УДК 004.4 DOI: 10.18664/ikszt.v30i1.326713

ФІЛІМОНЧУК Т. В., к.т.н., доцент,
МАЙСТРЕНКО Г. В.
ОЦЕВИК В. А., магістрант
ВОЛК Д. М., студентка
(Харківський національний університет радіоелектроніки)

Модель уніфікованого підходу для розроблення мобільних і вебзастосунків із використанням мови swift

Анотація. Актуальність дослідження зумовлена стрімким розвитком мобільних і вебтехнологій, який висуває нові вимоги до створення універсальних, продуктивних і масштабованих застосунків. **Об'єктом дослідження** є уніфікований підхід для розроблення мобільних і вебзастосунків, що використовує мову Swift для реалізації клієнтської і серверної частин. Запропонована модель об'єднує архітектурні принципи, які дають змогу інтегрувати інтерфейс користувача, бізнес-логіку, базу даних, API і серверний шар у межах єдиного рішення. Такий підхід знижує витрати на розроблення, скорочує час впровадження нових функцій і забезпечує високу продуктивність системи. **Предметом дослідження** є оптимізація взаємодії між компонентами уніфікованої архітектурної моделі, побудованої мовою Swift, а також її здатність забезпечувати консистентність, безпеку та ефективність у багатоплатформних проєктах. Застосування мови Swift для всіх складових системи дає змогу створити цілісну та масштабовану екосистему, яка відповідає сучасним вимогам продуктивності, безпеки та зручності підтримки. **Результати** дослідження демонструють, що використання мови Swift для побудови уніфікованої моделі забезпечує зменшення кількості інструментів і мов програмування, необхідних для реалізації проєкту, що сприяє кращій взаємодії між компонентами, підвищенню стабільності, спрощенню інтеграції нових функцій та ефективному управлінню даними. Висновки дослідження підтверджують, що запропонована модель є надійною основою для розроблення багатоплатформних застосунків, які відповідають сучасним стандартам продуктивності, масштабованості та безпеки.

Ключові слова: мова Swift, мобільний застосунок, вебзастосунок, уніфікована архітектура, продуктивність, безпека, інтеграція, багатоплатформна розробка, клієнт-серверна модель.

Постановка проблеми

Розроблення програмних застосунків (веб або мобільних) починається з аналізу їхніх складових. Щоб у результаті застосунок став зручним і зрозумілим для кінцевого користувача, слід серйозно підійти до його проєктування та дотримуватися логіки з поданням інформації. Зазвичай застосунок буде приваблювати користувачів лише тоді, коли його розділи мають гарно структуровану ієрархію. На сьогодні модель застосунку можна порівняти з інструментарієм, який допомагає розробнику прогнозувати подальший його розвиток, тому розроблення уніфікованої моделі побудови застосунків (веб або мобільних) є актуальним напрямом.

З розвитком мобільних і вебтехнологій дедалі більше компаній і розробників прагнуть створювати універсальні рішення, які могли б одночасно функціонувати на кількох платформах. В умовах сучасного швидкого темпу розвитку індустрії вимоги до програмного забезпечення значно підвищилися:

користувачі сподіваються на високу продуктивність, швидке реагування, гнучкість і зручність використання. Це створює потребу в підходах, щоб об'єднати мобільний front-end і серверний back-end, забезпечуючи при цьому ефективність і легкість підтримки. Одним із таких перспективних підходів є використання мови Swift для розроблення не тільки мобільних, але й серверних компонентів.

Мова Swift була створена компанією Apple для розроблення додатків під iOS, але поступово трансформувалася в універсальну мову програмування з відкритим кодом, яка прийнятна для широкого спектру завдань, включаючи серверне розроблення [1]. Завдяки такому фреймворку, як Vapor, Swift отримала підтримку серверних технологій, що дає змогу створювати вебсервіси, керувати базами даних, убезпечувати додатки та реалізовувати їхню бізнес-логіку. Цей новий підхід відкриває можливості для розробників створювати інтегровані додатки з єдиною кодовою базою, що значно спрощує роботу, зменшує час розроблення та підтримки, а також сприяє швидкому реагуванню на зміну вимог ринку.

Аналіз останніх досліджень і публікацій

У процесі аналізу сучасних підходів для розроблення програмних застосунків (веб і мобільних) розглянуто низку наукових публікацій, які стосуються архітектури, інструментарію та принципів їх проєктування.

У роботі [2] проведено порівняльний аналіз сучасних засобів розроблення мобільних застосунків, таких як React Native, Swift і Kotlin. Автори наводять переваги кросплатформних рішень щодо швидкості розроблення, особливо у випадках, коли необхідно швидко вивести продукт на ринок. Водночас зазначено, що нативні інструменти, такі як мова Swift для iOS, забезпечують найвищу продуктивність, стабільність і глибшу інтеграцію з платформою, що є головним для додатків із високими вимогами щодо продуктивності.

