Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

на тему: «Розробка системи автоматизованого функціонального та регресійного тестування веб-платформи»

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студентка гр. ІПЗ-22-2
_____ / Г. О. Пашкова /

Керівник
кваліфікаційної роботи _____ / А.М. Гриценко /

Завідувач кафедри _____ / А.М. Стрюк /

Кривий Ріг
2026

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А. М. Стрюк

«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студентці групи ІПЗ-22-2 Пашковій Ганні Олегівні

1. Тема: Розробка системи автоматизованого функціонального та регресійного тестування веб-платформи.

затверджено наказом по КНУ № 22 від «13» лютого 2026 р.

2. Термін подання студентом закінченої роботи: «09» червня 2026 р.

3. Вихідні дані по роботі: необхідно розробити вебсистему з реєстрацією, підтвердженням email, ролями, керуванням вебресурсами, ручними і плановими перевірками, аналізом HTTP, SSL, Lighthouse, SEO, JS-помилок, додаткових ресурсів, сценаріїв, логів, командної взаємодії та PDF/аналітичних звітів.

4. Зміст пояснювальної записки: виконати аналіз предметної області та аналогів; сформувати вимоги; спроєктувати архітектуру програмного забезпечення; розробити структуру даних; реалізувати вебсистему; описати особливості застосування; виконати тестування; підготувати висновки та ілюстративні матеріали.

5. Перелік ілюстративного матеріалу: UML-діаграми, архітектура, модель даних, схема ролей, процес перевірки, аналітичний зріз, індекс якості, знімки екранних форм діючого програмного забезпечення. Усі схеми подано у градаціях сірого відповідно до вимог оформлення.

Зворотна сторінка аркуша завдання на кваліфікаційну роботу

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання
1	Огляд літератури, методичних вимог і аналіз предметної області	лютий 2026 р.
2	Порівняльний аналіз існуючих рішень моніторингу та тестування вебсайтів	березень 2026 р.
3	Формування функціональних і нефункціональних вимог	березень 2026 р.
4	Проєктування архітектури, UML-моделей і структури даних	квітень 2026 р.
5	Реалізація модулів моніторингу, перевірок, логів і звітності	квітень – травень 2026 р.
6	Розробка інтерфейсу, профілю користувача, багатомовності та onboarding	травень 2026 р.
7	Тестування системи, перевірка безпеки доступу і виправлення помилок	травень 2026 р.
8	Підготовка пояснювальної записки, рисунків, таблиць і додатків	червень 2026 р.
9	Фінальне редагування, нормоконтроль і підготовка до захисту	червень 2026 р.

Дата видачі завдання: «15» лютого 2026 р.

Студентка _____ / Г. О. Пашкова /

Керівник роботи _____ / А. М. Гриценко /

РЕФЕРАТ

ВЕБСАЙТ, МОНІТОРИНГ, ТЕСТУВАННЯ, QA, DJANGO, ТЕХНІЧНИЙ АУДИТ, БРАУЗЕРНІ СЦЕНАРІЇ, АНАЛІТИКА, ЗВІТ, БЕЗПЕКА.

Пояснювальна записка: 68 с., 11 табл., 16 рис., 5 дод., 16 джерел.

Актуальність теми. Сучасні веб-платформи характеризуються високою динамічністю змін, складною клієнт-серверною архітектурою та критичною залежністю бізнесу від їхньої безперебійної працездатності. Проте процеси контролю якості та технічного аудиту часто залишаються фрагментованими: моніторинг доступності (uptime), технічний аналіз швидкодії сторінок, фіксація фронтенд-помилок та перевірка функціональних сценаріїв користувача виконуються різними ізольованими сервісами. Це створює суттєві незручності для невеликих веб-студій, QA-команд початкового рівня та розробників індивідуальних проєктів, які потребують єдиного, інтегрованого робочого простору без необхідності складного консольного налаштування або закупівлі дорогих промислових платформ.

Об'єктом дослідження є процес автоматизованого контролю якості, технічного аудиту та безперервного моніторингу технічного стану сучасних вебресурсів.

Предметом дослідження є архітектурні принципи, моделі даних, алгоритми та програмні засоби інтегрованої вебсистеми для комплексного ручного та машинного тестування вебсайтів.

Метою кваліфікаційної роботи є проєктування та програмна реалізація веборієнтованої системи BitGuard, яка забезпечує централізований плановий моніторинг доступності, запуск комплексних ручних і автоматизованих перевірок технічних показників вебсайтів, збереження глибокої історії вимірювань, підтримку командної взаємодії та генерацію структурованої аналітичної звітності.

Методи розробки та технології. Серверна частина системи реалізована на базі мови програмування Python із використанням високорівневого

вебфреймворку Django 6. Для виконання тривалих і ресурсомістких технічних перевірок (аудит продуктивності Lighthouse, виконання браузерних сценаріїв Playwright, HTTP/SSL валідація) без блокування основного інтерфейсу впроваджено архітектуру асинхронних фонових задач за допомогою розподіленої черги Celery та брокера повідомлень Redis. Клієнтська частина побудована з використанням шаблонізатора Django, компонентів Bootstrap та бібліотеки Chart.js для інтерактивної візуалізації зібраних метрик.

Отримані результати та їх новизна. Результатом роботи є діюча вебсистема BitGuard, яка об'єднує в єдиному вебкабінеті модулі моніторингу uptime, глибокого інфраструктурного аналізу (SSL-сертифікати, заголовки безпеки), оцінювання продуктивності й SEO за стандартами Lighthouse, а також підсистему трекінгу JavaScript-помилок у реальному часі. Розроблено гнучкий модуль машинного тестування, що дозволяє користувачам безпосередньо через інтерфейс конструювати послідовності дій користувача (кліки, введення, перевірка тексту) та аналізувати результати їх виконання з фіксацією логів і скріншотів помилок. Особливістю рішення є впровадження сторінки Analytics Pro для побудови індивідуальних аналітичних зрізів за допомогою фільтрації блоків даних , а також розробка строгих контурів безпеки: інтегровано механізм обмеження частоти запитів (rate limiting), валідацію публічності URL для запобігання SSRF-атакам та багаторівневу рольову модель керування доступом (RBAC) із обов’язковим адміністративним підтвердженням підвищених прав.

Практичне значення. Створений програмний продукт повністю готовий до експлуатації та орієнтований на використання у внутрішніх QA-процесах невеликих IT-компаній, цифрових агенцій, а також під час виконання навчальних студентських розробок. Система дозволяє суттєво оптимізувати часові витрати на первинну діагностику вебресурсів, покращити комунікацію в тріаді «менеджер – розробник – тестувальник» завдяки інтегрованому проєктному чату та спростити демонстрацію результатів аудиту через генерацію детальних структурованих PDF-звітів.

ABSTRACT

WEBSITE, MONITORING, TESTING, QA, DJANGO, TECHNICAL AUDIT, BROWSER SCENARIOS, ANALYTICS, REPORT, SECURITY.

Explanatory note: 68 pages, 11 tables, 16 figures, 5 appendices, 16 references.

Relevance of the topic. Modern web platforms are characterized by highly dynamic changes, complex client-server architecture, and critical dependence of business operations on their continuous availability and performance. However, quality control and technical audit processes often remain heavily fragmented: uptime monitoring, technical page speed analysis, frontend error tracking, and user scenario validation are typically performed by isolated, separate services. This creates significant inefficiencies for small web development studios, entry-level QA teams, and individual project developers who require a single, integrated workspace without the overhead of complex command-line configurations or purchasing expensive industrial APM platforms.

The object of the study is the process of automated quality control, technical auditing, and continuous state monitoring of modern web resources.

The subject of the study encompasses architectural principles, data models, algorithms, and software tools of an integrated web system for comprehensive manual and automated website testing.

The purpose of the qualification paper is the design and software implementation of the BitGuard web-based application, which provides centralized scheduled availability monitoring, launches comprehensive manual and automated audits of website technical indicators, accumulates deep measurement history, supports team collaboration, and generates structured analytical reports.

Development methods and technologies. The server side of the system is built using the Python programming language and the high-level Django 6 web framework. To handle long-running and resource-intensive technical evaluations (Lighthouse performance audits, Playwright browser scenario executions, HTTP/SSL validations) without blocking the primary user interface, an

asynchronous background task architecture has been implemented using the Celery distributed task queue and Redis message broker. The client side is designed using the Django template engine, Bootstrap components, and the Chart.js library for interactive visualization of collected metrics.

Results obtained and their novelty. The main outcome of this work is the fully functional BitGuard web system, which consolidates uptime monitoring modules, deep infrastructural audits (SSL validation, security headers), performance and SEO metrics following Lighthouse standards, and a real-time JavaScript runtime error tracking subsystem into a single web dashboard. A flexible automated testing module has been developed, allowing users to build step-by-step user interaction workflows (clicks, inputs, text assertions) directly through the UI and analyze execution results complete with execution logs and error screenshots. A key novelty of the solution lies in the introduction of the Analytics Pro page for custom analytical slicing via data block filtering , alongside the implementation of strict security perimeters: rate-limiting mechanisms, public URL validation to prevent SSRF attacks , and a multi-tiered Role-Based Access Control (RBAC) model requiring administrative approval for elevated permissions.

Practical significance. The created software product is fully production-ready and tailored for deployment within internal QA workflows of small IT companies, digital agencies, as well as individual student project developments. The system substantially optimizes time expenditures required for primary web resource diagnostics, enhances cross-functional communication within the "manager – developer – tester" triad via an integrated project chat , and simplifies audit result demonstration through the automated generation of detailed, structured PDF reports.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

Скорочення, які використовуються в пояснювальній записці, наведено в таблиці нижче.

Таблиця 0.1 – Перелік скорочень

Скорочення	**Пояснення**
API	інтерфейс програмування застосунків
CSS	каскадні таблиці стилів
HTTP	протокол передавання гіпертексту
JS	JavaScript, мова програмування браузерної частини
QA	забезпечення якості програмного забезпечення
RBAC	рольова модель керування доступом
SEO	оптимізація сторінки для пошукових систем
SSL	криптографічний протокол захищеного з'єднання
TTFB	час до першого байта відповіді сервера
Uptime	частка успішних перевірок доступності вебресурсу

ВСТУП

Сучасний вебсайт уже давно не є просто сторінкою з інформацією. Під час підготовки цієї роботи я виходила з дуже практичної ситуації: навіть невеликий сайт може одночасно мати проблеми з доступністю, повільним завантаженням, SSL-сертифікатом, JavaScript-помилками або битими посиланнями. Якщо перевіряти все це вручну різними інструментами, результат швидко стає розрізненим і незручним для пояснення керівнику, замовнику або членам команди.

Актуальність роботи пов'язана з тим, що перевірка вебсайту часто розкидана між різними інструментами: окремо перевіряють uptime, окремо продуктивність, окремо помилки в браузері, а потім вручну збирають висновки для звіту. Для студентського або невеликого робочого проєкту такий процес незручний, тому потрібен один зрозумілий вебінтерфейс.

Метою кваліфікаційної роботи є проєктування та розробка вебсайту BitGuard для моніторингу та ручного/машинного тестування вебсайтів.

У межах цієї кваліфікаційної роботи назва BitGuard використовується як умовна назва розробленої навчальної вебсистеми. Вона не пов'язана з однойменними сторонніми продуктами, компаніями або іншими згадками в мережі Інтернет. Таке уточнення потрібне для уникнення плутанини під час пошуку інформації та перевірки роботи.

Об'єктом дослідження є процес контролю якості вебресурсів. Предметом розробки є вебсистема, що забезпечує моніторинг доступності, запуск перевірок, накопичення результатів, аналіз технічних показників і підготовку звітів у зручному для користувача вигляді.

Для досягнення мети поставлено такі завдання: проаналізувати існуючі засоби моніторингу та тестування вебсайтів; визначити функціональні та нефункціональні вимоги; спроєктувати архітектуру вебзастосунку; реалізувати модулі керування проєктами, тестування, додаткових ресурсів, сценаріїв, аналітики, логів, звітів, профілю користувача, командної взаємодії та

налаштувань; забезпечити розмежування доступу до об'єктів користувачів і рольове підтвердження; підготувати інтерфейс з можливістю перемикання мов.

Галузь застосування результатів охоплює навчальні проєкти, невеликі вебстудії, внутрішні QA-процеси, студентські розробки та попередній технічний контроль вебсайтів перед демонстрацією або публікацією. Практичне значення роботи полягає у тому, що користувач отримує не набір окремих команд і сервісів, а один робочий простір, де можна додати сайт, запустити перевірку, переглянути історію та підготувати звіт.

Під час роботи над темою стало зрозуміло, що для користувача важливий не лише сам факт перевірки сайту, а й зрозуміле подання результату. Якщо система показує тільки набір технічних чисел, користувачеві все одно доводиться самостійно пояснювати, що саме сталося і наскільки це критично. Тому в роботі особливу увагу приділено не тільки збору показників, а й їх інтерпретації: статусам, рекомендаціям, логам, графікам і звітам, які можна використати під час обговорення з керівником, розробником або замовником.

