DOI: https://doi.org/10.36910/6775-2524-0560-2025-60-46
УДК 004.9:004.75:004.056.55
Бурлаченко Іван Сергійович, старший викладач
https://orcid.org/0000-0001-5088-6709
Заворотній Дмитро Олександрович, бакалавр
https://orcid.org/0009-0005-3895-8964
Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна

МІКРОСЕРВІСНА АРХІТЕКТУРА СИСТЕМИ АДАПТИВНОЇ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ ФІНАНСОВИХ УСТАНОВ

Бурлаченко І. С., Заворотній Д. О. Мікросервісна архітектура системи адаптивної автентифікації користувачів фінансових установ. У статті розглядається проблема створення мікросервісної архітектури системи адаптивної автентифікації користувачів фінансових установ як ключового напряму підвищення рівня безпеки, масштабованості та гнучкості інформаційних систем. Авторами проаналізовано сучасні методи автентифікації, зокрема однофакторні, двофакторні та багатофакторні підходи, а також підкреслено особливу значущість адаптивної автентифікації, що забезпечує динамічну зміну параметрів перевірки особи залежно від оцінки ризику операції. Досліджено методи, які успішно застосовуються у фінансовому секторі, серед яких поведінкова біометрія, контекстуальна автентифікація, аналіз аномалій та використання алгоритмів машинного навчання. Запропонована концептуальна модель системи реалізує поділ функціоналу на незалежні мікросервіси, що сприяє гнучкості, модульності та легкій інтеграції з національними й міжнародними стандартами ідентифікації (зокрема через сервіси BankID та «Дія»). Окрему увагу приділено візуалізації архітектурних залежностей, UML-діаграмам та використанню сучасних інструментів клієнтської і серверної розробки програмного забезпечення: Vue.js, Nest.js та Spring Boot. Система забезпечує багатоканальну автентифікацію (email, SMS, QR-код), підтримує балансування навантаження та механізми стійкості до відмов, що є критично важливим для фінансових установ. Результати дослідження можуть стати основою для впровадження масштабованих та надійних рішень у сфері кібербезпеки, а також послужити базою для подальших наукових робіт у галузі адаптивної автентифікації.
Ключові слова: адаптивна автентифікація, багатофакторна автентифікація, мікросервісна архітектура, OTP, фінтех.

Burlachenko I., Zavorotnii D. Microservice Architecture of an Adaptive Authentication System for Users of Financial Institutions. The article addresses the problem of developing a microservice architecture for an adaptive authentication system of financial institutions' users as a key direction for enhancing the security, scalability, and flexibility of modern information systems. The authors analyze current authentication methods, including single-factor, two-factor, and multi-factor approaches, and emphasize the particular importance of adaptive authentication, which provides dynamic adjustment of identity verification parameters depending on the assessed risk level of an operation. The study explores methods successfully applied in the financial sector, such as behavioral biometrics, contextual authentication, anomaly detection, and the use of machine learning algorithms. The proposed conceptual model implements a division of functionality into independent microservices, which facilitates flexibility, modularity, and easy integration with national and international identification standards (including BankID and *Diia* services). Special attention is given to the visualization of architectural dependencies, UML diagrams, and the use of modern client- and server-side software development tools such as Vue.js, Nest.js, and Spring Boot. The system provides multi-channel authentication (email, SMS, QR code), supports load balancing, and incorporates fault tolerance checking, which are critically significant for financial organizations. The results of the study may serve as a foundation for implementing scalable and reliable cybersecurity solutions, as well as a basis for further research in the field of adaptive authentication.
Keywords: adaptive authentication, multi-factor authentication, microservice architecture, OTP, fintech.

Постановка наукової проблеми. Проєктування мікросервісної архітектури програмної системи автентифікації фінансових установ є актуальною науковою задачею, що зумовлена зростанням вимог до безпеки, масштабованості та гнучкості сучасних інформаційних систем. У фінансовій сфері критично важливим є забезпечення надійної ідентифікації користувачів та захисту їхніх персональних даних від несанкціонованого доступу. Традиційні монолітні архітектури часто виявляються обмеженими у швидкості адаптації до нових регуляторних норм, змін бізнес-процесів і зростання навантаження. Використання мікросервісного підходу дозволяє розділити функціональність системи автентифікації на незалежні сервіси, що забезпечує гнучкість, модульність та простоту оновлення. Основним викликом є забезпечення цілісності безпекових механізмів при розподіленій обробці запитів та узгодженості даних між мікросервісами. Додаткової уваги потребує дослідження ефективних протоколів обміну даними, що мінімізують затримки та гарантують захищеність транзакцій. Науковий інтерес становить також інтеграція системи з національними та міжнародними стандартами фінансової автентифікації. Важливим завданням є оптимізація механізмів балансування навантаження та стійкість до відмов, що визначає рівень надійності системи. Таким чином, постановка задачі полягає у розробці концептуальної та технічної моделі мікросервісної архітектури автентифікаційної системи, здатної задовольнити високі вимоги

фінансових установ. Результати дослідження можуть стати основою для створення універсальних програмних рішень у сфері захищеної автентифікації та кібербезпеки фінансових установ.

Аналіз останніх досліджень і публікацій. Автентифікація — це поняття, яке позначає процес перевірки та підтвердження достовірності певного об'єкта, факту чи документа. У більшості випадків воно використовується для встановлення та підтвердження особистості користувача під час взаємодії з інформаційними системами чи сервісами [1] . Сутність автентифікації полягає у зіставленні наданих користувачем даних (наприклад, пароля, біометричних характеристик або цифрового сертифіката) із заздалегідь збереженою еталонною інформацією. Метою цього процесу є гарантування того, що доступ до системи отримує саме та особа чи суб'єкт, який має на це право. Автентифікація є базовим елементом сучасної кібербезпеки та лежить в основі захисту даних, фінансових операцій і комунікацій. У різних сферах застосовуються різні методи автентифікації від простих однофакторних до багатофакторних рішень, що поєднують незалежні перевірки [2]. У фінансовій галузі вона має особливе значення, оскільки від рівня надійності цього процесу залежить запобігання шахрайству та збереження конфіденційності клієнтів. Таким чином, автентифікація виступає ключовим інструментом довіри у цифровому середовищі, де підтвердження ідентичності є критично важливим для безпеки та стабільності роботи систем. Найчастіше клієнти фінансових установ вирішують підтверджувати власну ідентичність шляхом введення облікових даних, що є узгодженим набором інформації, яка передається між клієнтом та системою автентифікації [3].

Сучасні системи інформаційної безпеки виокремлюють декілька основних видів автентифікації, класифікація яких базується на використовуваних факторах, а саме: однофакторна, двофакторна, багатофакторна. Однофакторна автентифікація (SFA) передбачає верифікацію користувача на основі єдиного параметра, найчастіше представленого парольним захистом [4]. Даний метод характеризується відносною простотою імплементації, проте демонструє вразливість до низки атак, включаючи методи соціальної інженерії та вичерпний пошук [5].

Двофакторна автентифікація (2FA) поєднує два незалежні фактори верифікації, що суттєво підвищує рівень захисту системи. Типова реалізація включає комбінацію пароля та одноразового коду, генерованого апаратним токеном або програмним застосунком. Багатофакторна (мультифакторна, MFA) автентифікація розширює двофакторний підхід, залучаючи три або більше факторів, що належать до різних категорій. Це максимізує ентропію системи автентифікації та мінімізує ймовірність несанкціонованого доступу [6].