Публікації [3, 4] зосереджені на виборі архітектурних підходів для мобільних додатків. Автори зазначають, що моделі MVC і MVVM мають значний потенціал для оптимізації розроблення front-end застосунку, оскільки можна розділити логіку програми, бізнес-логіку та інтерфейс користувача. Такий підхід спрощує розроблення, тестування та подальше масштабування програмного забезпечення. У публікації [3] також розглянуто використання архітектурного принципу REST API як основного засобу комунікації між клієнтською та серверною частинами для стандартизації передавання даних і полегшення інтеграції різних компонентів.

Описаний у роботі [5] підхід зосереджений на оптимізації архітектури клієнтської частини мобільного Andriod-додатка через пошарову структуру, реалізовану за допомогою мови Kotlin і набору бібліотек і компонентів. Використання у проєкті підходу Clean Architecture дає змогу подати кінцевий продукт як набір шарів, з'єднаних відповідними правилам, щоб масштабувати проєкт без зайвих зусиль.

Об'єктом дослідження в роботі [6] виступає модель мобільного застосунку, яку розглядають з точки зору множини взаємопов'язаних між собою елементів. Автори роблять спробу розглянути модель з точки зору розробника: зручний інтерфейс користувача, клієнтська та серверна частини, використання заходів безпеки та модулів тестування. Також не останнє місце в запропонованій моделі відведено модулю, який відповідає за моніторинг функціонування застосунку з подальшою аналітикою отриманих результатів. У роботі проаналізовано вплив кожного окремого компонента моделі на функціональність і продуктивність застосунку в цілому, спрямовуючи увагу на оптимізацію його роботи для досягнення кращих результатів.

У роботі [7] проаналізовано фреймворк, який можна використовувати для побудови різних чат-ботів. Модель фреймворку, яку пропонують автори, містить нові архітектурні рішення, спрямовані на вирішення специфічних завдань чат-боту. Використання в моделі модулів, які присутні в будь-якому фреймворку, але з огляду на нову бот-перспективу, допомагає розробникам у вирішенні специфічних завдань. Наприклад, рекомендовано розширити функціонал модуля безпеки з шифруванням тексту повідомлень, модуль локалізації та модуль адміністрування чат-боту.

Автори роботи [8] описують структуру та функціонал мобільного застосунку у зв'язці зі службами, із якими він взаємодіє. Основну увагу приділено інтерфейсу користувача, який є основним об'єктом розроблення, працюючи зі сторонніми ресурсами і базою даних. Застосунок демонструє високу гнучкість і масштабованість завдяки додаванню окремих служб і використанню інтерфейсу сторонньої компанії, що у свою чергу дає йому швидкі та точні відповіді на запити користувача.

У роботі [9] здійснена спроба надати модель мобільного застосунку для операційної системи iOS, яка містить набір шаблонів для створення відповідних модулів, використовує сервіси для подальшої роботи з мережею та базами даних. Також модель пропонує не тільки додавання сучасних механізмів, а також розширення базових класів і підмодуів, які у свою чергу допомагають розробнику з навігацію, побудованою на основі координатора.

У статті [10] розглянуто переваги та недоліки використання JavaScript-фреймворків, таких як Angular, React, Vue, Backbone, Ember, Knockout, для створення front-end. Кожен із розглянутих інструментів дає розробнику низку переваг, але, розробляючи кінцевий проєкт, слід орієнтуватися на потреби замовника, які необхідно реалізувати. Більшість розглянутих фреймворків не забезпечує можливість реалізації як клієнтської, так і серверної частин застосунків однією мовою, що ускладнює інтеграцію та збільшує витрати на їх розроблення.

У роботі [11] модель вебсайту пропоновано розглядати як граф і використовувати її у зв'язці з алгоритмом сканування сторінок, розробленим авторами. Отриману після сканування інформацію зберігають у зручному вигляді та використовують у подальшому в обчисленнях деяких метричних характеристик, щоб проаналізувати структурну зв'язність вебзастосунків різного типу складності.

У статті [12], присвяченій аналізу фреймворку Ember.js для створення вебзастосунків, ідеться про його ефективність у розробленні front-end. Використання фреймворку Ember.js наділяє розробника низкою інструментів для розгортання складних клієнтських застосунків, які мають високу продуктивність і дають змогу ефективно керувати його станом, але не забезпечують універсальність, реалізуючи клієнтську та серверну частини однією мовою.

