємозв'язку (часто це текстові файли). Користувачі можуть вказувати атрибути даних, такі як стовпці та типи даних. Стандартні реляційні БД дозволяють користувачам керувати попередньо визначеними відношеннями даних у декількох БД. Популярні реляційні БД – це Microsoft SQL Server, Oracle Database, MySQL і PostgreSQL. Реляційні БД на основі хмарних обчислень або БД як служба (DBaaS) також широко використовуються, оскільки дозволяють компаніям передавати на аутсорсинг обслуговування БД, виправлення збоїв та підтримку інфраструктури. Хмарні реляційні БД – це служба реляційних БД Amazon (RDS), Google Cloud SQL, IBM DB2 в хмарі, БД Microsoft Azure SQL і служба хмарних служб Oracle Database. Бази даних NoSQL є альтернативою реляційним БД, що особливо корисно для роботи з великими наборами розподілених даних. Ці БД можуть підтримувати різноманітні моделі даних, включно з ключовими, документальними, стовпчастими та графовими форматами.

Графові БД розширюються за межі традиційних моделей реляційних БД на

основі колонок і рядків. Це БД NoSQL, що використовує вузли і ребра, які інтерпретують зв'язки між відношеннями даних і можуть виявляти нові зв'язки між даними. Графові БД є більш складними, ніж реляційні, і типовим прикладом їх використання є додатки для виявлення шахрайства (їх ще називають веб-рекомендаційними додатками).

Основними перевагами реляційних БД є можливість класифікувати та зберігати дані, які згодом можуть бути запитані та відфільтровані, щоб витягувати конкретну інформацію для звітів. Реляційні БД легко розширюються і не залежать від фізичної організації. Після створення оригінальної БД може бути додана нова категорія даних без зміни всіх існуючих програм. До інших переваг реляційної БД належать: точність: дані зберігаються лише один раз; гнучкість: складні запити легко виконувати користувачам; спільна робота: кілька користувачів працюють з однією БД; довіра: моделі реляційних БД є зрілими і добре зрозумілими; безпека: доступ до даних СУБД може бути обмеженим, щоб дозволити доступ лише окремим користувачам.

**Інфраструктура веб-додатків.**

Для доставлення визначеного компоненту додатку в робоче середовище є спеціальний підрозділ технологій – інфраструктура. Інфраструктура – це комплекс програмно-технічних засобів, які роблять можливим або спрощують доставлення (в тому числі безперервне) додатку в робоче середовище.

Останнім часом процедури доставлення ПЗ зазнали серйозних змін: якщо ще 10 років назад світом IT керували сервери з віртуалізацією і системні адміністратори; то зараз це – контейнерний підхід, процеси безперервної доставки контейнерів і фахівці з DevOps, які налаштовують допоміжні системи, замість організації всіх процесів «з нуля».

Зазвичай інфраструктура веб-додатку складається з кількох компонентів: інфраструктури самого додатку, інфраструктури БД, інфраструктури інших допоміжних систем. В залежності від потреб кожен компонент може бути розгорнуто вручну, або може бути використано готовий сервіс в форматі SaaS (Software as a service – ПЗ в ролі сервісу). Наприклад, це може бути DBaaS (Database as a service) на базі одного чи декількох хмарних провайдерів (Google Cloud SQL, Amazon RDS тощо). Кожна конкретна конфігурація підбирається спеціалістом DevOps спільно з розробниками ПЗ і представниками бізнесу, з метою визначення найбільш оптимального способу запуску додатку.

Правилом гарного тону є побудова гнучкої інфраструктури з самої першої версії продукту для можливості масштабування і переведення інфраструктури між різними провайдерами (наприклад для визначення найоптимальнішого для конкретних потреб продукту).

Для розвертання серверного веб-додатку також є можливість використати SaaS (наприклад Google App Engine), таким чином забрати з себе обов’язки підтримки, моніторингу, масштабування і відновлення додатка. Але такий підхід потребує розробки спеціального ПЗ для інтеграції розробленого додатку з SaaS системою, що в результаті ускладнює переведення такого ПЗ до іншого хостера, тому вибір використання SaaS продукту має бути аргументованим.