Питання безпеки доступу до даних та сервісів є критично важливим у фінансовому секторі. У відповідь на існуючі кіберзагрози з'явилась концепція адаптивної автентифікації, яка забезпечує динамічну зміну рівня перевірки особи користувача залежно від різних факторів ризику. Адаптивна автентифікація – це динамічний процес перевірки особи користувача, що змінює свої параметри та вимоги залежно від оцінювання рівня ризику конкретної операції. Система адаптивної автентифікації аналізує різноманітні фактори, включаючи поведінкові патерни користувача, геолокацію, час доступу, тип пристрою та характер запитуваної операції, для визначення необхідного рівня захисту [7]. Концептуальна відмінність адаптивної автентифікації від традиційних методів полягає у переході від статичного до динамічного підходу при визначенні вимог до перевірки особи. Замість застосування однакових процедур для всіх користувачів і операцій, адаптивна система індивідуалізує процес автентифікації на основі контекстуальних даних та аналізу ризиків [8]. Основними методами адаптивної автентифікації, які наразі успішно використовуються у фінансовому секторі, є: поведінкова біометрія, контекстуальна автентифікація, біометрична автентифікація, аналіз аномалій та машинне навчання (ML), автентифікація на основі знань (KBA).

Виділення невирішених раніше частин загальної проблеми. Враховуючи наявність великої кількості адаптивних методів автентифікації слід зазначити, що вони мають вузькоспеціалізоване застосування. Умови в яких використовуються альтернативні рішення в більшості випадків є закритими та не повністю розкривають критично важливі практичні аспекти реалізації автентифікації клієнтів фінансових установ. З цієї причини новітні розроблювані рішення не мають достатньо повної функціональності для забезпечення гнучкості автентифікаційних процесів. Щоб вирішити вищезазначені аспекти проблеми у процесі дослідження необхідно розв'язати наступні завдання:

1. проаналізувати існуючі адаптивні методи автентифікації користувачів у фінансових установах та визначити їх переваги і недоліки;

2. реалізувати програмну частину вебсистеми, включаючи: клієнтську частину з використанням Vue.js та Socket.IO та серверну частину з використанням мікросервісів на базі технологій Nest.js та Spring Boot;

3. спроєктувати структуру бази даних для вебсистеми адаптивної автентифікації з використанням PostgreSQL;

4. спроєктувати архітектуру вебсистеми адаптивної автентифікації відповідно до актуальних стандартів інформаційної безпеки.

Мета дослідження полягає у розроблені концептуальної та технічної моделі мікросервісної архітектури автентифікаційної системи, здатної задовольнити високі вимоги продуктивності фінансових установ, для пришвидшення надійного процесу автентифікації за рахунок адаптивності системи до умов варіантів використання при вірогідних сценаріях кібератак.

Виклад основного матеріалу дослідження. Інструменти для візуального графу залежностей модулів допомагають швидко представити структуру проєкту та взаємозв'язки між його частинами. Це особливо корисно на етапах аналізу та проєктування, де важливо виявити надмірні зв'язки чи циклічні залежності. Подібні інструменти спрощують onboarding нових розробників, які можуть швидше зорієнтуватися у складній архітектурі. Граф, який представлений на рис. 1, полегшує рефакторинг, даючи змогу виявити «вузькі місця» чи надмірно зв'язані компоненти. Інструменти візуалізації залежностей модулів також корисні для виявлення прихованих залежностей, які можуть призводити до помилок під час змін у коді. Візуалізація залежностей підтримує процес документування ПЗ і полегшує комунікацію між командами. Одним з недоліків є перевантаження інформацією у великих проєктах, де граф може стати нечітким і важким для аналізу. Автоматично згенеровані графи іноді можуть показувати лише поверхневі зв'язки, не враховуючи динамічну поведінку модулів. Інтеграція з CI/CD або IDE може вимагати додаткових налаштувань, що збільшує час впровадження. Також існує ризик надмірної довіри до візуалізації, що може відволікати від аналізу реального коду й логіки програми.

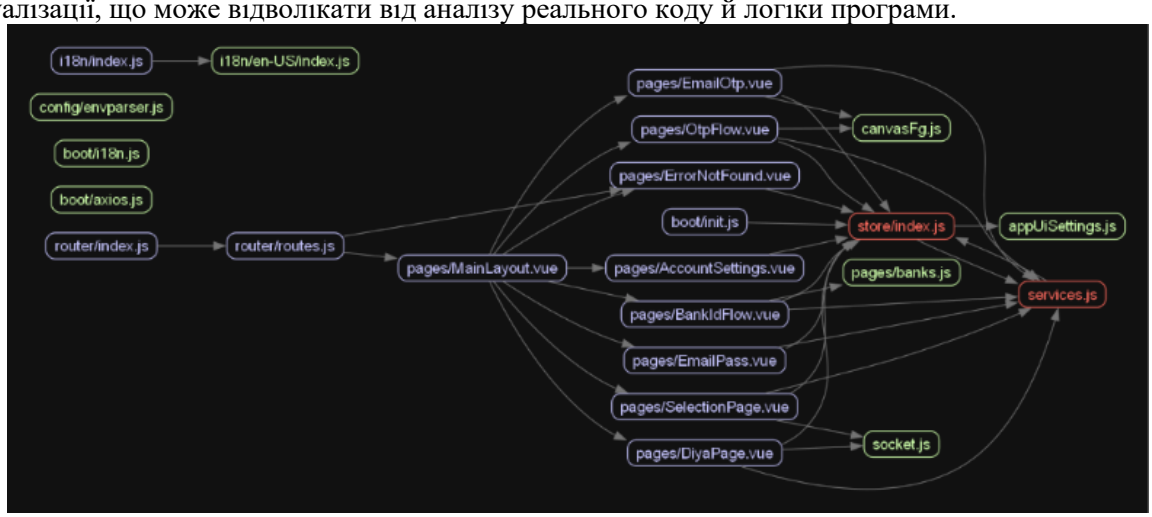

Рис.1. Діаграма зв'язків компонентів модуля OTP widget вебсистеми адаптивної автентифікації

Автоматично згенеровані графи іноді можуть показувати лише поверхневі зв'язки, не враховуючи динамічну поведінку модулів. Інтеграція з CI/CD або IDE може вимагати додаткових налаштувань, що збільшує час впровадження. Також існує ризик надмірної довіри до візуалізації, що може відволікати від аналізу реального коду й логіки програми.

Компонент EmailOtp (рис. 2) відповідає за двофакторну автентифікацію користувача за допомогою email і одноразового коду. На першому етапі користувач вводить свою email-адресу, яка перевіряється та надсилається на сервер для генерації OTP.