Проведений порівняльний аналіз виявив, що на сьогодні не існує уніфікованої моделі для побудови мобільних і вебзастосунків. Є спроби навести моделі

окремо для веброзробок і мобільних застосунків із точки зору їхньої цільової спрямованості. Аналіз також показав, що в кінцевому варіанті застосунку присутні такі складові: UI – інтерфейс користувача; BL – модуль бізнес-логіки, який реалізує основні функції застосунку; БД, яка зберігає та організує дані застосунку; API – інтерфейс, який забезпечує взаємодію між клієнтською та серверною частинами:

$$M = \{UI, BL, DB, API\}. \quad (1)$$

Слід зазначити, що наведена модель (1) є основою лише для одного напряму розроблення та впровадження програмного застосунку (веб або мобільного) для задоволення вимог користувачів.

Мета дослідження

Метою дослідження є впровадження уніфікованого підходу для розроблення мобільних і вебзастосунків різного рівня складності.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати сучасні складові моделей, орієнтовані на розроблення мобільних і вебзастосунків;

– виявити переваги та недоліки сучасних моделей, зокрема їхні гнучкість і масштабованість;

– модифікувати математичну модель за рахунок додавання складових, які дають змогу отримати від кінцевого програмного продукту високий рівень швидкодії, продуктивності, адаптивності та безпеки.

Викладення основного матеріалу

Проведений аналіз наявних аналогів виявив певні обмеження в розумінні сучасних підходів для розроблення застосунків. Більшість із них зосереджені на використанні сторонніх фреймворків для back-end, таких як Node.js або Django, тоді як застосування нативного back-end, написаного мовою Swift із використанням фреймворку Vapor, залишається недостатньо вивченим. Такий підхід може суттєво підвищити продуктивність системи та забезпечити більш природну інтеграцію між клієнтською та серверною частинами. Крім того, у дослідженнях недостатньо уваги приділено перевагам SwiftUI як сучасного інструменту для створення адаптивного та інтерактивного інтерфейсу користувача, який значно спрощує розроблення завдяки використанню декларативного підходу.

Зважаючи на виявлені недоліки, подано власну модель, яка дає змогу подолати наявні обмеження та забезпечує ефективну інтеграцію front-end і back-end компонентів, високий рівень продуктивності, безпеки та легкість масштабування. Ця модель враховує всі основні аспекти сучасної розробки, забезпечуючи комплексний підхід для створення програмних застосунків (веб і мобільних):

$$M=\{P, A, PL, UI, UX, SC, DB, NI, S, TS, AM\}, \quad (2)$$

де P (platform) – платформа розробки;

A (architecture) – архітектура розробки;

PL (programming language) – мова програмування;

UI (user interface) – інтерфейс користувача;

UX (user experience) – досвід користувача;

SC (server component) – серверна складова застосунку;

DB (database) – база даних;

NI (network interface) – мережевий інтерфейс;

S (security) – безпека застосунку;

TS (test suite) – набір тестів;

AM (analytics and monitoring) – аналітика та моніторинг застосунку.

Запропонована модель також зменшує вартість розроблення завдяки єдиному стеку технологій. Оскільки всі компоненти моделі можуть бути реалізовані за допомогою мови Swift, зникає потреба у використанні додаткових мов або технологій для різних частин системи, що особливо важливо в умовах обмежених ресурсів, коли кожна година розробника має значення. Крім того, цей підхід підвищує консистентність коду, оскільки всі частини проєкту розроблено в єдиному стилі та з використанням єдиних парадигм. Кожен із цих компонентів впливає на загальну структуру застосунку, створюючи стабільну, продуктивну та легко масштабовану архітектуру. Розглянемо більш детально кожен із них і його роль у запропонованій моделі.

На сучасному ринку розроблення програмного забезпечення основними платформами **(P)** є iOS, Android, веб і кросплатформні рішення. Кожна з них має свої переваги: iOS вирізняється стабільністю і глибокою інтеграцією з екосистемою Apple; Android забезпечує гнучкість і широке охоплення пристроїв; веб пропонує універсальність, а кросплатформні інструменти дають змогу одночасно створювати програми для кількох середовищ.

Запропонована модель орієнтована на платформу iOS із можливістю розширення для iPadOS. Вона забезпечує високу продуктивність, глибоку інтеграцію з екосистемою Apple (iCloud, Apple Pay, HealthKit) і підтримку інноваційних технологій, таких як ARKit і Core ML, що робить її ідеальним вибором для створення сучасних, стабільних і масштабованих рішень.

