МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ

Кафедра комп'ютерних наук та кібербезпеки

На правах рукопису

ЦВИД ВЛАДИСЛАВ ВАСИЛЬОВИЧ

РОЗРОБКА ЗАХИЩЕНОЇ ІНТЕГРОВАНОЇ СИСТЕМИ АДМІНІСТРУВАННЯ МОБІЛЬНИХ ДОДАТКІВ

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології
Робота на здобуття освітнього ступеня «магістр»

Науковий керівник:
ЛАПТЄВ ОЛЕКСАНДР АНАТОЛІЙОВИЧ,
д-р техн. наук, доцент кафедри
комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ

Протокол №_____
засідання кафедри комп'ютерних наук
та кібербезпеки
від _____ 20___ р.
Завідувач кафедри
(_____)_____
(підпис) ПІБ

ЛУЦЬК – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – Операційна Система

ПЗ – Програмне Забезпечення

ШПЗ – Шкідливе Програмне Забезпечення

ACL – Access Control List

API – Application Programming Interface

CRUD – Create , Read , Update , Delete

DoS – Denial-of-Service

HTML – HyperText Markup Language

HTTP(s) – HyperText Transfer Protocol (secure)

JSON – JavaScript Object Notation

MVC – Model-View-Controller

OWASP – Open Web Application Security Projec

PWA – Progressive Web Application

SQL – Structured Query Language

ВСТУП

Актуальність теми: У сучасних умовах стрімкого розвитку цифрових технологій ефективність роботи IT-компаній безпосередньо залежить від рівня автоматизації їхніх бізнес-процесів. Зростання конкуренції на ринку мобільних застосунків, постійні зміни в політиках Google Play, App Store та рекламних платформ, а також збільшення обсягів рутинних операцій створюють потребу у впровадженні сучасних інструментів цифрової трансформації.

Важливим складником цифрової трансформації є застосування автоматизованих систем, серед яких значне місце займають Telegram-боти. Вони забезпечують можливість моніторингу стану мобільних продуктів, оперативного обміну інформацією, оптимізації внутрішніх процесів та зменшення впливу людського фактору. Використання чат-ботів дає змогу IT-компаніям оперативно реагувати на зміни, підвищувати ефективність управління цифровими продуктами та підтримувати стабільність бізнес-процесів.

Тема має особливе значення для України, де цифрова економіка продовжує розвиватися, а IT-сфера залишається одним із ключових драйверів національної конкурентоспроможності. Автоматизація та запровадження інтелектуальних систем сприяють стабільній роботі українських компаній навіть за умов воєнних викликів та обмежених ресурсів.

Отже, дослідження сучасних підходів до цифрової трансформації, а також створення та впровадження автоматизованого інструмента є актуальним як у науковому контексті, так і з практичної точки зору.

Наукова новизна. Наукова новизна роботи полягає у комплексному дослідженні процесів цифрової трансформації IT-компаній та обґрунтуванні ролі автоматизованих інструментів у підвищенні ефективності управління мобільними продуктами. У межах дослідження вперше здійснено системний аналіз викликів та перспектив застосування чат-ботів у процесах моніторингу, оперативної взаємодії та автоматизації внутрішніх бізнес-процесів.

Особливу цінність становить розроблена Telegram-система, яка поєднує функції моніторингу стану мобільних застосунків, оперативного інформування та оптимізації рутинних операцій. У роботі запропоновано підхід до інтеграції чат-ботів у загальну цифрову екосистему компанії, що дозволяє підвищити стабільність та якість прийняття управлінських рішень.

Мета дослідження полягає в обґрунтуванні теоретико-методичних засад цифрової трансформації ІТ-компаній та вивченні практичних аспектів застосування автоматизованих інструментів, зокрема чат-ботів, для підвищення ефективності управління бізнес-процесами і мобільними цифровими продуктами.

Для досягнення окресленої мети були визначені такі **завдання:**

- дослідити сутність поняття та еволюцію цифрової трансформації;
- розкрити роль автоматизації як чинника підвищення ефективності діяльності ІТ-компаній;
- ідентифікувати сучасні інструменти, технології та платформи автоматизації бізнес-процесів;
- проаналізувати функціональні можливості чат-ботів у контексті цифрових екосистем;
- визначити вимоги до створення автоматизованої системи моніторингу мобільних застосунків;
- розробити та реалізувати Telegram-бота як інструмент автоматизації операційних процесів;
- здійснити тестування розробленого рішення та оцінити його ефективність;
- виявити виклики, які супроводжують впровадження автоматизованих систем у сучасних компаніях;
- охарактеризувати перспективи розвитку технологій автоматизації у контексті глобальних цифрових змін та трансформації бізнес-моделей України.

Об’єктом дослідження є процеси захисту та управління життєвим циклом мобільних додатків у контексті впровадження інтегрованих систем адміністрування на базі веб-технологій та месенджер-платформ.

Предметом дослідження є методи підвищення захисту мобільних додатків за рахунок створення інтегрованої системи адміністрування з використанням PHP, MySQL та Telegram Bot API для автоматизації бізнес-процесів.

Матеріал дослідження. Під час написання магістерської роботи були використані наукові публікації у сфері цифрової трансформації, інформаційні ресурси провідних міжнародних організацій, аналітичні звіти щодо використання автоматизованих систем у бізнесі, електронні періодичні видання, а також офіційна документація технологічних платформ (Google, Meta, Telegram) та власні практичні розробки автора щодо створення Telegram-бота.

Методи дослідження включають системний підхід для аналізу процесів цифрової трансформації, порівняльний метод для оцінювання різних технологічних інструментів автоматизації, функціонально-структурний аналіз для визначення архітектури розробленої системи, статистичні методи для оцінки ефективності впровадженого інструмента, а також метод моделювання для формування концепції Telegram-бота як елементу цифрової екосистеми IT-компанії.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ ЗАХИСТУ МОБІЛЬНИХ ДОДАТКІВ ТА ІНТЕГРОВАНИХ СИСТЕМ АДМІНІСТРУВАННЯ

1.1. Сучасні підходи до забезпечення безпеки мобільних додатків

Швидке зростання мобільного ринку, поєднане з масштабним переходом бізнесу до цифрових екосистем, зумовило значне збільшення обсягів обробки конфіденційних даних за допомогою мобільних додатків. У результаті мобільні платформи перетворилися на один із ключових об’єктів інтересу зловмисників, які використовують широкий спектр методів для отримання несанкціонованого доступу, компрометації даних або порушення роботи програмних сервісів. Саме тому питання забезпечення безпеки мобільних додатків сьогодні є одним із центральних у процесі їх розробки, публікації та подальшої експлуатації.

Безпека мобільних продуктів не обмежується лише захистом вихідного коду чи інструментами шифрування на пристрої. Вона охоплює цілісний комплекс заходів, що включає захист з’єднань, структур бази даних, інтеграційних модулів, сторонніх SDK, API-комунікацій і серверної частини. В умовах поширеності хмарної інфраструктури, мікросервісних архітектур і активного використання мобільних додатків у фінансовому, медичному, логістичному та інших секторах, вимоги до захищеності систем стають дедалі жорсткішими [19].

Забезпечення стабільної роботи мобільних додатків пов’язане не лише із захистом користувачів та їхніх даних, але й із убезпеченням самого програмного продукту від блокувань, санкцій або обмежень з боку магазинів застосунків, інтеграційних сервісів та сторонніх API. У сучасному цифровому середовищі мобільний додаток функціонує в екосистемі численних правил, технічних вимог та політик, порушення яких може призвести до його видалення з маркетплейсів, відключення ключових функцій або навіть заборони на публікацію нових версій.

Одним із найважливіших факторів стабільної роботи мобільного ПЗ є дотримання вимог магазинів застосунків, таких як Google Play та Apple App Store, що відображені у табл.1.1.

Таблиця 1.1

Порівняльна таблиця вимог до мобільних додатків Google Play та Apple App Store

Категорія вимог	Google Play (Android)	Apple App Store (iOS)
Процес публікації	Менш суворий модераційний процес, автоматизовані перевірки займають 1–3 години, інколи до 24 годин.	Дуже сувора перевірка вручну, автоматичний аналіз коду. Час модерації: 24–48 годин.
Вимоги до контенту	Заборона шкідливих, оманливих, шахрайських матеріалів; обмеження для азартних ігор, крипти, фінтеху.	Більш жорстка політика щодо контенту, особливо в частині азартних ігор, VPN, скрин-ридерів, фінансових сервісів.
Обробка персональних даних	Обов’язковий Privacy Policy; дотримання GDPR; заборона невмотивованого збору даних.	Обов’язковий Privacy Policy та App Tracking Transparency (ATT), вимога чітко описувати, які дані збираються і навіщо.
Вимоги до UI/UX	Гнучкіші вимоги, основний акцент на стабільність та працездатність.	Дуже строгі вимоги до дизайну, логіки інтерфейсу, взаємодії та зручності. Відхиляють додатки за будь-яку неузгодженість.
Технічні вимоги	Додатки повинні відповідати останній версії SDK; необхідна підтримка нових API протягом 1 року після релізу.	Обов'язкова підтримка останніх версій iOS; нові фреймворки стають обов’язковими дуже швидко.
Безпека	Заборона використання застарілих API, обов’язкові перевірки Play Protect, вимога шифрування трафіку HTTPS.	Строгі вимоги до Code Signing, захисту від reverse engineering, використання HTTPS/TLS, заборона динамічного виконання коду.

Монетизація	Можливе використання сторонніх платіжних систем (з обмеженнями). Google Billing - рекомендований.	Усі внутрішні покупки *обов'язково* через Apple In-App Purchase (IAP). Сторонні Billing-сервіси заборонені.
Реклама	Дозволяє різноманітні рекламні SDK, але вимагає прозорості щодо трекінгу.	Дуже суворі вимоги до рекламних SDK; сторонні трекери можуть спричинити відмову у публікації.
Політика щодо оновлень	Оновлення проходять модерацію швидко - у межах кількох годин.	Оновлення проходять так само суворе рев'ю, як і перша версія; часто займає 1–2 доби.
Вимоги до стабільності	Висока кількість крашів - автоматичне зниження рейтингу і можливий бан.	Краші, нестабільність або некоректний інтерфейс - гарантована відмова у модерації.
Впровадження нового функціоналу	Гнучкі підходи - Google дозволяє експериментальний функціонал.	Будь-які нестандартні функції ретельно перевіряються; Apple часто вимагає додаткові пояснення.
Юридичні вимоги	Для окремих категорій (наприклад, азартні ігри) потрібні документи.	Більш суворі юридичні вимоги: ліцензії, сертифікати, докази права на контент.

Джерело: складено автором за [5].

Ці платформи постійно оновлюють свої політики, запроваджують нові стандарти безпеки, конфіденційності та UI/UX-вимоги. Відсутність своєчасної адаптації до таких змін може призвести до автоматичного бану, видалення додатка або блокування можливості оновлення. Найчастіше до санкцій призводять: використання застарілих SDK, некоректна робота з API, порушення політики конфіденційності, відсутність чіткого опису функціональності або вбудовані механізми збору даних без належного повідомлення користувача [15].

За останні роки Google Play суттєво скоротив кількість доступних додатків. Згідно з аналітикою Appfigures, на початку 2024 року в магазині було близько 3,4

млн застосунків, тоді як станом на середину року їх кількість зменшилась до приблизно 1,8 млн. Таким чином, обсяг контенту знизився майже на 47% [27].

При цьому аналогічної тенденції в Apple App Store не спостерігається: кількість додатків для iOS за той самий період навіть трохи зросла — із 1,6 млн до приблизно 1,64 млн.

Скорочення обсягу застосунків у Google Play значною мірою пов’язане зі змінами у політиках Google. Протягом багатьох років порівняно м’які вимоги до модерації призводили до появи у магазині великої кількості спам-додатків, генераторів контенту низької якості або програм із сумнівним функціоналом. У 2024 році компанія почала активніше усувати такі програми, щоб підвищити загальну якість екосистеми та полегшити користувачам пошук корисних додатків.

У липні 2024 року Google запровадив оновлені правила, які розширили перелік критеріїв для блокування додатків. Під заборону почали потрапляти не лише некоректно працюючі додатки, а й ті, що мають «обмежену функціональність або контент». До таких віднесли статичні програми без унікального функціоналу (наприклад, переглядачі PDF або додатки з одним типом контенту), а також додатки, які фактично не виконують корисних дій [48].

Паралельно Google посилив механізми перевірки розробників і впровадив [52]:

- розширені ручні перевірки на предмет шахрайства;
- обов’язкове тестування для нових акаунтів;
- додатковий аналіз застосунків за допомогою інструментів машинного навчання;
- жорсткіші вимоги до конфіденційності та обробки даних.

У результаті за 2024 рік Google заблокував понад 158 000 акаунтів розробників та запобіг публікації понад 2,36 млн додатків, що порушували політики платформи [14].

Додатково на ситуацію вплинуло нове правило ЄС щодо «статусу трейдера», яке зобов'язує розробників публікувати свої контактні дані у описі

застосунку. Невиконання цієї вимоги також призводило до видалення додатків у європейському регіоні [20].

Окремою проблемою є ризик потрапляння додатка під автоматичні фільтри маркетів, які розпізнають підозрілу активність. Сучасні системи машинного аналізу Google та Apple моніторять поведінку додатків, оцінюють частоту оновлень, аналізують краш-репорти, відстежують мережеву активність та інтерпретують аномальні патерни роботи як тіньовий функціонал. Наприклад, часті запити до зовнішніх API без чіткої причини або використання нестандартних бібліотек можуть спричинити автоматичну перевірку чи тимчасове призупинення роботи застосунку [13].

Важливою частиною захисту мобільного додатка є боротьба з баном акаунтів розробника. Це особливо актуально для команд, що працюють у комерційних нішах, пов'язаних із високими ризиками - наприклад, фінтех, геймінг, азартні ігри, VPN-сервіси, стрімінгові платформи [26].

Платформи можуть застосувати санкції не тільки до окремої програми, але й до всього видавничого профілю, особливо у випадках систематичних порушень політик або підозри у "пов'язаних" акаунтах. Задля запобіганню бану, потрібно дотримуватись правил зображених на рис.1.1

Рисунок 1.1 – Методи запобігання бану мобільних застосунків
Джерело: складено автором за [2].

Це особливо актуально для команд, що працюють у комерційних нішах, пов’язаних із високими ризиками - наприклад, фінтех, геймінг, азартні ігри, VPN-сервіси, стрімінгові платформи [26].

Не менш критичним є забезпечення стійкості до блокування інтеграційних сервісів. Багато мобільних застосунків залежать від сторонніх API: систем аналітики, платіжних шлюзів, антифрод-сервісів, S2S постбеків, рекламних платформ тощо. У разі порушення вимог щодо частоти запитів, автентифікації або умов використання API може бути заблоковано доступ до інструментів, що призведе до часткової або повної втрати функціональності додатка. Тому необхідно реалізовувати автоматизовані механізми контролю запитів, логування, керування ключами доступу та резервні схеми маршрутизації [81].

Важливим аспектом безпеки є й захист від reverse engineering та копіювання функціоналу. Зловмисники можуть модифікувати застосунок, імітувати його роботу або створити підроблену версію, яка негативно вплине на репутацію бренду та призведе до скарг користувачів. У результаті магазин може тимчасово обмежити або заблокувати оригінальний додаток через збільшення кількості негативних відгуків, крашів або підозрілої статистики. Для зниження таких ризиків застосовуються обфускація коду, перевірка цілісності APK/IPA-файлів, динамічний аналіз середовища виконання та механізми виявлення рутованих/джейлбрейк-пристроїв [66].

Проведений аналіз сучасних підходів до забезпечення безпеки мобільних додатків свідчить про те, що мобільна екосистема Android та iOS формує комплексний багаторівневий механізм захисту, який охоплює ізоляцію додатків, контроль дозволів, перевірку цілісності коду, шифрування даних та постійний моніторинг поведінки застосунків. Попри спільну мету, платформи реалізують ці механізми по-різному: Android робить акцент на гнучкості та відкритості системи, тоді як iOS застосовує більш сувиту політику доступу й контролю, забезпечуючи підвищений рівень захисту на рівні ОС.

Водночас сучасні тенденції, включно зі значним скороченням кількості доступних додатків у Google Play та посиленням політик модерації,

демонструють зміщення акценту на підвищення якості та безпечності контенту. Платформи переходять від базового технічного контролю до комплексного аналізу функціональності, контенту та моделей поведінки розробників. Це підтверджує необхідність дотримання нових стандартів безпеки, регулярного оновлення технологічної бази та забезпечення прозорості обробки даних.

Таким чином, безпека мобільних додатків розглядається не лише як технічне завдання, але як системний процес, що охоплює архітектурні рішення, політики конфіденційності, відповідність вимогам маркетів та правильну організацію життєвого циклу продукту. Розуміння цих принципів є фундаментом для подальшої розробки інтегрованої системи адміністрування, яка дозволить підвищити захищеність застосунків та забезпечити їх стабільну роботу в умовах динамічних вимог і ризиків.

1.2. Принципи управління життєвим циклом мобільних продуктів та їх захисту

Управління життєвим циклом мобільного додатка є ключовим елементом ефективної розробки, експлуатації та підтримки сучасних цифрових продуктів. Зі зростанням складності мобільних екосистем і підвищенням вимог до безпеки, продуктивності та якості програмного забезпечення, питання комплексного контролю кожного етапу життєвого циклу набувають особливого значення. Мобільний продукт перестає бути одноразовим програмним рішенням — він перетворюється на динамічний, постійно оновлюваний сервіс, який повинен стабільно функціонувати в умовах високої конкуренції та швидкої зміни технологічних вимог.

Життєвий цикл мобільного додатка охоплює весь спектр процесів — від формування початкової концепції та збору вимог до розробки, тестування, розгортання, підтримки, безпеки, моніторингу та подальшої модернізації продукту. У сучасній практиці ці процеси формалізуються в рамках моделей SDLC (Software Development Life Cycle) [94] та PDLC (Product Development Life

Cycle) [62], які дозволяють систематизувати всі етапи роботи над цифровим продуктом та забезпечують структурований підхід до його створення й супроводу та мають відмінності, що відображені у табл.1.2.

Таблиця 1.2

Таблиця порівняння моделей SDLC та PDLC

Критерій	**SDLC (Software Development Life Cycle)**	**PDLC (Product Development Life Cycle)**
Основна мета	Розробка програмного забезпечення як технічного продукту	Розвиток програмного продукту як бізнес-рішення
Фокус	Технічні етапи розробки	Стратегічний розвиток, маркетинг, аналітика
Етапи	Аналіз → Дизайн → Розробка → Тестування → Деплой → Підтримка	Ідея → Ринок → Планування → Розробка → Маркетинг → Підтримка → Монетизація
Орієнтація	Технічна команда (dev, QA, ops)	Бізнес + технічна команда
Вимірювання успіху	Якість та стабільність коду, відсутність багів	Успіх на ринку, KPI/ROI, залучення та утримання користувачів
Оновлення продукту	За потребою	Постійно згідно з ринковими циклами та поведінкою аудиторії
Захист і безпека	Орієнтовані на технічні ризики	Орієнтовані на бізнес-ризики + технічні
Підходить для	Технічно складних проектів	Комерційних цифрових рішень, мобільних додатків

Джерело: складено автором за [67].

Життєвий цикл розробки програмного забезпечення (SDLC) — це структурована модель створення цифрового продукту, яка включає послідовність етапів, необхідних для отримання якісного, стабільного та безпечного мобільного застосунку. Як і будь-який життєвий цикл, SDLC охоплює чітко визначені фази, у межах яких команда виконує певні завдання,

рухаючись від формування вимог до підтримки готового продукту. SDLC забезпечує системний та передбачуваний підхід до розробки програмного забезпечення, що дозволяє створювати якісні, ефективні та стійкі рішення незалежно від складності проєкту [92].

Ключовою метою SDLC є отримання високоякісного програмного продукту, який відповідає або перевищує очікування користувачів і стейкґолдерів, розробляється у визначені строки та в межах узгодженого бюджету. Завдяки системності цей підхід дозволяє контролювати якість, своєчасно виявляти ризики, уникати надмірних витрат і забезпечувати стабільність мобільних додатків на всіх етапах їхнього життєвого циклу [72].

Важливо розуміти, що SDLC не є жорсткою чи уніфікованою моделлю: її конкретні етапи, тривалість та інструменти залежать від масштабу, складності та призначення програмного продукту. Водночас SDLC виступає універсальною рамкою, яка впорядковує процес розробки й допомагає уникати хаотичних або непрозорих рішень [73].

Кожний етап життєвого циклу спрямований на отримання конкретного результату, який забезпечує відповідність мобільного додатка функціональним, технічним та бізнес-вимогам. Незалежно від того, чи йдеться про простий мобільний застосунок або комплексну enterprise-систему, головною метою SDLC є створення продукту, який відповідає очікуванням користувачів за якістю, продуктивністю та зручністю.

У межах SDLC команда виконує такі ключові процеси [71]:

- опрацювання вимог і формування user stories, що визначають очікувану поведінку продукту;
- створення прототипів та дизайн-макетів, які дозволяють перевірити гіпотези;
- написання програмного коду, що реалізує функціональну частину додатка;
- деплой продукту у робоче середовище, де ним можуть користуватися реальні користувачі.

Важливо розуміти, що SDLC не є жорсткою чи уніфікованою моделлю: її конкретні етапи, тривалість та інструменти залежать від масштабу, складності та призначення програмного продукту. Водночас SDLC виступає універсальною рамкою, яка впорядковує процес розробки й допомагає уникати хаотичних або непрозорих рішень та має важливе значення у розробці мобільних продуктів (див.табл.1.3).

Таблиця 1.3

Значення SDLC у процесі розробки мобільних продуктів

№	Аспект значущості SDLC	Сутність та вплив на розробку мобільних продуктів
1	**Підвищення якості програмного забезпечення**	Чітка структуризація процесів дозволяє виявляти та усувати помилки на ранніх етапах, що зменшує ризик критичних збоїв після публікації додатка.
2	**Зменшення витрат на розробку та підтримку**	Раннє виявлення помилок скорочує витрати на їхнє виправлення; SDLC знижує технічний борг і запобігає дорогим пост-релізним проблемам.
3	**Підвищення продуктивності команди**	Стандартизована структура процесів дозволяє ефективно розподіляти завдання, уникати дублювання роботи та зосереджуватися на пріоритетних функціях.
4	**Покращення комунікації між учасниками проєкту**	Етапи збору вимог та тестування забезпечують регулярний зворотний зв’язок, підвищують узгодженість дій між розробниками, замовниками та аналітиками.
5	**Відповідність стандартам та нормативам**	Завдяки документації та контролю SDLC забезпечує відповідність продукту вимогам безпеки, регуляціям та політикам мобільних платформ (Google Play, App Store).
6	**Полегшення супроводу та оновлення продукта**	Документування процесів спрощує подальшу підтримку, адаптацію до нових версій ОС та впровадження нового функціоналу.
7	**Підвищення задоволеності користувачів**	Завдяки системному підходу продукт має кращу стабільність, менше дефектів та краще відповідає очікуванням кінцевих користувачів.

Джерело: складено автором за [93].

Життєвий цикл розвитку продукту (Product Development Life Cycle, PDLC) — це концептуальна модель, яка описує всі основні стадії, через які проходить продукт від моменту появи ідеї до завершення його існування на ринку. У класичному вигляді PDLC включає такі фази, як розроблення нового продукту, його виведення на ринок, етап зростання, зрілості та подальшого спаду. Ця модель застосовується до будь-яких типів продуктів, зокрема й мобільних застосунків, IoT-рішень або цифрових сервісів [53].

