ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)

УДК 004.75:004.053 DOI: https://doi.org/10.30837/ITSSI.2023.26.068

Н. Романків, Д. Ситніков

# АНАЛІЗ І ВИБІР МЕТОДІВ КАСТОМІЗАЦІЇ *SAAS*-РІШЕНЬ, ПОБУДОВАНИХ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ *CLOUD-NATIVE*

**Предметом дослідження** є методи кастомізації *SaaS*-рішень. **Мета статті** – визначити цілісну стратегію кастомізації *SaaS*-рішень, розроблених за допомогою технологій *cloud-native*. **Завдання**: проаналізувати сучасні підходи щодо архітектури *SaaS*-застосунків; виявити основні методи кастомізації для сучасних *SaaS*-застосунків; дослідити та встановити спосіб кастомізації інтерфейсу *SaaS*-застосунків; на підставі проведеного дослідження визначити цілісну стратегію кастомізації *SaaS*-застосунків. Упроваджуються такі **методи**: аналіз і синтез – для вивчення технологій, що використовуються для побудови *SaaS*-застосунків; абстрагування та узагальнення – для визначення загальної архітектури *SaaS*-застосунку; синтез вебтехнологій – для вибору методів кастомізації *SaaS*-застосунків та побудови алгоритму вибору методу кастомізації. Досягнуті **результати**: досліджено сучасну архітектуру *SaaS*-застосунків; обрано методи кастомізації *SaaS*-застосунків, а також визначено цілісну стратегію щодо кастомізації *SaaS*-застосунків; запропоновано метод модифікації інтерфейсу *SaaS*-застосунків, що дає змогу виконувати кастомізацію вебінтерфейсу *SaaS*-застосунку незалежно від конкретного фреймворку *front-end*. **Висновки**: сучасна архітектура *SaaS*-застосунків є комплексною та основана на мікросервісній архітектурі, підході *multi-tenant*, хмарних технологіях і веббраузері; методи кастомізації *SaaS*-застосунків мають розроблятися незалежно від певних технологічних стеків, щоб задовільнити потреби більшості або всіх користувачів *SaaS*-застосунку в кастомізації; підхід *API-first* є фундаментальним у побудові кастомізованого *SaaS*, оскільки він є основою для створення будь-якої автоматизації чи пов’язаної бізнес-логіки; *webhooks* є найкращою опцією для реалізації підписки на події, що відбуваються в *SaaS*-застосунку, незалежно від технологічного стеку; визначена стратегія кастомізації *SaaS*-застосунків дає змогу робити кастомізацію незалежно від певного технологічного стеку й, окрім реалізації бізнес-логіки, також покриває модифікацію інтерфейсу.

**Ключові слова**: *SaaS*; кастомізація; *web; webhooks, Iframe, API-first*.

## Вступ

Сучасні технології та інтернет дали змогу моделі *SaaS* [1] (програмне забезпечення як послуга) з’явитися та домінувати у світі. Чимало багатомільярдних компаній були побудовані на *SaaS*, зокрема *Netlifx, Amazon, Facebook, Airbnb.* А згідно з 360 *Research Reports* компанія *SaaS* у глобальному ринку оцінювалася в 1 777 098 доларів 2021 р., а також продовжуватиме зростати й досягне 1 177 188 доларів до 2027 р. Побудова сучасного *SaaS*-застосунку зазнає чимало технічних проблем, які необхідно вирішити, особливо в сучасному світі з широким упровадженням доступних хмарних технологій і нових технологій *cloud-native* [2], що продовжують з’являтися.

Окрім того, дуже високої популярності набуває *MACH Allience* [3] й використання архітектури *composable ecommerce* [4], що допомагає бізнесу будувати систему як конструктор, який збирається із різних *SaaS*-систем і в якому окремі системи відповідають за різні технічні можливості електронної комерції, таких як пошук, персоналізація, керування замовленнями тощо.

Хоча такий підхід дає змогу бізнесу бути гнучким, а також прискорює розроблення IT-системи, водночас він обмежує розробників системи, оскільки сторонні сервіси є *black box* і зазвичай позбавляють можливості модифікувати їх системи. Важливо, щоб *Enterprise*-системи мали доступ до налаштувань, але часто вимоги до налаштувань від клієнта виходять за межі того, що вендор програмного забезпечення може передбачити наперед. В епоху використання локальних систем клієнти робили глибокі налаштування поза передбаченням вендора, безпосередньо змінюючи його вихідний код, а потім будуючи та експлуатуючи вендор у власному дата-центрі. Коли програмне забезпечення перейшло до хмарного багатокористувацького *SaaS*, клієнтам уже не можна безпосередньо змінювати вихідний код вендора, оскільки той самий екземпляр коду використовується декількома клієнтами одночасно в процесі виконання. Отже, постає питання розроблення та проєктування методів кастомізації *SaaS*-систем, які давали б змогу інженерам і користувачам систем модифікувати її під свої потреби, без впливу на інших користувачів *SaaS*-системи.

ISSN 2522-9818 (print)
*Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4 (26)* *ISSN 2524-2296 (online)*