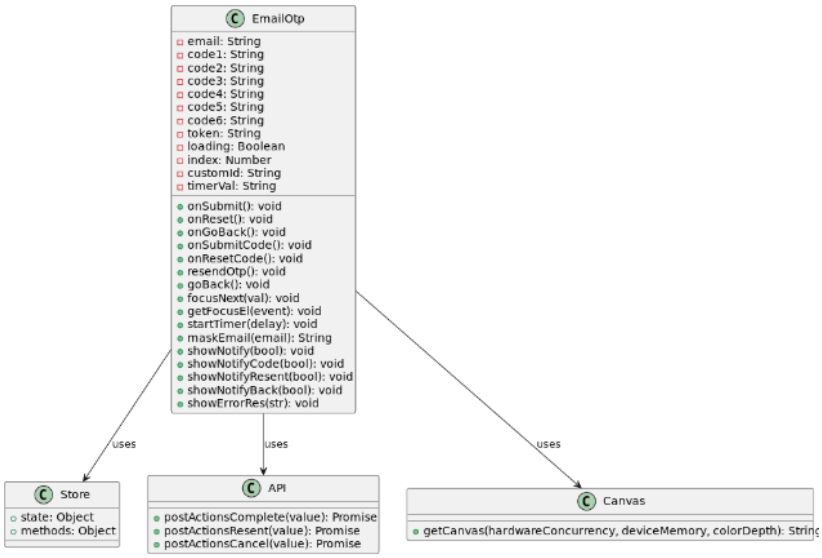

Рис.2. Діаграма класів компонента EmailOtp.vue

Після успішної перевірки email, компонент переходить до другої сторінки, де відображається форма введення 6-значного коду підтвердження. Кожне поле вводу коду пов'язане з відповідною змінною (code1-code6), а при введенні фокусується наступне поле автоматично. Якщо код введено неправильно, користувачу виводиться повідомлення про помилку, і він може повторити спробу. За потреби користувач може надіслати код повторно, але лише після завершення таймера, який відображає час до повторної відправки. Кнопка «Назад» дозволяє скасувати процес і повернутися до попереднього кроку, очищаючи всі поля. Повідомлення про помилки або успішні дії відображаються через сповіщення, що викликаються методом notify з бібліотеки Quasar.

Компонент BankId (рис. 3) містить логіку автентифікації користувача через BankID. Клас управляє внутрішнім станом, таким як завантаження, токен, номер телефону та коди OTP.

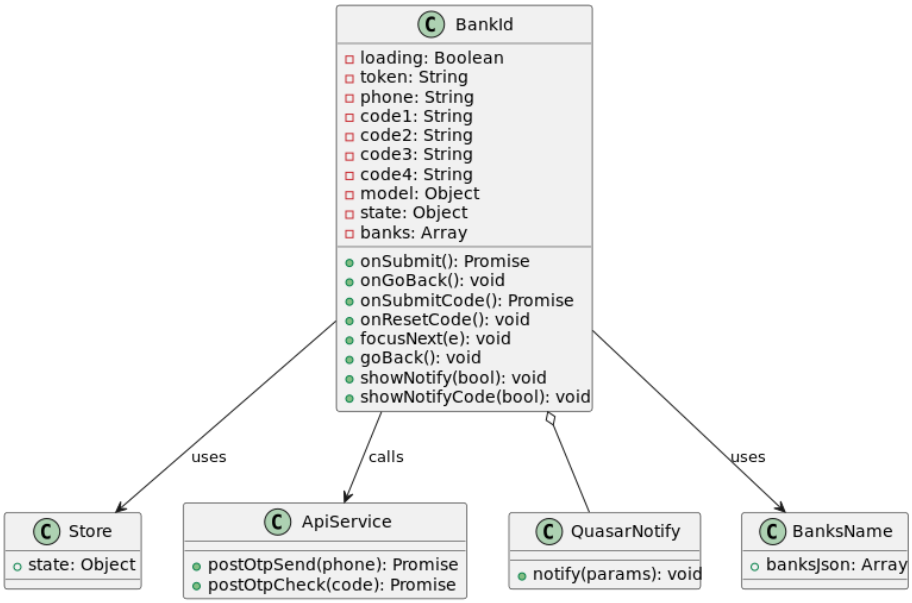

Рис.3. Діаграма класів компонента BankId.vue

Даний клас містить методи для відправки номера телефону (onSubmit), перевірки коду (onSubmitCode), обробки повернення назад (onGoBack, goBack), фокусування між інпутами (focusNext) та скидання коду (onResetCode). Компонент взаємодіє з класом ApiService, який відповідає за HTTP-запити до серверної частини для надсилання та перевірки OTP. Клас QuasarNotify використовується для виводу повідомлень користувачеві про помилки чи успіх операцій. Також використовується клас Store, що містить глобальний стан застосунку, зокрема такі властивості як state.returnUrl чи state.urlFromCiam. Клас BanksName надає список банків, які використовуються як опції для q-select. Усі ці класи спільно забезпечують повний цикл авторизації

користувача через BankID. Вони також сприяють розділенню обов'язків: інтерфейс, логіка, стан і дані чітко відокремлені.

Компонент OptFlow (рис. 4) відповідає за двоетапну автентифікацію користувача через телефон. Спочатку він відображає форму введення номера телефону, і при його успішному введенні надсилає запит до API для ініціалізації сесії. Якщо запит проходить успішно, на інтерфейсі з'являється форма введення одноразового коду (OTP).

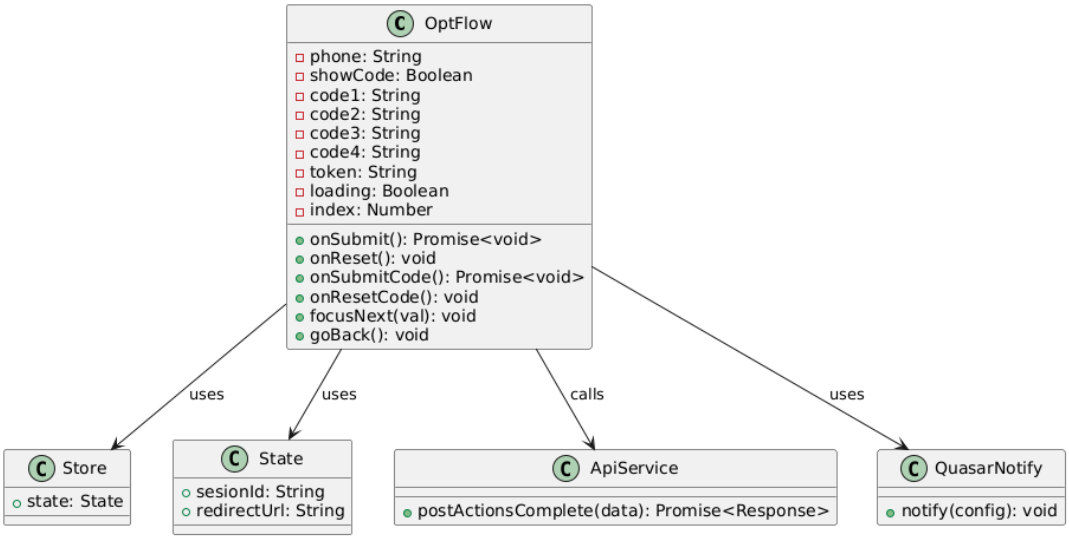

Рис. 4. Діаграма класів компонента OptFlow.vue

Введений код перевіряється шляхом повторного запиту до API з цим кодом та sessionId. Компонент використовує локальні змінні (phone, code1–4, loading, showCode) для управління станом. Для UX покращення застосовуються Quasar-компоненти q-input, q-btn, q-toggle і q-form. Повідомлення про помилки або успішні дії відображаються через $q.notify. Користувач може повернутися назад з форми OTP, натиснувши кнопку «Назад». Також реалізована логіка автоматичного переходу між полями коду. Загалом компонент організовує безпечний вхід користувача за допомогою OTP.