Розуміння життєвого циклу продукту є критично важливим для компаній, які планують розробку та комерціалізацію нових рішень. PDLC допомагає ефективно організувати фінансове, управлінське та маркетингове планування, адже кожна фаза вимагає різних ресурсів, інструментів і стратегічних рішень. Чітке розуміння того, на якій стадії перебуває продукт, дозволяє оптимізувати бізнес-процеси та своєчасно реагувати на зміни ринку.

Початковим етапом PDLC є створення нового продукту. У цей період ідея перетворюється на повноцінну концепцію, а згодом — на корисне рішення, яке можна масштабувати та запускати у виробництво чи публікувати на ринку. Паралельно формується бізнес-модель, стратегія розвитку та план переходу до наступних етапів життєвого циклу. Для мобільних продуктів це включає визначення функціональності, створення прототипів та підготовку технічної реалізації [58].

Наступною фазою є виведення продукту на ринок. Цей етап передбачає активне представлення продукту аудиторії, формування бренду, початок комунікації з користувачами та побудову початкової клієнтської бази. На цьому етапі надзвичайно важливими є маркетинг, ціноутворення, вибір каналів дистрибуції та перші відгуки користувачів. Для мобільних застосунків це також включає ASO-оптимізацію, публікацію в магазинах додатків і технічну підготовку до масштабування [59].

Коли продукт успішно проходить етап упровадження, розпочинається фаза зростання. На цій стадії зростає кількість користувачів, підвищується попит, та з’являється можливість отримувати прибуток. Основними завданнями компанії

стають підтримка високої якості, швидке реагування на потреби нових користувачів, оптимізація функціоналу та формування лояльної аудиторії. З розвитком продукту з’являються перші конкуренти, що бачать потенціал ринку та прагнуть запропонувати власні альтернативи.

Фаза зрілості характеризується стабілізацією продукту на ринку, максимальними обсягами продажів і високим рівнем впізнаваності серед користувачів. На цьому етапі конкуренція посилюється, і компанії доводиться докладати значних зусиль, щоб зберегти свою позицію, утримати клієнтів і постійно оновлювати продукт. У сфері мобільних застосунків цей період є ключовим з точки зору оптимізації, підвищення безпеки, реалізації покращеного функціоналу та технологічних оновлень [80].

Завершальним етапом життєвого циклу є фаза спаду. Вона настає тоді, коли інтерес користувачів зменшується, продукт поступається новим технологіям або з’являються конкурентні рішення з більш широкими можливостями. Продажі скорочуються, знижується активність користувачів. Щоб уникнути стрімкого падіння, компанія може вдатися до модернізації продукту, випуску другої версії або трансформації концепції. Якщо ж продукт більше не відповідає цілям компанії чи не має потенціалу розвитку, приймається рішення про його зняття з ринку [61].

Варто підкреслити, що PDLC не є універсальною прогнозною моделлю. У реальних умовах продукт може затримуватися на певних стадіях або навіть не перейти до етапу спаду протягом тривалого часу. Проте практична цінність PDLC полягає в тому, що вона дозволяє бізнесу планувати розвиток, своєчасно адаптуватися до змін ринку та не припиняти інноваційної діяльності. Адже відмова від удосконалення продукту на етапі зрілості неминуче призводить до швидкого переходу в стадію спаду, де конкуренти можуть забрати ключову аудиторію.

Особливого значення набуває підхід DevSecOps, що інтегрує принципи безпеки на всіх етапах розробки, усуваючи розрив між командами розробки, операційної підтримки та інформаційної безпеки. На відміну від традиційних

моделей, де захист розглядався як завершальний етап, DevSecOps передбачає превентивне виявлення ризиків, автоматизацію перевірок, постійний аудит та безперервне вдосконалення процесів безпеки.

DevOps (Development + Operations) — це методологія організації процесів розробки та експлуатації програмного забезпечення, яка забезпечує тісну співпрацю між командами розробників та операційної підтримки. Основною метою DevOps є створення безперервного, автоматизованого та керованого циклу доставки програмного продукту, що дозволяє зменшити ризики, прискорити випуск оновлень та підвищити стабільність системи [17].

Для сучасних мобільних додатків DevOps є одним із ключових підходів, адже мобільні продукти потребують частих оновлень, стабільної роботи бекенду, високоавтоматизованих процесів тестування та швидкої реакції на інциденти. У середовищі, де Google Play і App Store встановлюють суворі вимоги до якості та безпеки, DevOps допомагає підтримувати конкурентоспроможність і відповідність стандартам.

У своїй основі DevOps ґрунтується на кількох фундаментальних принципах, які забезпечують цілісність та ефективність розробки й експлуатації програмного забезпечення. Одним із ключових елементів є безперервна інтеграція, що передбачає регулярне об’єднання змін у єдину кодову базу. Такий підхід дозволяє своєчасно виявляти помилки, уникати конфліктів версій і підтримувати стабільність проєкту під час його швидкого розвитку. Для мобільних застосунків це має особливе значення, оскільки забезпечує коректну збірку продукту для різних пристроїв та версій операційних систем [88].

Наступним важливим принципом є безперервна доставка та розгортання, які передбачають автоматизовану підготовку та випуск оновлень у робоче середовище. Це дозволяє скоротити час між створенням нових функцій і їхнім появленням у користувачів, забезпечити передбачуваність релізного процесу й зменшити ризик помилок під час публікації мобільних додатків на платформах Google Play та App Store. Автоматизація розгортання також усуває залежність від ручних операцій, що знижує ймовірність людських помилок [40].

Важливе місце в DevOps займає принцип «інфраструктура як код», згідно з яким сервери, мережеві ресурси та інші інфраструктурні компоненти описуються у вигляді програмного коду. Це дозволяє швидко розгортати стандартизовані середовища, масштабувати бекенд-продукти мобільних додатків та забезпечувати відтворюваність конфігурацій у будь-яких умовах. Такий підхід спрощує управління інфраструктурою, значно підвищує її надійність і дає можливість автоматично відновлювати роботу системи після збоїв.

Ще одним принципом є безперервний моніторинг та логування, які забезпечують постійний контроль за станом системи, продуктивністю серверів, стабільністю мобільного додатка, навантаженням і можливими інцидентами. Завдяки цьому команда отримує можливість оперативно реагувати на проблеми, проводити аналіз причин збоїв і забезпечувати високу якість продукту в реальних умовах. Моніторинг відіграє центральну роль у мобільній розробці, оскільки дозволяє контролювати поведінку додатка на різних пристроях та в умовах різного мережевого середовища.

Невід'ємною частиною DevOps є автоматизація тестування, що забезпечує прискорення перевірки якості програмного забезпечення. Автоматизовані тести дозволяють швидко оцінювати коректність роботи окремих модулів, перевіряти інтеграцію між компонентами та виявляти потенційні вразливості. Для мобільних застосунків автоматизація тестування має особливе значення, оскільки дає змогу перевіряти роботу додатка на різних пристроях, екранах, версіях операційних систем та конфігураціях [41].

У комплексі ці принципи формують цілісний підхід, який значно прискорює створення якісного мобільного програмного забезпечення, підвищує його безпеку, стабільність і дозволяє організувати ефективний процес доставки оновлень. DevOps забезпечує прозору взаємодію між командами, скорочує час від ідеї до реалізації та робить життєвий цикл мобільного продукту більш передбачуваним і керованим.

Отже, сучасні методології управління життєвим циклом програмних і мобільних продуктів забезпечують цілісний та системний підхід до їх створення, розвитку та підтримки. SDLC формує технічну послідовність розробки, PDLC визначає стратегічну еволюцію продукту на ринку, тоді як DevOps інтегрує процеси розробки й експлуатації, забезпечуючи швидкість, стабільність та автоматизацію. Сукупне використання цих підходів дозволяє підвищити якість мобільних додатків, скоротити час виходу оновлень, мінімізувати ризики та забезпечити їх довгострокову конкурентоспроможність. У результаті розробка стає більш передбачуваною, ефективною та орієнтованою на потреби користувачів і бізнесу.

1.3. Модульні архітектури та MVC-паттерн у розробці безпечних систем

Model-View-Controller (MVC) — це один із найбільш поширених архітектурних шаблонів у розробці програмного забезпечення, який забезпечує структуроване та логічне розділення компонентів системи. Його застосування дає змогу впорядкувати програмний код, підвищити масштабованість, забезпечити гнучкість під час внесення змін та спростити повторне використання модулів [18].

У веб- і мобільних додатках MVC відіграє ключову роль, оскільки дозволяє розробникам розділяти відповідальність між різними логічними частинами системи, що особливо важливо під час проєктування інтегрованих адміністративних панелей або серверної логіки (див. рис. 1.2).

Модель (Model) — відповідає за роботу з даними, бізнес-логіку, обробку інформації та взаємодію з базою даних.

Представлення (View) — забезпечує формування користувацького інтерфейсу та відображення даних, отриманих від моделі.

Контролер (Controller) — приймає та обробляє запити користувача, координує взаємодію між моделлю та представленням, а також визначає, які дані і в якому вигляді необхідно відобразити.

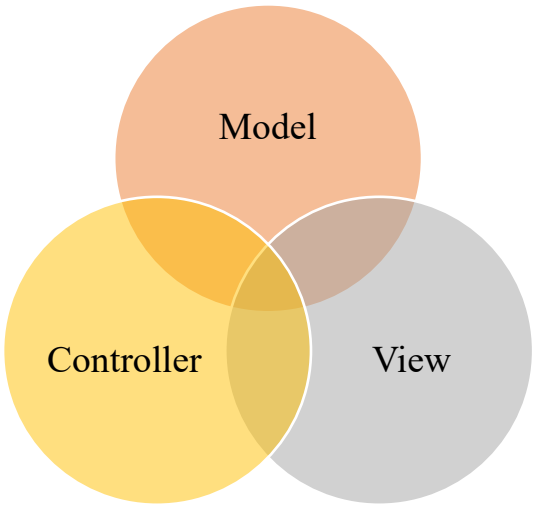


Рисунок 1.2 – Компоненти MVC
Джерело: складено автором за [89].

У межах такої системи важливо коректно організувати архітектуру відповідно до принципів MVC. Модель у цьому випадку відповідає за реалізацію бізнес-логіки, зокрема за обробку даних та взаємодію з базою даних. Контролер виконує роль посередника між користувачем і моделлю, отримуючи запити, інтерпретуючи їх і перетворюючи на виклики методів бізнес-логіки. Важливо, що один запит, опрацьований контролером, може ініціювати виконання декількох методів моделі [56].

Паралельно система має включати представлення (Views) — окремі сторінки або інтерфейсні шаблони, які відображають різні аспекти даних: каталог книг, картку певного товару, кошик, перелік замовлень тощо. Кожен елемент інтерфейсу фактично є окремим представленням, що відображає інформацію, отриману від моделі через контролер.

Основна мета архітектурного шаблону Model-View-Controller (MVC) полягає у відокремленні ключових складових програмного забезпечення — логіки введення, бізнес-логіки та користувацького інтерфейсу — з одночасним

забезпеченням слабкого зв’язку між цими компонентами. Такий поділ дозволяє розміщувати бізнес-логіку та операції з даними в моделі, інтерфейс і відображення інформації — у представленні, а обробку користувацьких запитів і маршрутизацію — у контролері. Завдяки цьому підходу розробник може концентруватися на окремих аспектах системи, що суттєво спрощує роботу зі складними застосунками [57].

Використання MVC має низку переваг, які роблять цей шаблон одним із найефективніших у проєктуванні масштабованих та підтримуваних систем. Однією з ключових переваг є можливість паралельної розробки: оскільки модель, представлення та контролер функціонують незалежно один від одного, різні члени команди можуть займатися своїми частинами проєкту, не блокуючи роботу інших. Це прискорює процес розробки та підвищує ефективність командної взаємодії.

Ще однією важливою характеристикою є повторне використання компонентів. Оскільки представлення відповідає лише за відображення даних, його можна адаптувати або використовувати в інших проєктах, змінюючи лише модель, яка постачає дані. Це не лише скорочує витрати на розробку, а й сприяє створенню уніфікованих і підтримуваних інтерфейсів.

Архітектура MVC також забезпечує хорошу масштабованість. У разі зростання навантаження або зниження продуктивності певного шару системи, наприклад бази даних, можна модернізувати лише відповідний компонент або змінити його інфраструктуру, не зачіпаючи інші частини застосунку. Слабка зв’язаність між компонентами дає змогу гнучко розвивати систему та адаптувати її до нових вимог [55].

Важливою перевагою є й підвищена розширюваність. Оскільки логіка поділена, внесення змін до одного з компонентів — для виправлення помилок, оптимізації або додавання нових функцій — не впливає на роботу інших частин. Це значно полегшує підтримку великих довгострокових проєктів і дозволяє швидко впроваджувати нові можливості [90].

Припустімо, розробнику необхідно створити вебзастосунок інтернет-магазину, що спеціалізуватиметься на продажу книг. Користувачі повинні мати змогу переглядати доступні видання, реєструватися, формувати замовлення, додавати товари у кошик, зберігати вподобані книги та здійснювати покупки.

Користувач натискає посилання на розділ рекомендацій, після чого браузер надсилає запит на адресу на кшталт */books/recommendations*. Контролер отримує цей запит і насамперед перевіряє, чи авторизований користувач, або чи існує механізм формування рекомендацій для гостей. Далі контролер звертається до моделі, запитуючи перелік книг, які можуть бути цікавими конкретному користувачу.

Модель, у свою чергу, виконує звернення до бази даних, аналізує різні джерела інформації: популярні видання, історію покупок користувача, вподобання його друзів або позиції з wish-list. На основі цих даних модель формує добірку з рекомендованих книг і повертає її контролеру.

Після отримання результату контролер оцінює отриманий список та ухвалює рішення щодо подальших дій. Якщо рекомендацій замало або їх немає, контролер може запросити універсальний список для неавторизованих відвідувачів. Якщо в магазині діє акція, контролер може доповнити список акційними товарами.

Далі контролер визначає, яке саме представлення потрібно показати користувачеві: сторінку з рекомендаціями, повідомлення про відсутність книг чи іншу відповідну сторінку. Після цього сервер формує потрібний шаблон, заповнює його даними — ім’ям користувача, списком книг тощо — і повертає клієнтові, де сторінка відображається у браузері.

Переваги такого підходу є очевидними. Насамперед це чіткий поділ між логікою відображення та логікою роботи застосунку. Окрім того, розділення серверної частини на модель і контролер робить систему значно гнучкішою. Наприклад, рішення, які дані повернути у випадку порожнього списку рекомендацій або під час проведення акцій, можуть визначатися контролером, не ускладнюючи логіку моделі. Це спрощує підтримку, дозволяє адаптувати

поведінку системи до нових вимог (наприклад, перегляд сторінки в режимі адміністратора без акцій чи з тестовою акцією) та запобігає надмірній концентрації логіки в одному компоненті.

1.4. Методи та засоби захисту веб-панелей адміністрування

У сучасних інформаційних системах веб-панелі адміністрування відіграють ключову роль, оскільки забезпечують централізований доступ до управління мобільними продуктами, конфігураціями серверних модулів та внутрішніми бізнес-процесами організації. Разом з тим саме адміністративні інтерфейси залишаються одними з найбільш вразливих компонентів програмних систем, оскільки містять підвищений рівень доступу та оперують чутливими даними. Будь-який успішний напад на адміністративну панель може призвести до витоку інформації, втрати керування продуктом, компрометації сервісів або повного зупинення роботи застосунку.

Тому питання забезпечення захисту веб-панелей набуває першочергового значення, особливо у контексті інтегрованих систем адміністрування мобільних додатків, які взаємодіють із зовнішніми API, Telegram-ботами, базами даних та загальнодоступними веб-ресурсами. Ефективна система захисту повинна враховувати комплекс загроз, таких як атаки на рівні автентифікації, спроби підбору паролів (brute-force), виконання несанкціонованих SQL-запитів, міжсайтові атаки CSRF, перехоплення даних у мережі тощо [35].

У цьому підпункті розглянуто основні методи і засоби захисту веб-панелей адміністрування, зокрема механізми протидії SQL-ін’єкціям, CSRF-атакам, списків контролю доступу (ACL). Детальний аналіз цих методів дозволить сформувати цілісне бачення щодо побудови безпечної інфраструктури для управління мобільними продуктами та мінімізації ризиків, пов’язаних із несанкціонованим доступом або кібератаками.

Атаки CSRF (Cross-Site Request Forgery) — це тип вебуразливостей, за якого зловмисник може змусити браузер користувача виконати небажану дію від

його імені. У літературі CSRF також відома під назвами XSRF, *sea surf*, *session riding* або *one-click attack*, що підкреслює її здатність використовувати авторизовані сесії без відома користувача [37].

Більшість вебзастосунків використовують механізми автентифікації, які дозволяють виконувати захищені та потенційно критичні операції — наприклад, змінювати налаштування, проводити фінансові транзакції або керувати ресурсами. Найчастіше автентифікація реалізується через сесійні cookies. Після успішного входу браузер зберігає cookie-файл та автоматично додає його до кожного подальшого запиту до вебзастосунку [30].

Рідше для автентифікації застосовуються інші механізми — Basic Auth, NTLM або ідентифікація користувача за IP-адресою. Однак у всіх випадках браузер автоматично додає відповідні дані автентифікації до запитів [29].

Варто врахувати, що браузер надсилає HTTP-запити не лише тоді, коли користувач натискає посилання або вводить URL. Частина запитів генерується автоматично під час завантаження сторінки — наприклад, для завантаження зображень, стилів чи скриптів. Такі запити можуть надсилатися на інші домени, відмінні від сайту, який переглядає користувач. Якщо ж вебзастосунок на сторонньому домені використовує ті самі механізми автентифікації (наприклад, cookie), він сприйматиме такого запитувача як автентифікованого. Типова схема виглядає так як описано нижче. [6]

1. Користувач авторизується в певному вебзастосунку (наприклад, *example.com*), і браузер зберігає cookie для автентифікації.
2. Зловмисник переконує користувача перейти на шкідливий сайт (наприклад, *malicious-site.com*) — через електронний лист, посилання у соцмережі тощо.
3. На шкідливій сторінці розміщено код (часто простий тег <img> або прихована форма), який змушує браузер автоматично надіслати запит на *example.com*.
4. Браузер додає до запиту збережені cookie, і сервер інтерпретує цей запит як дійсний та виконаний дозволеним користувачем.

5. У результаті у вебзастосунку можуть бути проведені небажані дії — зміна паролю, переказ коштів, видалення ресурсу, активація небезпечної функції тощо.

Важливо зазначити, що CSRF раніше була окремим пунктом у рейтингу OWASP Top 10 (наприклад, у категорії A5:2013). Однак через розвиток кращих механізмів захисту та поступове зменшення частоти таких атак у сучасних застосунках, починаючи з 2017 року, OWASP інтегрував CSRF до ширшої групи проблем, пов’язаних з недостатньою валідацією та авторизацією запитів [3].

Найбільш ефективним та загальноприйнятим механізмом протидії CSRF-атакам є використання шаблону синхронізатора токенів. Суть цього підходу полягає у тому, що для кожного HTTP-запиту, який змінює стан системи, сервер вимагає наявності спеціального випадково згенерованого значення — CSRF-токена. Це значення передається в межах форми або окремого заголовка запиту, тобто в тій частині HTTP-комунікації, яка не додається браузером автоматично [78].

При надходженні запиту сервер отримує надісланий CSRF-токен і порівнює його з очікуваним значенням, що зберігається на стороні сервера (наприклад, у сесії користувача). Якщо токен у запиті відсутній, неправильний або не збігається з очікуваним, запит має бути відхилений як потенційно небезпечний. Таким чином, шкідлива сторінка не може виконати CSRF-атаку, оскільки не має можливості отримати або передати правильний токен [82].

Важливо підкреслити, що CSRF-токен не може зберігатися у cookie, оскільки браузер надсилає cookies автоматично при кожному запиті, незалежно від того, з якої сторінки вони були ініційовані. Використання токену у cookie не забезпечує жодного додаткового захисту, а тому є неправильним рішенням.

Для підвищення зручності користувача та зменшення навантаження на розробку токени необхідно вимагати лише для запитів, які змінюють стан програми (наприклад, POST, PUT, DELETE). Запити, що не змінюють дані (GET), можуть залишатися без CSRF-токена, але за умови, що вони дійсно є ідемпотентними, тобто не впливають на стан системи. Це також запобігає

випадковому витоку токена через URL-параметри, які можуть потрапити у журнали вебсерверів або кеш браузера [51].

Використання шаблону синхронізатора токенів забезпечує надійний рівень захисту веб-панелей адміністрування, оскільки унеможливлює виконання несанкціонованих дій від імені користувача з іншого домену, навіть якщо його сесія залишається активною.

Атаки типу CSRF (Cross-Site Request Forgery) залишаються однією з актуальних загроз для сучасних вебзастосунків. Попри зростання рівня безпеки та появу нових механізмів захисту, CSRF усе ще активно використовується зловмисниками, оскільки значна частина систем покладається на cookie-автентифікацію та часто не впроваджує додаткових рівнів перевірки запитів. Особливо вразливими є сервіси з високою інтенсивністю користувацької взаємодії — платформи електронної комерції, банківські та фінансові застосунки, системи охорони здоров'я й соціальні мережі. Для цих секторів CSRF становить не лише кіберзагрозу, але й суттєвий економічний ризик [36].

Актуальні дані свідчать, що у 2023 році CSRF-атаки становили приблизно 15% усіх атак на вебзастосунки. Особливо помітним є збільшення кількості інцидентів у галузях, що працюють із фінансовими даними або персональною інформацією користувачів. Зокрема [3]:

- у сфері електронної комерції зафіксовано зростання CSRF-атак на 20%;
- у фінансовому секторі кількість витоків даних, спричинених CSRF, підвищилась на 12%;
- у мобільних застосунках за останній рік вразливості типу CSRF збільшилися на 18%;
- середня економічна шкода від CSRF-інцидентів зросла приблизно на 10% у порівнянні з попереднім періодом.

Найчастіше під ударом перебувають сектори, що оперують конфіденційною або чутливою інформацією: фінансові установи, онлайн-торгівля та медичні сервіси (див. рис. 1.3). Це підтверджує необхідність

активного впровадження механізмів протидії CSRF на всіх рівнях — від архітектурного проєктування до регулярного аудиту безпеки.