**Аналіз останніх досліджень і публікацій**

Аналіз джерел довів, що кастомізації *SaaS*-систем приділялась увага в багатьох наукових працях, і це також підтверджує актуальність окресленої проблеми. Так, Ху Сонг, Франк Чаувел та інші провели ґрунтовне дослідження щодо підтримки кастомізації *SaaS* [5]. Автори вивчають очікування консультантів, які спеціалізуються на кастомізації корпоративних програмних систем. Кастомізація традиційно ґрунтується на припущенні, що програмне забезпечення встановлено у власному дата-центрі клієнта, тому клієнт має повний контроль над системою та може вільно її змінювати. Це припущення вже не правильне, оскільки корпоративне програмне забезпечення переходить до багатокористувацького *Software as a Service*, бо воно працює в хмарі та контролюється вендором *SaaS*. Отже, дехто вважає, що налаштування *SaaS* є занадто складним і від нього необхідно відмовитися. Водночас практика в галузі *SaaS* визнає перевагу конфігурації над налаштуванням. Два вендори програмного забезпечення, які взяли участь у цьому дослідженні, також розпочали з підтримки конфігурації на своїх рішеннях *SaaS*. Однак виявляється, що чимало їхніх партнерів (користувачів *SaaS*-системи) мають інший погляд, і тому ці вендори постійно отримують запитання щодо підтримки налаштувань на своїх продуктах *SaaS*. Також у роботі [5] подається теорія, побудована на висновках, що кастомізація залишається важливим для *SaaS*, однак багатокористувацькість (*multi-tenancy*) змінює відповідальність зацікавлених сторін. Партнери (користувачі *SaaS*) тепер не можуть самостійно змінити продукт, щоб досягти побажань своїх клієнтів, але мають покладатися на вендорів *SaaS* для розроблення та впровадження змін і нових функції системи. Зрештою партнерам необхідно відмовитися від старого методу вільного розроблення з довільними модифікаціями, розсіяними в продукті. Замість цього вони мають прийняти новий формат розроблення з більшою кількістю обмежень та участю вендорів і приділити значну увагу бізнесу. Ця трансформація конфліктує з традиційними інтересами партнерів, а саме: мінімізацією витрат для клієнтів, безшовною інтеграцією власного коду з основним продуктом і забезпеченням гнучкості. Тому критичним фактором успіху для екосистеми налаштувань є те, що вендори мають пропонувати не лише висококастомізований *SaaS*, але й ефективні засоби підтримки, які будуть відповідати новим видам налаштувань і водночас компенсувати ці основні інтереси. Необхідність у кастомізації виникає через розрив між клієнтами та вендорами: останні зосереджуються на загальному корпоративному програмному забезпеченні, але не мають знань і ресурсів, щоб зрозуміти бізнес кожного окремого клієнта. Утім конфігурація не може замінити кастомізацію, оскільки вона ігнорує цей розрив, намагаючись зрозуміти, що вендори здатні передбачити та реалізувати всі ймовірні функції.

Також Ральф Мітзнер, Андреас Матзгер та інші в роботі [6], наводять такий аргумент: щоб використовувати економію масштабу, вендорам *SaaS* необхідно успішно привертати значну кількість користувачів для своїх *SaaS*-застосунків. Очевидно, що функціональність і якість, які індивідуальні клієнти очікують від програмного застосування, можуть відрізнятися. Як наслідок, постачальникам *SaaS* необхідно брати до уваги варіативні вимоги широкого спектра потенційних клієнтів. Це означає, що *SaaS*-застосунки мають дозволяти конфігурацію та налаштування для кожного користувача.

Окрім цього, Веі Сун та інші в роботі [7] дійшли висновку, що конфігурація та кастомізація є критичними аспектами для вендорів *SaaS* у процесі розроблення своїх систем. Хоча найважливіше – це добре розробити функції *SaaS*-застосунку, щоб задовольняти якнайбільше вимог у цільовому сегменті клієнтів і галузі застосування; рівень можливостей, наданих для конфігурації та кастомізації, дає ключові конкурентні переваги на ринку. У багатьох ситуаціях, якщо дуже важко розробити *SaaS*-застосунок як стандартну пропозицію для більшості клієнтів, висока здатність до конфігурації та кастомізації стане ключовим фактором успіху.

У студіях [8, 9] автори досліджували використання мікросервісів для кастомізації *SaaS*-систем. Такий метод кастомізації пропонується для застосування, коли необхідно зробити глибоку кастомізацію.

Підхід, згаданий у праці [8], занадто детально прив’язаний до стеку технологій *.Net* та розроблений на припущені, що вендор *SaaS* надасть доступ до коду системи, який здебільшого є неможливим, оскільки для компаній, що продають *SaaS*, код програмної системи – це основний актив бізнесу, який приносить прибуток.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)

Водночас підхід, розроблений у роботі [9], пропонує сучасний *cloud-native* та динамічний підхід до використання мікросервісів. У такому підході кожна кастомізація реалізується за допомогою мікросервісу. Робота демонструє використання синхронного виклику, окремої бази даних *NoSQL* та середовища на основі *Docker* для реалізації такої архітектури. Але цей метод не покриває підхід до модифікації інтерфейсу *SaaS*-застосунку й побудований на особливому фреймворку *MiSC-Cloud*, що робить такий підхід майже непридатним до застосування.

Еспен Нордлі в роботі [10] розглядає можливості глибокої модифікації завдяки подіям. Архітектура застосунку також залежить від специфічного фреймворку *MiSC-Cloud* й побудована на припущені, що вендор застосовуватиме специфічний *event bus* (шина подій) й надасть доступ до шини для мікросервісів, які мають виконувати кастомізацію способом підписки на події та їх модифікацію.

## Визначення не вирішених раніше частин загальної проблеми. Мета роботи, завдання

Отже, аналіз літературних джерел довів, що в роботах, присвячених кастомізації *SaaS*-систем, описуються дуже специфічні методи модифікації, основані на конкретному наборі технологій, що є помилковим і не може масштабуватися на будь-який *SaaS*. Також немає цілісної стратегії щодо кастомізації *SaaS*-застосунків, яка передбачала б сучасні й різнобічні методи для модифікації. Окрему увагу необхідно приділити кастомізації інтерфейсу *SaaS*, оскільки це важливий аспект для модифікації *SaaS* під потреби окремих клієнтів компанії, який був упущений у згаданих роботах.

**Мета статті** – визначення цілісної стратегії кастомізації *SaaS*-систем. Для досягнення поставленої мети потрібно виконати такі **завдання**: проаналізувати сучасні підходи щодо архітектури *SaaS*-застосунків; визначити основні методи кастомізації для сучасних *SaaS*-застосунків; дослідити та обрати спосіб модифікації інтерфейсу *SaaS*-застосунків; на підставі проведеного дослідження визначити цілісну стратегію кастомізації *SaaS*-застосунків.