Ще однією причиною вибору цієї теми стала потреба поєднати ручне і машинне тестування в одному середовищі. У реальній роботі QA-інженер часто виконує частину перевірок вручну, а частину намагається автоматизувати, але результати цих дій не завжди зберігаються в одному місці. Розроблена система має показати, як можна об’єднати додавання вебпроєкту, запуск перевірки, контроль додаткових ресурсів, сценарії користувача, історію запусків і підготовку звіту в межах одного вебзастосунку.

Окремо варто зазначити, що BitGuard розробляється як демонстраційно-практична система: вона має бути достатньо зрозумілою для захисту дипломної роботи, але водночас показувати реальні підходи до контролю якості вебресурсів, роботи з ролями, звітами та історією перевірок.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ НАПРЯМКУ РОЗРОБКИ

1.1 Актуальність розробки

Предметною областю роботи є контроль якості вебсайтів: чи відкривається сайт, як швидко він відповідає, чи чинний SSL-сертифікат, чи немає помилок JavaScript, чи доступні додаткові ресурси та чи можна швидко пояснити результат перевірки.

Розроблювана система має зібрати ці дії в одному місці. Вона не претендує на рівень великої корпоративної платформи, але для навчального проєкту, демонстрації на захисті та щоденного контролю невеликого сайту такий формат є достатнім і практичним.

1.2 Аналіз існуючих рішень

Для визначення напряму розробки розглянуто декілька груп інструментів: сервіси моніторингу доступності, інструменти аудиту продуктивності, засоби автоматизації браузерних сценаріїв та системи звітності [5-9]. Порівняння наведено в таблиці 1.1.

Під час порівняння аналогів я окремо звертала увагу не тільки на перелік функцій, а й на те, наскільки зручно ці інструменти можна показати в межах навчального проєкту. Наприклад, частина сервісів добре підходить для постійного моніторингу доступності, але не пояснює студенту логіку ручного тестування. Інші інструменти дають технічні метрики, проте не створюють єдиного робочого простору з проєктами, логами, ролями та звітами.

Саме через це в роботі було обрано не шлях копіювання одного готового сервісу, а поєднання кількох ідей: простота панелі моніторингу, ручний запуск перевірок, збереження історії, командна видимість, додаткові ресурси та формування зрозумілого звіту. Такий підхід краще відповідає темі дипломної роботи, тому що демонструє і програмну реалізацію, і сам процес контролю якості вебресурсу.

Результати порівняння існуючих рішень подано в табл. 1.1.

Таблиця 1.1 – Порівняння існуючих рішень

Рішення	**Основне призначення**	**Переваги**	**Обмеження для даної роботи**
UptimeRobot	Моніторинг доступності	Просте налаштування перевірок, повідомлення про падіння	Не орієнтований на ручні QA-сценарії, немає комплексного аналізу сторінки
Google Lighthouse	Аудит продуктивності, SEO та доступності	Дає технічні метрики якості сторінки	Потребує окремого запуску або інтеграції, не веде повну історію проєктів
Selenium IDE / Playwright	Автоматизація дій у браузері	Дозволяє моделювати користувацькі сценарії	Не є самостійною системою моніторингу та звітності
Pingdom	Моніторинг доступності та швидкодії	Розвинена аналітика і повідомлення	Комерційний характер і надлишковість для навчального вебзастосунку
BitGuard	Єдиний простір моніторингу і QA-перевірок	Поєднання ручного запуску, історії, метрик, ресурсів, сценаріїв і звітів	Розробляється в межах бакалаврської роботи, тому має обмежений масштаб порівняно з промисловими платформами

Проведений аналіз показує, що доцільним є створення системи, яка не копіює один конкретний аналог, а поєднує базові можливості декількох класів інструментів у межах одного вебінтерфейсу.

1.3 Вибір та обґрунтування напрямку рішення

Після порівняння аналогів обрано напрям створення вебсистеми, яка працює як єдина панель контролю якості сайту; підхід до оформлення результатів узгоджено з вимогами до звітів [1; 2]. Такий підхід є доцільним для бакалаврської роботи, оскільки дозволяє показати повний цикл розробки: від аналізу предметної області до реалізації моделей даних, інтерфейсу, фонових задач, звітності та захисту доступу.

Рішення не орієнтоване на заміну великих комерційних платформ. Його цінність полягає в іншому: користувач отримує компактний інструмент, де в одному місці зберігаються сайти, ручні перевірки, плановий моніторинг, додаткові ресурси, логи, рекомендації та звіти. Це зменшує кількість ручних дій і робить процес тестування зрозумілішим для людини, яка не хоче працювати з командним рядком.

Архітектурно обрано серверний вебзастосунок на Django, тому що цей фреймворк надає готові механізми маршрутизації, шаблонів, моделей, форм, авторизації та адміністративної частини [3]. Для довготривалих перевірок використано фонові задачі, а для візуалізації результатів - таблиці, індикатори, графіки та PDF-звіти.

Основні проєктні рішення описано текстово: вебінтерфейс обрано для зручного запуску перевірок без консолі; Django використано через ORM, авторизацію та швидку побудову форм; Celery винесено для фонових перевірок [4]; інструменти браузерного та технічного аудиту застосовано для збору технічних показників [5; 6]; локальні правила аналізу забезпечують зрозумілі висновки без розкриття службових інтеграцій.

Окремою причиною вибору саме такого напряму стала можливість поступово розширювати систему без повної зміни її архітектури. До вже

реалізованої основи можна додавати нові типи перевірок, додаткові графіки, сценарії тестування, інтеграції зі сповіщеннями або розширені звіти, не змінюючи головну ідею єдиного робочого простору.

1.4 Формування вимог до системи

Основні функціональні вимоги до системи сформовано в табл. 1.2.

Таблиця 1.2 – Основні функціональні вимоги

Вимога	Очікуваний результат
Користувач може зареєструватися, увійти та підтвердити email	Профіль має стан підтвердження, а користувач отримує доступ до робочого простору
Користувач може додавати, редагувати та видаляти вебпроєкти	Система зберігає назву, URL, профіль перевірки, інтервал моніторингу та відповідальних осіб
Система виконує ручну і планову перевірку сайту	Оновлюються статус, час відповіді, SSL, Lighthouse-метрики і т. д.
Користувач може додавати додаткові ресурси	CSS, sitemap, API або інші URL перевіряються разом із основним сайтом
Система підтримує сценарії машинного тестування	Користувач створює кроки, запускає сценарій і переглядає результат виконання
Система має рольову модель	QA, Developer, Manager і Viewer отримують різні можливості, а підвищені ролі потребують підтвердження
Користувач бачить лише свою команду та причетних учасників	Сторінка команди і чат не відкривають сторонніх користувачів

Продовження Таблиці 1.2

Вимога	Очікуваний результат
Analytics Pro формує вибірковий зріз даних	Користувач обирає сайт і типи даних, а система показує лише доступні фактичні метрики
Система формує PDF-звіти	Користувач обирає блоки звіту і отримує структурований документ для демонстрації або аналізу

Нефункціональні вимоги, що визначають якість і безпечність системи, наведено в табл. 1.3.

Таблиця 1.3 – Нефункціональні вимоги

Характеристика	Вимога	Метрика
Функціональна придатність	Основні сценарії мають виконуватися через вебінтерфейс	Додавання сайту, запуск перевірки, лог, аналітика, PDF і профіль виконуються без консолі
Безпека	Захист об'єктів користувача, ролей і секретів	Доступ через accessible_projects; URL validation; .env; маскування ключів; журнал SecurityAuditLog
Надійність	Система зберігає деталі навіть при помилці перевірки	Кожен запуск створює TestResult з severity і error_message
Зручність	Користувач має швидко зрозуміти стан сайту	Статус, uptime, рекомендації, профільний індекс і Analytics Pro доступні в інтерфейсі

Продовження Таблиці 1.3

Характеристика	Вимога	Метрика
Супроводжуваність	Код має бути поділений на моделі, форми, задачі, представлення і шаблони	Окремі файли models.py, forms.py, views.py, tasks.py, security.py, templates
Сумісність	Інтерфейс має працювати на різних розмірах екрана	Сторінки використовують responsive-сітки Bootstrap і власні CSS-обмеження

Наведені таблиці використовуються як основа для подальшого проєктування, оскільки саме з них випливають моделі даних, сценарії використання, перевірки доступу та критерії приймання готової системи. Завдяки цьому вимоги не залишаються формальним описом, а напряму пов’язуються з реалізованими модулями BitGuard.

1.5 Зацікавлені сторони, межі та припущення розробки

Для коректного опису предмета розробки важливо визначити не лише функції системи, а й коло осіб, які зацікавлені у її використанні. У межах цієї роботи система розглядається як навчально-прикладний продукт, призначений для студентів, QA-фахівців початкового рівня, веброзробників і керівників невеликих вебпроєктів.

Основним користувачем системи є людина, яка має вебсайт або декілька вебресурсів і хоче періодично перевіряти їх стан без складного налаштування промислових сервісів. Для такого користувача важливо, щоб інтерфейс був зрозумілим, перевірка запускалася кнопкою, а результат подавався у вигляді коротких статусів, логів і структурованого звіту.

Межі системи визначено таким чином: BitGuard виконує базовий моніторинг і тестування вебресурсів, зберігає результати, формує висновки, підтримує рольову модель та командну роботу, але не бере на себе функції

повноцінної корпоративної APM-платформи або системи управління інцидентами. Такий масштаб відповідає темі бакалаврської роботи і дозволяє показати завершений програмний продукт.

Стейкхолдери згруповані за впливом: керівник роботи та демонстраційний замовник мають високий вплив на повноту функцій; QA Engineer і студент-розробник мають високий інтерес до щоденного використання; Developer та Project Manager впливають на вимоги до команд, ролей і звітів; зовнішні сервіси можуть обмежувати перевірки через доступність API або політики сайтів.

Межі системи зручніше описати як коротке правило: BitGuard допомагає побачити проблему, зафіксувати її та підготувати звіт, але не виконує роль великої корпоративної платформи управління інцидентами.

До системи входить додавання сайтів, ручний запуск перевірки, плановий моніторинг, аналіз HTTP/SSL та показників продуктивності, перевірка додаткових ресурсів, історія запусків, рекомендації, PDF-звіти, профілі користувачів, ролі, підтвердження email, командна сторінка, проєктний чат і налаштування доступу.

Поза межами системи автоматичне виправлення знайдених помилок, повноцінна заміна CI/CD або APM-платформи, глибокий аудит чужого вихідного коду без доступу до репозиторію та промислове управління великими організаціями.

1.6 Ризики предметної області та обмеження рішення

Предметна область моніторингу вебсайтів має низку ризиків, які необхідно враховувати під час розробки. Частина ризиків пов'язана з технічною нестабільністю зовнішніх ресурсів: сайт може тимчасово не відповідати, блокувати автоматизовані запити, повільно завантажуватися або повертати різні результати залежно від мережевих умов.

Інша група ризиків стосується безпеки. Якщо система виконує запити за URL, який вводить користувач, потрібно не допустити небажаного звернення

до локальних, службових або приватних адрес. Саме тому у проєкті передбачено перевірку URL та обмеження доступу до об'єктів за поточним користувачем.

Також слід враховувати обмеження інструментів аналізу. інструменти браузерного аудиту дають корисні технічні показники, але результати можуть відрізнятися залежно від версії браузера, швидкості мережі, стану цільового сервера і параметрів запуску. Тому система подає оцінки як допоміжний інструмент, а не як абсолютний вирок якості сайту.

1.7 Опис основних ризиків предметної області

Окремо враховано ризики, пов’язані з інструментами технічного аналізу. Наприклад, помилки модуля аудиту продуктивності або відсутність окремих метрик не блокують інші етапи перевірки, оскільки HTTP, SSL, JavaScript-помилки, додаткові ресурси та сценарії можуть бути проаналізовані незалежно. Секретні параметри винесено у файл .env, URL проходить перевірку безпечності, а доступ до чужих об’єктів обмежено через вибірку доступних проєктів. Якщо звіт сформовано на основі неповних або застарілих даних, користувач може виконати новий запуск і повторно сформувати результат.

Також існує ризик людської помилки, коли користувач вводить неправильну адресу сайту, обирає невідповідний профіль перевірки або запускає аналіз без достатньої кількості попередніх даних. Для зменшення цього ризику в системі передбачені підказки, валідація форм і повідомлення про те, які дії потрібно виконати перед формуванням аналітики. Важливим є і ризик надмірного навантаження на цільовий вебресурс, тому активний моніторинг має виконуватися з контрольованим інтервалом. Додатково система зберігає історію запусків, що дозволяє відрізняти одиничний збій від повторюваної проблеми.