Компонент DiyaPage (рис. 5) Vue-компонент DiyaPage відповідає за генерацію та обробку QR-коду для авторизації через застосунок Дія. Він ініціалізує сокет-з'єднання для обробки callback-ів від сервера після сканування QR-коду користувачем. При монтуванні компонента запускається таймер зворотного відліку, який оновлює інтерфейс до завершення сесії. Якщо користувач натискає кнопку «Оновити QR-код», попереднє сокет-з'єднання розривається, генерується новий deepLink, і створюється нове сокет-з'єднання.

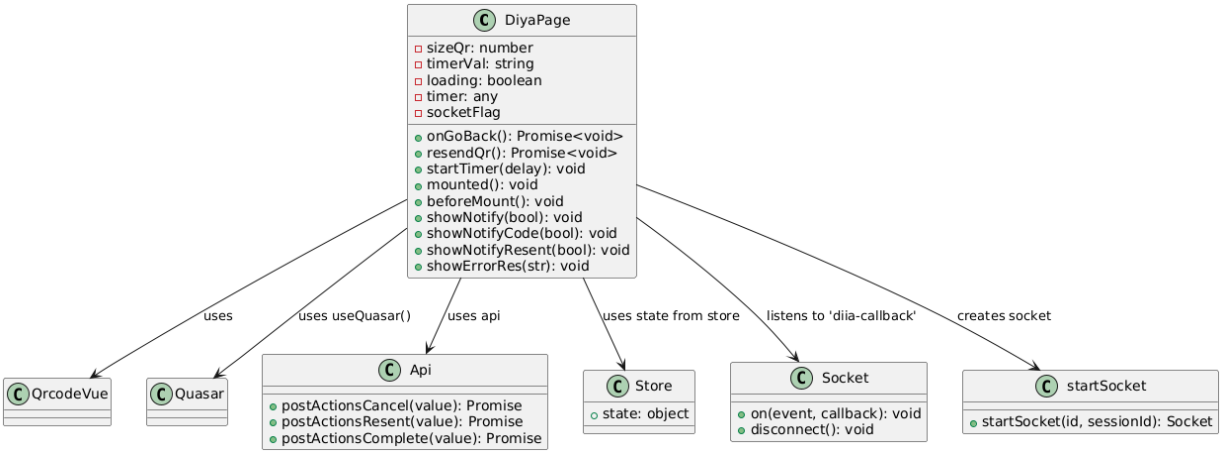

Рис.5. Діаграма класів компонента DiyaPage.vue

Компонент підтримує адаптивність: для десктопних і мобільних пристроїв відображається відповідний варіант QR-коду або посилання. У методі onGoBack реалізована логіка скасування підпроцесу авторизації та повернення користувача назад. При отриманні повідомлення diia-callback з позитивним статусом, відбувається завершення кроку авторизації через API і перенаправлення користувача. Якщо авторизація неуспішна, виводяться відповідні повідомлення про помилки. Для

інформування користувача компонент використовує систему сповіщень Quasar. Додатково в компоненті є кілька методів для виведення сповіщень про статус дій, таких як успішна відправка або помилка.

Після проєктування Front-End частини переходимо до проєктування серверної частини. Необхідно визначити основні абстракції під час використання архітектурних патернів у різних Back-End фреймворках (рис. 6).

Клас AuthService містить методи, що реалізують основну бізнес-логіку аутентифікації та авторизації користувачів. Сюди входять методи для побудови параметрів запиту (buildParams), отримання параметрів аутентифікації (getAuthParams, getAuthParamsByCustomerId), перевірки валідності масивів (isValidArray), а також різні сценарії входу або через примусову аутентифікацію (authByForce), refresh token (authByRefreshToken) або через стандартний логін (login). Додатково сервіс дозволяє виконувати авторизацію (authorization), генерувати кроки автентифікації (generateSteps), перевіряти стан токена (isAlive), виходити з системи (logout), оновлювати токени (refresh), відкликати їх (revoke), підписувати дані (signData), отримувати токени (token), атрибути користувача (userAttributes), інформацію про користувача (userinfo) та валідувати токени (validate).

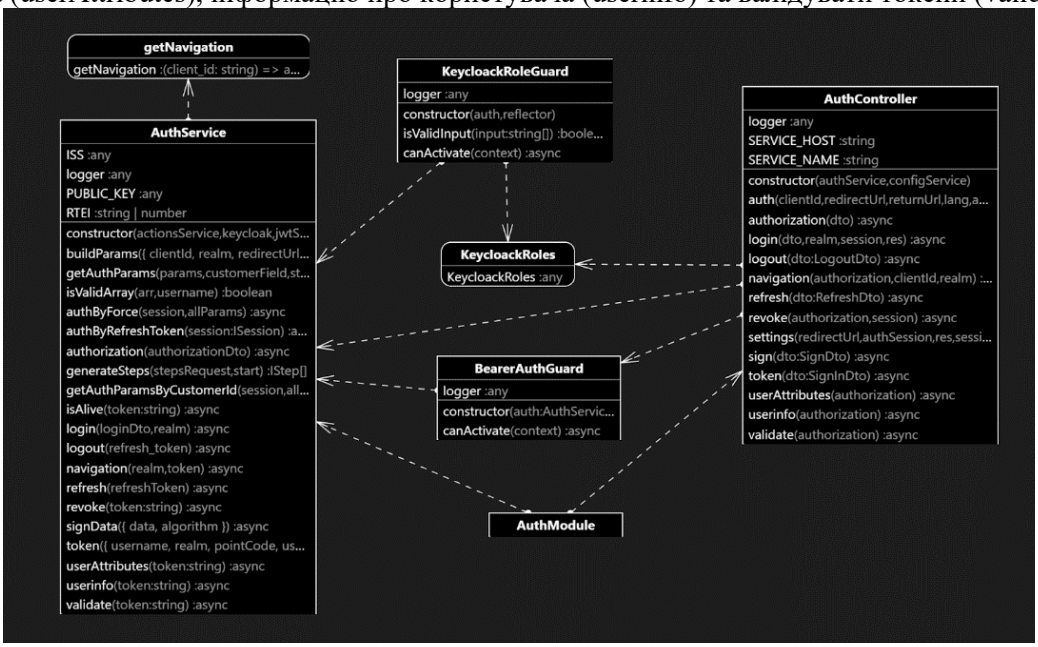

Рис.6 Діаграма класів AuthModule вебсистеми адаптивної автентифікації

Клас AuthController відповідає за обробку HTTP-запитів, пов'язаних із аутентифікацією та авторизацією. Він надає методи для ініціалізації аутентифікації (auth), логіну (login), виходу з системи (logout), отримання навігаційних даних (navigation), оновлення токенів (refresh), відкликання сесій чи токенів (revoke), отримання налаштувань (settings), підпису даних (sign), отримання токенів (token), атрибутів користувача (userAttributes), інформації про користувача (userinfo) і валідації авторизації (validate). Гарди KeycloakRoleGuard і BearerAuthGuard забезпечують захист маршрутів відповідно до ролей користувача та наявності валідного токена. KeycloakRoleGuard містить методи для перевірки валідності вхідних ролей (isValidInput) і визначення, чи може користувач отримати доступ до ресурсу (canActivate), використовуючи інформацію про ролі (KeycloakRoles). BearerAuthGuard перевіряє наявність і дійсність токена за допомогою методу canActivate, використовуючи функціонал AuthService для авторизації запитів.