Архітектура **(A)** розроблення мобільного та вебзастосунків – це сукупність підходів, правил і шаблонів, які забезпечують ефективну організацію роботи програмного забезпечення. На сьогодні найбільш поширеними архітектурними підходами є:

- **MVC (Model-View-Controller)** – використовують для структурування застосунків шляхом розподілу даних (Model), інтерфейсу користувача (View) і логіки управління (Controller). Цей архітектурний підхід прийнятний для невеликих і середніх проєктів, забезпечує простоту реалізації та швидке тестування, але ускладнений процес масштабування застосунку в разі потреби;

- **MVVM (Model-View-ViewModel)** – забезпечує чіткий розподіл логіки та інтерфейсу завдяки використанню ViewModel, що спрощує процеси тестування, масштабування та роботу з асинхронними даними. Використання цього підходу ідеальне для front-end розроблення мобільних і вебзастосунків, де необхідна гнучкість і чітка структура;

- **VIPER (View-Interactor-Presenter-Entity-Router)** – слід використовувати для складних проєктів, де необхідне забезпечення модульності та високої масштабованості, чого досягають завдяки розподілу відповідальності. Як зауваження для використання цього архітектурного підходу слід зазначити, що він є складнішим у реалізації та потребує більше часу на розроблення;

- **Clean Architecture** – передбачає побудову системи з чітко визначеними рівнями (інтерфейс, бізнес-логіка, дані), що робить її максимально масштабованою та адаптивною. Використання такої архітектури доцільно для великих проєктів, але також потребує додаткових ресурсів.

У запропонованій моделі використовують архітектурний підхід MVVM на front-end і MVC на серверній частині, оскільки вони поєднують простоту реалізації та високу ефективність:

- **MVVM** забезпечує відокремлення логіки від інтерфейсу, що сприяє легкому оновленню дизайну та підтримці коду, а також дає змогу ефективно працювати з SwiftUI;

- **MVC** у фреймворку Vapor структурує серверний код, роблячи його зрозумілим, підтримуваним і зручним для інтеграції з базою даних та API.

Отже, вибрані архітектурні підходи дають змогу створити гнучку, масштабовану і стабільну систему, яка відповідає сучасним вимогам розроблення мобільних і вебзастосунків.

Одним із популярних підходів для кросплатформного розроблення є використання фреймворку Flutter і мови програмування Dart. Ця зв’язка дає змогу створювати застосунки для iOS та Android з єдиним кодом, пропонує широкий набір віджетів для інтерфейсу, підтримує плавні анімації та асинхронну обробку даних. Проте її обмеження включають знижену продуктивність порівняно з нативними рішеннями, труднощі доступу до платформних API і збільшений розмір додатка.

Запропонована модель використовує мову Swift для клієнтської частини та фреймворк Vapor для серверної. Мова програмування Swift забезпечує високу продуктивність і сучасні можливості, такі як асинхронна обробка даних, а фреймворк Vapor дає змогу створювати REST API, керувати базами даних через ORM Fluent і убезпечувати застосунок. Завдяки єдиній мовній базі забезпечена безшовна інтеграція між компонентами, що скорочує час розроблення, підвищує стабільність і спрощує масштабування застосунку в разі потреби.

Інтерфейс користувача **(UI)** є основним елементом будь-якого застосунку, і сучасні фреймворки значно спрощують його розроблення. Наприклад, SwiftUI забезпечує декларативний підхід, що дає змогу розробникам створювати складні інтерфейси з мінімумом коду. Компоненти, створені за допомогою SwiftUI, автоматично адаптуються під різні розміри екранів, що особливо важливо для пристроїв iOS та iPadOS. Крім того, SwiftUI підтримує інтерактивні елементи, анімації та плавні переходи, покращуючи загальний досвід користувача.

Ще одним прикладом є технологія Jetpack Compose для ОС Android, яка також використовує декларативний підхід. Вона спрощує створення складних макетів і взаємодій, а також забезпечує високу швидкість розроблення завдяки своїй інтеграції з іншими інструментами екосистеми Android.

У випадку використання SwiftUI головною перевагою є глибока інтеграція з екосистемою Apple, що дає змогу ефективно використовувати унікальні можливості платформи, такі як динамічні шрифти, системні кольори та доступність, що робить його ідеальним вибором для розроблення інтерфейсів, які відповідають сучасним стандартам продуктивності та дизайну.

Досвід користувача **(UX)** визначений емоціями, враженнями та рівнем задоволення, які отримує користувач під час взаємодії з програмним продуктом. Як уважає розробник, головним завданням є створення такого інтерфейсу та функціоналу, які не лише задовольняють потреби користувача, але й перевищують його очікування, забезпечуючи зручність, доступність і естетичну привабливість. Дотримання рекомендацій Apple у Human Interface Guidelines (HIG) допомагає розробникам створювати інтуїтивно зрозумілі та гармонійні застосунки. Нативні компоненти, такі як системні кнопки, списки, вкладки та піктограми, забезпечують єдиний стиль і спрощують адаптацію до інтерфейсу. Принципи HIG гарантують не лише функціональність, але й узгодженість дизайну з іншими додатками екосистеми Apple.