Додатково ризики предметної області можна поділити на декілька груп, оскільки вони виникають не тільки в програмному коді, а й у самій природі перевірки зовнішніх вебресурсів:

- *технічні ризики* – сайт може бути тимчасово недоступним, повільно відповідати, блокувати автоматизовані запити або повертати нестабільні результати;
- *ризики даних* – частина метрик може бути відсутня, застаріла або отримана під час короткочасного збою;
- *ризики безпеки* – користувач може ввести небезпечну URL-адресу, а система повинна не допустити звернення до локальних чи службових ресурсів;
- *організаційні ризики* – різні учасники команди можуть мати різні права доступу, тому важливо не відкривати чужі проєкти всім користувачам;

У межах розробленої системи ці ризики не усуваються повністю, оскільки частина з них залежить від зовнішніх сайтів, мережі або дій користувача. Проте система зменшує їхній вплив: фіксує помилки в логах, не зупиняє всю перевірку через збій одного модуля, обмежує небезпечні URL, зберігає історію запусків і показує користувачу пояснення замість “сирих” технічних даних.

Ще одним важливим ризиком є залежність результатів перевірки від умов, у яких вона виконується. Один і той самий сайт може показувати різний час відповіді залежно від швидкості мережі, навантаження на сервер, тимчасових обмежень з боку хостингу або роботи сторонніх сервісів. Тому результати окремого запуску не варто сприймати як остаточний висновок про якість вебресурсу. Саме для цього в системі зберігається історія перевірок, яка дозволяє оцінювати не випадкове значення, а загальну тенденцію.

Якщо після останньої перевірки сайт було змінено, оновлено або перенесено на інший сервер, старі результати вже не повністю відображають його поточний стан. Тому система має заохочувати користувача періодично повторювати перевірки, особливо перед демонстрацією, релізом або формуванням підсумкового звіту.

2 ПРОЄКТУВАННЯ ВЕБСИСТЕМИ BITGUARD

2.1 Призначення та сценарії використання

BitGuard призначена для користувача, який хоче контролювати стан одного або декількох вебресурсів через браузерний інтерфейс. Типовий сценарій передбачає реєстрацію, додавання сайту, запуск перевірки, перегляд результатів, аналіз рекомендацій і формування звіту.

Загальну взаємодію ролей із системою показано на рис. 2.1.

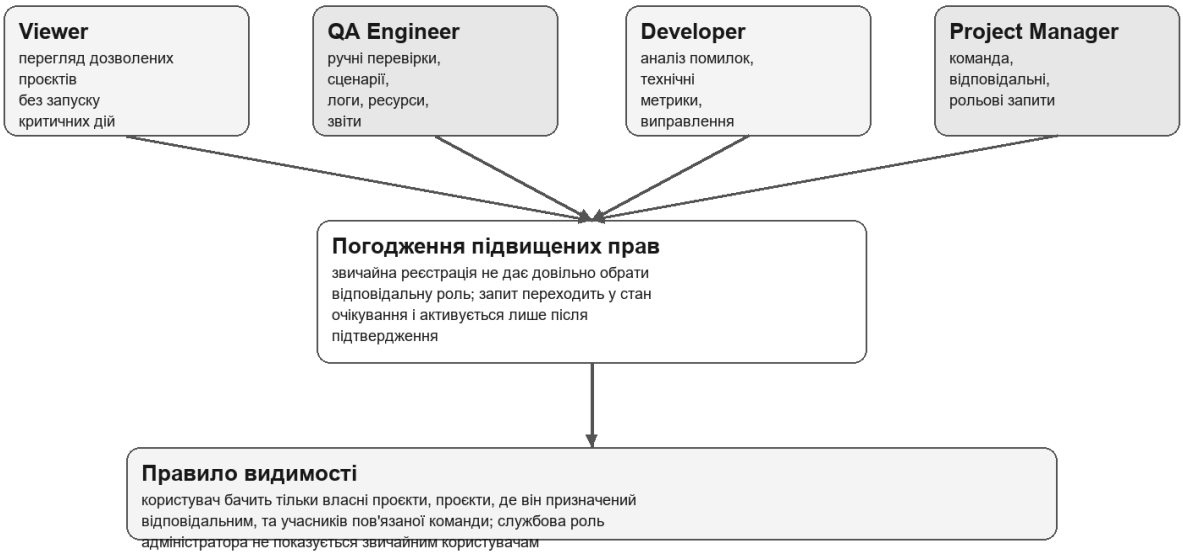

Рисунок 2.1 – UML-діаграма варіантів використання з рольовою моделлю

UML-діаграма варіантів використання показує, які основні дії можуть виконувати різні типи користувачів у системі BitGuard. Вона допомагає зрозуміти, що вебзастосунок розрахований не лише на одного користувача, а на роботу кількох ролей із різним рівнем доступу.

Звичайний користувач може зареєструватися, підтвердити email, переглядати доступні проєкти та результати перевірок. QA Engineer має ширші можливості: запускати перевірки, створювати сценарії тестування, переглядати логи та аналізувати помилки. Developer використовує систему переважно для технічного аналізу проблем, перегляду метрик, ресурсів і результатів

сценаріїв. Project Manager працює з командою, відповідальними особами, звітами та загальним станом проєктів.

Окремо на діаграмі відображено рольове погодження. Це означає, що користувач не може довільно отримати розширені права одразу після реєстрації. Підвищена роль має бути підтверджена відповідальною особою, що зменшує ризик несанкціонованого доступу до чужих проєктів або налаштувань.

2.2 Методика виконання роботи

Методика виконання роботи була побудована як послідовність практичних етапів. Спочатку визначено проблему та цільову аудиторію, після чого проаналізовано аналоги й сформовано вимоги. Далі виконано проєктування структури даних, архітектури, користувацьких сценаріїв та інтерфейсів.

Під час реалізації робота поділялася на функціональні модулі: авторизація, моніторинг, запуск перевірок, додаткові ресурси, машинні сценарії, аналітика, логи, PDF-звіти, профіль користувача, багатомовність та onboarding. Після кожного блоку виконувалася ручна перевірка поведінки системи через браузер.

Для оцінювання якості вебресурсу використано поєднання фактичних технічних показників і локальних правил інтерпретації. Система аналізує статус відповіді, час відповіді, SSL, помилки JavaScript, додаткові ресурси та оцінки продуктивності. На основі цих даних формуються рекомендації та інтегральний висновок.

1. Аналіз. Спочатку було визначено проблему, цільову аудиторію, аналоги та вимоги до майбутньої системи

2. Проєктування. Після цього сформовано сценарії роботи, моделі даних, архітектуру, UML-діаграми та структуру інтерфейсу

3. Реалізація. Далі розроблено модулі Django, фонові задачі Celery, форми, шаблони, API та звітність

4. Перевірка. Після реалізації виконувались ручні перевірки сценаріїв, логів, звітів, доступу до об'єктів і поведінки інтерфейсу

5. Оформлення. На завершальному етапі підготовлено пояснювальну записку, рисунки, місця під скріншоти та матеріали для захисту

2.3 Архітектура програмного забезпечення

Система реалізована як вебзастосунок на основі Django. У структурі проєкту виділено окремий додаток core, у якому зібрано основну логіку роботи системи: моделі даних, маршрути, форми, представлення, перевірки доступу, сторінки інтерфейсу, запуск тестування, роботу з аналітикою, логами та звітами. Такий поділ робить проєкт зрозумілішим, оскільки головні функції системи не розкидані хаотично, а згруповані в одному функціональному модулі.

Серверна частина відповідає за обробку запитів користувача відповідно до архітектурного підходу, який підтримує Django [3]. Коли користувач додає сайт, відкриває сторінку моніторингу, запускає перевірку або формує PDF-звіт, запит спочатку надходить до Django. Далі система перевіряє, чи користувач авторизований, чи має він доступ до потрібного проєкту, обробляє введені дані, звертається до бази даних і повертає результат у вигляді HTML-сторінки або API-відповіді.

Клієнтська частина побудована на HTML-шаблонах Django, CSS, Bootstrap-компонентах і JavaScript; Bootstrap використано як основу для типових компонентів інтерфейсу [15]. Вона відповідає за те, як користувач бачить систему: таблицю моніторингу, кнопки запуску, модальні вікна, підказки, графіки, форми, панелі налаштувань і повідомлення про стан перевірки. JavaScript використовується для інтерактивних дій, наприклад оновлення стану запуску, відкриття меню, роботи з графіками або показу елементів без повного перезавантаження сторінки.

Оскільки перевірка вебресурсу може тривати довше, ніж звичайний запит до сторінки, такі операції винесено у фонові задачі Celery [4]. Це стосується перевірки доступності сайту, SSL-сертифіката, додаткових ресурсів, браузерного аналізу, сценаріїв тестування та формування окремих результатів.

Завдяки цьому інтерфейс не зависає: користувач натискає кнопку запуску, а система виконує перевірку у фоні та після завершення оновлює збережені дані. Для черги повідомлень у такій архітектурі може використовуватися Redis [13].

Для планового моніторингу використовується celery-beat. Він дозволяє запускати перевірки не тільки вручну, а й за заданим інтервалом. Наприклад, сайт може перевірятися кожні кілька хвилин або годин, а результати кожного запуску зберігаються у базі даних. Надалі ці дані використовуються для таблиці моніторингу, логів, графіків, Analytics Pro і PDF-звітів.

Таким чином, архітектура системи складається з трьох основних рівнів: інтерфейсу користувача, серверної логіки Django та фонового виконання перевірок. Такий підхід дозволяє поєднати зручність вебінтерфейсу, збереження історії роботи системи та виконання технічно складних операцій без блокування користувача.

2.4 Структура даних та бази даних

У системі виділено декілька основних сутностей. Project описує сайт, що перевіряється; TestResult зберігає історію запусків; ProjectResource відповідає за додаткові файли та ресурси; UserProfile зберігає роль, запитану роль, статус підтвердження, аватар, стан onboarding, Telegram та email verification; SecurityAuditLog фіксує важливі дії; ProjectChatMessage забезпечує обговорення в межах проєкту; TestScenario, ScenarioStep і ScenarioResult описують сценарії машинного тестування.

Перелік головних моделей даних та їх призначення подано в табл. 2.1.

Таблиця 2.1 – Основні моделі даних

Модель	Призначення	Ключові дані
Project	Вебресурс користувача	Назва, URL, статус, профіль перевірки, інтервал, відповідальні, SSL, Lighthouse, SEO, runtime, ресурси

Продовження Таблиці 2.1

Модель	Призначення	Ключові дані
TestResult	Результат запуску перевірки	Дата, код статусу, час відповіді, успішність, критичність, деталі помилки
ProjectResource	Додатковий ресурс проєкту	Назва, URL, HTTP-статус, тип контенту, розмір, час відповіді, помилка
UserProfile	Розширення акаунта користувача	Роль, запитана роль, статус погодження, аватар, bio, Telegram, email verification, onboarding
SecurityAuditLog	Журнал важливих дій	Користувач, проєкт, дія, IP, user-agent, деталі, час
ProjectChatMessage	Командне обговорення проєкту	Проєкт, автор повідомлення, текст, дата створення
TestScenario	Сценарій машинного тестування	Проєкт, назва, опис, передумови, версія, очікуваний результат, ознака планового запуску
ScenarioStep	Окремий крок сценарію	Порядок, дія, селектор, значення
ScenarioResult	Результат запуску сценарію	Статус, фактичний результат, логи, скріншот помилки
ReportVersion	Версія сформованого аналітичного звіту	Проєкт, джерело, текст звіту, оцінка якості, дата

База даних використовується не лише для збереження довідкової інформації про сайти, а й для накопичення історії вимірювань. Завдяки цьому

система може показувати не тільки поточний стан, але й динаміку якості, повторювані помилки та результативність виправлень.

2.5 Захист доступу та обмеження ризиків

Оскільки система працює з користувацькими об'єктами, важливо не допустити доступу до чужих проєктів, ресурсів, сценаріїв, логів, чатів і звітів. У представленнях застосовано вибірку об'єктів з урахуванням доступних проєктів користувача, а дії редагування додатково перевіряються через роль і статус підтвердження.

Окрему увагу приділено роботі з секретами відповідно до загальних рекомендацій веббезпеки [10; 11]. Приватні ключі та службові параметри мають зберігатися у файлі .env, який не потрапляє до репозиторію. У профілі користувача чутливі значення відображаються лише у маскованому вигляді, а дії експорту та змін фіксуються у журналі безпеки.

Для URL-адрес передбачена перевірка публічності ресурсу, що знижує ризик небажаних запитів до локальних або службових адрес. Такий підхід важливий для вебзастосунків, які виконують HTTP-запити за URL, введеними користувачем.