## Матеріали й методи

Методологічною основою роботи є такі **методи** наукового дослідження: аналіз і синтез застосовуються для вивчення технологій, що впроваджуються з метою побудови *SaaS*-застосунку; абстрагування та узагальнення – для визначення загальної архітектури *SaaS*-застосунку; синтез вебтехнологій – для вибору методів кастомізації *SaaS*-застосунків і побудови алгоритму вибору методу кастомізації.

## Результати досліджень та їх обговорення

Сучасна архітектура *SaaS*-застосунків є складною та комплексною, що підтверджується статтями [11, 12], оскільки впровадження сучасних *SaaS*-застосунків супроводжується такими труднощами:

– забезпечення високого рівня безпеки для збереження конфіденційності та цілісності даних користувачів;

– забезпечення ефективного масштабування, щоб застосунок міг витримати зростання обсягу користувачів і бізнесу;

– реалізація зручних інтеграцій з іншими системами та послугами для покращення функціональності;

– розроблення механізмів відмовостійкості для забезпечення неперервності роботи застосунку в умовах можливих збоїв;

– ефективне керування витратами та оптимізація використання ресурсів для забезпечення ефективності бізнес-моделі;

– забезпечення швидкого розроблення та впровадження нового функціоналу для відповіді на мінливі потреби ринку.

Отже, розглянемо детально підходи, що застосовуються в сучасній архітектурі *SaaS*-застосунків.

Більшість успішних *SaaS*-застосунків, наприклад *Netflix, Twitter, Slack,* використовують мікросервісну архітектуру, що є достатньо ефективною. Гнучкість і масштабованість є ключовими перевагами мікросервісної архітектури в контексті *SaaS*. Кожен мікросервіс може функціонувати незалежно, що дає змогу розробникам ефективно впроваджувати новий функціонал та масштабувати окремі частини системи відповідно до потреб користувачів. Незалежність та ізоляція мікросервісів стає важливою особливістю для *SaaS*. Це дає змогу забезпечити високий рівень ізоляції між різними функціональними частинами системи, що важливо для забезпечення безпеки та конфіденційності даних користувачів. Можливість швидко розгортати

*Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4 (26)* *ISSN 2522-9818 (print)* *ISSN 2524-2296 (online)*

та оновлювати окремі мікросервіси без впливу на інші компоненти системи є важливим аспектом для реалізації гнучких методологій розроблення та впровадження нового функціоналу в *SaaS*-застосунках. Однак попри всі переваги мікросервісна архітектура стикається з викликами. Складність управління багатьма мікросервісами та їх координація може виявитися непростим завданням, особливо в розробленні та експлуатації великих *SaaS*-застосунків. Важливо враховувати збільшені витрати на інфраструктуру через підтримку багатьох мікросервісів, особливо за умови великої кількості користувачів та операцій. Складність тестування та відлагодження може спричинити розподілений характер мікросервісів, що потребує ретельного планування та впровадження відповідних інструментів для забезпечення якості застосунку.

Окрім цього, сучасні *SaaS*-застосунки використовують архітектуру *multi-tenant* [13, 14]. Зазначена архітектура є ключовою для реалізації *SaaS*-застосунків, оскільки вона дає змогу ефективно обслуговувати багато користувачів на одній і тій самій інфраструктурі та програмному забезпеченні. У *multi-tenant* інфраструктурні ресурси використовуються спільно, але водночас дані користувача ізольовані. Такий підхід дозволяє економити ресурси, оскільки різні користувачі застосовують спільну інфраструктуру та обслуговуються однією версією програмного забезпечення. Для постачальників *SaaS* це означає, що вони можуть забезпечити більш ефективне використання ресурсів і зменшити загальні витрати на утримання системи та збільшити прибутки. Також, оскільки в такій архітектурі використовується один екземпляр програмного забезпечення, це полегшує управління та оновлення системи. Постачальники *SaaS* можуть ефективно впроваджувати нові версії та функціонал, не припиняючи роботу інших користувачів.

Під час створення методів модифікації необхідно брати до уваги те, що одна й та сама інфраструктура *SaaS*-застосунку використовується одночасно безліччю користувачів, а тому модифікації на рівні інфраструктури *SaaS*-застосунку для досягнення кастомізації є неможливими або занадто складними для реалізації. А отже, будь-яка модифікація має відбуватися поза рівнем інфраструктури *SaaS*-застосунку.

Також у розробленні *SaaS*-застосунків використовують технології *cloud-native*, оскільки впровадження хмарних сервісів (зокрема *AWS, Azure, Google Cloud*) має чисельні переваги для розроблення та експлуатації *SaaS*-застосунків. Розглянемо деякі з них більш детально.

– Еластичність. Хмарні платформи дають змогу легко змінювати обсяг ресурсів залежно від навантаження. Це допомагає збільшувати чи зменшувати потужність обчислення та інші ресурси відповідно до потреб.

– Масштабованість. Хмарні сервіси дають змогу ефективно масштабувати застосунки горизонтально або вертикально, щоб забезпечити високу доступність і продуктивність.

– Гнучкість. Хмарні рішення допомагають легко адаптувати конфігурацію та параметри інфраструктури під потреби конкретного *SaaS*-застосунку.

– Автоматизація. Засоби автоматизації в хмарних сервісах спрощують розгортання, моніторинг, інтеграцію та керування застосунками, що полегшує розроблення та експлуатацію.

– Хмарні технології дають змогу користувачам отримати доступ до *SaaS*-застосунків із будь-якого місця світу, де є інтернет. Це робить робочий процес більш гнучким і мобільним.

– Відсутність потреби в інвестуванні у власну апаратуру та інфраструктуру допомагає знизити витрати на створення та утримання інфраструктури.

– Хмарні послуги дають змогу швидко впроваджувати нові версії застосунків і вносити зміни в реальному часі.

Варто приділити особливу увагу розподілу ринку хмарних сервісів між основними провайдерами зазначених послуг. Інформація про розподіл [15] подається в табл. 1. Аналізуючи показники з розподілу ринку між різними провайдерами, можемо дійти висновку, що методи кастомізації мають бути *cloud-agnostic*, тобто не прив'язуватися до певної публічної хмари. Або якщо й розробляти специфічний для публічної хмари метод модифікації, то це мають бути найбільші гравці на ринку, а саме *AWS, GCP, Azure,* щоб покрити найбільшу долю ринку із найменшими витратами на розроблення та підтримку таких методів.