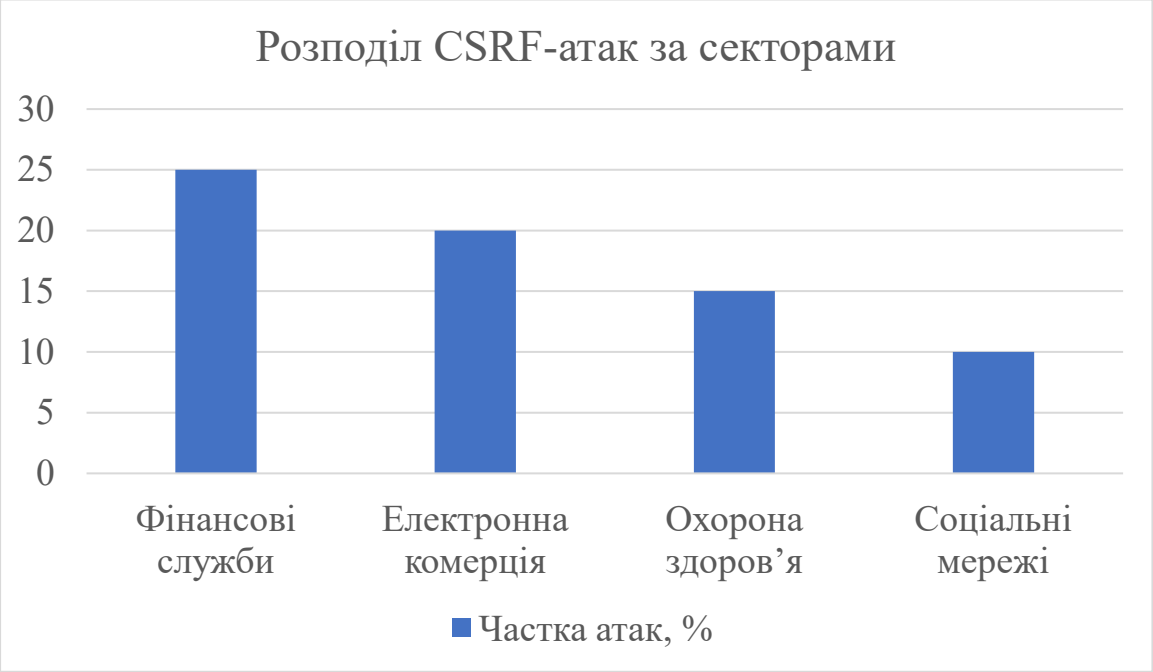


Рисунок 1.3 – Розподіл CSRF-атак за секторами
Джерело: складено автором за [3].

Звітність з кібербезпеки підтверджує, що CSRF-атаки постійно еволюціонують, і з'являються нові модифікації, які обходять традиційні механізми захисту. У зв'язку з цим компанії змушені регулярно оновлювати стратегії безпеки, впроваджувати патчі та проводити pen-тестування. Надійними методами протидії залишаються шаблон синхронізатора токенів (Synchronizer Token Pattern), механізм «подвійної відправки cookie» (Double Submit Cookie), а також багаторівнева авторизація та перевірка походження запитів [38].

Проактивний підхід до оцінки вразливостей, регулярне тестування та підвищення обізнаності користувачів дають змогу суттєво зменшити ризики, пов'язані з CSRF, і мінімізувати можливий вплив таких атак на критичні системи.

SQL-ін'єкція (SQLi) — це одна з найнебезпечніших вебуразливостей, яка дозволяє зловмиснику втручатися в SQL-запити, що виконуються вебзастосунком під час взаємодії з базою даних. Експлуатація такої вразливості

дає можливість отримати доступ до інформації, яку користувач або навіть сама система не повинні розкривати. Йдеться про дані інших користувачів, службові записи або будь-яку інформацію, до якої застосунок має прямий доступ. У більш серйозних випадках атакувальник може змінювати або видаляти дані, що спричиняє постійні порушення роботи системи та модифікацію її поведінки [77].

У деяких сценаріях SQL-ін’єкцію можна використати для глибшого проникнення у внутрішню інфраструктуру — наприклад, для отримання контролю над сервером або іншими компонентами бекенду. Уразливість також може застосовуватися для проведення атак типу відмови в обслуговуванні (DoS), коли сервер перевантажується надмірно складними або тривалими SQL-операціями [76].

Реалізація SQL-ін’єкції може призвести до несанкціонованого доступу до конфіденційної інформації, такої як [69]:

- облікові дані користувачів (паролі, хеші, токени);
- реквізити платіжних карт;
- персональна інформація (PII).

SQL-атаки неодноразово ставали причиною масштабних витоків даних, які завдавали організаціям значних фінансових збитків, втрат ділової репутації та юридичних санкцій. У деяких випадках зловмисники створювали приховані бекдори, що дозволяло їм тривалий час непомітно втручатися в роботу системи та підтримувати контроль над інфраструктурою.

Виявити SQL-вразливість можна вручну шляхом послідовного тестування всіх точок введення даних у додатку. Найпоширеніші техніки ручного аналізу включають [4]:

- введення одинарної лапки ' для виявлення синтаксичних помилок, що вказують на некоректну обробку введення;
- використання SQL-конструкцій, які повертають базове (очікуване) значення та альтернативне значення, й аналіз різниці у відповідях;
- тестові булеві умови (наприклад, OR 1=1 та OR 1=2) для перевірки наявності змін у відповіді застосунку;

- введення пейлоадів із затримкою виконання (наприклад, SLEEP(5)), щоб відстежити зміну часу відповіді — це свідчить про виконання SQL-команди;
- використання OAST-пейлоадів, що генерують зовнішню мережеву активність при виконанні в SQL-контексті, що дозволяє виявити уразливість за допомогою моніторингу сторонніх взаємодій.

У практичних умовах більшість SQL-ін’єкцій можуть бути знайдені автоматизованими інструментами. Зокрема, комплексні сканери, такі як Burp Scanner, здатні швидко та ефективно визначати небезпечні точки введення і проводити детальний аналіз поведінки застосунку [32].

SQL-ін’єкція — це техніка впровадження шкідливого коду, яка використовується проти вебзастосунків, що працюють з базою даних. Суть атаки полягає в тому, що зловмисник надсилає спеціально сформований SQL-запит, який змушує сервер виконати небажані дії: отримати конфіденційні дані, змінити їх, додати нові записи або навіть видалити існуючі [74].

Щоб зрозуміти механізм атаки, уявімо просту ситуацію, що вебпанель адміністрування має форму авторизації, де адміністратор вводить логін:

```
https://example-admin.com/login?user=admin
```

Рисунок 1.4 – Форма атворизації

У фоні надсилається SQL-запит:

```
SELECT * FROM admins WHERE username = 'admin'
```

Рисунок 1.5 – SQL-запит

Зловмисник замість звичайного логіна вводить:

```
admin' OR '1'='1
```

Рисунок 1.6 – Хибне знаечння логіна

Тоді запит перетворюється на:

```
SELECT * FROM admins WHERE username = 'admin' OR '1'='1'
```

Рисунок 1.7 – Новий запит

Умова '1'='1' завжди істинна, тому база поверне перший доступний запис, фактично дозволивши увійти без пароля.

Ще небезпечніше, якщо в полі пошуку або у формі додавання даних зловмисник спробує виконати UNION-ін’єкцію:

```
admin' UNION SELECT email, password FROM users --
```

Рисунок 1.8 – UNION-ін’єкція

Увідповідь вебпанель може показати:

- електронні адреси користувачів,
- хешовані паролі,
- інші приватні дані, що зберігаються у таблиці *users*.

SQL-ін’єкції й досі залишаються однією з найрозповсюдженіших загроз, головним чином через простоту їх виявлення й експлуатації. Атакувальнику достатньо знайти будь-який параметр вебзапиту — поле пошуку, форму авторизації, фільтри, навіть URL-параметри — який безпосередньо вставляється у SQL-запит без належної фільтрації. Така вразливість миттєво стає потенційною точкою проникнення.

Проблема ускладнюється тим, що практично кожен сайт або мобільний застосунок працює з базою даних. Це означає, що будь-який ресурс, який не використовує сучасні механізми захисту, автоматично є мішенню для SQL-атаки.

Історично багато популярних SQL-клієнтів, ORM-бібліотек та фреймворків також були вразливими до подібних атак, що робило SQLi найбільш очевидним і простим вектором атаки для кіберзлочинців.

Якщо застосунок використовує параметризований запит, і зловмисник введе: ' OR 1=1, то то база даних не сприйме це як логічну умову. Запит шукатиме

буквально рядок: ' OR 1=1. Тобто ін’єкція втрачає силу, і структура SQL-команди залишається незмінною [75].

SQL-ін’єкції залишаються однією з найнебезпечніших та найпоширеніших вразливостей веб-систем через простоту їх виявлення та високу результативність при експлуатації. Будь-який застосунок, що взаємодіє з базою даних та некоректно обробляє користувацьке введення, автоматично потрапляє до групи ризику. Водночас правильне застосування параметризованих запитів, підготовлених виразів, ORM-механізмів та принципів безпечної розробки практично повністю усуває можливість ін’єкції.

Таким чином, SQLi є проблемою не технологічною, а дисциплінарною: її успішне усунення залежить від дотримання розробниками базових практик безпеки та регулярного проведення аудитів коду.

Після аналізу вразливостей, пов’язаних із підробкою міжсайтових запитів та маніпулюванням SQL-запитами, стає очевидним, що значна частина атак успішна через відсутність або неналежне налаштування механізмів контролю доступу. Навіть за умови наявності захисту від CSRF та SQL-ін’єкцій, застосунок залишається вразливим, якщо зловмисник може отримати доступ до функцій чи даних, що не призначені для його ролі. Саме тому наступним критично важливим аспектом безпеки веб-панелей є реалізація чіткої та ієрархічної моделі контролю доступу (ACL), яка визначає, хто й до яких ресурсів може отримувати доступ, а які дії залишаються заблокованими.

Списки контролю доступу (ACL, Access Control List) є одним із ключових механізмів забезпечення інформаційної безпеки в сучасних комп’ютерних системах та мережах. Їхнє основне призначення полягає у формуванні чітких правил, які визначають, хто саме та за яких умов може отримати доступ до певних ресурсів, даних або функцій системи. Використання ACL дозволяє адміністраторам ефективно обмежувати або надавати доступ, запобігаючи несанкціонованому втручанню, зловживанню правами чи небажаним змінам у критично важливих компонентах інфраструктури [11].

ACL широко застосовується у корпоративних середовищах, де безпека даних та контроль доступу мають особливе значення. У великих організаціях, де одночасно працюють сотні або тисячі користувачів із різними ролями, коректно налаштована модель контролю доступу є фундаментом захисту внутрішніх сервісів, веб-панелей адміністрування та бізнес-критичних систем.

Механізм ACL підтримується більшістю сучасних операційних систем, включно з Windows, Linux та macOS, що дозволяє гнучко налаштовувати доступ до файлів, каталогів, сервісів або мережевих інтерфейсів. Крім того, ACL інтегровані у мережеве обладнання — маршрутизатори, комутатори, точки доступу, — де вони визначають правила фільтрації трафіку та обмеження взаємодії між різними сегментами мережі [25].

Списки контролю доступу (ACL) дозволяють точно визначати, які користувачі чи групи користувачів можуть взаємодіяти з певними ресурсами системи, а також встановлювати рівень доступу — читання, запис, виконання, видалення тощо. Такий механізм забезпечує гнучке керування правами й дозволяє створювати багаторівневі моделі безпеки.

Сфери застосування ACL є доволі широкими та охоплюють як локальні системи, так і розподілені інфраструктури (табл.1.4.). Щоб наочно продемонструвати різноманітність середовищ, у яких використовується механізм списків контролю доступу, у таблиці нижче наведено узагальнену класифікацію основних напрямів їхнього застосування [86].

Однією з ключових характеристик ACL є гранулярність, тобто ступінь деталізації, з якою можна встановлювати права доступу. Від рівня гранулярності залежить точність контролю над системою: чим вона вища, тим більш детально можна налаштувати доступ, але водночас зростає складність адміністрування та ризик помилок (див. рис. 1.5.).

Виділяють кілька рівнів гранулярності ACL [85].

Гранулярність на рівні об’єкта. Права доступу встановлюються безпосередньо для конкретних ресурсів — файлів, каталогів, таблиць бази даних.

Таблиця 1.4

Основні сфери використання ACL

Сфера застосування	Короткий опис використання ACL
Корпоративні мережі	Використовуються для розмежування доступу співробітників та груп до внутрішніх ресурсів. За допомогою ACL адміністратори можуть регулювати можливість роботи з файлами, папками, мережевими пристроями, поштовими сервісами чи вебресурсами.
Системи керування базами даних (СКБД)	ACL визначають права доступу до елементів бази — таблиць, представлень, процедур, тригерів. Це дозволяє контролювати, хто може читати, змінювати чи виконувати певні об'єкти БД.
Мережеве обладнання	На маршрутизаторах і комутаторах ACL застосовуються для обмеження доступу до портів, протоколів і сервісів, а також для фільтрації небажаного або небезпечного трафіку.
Хмарні сервіси	У хмарних платформах ACL керують дозволами на роботу з ресурсами: сховищами даних, контейнерами, віртуальними машинами, функціями й мікросервісами. Забезпечують сегментацію доступу між користувачами та застосунками.
Мобільні пристрої	На смартфонах і планшетах списки доступу регулюють, які програми або системні процеси можуть взаємодіяти з файлами, сенсорами чи іншими ресурсами пристрою.
Операційні системи (Windows, Linux, macOS)	ACL реалізують управління доступом до локальних ресурсів: файлів, директорій, пристроїв, системних служб. Дозволяють визначати точні права читання, запису та виконання для користувачів і груп.

Джерело: складено автором за [24].

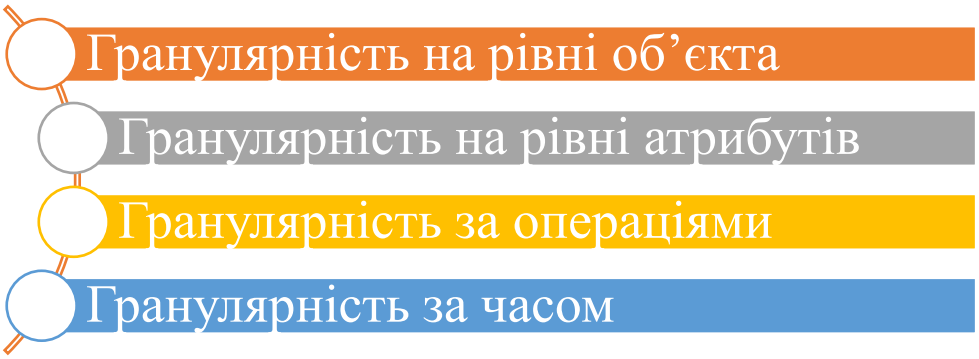


Рисунок 1.5 – Рівні гранулярності ACL

Джерело: складено автором за [21].

Наприклад, один і той самий файл може бути доступним для читання для однієї групи користувачів і повністю закритим для іншої.

Гранулярність на рівні атрибутів. У цьому випадку доступ визначається за характеристиками об’єкта: типом даних, місцем розташування, рівнем конфіденційності. Так, конфіденційні документи можуть бути доступні лише адміністраторам, тоді як звичайні файли — усім співробітникам.

Гранулярність за операціями. Користувачам або групам можуть надаватися права на виконання окремих дій — читання, редагування, створення, видалення. Наприклад, співробітники можуть мати право на читання документа, але лише керівник — на внесення змін.

Гранулярність за часом. Доступ до ресурсів може обмежуватися часовими рамками — певними годинами, днями тижня або періодами активності системи. Це корисно для високозахищених систем, де робота поза робочим часом небажана.

Підвищення рівня гранулярності дозволяє впроваджувати точні та ефективні механізми контролю доступу, але водночас вимагає ретельного планування та правильного адміністрування, щоб уникнути надмірної складності та помилкових налаштувань.

Існує кілька основних типів списків контролю доступу (ACL), які застосовуються для регулювання доступу до ресурсів у комп’ютерних системах та мережах (табл.1.5). Кожен із підходів відрізняється логікою прийняття рішень та рівнем контролю над тим, хто і за яких умов може виконувати певні дії з даними або об’єктами.

Захист веб-панелей адміністрування є ключовим елементом безпеки будь-якої цифрової інфраструктури, адже саме ці інтерфейси забезпечують доступ до критичних даних, налаштувань системи та механізмів керування бізнес-процесами. Розглянуті методи — захист від CSRF-атак, запобігання SQL-ін’єкціям, протидія brute-force атакам, використання токенів, шифрування трафіку через HTTPS та впровадження ACL — формують комплексний багаторівневий підхід до безпеки.

Таблиця 1.5

Основні види ACL і їх опис

Тип контролю доступу	Сутність підходу	Приклад застосування
DAC (Discretionary Access Control)	Власник ресурсу сам визначає, кому і які дії дозволені.	Користувач створює файл та вручну надає доступ лише окремим співробітникам.
MAC (Mandatory Access Control)	Права доступу визначаються централізованою політикою безпеки, а не власником ресурсу.	Державні інформаційні системи, де документи з грифом «Таємно» недоступні користувачам без певного рівня допуску.
RBAC (Role-Based Access Control)	Доступ встановлюється відповідно до ролей користувача в системі.	Менеджери мають доступ до фінансових звітів, а звичайні працівники — ні.
ABAC (Attribute-Based Access Control)	Рішення приймаються на основі атрибутів користувача, ресурсу чи середовища.	Користувач може змінювати дані лише у межах свого відділу та лише в робочий час.
PBAC (Policy-Based Access Control)	Доступ визначається набором політик безпеки, що враховують різні параметри запиту.	Доступ дозволено тільки якщо запит надходить з корпоративної мережі та відповідає політиці автентифікації.

Джерело: *складено автором за [22].*

Кожен із цих механізмів виконує власну роль: одні забезпечують цілісність запитів та даних, інші контролюють автентифікацію й авторизацію користувачів, а окремі — захищають канали зв’язку та зменшують ризики компрометації акаунтів. У сукупності вони створюють надійний захисний контур, що значно ускладнює можливість несанкціонованого доступу або маніпулювання системою.

Таким чином, ефективне впровадження розглянутих методів дозволяє підвищити рівень захищеності веб-панелей адміністрування, мінімізувати потенційні вразливості та забезпечити стабільну роботу інтегрованих систем адміністрування мобільних додатків.

1.5. Огляд існуючих інструментів для автоматизації адміністрування мобільних продуктів

У сучасній цифровій екосистемі мобільні додатки стали однією з ключових складових діяльності комерційних компаній, державних установ, стартапів і технологічних підприємств. Висока конкуренція, швидкі оновлення мобільних операційних систем, зростаючі вимоги користувачів до стабільності та функціональності продуктів зумовлюють необхідність застосування інструментів, що забезпечують автоматизацію адміністративних процесів, моніторинг роботи додатків та оптимізацію життєвого циклу продукту. Від ефективності цих інструментів залежить швидкість випуску оновлень, стабільність роботи мобільного застосунку, рівень безпеки, відповідність вимогам маркетплейсів та здатність компанії оперативно реагувати на інциденти.

Сучасний ринок пропонує широкий спектр платформ, які допомагають розробникам управляти мобільними додатками впродовж усього життєвого циклу: від етапу їх створення до розгортання, тестування, публікації, аналітики й підтримки. Такі інструменти дозволяють автоматизувати операції, що раніше виконувались лише вручну: збирання статистики використання, відстеження стабільності, управління оновленнями, обробку краш-логів, роботу з користувацькими відгуками, конфігураціями та маркетинговими параметрами.

Одними з найвідоміших екосистем, що забезпечують повний спектр сервісів для адміністрування мобільних продуктів, є Google Firebase Console, Huawei AppGallery Connect та Apple App Store Connect. Ці платформи об’єднують у собі інструменти аналітики, тестування, управління релізами, відстеження продуктивності та роботи з користувачами. Вони є невід’ємною частиною архітектури мобільних додатків, оскільки виступають центральними консолями керування доступом, конфігураціями, комунікаціями з користувачами та процесами публікації в маркетплейсах [23].

Проте, крім офіційних інструментів маркетплейсів, активно набирають популярність сторонні системи адміністрування (third-party dashboards), які розширюють функціональність стандартних платформ і дозволяють автоматизувати задачі, специфічні для певного бізнесу. До таких систем належать багатофункціональні панелі, аналітичні платформи, системи crash-моніторингу, сервіси маркетингової атрибуції, внутрішні корпоративні панелі керування. Вони відіграють важливу роль у компаніях, де необхідно об’єднати дані з різних джерел — Firebase, App Store Connect, AppGallery Connect, Google Analytics, рекламних кабінетів та ін [50].

Зростання складності мобільних продуктів створює потребу у централізованих адміністративних середовищах, де розробники, аналітики, менеджери з маркетингу та DevOps-інженери можуть працювати з узагальненою інформацією. Саме тому сьогодні активно використовуються інструменти інтеграції — API, webhook-сервіси, S2S-взаємодія та автоматизовані пайплайни доставки (CI/CD), що дають змогу поєднувати розрізнені дані та автоматизувати частину процесів, які раніше вимагали значних часових затрат [8].

Кожна з офіційних платформ має власну логіку роботи та специфічний набір функцій [39].

Firebase Console надає потужні інструменти для crash-моніторингу, аналітики, хостингу, автоматичних оновлень конфігурацій, ролей доступу, push-повідомлень і A/B-тестування.

AppGallery Connect пропонує унікальні можливості для роботи в екосистемі Huawei, включаючи сервіси монетизації, аналітики, методу сінхронізації з додатками та окремі функції для тестування.

App Store Connect забезпечує адміністрування додатків для iOS: управління збірками, роботу з відгуками, TestFlight, аналітику, фінансову інформацію та процеси модерації.

Сторонні панелі адміністрування створюють додатковий рівень автоматизації — об’єднання статистики, інтеграція маркетингової атрибуції,

керування доступом, централізація S2S-запитів і кастомні модулі, які розширюють можливості стандартних інструментів.

У сучасних умовах автоматизація адміністративних процесів дозволяє суттєво зменшити операційні витрати, пришвидшити цикли публікації продуктів, підвищити прозорість роботи команд і забезпечити високий рівень безпеки мобільного додатку. Саме тому огляд таких інструментів є критично важливим для розуміння архітектури сучасних систем адміністрування та для вибору оптимальних засобів у межах конкретного проєкту.

Firebase — це хмарна платформа, створена компанією Google, яка працює за моделлю *Backend-as-a-Service (BaaS)*. Вона пропонує розробникам широкий набір інструментів і сервісів для побудови, розгортання та підтримки серверної частини мобільних і вебзастосунків без необхідності самостійно керувати інфраструктурою. Firebase забезпечує готові рішення для зберігання даних, автентифікації користувачів, надсилання push-сповіщень, хостингу та аналітики, що істотно спрощує процес розробки і зменшує обсяг рутинних операцій [45].

Однією з головних переваг платформи є простота інтеграції та швидкість налаштування. Розробники мають змогу додати підтримку Firebase до мобільного застосунку протягом кількох хвилин, використовуючи офіційну консоль, яка пропонує інтуїтивно зрозумілий інтерфейс і широкий набір конфігураційних можливостей.

У центрі Firebase лежать дві NoSQL-бази даних — Firestore та Realtime Database, які використовують модель зберігання даних у вигляді колекцій і документів замість традиційних SQL-таблиць. Такий підхід значно спрощує роботу розробників, дозволяє швидко змінювати структуру даних і забезпечує високий рівень масштабованості [96].

Крім цього, Firebase здатна виконувати широкий спектр бекенд-функцій [70]:

- авторизація та управління користувачами;
- зберігання медіафайлів;
- надсилання push-сповіщень через Firebase Cloud Messaging;

- моніторинг продуктивності;
- аналітика користувацької активності;
- проведення A/B-тестів;
- автоматичне оновлення конфігурацій застосунку;
- інтеграція з машинним навчанням через Firebase ML.

Google активно розвиває платформу, регулярно оновлюючи її функціональні можливості. Завдяки цьому Firebase залишається одним із найбільш популярних інструментів для створення сучасних мобільних застосунків, а її консоль забезпечує зручний спосіб адміністрування та взаємодії з усіма сервісами екосистеми.