**Висновки**.

В цій оглядовій дослідницькій роботі ґрунтовно розглядаються механізми і алгоритми роботи серверних додатків. Тут оглянуті технології розробки клієнт-серверних додатків, технології розробки та підтримання найбільш популярних баз даних, технології збирання та розгортання веб-додатків та варіанти їх розміщення в робочому середовищі.

Основним результатом роботи є формування такого технологічного стеку роботи веб-додатків: *презентації*, *додатку* і *зберігання*. Такий підхід створює підґрунтя для подальшого наукового пошуку в напрямку розроблення додатків, кінцевим завданням яких є властивості відмовостійкості та автомасштабування.

Література:

1. Захарченко С. М. Застосування односторінкових веб-орієнтованих інтерфейсів в соціально значущих проектах. / С. М. Захарченко, Т. І. Трояновська, О. В. Бойко В. С. Рибаченко // Вісник ХНУ, №3, 2016р., с. 33-39.

2. Трояновська Т. І. Методи та засоби популяризації комерційних веб-ресурсів / Т. І. Трояновська, Л. А. Савицька, В. Ю. Тарануха // Інформаційні технології та комп'ютерна інженерія. – Вінниця, 2017. – №2, С. 23-30.

3. Трояновська, Т. Алгоритм структурованої візуалізації XML-файлів [Текст] / Трояновська Т. І., Бойко О. В. // „Intrenet-Education-Science" : Міжнародна науково-технічна конференція, 11–14 жовтня 2016 р. – Вінниця : КІВЦ ВНТУ, 2016. – С. 142–144. ISBN 966–641–102–4.

4. Гороховський О. І Інформаційна технологія доставки контенту у системах комп'ютеризованої підготовки спеціалістів. // Гороховський О. І., Азаров О. Д., Трояновська Т. І. Монографія. Вінниця : ВНТУ, 2016.–160 с.

5. Коробейнікова Т.І. Комбінований метод масштабування баз даних. / Коробейнікова Т.І., Захарченко С. М. // International scientific journal «Grail of Science» – 2022. – № 14-15 (May, 2022). – С. 320–326. ISSN: 2710–3056. ISBN 979-8-88526-799-1.

6. Коробейнікова Т.І. Відмовостійкість та автомасштабування веб-ресурсу. / Коробейнікова Т.І., Захарченко С. М. // International scientific journal «Grail of Science» – 2022. – № 14-15 (May, 2022). – С. 312–319. ISSN: 2710–3056. ISBN 979-8-88526-799-1.

7. Трояновська Т. І. Засоби та модель моніторингу даних мікросервісної системи / Т. І. Трояновська, Л. А. Савицька, В. Л. Комаров // Інформаційні технології та комп'ютерна інженерія. – Вінниця, 2018. – №1, С. 28-33.

8. Трояновська Т. І. Аналітика роботи сучасних мікросервісних систем при високих навантаженнях / Трояновська Т. І., Комаров В. Л. Зимові наукові підсумки 2017 року: II Міжнародна науково-практична інтернет-конференція: тези доповідей, Дніпро, 25 грудня 2017 р. – Ч. 1. – Дніпро: НБК, 2017, с. 26-30.

9. Коробейнікова Т. І. Удосконалений метод розробки API підвищеної швидкодії / Т. І. Коробейнікова, Л. А. Савицька // Інформаційні технології та комп'ютерна інженерія. – Вінниця, 2021. – №1 (том 50), С. 31-35.

***Abstract.*** *This work examines the mechanisms and algorithms of the server-side applications, in particular the technologies on which the applications are being developed, the technologies of the most popular databases, the technologies for assembling and deploying applications and options for placing them in the executing environment.*

***Key words:*** *web application, three-tier architecture, web application technology stack, presentation layer, application layer, storage layer, SPA, CDN, monolithic architecture, microservice, database, web application infrastructure, SaaS.*

Статтю відправлено: 29.08.2022 г.