Тепер, коли ми подали ключові елементи архітектури *back-end SaaS*-застосунків, необхідно також розглянути архітектуру *front-end.*

*SaaS*-модель стала можлива завдяки активному розвитку вебтехнологій, що дають змогу завантажити користувачам застосунок безпосередньо у веббраузер без необхідності встановлювати будь-що на свій персональний комп'ютер чи мобільний пристрій.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)
Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)

Отже, традиційний *SaaS*-застосунок оснований саме на вебтехнології для реалізації *front-end*, тобто клієнтської частини застосунку.

**Таблиця 1**. *Розподіл ринку між провайдерами хмарних послуг*

| Провайдер хмарних послуг | Відсоток ринку станом на 2023 р. |
|---|---|
| AWS | 32 |
| Azure | 22 |
| GCP | 11 |
| Alibaba Cloud | 4 |
| IBM Cloud | 3 |
| SalesForce | 3 |
| Oracle | 2 |
| Tencent Cloud | 2 |

Застосування браузера для доступу *SaaS* має численні переваги, що сприяють зручності та доступності для користувачів. Перелічимо деякі ключові причини.

– Користувачі можуть отримати доступ до *SaaS*-застосунків у будь-якому місці, де є інтернет, використовуючи будь-який пристрій з веббраузером. Це робить використання застосунків більш зручним і мобільним.

– Браузерний доступ до *SaaS* робить застосунки незалежними від конкретної операційної системи. Це означає, що користувачі можуть застосовувати *SaaS* як на комп’ютерах, так і на мобільних пристроях, незалежно від конкретного середовища.

– Користувачі мають миттєвий доступ до оновлень і нових функцій, оскільки оновлення відбуваються на рівні серверів *SaaS*, і не потрібно чекати на встановлення нових версій на локальних пристроях.

– Постачальники *SaaS* можуть здійснювати оновлення та покращення централізовано на своїх серверах, замість того, щоб вимагати в користувачів регулярно встановлювати нові версії застосунків.

– Використання браузера дає змогу уникнути потреби в установленні та оновленні спеціалізованого клієнтського програмного забезпечення. Користувачам просто потрібно відкрити браузер та увійти в обліковий запис *SaaS*.

*JavaScript* є основною мовою для розроблення клієнтської частини веб-, а отже, *SaaS*-застосунків. Вона дає змогу розробникам створювати динамічні та інтерактивні інтерфейси, що покращують користувацький досвід. Хоча всі або принаймні більшість вебзастосунків використовують *JavaScript* для побудови клієнтської частини, у сучасній веброзробці домінують фреймворки *React, Angular, Vue* тощо. Популярність фреймворків *front-end* подається в табл. 2 [16].

**Таблиця 2**. *Розподіл ринку між фреймворками front-end*

| Провайдер хмарних послуг | Відсоток ринку станом на 2023 р. |
|---|---|
| React | 37 |
| GSAP | 14 |
| Vue.js | 11 |
| Styled-components | 8 |
| Backbone.js | 7 |
| Require.js | 6 |
| Emotion | 6 |
| AngularJS | 4 |
| Handlebars | 4 |
| Stimulus | 4 |

Тобто під час розроблення методів модифікації інтерфейсу необхідно зважати, що користувачі *SaaS*-застосунків в ідеалі мають давати змогу використовувати будь-який фреймворк для кастомізації вебчастини, щоб задовільнити потребу більшості користувачів *SaaS*-застосунків, без необхідності наймати розробника із знанням однієї технології, тільки заради кастомізації застосунку.

Отже високорівнева архітектура *SaaS*-застосунку має такі ключові елементи (рис. 1):

– мікросервіси;
– архітектура *multi-tenant*;
– *cloud-native*;
– браузер як клієнтська частина.

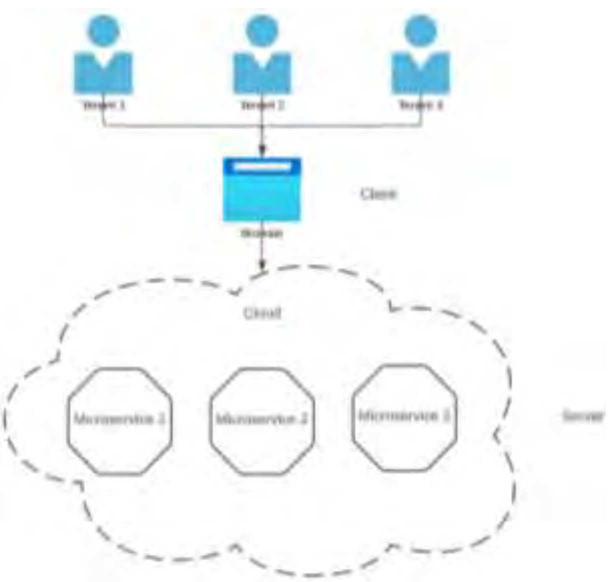

**Рис. 1**. Високорівнева архітектура *SaaS*-застосунку

ISSN 2522-9818 (print)
*Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4 (26)* ISSN 2524-2296 (online)

Для вибору методів модифікації *SaaS*-застосунків і визначення цілісної стратегії кастомізації спершу визначимо потреби для модифікації *SaaS*-застосунків. Такі потреби можуть виникати з різноманітних причин, пов'язаних з особливими вимогами користувачів та бізнес-потребами. Перелічимо деякі основні потреби:

– кастомізація може бути необхідною для того, щоб адаптувати *SaaS*-застосунок до конкретних бізнес-процесів і потреб компанії. Це може передбачати налаштування робочих потоків, структури даних та автоматизації завдань;

– компанії можуть мати власні системи, інструменти чи сервіси, з якими *SaaS*-застосунок має інтегруватися;

– підприємства можуть потребувати оптимізації робочих процесів. Це передбачає кастомізацію застосунку для полегшення виконання завдань та підвищення продуктивності;

– користувачі можуть мати різні вподобання та потреби. Модифікація інтерфейсу, функціональності або налаштувань дає змогу користувачам адаптувати застосунок до своїх власних потреб і стандартів.