Firebase пропонує широкий спектр сервісів і інструментів, серед яких Cloud Firestore, Realtime Database, Firebase Authentication, Cloud Storage, Cloud Messaging, а також модулі машинного навчання Firebase ML. Завдяки такій різноманітності можливостей платформа охоплює практично всі базові потреби серверної частини мобільного застосунку [44].

Ключовою перевагою Firebase є те, що вона позбавляє розробника необхідності самостійно створювати й обслуговувати бекенд-інфраструктуру. Іншими словами, розробник може не витрачати ресурси на написання серверної логіки, проєктування та підтримку серверів чи конфігурацію мережевих сервісів. Це дозволяє зосередити основну увагу на фронтенд-частині — інтерфейсі та користувацькому досвіді, що суттєво прискорює розробку та покращує якість кінцевого продукту.

Firebase є типовим представником класу BaaS (Backend as a Service) — рішень, що об’єднують у собі сервер, базу даних, автентифікацію, хостинг та інші необхідні бекенд-функції в межах єдиної екосистеми. Це дає змогу створювати повноцінні мобільні додатки без піднімання власних серверів або налаштування складних інфраструктурних компонентів [31].

Одним із центральних сервісів платформи є Firebase Realtime Database, яка забезпечує розробникам API для миттєвої синхронізації даних між усіма клієнтами застосунку. Підключення до бази даних здійснюється через WebSocket

— постійне двостороннє з’єднання між пристроєм користувача та хмарним сервером Firebase (див. табл. 1.6).

Таблиця 1.6

Порівняння Cloud Firestore та Realtime Database у Firebase

Критерій	Cloud Firestore	Realtime Database
Модель даних	Документно-орієнтована база даних, де інформація організована у вигляді колекцій та документів. Така структура є гнучкішою та краще підходить для складних моделей даних.	Деревоподібна база даних у форматі JSON, де дані розміщені у вигляді вузлів і шляхів. Підходить для простих та лінійних структур.
Можливості запитів	Підтримує розширені та композитні запити, індексацію, фільтрування та сортування даних.	Можливості запитів обмежені; складні запити потребують додаткової логіки на стороні клієнта.
Режим синхронізації	Надає майже реальний час, але оптимізований для масштабованих операцій і складних транзакцій.	Забезпечує миттєву синхронізацію даних у реальному часі між всіма клієнтами.
Масштабованість	Підтримує автоматичне горизонтальне масштабування та краще працює з великими обсягами даних.	Може стикатися з проблемами масштабування в складних сценаріях та при високому навантаженні.
Структура зберігання	Ієрархічні колекції та підколекції дозволяють логічно структурувати дані.	Усі дані зберігаються в єдиному JSON-дереві, що може спричиняти дублювання та складність у підтримці.
Ціна	Оплата залежить від кількості операцій читання/запису та зберігання. Часто вигідніша для складних систем.	Оплата базується на обсязі переданих даних. Вигідніше при простих структурах і мінімальній кількості вузлів.
Кращий сценарій використання	Великі та складні системи, застосунки з великою кількістю колекцій, аналітичні моделі, e-commerce, корпоративні рішення.	Чати, стрімінгові системи, системи відстеження, IoT-застосунки, де головна вимога — швидкість оновлень.

Джерело: складено автором за [63].

Завдяки цьому будь-які зміни даних оновлюються в реальному часі на всіх підключених клієнтах без необхідності ручного оновлення чи повторного надсилання запитів. Така модель роботи особливо корисна для чатів, онлайн-ігор, стрімінгових застосунків, панелей адміністрування та систем аналітики [83].

Головна різниця між цими базами даних полягає у моделі організації даних [47].

Realtime Database використовує деревоподібну JSON-структуру, де всі записи представлені у вигляді вузлів і шляхів. Це робить її простою у використанні, але менш зручною при роботі зі складними структурами або великим обсягом даних.

Cloud Firestore, навпаки, ґрунтується на документно-орієнтованій моделі, де дані зберігаються у вигляді колекцій та документів. Така структура краще підходить для масштабних застосунків і природніше відображає об’єкти реального світу.

Ще однією ключовою відмінністю є можливості запитів. Firestore підтримує складні та композитні запити й індексацію, що дозволяє ефективно працювати з великими наборами даних. Realtime Database оптимізована для миттєвої синхронізації, але має обмеження у гнучкості запитів.

Також платформи відрізняються рівнем масштабованості. Firestore краще справляється з високими навантаженнями завдяки горизонтальному масштабуванню та більш раціональному підходу до обробки транзакцій. Натомість Realtime Database у складних сценаріях може стикатися з проблемами продуктивності, оскільки її модель синхронізації оптимізована для швидких, але простих структур даних.

База даних містить низку зручних інструментів, зокрема вбудований конструктор запитів, який дає змогу виконувати пошук даних за визначеними критеріями. Це значно спрощує роботу з вибірками та аналізом структурованої інформації.

Окрім цього, Firebase включає модуль Cloud Storage, призначений для зберігання та управління файлами. У більшості випадків доцільно організовувати структуру збереження файлів аналогічно до структури даних у базі: прив’язувати файли до відповідних елементів бази даних через чітко визначені шляхи. Це забезпечує порядок, узгодженість даних і полегшує подальше адміністрування [46].

У випадках, коли певну бізнес-логіку необхідно винести за межі мобільного застосунку — наприклад, для підвищення рівня безпеки або щоб не зберігати чутливий функціонал у клієнтській частині — Firebase пропонує спеціальний інструмент Cloud Functions. Цей сервіс дає змогу виконувати серверний код у хмарному середовищі без потреби розгортати або адмініструвати власні сервери, що фактично реалізує модель *serverless*-архітектури [43].

Cloud Functions використовує JavaScript або TypeScript і працює на базі середовища Node.js, дозволяючи запускати програмну логіку у відповідь на певні події, такі як створення нового запису в базі даних, автентифікація користувача, оновлення документа або HTTP-запит. Таким чином, розробник може реалізувати складні бекенд-функції, не виходячи за межі екосистеми Firebase [33].

Типовим прикладом є функція, яка автоматично надсилає push-сповіщення щоразу, коли в системі з’являється новий зареєстрований користувач (див. Додаток А). Подібні функції дозволяють швидко створювати реактивну логіку, забезпечувати інтеграцію між модулями та автоматизувати частину адміністративних процесів мобільного додатку.

Firebase включає широкий спектр інструментів, які можуть суттєво спростити та прискорити розробку мобільних і вебзастосунків. Повний перелік сервісів є досить великим, тому нижче наведено короткий огляд найбільш корисних і популярних серед розробників рішень описаний у табл. 1.7.

Таблиця 1.7

Основні сервіси Firebase та їх функціональність

Сервіс Firebase	Призначення та можливості
Cloud Messaging (FCM)	Забезпечує надсилання push-сповіщень і повідомлень на мобільні пристрої та вебклієнти. Дозволяє реалізувати автоматичні та персоналізовані нотифікації.
Machine Learning (ML Kit)	Надає інструменти для створення, тренування та розгортання моделей машинного навчання у хмарі. Підтримує готові ML-функції (розпізнавання тексту, облич тощо) та власні моделі.
Firebase Extensions	Містить набір готових модулів, що дозволяють автоматизувати типові процеси (обробка зображень, синхронізація даних, інтеграції зі сторонніми сервісами). Не потребує складного налаштування.
Remote Config	Дає змогу змінювати параметри роботи застосунку у реальному часі без публікації нової версії. Підходить для A/B-тестування, зміни інтерфейсу та активації експериментальних функцій.
Google Analytics for Firebase	Аналізує поведінку користувачів, ефективність застосунку, рівень утримання та залученості. Працює інтегровано з іншими сервісами Firebase і рекламними інструментами Google.
Performance Monitoring	Дозволяє виявляти проблеми з продуктивністю (повільні мережеві запити, збої, низька швидкість завантаження). Допомагає оптимізувати продуктивність мобільного застосунку.
Cloud Firestore / Realtime Database	NoSQL-бази даних для зберігання та синхронізації даних у реальному часі. Firestore забезпечує кращу масштабованість і гнучкість, тоді як Realtime Database — підходить для простих, швидких рішень.
Cloud Storage	Сховище файлів для збереження зображень, документів, відео та інших ресурсів. Забезпечує надійну роботу з великими файлами та гнучкі правила доступу.
Cloud Functions	Дає змогу виконувати бекенд-логіку у хмарі без власних серверів. Підтримує Node.js. Підходить для тригерів, обробки подій, генерації повідомлень тощо.

Джерело: складено автором за [1].

Важливо*:* частина сервісів Firebase працює за моделлю pay-as-you-go [9].

Firebase є комплексною платформою, яка об’єднує бекенд-сервіси, інструменти адміністрування, аналітику та засоби автоматизації в межах одного

екосистемного середовища. Вона значно зменшує потребу створювати власну серверну інфраструктуру та дозволяє розробникам зосередитися на бізнес-логіці, інтерфейсі та користувацькому досвіді.

Завдяки простій інтеграції, широкому набору функцій, гнучкій системі тарифікації та тісній взаємодії з іншими продуктами Google, Firebase підходить як для невеликих стартапів, так і для великих комерційних проєктів. Якщо застосунок розробляється з нуля, платформа забезпечує всі необхідні можливості для створення, тестування, запуску та подальшого масштабування сучасного мобільного продукту.

App Store Connect — це офіційна платформа Apple, яка забезпечує розробників інструментами для публікації, керування та моніторингу мобільних застосунків в екосистемі App Store. За допомогою цієї панелі адміністрування команди можуть завантажувати збірки додатків, керувати метаданими, відстежувати аналітику, взаємодіяти з тестувальниками через TestFlight та контролювати весь життєвий цикл продукту [28].

Платформа також підтримує робочі процеси командної взаємодії: можливість додавати або видаляти членів команди, призначати ролі та керувати доступами. Це дозволяє організувати ефективний розподіл відповідальностей у великих командах розробки. Наприклад, адміністратори можуть редагувати або видаляти додатки, а розробники мають доступ до завантаження збірок та технічної інформації.

Для взаємодії з App Store Connect можуть використовуватися й мобільні клієнти (наприклад, App Store Connect app для iOS), які дозволяють командам швидко переглядати аналітику, відповідати на відгуки користувачів, слідкувати за статусом перегляду додатка та отримувати оперативні сповіщення.

У корпоративних середовищах App Store Connect часто працює у зв’язці з системами мобільного керування (MDM). Одним з таких рішень є AppConnect (MobileIron), яке забезпечує контейнеризацію мобільних додатків на iOS і Android. Програми, інтегровані з AppConnect, ізолюються у захищеному середовищі, де їхні дані шифруються та захищені від несанкціонованого

доступу. Це гарантує безпечне використання корпоративних застосунків навіть на особистих пристроях співробітників [54].

У межах App Store Connect доступне й керування обліковими записами. Власники акаунтів несуть юридичну відповідальність за всі дії команди та не можуть залишити організацію, доки не передадуть роль власника іншій уповноваженій особі. Адміністратори та редактори мають можливість змінювати профілі, додавати користувачів, а також видаляти застосунки за умови, що їхній статус дозволяє це зробити.

Окремі сервіси Apple також підтримують автентифікацію через Mobile Connect — безпечний універсальний механізм ідентифікації, який дозволяє користувачам підтверджувати особу при роботі з цифровими сервісами та передавати конфіденційні дані з підвищеним рівнем захисту.

РОЗДІЛ 2
РОЗРОБКА ТА ВПРОВАДЖЕННЯ АВТОМАТИЗОВАНОГО ІНСТРУМЕНТА ДЛЯ ІТ-КОМПАНІЙ

2.1. Постановка задачі: цілі дослідження, функціональні вимоги та технічне завдання

Метою даного дослідження є розробка та впровадження інтегрованої системи адміністрування мобільних додатків, яка забезпечує централізоване керування, підвищений рівень захисту, автоматизацію ключових бізнес-процесів та оперативну взаємодію між веб-панеллю й Telegram-ботом. Система створюється для використання у компаніях, що працюють у сфері публікації, моніторингу та підтримки мобільних застосунків (зокрема Android), де важливими є швидкість реагування, надійність, безпека та можливість масштабування.

Інтегрований комплекс складається з адмін-панелі (web-модуля) та Telegram-бота, які взаємодіють через REST API, обмінюються подіями, обробляють запити адміністраторів і менеджерів, відображають аналітику, дозволяють керувати додатками, доступами, логами, S2S-подіями та іншими операціями.

Система створена для того, щоб усунути фрагментарність інструментів керування — замість використання кількох різних сервісів адміністратор отримує єдину платформу, яка охоплює всі процеси життєвого циклу мобільного продукту.

При розробці інтегрованої системи адміністрування враховувалися такі важливі аспекти:

- інтерфейс веб-панелі повинен бути інтуїтивно зрозумілим, щоб адміністратори могли швидко знаходити потрібні модулі, переглядати стани додатків, аналізувати логи та керувати користувачами;

- Telegram-бот має забезпечувати оперативність і мобільність управління, дозволяючи виконувати ключові дії без доступу до ноутбука чи браузера;
- система повинна працювати стабільно під час високих навантажень, зберігаючи цілісність даних та їхню доступність;
- архітектура має бути модульною, щоб забезпечити можливість розширення функціоналу без переробки всієї системи;
- особлива увага приділяється захисту даних, оскільки система оперує потенційно чутливою інформацією про мобільні застосунки, їхню аналітику, канали трафіку та діяльність менеджерів.

Інтегрована система адміністрування призначена для:

- централізованого управління всіма мобільними додатками компанії;
- оперативного моніторингу станів (live/ban/update);
- отримання реальних аналітичних даних щодо роботи застосунків, встановлень, S2S-подій;
- управління командами, ролями, правами доступу;
- автоматизації процесів обробки attribution-параметрів та перенаправлення користувачів через redirect-модулі;
- створення логів подій, історії дій та технічної аналітики;
- швидкої взаємодії менеджерів з системою через Telegram-бота (сповіщення, виконання команд, перегляд інформації);
- підвищення безпеки за рахунок багаторівневого контролю (ACL, CSRF-захист, SQL-фільтрація, токени сесій);
- зменшення кількості рутинних операцій завдяки автоматизації (webhooks, cron-завдання, S2S-запити).

Функціональні вимоги:

- можливість створення, редагування та перегляду інформації про мобільні додатки (CRUD-операції);
- формування та обробка attribution links з автоматичним записом параметрів у базу даних;

- підтримка Telegram-бота як мобільного інструмента управління: сповіщення, інтерактивні команди, керування командами;
- ведення журналів подій, S2S-логів, дій користувачів;
- робота з аналітикою встановлень та статусів додатків;
- інтеграція з Google Play для перевірки, оновлення чи аналізу стану додатків;
- модуль контролю доступів (ACL) для керування ролями (Admin, Manager, Viewer та ін.);
- система авторизації з шифруванням паролів та захистом від підбору;
- стабільна робота веб-панелі як PWA-додатка, що дозволяє працювати з системою навіть при нестабільному з’єднанні.

Нефункціональні вимоги:

- висока надійність та працездатність при збільшенні кількості додатків і користувачів;
- безпечне зберігання даних, захист від SQL-ін’єкцій, CSRF-атак та сесійних підробок;
- модульність структури для легкого масштабування;
- підтримка сучасних стандартів REST API;
- швидка взаємодія між системними компонентами та мінімізація часу відповіді сервера;
- логічна структура бази даних з мінімальною надмірністю.

2.2. Методологія дослідження та підходи до розробки ПЗ

Розробка інтегрованої системи адміністрування мобільних додатків потребує застосування сучасних методологій, які одночасно забезпечують структурованість процесу, масштабованість архітектури, безпеку даних та гнучкість у взаємодії між компонентами. Враховуючи комплексність проєкту та його взаємодію з Telegram Bot API, веб-панеллю, REST-сервісами та базою даних, важливо обрати методологічний підхід, який дозволяє уникнути

хаотичності розробки, мінімізує ризики та забезпечує передбачуваність результатів.

На цьому етапі визначаються ключові принципи, які ляжуть в основу реалізації системи: архітектурний шаблон MVC, REST-підхід до побудови API, специфіка розробки Telegram-ботів та практики безпечного SDLC.

Шаблон Model-View-Controller (MVC) є одним із найпоширеніших підходів до проєктування програмного забезпечення. Його використання у веб-панелі адміністрування дозволяє чітко розмежувати бізнес-логіку, інтерфейс користувача та обробку запитів. Модель відповідає за роботу з даними, контролер — за обробку HTTP-запитів і бізнес-правил, а представлення — за відображення інформації у вебінтерфейсі.

Для розробки цього проєкту MVC забезпечує:

- незалежність між шарами системи;
- можливість паралельної роботи над різними частинами;
- зручне тестування моделей і контролерів;
- кращу підтримуваність та легке масштабування.

У контексті веб-панелі адміністрування цей підхід є оптимальним, оскільки система включає велику кількість CRUD-операцій, логіку ролей доступу, модуль атрибуції, журнали подій і взаємодію через API.

Для реалізації зазначених функціональних вимог у складі веб-модуля інтегрованої системи було розроблено головний екран адміністрування, логіка якого зосереджена у файлі dashboard.php. Саме цей модуль відображає ключові показники роботи системи (кількість додатків, стан інтеграцій, активність користувачів), забезпечує швидкий доступ до основних розділів адмін-панелі та виступає стартовою точкою взаємодії адміністратора з системою.

Модуль dashboard.php під час завантаження звертається до бази даних і формує агреговані показники (кількість додатків, користувачів, атрибуційних посилань тощо), які відображаються на головній сторінці панелі. Це дозволяє адміністратору з першого екрану отримати оглядовий стан системи без необхідності переходу до інших розділів.

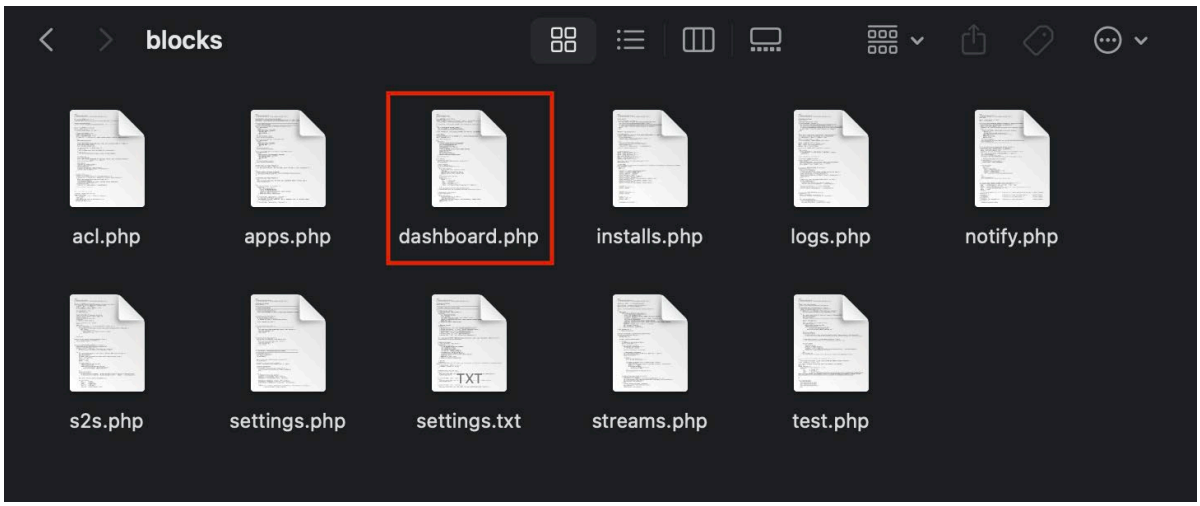


Рисунок 2.1 – Модуль dashboard.php

Нижче наведено фрагмент коду, який демонструє механізм безпечного підключення до бази даних у проєкті (див.рис.2.2) Перед встановленням з'єднання система перевіряє наявність конфігураційного файлу, що містить параметри доступу. У разі його відсутності сервер повертає помилку та припиняє виконання програми.

Підключення до MySQL здійснюється за допомогою розширення PDO з увімкненим режимом генерування винятків (PDO::ERRMODE_EXCEPTION) та асоціативним форматом отримання даних (FETCH_ASSOC). Такий підхід відповідає сучасним рекомендаціям безпечного програмування та знижує ризики SQL-ін'єкцій [91].

У наведеному фрагменті коду (рис.2.2) реалізовано ключові механізми серверної безпеки.Перед підключенням до бази даних система перевіряє наявність конфігураційного файлу (config.php). У разі його відсутності сервер завершує виконання запиту з відповідним статусом помилки.

Для роботи з базою даних використовується розширення PDO з увімкненим режимом винятків та асоціативним форматом вибірки — це забезпечує захист від SQL-ін'єкцій та централізовану обробку помилок [60].

Нижче наведено механізм генерації CSRF-токена в межах серверної сесії. Токен використовується у формах адміністративної панелі для запобігання атакам підміни запиту (CSRF) та є важливим елементом безпечного життєвого циклу ПЗ.

```php
if (!$cfg_found) { http_response_code(500); echo "config.php not found"; exit; }

try {
  $pdo = new PDO(DB_DSN, DB_USER, DB_PASS, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
  ]);
} catch (Throwable $e) { http_response_code(500); echo "DB error: ".$e->getMessage(); exit; }

// === CSRF ===
session_start();
if (empty($_SESSION['csrf'])) $_SESSION['csrf'] = bin2hex(random_bytes(16));
$CSRF = $_SESSION['csrf'];
```

Рисунок 2.2 – Фрагмент серверної логіки з перевіркою конфігурації, ініціалізацією PDO та генерацією CSRF-токена

Таким чином, файл dashboard.php є практичною реалізацією постановки задачі та вимог до програмного засобу, сформульованих у даному підрозділі.

Другим важливим елементом методології є застосування REST (Representational State Transfer), що формує основу для створення API, який обслуговує як веб-панель, так і Telegram-бота.

REST дозволяє [65]:

- будувати передбачувані маршрути (endpoints) для доступу до ресурсів;
- відокремлювати інтерфейс користувача від бекенд-логіки;
- забезпечувати легку інтеграцію зі сторонніми сервісами (Google Play Developer API, S2S постбеки, вебхуки рекламних платформ);
- досягати масштабованості завдяки безстанності запитів.

У межах системи кожен ресурс — користувачі, боти, групи, логи, атрибуція — має власний набір REST-ендпоїнтів, що забезпечують створення, читання, оновлення та видалення (CRUD). Через REST взаємодіють також внутрішні модулі: Telegram-бот надсилає запити на сервер для оновлення таймерів, отримання інформації про користувача або створення записів у логу.

Telegram Bot API — це основний інструмент для реалізації інтерактивної частини системи, відповідальної за автоматизацію комунікації з менеджерами та клієнтами [79].

Telegram-бот у межах проєкту виконує такі функції:

- обробляє повідомлення, оновлення та callback-запити;
- запускає таймери взаємодії (SLA) і надсилає відповідні сповіщення;
- інтегрується з внутрішнім API для отримання ролей, логування дій, управління групами;
- забезпечує session management на рівні чатів і користувачів.

Застосування Telegram Bot API визначає окремий підхід до реалізації логіки — через webhook або long polling. У цьому проєкті обрано webhook-механізм, що дозволяє серверу оперативно отримувати інформацію про нові події й обробляти їх у реальному часі. Це підвищує ефективність системи, зменшує затримки та забезпечує швидку реакцію на дії користувача [84].