Сервіс getNavigation надає допоміжний метод getNavigation, який повертає навігаційні дані для певного клієнта за його ідентифікатором. Всі ці компоненти об'єднуються в модулі AuthModule, що дозволяє централізовано керувати залежностями, маршрутами та логікою аутентифікації у всій системі. Нижче наведено діаграму класів, що демонструє модуль дій (ActionsModule), який координує взаємодію між різними сервісами, відповідальними за обробку дій користувача, аутентифікацію та OTP (одноразові паролі) (рис. 7).

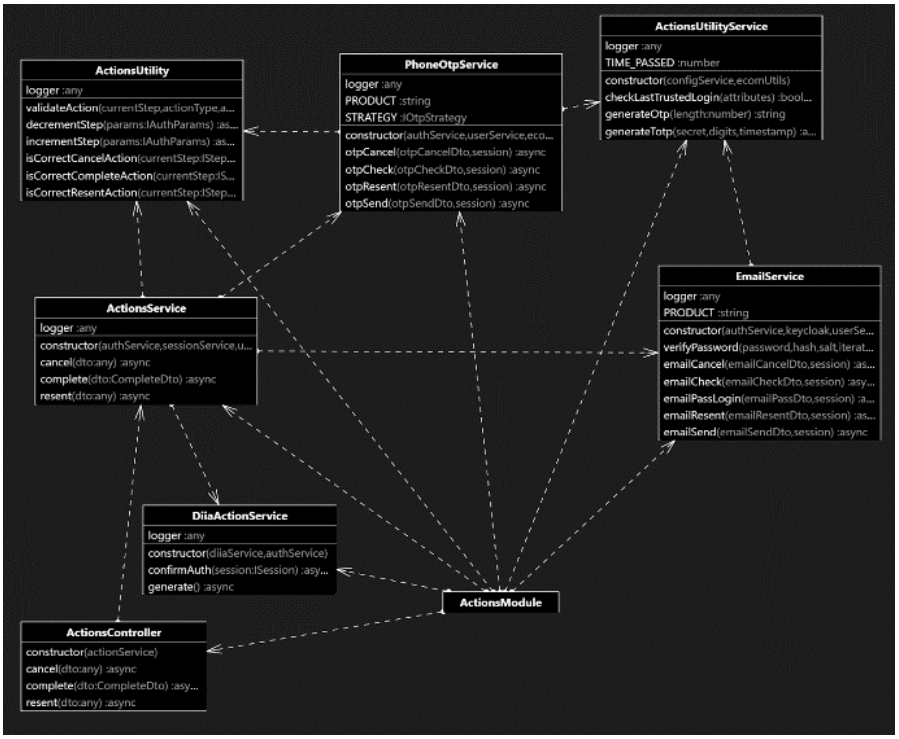

Рис.7 Діаграма класів ActionsModule вебсистеми адаптивної автентифікації

Клас ActionsController виступає як точка входу для зовнішніх запитів і викликає відповідні методи сервісу ActionsService, що реалізує бізнес-логіку для скасування (cancel), завершення (complete) та повторної відправки дії (resent). ActionsService взаємодіє з іншими службами, як-от PhoneOtpService, EmailService, DiiaActionService для виконання конкретних операцій, таких як відправка OTP по телефону або електронній пошті. Клас ActionsUtility використовується для перевірки правильності кроків виконання дій, зокрема, чи можна скасувати, завершити або повторити дію на поточному етапі.

Клас PhoneOtpService реалізує стратегію для OTP по телефону (згідно з інтерфейсом IOtpStrategy) і надає методи для надсилання, перевірки, повторного надсилання або скасування OTP. EmailService, зі свого боку, виконує аналогічну роль, але для обробки OTP через email дозволяє перевіряти паролі, надсилати або скасовувати OTP, перевіряти введені коди тощо. Клас DiiaActionService має справу з інтеграцією з державним застосунком «Дія», а саме реалізує підтвердження дії та генерацію відповідних даних. Клас ActionsUtilityService надає допоміжні функції, зокрема, генерацію OTP-кодів та перевірку останніх надійних входів користувача. Усі сервіси використовують Logger для журналювання операцій, що полегшує відстеження та діагностику.

Клас ActionsModule слугує як модуль для зв'язку між цими компонентами в межах одного функціонального блоку. Він імпортує всі потрібні сервіси, включно з ActionsUtility, ActionsUtilityService, PhoneOtpService, EmailService, DiiaActionService, та об'єднує їх для централізованої роботи. Основна логіка полягає в делегуванні задач відповідним сервісам згідно з типом дії та каналом комунікації (телефон, email або застосунок «Дія»). Наприклад, при спробі користувача скасувати OTP, ActionsService перевіряє правильність поточного кроку через ActionsUtility, після чого викликає PhoneOtpService.otpCancel або EmailService.emailCancel, залежно від каналу. Ця структура модульна, розширювана і підтримує принципи розділення відповідальності, що забезпечує легкість підтримки та масштабування системи. Хоча взаємозв'язки численні, вони чітко поділені за функціональним призначенням, що дозволяє уникати дублювання логіки та спрощує тестування.

Діаграма класів описує модуль CommunicationModule, який забезпечує відправку повідомлень і електронних листів через зовнішні сервіси Codot та Universalna (рис. 8).

Центральним елементом є клас CommunicationService, який інкапсулює логіку взаємодії з цими зовнішніми системами через методи sendEmail та sendMessage. Сервіс має конфігураційні властивості EMAIL_CONFIG і SMS_CONFIG, що визначають, через який провайдер буде виконуватись надсилання. Метод isNotEmptyObject використовується для перевірки валідності

вхідних об'єктів перед передачею у відповідні сервіси. Журналювання та властивість PRODUCT дозволяють відстежувати процеси та налаштовувати поведінку в залежності від контексту продукту.

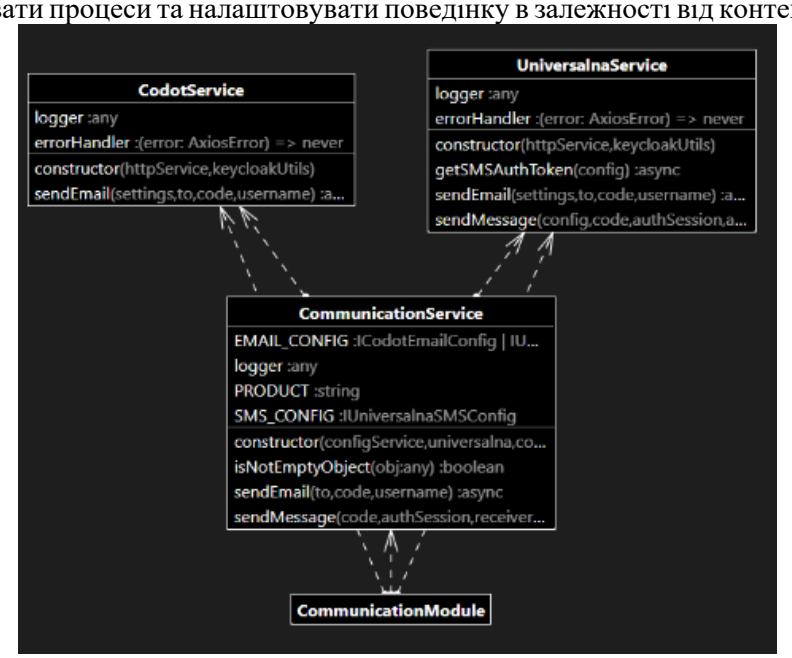

Рис.8 Діаграма класів CommunicationModule вебсистеми адаптивної автентифікації

Клас CodotService відповідає за надсилання електронної пошти через сервіс Codot і містить метод sendEmail, який приймає налаштування, адресу одержувача, код і ім'я користувача. Він також має обробник помилок errorHandler, що реагує на помилки типу AxiosError – типові для HTTP-запитів. UniversalnaService, в свою чергу, надає ширший спектр функціональності: крім sendEmail, він реалізує метод sendMessage, що дозволяє надсилати SMS, а також getSMSAuthToken, який використовується для автентифікації перед відправкою SMS. Обидва сервіси підключаються до CommunicationService, який делегує їм реальні дії в залежності від конфігурації. Це дозволяє централізовано управляти всіма повідомленнями через один сервіс, не турбуючись про особливості конкретних провайдерів.