На думку користувача, плавні анімації, інтерактивні елементи та логічно організовані розділи допомагають утримувати увагу і створюють відчуття комфорту під час роботи з застосунком. Використання рекомендацій HIG забезпечує зручну навігацію та естетику, що сприяє довірі до продукту. Наприклад, адаптація дизайну для різних екранів, оптимізація часу відгуку та забезпечення доступності

для людей з обмеженими можливостями роблять UX основою успішного застосунку.

Серверна частина **(SC)** є важливим компонентом мобільних і вебзастосунків, забезпечуючи обробку даних, реалізацію бізнес-логіки та інтеграцію з іншими сервісами. У мобільних додатках сервер обробляє запити клієнтів і зберігає дані, тоді як у вебзастосунках він часто також відповідає за динамічне формування контенту. Основними викликами серверної розробки є забезпечення продуктивності, безпеки, масштабованості та ефективної взаємодії з базами даних.

У запропонованій моделі серверна частина реалізована за допомогою фреймворку Vapor мовою Swift. Vapor дає змогу створювати REST API для взаємодії з клієнтом, ефективно працювати з базами даних через ORM Fluent і забезпечує високий рівень безпеки завдяки підтримці middleware для аутентифікації та шифрування. Асинхронна архітектура Vapor забезпечує високу продуктивність і здатність обробляти велику кількість запитів, роблячи його ідеальним вибором для сучасних застосунків. Використання єдиної мовної бази на основі Swift забезпечує природну інтеграцію між клієнтською та серверною частинами, створюючи цілісну та ефективну систему.

База даних **(DB)** є основним компонентом будь-якого сучасного застосунку, що забезпечує надійне зберігання та управління інформацією. У вебзастосунках база даних використовувана для зберігання контенту, даних користувача, а також журналів транзакцій. У мобільних застосунках вона часто є локальним сховищем для тимчасового зберігання даних або кешування, що забезпечує швидкий доступ до інформації навіть без підключення до мережі.

Для зберігання даних у запропонованій моделі використовують PostgreSQL – реляційну базу даних, відому своєю стабільністю, безпекою та масштабованістю. ORM Fluent у Vapor дає змогу значно спростити роботу з PostgreSQL. Він забезпечує створення моделей для роботи з даними, управління міграціями та побудову запитів без необхідності прямого використання SQL. Це сприяє скороченню часу розроблення та зменшенню кількості помилок, пов’язаних із взаємодією з базою даних.

Така інтеграція дає змогу забезпечити швидку і зручну взаємодію між серверною частиною застосунку та базою даних. PostgreSQL разом із Fluent робить модель ефективною, масштабованою та готовою до роботи з великими обсягами даних і складною бізнес-логікою. Цей підхід забезпечує стабільну основу для зберігання інформації, адаптовану для мобільних і вебзастосунків.

Мережеві з'єднання **(NI)** забезпечують обмін даними між клієнтською та серверною частинами через REST API. У сучасних застосунках це здійснюється через стандартні протоколи HTTP/HTTPS, які гарантують безпеку та простоту передавання даних. Наприклад, клієнт (мобільний або вебзастосунок) формує запит до сервера за допомогою таких інструментів, як URLSession на iOS, і отримує відповіді у форматі JSON, що легко обробляти.

Використання архітектурного стилю REST API в запропонованій моделі є оптимальним вибором для мережевої взаємодії завдяки його простоті, стандартизованості та ефективності. Він дає змогу використовувати зрозумілі HTTP-методи (GET, POST, PUT, DELETE) для виконання операцій над ресурсами, а передавання даних у форматі JSON спрощує інтеграцію між клієнтом і сервером. У запропонованій моделі реалізація стилю REST API здійснюється за допомогою фрейморку Vapor, що дає змогу легко створювати маршрути для обробки запитів і повертати структуровані дані. Такий підхід забезпечує стабільну, швидку та безпечну комунікацію між компонентами системи.

Безпека **(S)** є одним із найважливіших аспектів сучасних програмних застосунків через зростаючу кількість оброблюваних конфіденційних даних. Витоки особистої інформації, фінансових даних або зломи систем можуть призводити до значних збитків, тому забезпечення захисту даних є пріоритетом для будь-якого кінцевого продукту.

У сучасних застосунках використано такі методи захисту, як шифрування запитів через протокол HTTPS, аутентифікація за допомогою JWT-токенів і двофакторна авторизація. Протокол HTTPS гарантує шифрування всіх переданих даних, захищаючи їх від перехоплення, тоді як JWT забезпечує безпечну ідентифікацію користувачів. Крім того, використовують захищені протоколи для зберігання даних і валідацію на боці сервера для запобігання SQL-ін’єкціям.