Одним із ключових аспектів методології є дотримання практик Secure Software Development Life Cycle (безпечного життєвого циклу розробки ПЗ). Використання цього підходу дозволяє проєктувати програмне забезпечення з урахуванням безпеки на кожному етапі — від аналізу вимог до тестування та впровадження [95].

Безпечний SDLC у проєкті передбачає [68]:

- збір вимог із фокусом на загрози (SQL-ін’єкції, CSRF, brute-force атаки, компрометація токенів);
- проєктування архітектури з використанням ACL, захищених сесій, HTTPS;
- реалізацію моделей і контролерів на основі prepared statements;
- перевірку даних та фільтрацію користувацького вводу;
- налаштування ролей і прав доступу для користувачів панелі;
- логування та аудит операцій, що важливо для виявлення інцидентів;
- тестування безпеки (автоматизовані тести, модульні тести, ручна перевірка).

Застосування Secure SDLC гарантує, що система адміністрування буде не лише функціональною, але й захищеною від типових загроз, які особливо актуальні для рішень, що працюють з персональними даними, логами взаємодії та атрибуційною інформацією.

Усі описані методології не існують окремо, а формують взаємопов'язану структуру. MVC забезпечує архітектуру веб-панелі; REST — механізм комунікації між компонентами; Telegram Bot API — канал взаємодії з користувачами; безпечний SDLC — основу для проєктування та впровадження всіх модулів з урахуванням загроз.

У результаті формується система, яка:

- легко масштабується;
- зручна в обслуговуванні та розширенні;
- інтегрується зі сторонніми сервісами;
- відповідає вимогам безпеки;
- має чітко структуровану архітектуру та зрозумілі блоки відповідальності.

Такий підхід дозволяє уникнути перевантаження коду, забезпечує передбачуваність поведінки системи та формує надійний технологічний фундамент для реалізації функціональних можливостей, визначених у попередньому розділі.

2.3. Архітектура системи та взаємодія структурних компонентів

Архітектура інтегрованої системи адміністрування мобільних продуктів складається з декількох логічно пов’язаних компонентів, кожен з яких виконує специфічні функції та взаємодіє з іншими модулями через стандартизовані інтерфейси. Система побудована за принципами модульності, розділення відповідальності, та використовує гібридний архітектурний підхід, який поєднує елементи MVC, REST-сервісів, компонентну організацію вебінтерфейсу та Telegram Bot API як окремий комунікаційний канал.

Інтегрована платформа включає такі основні компоненти:

- вебпанель адміністрування (адмінінтерфейс для менеджерів та операторів);
- Telegram-бот, який виконує роль мобільного інструменту доступу до системи;
- REST-API та API-gateway, що відповідають за доступ до даних та маршрутизацію запитів;
- базу даних MySQL;
- PWA-модуль, який забезпечує автономність та швидке завантаження вебінтерфейсу;
- служби безпеки, включно з ACL, автентифікацією, CSRF та SQL-захистом.

Завдяки такій структурі система зберігає високу масштабованість і може обробляти велику кількість одночасних запитів як з вебпанелі, так і через Telegram-бота.

Архітектура системи будується за принципом «клієнт — сервер», де клієнтом виступають:

- браузер адміністратора,
- мобільний додаток Telegram,
- бекенд інших інтегрованих систем (Google Play, S2S-джерела, атрибуційні платформи).

Серверна частина включає:

- PHP-контролери та REST-ендпоінти,
- модулі бізнес-логіки,
- Telegram-бот логіку,
- шари безпеки,
- рівень доступу до даних (DAL).

Вебпанель — це центральний інструмент керування мобільними застосунками, атрибуційними посиланнями, користувачами та їх ролями, файлами, логами, статистикою. Архітектурно панель побудована на основі PHP та використовує MVC-подібний підхід.

Структурні компоненти вебпанелі:

1. View-шари — директорії:
   - /assets — стилі, JavaScript, зображення;
   - /blocks — часткові шаблони інтерфейсу;
   - /pwa — компоненти для перетворення панелі на PWA;
2. Controller-шари — файли:
   - index.php — головний контролер;
   - login.php, logout.php — контролери автентифікації;
   - redirect.php — логіка атрибуційного перенаправлення;
   - tools/*.php — дії над сутностями (CRUD);
3. Model-шари — директорія:
   - /inc — PHP-файли з логікою роботи з базою даних (підключення, запити, утилітні функції);
4. Системні компоненти:
   - /cache — кэш тимчасових даних;
   - /logs — логи системи.

Telegram-бот виконує роль мобільного каналу доступу до можливостей системи, дозволяючи керувати застосунками, переглядати статуси, запускати операції та взаємодіяти з API. Бот працює через Webhook, отримуючи оновлення напряму від Telegram API.

Система використовує власний внутрішній REST-шар, який обробляє запити як від Telegram-бота, так і від вебпанелі. API-gateway виконує роль маршрутизатора, який:

- приймає HTTP-запит,
- визначає ресурс та дію (/apps/list, /apps/update, /user/login),
- викликає відповідний контролер,
- проводить авторизацію (ACL),
- повертає JSON-відповідь.

```
/api/apps/list.php
/api/apps/get.php?id=12
/api/links/create.php
```

Рисунок 2.3 – **Типовий REST-ендпоінт**

Для запобігання атакам підміни міжсайтових запитів (CSRF) у системі реалізовано механізм генерації та перевірки унікального токена, прив’язаного до серверної сесії користувача. Після створення токена він зберігається у змінній $_SESSION['csrf'] та передається у всі важливі форми адміністративної панелі.

Передача токена здійснюється у вигляді прихованого HTML-поля (<input type="hidden">). Це дозволяє браузеру відправляти токен лише тоді, коли запит сформовано з легітимної форми, вбудованої у систему.

Під час обробки POST-запиту сервер порівнює отримане значення токена з тим, що зберігається у сесії. Якщо токен відсутній або не збігається — запит вважається потенційно шкідливим і завершується з помилкою доступу.

Такий підхід унеможливлює автоматичну відправку шкідливих запитів сторонніми сайтами, оскільки CSRF-токен не передається браузером автоматично та може бути отриманий виключно через легітимний інтерфейс адміністративної панелі.

```php
session_start();

if (!isset($_POST['csrf']) || $_POST['csrf'] !== $_SESSION['csrf']) {
    http_response_code(403);
    exit("Invalid CSRF token");
}
```

Рисунок 2.4 – Механізм генерації та перевірки унікального токена

Перевірка CSRF-токена відбувається на стороні сервера до виконання будь-якої бізнес-логіки. Лише у разі коректної валідації запит допускається до обробки модулем CRUD. Така архітектура повністю відповідає принципам

безпечного SDLC та дозволяє мінімізувати ризики несанкціонованих змін у системі адміністрування.

Підсумовуючи викладене, реалізований комплекс механізмів безпеки забезпечує захист інтегрованої системи на всіх ключових рівнях — від запобігання типових вебзагроз до контролю доступу та коректної роботи зовнішніх інтеграцій. Використання підготовлених запитів і PDO мінімізує ризик SQL-ін’єкцій, а впровадження криптографічних CSRF-токенів гарантує достовірність кожного запиту, що змінює стан системи.

Поєднання безпечної автентифікації, хешування паролів, сесійного менеджменту та гнучкої системи ACL формує надійний багаторівневий контроль доступу й запобігає несанкціонованим діям адміністративних користувачів. Водночас інтеграція сторонніх сервісів через S2S-запити, webhooks та API виконується з урахуванням вимог безпеки та гарантує коректність обміну даними з мобільними додатками та сервісами аналітики.

У сукупності ці рішення створюють цілісну модель безпеки, що відповідає принципам безпечного SDLC і забезпечує надійну роботу всієї системи навіть за умови зростання навантаження, появи нових модулів та інтеграцій. Реалізований підхід дозволяє підтримувати стабільність, цілісність та конфіденційність даних, що є критично важливим для інструментів адміністрування мобільних продуктів.

2.4. Проєктування бази даних та взаємодія з Telegram-ботом

Проєктування бази даних у межах інтегрованої системи адміністрування мобільних додатків є одним із ключових етапів створення програмного засобу. Саме база даних забезпечує зберігання структурованої інформації про застосунки, користувачів, атрибуційні посилання, журнали подій та інші сутності, які необхідні для коректної роботи вебпанелі та Telegram-бота. Ефективна взаємодія між серверною частиною, адміністративною панеллю та ботом можлива лише за умови чітко визначеної структури даних, оптимізованих зв’язків між таблицями та реалізованого механізму керування сесіями.

Для забезпечення стабільної роботи інтегрованої системи ключовим елементом стає правильно спроєктована база даних. Саме вона визначає, яким чином адміністраторська панель, Telegram-бот і PWA-модуль обмінюються інформацією, зберігають дані про додатки, користувачів, статистику та логи. Тому перед реалізацією функціональних модулів необхідно розглянути логічну модель БД і визначити зв'язки між її основними сутностями [34].

Основною вимогою до моделі даних було створення *гнучкої*, *масштабованої* та *безпечної* структури, яка дозволить:

- централізовано керувати переліком мобільних застосунків;
- реєструвати атрибуційні події, переходи за рекламними посиланнями та параметри трекінгу;
- зберігати інформацію про користувачів адміністративного інтерфейсу та їхні ролі;
- вести журнали подій (logs), що надходять як із вебпанелі, так і з Telegram-бота;
- забезпечити Telegram-боту можливість читати й змінювати дані відповідно до прав доступу;
- підтримувати механізми ACL, CSRF та сесійного управління.

Усі сутності були нормалізовані таким чином, щоб уникнути надмірності даних, підвищити швидкість доступу та забезпечити можливість подальшого масштабування системи.

ER-діаграма допомагає побачити, як саме організований рух даних у системі, які сутності є центральними, а які — допоміжними, та як база даних підтримує цілісність і узгодженість інформації. ER-діаграма (Entity–Relationship) відображає ключові таблиці та зв'язки між ними.Стандартний набір сутностей проєкту включає [42]:

- apps – список мобільних додатків;
- links – атрибуційні посилання, пов'язані з додатками;
- users – адміністратори системи;
- roles – ролі для ACL;

- logs – журнал подій;
- sessions – таблиця керування сесіями бота (опціонально, якщо не використовується лише файлова сесія PHP).

Процес інтеграції бази даних у проєкт розпочався з налаштування серверного середовища на основі MySQL та створення окремої бази даних, що використовується адміністративною панеллю та Telegram-ботом. Первинне керування структурою БД здійснювалося через візуальний інструмент phpMyAdmin, де було створено основні таблиці та задано необхідні зв’язки між ними. Для роботи застосунку-бота в MySQL було створено окремого користувача з обмеженими правами доступу, що підвищує рівень безпеки.

Взаємодія веб-модуля з БД реалізована через PDO-механізм у PHP, який забезпечує використання підготовлених запитів і захищає систему від SQL-ін’єкцій. У файлі *config.php* визначено параметри підключення: DSN, логін, пароль та набір атрибутів, що активують обробку помилок, режим винятків та відключення емуляції prepared statements. Приклади ініціалізації з’єднання та обробки помилок наведено у Додатку А (див. Рис. А.4).

Структуру бази даних було спроєктовано відповідно до вимог системи адміністрування мобільних продуктів (рис 2.5).

Вона включає сутності apps, users, teams, logs, client_flows, notifications, а також додаткові таблиці, що забезпечують роботу модулів атрибуції та Telegram-бота. Логічні зв’язки між таблицями відображені на ER-діаграмі (див. Рис. 2.4), яка демонструє використання зовнішніх ключів, нормалізацію даних та каскадні зв’язки.

Особливу роль у системі відіграє таблиця users, де зберігаються облікові записи адміністраторів та хешовані паролі, що формують основу моделі автентифікації. Таблиця apps використовується для зберігання інформації про мобільні додатки, а таблиця logs фіксує всі ключові адміністративні операції, що забезпечує аудит і контроль активностей.

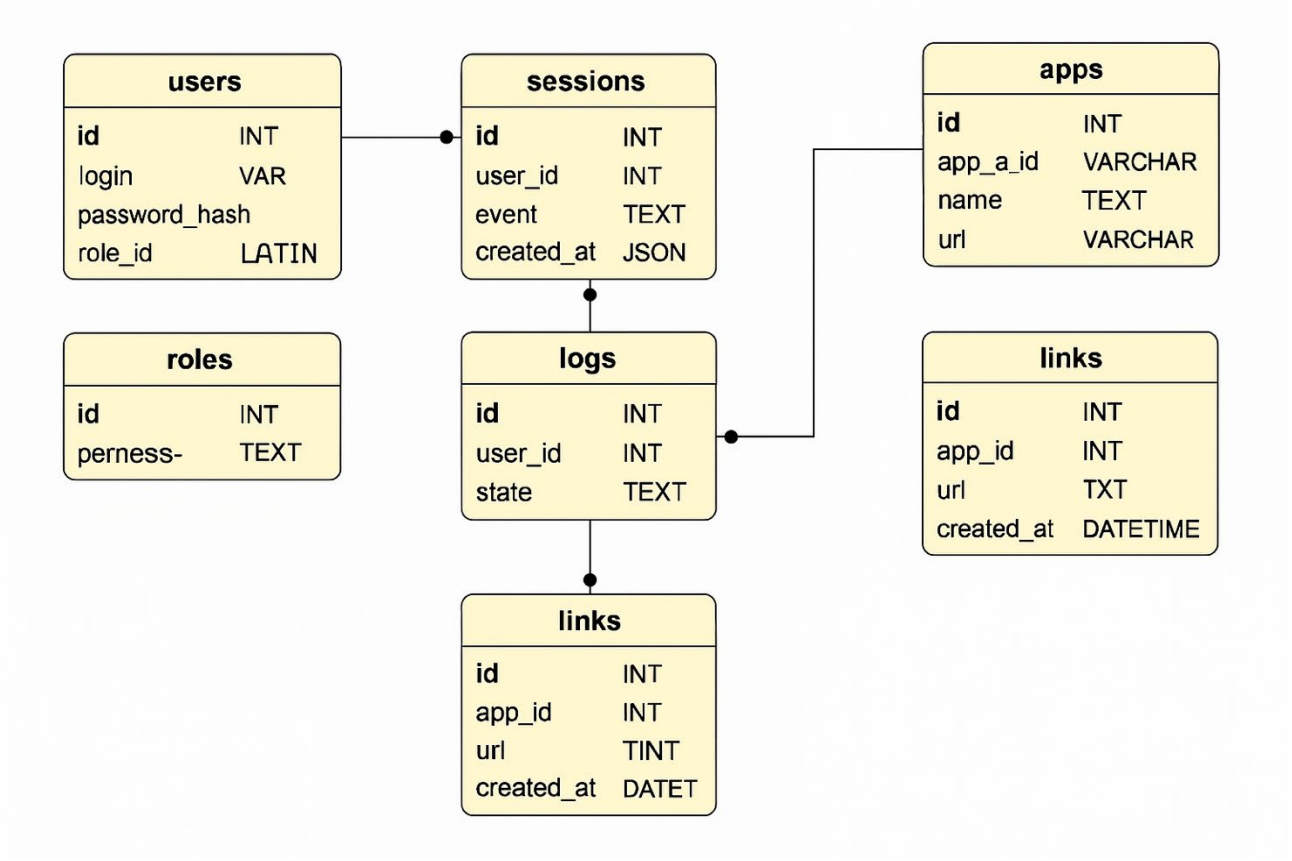


Рисунок 2.5 – ER-діаграма бази даних інтегрованої системи адміністрування

Джерело: *складено автором за [10].*

Таблиця 2.1

Таблиця apps

Поле	**Тип**	**Опис**
id	INT, PK	Унікальний ідентифікатор адміністратора
username	VARCHAR	Логін користувача
password_hash	VARCHAR	Хешований пароль (bcrypt/argon2)
role	ENUM	Роль у системі (admin, manager, operator)
team_id	INT, FK	Прив’язка до команди
created_at	DATETIME	Дата створення запису

Джерело: складено автором за [64].

Хешування паролів реалізовано через стандартні PHP-функції password_hash(), що відповідає вимогам безпечного SDLC.

Таблиця використовується для зберігання інформації про мобільні додатки. Дані з apps активно використовуються Telegram-ботом для відправки оновлень командам, а також відображаються у Dashboard адміністративної панелі.

Таблиця 2.2

Таблиця logs

Поле	**Тип**	**Опис**
id	INT	Первинний ключ
event	TEXT	Опис події
user_id	INT	Хто виконав дію
created_at	DATETIME	Час події
data	JSON	Додаткова інформація

Джерело: складено автором за [12].

Telegram-бот взаємодіє з базою даних через внутрішній PHP-модуль, що міститься в директорії *telegram-bot/*. При отриманні подій через Webhook бот звертається до БД для читання або запису інформації. Це дозволяє синхронізувати дані між ботом та адмінпанеллю: наприклад, оновлення статусу додатка, створення записів про бан, надсилання повідомлень команді тощо.

У системі реалізовано механізм session management, що забезпечує збереження CSRF-токенів, поточних користувацьких сесій та історії роботи адміністраторів. Усі запити, які змінюють стан системи (редагування додатків, зміна доступів, створення посилань), супроводжуються перевіркою CSRF-токена, згенерованого на етапі ініціалізації сесії (див. Рис. А.5).

Побудована модель БД є достатньо гнучкою для подальшого розширення. Дані Telelgram-бота, веб-панелі та атрибуційних механізмів зберігаються

централізовано, що забезпечує коректну взаємодію між модулями та масштабованість системи у випадку збільшення кількості додатків чи команд.

Підключення до MySQL реалізовано через PDO — сучасний та безпечний спосіб роботи з SQL у PHP. У конфігураційному файлі визначено DSN, логін і пароль, які зберігаються у прихованому середовищі сервера.

```php
if (!$cfg_found) {
    http_response_code(500);
    echo "config.php not found";
    exit;
}

try {
  $pdo = new PDO(DB_DSN, DB_USER, DB_PASS, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
  ]);
} catch (Throwable $e) {
  http_response_code(500);
  echo "DB error";
  exit;
}

// CSRF-токен
session_start();
if (empty($_SESSION['csrf'])) {
    $_SESSION['csrf'] = bin2hex(random_bytes(16));
}
```

Рисунок 2.6 – Приклад реалізації підключення до бази даних через PDO та генерації CSRF-токена у dashboard.php

Telegram-бот працює у режимі Webhook: сервер приймає запит від Telegram, обробляє команду, читає або записує дані в MySQL, після чого надсилає відповідь користувачу.

Оскільки веб-панель використовують адміністратори, був розгорнутий механізм захищених сесій, який включає:

- збереження CSRF-токенів;
- захищений ідентифікатор сесії;
- перевірку ролей та ACL;

- автоматичне завершення сесій при неактивності;
- прив’язку сесії до IP або user-agent (за потреби).

Особлива увага приділена тому, щоби сесії не можна було викрасти або підробити.

Це критично, оскільки панель керує приватними даними додатків та командами у виробничому середовищі.

Реляційна модель MySQL забезпечує структуроване зберігання даних, підтримує масштабування та дозволяє безпечно виконувати складні операції. Взаємодія PHP-модуля з БД через PDO, реалізація CSRF-токенів, використання журналів дій та централізована модель управління сесіями гарантують надійність і безпечність роботи системи.

2.5. Етапи реалізації веб-модуля та Telegram-бота

Під час створення веб-модуля важливим стало формування універсального та масштабованого механізму роботи з даними, що був реалізований за допомогою CRUD-операцій. Саме цей функціональний блок визначає базовий набір можливостей системи: створення, читання, оновлення та видалення інформації, яка зберігається у спільній базі даних і використовується як веб-інтерфейсом, так і Telegram-ботом. CRUD-логіка реалізована через моделі, контролери та API-ендпоїнти, що утворюють основу взаємодії між користувачем і серверною частиною.

Реалізація CRUD-операцій (Create, Read, Update, Delete) є фундаментальною частиною архітектури інтегрованої системи адміністрування мобільних додатків. Саме CRUD забезпечує можливість створення, перегляду, редагування та видалення даних у системі. У розробленому веб-модулі та Telegram-боті ці операції відіграють ключову роль у підтримці актуальності інформації, управлінні контентом, обробці налаштувань застосунків та організації логування подій.

CRUD (Create, Read, Update, Delete) — це чотири фундаментальні операції, на яких ґрунтується взаємодія між користувачем, контролерами застосунку та базою даних. У межах інтегрованої системи «Адмін-панель + Telegram-бот» CRUD-логіка забезпечує керування записами про додатки, команди, S2S-події, журнали логів, користувачів та інші структурні одиниці.

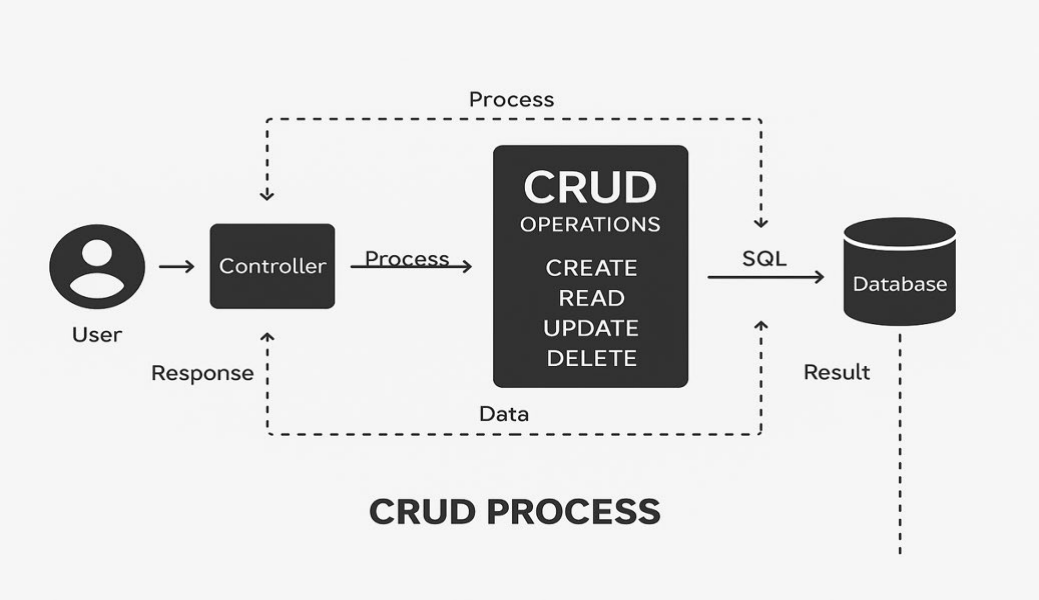


Рисунок 2.7 — Загальна схема CRUD-процесу

Джерело: *складено автором за [16].*

Завдяки поділу логіки відповідно до шаблону Model–View–Controller (MVC), CRUD-операції реалізовано структуровано та безпечно. Моделі відповідають за взаємодію з базою даних, контролери — за обробку HTTP- або Telegram-запитів, а API-ендпоїнти — за передачу даних між веб-інтерфейсом, ботом та сервером.

Моделі в системі реалізовані у вигляді PHP-класів, що інкапсулюють виконання SQL-запитів через PDO. Такий підхід забезпечує:

- безпечне виконання запитів (підготовлені вирази — Prepared Statements),
- повторне використання функцій моделі,
- відділення бізнес-логіки від обробки HTTP-запитів.

```php
class AppModel {
    private PDO $db;

    public function __construct(PDO $pdo) {
        $this->db = $pdo;
    }

    public function getApps() {
        $sql = "SELECT id, name, package, created_at FROM apps ORDER
BY id DESC";
        $stmt = $this->db->prepare($sql);
        $stmt->execute();
        return $stmt->fetchAll();
    }
}
```

Рисунок. 2.8 — Приклад моделі отримання даних

Цей фрагмент демонструє принцип Read-операції: модель повертає список додатків для відображення у веб-панелі або Telegram-боті.