Уся архітектура об'єднується у модулі CommunicationModule, який слугує точкою з'єднання між залежностями та основним сервісом. Завдяки цьому модулю CommunicationService отримує необхідні сервіси (CodotService, UniversalnaService) та конфігурацію для коректного функціонування. Логіка організована так, щоб можна забезпечити максимальну гнучкість та легко змінювати провайдерів або додавати нові без суттєвих змін у коді CommunicationService. Це також дозволяє реалізувати fallback-механізми, коли при відмові одного сервісу, інший може його замінити. Подібна структура відповідає принципам інверсії залежностей і відкритості/закритості (SOLID), що робить систему масштабованою та зручною в обслуговуванні. Таким чином, система чітко поділена на шари відповідальності з мінімальним зв'язуванням, що спрощує її розширення та підтримку.

Основним класом який реалізує базові методи для моніторингу стану системи є HealthController, зокрема, check() та testEndpoint(). Клас містить дві важливі константи IDENTITY_SERVICE_PORT та SERVICE_NAME, що дозволяють ідентифікувати сервіс серед інших мікросервісів у системі. Конструктор приймає кілька залежностей, пов'язаних із моніторингом: наприклад, health, http, memory, disk, тощо. Дані сервіси дозволяють виконати комплексну оцінку стану та перевіряється доступність HTTP-з'єднань, використання пам'яті та стан файлової системи.

Метод check() викликає всі доступні перевірки для визначення, чи система здорова (healthy), і повертає агрегований результат. Цей метод може використовуватись сторонніми сервісами або інструментами моніторингу (наприклад, Prometheus або Kubernetes liveness probes). Метод testEndpoint() виконує простішу перевірку: повертає статус 200 OK для підтвердження, що сервіс працює і приймає запити. Це дозволяє розділити повну перевірку і базову, а саме: одна для внутрішньої діагностики, а інша для зовнішніх health-чеків. Такий підхід є стандартом у сучасних мікросервісних архітектурах.Модуль HealthModule інкапсулює логіку контролера HealthController, забезпечуючи його залежностями та дозволяючи інтеграцію з іншими модулями системи. Його

легко імпортувати у будь-який інший модуль, щоб додати базову діагностику сервісу. Подібна реалізація дозволяє централізовано слідкувати за здоров'ям усієї системи. Вона також зменшує ризик непомічених збоїв, оскільки дозволяє виявляти проблеми на рівні ресурсів ще до критичного падіння. Крім того, вона корисна для DevOps-команд під час CI/CD-процесів. Таким чином, HealthModule є фундаментальним елементом надійності та стабільності системи.

На діаграмі класів зображено взаємодію між модулями та класами в системі, яка реалізує електронну ідентифікацію через сервіси «Дія» та BankID (рис. 9). Клас DiiaController відповідає за прийом даних від клієнтів і виклик відповідних сервісних методів. Метод documentsReceiver(dto: IDiiaCallback) приймає DTO (Data Transfer Object), який містить дані, отримані після авторизації користувача через сервіс Дія. У цьому методі відбувається перевірка підпису, розшифрування даних та їх обробка або збереження. Конструктор класу ініціалізує необхідні сервіси, зокрема, sessionService і widgetService, що використовуються для збереження сесій або показу інтерактивних елементів. Залежність від DiiaService дозволяє контролеру викликати функції глибокого посилання або верифікації. Змінна logger використовується для логування подій і помилок, а ENVIRONMENT для визначення конфігурацій (наприклад, продакшн чи тестове середовище). Контролер виступає точкою входу для зовнішніх запитів у модулі DiiaModule, що входить до загального IdpModule (модуля ідентифікації). Таким чином, DiiaController концентрує логіку маршрутизації запитів і делегування відповідним сервісам.