Фундаментальним підходом у побудові кастомізованого *SaaS* вважаємо використання підходу *API-first* у розробленні *SaaS*-застосунку. Підхід *API-first* передбачає, що дизайн і розроблення *API* (інтерфейсу програмування застосунків) визначається та впроваджується перед створенням інших частин застосунку. Цей підхід має декілька переваг, особливо для *SaaS*-продуктів, але в контексті кастомізації *SaaS*-застосунків для нас важливо, що зосередження на розробленні *API* сприяє зручній внутрішній інтеграції між різними частинами системи. Одночасно це також полегшує забезпечення зовнішньої інтеграції для інших сервісів і розробників.

Наявність *API* в *SaaS*-застосунку дає змогу його клієнтам використовувати *API* для кастомізації такими способами:

– користувачі можуть застосовувати *API* для інтеграції *SaaS*-застосунку зі своїми власними даними чи іншими системами;

– *API* може використовуватися для внесення власної бізнес-логіки чи автоматизації в робочі потоки *SaaS*-застосунку;

– за допомогою *API* можна створювати власні звіти та аналітичні інструменти, використовуючи дані, що надає *SaaS*-застосунок.

Для розроблення публічного *API* необхідно брати до уваги, що *API* має бути незалежним від мови програмування чи технологічного стеку, яким послуговуються клієнти, щоб гарантувати можливість інтеграції з *API* для кожного користувача *SaaS*-застосунку. *REST API* [17] є найкращим вибором для реалізації публічного *API*, оскільки:

– *REST (Representational State Transfer)* – простий і легкий для розуміння стандарт. Він використовує стандартні *HTTP*-методи (*GET, POST, PUT, DELETE*) та оснований на стандарті ресурсів (ресурси, що можуть бути подані та змінені за допомогою *URI*);

– *REST* – стандарт для взаємодії в мережі Інтернет, його підтримка вбудована у більшість сучасних технологій та бібліотек. Це робить його легко інтегрованим і доступним для різних платформ і мов програмування;

– *REST API* природно інтегрується з архітектурою вебзастосунків. Користувачі можуть легко взаємодіяти з *REST API* з допомогою стандартних веббраузерів, що робить його доступним для великої аудиторії.

Окрім *API*, також необхідно вирішити проблему відстежування та реагування на події в *SaaS*-застосунку, оскільки це базова потреба в разі реалізації будь-якої бізнес-логіки в застосунку. Зважаючи, що використання інфраструктурних змін не можливе з причин, описаних вище, необхідно знайти підхід, який би дав змогу отримувати події із *SaaS*-системи незалежно від певної публічної хмари, брокера подій чи інфраструктури.

Проаналізувавши можливі опції, дійшли висновку, що найкращим варіантом для реалізації підписки на події в *SaaS*-застосунку є *webhook* [18] – механізм, який дає змогу одній системі (часто вебсерверу) автоматично надсилати дані або повідомлення іншій системі за умови виникнення якоїсь події. Коли в системі, що підтримує вебхуки, відбувається певна подія, вона надсилає *HTTP*-запит (зазвичай *POST*-запит) іншій системі за попередньо визначеним *URL*. Основні властивості вебхуків:

– допомагають системам взаємодіяти асинхронно. Замість того щоб постійно запитувати іншу систему про оновлення, система отримує повідомлення тільки тоді, коли відбувається певна подія;

– їх використовують для реагування на конкретні події в системі. Наприклад, вебхук може бути застосований для повідомлення про створення нового користувача, завершення оплати чи оновлення даних;

– спрощують інтеграцію між різними системами, оскільки вони передають дані безпосередньо

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)*

від одного сервера до іншого в разі виникнення певної події;

– дають змогу отримувати інформацію майже в реальному часі. Це особливо важливо для систем, де потрібно негайно реагувати на події;

– допомагають реалізувати будь-якою мовою програмування чи на стеку технологій як для сервера, так і клієнта.

Реалізація надійного й масштабованого механізму вебхуків є питанням, що необхідно детально розглянути окремо. Зазначена реалізації має відповідати таким вимогам:

– процес підписання на вебхуки має бути доступним з допомогою *API* або *UI*. Також має бути можливість конфігурувати кожну окрему підписку на те, які події ми хочемо отримати;

– доставка подій до вебхука має бути надійною, і необхідна стратегія для повторення спроб доставки, якщо сервіс вебхука був тимчасово не досяжний;

– інтерфейс *SaaS*-застосунку має давати змогу переглядати історію відправлень подій до вебхуків.

Як вже зазначалося вище, для кастомізації інтерфейсу необхідно визначити такий метод, який би дав змогу використовувати будь-який фреймворк для кастомізації. Серед наявних можливостей веббраузера є лише один інструмент, що задовольняє такі потреби. Це *Inline Frame*, або просто *iframe*, – HTML-тег, який допомагає вбудовувати інший документ *HTML* у поточний документ. Цей тег є частиною веброзмітки й використовується для створення вкладених вікон (фреймів), що можуть мати вміст іншого ресурсу або сторінки.

Тобто для реалізації кастомізації інтерфейсу пропонується застосовувати вбудовані вебсторінки. Отже, користувач, який хоче кастомізувати інтерфейс застосунку, може зробити це, застосовуючи будь-який фреймворк. Для цього необхідно буде облаштувати хостинг вебсторінки, який підключатиметься в *iframe*.