Контролери приймають запити від користувача (адміністратора), виконують валідацію введених даних, звертаються до моделі та повертають відповідь у відповідному форматі (HTML, JSON або повідомлення Telegram).

```php
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $name = $_POST['name'] ?? '';
    $package = $_POST['package'] ?? '';

    if (!$name || !$package) {
        echo "Помилка: усі поля обов'язкові!"; exit;
    }

    $stmt = $pdo->prepare("INSERT INTO apps (name, package) VALUES
(?, ?)");
    $stmt->execute([$name, $package]);

    header("Location: dashboard.php");
}
```

Рисунок. 2.9 — Приклад логіки створення нового запису

Це типовий приклад Create-операції у веб-модулі.

Telegram-бот не працює напряму з базою даних — він взаємодіє з веб-модулем через REST-ендпоїнти. Це дозволяє:

- підтримувати єдине джерело даних,
- повторно використовувати логіку контролерів,
- забезпечувати безпеку доступу через токени.

```php
header('Content-Type: application/json');

$stmt = $pdo->query("SELECT id, name, package FROM apps ORDER BY id
DESC");
echo json_encode($stmt->fetchAll());
```

Рисунок. 2.10 — Типовий ендпоїнт повернення списку додатків

Telegram-бот дозволяє адміністраторам виконувати ті самі дії, що й веб-панель, але через інтерактивний інтерфейс.

```php
$apps = json_decode(file_get_contents(API_URL . "/apps.php"), true);

foreach ($apps as $app) {
    $text .= "📱 {$app['name']} ({$app['package']})\n";
}
```

Рисунок. 2.11 — Отримання ботом дані

А якщо адміністратор натискає кнопку Видалити додаток, то бот надсилає запит:

```php
file_get_contents(API_URL . "/delete_app.php?id=" . $app_id .
"&token=" . ADMIN_TOKEN);
```

Рисунок. 2.12 — Запит бота

Веб-панель містить форми для введення інформації про додатки, ключі доступу, налаштування редиректів, атрибуційних лінок та системних параметрів.

Використовуються такі механізми безпеки:

- CSRF-токен у прихованих полях форми,
- server-side валідація всіх даних,
- підготовлені SQL-запити,
- обмеження прав доступу залежно від ролі користувача.
-

```
<input type="hidden" name="csrf" value="<?= $CSRF ?>">
```

Рисунок. 2.13 — Приклад вставки CSRF

CRUD є “хребтом” всіх бізнес-процесів системи:

- веб-панель керує записами — додатками, ключами, атрибуційними лінками;
- Telegram-бот дозволяє виконувати ті самі операції дистанційно;
- API-ендпоїнти забезпечують комунікацію між модулями;
- логування CRUD-операцій зберігається в таблиці logs, що підвищує рівень безпеки.

Таким чином CRUD-механізм забезпечує підтримку цілісності та керованості всієї інфраструктури мобільних додатків.

Другим важливим компонентом є механізм attribution links, який дозволяє відстежувати джерела трафіку, переходи користувачів і параметри рекламних кампаній. Він дозволяє визначити, звідки саме прийшов користувач, яка рекламна кампанія згенерувала перехід, які параметри передавались у момент запуску застосунку або відкриття промосторінки. Він реалізований через спеціальний сценарій redirect.php, що обробляє вхідні UTM-параметри, виконує перенаправлення на сторінку застосунку та зберігає параметри у логах. Це забезпечує точний збір статистики та подальшу аналітику дій користувачів.

Основна мета атрибуції — збір, збереження та аналіз параметрів переходів користувачів на сторінку мобільного застосунку. Завдяки цьому система дозволяє:

- визначати джерело трафіку (Facebook Ads, Google Ads, Telegram, партнерські мережі тощо),
- оцінювати ефективність рекламних кампаній,
- збирати додаткові параметри (utm-мітки, партнерські ID, субідентифікатори),
- будувати воронки користувацьких переходів,
- виявляти аномалії та некоректну активність у кампаніях,
- зіставляти інсталяції мобільного застосунку з реальними рекламними джерелами.

Усі дані, отримані під час переходу за attribution link, фіксуються у таблиці логів, що дозволяє системі адміністрування аналізувати поведінку трафіку безпосередньо в панелі керування (див. рис.2.14).

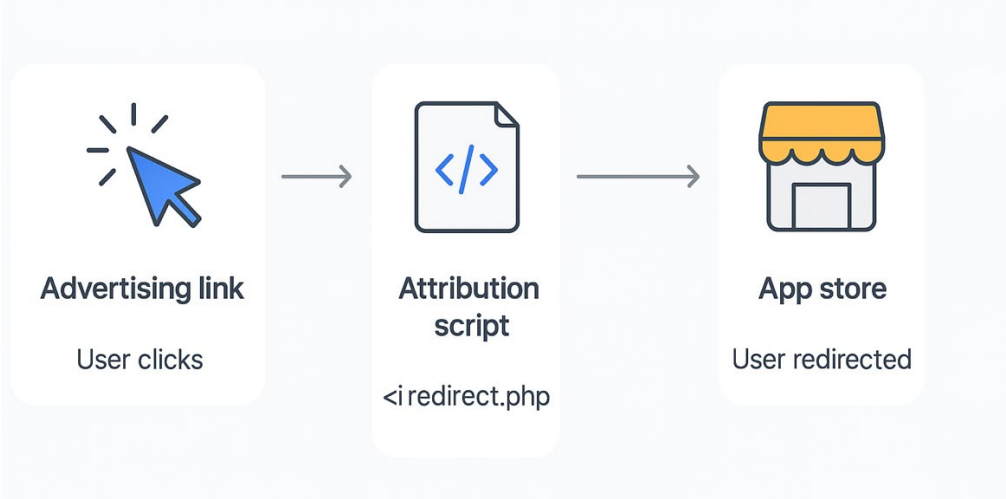


Рисунок 2.14 — Ілюстрація схеми атрибуції

Джерело: складено автором за [7].

Telegram-бот також інтегрується з механізмом атрибуції. Його задачі включають:

- генерацію партнерських лінків для команд або менеджерів,
- передавання параметрів у redirect.php через динамічні кнопки,
- автоматичне формування UTM-міток,
- відстеження ефективності роботи кожної команди.

Це дозволяє централізовано відстежувати результативність менеджерів і команд у рамках однієї системи.

Наступним етапом стало впровадження ACL (Access Control List) — системи керування ролями та правами доступу. ACL гарантує коректне розмежування повноважень між різними категоріями користувачів веб-панелі: адміністраторами, менеджерами, аналітиками та технічними працівниками. Механізм реалізується через middleware-перевірки, авторизаційні токени та відповідні правила у моделі доступів.

У реалізованій системі доступ користувача до будь-якого ресурсу визначається двома основними параметрами:

1. Роль користувача
2. Перелік дозволених дій відповідно до ролі (permissions)

Це дозволяє не лише відрізняти адміністратора від менеджера, а й гнучко налаштовувати рівень доступу до окремих функцій, включаючи:

- перегляд та редагування програм;
- доступ до логів;
- створення attribution-лінків;
- управління командами;
- роботу з API-ключами та інтеграціями;
- доступ до аналітики;
- надсилання ручних повідомлень через бота.

На рівні БД для цього використано таблиці users, roles, user_roles, де зберігається інформація про те, хто і які права має.

У системі визначено кілька типових ролей (рис. 2.15).

Таке розмежування мінімізує ризики, пов’язані з людським фактором, і дозволяє делегувати завдання без надання надмірних прав.

Усі запити до веб-панелі проходять через спеціальний проміжний шар — middleware. Він виконує кілька функцій.

1. Перевірка автентифікації (чи існує активна сесія).
2. Перевірка CSRF-токенів, якщо запит змінює дані.

3. Зіставлення ролі користувача та дозволених дій.
4. Реагування на спроби несанкціонованого доступу.

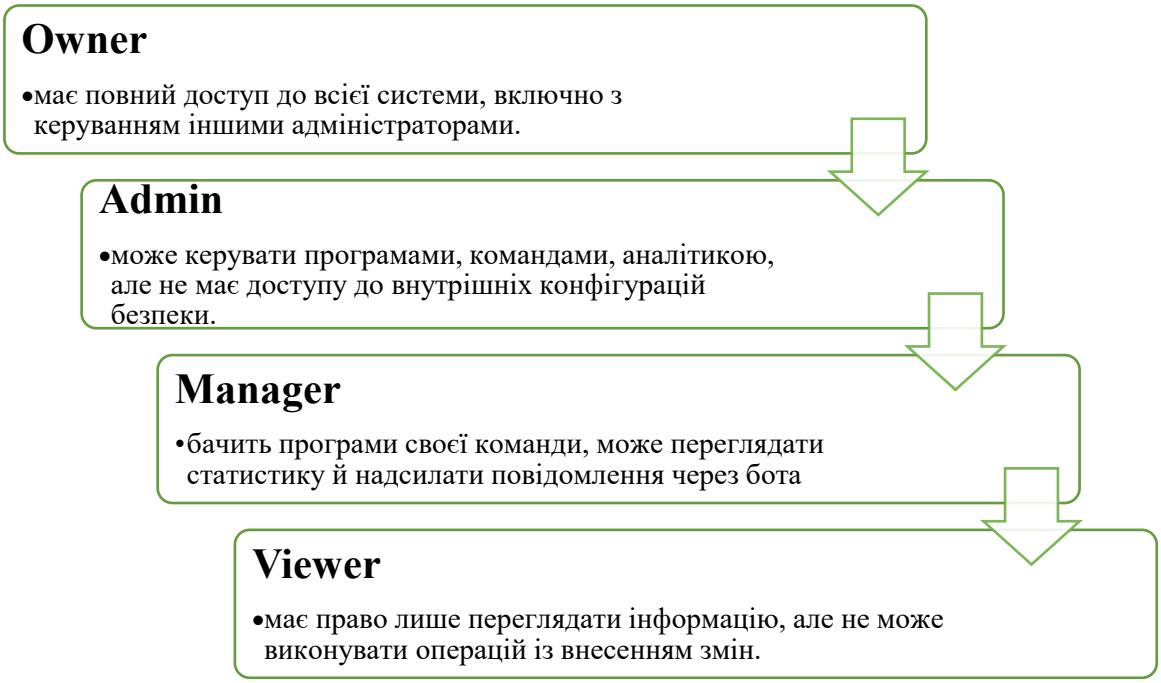


Рисунок 2.15 — Ролі користувачів

Джерело: складено автором за [7].

```php
function requireRole($roleList) {
    if (!isset($_SESSION['user_role']) ||
!in_array($_SESSION['user_role'], $roleList)) {
        http_response_code(403);
        echo "Access denied";
        exit;
    }
}
```

Рисунок 2.16 — Приклад фрагмента коду, який реалізує описану логіку

Оскільки Telegram-бот у системі виконує не лише інформаційні, а й адміністративні функції — зокрема, надсилання сповіщень, керування командами або запуск окремих сервісних операцій — механізм ACL поширюється й на нього. Кожен запит, який надходить до бота, проходить внутрішню перевірку: бот встановлює, чи прив’язаний Telegram-ID користувача

до існуючого адміністративного акаунта, визначає роль цього користувача в системі та оцінює, чи має він достатній рівень прав для виконання відповідної команди.

Лише після успішного проходження цих перевірок команда фактично виконується. Завдяки такому підходу система унеможливлює випадковий або навмисний доступ сторонніх осіб до чутливих функцій, забезпечуючи цілісність, контрольованість і безпечність адміністративних операцій.

У веб-панелі адміністратора система ACL визначає не лише те, які дії доступні користувачам на бекенді, але й формує персоналізовану взаємодію з інтерфейсом на рівні front-end. Правила доступу впливають на відображення елементів інтерфейсу, приховуючи або показуючи окремі модулі, блокуючи недоступні форми редагування та налаштовуючи зміст Dashboard відповідно до ролі користувача. Завдяки цьому менеджер, авторизувавшись у системі, бачить лише ті програми, що належать його команді, тоді як розділ «Аналітика» може бути для нього частково обмеженим або спрощеним. Інтерфейс таким чином адаптується до повноважень кожного користувача, зменшуючи ризик помилкових змін і водночас роблячи роботу з панеллю зручнішою.

Впровадження ACL у проєкті суттєво посилює безпеку системи, адже унеможливлює несанкціоновані або випадкові дії, що можуть вплинути на критичні дані чи роботу сервісу. Ця модель доступу добре масштабується: за потреби можна додавати нові ролі, деталізувати існуючі або гнучко змінювати їхній набір дозволів. Централізація логіки перевірки прав робить управління доступом прозорим і контрольованим, а підтримка детальних, «гранульованих» дозволів дозволяє визначати доступ не лише на рівні модулів, а й окремих операцій. У результаті система отримує гнучкий, безпечний і керований механізм взаємодії як на рівні бекенду, так і на рівні користувацького інтерфейсу.

Також важливою складовою роботи інтегрованої системи є інтеграція з Google Play, оскільки саме цей етап забезпечує отримання актуальних даних про мобільні застосунки, контроль за їхнім станом, відстеження інсталяцій,

перевірку оновлень, а також можливість автоматизації бізнес-процесів, пов’язаних із публікацією та аналітикою. У рамках розроблюваної системи інтеграція необхідна для того, щоб веб-панель адміністрування та Telegram-бот могли працювати з актуальними показниками кожного застосунку та коректно виконувати завдання з обробки атрибуції, моніторингу та підтримки.

Функціонально інтеграція базується на Google Play Developer API — інструменті, який дозволяє автоматизувати низку операцій, що зазвичай виконуються через вебінтерфейс Play Console. Це включає доступ до фінансових звітів, інформації про країни доступності, інсталяції, статуси оновлень, а також перевірку підписів та керування релізами. Для коректної роботи API система використовує облікові дані сервісного акаунту, які зберігаються у захищеному вигляді в конфігурації сервера.

У межах проєкту API інтегровано у бекенд через спеціальний модуль, який відповідає за формування запитів, авторизацію через OAuth 2.0 та обробку відповідей Google Play. Така реалізація дає змогу централізовано керувати всіма даними застосунків і забезпечує узгодженість між веб-панеллю та Telegram-ботом. Наприклад, коли адміністратор переглядає у Dashboard інформацію про застосунок, дані щодо його статусу, кількості активних установок або доступних країн отримуються не з локальної БД, а безпосередньо з API Google Play, що гарантує їхню актуальність.

Окрему увагу слід приділити обробці атрибуції інсталяцій. Після того як користувач здійснює клік по рекламному посиланню, система фіксує параметри у redirect.php, записує їх у таблицю logs і надалі через API Google Play звіряє, чи відбулася інсталяція. Це дозволяє зіставити рекламні переходи з реальними установками, формуючи точну маркетингову статистику. Такий механізм зменшує кількість помилкових атрибуцій і забезпечує прозорість рекламних кампаній.

Також API Google Play використовується для автоматизації частини дій, які інакше потрібно було б виконувати вручну. Наприклад, можна отримувати автоматичні сповіщення про відхилені релізи, статуси перевірки або технічні

вимоги Play Store. Інтеграція з Telegram-ботом дозволяє надсилати адміністраторам повідомлення про критичні зміни, що значно пришвидшує реакцію на події.

Для захисту інтеграції застосовано низку механізмів: обмеження прав сервісного акаунту, використання ізольованих ключів, зберігання конфіденційних даних у закритих системних файлах та перевірку кожного запиту через власний ACL. Усе це запобігає несанкціонованому доступу до Google Play Console і забезпечує безпеку роботи системи в цілому.

У процесі інтеграції з Google Play система отримує через Google Play Developer API низку показників, що характеризують стан мобільного застосунку, його версії та активність користувачів. У таблиці 2.3 наведено приклади основних полів, які можуть використовуватися веб-панеллю та Telegram-ботом для відображення аналітики, контролю публікацій і моніторингу життєвого циклу продукту.

Для захисту інтеграції застосовано низку механізмів: обмеження прав сервісного акаунту, використання ізольованих ключів, зберігання конфіденційних даних у закритих системних файлах та перевірку кожного запиту через власний ACL. Усе це запобігає несанкціонованому доступу до Google Play Console і забезпечує безпеку роботи системи в цілому.

У межах реалізації Telegram-бота було створено повноцінну інтегровану систему, яка охоплює ключові аспекти сучасної розробки та адміністрування мобільних продуктів. На рівні бекенду впроваджено структуровані CRUD-операції, що забезпечують зручне керування даними через моделі, контролери та API-ендпоїнти. Це дало змогу стандартизувати модифікацію інформації, зберегти її цілісність та підвищити надійність системи.

Механізм атрибуційних посилань реалізовано через файл redirect.php, який дозволяє відстежувати переходи користувачів, фіксувати UTM-параметри та передавати їх до відповідних магазинів застосунків.

Таблиця 2.3

Основні поля, що отримуються з Google Play Developer API

№	Поле / атрибут	Приклад значення	Опис даних
1	packageName	com.company.appname	Унікальний ідентифікатор пакета застосунку в Google Play
2	versionCode	105	Внутрішній числовий код версії (для оновлень, релізів, порівнянь)
3	versionName	1.5.0	Людинозрозуміла назва версії, що відображається користувачам
4	track	production, beta, internal	Канал розповсюдження (продакшн, бета-тест, внутрішнє тестування)
5	status	draft, inReview, published, halted	Поточний статус релізу або застосунку в консолі Google Play
6	country	US, UA, DE	Країна, для якої аналізуються показники (у випадку регіональної статистики)
7	installCount	120 540	Загальна кількість інсталяцій застосунку
8	activeDeviceCount	58 320	Кількість активних пристроїв з установленим застосунком
9	canceledSubscriptions	120	Кількість скасованих підписок (для застосунків з підписками)
10	updateTime	2025-01-18T12:45:00Z	Дата та час останнього оновлення застосунку в Google Play
11	rating	4.6	Середній рейтинг застосунку за відгуками користувачів
12	ratingsCount	8 930	Кількість залишених оцінок

Джерело:складено автором за [49].

Система контролю доступу (ACL) забезпечує гнучке та безпечне розмежування прав, охоплюючи як веб-панель, так і Telegram-бота. Завдяки ролям, middleware-перевіркам і централізованій логіці доступів було досягнуто високого рівня захисту адміністративних функцій та контрольованості внутрішніх процесів.

Інтеграція з Google Play Developer API доповнила загальну архітектуру, надавши можливість автоматично отримувати актуальні дані щодо встановлень, оновлень та статусу релізів. Це дозволило об’єднати технічну підтримку продуктів і бізнес-аналітику в єдиному інформаційному середовищі.

У сукупності всі реалізовані підсистеми формують надійну, безпечну та масштабовану платформу, здатну підтримувати життєвий цикл мобільних продуктів і забезпечувати ефективну взаємодію між адмініструванням, маркетингом та розробкою.

2.6. Забезпечення безпеки інтегрованої системи

Telegram-бот у створеній системі працює не лише як інтерфейс для взаємодії користувача, а й як інструмент адміністрування — він може виконувати дії, які впливають на дані у базі: додавання нових застосунків, зміна ролей користувачів, надсилання цільових повідомлень, формування аналітики та інше. Саме тому питання SQL-безпеки у випадку бота є не менш важливим, ніж для веб-панелі.

На відміну від браузерного інтерфейсу, Telegram-бот отримує дані безпосередньо від користувача у форматі:

- текстових повідомлень,
- параметрів команд (/add_user, /set_role, /broadcast),
- callback-даних із кнопок,
- аргументів URL-посилань.

Будь-який із цих каналів уразливий до SQL-ін’єкцій, якщо дані безпосередньо потраплятимуть у SQL-запит. Особливість Telegram-бота полягає

в тому, що зловмисникові навіть не потрібен доступ до веб-панелі — достатньо надіслати боту спеціально сформований текст.

Через те, що ваш bot-backend і веб-панель працюють з однією базою (спільні таблиці users, apps, logs), то:

- SQL-ін’єкція у боті може дати повний доступ до права адміністратора,
- зловмисник може змінити ACL,
- може отримати роль owner та увійти у веб-панель,
- може сфальсифікувати атрибуційні логи (logs),
- видалити або змінити дані менеджерів,
- зламати всю аналітику.

Telegram-бот — це "двері" до бази даних, і якщо вона недостатньо захищена, то зламати всю систему можна навіть без браузера.

У контексті Telegram-бота загроза SQL-ін’єкцій є ще більш критичною, ніж у веб-панелі, оскільки бот безпосередньо приймає текстові дані від користувача. У розробленій системі застосовано комплексну модель захисту: prepared statements, строгий контроль типів, введення whitelist-фільтрів, ACL-поля перевірки доступів та централізоване логування. Це забезпечує стійкість до SQL-ін’єкцій навіть у випадку, коли зловмисник безпосередньо взаємодіє з Telegram-ботом та намагається передати шкідливий код через команди чи callback-кнопки.

У Telegram-боті реалізовано захист від SQL-ін’єкцій наступним чином.

1. Всі SQL-запити виконуються виключно через PDO prepared statements

2. Якщо команда очікує ID — бот примусово приводить його до integer, що повністю нейтралізує SQL-ін’єкції у числових параметрах

3. Назви застосунків, повідомлення, коментарі проходять обмеження на довжину, заборону спеціальних символів, фільтрацію через filter_var()

4. Навіть якщо хтось намагається надіслати шкідливу команду, бот здійснює сегментацію прав доступу через ACL.

5. Усі помилки PDO логуються у таблицю logs, що дозволяє виявляти підозрілі параметри, повторювані спроби SQL-ін’єкцій, несанкціоновані команди.

Однією з критичних загроз для будь-якої веб-адміністраторської системи є атаки типу CSRF (Cross-Site Request Forgery) — *міжсайтове підроблення запиту*. Особливо небезпечними вони стають у випадках, коли система містить інструменти, що дозволяють змінювати дані додатків, створювати або видаляти записи, керувати посиланнями, переглядати логування, а також взаємодіяти з Telegram-ботом.

Telegram-бот сам не потребує CSRF-захисту, оскільки він не працює через браузер, отримує запити через Webhook або Long Polling та користувач не може "обманом" змусити бота виконати дію від третьої особи.

Але CSRF критично важливий у тому місці, де веб-панель надсилає команди боту (наприклад, тригер масової розсилки).

У випадку проєкту, що розглядається у даній роботі, веб-панель використовується менеджерами та адміністраторами для керування мобільними додатками, а тому будь-який несанкціонований запит може призвести до серйозних наслідків: видалення програм, підміни даних у таблицях, несанкціонованого створення посилань атрибуції, або навіть блокування облікових записів.

Саме тому у системі реалізовано повний механізм CSRF-захисту**,** заснований на використанні унікальних токенів сесії, які передаються разом з кожним критичним запитом. Це дозволяє забезпечити, щоб будь-яка дія у веб-панелі була виконана лише тим користувачем, який справді працює у системі, а не зовнішнім зловмисником.

На рисунку 2.17 наведено узагальнену схему процесу CSRF-перевірки у веб-панелі. Після завантаження форми сервер генерує випадковий CSRF-токен, зберігає його у сесії користувача та додає до форми як приховане поле. Під час надсилання POST-запиту токен повертається на сервер, де його значення порівнюється з токеном у сесії. У разі збігу запит вважається легітимним і

відповідна операція (створення, редагування чи видалення даних) виконується. Якщо токен відсутній або не збігається, запит блокується з поверненням коду помилки доступу.