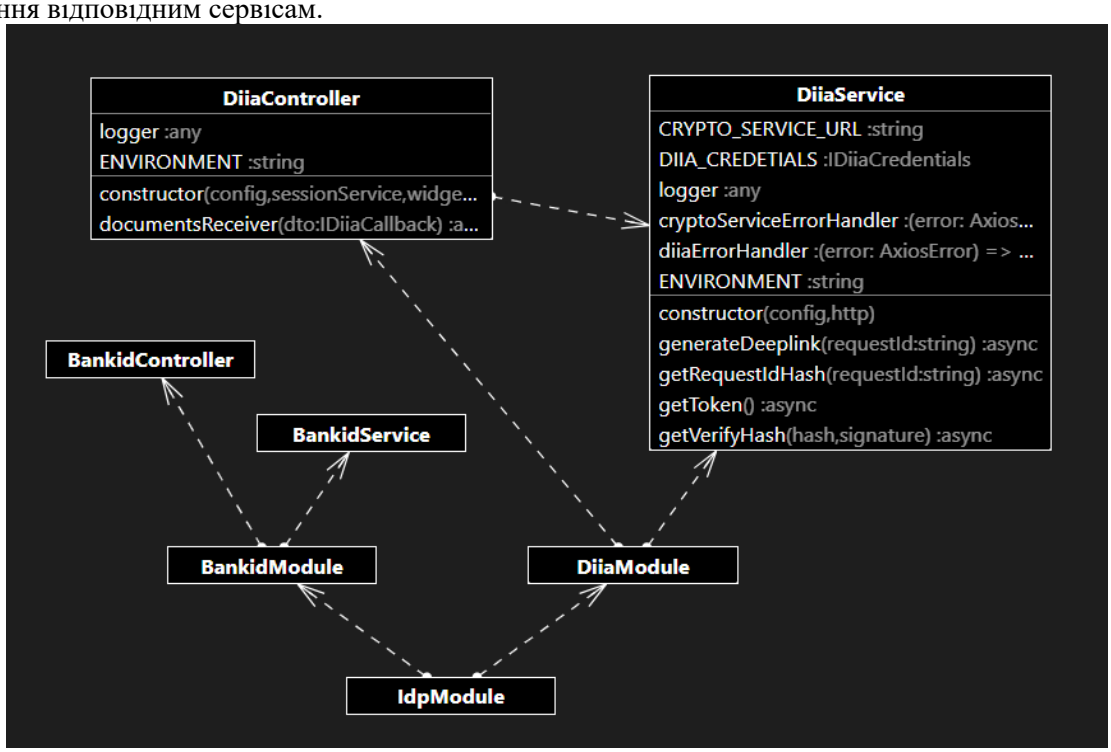

Рис.9 Діаграма класів IdpModule вебсистеми адаптивної автентифікації

Клас DiiaService є ядром обробки логіки, пов'язаної із сервісом «Дія». Метод generateDeeplink створює URL-посилання, яке клієнт може використовувати для запуску ідентифікації в застосунку «Дія», передаючи унікальний requestId. Метод getRequestIdHash створює хеш для запиту, що забезпечує перевірку автентичності клієнтського запиту. Метод getToken повертає токен для доступу до захищених даних або API, а getVerifyHash перевіряє хеш-підпис, забезпечуючи криптографічну перевірку достовірності даних. Усі методи є асинхронними, тобто підтримують неблокуючі виклики до зовнішніх API, зокрема HTTP-запити до «Дії». CryptoServiceErrorHandler та diiaErrorHandler – це функції обробки помилок, які гарантують, що система адекватно реагує на збої у зв'язку або невірні відповіді. Конструктор ініціалізує конфігурацію та HTTP-клієнт, який взаємодіє з зовнішніми сервісами. Змінні CRYPTO_SERVICE_URL і DIIA_CREDENTIALS містять необхідну інформацію для авторизації та взаємодії з API. Цей сервіс є ключовим компонентом модуля DiiaModule, який у свою чергу забезпечує сервісну логіку для контролера та інших частин системи.

На відміну від модуля DiiaModule, BankidModule реалізує логіку для автентифікації через BankID. BankidController і BankidService працюють подібно до компонентів Diia, але для іншого

провайдера ідентифікації. Обидва модулі (Diia і Bankid) інкапсулюються всередині IdpModule, який є головним інтерфейсом для автентифікації користувачів у системі. Це дозволяє системі бути універсальною і гнучкою, тобто вона може обробляти запити як від користувачів із застосунком «Дія», так і від тих, хто використовує BankID. Взаємодія між модулями реалізована через залежності, які зображено пунктирними стрілками. Загалом, така архітектура забезпечує масштабованість, безпеку та розширюваність системи, дозволяючи при необхідності додати нові провайдери ідентифікації. Вона дотримується принципів розділення обов'язків: контролери обробляють запити, сервіси – бізнес-логіку, а модулі – структурну організацію. Завдяки цьому код залишається зручним для читання, легко тестується і підтримується.

На діаграмі класів модуля WidgetModule показана організація логіки взаємодії з віджетами: невеликими інтерактивними компонентами частиною інтерфейсу або сесій користувача (рис. 10). Клас WidgetService є центральною ланкою бізнес-логіки, що координує запуск віджета (start(sessionId: string)) і підтвердження сесії (confirm(session, authSession, ott)). Конструктор цього сервісу приймає кілька залежностей, зокрема сервіси конфігурацій і сесій, що вказує на широке використання в інфраструктурі. Його асинхронні методи обробляють взаємодії в рамках сесії, використовуючи авторизаційні токени або тимчасові коди OTT. Цей сервіс також використовує метод getLayout, що отримує конфігурацію або зовнішній вигляд віджета для певного клієнта. Таким чином, WidgetService виконує роль посередника між сесією, зовнішнім виглядом і підтвердженням ідентичності. Його дизайн відображає гнучкість і підтримку багатьох потоків взаємодії. Це дозволяє системі адаптуватися до різних сценаріїв, таких як мобільні інтерфейси або інтеграція із зовнішніми клієнтами.

WidgetController – це компонент, який відповідає за виклик публічних точок доступу для ініціалізації та підтвердження віджетів. Він викликає відповідні методи WidgetService у відповідь на зовнішні запити, зокрема start(sessionId) для запуску віджета та confirm(authSession, ott, session) для перевірки автентичності сесії. Контролер ізолює HTTP або WebSocket запити від логіки обробки даних, що дозволяє чітко розмежувати рівні архітектури. Його єдиною залежністю є WidgetService, що робить його простим для тестування та розширення. Таке розмежування дозволяє централізовано керувати потоком автентифікації користувача, починаючи з отримання OTT до повного запуску віджета. Таким чином, контролер обслуговує зовнішній API або маршрути системи. Його робота фокусується на отриманні та перевірці запитів, делегуючи бізнес-логіку в сервіс. Це відповідає принципам REST/GraphQL або Socket API, де контролер лише обслуговує запити й відповіді. В результаті цей клас забезпечує чисту й логічну взаємодію з зовнішнім середовищем.

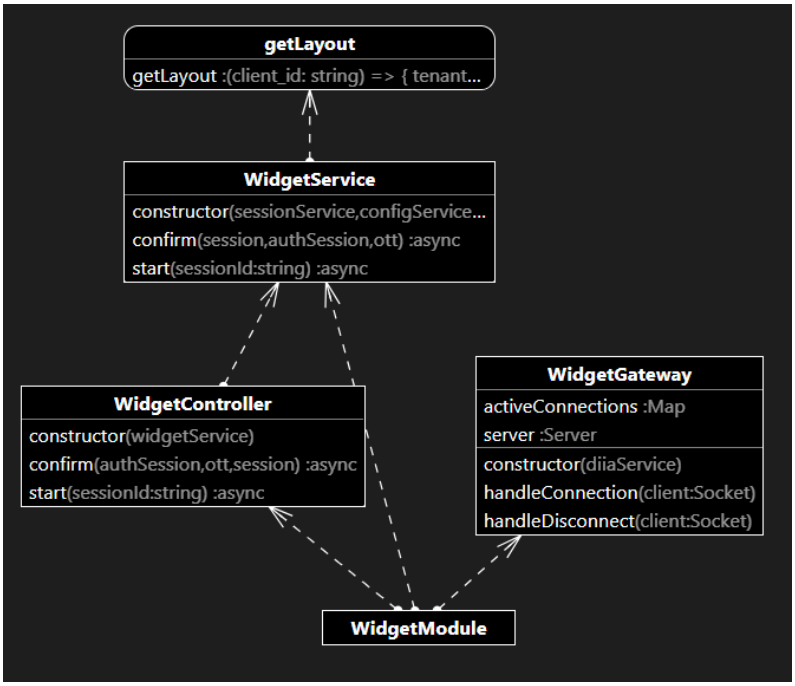

Рис.10 Діаграма класів WidgetModule вебсистеми адаптивної автентифікації

WidgetGateway відповідає за обробку WebSocket-з'єднань, тобто двосторонньої комунікації між клієнтом і сервером. Він керує підключенням (handleConnection(client: Socket)) і відключенням клієнтів (`handleDisconnect(client: Socket)), а також має змінну activeConnections, яка зберігає поточні з'єднання. Це особливо важливо для динамічних віджетів, які повинні реагувати на дії користувача в реальному часі. Крім того, конструктор класу приймає залежність diiaService, що вказує на можливу інтеграцію з платформою державних послуг «Дія», наприклад, для автентифікації або передачі документів. Gateway дозволяє легко розширювати взаємодію через події, без потреби в ручному контролі сесій. Архітектура враховує живу взаємодію через сокети й асоційовану логіку керування підключеннями. Таким чином, WidgetGateway – це транспортний рівень комунікації в реальному часі. Інтеграція з іншими компонентами модуля гарантує, що дані користувача залишаються синхронізованими й захищеними. Це критично для сервісів, що потребують миттєвого зворотного зв'язку або взаємодії, таких як електронні кабінети або віджети цифрової ідентифікації.