Цей підхід можна поєднувати із підходом *API-first*, оскільки вебсторінка, завантажена в *iframe,* має змогу використовувати публічне *API SaaS*-застосунку для виконання бізнес-функцій тощо. Але щоб такий підхід працював, вебсторінка, завантажена в *iframe*, має авторизуватися із *SaaS*-вебзастосунком та отримати необхідну інформацію про конфігурацію тощо. Для вирішення цієї проблеми пропонується використовувати стандартний *API* веббраузера, що уможливлює спілкування основної вебсторінки з вбудованою. Вебсторінка, завантажена в *iframe*, здатна прослуховувати повідомлення, відправлені основною сторінкою, використовуючи підписку на події в об’єкта *window*, і надсилати повідомлення основній сторінці, застосовуючи метод *postMessage*. Водночас аналогічні дії може виконувати основна сторінка щодо вбудованої в *iframe*. Так, під час завантаження сторінки можливе виконання ініціалізації та передачі конфігурації та необхідної авторизаційної інформації до вбудованої сторінки.

Отже, високорівнева архітектура, що зображує всі три розглянуті методи кастомізації *SaaS*-застосунків, подана на рис. 2. Високорівнева архітектура відображає такі елементи:

– *Client, Static Hosting for iframe* – хостинг, що містить статичні файли, необхідні для роботи вебсторінки, яку завантажують в *iframe* для кастомізації інтерфейсу;

– *Client Web Server* – сервіс *back-end*, що містить бізнес-логіку, яку хочуть реалізувати клієнти *SaaS*-застосунку. Цей вебсервер отримує вебхук повідомлення від *SaaS Webhook Delivery Servers* та реагує на події. Водночас цей вебсервер використовує *SaaS API* для реалізації бізнес-логіки;

– *SaaS Webhook Delivery Servers* – сервери *back-end SaaS*-застосунку, відповідальні за доставку повідомлень до вебхуків;

– *SaaS API* – сервери *back-end SaaS*-застосунку, відповідальні за реалізацію публічного *API SaaS*-застосунку.

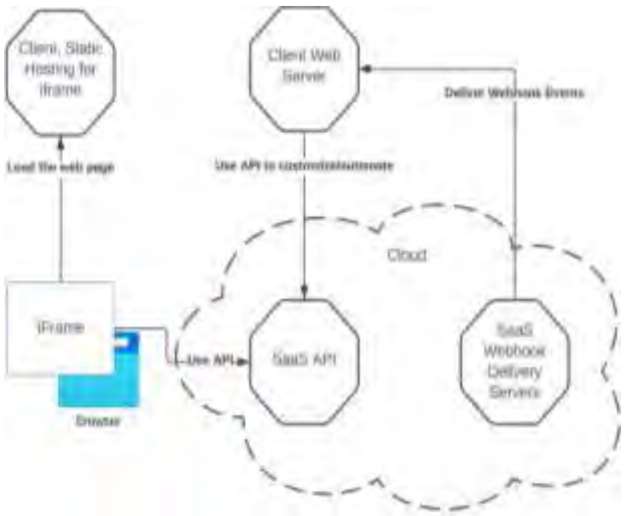


**Рис. 2**. Методи кастомізації *SaaS*-застосунків

Також для прийняття рішення щодо того, коли та який саме метод кастомізації з трьох обраних необхідно використовувати, було розроблено спеціальний алгоритм (рис. 3).

ISSN 2522-9818 (print)
*Сучасний стан наукових досліджень та технологій в промисловості. 2023. № 4 (26)* ISSN 2524-2296 (online)

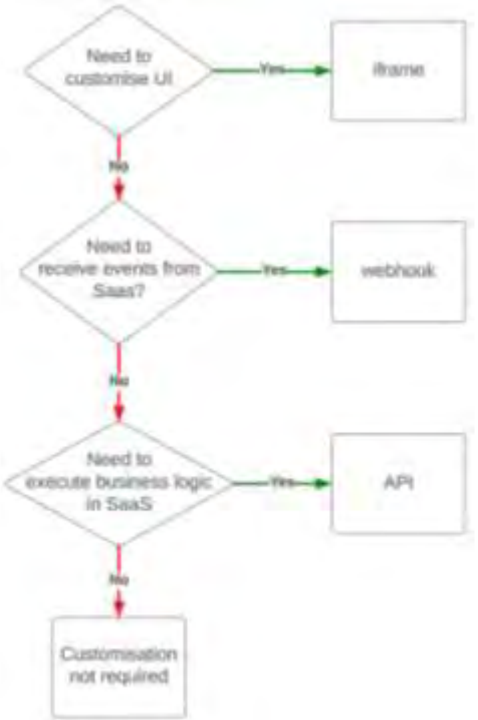


**Рис. 3**. Алгоритм вибору методів для кастомізації *SaaS*-застосунку

### Висновки й перспективи подальшого розвитку

Під час дослідження розроблено цілісну концепцію кастомізації *SaaS*-застосунків. За результатами роботи можна зробити такі висновки:

– аналіз джерел довів, що проблема кастомізації *SaaS*-застосунків є актуальною;

– сучасна архітектура *SaaS*-застосунків є комплексною та будується на мікросервісній архітектурі, підході *multi-tenant*, використанні хмарних технологій та веббраузера;

– методи кастомізації *SaaS*-застосунків мають розроблятися незалежно від певного технологічно стеку, щоб задовільнити потреби більшості чи всіх користувачів *SaaS*-застосунку в кастомізації;

– підхід *API-first* є фундаментальним у побудові кастомізованого *SaaS*, оскільки він фундаментальний для розроблення будь-якої автоматизації чи бізнес-логіки;

– вебхуки є найкращою опцією для реалізації підписки на події, що відбуваються в *SaaS*-застосунку незалежно від технологічного стеку;

– *Iframe* є найкращою опцією для реалізації кастомізації інтерфейсу вебзастосунку, оскільки уможливлює використання будь-якого фреймфорку для побудови інтерфейсу.

Для демонстрації всіх трьох опцій розроблено високорівневу діаграму, що зображує використання трьох методів кастомізації, а також створено алгоритм вибору методу кастомізації. Зазначена стратегія кастомізації *SaaS*-застосунків дає змогу виконувати кастомізацію незалежно від певного технологічного стеку й, окрім реалізації бізнес-логіки, також покриває модифікацію інтерфейсу завдяки використанню *iframe*.

Для подальшого розвитку поставленого завдання необхідно розглянути підходи, які б уможливлювали кастомізацію поведінки *API SaaS*-застосунку без зміни коду чи інфраструктури *SaaS*-застосунку, а також спроєктувати надійний і масштабований механізм вебхуків.