Модель доступу доповнено рольовим погодженням, що узгоджується з принципом обмеження прав доступу [12]. Під час реєстрації користувач може запросити роль, однак права на редагування і керування проєктами надаються лише після підтвердження відповідальною особою. Це зменшує ризик, що випадковий користувач самостійно призначить собі підвищені можливості.

Фрагмент 2.1 – Обмеження видимості проєктів за користувачем:

```
 def accessible_projects(user):
if user.is_superuser:
    return Project.objects.all()
return Project.objects.filter(
    Q(user=user) |
    Q(manager=user) |
    Q(qa_owner=user) |
    Q(developer_owner=user)
).distinct()
```

Цей фрагмент показує, що доступ до проєкту визначається не лише власником, а й відповідальними учасниками команди. Завдяки цьому користувач не бачить чужі сайти, але може працювати з тими ресурсами, до яких його призначено.

Фрагмент 2.2 – Перевірка безпечності URL перед запуском моніторингу:

```
    def validate_public_url(url):
    parsed = urlparse(str(url or '').strip())
    if parsed.scheme not in {'http', 'https'}:
        raise ValidationError('Дозволені тільки http та https
URL.')
    if parsed.hostname in {'localhost',
'localhost.localdomain'}:
        raise ValidationError('Локальні адреси заборонені.')
    for address in socket.getaddrinfo(parsed.hostname, None,
type=socket.SOCK_STREAM):
        ip = ipaddress.ip_address(address[4][0])
        if not ip.is_global:
            raise ValidationError('URL веде на приватну або слу-
жбову адресу.')
```

Перевірка URL потрібна тому, що система виконує мережеві запити за адресами, введеними користувачем. Обмеження локальних і приватних адрес зменшує ризик небажаного доступу до внутрішніх ресурсів.

Модель погодження ролей і командної видимості подано на рис. 2.2.

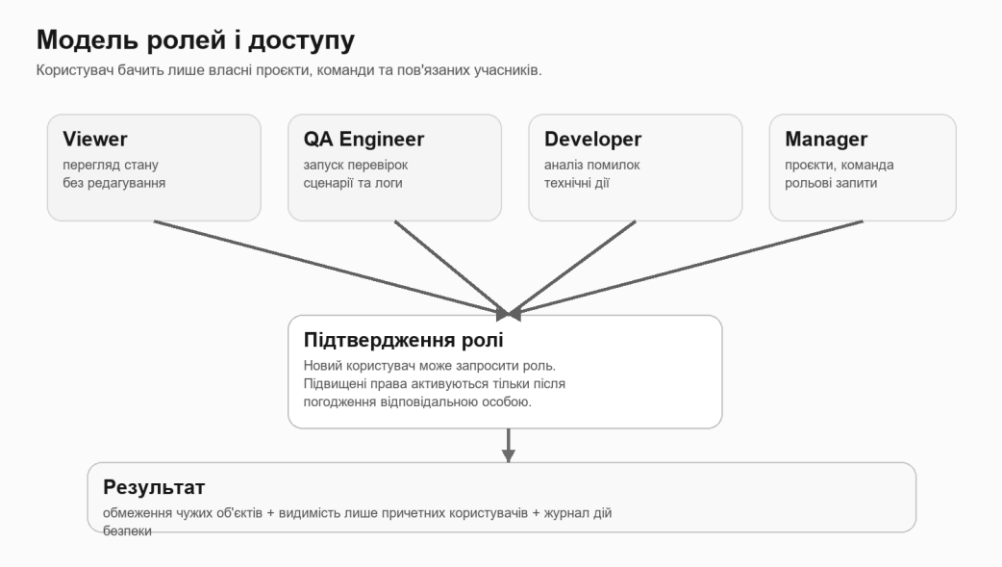

Рисунок 2.2 – Модель ролей, погодження та командної видимості

2.6 UML-моделювання системи

Для формалізації структури та поведінки системи використано UML-діаграми. Вони дозволяють показати не лише зовнішній вигляд вебзастосунку, а й внутрішні зв'язки між моделями, порядок взаємодії компонентів і можливі стани проєкту під час перевірки.

Актуальну UML-діаграму класів системи наведено на рис. 2.3.

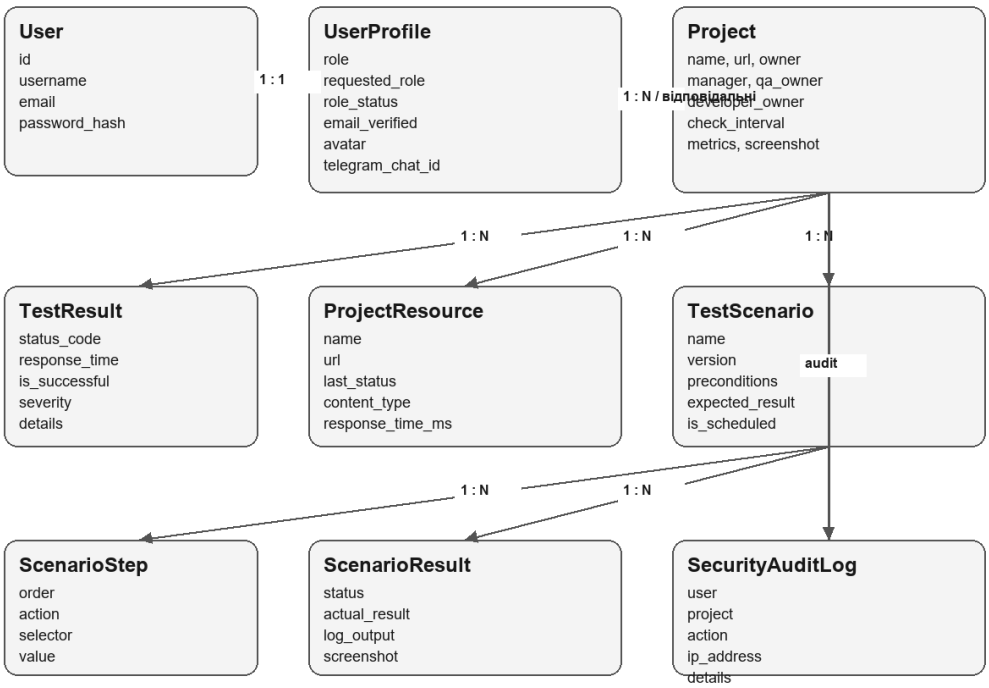

Рисунок 2.3 – UML-діаграма класів актуальної моделі даних

Діаграма класів відображає актуальні сутності системи після додавання ролей користувачів, командної видимості, додаткових ресурсів, сценаріїв тестування, чату проєкту та журналу безпеки. Центральною сутністю залишається Project, оскільки саме навколо вебпроєкту зберігаються основні дані: назва, URL-адреса, профіль перевірки, інтервал моніторингу, останні технічні показники, відповідальні особи та результати тестування.

Доступ до Project визначається не лише власником запису, а й призначеними учасниками команди: керівником проєкту, відповідальним QA-

інженером і відповідальним розробником. Це дозволяє реалізувати командну роботу без відкриття всіх даних усім користувачам системи.

Сутність UserProfile розширює стандартного користувача Django та зберігає роль, запитану роль, статус підтвердження ролі, аватар, стан підтвердження email і параметри сповіщень. Завдяки цьому система може відрізняти звичайного переглядача від QA-інженера, розробника або менеджера проєкту. Підвищені ролі не активуються автоматично, а потребують погодження відповідальною особою.

Для збереження результатів перевірок використовується модель TestResult, яка містить HTTP-статус, час відповіді, ознаку успішності, критичність і текстові деталі помилки. Додаткові файли та ресурси сайту описуються через ProjectResource, що дає можливість перевіряти не тільки головну сторінку, а й пов’язані CSS, JS, API або інші ресурси.

Окремий блок моделей відповідає за сценарії тестування: TestScenario, ScenarioStep і ScenarioResult. Вони дозволяють створювати послідовність дій користувача, запускати її для конкретного сайту та зберігати результат виконання.

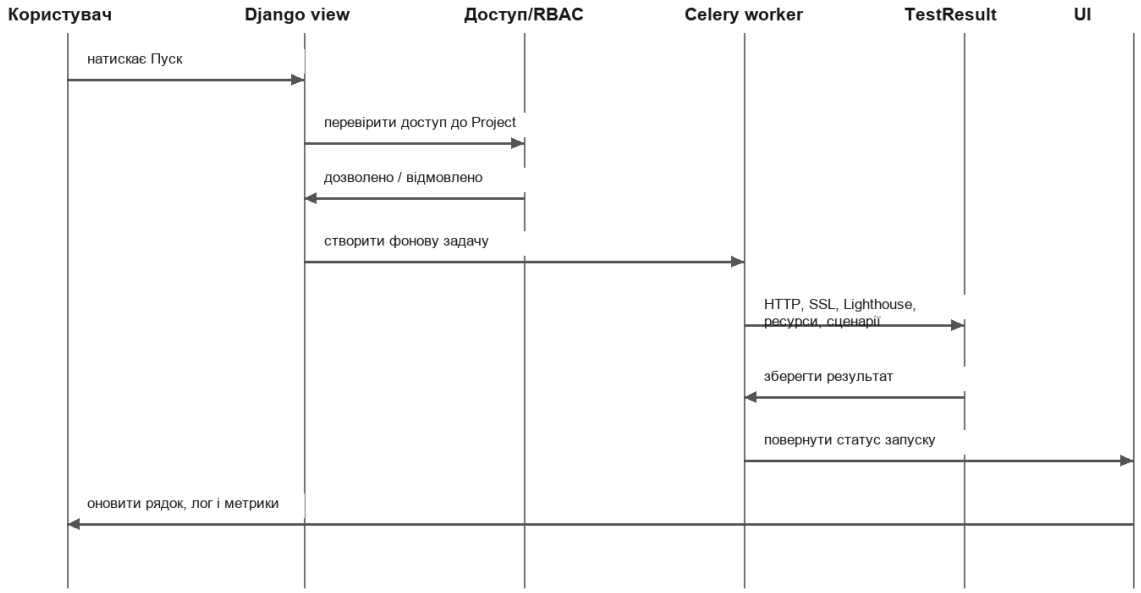

Рисунок 2.4 – UML-діаграма послідовності ручної перевірки

Діаграма послідовності демонструє порядок взаємодії між користувачем, вебінтерфейсом, серверною частиною, модулем перевірки доступу, фоновою задачею та базою даних під час ручного запуску тестування. Процес починається з того, що користувач натискає кнопку запуску перевірки у таблиці моніторингу. Для користувача ця дія виглядає як звичайна взаємодія з інтерфейсом, однак усередині системи вона запускає кілька послідовних етапів.

Спочатку браузер надсилає запит до Django-застосунку. Серверна частина визначає, який саме проєкт потрібно перевірити, і обов'язково перевіряє, чи має поточний користувач право працювати з цим об'єктом. Це важливо, тому що система підтримує командну модель доступу: користувач може запускати перевірку лише для власного проєкту або для проєкту, де він призначений відповідальною особою.

Після успішної перевірки доступу сервер не виконує довгу технічну операцію безпосередньо в HTTP-запиті, а передає її у фонову задачу. Такий підхід дозволяє не блокувати інтерфейс і не змушувати користувача чекати завершення всіх перевірок у відкритому запиті. У цей момент у таблиці моніторингу може відображатися стан виконання, підсвічування рядка або індикатор завантаження.

Фонова задача виконує основні технічні дії: перевіряє доступність сайту, HTTP-статус, час відповіді, SSL-сертифікат, JavaScript-помилки, метрики продуктивності, додаткові ресурси та за потреби пов'язані сценарії. Якщо окремий етап завершується помилкою, система не зупиняє весь процес без пояснення, а фіксує деталі збою в результаті перевірки.

Після завершення перевірки створюється або оновлюється запис TestResult, де зберігаються ключові показники: успішність, критичність, час відповіді, код статусу та текстові деталі. Далі інтерфейс отримує оновлений стан і показує користувачу результат без необхідності працювати з консоллю або запускати сторонні інструменти вручну. Така послідовність робить тестування зрозумілим для QA-інженера, менеджера або студента, який демонструє роботу системи.