Висновки та перспективи подальших досліджень. Проєктування є першим етапом процесу розробки програмних проєктів. За допомогою графу залежностей, діаграм класів та діаграм послідовностей. UML проєктування надає значні переваги при розробці модулів, таких як AuthModule, ActionsModule, CommunicationModule, HealthModule, IdpModule, KeycloakModule, UserModule та WidgetModule. Завдяки діаграмам класів, компонентів та взаємодій розробники можуть наочно бачити структуру та зв'язки між модулями. Це полегшує планування інтеграцій, наприклад, як AuthModule взаємодіє з KeycloakModule або IdpModule. UML допомагає визначити ймовірні критичні аспекти на ранньому етапі, зменшуючи ризики помилок у логіці або інтерфейсах. Завдяки стандартній нотації UML покращується комунікація між командами, особливо коли в проєкті беруть участь аналітики, тестувальники та розробники. Документування за допомогою UML полегшує супровід і масштабування системи. Діаграми станів та послідовностей корисні для моделювання поведінки таких модулів, як UserModule та CommunicationModule. UML також сприяє кращому розумінню бізнес-логіки, наприклад, у ActionsModule чи WidgetModule. У модулі HealthModule UML може показати критичні точки моніторингу та зв'язки з іншими сервісами. Загалом, UML-аналіз допомагає створити узгоджену, масштабовану й підтримувану архітектуру.

Для покращення надійності та масштабованості CI-конфігурації доцільно внести кілька оптимізацій. По-перше, варто додати окрему стадію test перед build, у якій будуть запускатися юніт-тести або лінтери, що дозволить виявити помилки до створення Docker-образу. По-друге, замість ручного видалення контейнерів через docker ps/grep, краще використовувати docker container prune або мітки з автоматичним очищенням, щоб уникнути конфліктів при одночасному запуску кількох CI робіт. По-третє, зберігати створені Docker-образи в Docker Registry (наприклад, GitLab Container Registry), що дозволить використовувати їх повторно без повторної побудови. Для цього потрібно додати крок docker push після docker build, а в CI роботах – налаштувати логін до реєстру через змінні CI (CI_REGISTRY, CI_REGISTRY_USER). Також варто розділити змінні середовища у .gitlab-ci.yml на глобальні та середовищі (environment: develop, environment: prod) для зручності підтримки. Замість копіювання файлів сертифікатів у кожній CI робіт, їх можна зберігати у GitLab CI Variables як Base64 і розпаковувати всередині контейнера, що підвищить безпеку. Бажано також винести повторювану логіку побудови контейнера в окремий шаблон YAML (include), щоб уникнути дублювання між main і develop. Також можна використовувати артефакти або кешування, щоб зберігати результати проміжних кроків. Для зручності відлагодження доцільно логувати версію образу (наприклад, git commit hash або timestamp) під час побудови. Нарешті, додавання умов when: manual або allow_failure: true до нестабільних CI робіт допомагає зменшити кількість зупинених конвеєрних процесів через незначні помилки.

Список бібліографічного опису:

1. Bumiller A., Challita S., et al. On Understanding Context Modelling for Adaptive Authentication Systems. *ACM Transactions on Autonomous and Adaptive Systems*. – 2023. – Vol. 18, No. 1. – P. 3:1–3:35. – DOI: https://doi.org/10.1145/3582696.
2. Bello H. O., Ige A. B., Ameyaw M. N. Adaptive machine learning models: Concepts for real-time financial fraud prevention in dynamic environments. *World Journal of Advanced Engineering Technology and Sciences.* — 2024. — Vol. 12, No. 2. — P. 021–034. — DOI: https://doi.org/10.30574/wjaets.2024.12.2.0266.
3. Tiwari A., Sanyal G. A Multi-factor Security Framework for Cloud Computing. *Journal of Cloud Computing*. – 2022. – Vol. 11, No. 2. – P. 143–167.
4. Bhatt S., Patwa P., Battu V. Secure adaptive authentication system using behavioral biometrics. *Journal of Information Security and Applications.* – 2023. – Vol. 62. – P. 102994.

5. Yang L., Guo Y., Ding X. A Comprehensive Review of Adaptive Authentication Systems for Financial Applications. *IEEE Access*. – 2023. – Vol. 11. – P. 9437–9451.
6. Chandre P., Gumaste S., Wangikar A., Deshmukh S. Adaptive Behavioral Authentication for Fraud Detection: Leveraging Real-Time User Behavior to Enhance Financial Security. *2024 First International Conference on Data, Computation and Communication (ICDCC)*. – IEEE, 2024. – P. 539–545. – DOI: 10.1109/ICDCC62744.2024.10961554.
7. Tandon A., Anitha C., Kataria A., Mohammed N. Q., Al-Khuzaie M. Y., Almulla A. A. Allometry Authentication in the Field of Finance: Creation of Well Secured System using AI Algo Based Systems. *2024 4th International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. – IEEE, 2024. – P. 962–967. – DOI: 10.1109/icacite60783.2024.10617124.
8. Aburbeian A. M., Fernández-Veiga M. Secure Internet Financial Transactions: A Framework Integrating Multi-Factor Authentication and Machine Learning. *AI* . – 2024. – Vol. 5, No. 1. – P. 177–194. – DOI: 10.3390/ai5010010.

References:

1. Bumiller A., Challita S., et al. On Understanding Context Modelling for Adaptive Authentication Systems. *ACM Transactions on Autonomous and Adaptive Systems*. – 2023. – Vol. 18, No. 1. – P. 3:1–3:35. – DOI: https://doi.org/10.1145/3582696.
2. Bello H. O., Ige A. B., Ameyaw M. N. Adaptive machine learning models: Concepts for real-time financial fraud prevention in dynamic environments. *World Journal of Advanced Engineering Technology and Sciences.* — 2024. — Vol. 12, No. 2. — P. 021–034. — DOI: https://doi.org/10.30574/wjaets.2024.12.2.0266.
3. Tiwari A., Sanyal G. A Multi-factor Security Framework for Cloud Computing. *Journal of Cloud Computing*. – 2022. – Vol. 11, No. 2. – P. 143–167.
4. Bhatt S., Patwa P., Battu V. Secure adaptive authentication system using behavioral biometrics. *Journal of Information Security and Applications.* – 2023. – Vol. 62. – P. 102994.
5. Yang L., Guo Y., Ding X. A Comprehensive Review of Adaptive Authentication Systems for Financial Applications. *IEEE Access*. – 2023. – Vol. 11. – P. 9437–9451.
6. Chandre P., Gumaste S., Wangikar A., Deshmukh S. Adaptive Behavioral Authentication for Fraud Detection: Leveraging Real-Time User Behavior to Enhance Financial Security. *2024 First International Conference on Data, Computation and Communication (ICDCC)*. – IEEE, 2024. – P. 539–545. – DOI: 10.1109/ICDCC62744.2024.10961554.
7. Tandon A., Anitha C., Kataria A., Mohammed N. Q., Al-Khuzaie M. Y., Almulla A. A. Allometry Authentication in the Field of Finance: Creation of Well Secured System using AI Algo Based Systems. *2024 4th International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. – IEEE, 2024. – P. 962–967. – DOI: 10.1109/icacite60783.2024.10617124.
8. Aburbeian A. M., Fernández-Veiga M. Secure Internet Financial Transactions: A Framework Integrating Multi-Factor Authentication and Machine Learning. *AI* . – 2024. – Vol. 5, No. 1. – P. 177–194. – DOI: 10.3390/ai5010010.