### Список літератури

1. Michael J. Kavis Architecting the Cloud Design Decisions for Cloud Computing Service Models (SaaS, PaaS, and IaaS): Wiley, 2014. 224 p. URL: https://www.wiley.com/en-us/Architecting+the+Cloud%3A+Design+Decisions+for+Cloud+Computing+Service+Models+%28SaaS%2C+PaaS%2C+and+IaaS%29-p-9781118617618 (дата звернення: 27.11.2023)

2. Tomas Erl, Ricardo Puttini, Zaigham Mahmood. Cloud Computing, Concepts, Technology & Architecture: Pearson, 2013. 747 p. URL: https://www.redalyc.org/pdf/6380/638067279007.pdf (дата звернення: 27.11.2023)

3. MACH Allience. URL: https://en.wikipedia.org/wiki/MACH_Alliance (дата звернення: 27.11.2023)

4. Composable Commerce. URL: https://www.elasticpath.com/composable-commerce (дата звернення: 27.11.2023)

5. Hui Song, Franck Chauvel, Arnor Solberg, Bent Foyn, Tony Yates. How to support customization on SaaS: A Grounded Theory from Customisation Consultants. *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 2017. P. 247–249. DOI: https://doi.org/10.1109/ICSE-C.2017.136

6. Ralph Mietzner, Andreas Metzger, Frank Leymann, Klaus Pohl. Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications. *2009 ICSE Workshop on Principles of Engineering Service Oriented Systems.* 2009. 814 p. DOI: https://doi.org/10.1109/PESOS.2009.5068815

7. Wei Sun, Xin Zhang, Chang Jie Guo, Pei Sun, Huis Su. Software as a Service: Configuration and Customization Perspectives. *2008 IEEE Congress on Services Part II (services-2 2008).* 2008. 2946 p. DOI: https://doi.org/10.1109/SERVICES-2.2008.29

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2023. No. 4 (26)

8. Franck Chauvel, Arnor Solberg. Using Intrusive Microservices to Enable Deep Customization of Multi-tenant SaaS. *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC).* 2018. 225 p. DOI: https://doi.org/10.1109/QUATIC.2018.00015

9. Hui Song, Phu H. Nguyen, Frank Chauvel, Jens Glattetre, Thomas Schjerpen Customizing Multi-Tenant SaaS by Microservices: A Reference Architecture”. *2019 IEEE International Conference on Web Services (ICWS).* 2019. 710 p. DOI: https://doi.org/10.1109/ICWS.2019.00081

10. Espen Tonnessen Nordli, Phu H. Nguyen, Franck Chauvel, Hui Song. Event-Based Customization of Multi-tenant SaaS Using Microservices. *2020 International Conference on Coordination Languages and Models.* 2020. P. 171–180. DOI: https://doi.org/10.1007/978-3-030-50029-0_11

11. What is SaaS Architecture? 10 Best Practices For Efficient Design. URL: https://www.cloudzero.com/blog/saas-architecture/ (дата звернення: 29.11.2023)

12. SaaS application architecture best practices. URL: https://acropolium.com/blog/saas-app-architecture-2022-overview/ (дата звернення: 29.11.2023)

13. Eng Lieh Ouh, Benjamin Kok Siew Gan. An Exploratory Study of Architectural Style and Effort Estimation for Multi-Tenant Microservices-Based Software as a Service (SaaS). *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C).* 2023. 103 p. DOI: https://doi.org/10.1109/ICSA-C57050.2023.00043

14. Wei-Tek Tsai, Peide Zhong. Multi-tenancy and Sub-tenancy Architecture in Software-as-a-Service (SaaS). *2014 IEEE 8th International Symposium on Service Oriented System Engineering.* 2014. 519 p. DOI: https://doi.org/10.1109/SOSE.2014.20

15. Worldwide market share of leading cloud infrastructure service providers. URL: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/ (дата звернення 29.11.2023)

16. JavaScript frameworks technologies market share. URL: https://www.wappalyzer.com/technologies/javascript-frameworks/ (дата звернення 29.11.2023)

17. Leonard Richardson, Mike Amundsen, Sam Ruby. RESTful Web APIs: O’Reilly Media, Inc. 2013, 406 p.

18. Webhook. URL: https://uk.wikipedia.org/wiki/Webhook (дата звернення 29.11.2023)

## References

1. Michael, J. "Architecting the Cloud Design Decisions for Cloud Computing Service Models" (SaaS, PaaS, and IaaS): Wiley, 2014. 224 p. available at: https://www.wiley.com/en-us/Architecting+the+Cloud%3A+Design+Decisions+for+Cloud+Computing+Service+Models+%28SaaS%2C+PaaS%2C+and+IaaS%29-p-9781118617618 (last accessed: 27.11.2023)

2. Tomas, Erl. R., Puttini, Z. "Cloud Computing, Concepts, Technology & Architecture: Pearson", 2013. 747 p. available at: https://www.redalyc.org/pdf/6380/638067279007.pdf (last accessed: 27.11.2023)

3. "MACH Allience", available at: https://en.wikipedia.org/wiki/MACH_Alliance (last accessed: 27.11.2023)

4. "Composable Commerce", available at: https://www.elasticpath.com/composable-commerce (last accessed: 27.11.2023)

5. Hui, Song, Franck, C., Arnor, S., Bent, F., Tony, Y. (2017), "How to support customization on SaaS: A Grounded Theory from Customisation Consultants". *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C).* P. 247–249. DOI: https://doi.org/10.1109/ICSE-C.2017.136

6. Ralph, M., Andreas, M, Frank, L., Klaus, P. (2009), "Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications". *2009 ICSE Workshop on Principles of Engineering Service Oriented Systems.* 814 p. DOI: https://doi.org/10.1109/PESOS.2009.5068815

7. Wei, Sun; Xin, Zhang, Chang, Jie Guo, Pei, Sun, Huis, Su. (2008), "Software as a Service: Configuration and Customization Perspectives". *2008 IEEE Congress on Services Part II (services-2 2008).* 2946 p. DOI: https://doi.org/10.1109/SERVICES-2.2008.29