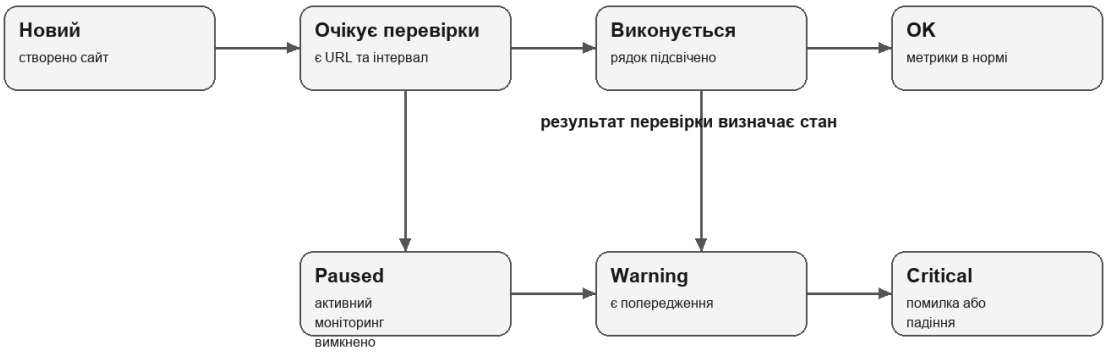

Рисунок 2.5 – UML-діаграма станів проєкту під час моніторингу

Діаграма станів демонструє життєвий цикл вебпроєкту в системі моніторингу від моменту його створення до отримання поточного технічного статусу. Після додавання сайту проєкт переходить у початковий стан, у якому вже збережено назву, URL-адресу, профіль перевірки, інтервал активного моніторингу та відповідальних користувачів. На цьому етапі система ще не має достатньої історії запусків, тому не може повноцінно оцінити стабільність ресурсу.

Наступним станом є очікування перевірки. У цьому режимі проєкт зберігається в системі, відображається у таблиці моніторингу та готовий до ручного або планового запуску тестування. Якщо активний моніторинг увімкнений, система може автоматично запускати перевірку відповідно до заданого інтервалу. Якщо моніторинг вимкнений, проєкт залишається доступним для ручного запуску, але не перевіряється за розкладом.

Після натискання кнопки запуску або спрацювання планового інтервалу проєкт переходить у стан виконання перевірки. У цей момент система виконує технічні операції: перевіряє HTTP-статус, час відповіді, SSL-сертифікат, показники продуктивності, JavaScript-помилки, додаткові ресурси та інші доступні дані. Для користувача цей стан має бути помітним в інтерфейсі, наприклад через підсвічування рядка, індикатор виконання або зміну іконки запуску.

Після завершення перевірки проєкт переходить в один із підсумкових станів. Якщо основні показники в нормі, сайт отримує стан OK. Якщо виявлено некритичні проблеми, наприклад зниження продуктивності, попередження за SSL або нестабільні метрики, проєкт переходить у стан Warning. Якщо сайт недоступний, повертає помилковий статус, має критичні проблеми безпеки або суттєві збої, система визначає стан як Critical.

Окремо передбачено стан паузи активного моніторингу. Він використовується тоді, коли користувач тимчасово вимикає автоматичні перевірки для конкретного сайту. У такому випадку історія попередніх запусків не видаляється, але нові планові перевірки не виконуються до повторного ввімкнення моніторингу. Така модель станів допомагає зрозуміти, що проєкт у системі не є статичним записом, а постійно змінює свій стан залежно від дій користувача та результатів технічного аналізу. Узагальнену діаграму розгортання системи наведено на рис. 2.7.

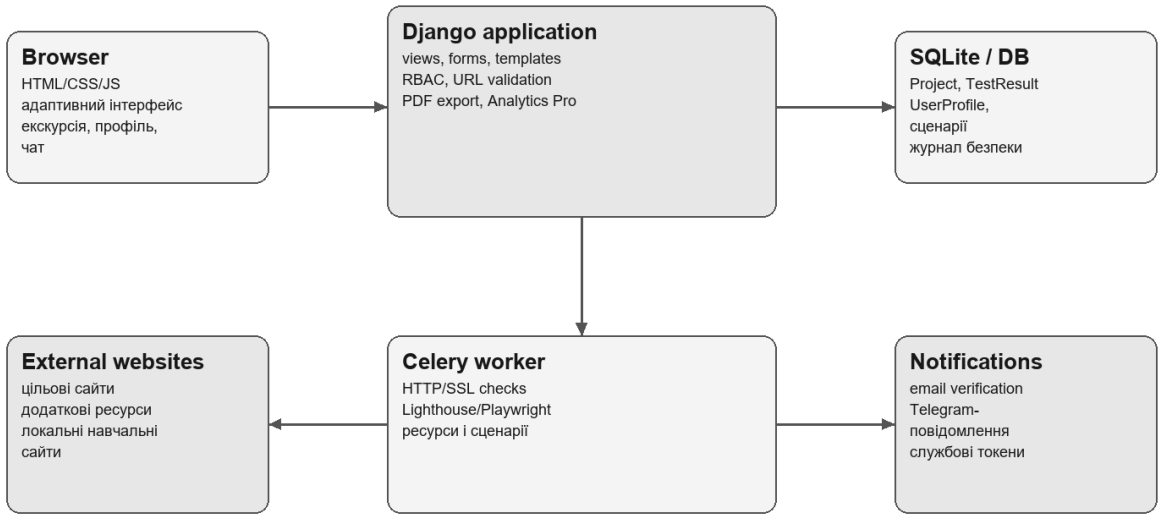

Рисунок 2.7 – UML-діаграма розгортання системи

Діаграма розгортання показує фізичну та логічну організацію компонентів системи BitGuard. Вона пояснює, які частини застосунку працюють на стороні користувача, які виконуються на сервері, де зберігаються дані та які

зовнішні ресурси залучаються під час моніторингу. Така діаграма важлива, тому що система не обмежується однією вебсторінкою, а складається з кількох взаємопов'язаних вузлів.

Першим вузлом є браузер користувача. Саме через нього користувач відкриває сторінку моніторингу, додає вебсайти, запускає перевірки, переглядає логи, аналітику, профіль, команду та звіти. Браузер відповідає за відображення HTML-сторінок, CSS-стилів і JavaScript-логіки інтерфейсу. Уся взаємодія користувача із системою відбувається без необхідності працювати з командним рядком.

Центральним серверним вузлом є Django-застосунок. Він приймає HTTP-запити від браузера, перевіряє авторизацію користувача, застосовує рольові обмеження, обробляє форми, формує сторінки інтерфейсу та надає API для окремих динамічних дій. Саме на рівні Django реалізовано логіку доступу до проєктів, профілі користувачів, командну видимість, створення звітів, роботу з налаштуваннями та збереження результатів.

Окремим вузлом виступає база даних. У ній зберігаються користувачі, профілі, ролі, вебпроєкти, результати перевірок, додаткові ресурси, сценарії тестування, повідомлення чату та записи журналу безпеки. Завдяки базі даних система може не тільки показати результат останньої перевірки, а й зберігати історію, будувати аналітику та формувати звіти на основі попередніх запусків.

Для виконання довгих або ресурсомістких операцій використовується фонове worker-середовище. До нього передаються задачі перевірки доступності сайту, SSL-сертифіката, метрик продуктивності, браузерних сценаріїв, додаткових ресурсів і технічних показників. Це потрібно для того, щоб основний вебінтерфейс не зависав під час тривалого аналізу. Користувач запускає перевірку кнопкою, а система виконує основну роботу у фоні та після завершення оновлює результат.

Окремо на діаграмі показані цільові вебресурси. Це сайти, API або локальні навчальні проєкти, які користувач додає до моніторингу. Система надсилає до них запити, аналізує відповіді, перевіряє доступність, швидкість, SSL,

структуру сторінки та додаткові файли. Водночас передбачена перевірка безпечності URL, щоб зменшити ризик небажаних запитів до приватних або службових адрес.

Канали сповіщення є допоміжним вузлом розгортання. Вони використовуються для підтвердження email, надсилання повідомлень про критичні події або інформування користувача про важливі зміни стану проєкту. Завдяки такій архітектурі система може поєднувати інтерактивний вебінтерфейс, фонове тестування, збереження історії та повідомлення користувачів у єдиному робочому середовищі.

2.7 Алгоритми перевірки та формування індексу якості

Алгоритм перевірки вебресурсу побудовано як послідовність незалежних етапів. Якщо один із етапів завершується помилкою, система не повинна припиняти роботу повністю. Вона фіксує проблему в деталях запуску і переходить до наступних доступних перевірок або формує звіт з позначенням відсутніх даних.

Такий підхід є важливим для реального моніторингу, адже зовнішній вебресурс може бути частково доступним: наприклад, HTTP-відповідь отримано, але аудит продуктивності не зміг завершити аудит; або головна сторінка працює, але додатковий CSS-файл повертає помилку. Усі ці ситуації мають відображатися в історії.

Індекс якості формується як узагальнений показник, який допомагає швидко оцінити стан сайту. Він не є офіційним стандартом якості, але використовується як зручний агрегований сигнал для користувача. Детальне пояснення завжди залишається важливішим за одне число.

Алгоритм перевірки виконується послідовно: система перевіряє право доступу до проєкту, валідність URL, виконує HTTP/SSL-аналіз, збирає метрики продуктивності та браузерні сценарії, перевіряє додаткові ресурси та сценарії, визначає критичність, зберігає TestResult і оновлює інтерфейс.

Таблиця 2.3 – Логіка інтерпретації основних показників

Показник	**Добрий стан**	**Попередження**	**Критичний стан**
HTTP-статус	200-299	300-399 або нестабільна відповідь	400-599 або відсутність відповіді
Uptime	Не менше 99 %	95-98,99 %	Менше 95 %
SSL	Сертифікат чинний	Сертифікат скоро завершується	Сертифікат відсутній або недійсний
Performance	90-100	50-89	0-49
JS-помилки	0 помилок	1-3 помилки	Понад 3 помилки
Додаткові ресурси	Усі ресурси доступні	Є повільні або перенаправлені ресурси	Є недоступні критичні ресурси

Таблиця 2.3 використовується для пояснення того, як система інтерпретує основні технічні показники після виконання перевірки вебресурсу. Вона не лише фіксує окремі значення, а й допомагає перетворити технічні результати на зрозумілий для користувача стан: добрий, попереджувальний або критичний. Такий підхід важливий, оскільки не кожен користувач системи має однаковий рівень технічної підготовки.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Обрані програмні засоби

Вибір програмних засобів здійснювався з урахуванням теми роботи, необхідності розробки гарного вебінтерфейсу, підтримки бази даних, фонових задач і можливості інтегрувати інструменти технічного аналізу вебсторінок.

Таблиця 3.1 – Використані програмні засоби

Засіб	Роль у системі
Django 6	Основний вебфреймворк для моделей, маршрутизації, форм і шаблонів
Celery	Виконання фонових та планових перевірок
Redis	Брокер повідомлень для Celery
Requests і BeautifulSoup	HTTP-перевірки, аналіз HTML та посилань
Playwright	Браузерні сценарії та машинне тестування
Lighthouse	Оцінка продуктивності, доступності та SEO
ReportLab	Формування PDF-звітів
Bootstrap і Chart.js	Інтерфейс користувача та графіки

Обраний набір засобів є достатнім для реалізації навчального, але практично придатного вебзастосунку. Django забезпечує основу серверної логіки, Celery дозволяє виконувати довші перевірки у фоні, а інструменти технічного аудиту розширюють систему від простого uptime-моніторингу до аналізу якості сторінки.

3.2 Особливості реалізації програмного забезпечення

Модуль моніторингу надає користувачу таблицю проєктів із поточним статусом, показником uptime, оцінками продуктивності, кнопками запуску перевірок, налаштуваннями та PDF-звітом. Під час запуску перевірки рядок проєкту змінює візуальний стан, що дозволяє зрозуміти, який сайт перевіряється.

Модуль додаткових ресурсів дозволяє прив'язати до сайту CSS-файли, XML-карти, API-адреси або інші URL. Під час перевірки система фіксує статус ресурсу, тип відповіді, розмір контенту та можливу помилку. Це розширює моніторинг за межі головної сторінки сайту.

Модуль глибокої аналітики формує структурований звіт на основі зібраних метрик, логів і фактичних результатів перевірок. Для користувача процес подано як технічний аналіз: коротке резюме, критичні проблеми, детальний розбір, план виправлень і підсумок.

Окрема сторінка Analytics Pro реалізує вибірковий режим аналізу. Користувач обирає сайт, позначає потрібні типи даних за допомогою чекбоксів, а система формує короткі картки з фактичних результатів. Якщо сайт ще не тестувався, інтерфейс не показує умовні значення, а пропонує спочатку виконати перевірку на сторінці моніторингу.

Додатково в реалізації передбачено журналювання важливих дій користувача: створення або редагування проєкту, запуск перевірки, додавання ресурсів, експорт звіту та заблоковані операції. Це дає змогу не лише показати поточний результат, а й відстежити історію взаємодії із системою. Також важливо, що частина довгих операцій виконується у фоновому режимі, тому користувач може продовжувати роботу з інтерфейсом, поки система завершує технічну перевірку [4].

Під час реалізації інтерфейсу не всі рішення виявилися вдалими з першої спроби. Наприклад, панелі налаштувань і помічника спочатку відкривалися поверх основного вмісту та могли перекривати таблицю моніторингу. Після

перевірки цього сценарію було змінено логіку розміщення панелей, щоб вони краще узгоджувалися з робочою областю сторінки.

Окрема проблема виникала з модальними вікнами для додавання ресурсів і перегляду зображень: після закриття деяких елементів сторінка могла залишатися затемненою. Це показало, що для демонстраційного продукту важлива не лише наявність функції, а й коректне повернення користувача до попереднього стану інтерфейсу.