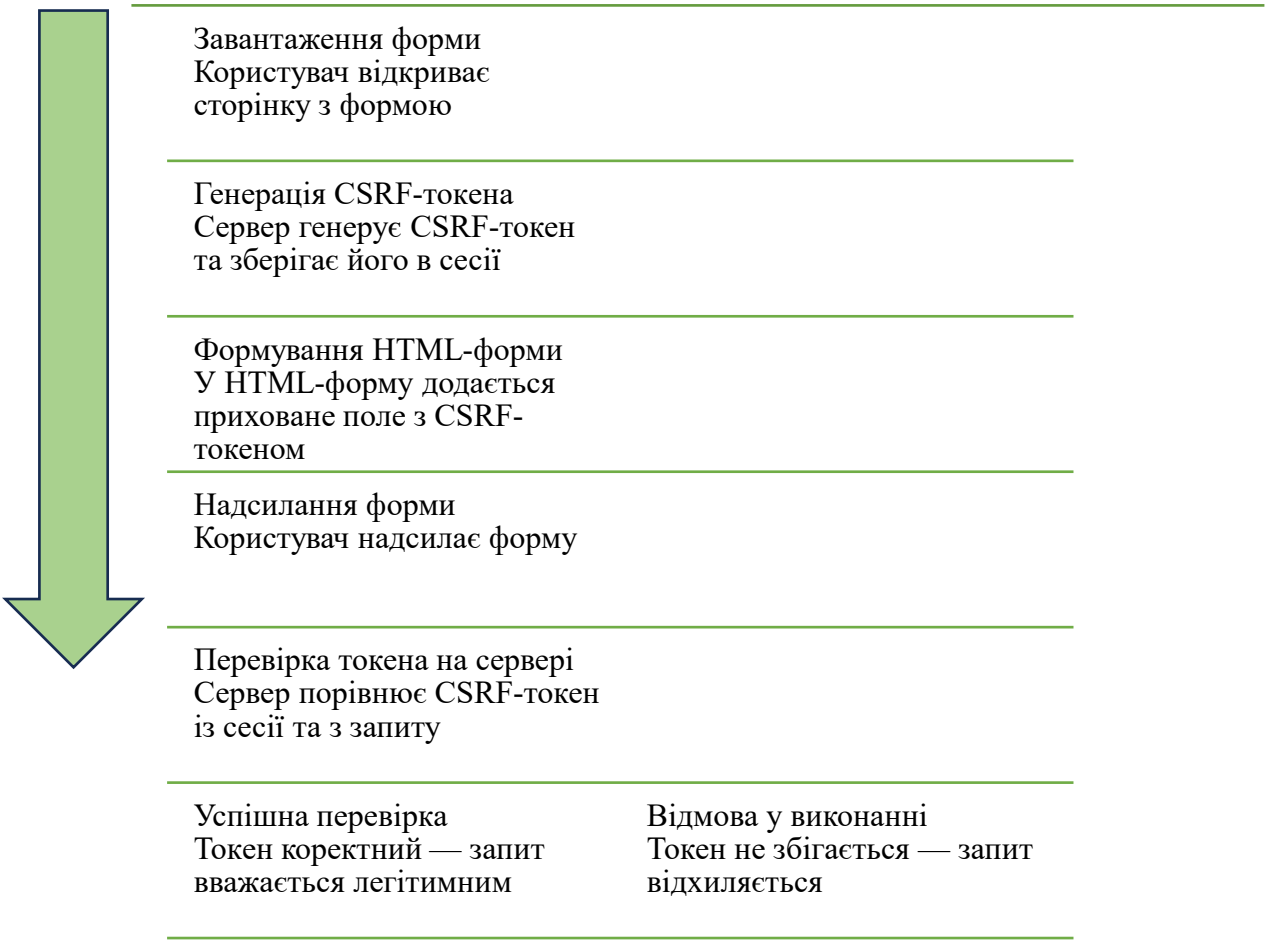


Рисунок 2.17 – Схема процесу CSRF-перевірки у веб-панелі адміністрування

Джерело: складено автором за [87].

Реалізація цього механізму в системі забезпечує високий рівень безпеки, а також робить неможливими сценарії, в яких зловмисник може змінити конфігурації, дані або взаємодію бота з API без дозволу адміністратора.

У системі також впроваджено ACL-підхід (Access Control List). Він дозволяє чітко визначати, які саме операції доступні різним ролям – адміністратору, менеджеру чи редактору. Наприклад, адміністратор може додавати нові застосунки, переглядати повні лог-файли та працювати з

атрибуційними посиланнями, тоді як менеджер отримує доступ лише до конкретних програм та обмеженого набору функцій. Завдяки цьому суттєво зменшується ризик випадкових або навмисних змін у критично важливих даних.

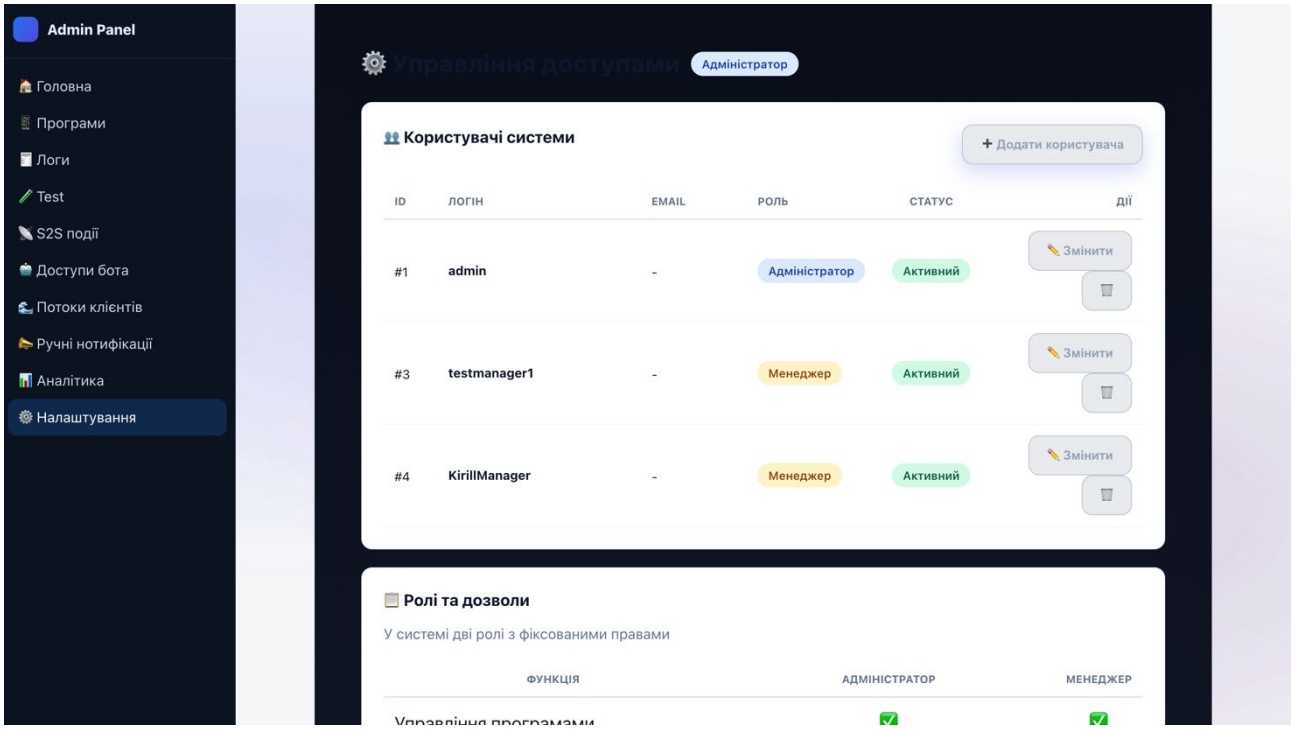


Рисунок 2.18 – Приклад впровадження ACL-підходу у панелі адміністратора

Особливістю системи є те, що ACL працює не лише у веб-панелі, а й у Telegram-боті. Під час кожної взаємодії бот перевіряє, чи прив’язаний Telegram-ID до облікового запису, яку роль має користувач та чи дозволено йому виконання відповідної команди. Це унеможливлює доступ сторонніх користувачів до адміністративних функцій навіть у випадку, якщо вони отримають пряме посилання або команду бота.

У рамках роботи інтегрована система активно взаємодіє з зовнішніми сервісами та внутрішніми модулями через API, webhook-події та S2S-взаємодію. Оскільки дані, що передаються між компонентами, мають високий рівень критичності (ідентифікатори користувачів, параметри атрибуції, службові

команди для Telegram-бота), безпека таких інтеграцій є обов’язковою умовою стабільної роботи всієї системи.

Одним із ключових механізмів є S2S-взаємодії (server-to-server), які використовуються під час обробки параметрів атрибуції, переходів через redirect.php та запису логу переходів у базу даних. Такий формат комунікації забезпечує передачу даних напряму між серверами без участі клієнтської частини, що унеможливлює їх підміну або модифікацію користувачем. S2S-запити доповнені валідацією параметрів, перевіркою сигнатур і обмеженням походження запитів, що мінімізує ризик фальсифікації атрибуційних подій.

Ще одним складником інтеграцій є webhook-механізми, які використовуються Telegram-ботом. Webhook дозволяє Telegram-платформі миттєво надсилати оновлення на визначений серверний endpoint. У системі використовується захищений webhook-URL, який приймає лише HTTPS-запити та містить у структурі унікальний секретний токен. Це гарантує, що сторонні сервіси не можуть підробити або надіслати небажані повідомлення боту. Обробка webhook-подій виконується через контролер, який проходить етапи фільтрації, логування та перевірки ACL, що запобігає виконанню команд неавторизованими користувачами.

Окрема частина інтеграцій пов’язана з використанням REST API усередині веб-панелі та допоміжних модулів. API застосовується для управління застосунками, генерації посилань, логування кліків, роботи з ACL та взаємодії з ботом. Усі API-ендпоїнти підтримують перевірку токена автентифікації та CSRF-токена (для POST-запитів), а відповіді стандартизовані за принципом JSON-формату. Таким чином значно спрощується використання API іншими модулями системи та сторонніми інструментами, зберігаючи при цьому необхідний рівень захисту.

Загалом інтеграційний рівень системи створений з урахуванням принципів secure-by-design: усі шляхи передачі даних проходять через шифрування HTTPS, жоден API-метод не працює без автентифікації, а webhook-механізм не приймає запити від невідомих джерел. Це дозволяє забезпечити безпечну та стійку роботу

всіх взаємопов’язаних компонентів, мінімізуючи ризики втручання або підробки даних під час комунікації між серверами.

2.7. Тестування та валідація системи

Тестування інтегрованої системи «веб-панель + Telegram-бот» є одним із ключових етапів її впровадження, оскільки саме на цьому етапі підтверджується коректність реалізованої логіки, стабільність роботи модулів, безпека передавання даних і зручність взаємодії користувачів з інтерфейсом. Оскільки система складається з кількох взаємопов’язаних компонентів, тестування проводилося комплексно: охоплюючи як серверну частину, так і клієнтський функціонал веб-панелі та Telegram-бота. Додаткову увагу приділено зручності використання інтерфейсу та поведінці системи в реальних сценаріях.

Під час перевірки безпеки системи особливий акцент був зроблений на виявленні потенційних загроз, які могли б виникнути через некоректну обробку користувацького введення або зовнішніх запитів. Наприклад, перевірка захисту від SQL-ін’єкцій включала спроби навмисно змінити параметри запитів під час входу, роботи з формами, фільтрами та переглядом елементів у панелі адміністратора. Було підтверджено, що всі критичні запити виконуються через підготовлені вирази, а тому система успішно блокує доступ до несанкціонованих даних. Під час тестування CSRF-захисту перевірялися форми редагування, додавання елементів та операції, які змінюють дані. Системою було правильно сформовано та перевірено CSRF-токени, що запобігло можливості неавторизованих запитів від зовнішніх сайтів.

Окремою частиною валідації стали функціональні тести, мета яких полягала у перевірці відповідності реальної роботи системи її вимогам. У веб-панелі тестувалися всі CRUD-операції: створення нових записів, редагування наявних, видалення даних та коректне їх відображення в інтерфейсі. Сюди зручно додати скріншот головного dashboard інтерфейсу, де відображаються ключові показники, кількість додатків, статистика переходів і елементи навігації.

Особливо важливою була перевірка коректності відображення модулів відповідно до ролі користувача, що підтверджує коректну роботу ACL. Для цього можна вставити скріншот сторінки входу або панелі менеджера, де видно, що певні модулі приховані або доступні лише для читання.

Telegram-бот тестувався у реальних сценаріях взаємодії з користувачем. Перевірялася коректність обробки команд, відповідність реакції бота очікуваній логіці, стабільність роботи webhook-механізму та відсутність затримок у відповідях. На цьому етапі доречно вставити скріншоти діалогів з ботом: наприклад, головного меню, повідомлення про помилку для неавторизованого користувача або приклад push-повідомлення. Це дозволить наочно продемонструвати роботу інтерфейсу, зручність спілкування та зрозумілість формулювань, що особливо важливо у дослідженнях, пов’язаних із UX.

На головній сторінці адміністративної панелі відображаються ключові системні метрики: загальна кількість додатків, кількість банів, установок, активних команд. Це дозволяє менеджерам швидко оцінити стан системи та оперативно реагувати на зміни. Блоки «Швидкий пошук додатків» та «Швидкі дії» забезпечують доступ до найважливіших функцій — додавання нового додатка, перевірки Google Play, надсилання повідомлень.

Архітектура інтерфейсу побудована так, щоб адміністратори могли аналізувати динаміку, контролювати робочі процеси та виявляти потенційні помилки. Цей екран виконує функцію «центру управління» та є основною точкою взаємодії користувачів із системою.

На головній сторінці адміністративної панелі відображаються ключові системні метрики: загальна кількість додатків, кількість банів, установок, активних команд. Це дозволяє менеджерам швидко оцінити стан системи та оперативно реагувати на зміни. Блоки «Швидкий пошук додатків» та «Швидкі дії» забезпечують доступ до найважливіших функцій — додавання нового додатка, перевірки Google Play, надсилання повідомлень.

Архітектура інтерфейсу побудована так, щоб адміністратори могли аналізувати динаміку, контролювати робочі процеси та виявляти потенційні

помилки. Цей екран виконує функцію «центру управління» та є основною точкою взаємодії користувачів із системою.

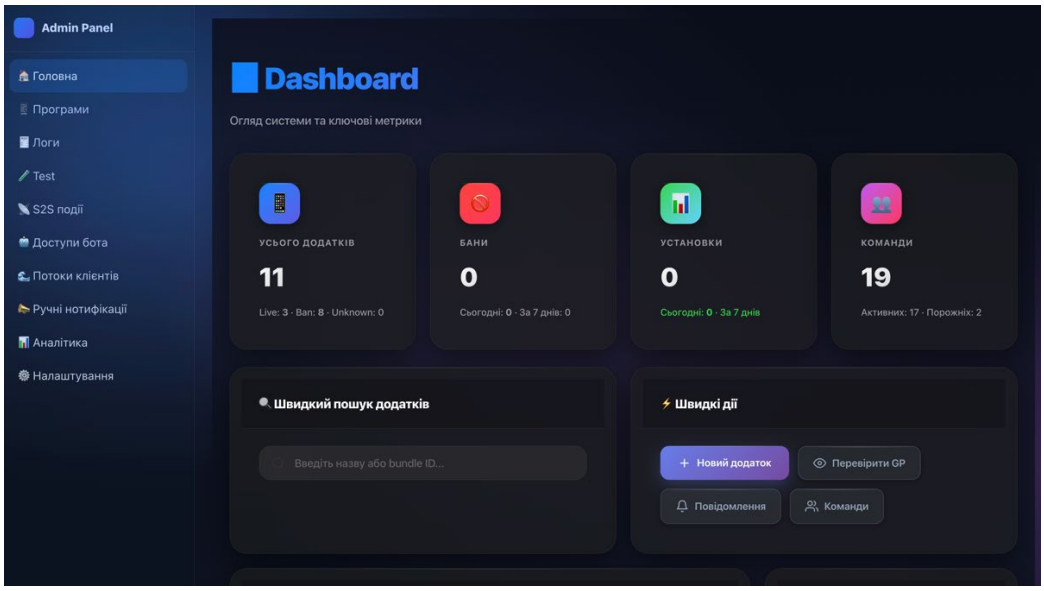


Рисунок 2.18 – Головна сторінка панелі адміністратора

Аналітичний розділ надає можливість переглядати дані щодо:

- кількості унікальних користувачів;
- джерел трафіку та ефективності каналів;
- конверсії атрибуції;
- періодичних інсталяцій;
- статистики за конкретними програмами.

Менеджери можуть обирати період, країну, джерело трафіку та конкретний мобільний продукт. Це дозволяє детально аналізувати якість залученого трафіку та приймати рішення щодо оптимізації рекламних кампаній.

Інтеграційні тести перевіряли взаємодію між компонентами — коли зміни у веб-панелі повинні викликати відповідні дії в Telegram-боті або коли бот надсилає дані на сервер, а система правильно їх обробляє, зберігає та відображає адміністратору. Наприклад, створення нового push-повідомлення у веб-панелі повинно відобразитися у Telegram негайно після відправки. У цьому розділі можна розмістити скріншот форми створення повідомлення у веб-панелі та

поруч — результат у Telegram-боті, що демонструє правильність роботи інтеграції.

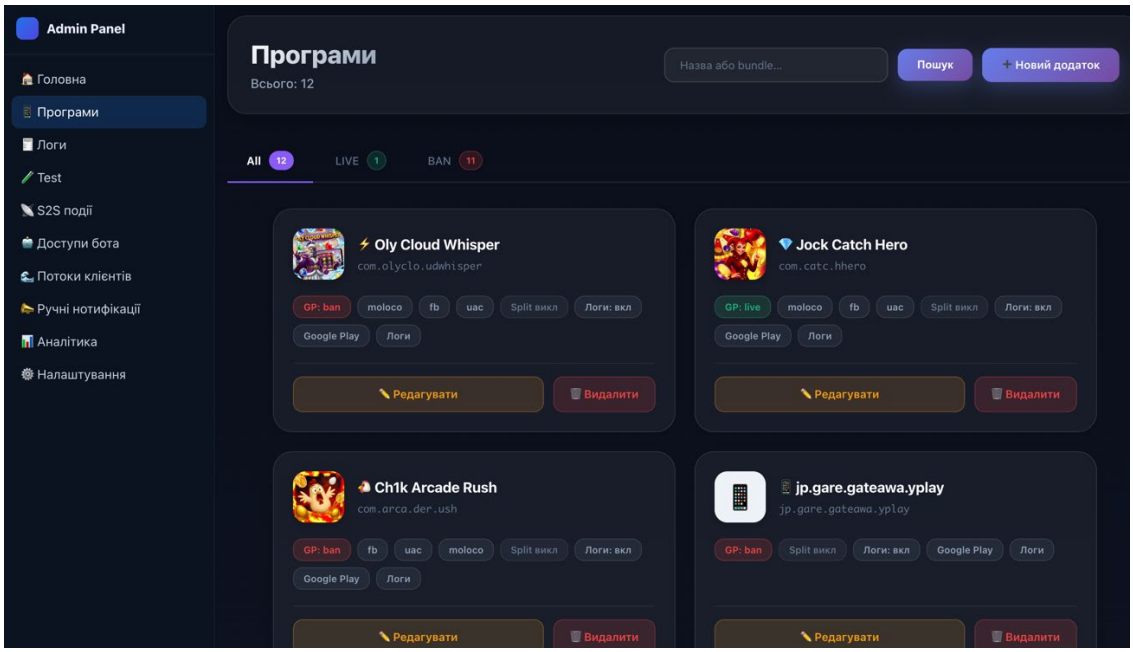


Рисунок 2.19 – Блок програм у панелі адміністратора

Під час тестування особлива увага приділялася зручності та інтуїтивності інтерфейсу. Для веб-панелі оцінювали, наскільки швидко користувач може знайти потрібні функції, як виглядають повідомлення про помилки, наскільки логічною є структура меню. Скріншот із прикладом повідомлення про помилку або успішне збереження даних буде доречно вставити саме в цьому місці. Telegram-бот також оцінювався з точки зору UX: чи є меню зрозумілим, чи достатньо інформативними відповіді, чи немає перевантаження коментарями або навпаки — браку інформації.

У ході тестування виявлено дрібні недоліки інтерфейсу, зокрема потребу покращити повідомлення про помилки та уточнити тексти в окремих командах. Проте загалом система продемонструвала високу стабільність, коректну роботу та здатність ефективно взаємодіяти у межах усіх компонентів. Отримані результати підтвердили відповідність системи вимогам безпеки, функціональності, інтеграцій та користувацького досвіду.

ВИСНОВКИ

У процесі виконання дипломної роботи було здійснено комплексне дослідження методів, архітектурних підходів та інструментів, необхідних для створення інтегрованої системи керування мобільними продуктами, що включає веб-панель адміністратора та Telegram-бот. Мобільні застосунки, як і сучасні платформи їх адміністрування, продовжують відігравати важливу роль у цифровому середовищі, забезпечуючи бізнесу оперативність, масштабованість та доступ до ключових аналітичних даних. Разом з тим зростає кількість вимог до безпеки, стабільності та якості таких рішень.

У роботі було сформульовано цілі дослідження та розроблено технічне завдання, що передбачало створення архітектурно цілісної системи з розмежуванням доступу, безпечним обміном даними та підтримкою повного циклу адміністрування мобільних продуктів. На основі вивчених підходів MVC, REST та принципів безпечного SDLC було запропоновано модель, що забезпечує стійкість проєкту до збоїв, логічну структуризацію компонентів та передбачувану масштабованість.

Особлива увага була приділена проєктуванню бази даних та реалізації механізмів взаємодії Telegram-бота з веб-модулем. Побудовано ER-діаграму, розроблено ключові таблиці (apps, teams, users, logs тощо), описано управління сесіями та створено механізми синхронізації даних між модулями.

У ході реалізації веб-панелі були розроблені CRUD-операції, API-ендпоїнти, моделі та контролери, що забезпечують керування додатками, командами та аналітикою. Запроваджено модуль Attribution links, який використовує redirect.php для логування переходів і визначення джерел установок. Система доступів реалізована через ACL, що дозволяє гнучко налаштовувати ролі користувачів і запобігає виконанню критичних операцій неавторизованими особами. Окремий модуль присвячений інтеграції з Google Play, що автоматизує перевірку статусу додатків і підвищує зручність роботи менеджерів.

Важливим результатом дослідження стало забезпечення безпеки інтегрованої системи. Були впроваджені механізми захисту від SQL-ін’єкцій (prepared statements, фільтрація даних), CSRF-токени для захисту форм адміністративної панелі, хешування паролів та контроль доступу через ACL. Розглянуто потенційні ризики взаємодії Telegram-бота з базою даних і реалізовано відповідні запобіжники.

На завершальному етапі було проведено функціональне, інтеграційне та безпекове тестування, що підтвердило коректність роботи системи, відповідність вимогам та стабільність у різних сценаріях навантаження.