8. Franck, C., Arnor, S. (2018), "Using Intrusive Microservices to Enable Deep Customization of Multi-tenant SaaS". *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC).* 225 p. DOI: https://doi.org/10.1109/QUATIC.2018.00015

9. Hui, Song, Phu, H. Nguyen, Frank, C., Jens, G., Thomas, S. (2019), "Customizing Multi-Tenant SaaS by Microservices: A Reference Architecture". *2019 IEEE International Conference on Web Services (ICWS).* 710 p. DOI: https://doi.org/10.1109/ICWS.2019.00081

10. Espen, T., Phu, H. Nguyen, Franck, C., Hui, Song (2020), "Event-Based Customization of Multi-tenant SaaS Using Microservices". *2020 International Conference on Coordination Languages and Models.* P. 171–180. DOI: https://doi.org/10.1007/978-3-030-50029-0_11

11. "What is SaaS Architecture? 10 Best Practices For Efficient Design", available at: https://www.cloudzero.com/blog/saas-architecture/ (last accessed: 29.11.2023)

12. "SaaS application architecture best practices", available at: https://acropolium.com/blog/saas-app-architecture-2022-overview/ (last accessed: 29.11.2023)

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

13. Eng, L., Benjamin, K. (2023), "An Exploratory Study of Architectural Style and Effort Estimation for Multi-Tenant Microservices-Based Software as a Service (SaaS)". *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C).* 103 p. DOI: https://doi.org/10.1109/ICSA-C57050.2023.00043

14. Wei-Tek, T., Peide, Z. (2014), "Multi-tenancy and Sub-tenancy Architecture in Software-as-a-Service (SaaS)". *2014 IEEE 8th International Symposium on Service Oriented System Engineering.* 519 p. DOI: https://doi.org/10.1109/SOSE.2014.20

15. "Worldwide market share of leading cloud infrastructure service providers", available at: https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/ (last accessed: 29.11.2023)

16. "JavaScript frameworks technologies market share", available at: https://www.wappalyzer.com/technologies/javascript-frameworks/ (last accessed: 29.11.2023)

17. Leonard Richardson, Mike Amundsen, Sam Ruby. RESTful Web APIs: O'Reilly Media, Inc. 2013, 406 p.

18. "Webhook", available at: https://uk.wikipedia.org/wiki/Webhook (last accessed: 29.11.2023)

*Надійшла 30.11.2023*

*Відомості про авторів / About the Authors*

**Романків Назарій Дмитрович** – Харківський національний університет радіоелектроніки, студент кафедри системотехніки, Харків, Україна; e-mail: ayzrian@gmail.com; ORCID ID: https://orcid.org/0009-0004-9893-6823

**Ситніков Дмитро Едуардович** – кандидат технічних наук, доцент, Харківський національний університет радіоелектроніки, професор кафедри системотехніки, Харків, Україна; e-mail: dsytnikov@googlemail.com; ORCID ID: https://orcid.org/0000-0003-1240-7900

**Romankiv Nazarii** – Kharkiv National University of Radio Electronics, Student at the Department of System Engineering, Kharkiv, Ukraine.

**Sytnikov Dmytro** – PhD (Engineering Sciences), Associate Professor, Kharkiv National University of Radio Electronics, Professor at the Department of System Engineering, Kharkiv, Ukraine.

## ANALYSIS AND SELECTION OF METHODS FOR CUSTOMIZING *SAAS* SOLUTIONS BUILT USING *CLOUD-NATIVE* TECHNOLOGIES

**The subject of the study** is the methods of customization of *SaaS* solutions. **The purpose of the article** is to determine a holistic strategy for customizing SaaS solutions developed using *cloud-native* technologies. **Objectives**: to analyze modern approaches to the architecture of *SaaS* applications; to identify the main methods of customization for modern *SaaS* applications; to investigate and establish a method for customizing the interface of *SaaS* applications; based on the study, to determine a comprehensive strategy for customizing SaaS applications. The following **methods** are implemented: analysis and synthesis – to study the technologies that are used to build *SaaS* applications; abstraction and generalization – to determine the overall architecture of the *SaaS* application; synthesis of web technologies – to select methods of customization of *SaaS* applications and build an algorithm for selecting a customization method. **Results** achieved: the modern architecture of SaaS applications is studied; methods of customization of *SaaS* applications are selected, and a holistic strategy for customization of *SaaS* applications is defined; a method for modifying the interface of *SaaS* applications is determined, which allows customization of the web interface of a *SaaS* application regardless of the specific *front-end* framework. **Conclusions**: modern architecture of *SaaS* applications is complex and based on microservice architecture, *multi-tenant* approach, cloud technologies and web browser; methods of customization of *SaaS* applications should be developed independently of certain technology stacks to meet the customization needs of most or all users of *SaaS* applications; the *API-first* approach is fundamental in building a customized *SaaS*, as it is the basis for creating any automation or related business logic; *webhooks* are the best option for implementing subscriptions to events occurring in a *SaaS* application, regardless of the technology stack; a defined strategy for customizing *SaaS* applications allows customization regardless of a particular technology stack and, in addition to implementing business logic, also covers interface modifications.

**Keywords:** *SaaS*; customization; *web; webhooks, Iframe, API-first.*

*Бібліографічні описи / Bibliographic descriptions*

Романків Н. Д., Ситніков Д. Е. Аналіз і вибір методів кастомізації *SaaS*-рішень, побудованих за допомогою технологій *cloud-native*. *Сучасний стан наукових досліджень та технологій в промисловості*. 2023. № 4 (26). С. 68–77. DOI: https://doi.org/10.30837/ITSSI.2023.26.068

Romankiv, N., Sytnikov, D. (2023), "Analysis and selection of methods for customizing *SaaS* solutions built using *Cloud-Native* technologies", *Innovative Technologies and Scientific Solutions for Industries*, No. 4 (26), P. 68–77. DOI: https://doi.org/10.30837/ITSSI.2023.26.068