Також у процесі розробки довелося уточнювати подання аналітичного звіту. Спочатку звіт виглядав як суцільний текст і був незручним для читання. Після цього його було поділено на змістові блоки: резюме, критичні проблеми, детальний аналіз, план виправлень і висновок. Це рішення зробило результат більш придатним для демонстрації.

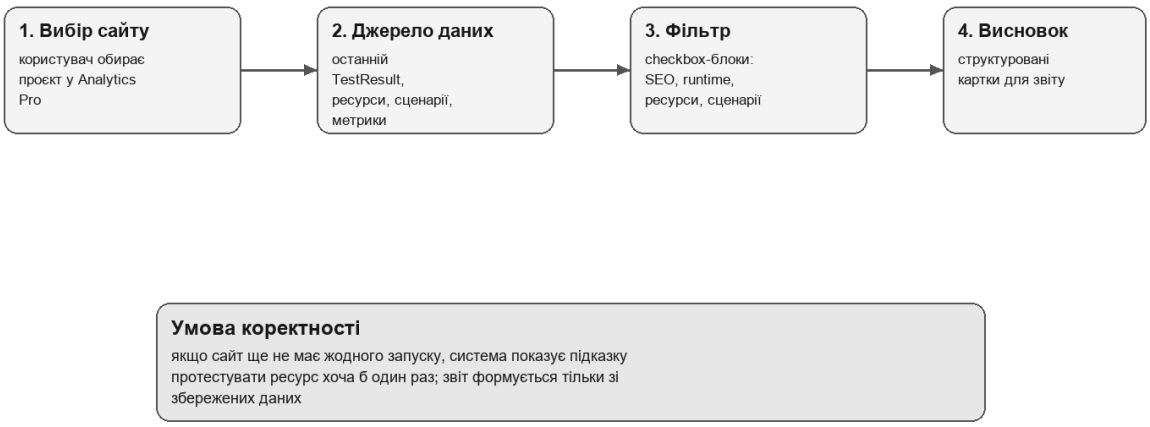

Рисунок 3.1 – Потік формування вибіркового аналітичного зрізу

Модуль логів та історії зберігає результати виконання перевірок. Для кожного запуску доступні дата, сайт, статус, тип запуску, критичність та деталі. Це важливо для відстеження повторюваних проблем і демонстрації того, що система не лише показує поточний стан, а й накопичує дані.

Модуль профілю користувача показує узагальнений індекс якості, кількість стабільних, проблемних і критичних сайтів, графік активності, роль,

статус підтвердження, аватар, контактні дані та персональні швидкі дії. Це перетворює профіль на короткий центр стану роботи користувача.

Сторінка налаштувань поділена на логічні розділи: акаунт, безпека, конфіденційність, сповіщення та налаштування браузера. Така структура робить профіль не формальною сторінкою, а місцем керування персональними параметрами і безпекою.

Модуль командної роботи обмежує видимість користувачів: учасник бачить не весь список акаунтів системи, а лише тих людей, які пов'язані з його проєктами або командою. Для обговорення конкретного сайту передбачено проєктний чат, що зберігає повідомлення у прив'язці до Project.

Моніторинг. Після ручного або планового запуску система оновлює стан проєкту, тому користувач бачить не старий запис, а актуальний технічний результат

Аналітика. Звіт формується з метрик, логів і результатів перевірок, тому він зрозумілий не лише розробнику, а й керівнику або замовнику

Додаткові ресурси. CSS, sitemap, API та інші URL перевіряються окремо, що допомагає знайти проблеми не тільки на головній сторінці

Сценарії. Кроки машинного тестування описуються через дії та селектори, тому систему можна розвивати в напрямі повноцінних браузерних тестів

Логи. Деталі кожного запуску зберігаються після перевірки, отже проблему можна розібрати пізніше, а не тільки в момент виконання

Багатомовність. Перемикач української, англійської та німецької мов робить інтерфейс зручнішим для демонстрації різним користувачам

3.3 Перевірка працездатності системи

Під час розробки виконувались ручні перевірки основних сценаріїв: реєстрація, підтвердження email, вхід, додавання сайту, вибір відповідальних осіб, запуск перевірки, відкриття налаштувань, додавання ресурсу, перегляд логів, формування PDF-звіту, запуск глибокої аналітики, побудова зрізу в

Analytics Pro, робота профілю, командної сторінки, чату, перемикання мов і проходження onboarding-екскурсії.

Стан реалізації перевірено за основними сценаріями: реєстрація з email, рольові запити, моніторинг, ресурси, логи, Analytics Pro, PDF, профіль, чат і сценарії доступні у вебінтерфейсі та пов'язані зі збереженими даними.

За результатами перевірки система демонструє працездатність основних функцій, однак подальше вдосконалення може включати повні unit-тести, інтеграційні тести для API та автоматизовану перевірку інтерфейсу через браузерні end-to-end перевірки.

Фрагмент 3.1 – Запуск ручної перевірки через фонову задачу:

```
project =
get_object_or_404(accessible_projects(request.user),
id=project_id)
if is_rate_limited(f'manual-
test:{request.user.id}:{project.id}', limit=3,
window_seconds=60):
    audit_action(request, 'blocked', project, 'manual_test
rate limit')
    return JsonResponse({'error': 'Забагато запусків.'},
status=429)
check_project_availability.delay(project_id, manual=True)
audit_action(request, 'manual_test', project, 'manual=True')
```

Фрагмент демонструє одразу кілька важливих рішень: перевірку доступу через командну видимість, обмеження надто частих запусків, постановку довгої перевірки у фонове виконання та запис дії до журналу безпеки.

Основною сторінкою для щоденної роботи є сторінка моніторингу. Вона містить список вебпроєктів, їх URL-адреси, поточний HTTP-статус, показник uptime, оцінки продуктивності, доступності та SEO, стан SSL-сертифіката, кількість JavaScript-помилок і короткі технічні позначки. Для кожного проєкту користувач може вручну запустити перевірку, перейти до графіків, відкрити налаштування, переглянути додаткові ресурси або сформувати PDF-звіт. Під час запуску тестування інтерфейс показує, що саме зараз виконується, тому користувач розуміє, який сайт перевіряється і на якому етапі перебуває процес.

Під час додавання або редагування сайту користувач задає не лише назву та URL-адресу, а й профіль перевірки, тип ресурсу, пристрій для емуляції, інтервал активного моніторингу, відповідальних осіб і додаткові правила контролю. Передбачено можливість пошуку ключового слова на сторінці. Це корисно, коли потрібно перевірити, чи не зник зі сторінки важливий текст, наприклад назва послуги, повідомлення про оплату, контактний блок або інший фрагмент контенту. Якщо очікуваний текст не знайдено, система може позначити це як проблему і зафіксувати її в результатах перевірки.

Окремою функцією є пошук битих посилань. Користувач може увімкнути сканування внутрішніх посилань і вибрати ліміт перевірки. Система аналізує знайдені переходи, визначає недоступні або помилкові URL і зберігає кількість проблемних посилань. Така можливість важлива не лише для технічного тестування, а й для UX та SEO, оскільки биті посилання знижують довіру до сайту, погіршують навігацію і можуть негативно впливати на індексацію сторінок. Для такого аналізу сторінка розглядається як HTML-документ, що може бути оброблений засобами парсингу на кшталт Beautiful Soup [14].

Під кожним проєктом передбачено блок додаткових файлів і ресурсів. До нього можна додавати CSS-файли, JavaScript-файли, API-адреси, зображення або інші пов’язані ресурси, які мають значення для коректної роботи сайту. Після додавання такі ресурси перевіряються разом із основним сайтом: система фіксує статус відповіді, час завантаження, тип контенту, розмір і можливу помилку. Це дозволяє бачити не тільки стан головної сторінки, а й стан допоміжних компонентів, від яких залежить повна працездатність вебресурсу.

Для більш складного контролю передбачено користувацькі сценарії тестування. Користувач може створити сценарій, описати його мету, передумови, очікуваний результат і додати послідовність кроків: перехід на сторінку, натискання, введення тексту, очікування, перевірка тексту або створення скріншота. Такий підхід дозволяє перевіряти не лише технічну доступність сайту, а й типові дії користувача. Наприклад, можна описати сценарій авторизації, пошуку товару, відправлення форми або перевірки появи повідомлення після

натискання кнопки. Результати запуску сценаріїв зберігаються окремо, разом зі статусом, фактичним результатом, логом виконання та скріншотом у разі помилки.

Сторінка детальної аналітики використовується тоді, коли потрібно отримати ширше пояснення стану сайту. На ній система виконує поглиблений технічний аналіз і формує структурований звіт. У звіті подається коротке резюме, критичні проблеми, детальний аналіз, план виправлень і висновок для користувача. Такий формат зручний для демонстрації результатів керівнику, розробнику або іншому учаснику команди, оскільки замість набору сирих метрик користувач отримує готове пояснення проблем і можливих дій.

У межах детальної аналітики також передбачено швидкий технічний блок перевірок. Він дозволяє оцінити інфраструктурні та сторінкові параметри: заголовки безпеки, SSL, доменну інформацію, IP-ознаки, SEO-структуру, наявність H1, description, alt-атрибутів, вагу сторінки, час першої відповіді сервера, швидкість завантаження, JavaScript-помилки та інші технічні сигнали. Цей блок потрібний для швидкої первинної діагностики, коли користувач хоче не чекати повного звіту, а одразу побачити слабкі місця сайту.

Окрема сторінка Analytics Pro дає змогу сформувати вибірковий аналітичний зріз. Користувач обирає конкретний сайт і відмічає, які саме блоки даних потрібно включити: доступність, продуктивність, SEO, безпеку, додаткові ресурси, сценарії, історію запусків або рекомендації. Якщо сайт ще жодного разу не перевірявся, система має повідомити, що для формування аналітики спочатку потрібно виконати хоча б один запуск. Це робить сторінку логічною: вона не вигадує дані, а працює зі збереженими результатами перевірок.

Важливою частиною системи є вкладка логів та історії. У ній користувач бачить, коли виконувалася перевірка, який сайт перевірявся, який був результат, який рівень критичності зафіксовано і які технічні деталі отримала система. Логи допомагають відстежити повторювані проблеми, відрізнити випадковий збій від стабільної помилки та підтвердити факт виконання тестування.

Це особливо корисно під час демонстрації роботи, оскільки користувач може показати не лише поточний стан, а й історію змін.

Для візуального аналізу в системі передбачено графіки та конструктор графіків. Користувач може переглядати динаміку доступності, часу відповіді, продуктивності, SEO, кількості помилок або інших технічних показників. Конструктор графіків дозволяє обрати тип візуалізації, потрібні метрики та спосіб представлення даних; для побудови графіків використано підхід, сумісний із бібліотекою Chart.js [16]. Це дає можливість адаптувати аналітику під різні задачі: для QA-інженера важливими можуть бути падіння тестів і JS-помилки, для менеджера – загальна стабільність, а для розробника – технічні метрики продуктивності.

Система також підтримує формування PDF-звітів. Під час створення звіту користувач може вибрати, які блоки інформації включити: загальні дані про проєкт, історію перевірок, метрики продуктивності, безпеку, биті посилання, додаткові ресурси, сценарії або аналітичний висновок. Такий підхід робить звіт не перевантаженим, а придатним для конкретної ситуації. Наприклад, для короткої демонстрації можна залишити тільки підсумок і критичні проблеми, а для технічного аналізу – додати метрики, ресурси, логи та сценарії.

Для командної роботи реалізовано профілі користувачів, ролі та призначення відповідальних осіб. Користувач може мати роль Viewer, QA Engineer, Developer або Project Manager. Підвищені ролі не повинні надаватися довільно: вони потребують підтвердження відповідальною особою. У межах проєкту можна призначити менеджера, відповідального тестувальника і відповідального розробника. Це дозволяє обмежити доступ до чужих об’єктів і водночас організувати роботу кількох людей над одним сайтом.

У профілі користувача відображаються особисті дані, роль, стан підтвердження email, аватар, персональні показники, швидкі дії та узагальнений індекс стану проєктів. Такий профіль виконує не лише інформаційну, а й навігаційну функцію: користувач швидко бачить, у якому стані його проєкти, що

потребує уваги і які дії варто виконати далі. Налаштування користувача винесені в окремий блок, де можна керувати акаунтом, безпекою, сповіщеннями, мовою, темою інтерфейсу та іншими параметрами.

Для взаємодії учасників передбачено командну сторінку і чат проєкту. Користувач не бачить усіх зареєстрованих людей системи, а тільки тих, хто пов’язаний із його проєктами або командою. Це зменшує інформаційний шум і підвищує приватність. Чат проєкту дозволяє залишати короткі повідомлення щодо знайдених проблем, результатів перевірки або подальших дій. Така функція робить систему ближчою до реального робочого середовища, де тестування пов’язане з комунікацією між QA-інженером, розробником і менеджером.