У запропонованій моделі реалізація безпеки здійснюється через фреймворк Vapor, який підтримує middleware для аутентифікації, обробку JWT-токенів і захист від несанкціонованого доступу. Шифрування запитів через HTTPS забезпечує надійність передавання даних, а налаштування middleware адаптують систему до вимог сучасних стандартів безпеки.

Тестування **(TS)** є обов’язковим етапом розроблення застосунку, який забезпечує стабільність, надійність і якість програмного забезпечення. Воно виявляє та усуває помилки на ранніх етапах, що мінімізує ризик збоїв після впровадження змін чи масштабування застосунку. Тестування також гарантує, що функціонал працює відповідно до заданих специфікацій, забезпечуючи якісний досвід користувача.

Для розроблення застосовують різні види тестування:

- unit-тести, які перевіряють окремі модулі чи компоненти системи;

- UI-тести, які оцінюють коректність роботи інтерфейсу користувача;

- інтеграційні тести, які перевіряють взаємодію між різними компонентами системи;

- тестування API, яке гарантує стабільність і продуктивність серверної частини.

У запропонованій моделі використано інструмент XCTest, інтегрований у середовище розробки Xcode, що дає змогу створювати і виконувати як unit, так і UI-тести для клієнтської частини. Для серверної частини, розробленої на Vapor, тестують API та інтеграційні тести, які забезпечують коректність взаємодії між сервером і клієнтом. Завдяки цьому модель гарантує стабільність застосунку навіть з внесенням змін або масштабуванням, а також зберігає високу якість роботи в реальних умовах.

Аналітика та моніторинг **(AM)** є важливими складовими сучасного застосунку, які забезпечують глибоке розуміння поведінки користувачів і продуктивності системи. Інтеграція інструментів аналітики дає змогу відстежувати активність користувачів, аналізувати взаємодію з різними елементами інтерфейсу та збирати ключові метрики, такі як конверсії, час сесій і залученість.

Моніторинг продуктивності системи забезпечує можливість своєчасного виявлення та вирішення потенційних проблем, таких як високий час відгуку сервера, перевантаження бази даних чи збої в роботі API. Використання таких інструментів, як Google Analytics, Firebase Analytics, Mixpanel або Sentry, автоматизує процес збору та аналізу даних, забезпечуючи актуальну інформацію для ухвалення рішень.

У запропонованій моделі аналітика та моніторинг інтегровані як невід'ємна складова системи, побудована з використанням мови Swift. Інструменти аналітики збирають дані на клієнтському боці через SwiftUI і URLSession, а моніторинг продуктивності реалізовано на серверній частині за допомогою фреймворку Vapor. Це дає змогу розробникам отримувати комплексну картину роботи застосунку, ураховувати реальні дані користувачів під час оптимізації функціоналу та забезпечувати стабільність системи.

Такий підхід підвищує якість досвіду користувача, сприяє своєчасному реагуванню на проблеми й адаптації функціонала відповідно до потреб цільової аудиторії. Інтеграція аналітики та моніторингу уніфікує процес управління даними, сприяючи підвищенню ефективності розроблення та підтримки застосунків.

Висновки

У результаті проведених авторами досліджень запропоновано уніфіковану модель для розроблення мобільних і вебзастосунків, що використовує мову Swift як основну мову програмування для клієнтської та серверної частин. Такий підхід має численні переваги, які роблять його особливо актуальним у сучасних умовах високих вимог щодо продуктивності, безпеки та інтеграції застосунків.

Основними перевагами уніфікованої моделі є:

- зменшення витрат і часу на розроблення: використання єдиної мовної бази дає змогу уникнути дублювання коду та інтегрувати функціональність як для клієнта, так і сервера, що у свою чергу спрощує розроблення, полегшує впровадження нових функцій та оновлень, а також знижує витрати на підтримку застосунку в подальшому;

- консистентність в узгодженість коду: спільна кодова база для мобільного інтерфейсу, бізнес-логіки та серверного шару застосунку підвищує узгодженість між його компонентами, що не лише зменшує кількість потенційних помилок, а й забезпечує єдність функціонала для різних платформ;

- гнучкість і масштабованість: архітектура на базі використання мови Swift дає змогу легко додавати нові функції та інтегрувати такі зовнішні сервіси, як аналітика або хмарні сховища, не змінюючи основну структуру. Крім того, запропонована модель легко адаптується для підтримки різних платформ Apple, як iPadOS, watchOS і macOS, що дає можливість охопити ширшу аудиторію та забезпечити мультиплатформну підтримку;

- покращена безпека: використання сучасних методів шифрування, JWT-токенів для авторизації та двофакторної аутентифікації підвищує захист даних користувачів і запобігає несанкціонованому доступу до інформації. Слід зазначити, що мова Swift забезпечує надійну реалізацію безпеки, що є особливо важливим для застосунків, які обробляють чутливі дані;

- інтеграція аналітики та моніторингу: інструменти аналітики дають змогу відстежувати активність користувачів, оцінювати продуктивність системи та своєчасно реагувати на можливі проблеми, щоб покращувати функціональність застосунку на основі реальних даних, що підвищує якість досвіду користувача;

- стабільність завдяки тестуванню: включення таких компонентів тестування, як XCTest для unit-тестування та інтеграційного тестування, підвищує стабільність роботи застосунку. Тестування виявляє та вирішує потенційні проблеми на ранніх етапах, зменшуючи ризик збоїв після випуску нових оновлень.