Підсумовуючи, отримана система є цілісним інтегрованим рішенням, яке об’єднує сучасні підходи до архітектури, автоматизації адміністрування та забезпечення безпеки. Результати роботи можуть бути використані як база для подальшого розвитку та масштабування продукту, а також як методологічний приклад для створення аналогічних систем у сфері мобільної розробки. Створена інфраструктура є практичним інструментом, придатним як для роботи менеджерів, так і для технічних фахівців, та може бути адаптована під різні бізнес-завдання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 6 Useful Firebase Services for Mobile Application Development. Medium : web site. URL: https://zealousweb.medium.com/6-useful-firebase-services-for-mobile-application-development-88e5fa7f6a4d (дата звернення: 10.10.2025).
2. Атаки підробки міжсайтових запитів (CSRF). Hostragons : blog. URL: https://www.hostragons.com/uk/блог/атаки-підробки-міжсайтових-запитів-csrf/ (дата звернення: 10.10.2025).
3. Безпека веб-додатків: OWASP Top-10. Hostragons : blog. URL: https://www.hostragons.com/uk/блог/безпека-веб-додатків-owasp-топ-10/#OWASP_Nedir_ve_Neden_Onemlidir (дата звернення: 09.10.2025).
4. Захист від SQL-ін’єкцій у PHP. PHP Manual : documentation. URL: https://php.org.ua/manual/uk/security.database.sql-injection.md (дата звернення: 12.10.2025).
5. Зміни в алгоритмах App Store і Google Play. ASO.ua : web site. URL: https://aso.ua/zminy-v-alhorytmakh-app-store-i-google-play/ (дата звернення: 09.10.2025).
6. Ідентифікація, автентифікація та авторизація. Kryga : web site. URL: https://www.kryga.ua/blog/ident-avtent-avtor/ (дата звернення: 08.10.2025).
7. Моделі атрибуції у вебаналітиці. SEO-Академія : web site. URL: https://seo-akademiya.com/ua/baza-znan/web-analitika/shho-take-modeli-atribucziyi/ (дата звернення: 17.10.2025).
8. Найкращі інструменти інтеграції даних. Unite.AI : web site. URL: https://www.unite.ai/uk/data-integration-tools/ (дата звернення: 015.10.2025).
9. Обговорення Firebase та бекенду. DOU : web site. URL: https://dou.ua/forums/topic/41941/ (дата звернення: 15.10.2025).
10. Поняття ER-моделі та атрибутів. BestProg : web site. URL: https://www.bestprog.net/uk/2019/01/24/the-concept-of-er-model-the-

concept-of-essence-and-communication-attributes-attribute-types-ua/ (дата звернення: 18.10.2025).

11. Списки контролю доступу: принципи та застосування. EITCA Academy : web site. URL: https://url-shortener.me/2YH0 (дата звернення: 10.10.2025).
12. Схеми баз даних: приклади та пояснення. Foxminded : web site. URL: https://foxminded.ua/skhemy-bazy-danyh/ (дата звернення: 20.10.2025).
13. Топ інструментів для дослідження ринку мобільних додатків. ASOMOBILE : web site. URL: https://surl.li/spvrrt (дата звернення: 21.10.2025).
14. У 2024 році Google заблокував 158 000 акаунтів розробників за публікацію шкідливих додатків. DEV.ua : web site. URL: https://url-shortener.me/2YH5 (дата звернення: 25.10.2025).
15. Штучний інтелект у мобільній розробці: досвід, думки та перспективи. DOU : web site. URL: https://dou.ua/forums/topic/51117/ (дата звернення: 02.10.2025).
16. Що таке CRUD: функції, переваги та приклади. Highload.Tech : публікація. URL: https://surl.li/arauqz (дата звернення: 11.11.2025).
17. Що таке DevOps і чому він важливий. WEZOM : web site. URL: https://surl.li/xtldku (дата звернення: 10.10.2025).
18. Що таке MVC: переваги та недоліки. AVA Hosting : web site. URL: https://ava.hosting/uk/faq/what-is-mvc-advantages-and-disadvantages-of-mvc/ (дата звернення: 05.11.2025).
19. Що таке SDK і чому він важливий для розробників. WEZOM : web site. URL: https://surl.li/mqxnte (дата звернення: 03.11.2025).
20. Як змінився ринок мобільних додатків у 2024 році. UA Geek : web site. URL: https://uageek.media/article/6545/ (дата звернення: 04.11.2025).
21. Access Control Granularity Explained. Trio : web site. URL: https://www.trio.so/blog/access-control-granularity (дата звернення: 04.11.2025).

22. Access Control List (ACL): Concepts and Implementation. Imperva : knowledge base. URL: https://www.imperva.com/learn/data-security/access-control-list-acl/ (дата звернення: 02.11.2025).
23. Access Control List (ACL): Concepts. Imperva : web site. URL: https://www.imperva.com/learn/data-security/access-control-list-acl/ (дата звернення: 03.11.2025).
24. Access Control List (ACL): Definition & Usage. Sentra.io : web site. URL: https://www.sentra.io/cloud-data-security-glossary/access-control-list-acl (дата звернення: 03.11.2025).
25. Access Control Measures: Access Control Lists. SearchInform : web site. URL: https://searchinform.com/articles/cybersecurity/measures/access-control/access-control-list/ (дата звернення: 04.11.2025).
26. App Store Review Guidelines. Apple Developer : web site. URL: https://developer.apple.com/app-store/review/guidelines/ (дата звернення: 13.11.2025).
27. AppFigures – App Analytics and Market Insights. AppFigures : web site. URL: https://appfigures.com/ (дата звернення: 07.11.2025).
28. Apple App Store Integration. GitLab Docs : web site. URL: https://docs.gitlab.com/user/project/integrations/apple_app_store/ (дата звернення: 08.10.2025).
29. Authentication — Concepts and Techniques. W4AF Documentation : web site. URL: https://w4af.readthedocs.io/en/latest/authentication.html (дата звернення: 02.11.2025).
30. Authentication: Definition, Methods, Features. ZEN : web site. URL: https://www.zen.com/ua/blog/personal-finance-uk/authentication-definition-methods-features/ (дата звернення: 12.11.2025).
31. Backend as a Service (BaaS). GeeksForGeeks : web site. URL: https://www.geeksforgeeks.org/blogs/backend-as-a-service-baas/ (дата звернення: 13.11.2025).

32. Burp Scanner Documentation. PortSwigger : web site. URL: https://portswigger.net/burp/documentation/scanner (дата звернення: 12.11.2025).
33. Configuring Cloud Function Tasks. Google Cloud : web site. URL: https://docs.cloud.google.com/application-integration/docs/configure-cloud-function-task (дата звернення: 04.11.2025).
34. Create PWAs with PWABuilder. Microsoft Learn : web site. URL: https://learn.microsoft.com/uk-ua/training/paths/create-pwas-with-pwabuilder/ (дата звернення: 28.10.2025).
35. Cross-Site Request Forgery (CSRF): механізм атаки та захист. Corewin : web site. URL: https://corewin.ua/blog/cross-site-request-forgery-csrf/ (дата звернення: 23.10.2025).
36. Cross-Site Request Forgery (CSRF). MDN Web Docs : web site. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/CSRF (дата звернення: 25.11.2025).
37. CSRF атака: принцип роботи та способи запобігання. Avolutech : web site. URL: https://avolutech.com/blog/csrf-атака/ (дата звернення: 25.11.2025).
38. CSRF Attack: How It Works and Potential Consequences. EITCA Academy : web site. URL: https://surl.li/wmkubm (дата звернення: 28.11.2025).
39. Data Integration Tools & Techniques. Analytics8 : web site. URL: https://www.analytics8.com/blog/data-integration-tools-techniques/ (дата звернення: 17.11.2025).
40. DevOps Lifecycle — Principles and Best Practices. Unity : web site. URL: https://unity.com/topics/devops-lifecycle (дата звернення: 13.11.2025).
41. DevOps Practices and Principles. Atlassian : web site. URL: https://www.atlassian.com/devops (дата звернення: 12.11.2025).
42. ER Diagram Tutorial. Guru99 : web site. URL: https://www.guru99.com/uk/er-diagram-tutorial-dbms.html (дата звернення: 19.11.2025).

43. Firebase Cloud Functions Use Cases. Firebase : web site. URL: https://firebase.google.com/docs/functions/use-cases (дата звернення: 20.11.2025).
44. Firebase Installations API. Google Cloud Console : web site. URL:https://console.cloud.google.com/apis/library/firebaseinstallations.googleapis.com (дата звернення: 21.11.2025).
45. Firebase Overview. Firebase : official web site. URL: https://firebase.google.com/ (дата звернення: 28.11.2025).
46. Firebase Storage Documentation. Firebase : web site. URL: https://firebase.google.com/docs/storage (дата звернення: 16.11.2025).
47. Firestore vs Realtime Database. Estuary : web site. URL: https://estuary.dev/blog/firestore-vs-realtime-database/ (дата звернення: 24.11.2025).
48. Google Play will purge “low-quality” Android apps in latest store update. The Verge : web site. URL: https://www.theverge.com/2024/7/19/24201756/google-play-store-update-purge-low-quality-android-apps (дата звернення: 23.11.2025).
49. Google Play: Сторінка довідки для розробників. Google Play Console : web site. URL: https://support.google.com/googleplay/android-developer/answer/16549787?hl=uk_ALL (дата звернення: 24.11.2025).
50. How Do I Set Up Firebase Analytics for an Apple iOS App? SoCast Digital : web site. URL: https://support.socastdigital.com/portal/en/kb/articles/how-do-i-set-up-firebase-analytics-for-apple-ios-app (дата звернення: 22.11.2025).
51. How is using Synchronizer Token Pattern to prevent CSRF safe? StackOverflow : web site. URL: https://stackoverflow.com/questions/16049721/how-is-using-synchronizer-token-pattern-to-prevent-csrf-safe (дата звернення: 14.11.2025).
52. How We Kept the Google Play Android App Ecosystem Safe in 2024. Google Security Blog : web site. URL: https://security.googleblog.com/2025/01/how-we-kept-google-play-android-app-ecosystem-safe-2024.html (дата звернення: 14.11.2025).

53. Introduction to DevOps. GeeksforGeeks : web site. URL: https://www.geeksforgeeks.org/devops/introduction-to-devops/ (дата звернення: 19.10.2025).
54. Mobile Device Management (MDM). IBM Think : web site. URL: https://www.ibm.com/think/topics/mobile-device-management (дата звернення: 19.10.2025).
55. MVC Architecture — Building Scalable Web Applications. Medium : blog. URL: https://medium.com/@harshc0707/mvc-architecture-building-scalable-web-applications-a7dd55610583 (дата звернення: 20.10.2025).
56. MVC Architecture — System Design. GeeksforGeeks : web site. URL: https://www.geeksforgeeks.org/system-design/mvc-architecture-system-design/ (дата звернення: 25.10.2025).
57. MVC Architecture: Beginner to Expert Guide. Talent500 : web site. URL: https://talent500.com/blog/mvc-architecture-beginner-expert-guide/ (дата звернення: 27.10.2025).
58. New Product Development Cycle — Infographic Guide. Haltian: resources. URL: https://haltian.com/resources/new-product-development-cycle-infographic/ (дата звернення: 26.10.2025).
59. PDLC — Product Development Life Cycle Definition. Usersnap : web site. URL: https://usersnap.com/glossary/pdlc (дата звернення: 27.11.2025).
60. PDO:connect — встановлення з’єднання з базою даних. PHP Manual: web site. URL: https://www.php.net/manual/uk/pdo.connect.php (дата звернення: 28.10.2025).
61. Product Development Life Cycle — Full Guide. Dragonboat : web site. URL: https://dragonboat.io/blog/product-development-life-cycle/ (дата звернення: 28.10.2025).
62. Product Development Lifecycle — Contribution Guide. Carbon Design System: web site. URL: https://carbondesignsystem.com/contributing/product-development-lifecycle/ (дата звернення: 28.10.2025).

63. Realtime Database vs. Firestore. Firebase : web site. URL: https://firebase.google.com/docs/database/rtdb-vs-firestore (дата звернення: 29.10.2025).
64. Relationship Diagram Guide. MindOnMap : web site. URL: https://www.mindonmap.com/uk/blog/relationship-diagram/ (дата звернення: 30.10.2025).
65. REST (Representational State Transfer) — визначення та принципи. Komprise : web site. URL: https://www.komprise.com/glossary_terms/rest-representational-state-transfer/ (дата звернення: 20.11.2025).
66. Reverse Engineering: Concepts and Applications. Siemens Software : web site. URL: https://www.sw.siemens.com/en-US/technology/reverse-engineering/ (дата звернення: 21.11.2025).
67. SDLC vs PDLC: Unraveling the Differences in Development Life Cycles. Medium : web site. URL: https://medium.com/@dhiyaaizaaztiras/sdlc-vs-pdlc-unraveling-the-differences-in-development-life-cycles-c58fe992417c (дата звернення: 22.10.2025).
68. Secure SDLC: Best Practices. Snyk : web site. URL: https://snyk.io/articles/secure-sdlc/ (дата звернення: 23.11.2025).
69. Security Testing: SQL Injection. QATestLab : web site. URL: https://training.qatestlab.com/blog/technical-articles/security-testing-sql-injection/ (дата звернення: 23.10.2025).
70. Service Accounts in Firebase. Firebase : web site. URL: https://firebase.google.com/support/guides/service-accounts (дата звернення: 24.10.2025).
71. Software Development Life Cycle (SDLC) — Best Practices. SonarSource : web site. URL: https://www.sonarsource.com/resources/library/sdlc/ (дата звернення: 12.11.2025).
72. Software Development Life Cycle (SDLC). Atlassian : web site. URL: https://www.atlassian.com/agile/software-development/sdlc (дата звернення: 12.11.2025).

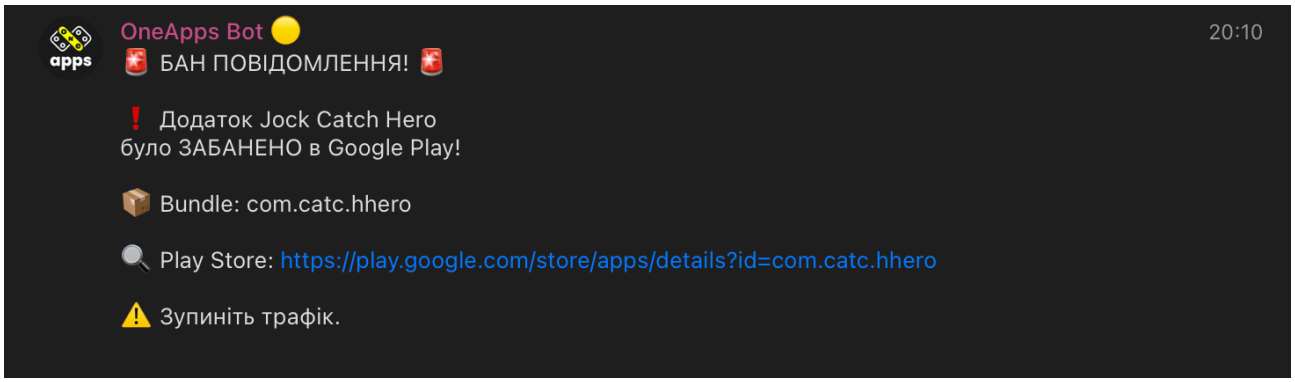
73. Software Development Life Cycle Explained. IBM : web site. URL: https://www.ibm.com/think/topics/sdlc (дата звернення: 10.11.2025).
74. SQL Commands Overview. Coursera : web site. URL: https://www.coursera.org/in/articles/sql-commands (дата звернення: 15.11.2025).
75. SQL Commands Overview. Coursera : web site. URL: https://www.coursera.org/in/articles/sql-commands (дата звернення: 14.11.2025).
76. SQL Injection: механізм атаки та запобігання. Acode : web site. URL: https://acode.com.ua/sql-injection/ (дата звернення: 15.10.2025).
77. SQL-ін’єкції: як працюють та як захищатися. Foxminded : web site. URL: https://foxminded.ua/sql-iniektsii/ (дата звернення: 15.11.2025).
78. Synchronizer Token Pattern. Medium : web site. URL: https://medium.com/@kaviru.mihisara/synchronizer-token-pattern-e6b23f53518e (дата звернення: 11.11.2025).
79. Telegram Bot Samples. Telegram : web site. URL: https://core.telegram.org/bots/samples (дата звернення: 15.11.2025).
80. The Product Development Life Cycle: Stages and Explanation. CareerFoundry: web site. URL: https://careerfoundry.com/en/blog/product-management/the-product-development-life-cycle/ (дата звернення: 13.11.2025).
81. Top Reasons for App Store Rejection: How to Avoid Being Rejected. Adapty : web site. URL: https://adapty.io/blog/app-store-rejection/?utm_source=chatgpt.com (дата звернення: 16.11.2025).
82. Understanding the Synchronizer Token Pattern. Cybrary : web site. URL: https://www.cybrary.it/blog/synchronizer-token-pattern (дата звернення: 16.11.2025).
83. WebSocket: основи та приклади. JavaScript.info : web site. URL: https://uk.javascript.info/websocket (дата звернення: 17.11.2025).

84. What Are Webhooks & How They Work? Hookdeck : web site. URL: https://hookdeck.com/webhooks/guides/what-are-webhooks-how-they-work (дата звернення: 16.11.2025).
85. What Is an Access Control List (ACL)? Delinea : web site. URL: https://delinea.com/what-is/access-control-list-acl (дата звернення: 17.11.2025).
86. What Is an Access Control List? Ping Identity : web site. URL: https://www.pingidentity.com/en/resources/blog/post/access-control-list.html (дата звернення: 17.11.2025).
87. What Is Cross-Site Request Forgery (CSRF)? SuperTokens : web site. URL: https://supertokens.com/blog/what-is-cross-site-request-forgery (дата звернення: 18.10.2025).
88. What Is DevOps? GitLab : web site. URL: https://about.gitlab.com/topics/devops/ (дата звернення: 19.11.2025).
89. What Is Model-View-Control (MVC)? Visual Paradigm : web site. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-model-view-control-mvc/ (дата звернення: 19.11.2025).
90. What Is Model-View-Controller (MVC)? Visual Paradigm : web site. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-model-view-control-mvc/ (дата звернення: 01.12.2025).
91. What is MySQL? Oracle : web site. URL: https://www.oracle.com/europe/mysql/what-is-mysql/ (дата звернення: 01.12.2025).
92. What Is SDLC and Why It Matters? Stfalcon : web site. URL: https://stfalcon.com/en/blog/post/SDLC-meaning (дата звернення: 01.12.2025).
93. What Is SDLC? A Complete Guide. GitHub Resources : web site. URL: https://github.com/resources/articles/what-is-sdlc (дата звернення: 28.11.2025).
94. What Is SDLC? Amazon Web Services (AWS) : web site. URL: https://aws.amazon.com/what-is/sdlc/ (дата звернення: 01.12.2025).

95. What Is Secure Software Development Life Cycle (SSDLC)? GeeksForGeeks : web site. URL: https://www.geeksforgeeks.org/ethical-hacking/what-is-secure-software-development-life-cycle-ssdlc/ (дата звернення: 02.12.2025).
96. What’s the Difference Between Cloud Firestore and the Firebase Realtime Database? StackOverflow : web site. URL: https://stackoverflow.com/questions/46549766/whats-the-difference-between-cloud-firestore-and-the-firebase-realtime-database (дата звернення: 19.11.2025).

ДОДАТКИ

ДОДАТОК А



Анотація

Цвид В. В. – Розробка захищеної інтегрованої системи адміністрування мобільних додатків. – Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 122 Комп'ютерні науки, освітньої програми Комп'ютерні науки та інформаційні технології. – Волинський національний університет імені Лесі Українки. – 2025 р.

У кваліфікаційній роботі досліджено теоретичні та практичні аспекти розробки захищеної інтегрованої системи адміністрування мобільних додатків в умовах сучасних цифрових екосистем. Актуальність теми обумовлена стрімким розвитком мобільних технологій, зростанням кількості мобільних застосунків та підвищенням вимог до їх безпеки, стабільності й відповідності політикам цифрових платформ Google Play та Apple App Store. Додатковим чинником актуальності є необхідність автоматизації процесів управління життєвим циклом мобільних продуктів і зменшення впливу людського фактору в діяльності IT-компаній.

У роботі проаналізовано сучасні підходи до забезпечення безпеки мобільних додатків, принципи управління їх життєвим циклом на основі моделей SDLC, PDLC та концепції DevSecOps. Розглянуто особливості застосування модульних архітектур і патерну Model–View–Controller у процесі створення безпечних і масштабованих систем адміністрування. Значну увагу приділено методам захисту вебпанелей адміністрування, контролю доступу, захисту API, а також протидії типових вебзагроз, що виникають у процесі експлуатації інтегрованих систем.

Практична частина роботи присвячена проєктуванню та реалізації захищеної інтегрованої системи адміністрування, що поєднує вебмодуль і Telegram-бот як інструмент автоматизації операційних процесів. Розроблена система забезпечує централізований моніторинг стану мобільних застосунків, оперативне інформування відповідальних осіб, управління даними та підтримку

процесів адміністрування в реальному часі. Для реалізації програмного рішення використано PHP, MySQL та Telegram Bot API.

Проведено тестування й валідацію функціональних можливостей системи, що підтвердило її працездатність, ефективність та відповідність вимогам інформаційної безпеки. Результати дослідження мають практичну цінність і можуть бути використані ІТ-компаніями для підвищення рівня захищеності мобільних продуктів, оптимізації процесів адміністрування та забезпечення стабільної роботи цифрових сервісів.

Ключові слова: мобільні додатки, інформаційна безпека, інтегрована система адміністрування, автоматизація бізнес-процесів, вебпанель адміністрування, життєвий цикл програмного забезпечення, MVC, DevSecOps, API.

Abstract

Tsvyd V. V. – Development of a Secure Integrated System for Mobile Application Administration. – Manuscript.

Qualification thesis submitted for the degree of **Master** in specialty **122 Computer Science**, educational program **Computer Science and Information Technologies**. – Lesya Ukrainka Volyn National University. – 2025.

The qualification thesis investigates the theoretical and practical aspects of developing a secure integrated system for administering mobile applications within modern digital ecosystems. The relevance of the research is driven by the rapid development of mobile technologies, the growing number of mobile applications, and the increasing requirements for their security, stability, and compliance with the policies of digital platforms such as Google Play and Apple App Store. An additional factor of relevance is the need to automate mobile product lifecycle management processes and reduce the impact of the human factor in the activities of IT companies.

The paper analyzes modern approaches to ensuring mobile application security and the principles of managing their lifecycle based on the SDLC and PDLC models, as well as the DevSecOps concept. The features of using modular architectures and the Model–View–Controller pattern in the development of secure and scalable administration systems are considered. Particular attention is paid to methods for protecting web-based administration panels, access control mechanisms, API security, and counteracting typical web threats that arise during the operation of integrated systems.

The practical part of the thesis is devoted to the design and implementation of a secure integrated administration system that combines a web module and a Telegram bot as a tool for automating operational processes. The developed system provides centralized monitoring of the status of mobile applications, prompt notification of responsible personnel, data management, and real-time support of administration processes. PHP, MySQL, and the Telegram Bot API were used to implement the software solution.

Testing and validation of the system's functional capabilities were carried out, confirming its operability, efficiency, and compliance with information security requirements. The research results have practical value and can be used by IT companies to enhance the security level of mobile products, optimize administration processes, and ensure the stable operation of digital services.

Keywords: mobile applications, information security, integrated administration system, business process automation, web administration panel, software lifecycle, MVC, DevSecOps, API.