Окремо передбачені сповіщення, зокрема через Telegram. Їхня задача полягає в тому, щоб користувач міг отримати повідомлення про критичний стан сайту, падіння перевірки або іншу важливу подію без постійного перегляду сторінки моніторингу. Це особливо корисно для активного моніторингу, коли сайт перевіряється за розкладом. У перспективі такі сповіщення можуть бути розширені налаштуваннями важливості, тихих годин і групування подій.

Додатковою службовою можливістю є webhook для автоматичного запуску перевірок. Він дозволяє інтегрувати систему з іншими процесами, наприклад із CI/CD або локальними навчальними сценаріями. Користувач може отримати спеціальну URL-адресу, через яку перевірка запускається автоматично. Це корисно, коли потрібно перевіряти сайт не лише вручну, а й після певної події, наприклад після оновлення коду або перед демонстрацією.

Для початкового знайомства із системою передбачено онбординг та екскурсію сайтом. Їхня мета – пояснити користувачу, де знаходиться моніторинг, як додати перший сайт, як запустити перевірку, де переглянути логи, як сформувати звіт і як працювати з профілем. Це зменшує поріг входу, особливо для користувачів, які не мають досвіду з інструментами тестування або звикли працювати тільки з готовими вебінтерфейсами.

Отже, особливість застосування системи полягає в тому, що вона поєднує одразу кілька напрямів контролю якості вебресурсів: доступність, продуктивність, безпеку, SEO, JavaScript-помилки, додаткові ресурси, биті посилання, ключовий контент, сценарії користувача, історію запусків, графіки, командну роботу та звітність. Завдяки цьому BitGuard QA може використовуватися не тільки як проста система перевірки доступності, а як комплексний робочий інструмент для ручного і автоматизованого тестування вебсайтів.

3.5 Інструкція користувача щодо роботи із системою

Для демонстрації практичного застосування системи доцільно описати типовий порядок роботи користувача. Інструкція не замінює технічну документацію розробника, але показує, як саме користувач взаємодіє з готовим вебзастосунком і які результати має отримати після виконання дій.

Перед початком роботи користувач запускає застосунок через підготовлений файл запуску або через сервер розробки, відкриває сторінку входу і створює обліковий запис. Після авторизації система показує стартову сторінку або сторінку моніторингу, де розміщено основні дії.

1. Увійти до системи. Користувач авторизується або створює новий обліковий запис і переходить до робочого простору

2. Додати сайт. У формі потрібно ввести назву, URL та профіль перевірки; після збереження сайт з'явиться у таблиці моніторингу

3. Додати ресурси. За потреби до сайту додаються CSS, sitemap, API або інші URL, які потрібно перевіряти разом з основною сторінкою

4. Запустити перевірку. користувач натискає кнопку запуску для конкретного сайту, а рядок переходить у стан виконання

5. Переглянути результат. Після завершення перевірки оновлюються статус, uptime, SSL, оцінки продуктивності та короткі рекомендації

6. Відкрити логи. Журнал показує попередні запуски, деталі виконання та помилки, які варто перевірити першими

7. Запустити аналітику. Користувач відкриває глибоку аналітику або Analytics Pro, обирає сайт і потрібні блоки даних; якщо сайт ще не тестували, система підказує спочатку виконати перевірку

8. Сформувати PDF. Користувач обирає потрібні блоки звіту й отримує матеріал для керівника, замовника або захисту

9. Переглянути профіль. Профіль показує узагальнений індекс якості, роль, статус підтвердження, персональні дані, аватар, швидкі дії та пов'язані проєкти

Під час виконання перевірки користувач не повинен оновлювати сторінку без потреби, оскільки частина результатів надходить після завершення фонової задачі. Якщо перевірка завершується помилкою, її деталі потрібно дивитися у журналі логів, а потім повторити запуск або змінити налаштування сайту. Які помилки можуть бути:

Сайт не перевіряється. Найчастіше це означає некоректний або недоступний URL; користувачу варто перевірити адресу, схему https/http і доступність сайту у браузері

метрики продуктивності відсутні. Інструмент міг завершитися з помилкою або сайт міг заблокувати аудит; у такому випадку потрібно переглянути логи і повторити перевірку пізніше

Звіт містить мало даних. Зазвичай це трапляється, якщо для сайту ще не було повної перевірки; спочатку варто запустити ручний аудит, а потім сформувати звіт повторно

Не видно чужий проєкт. Це не помилка, а очікувана поведінка безпеки: система показує лише об'єкти поточного користувача

Аналітичний висновок недоступний. Якщо зовнішній сервіс або ключ не налаштований, система може використати локальний режим звіту без зупинки основної перевірки

3.6 Наочний огляд інтерфейсу та дизайнерських рішень

Для демонстрації роботи системи доцільно показувати інтерфейс не як набір окремих сторінок, а як послідовний маршрут користувача. Спочатку користувач бачить сторінку входу (рис. 3.2) або реєстрації з коротким поясненням призначення BitGuard. Дизайн цієї частини має створювати перше враження про продукт: система призначена для контролю якості вебресурсів, тому інтерфейс не перевантажений зайвими декоративними елементами, а використовує зрозумілі акценти, службові підказки та спокійну палітру.

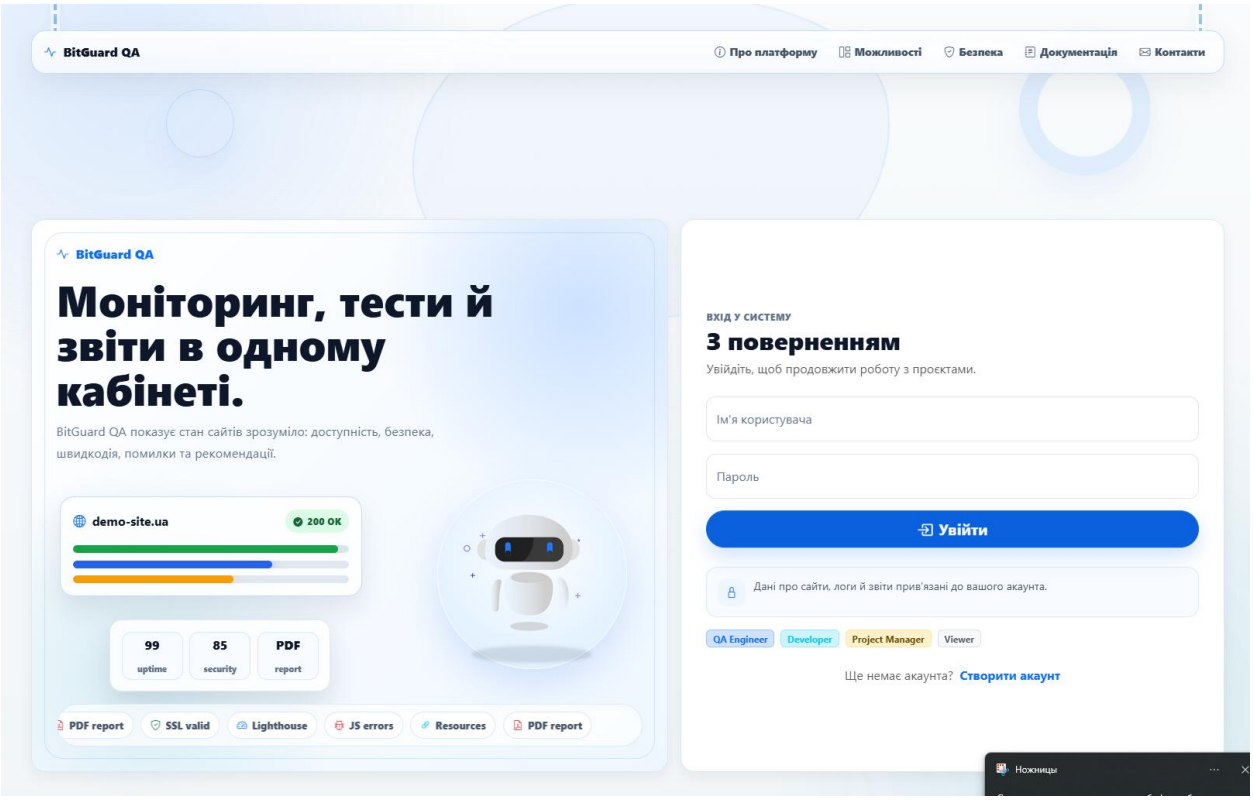

Рисунок 3.2 – Сторінка входу

Після входу головним робочим екраном стає моніторинг. Тут користувач одразу бачить список сайтів, статуси, uptime, SSL, оцінки продуктивності, короткі рекомендації та кнопки дій. Саме ця сторінка має найбільше практичне значення, тому в дизайні використано табличну структуру, компактні статусні позначки, іконки дій і візуальне підсвічування процесу запуску. Кнопки

розміщені поруч із робочою областю, щоб користувач не шукав основні дії по всій сторінці.

Сторінка моніторингу є головним робочим простором системи, де користувач одразу бачить стан усіх доступних вебпроєктів. Вона поєднує таблицю сайтів, статуси доступності, uptime, SSL, технічні оцінки, короткі рекомендації, кнопки запуску перевірки, перехід до графіків, налаштувань і PDF-звіту. Її перевага полягає в тому, що користувачу не потрібно відкривати кілька різних сервісів: основні показники якості сайту зібрані в одному місці та подані у зрозумілому вигляді.

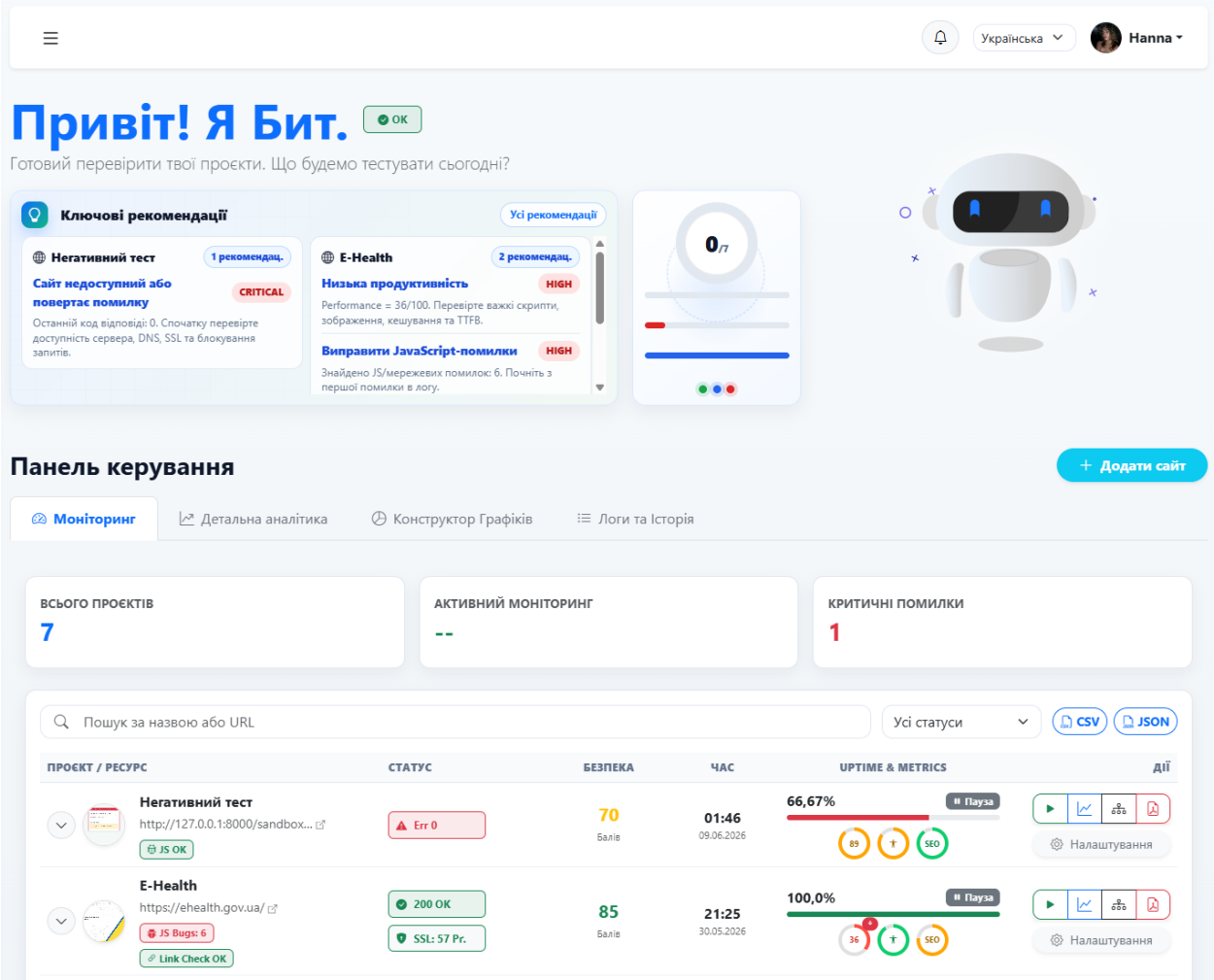

Рисунок 3.3 – Сторінка моніторингу

Сторінка додавання сайту побудована не як проста форма з двома полями, а як повноцінний центр налаштування майбутнього моніторингу.