Слід зазначити, що запропонована модель може здатися надто складною для простих мобільних і вебзастосунків, однак навіть часткове її використання дає змогу розробникам швидко масштабувати проєкт у подальшому. Наприклад, спочатку можна створити простий застосунок, використовуючи лише необхідні компоненти, а згодом, без значних змін в основній структурі, додати нові функції.

Як приклад можна навести найпростіший мобільний застосунок для підрахування калорій. Розробляючи його структуру, на першому кроці можна використовувати такі компоненти, як фреймворк Flutter для клієнтської частини, а саме модулів інтерфейсу взаємодії, клієнтської частини та обробки даних, а Firebase для розгортання бази даних. Іншим прикладом є розроблення трейдингової біржи, яка потребує більшої функціональності та безпеки. Для реалізації цього застосунку розробник може використати Flutter для клієнтської частини, FastAPI – серверної та бізнес-логіки, Postgres – сховища даних, Docker – контейнеризації, Splunk – подальшого логування, Sentry – аналітики, kafka – кешування, Github Actions – автоматизації оновлення застосунку на сервері або в хмарі.

Але попри численні переваги застосування запропонованої моделі, є певні виклики, зокрема обмежена кількість інструментів і бібліотек для серверного розроблення мовою Swift порівняно з іншими мовами, такими як Python чи Node.js. Також розробникам може знадобитися додатковий час для вивчення фреймворків на кшталт Vapor, які менш поширені, ніж інші серверні рішення.

Загалом уніфікована модель на базі мови Swift є перспективним рішенням для розробників, які прагнуть створити мультиплатформний застосунок з єдиною кодовою базою. Вона полегшує розроблення, скорочує витрати і час, необхідні для впровадження функцій, і дає високий рівень продуктивності та безпеки. Такий підхід може стати базою для створення нових інноваційних продуктів, що відповідають сучасним вимогам ринку та легко адаптуються до швидкозмінного середовища технологій.

Список використаних джерел

1. Apple Inc: Swift Documentation. URL: https://developer.apple.com/swift/.

2. Ічанська Н. В., Улько С. І. Основні аспекти створення мобільних додатків та вибір інструментів їх розробки. *Системи управління, навігації та зв'язку. Збірник наукових праць*. Полтава: ПНТУ, 2020. Вип. 1(59). С. 74-78. doi: 10.26906/SUNZ.2020.1.074.

3. Середюк Г. В., Паламарчук Є. А. Мобільний додаток на платформі IOS з використанням архітектурного патерну MVVM. *Матеріали L науково-технічної конференції підрозділів ВНТУ*. Вінниця, 2021. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2021/paper/view/12157.

4. Данілов Д. В. Дослідження основних шаблонів Model View Controller та Model View View Model для проектування Android додатків. *III Всеукраїнська конференція здобувачів вищої освіти і молодих учених «Інноватика в освіті, науці та бізнесі: виклики та можливості»*. 2022. С. 129-132.

5. Козуб Г. О., Козуб Ю. Г., Могильний Г. А., Жуков А. В. Розробка мобільного Android-додатку з застосуванням принципів Clean Architecture. *Вісник Східноукраїнського національного університету імені Володимира Даля*. 2021. № 5 (269). С. 5-10. doi: 10.33216/1998-7927-2021-269-5-5-10.

6. Бешта В. С., Комаричев А. В., Філімончук Т. В., Баранєй Д. І. Модель мобільного додатку, яка орієнтована на обробку даних. *Системи управління, навігації та зв'язку. Збірник наукових праць*. Полтава: ПНТУ, 2024. Т. 3 (77). С. 80-83. doi: doi.org/10.26906/SUNZ.2024.3.080.

7. Філімончук Т. В., Хабазня Д. Ю. Специфіка чат-ботів як парадигма розробки моделі фреймворку. *Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління: тези доповідей 11 Міжнародної науково-технічної конференції*. Баку: ВА ЗС АР; Харків: НТУ «ХПІ»; Київ: ДП «ПДПРОНДІАВІАПРОМ»; Жиліна: УмЖ, 2021. Т. 2. С. 40.

8. Петрунь В. М., Філімончук Т. В., Кравченко П. О. Мобільний застосунок для дослідження космосу з інтеграцією чат боту. *Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління: тези доповідей 14 Міжнародної науково-технічної конференції*. Баку: НУО АР; Харків: НТУ «ХПІ»; Харків: ХНУРЕ; Харків: НАУ «ХАІ»; Жиліна: УмЖ, 2024. Т. 1: секція 2. С. 109.

9. Філімончук Т. В., Оцевик В. А. Модель мобільного застосунку iOS. *Проблеми інформатизації: тези доповідей 9 Міжнародної науково-технічної конференції*. Черкаси: ЧДТУ; Харків: НТУ «ХПІ»; Баку: ВАЗС АР; Бельсько-Бяла: УТіГН, 2021. Т. 2. С. 98.

10. Танянський О., Руденко Д. Порівняльний аналіз популярних JavaScript-фреймворків та бібліотек для front-end розробки. *Інформаційні системи та технології : матеріали статей 7-ї Міжнародної науково-технічної конференції, Коблеве-Харків, 2018*. Харків: ХНУРЕ, 2018. С. 347-349.

11. Гук Н. А., Диханов С. В., Матющенко О. Д. Алгоритм побудови моделі веб-сайту. *Вісник Харківського національного університету імені В. Н. Каразіна*. 2020. Вип. 47. С. 25-34. doi: 10.26565/2304-6201-2020-47-03.

12. Прудченко А. Ю. Аналіз фреймворку Ember.js для створення високопродуктивних веб-додатків. *Матеріали 78-ї студентської науково-технічної конференції «Тиждень студентської науки». Секція «Інформаційні та телекомунікаційні технології». НТУ «Дніпровська політехніка».* 2023. С. 382-384.

T. Filimonchuk, H. Maistrenko, V. Otsevyk, D. Volk
Unified approach model for mobile and web applications development using Swift

Abstract. *The relevance of the research is due to the rapid development of mobile and web technologies,*

which puts forward new requirements for creating universal, productive and scalable applications. ***The object of the research*** *is a unified approach to developing mobile and web applications. It uses Swift language to implement the client and server parts. The proposed model combines architectural principles that allow integration of the user interface, business logic, database, API and server layer in a single solution. This approach reduces development costs, shortens the time to implement new functions and ensures high system efficiency.* ***The subject of the research*** *is the optimisation of interaction between components of a unified architecture model and its ability to ensure consistency, security and efficiency in multi-platform projects. Using the Swift programming language for all system components creates a holistic and scalable ecosystem. It matches modern requirements for efficiency, security and ease of support.* ***The study results*** *show that using the Swift programming language reduces the number of tools and programming languages. This contributes to better interaction between components, increased stability, easier integration of new functions and efficient data management. The study's findings confirm that the proposed model is a reliable basis for developing multi-platform applications that meet modern efficiency, scalability and security standards.*

Keywords: *Swift language, mobile application, web application, unified architecture, efficiency, security, integration, multi-platform development, client-server model.*

Філімончук Тетяна Володимирівна, *кандидат технічних наук, доцент кафедри «Електронних обчислювальних машин», Харківський національний університет радіоелектроніки, Харків, Україна, E-mail: tetiana.filimonchuk@nure.ua, ORCID: 0000-0002-4380-504X*

Майстренко Галина Валеріївна, старший викладач кафедри «Електронних обчислювальних машин», Харківський національний університет радіоелектроніки, Харків, Україна, e-mail: halyna.maistrenko@nure.ua, ORCID: 0000-0002-8126-9997

Оцевик Владислав Андрійович, магістрант, Харківський національний університет радіоелектроніки, Харків, Україна, *E*-mail: vladyslav.otsevyk@nure.ua

Волк Дар'я Максимівна, студентка, Харківський національний університет радіоелектроніки, Харків, Україна, e-mail: daria.volk@nure.ua, ORCID:0009-0008-1425-485X

Tetiana Filimonchuk, *PhD, Associate Professor of the Department of Electronic Computers, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine, E-mail: tetiana.filimonchuk@nure.ua, ORCID: 0000-0002-4380-504X*

Halyna Maistrenko, *Senior Lecturer of the Department of Electronic Computers, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine, E-mail: halyna.maistrenko@nure.ua, ORCID: 0000-0002-8126-9997*

Otsevyk

Vladyslav Otsevyk, master's student, *Kharkiv National University of Radio Electronics, Kharkiv, Ukraine, E-mail:* vladyslav.otsevyk@nure.ua

Darya Volk, student, *Kharkiv National University of Radio Electronics, Kharkiv, Ukraine, E-mail:* daria.volk@nure.ua, ORCID:0009-0008-1425-485X