Користувач може вказати URL, тип ресурсу, профіль перевірки, інтервал активного моніторингу, пристрій для емуляції, ключове слово для контролю контенту, сканування битих посилань і відповідальних учасників команди. Саме це робить сторінку сильною: ще на етапі створення проєкту користувач налаштовує не просто адресу сайту, а повний сценарій його подальшого контролю.

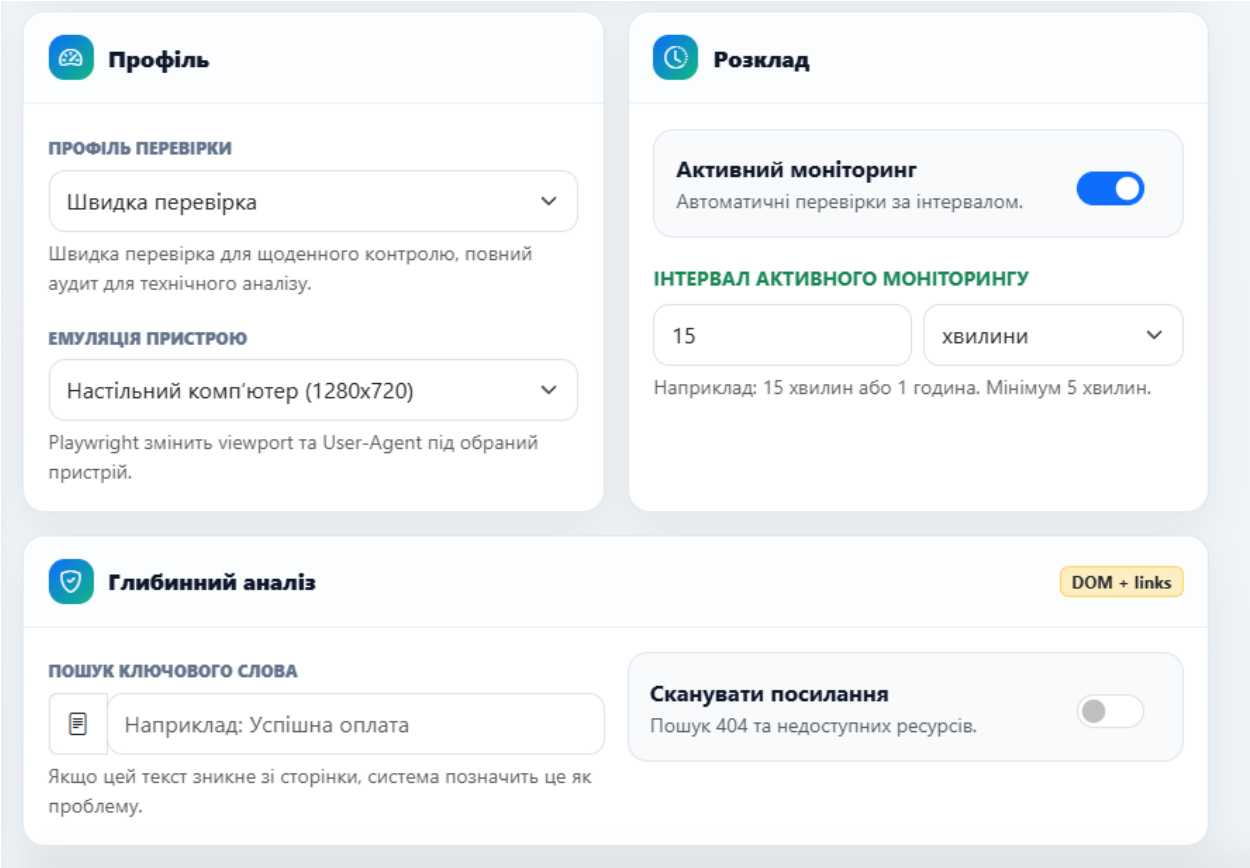

Рисунок 3.4 – Форма додавання сайту

Для аналізу результатів використовуються дві логіки подання: детальна аналітика і Analytics Pro. Перша підходить для готового структурованого висновку, друга – для вибіркового зрізу, коли користувач сам обирає потрібні блоки даних. Дизайнерське рішення полягає в тому, щоб не змушувати користувача читати суцільний технічний текст: висновки, критичні проблеми, графіки, ресурси і сценарії подаються окремими змістовими блоками.

Сторінка Analytics Pro призначена для вибіркового аналізу результатів перевірок. Користувач обирає конкретний сайт і сам визначає, які блоки даних

йому потрібні: доступність, продуктивність, SEO, безпека, додаткові ресурси, сценарії або історія запусків. Такий підхід робить аналітику гнучкою та не перевантажує користувача зайвою інформацією. Сторінка особливо корисна тоді, коли потрібно швидко підготувати зріз стану сайту для керівника, розробника або захисту роботи.

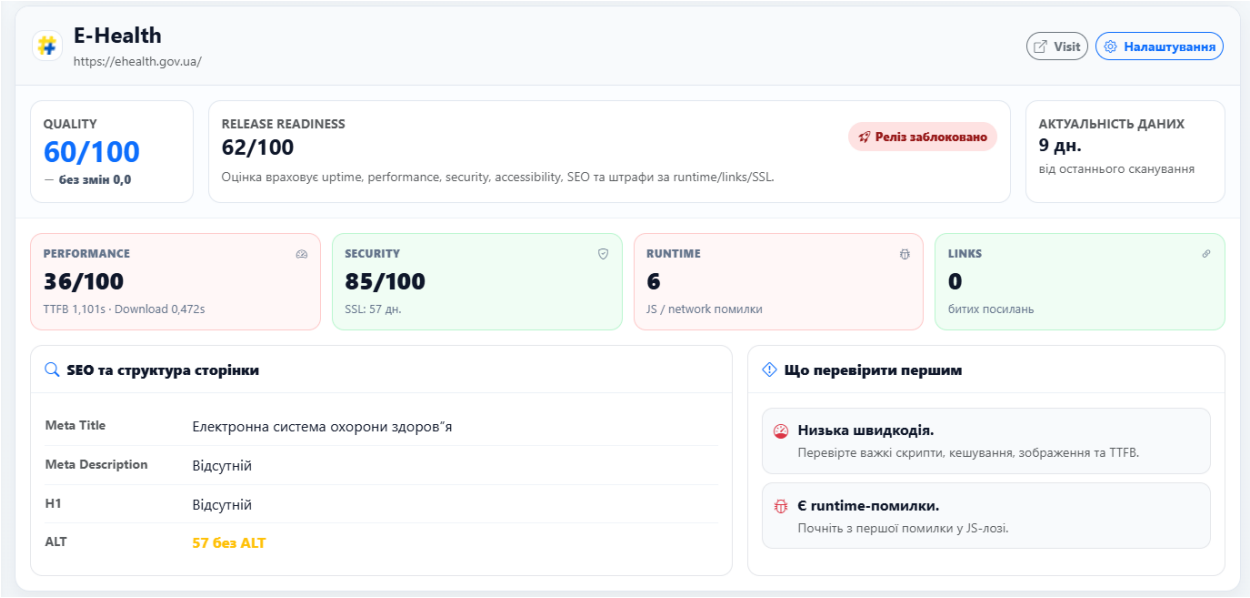

Рисунок 3.5 – Форма детального аналізу сторінки Analytics Pro

Окрему роль відіграють профіль користувача, команда і чат проєкту. Вони показують, що система розрахована не тільки на одиночний запуск тесту, а й на реальну командну роботу. У профілі відображаються роль, аватар, персональний стан проєктів і швидкі дії, а на сторінці команди користувач бачить лише пов’язаних із ним учасників. Це підтримує і зручність, і приватність. Профіль показує не лише інформацію про акаунт, а й загальний стан роботи користувача в системі. Завдяки цьому після входу можна швидко зрозуміти, які проєкти потребують уваги і що варто зробити далі.

Сторінка профілю також підкреслює персоналізацію системи. Користувач може завантажити власний аватар, бачити свою роль у команді та відстежувати, наскільки повно налаштований його акаунт. Це робить систему більш “живою”, тому що вона сприймається не просто як технічна таблиця з

перевірками, а як особистий робочий простір тестувальника, розробника або менеджера.

Важливою перевагою профілю є те, що він пов’язує особисті дані з реальним станом проєктів. Користувач бачить не абстрактну інформацію про себе, а практичні показники: скільки сайтів стабільні, скільки потребують уваги, які дії можна виконати швидко.

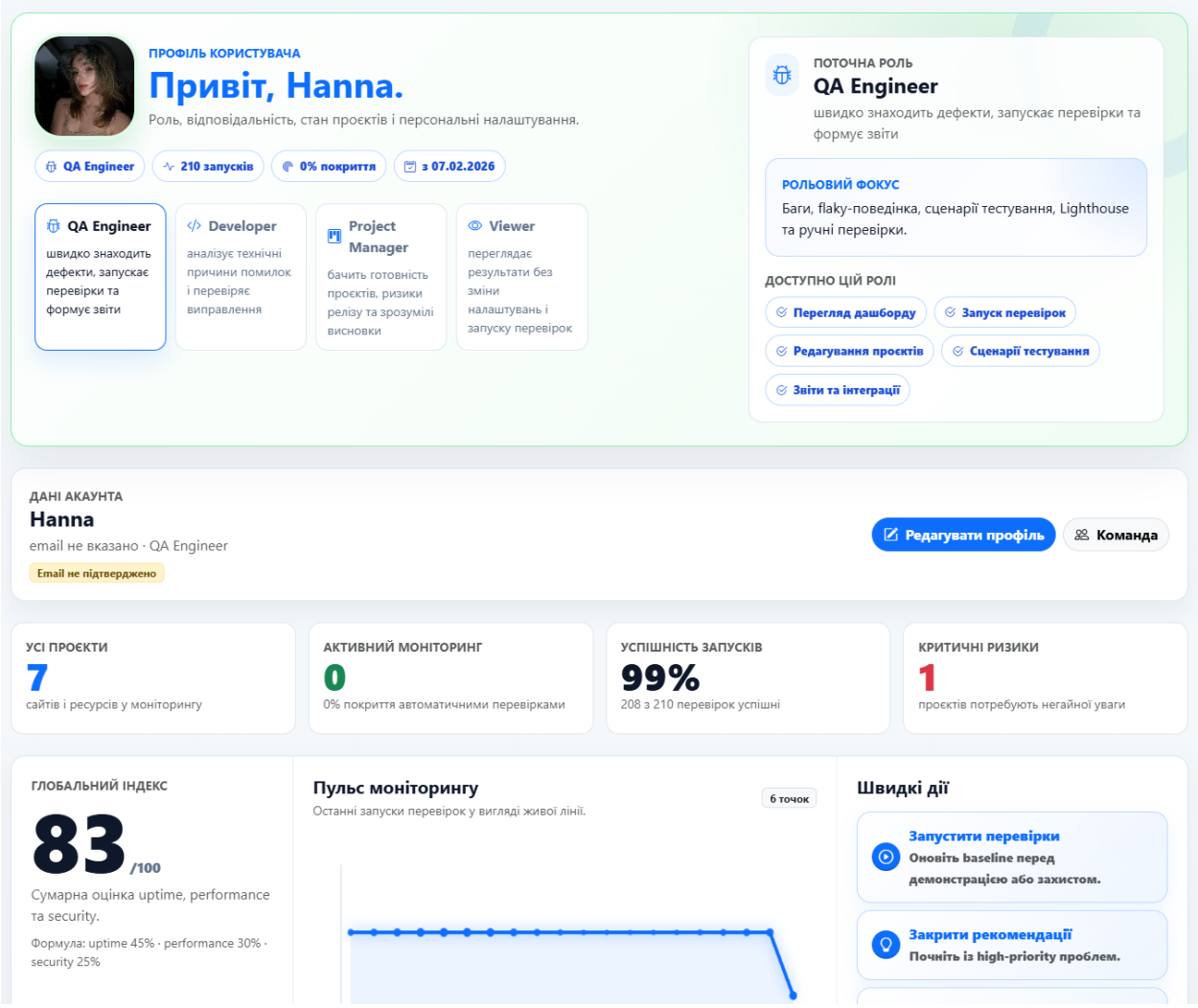

Рисунок 3.6 – Профіль користувача

Додатково сторінка профілю виконує роль навігаційного центру, з якого користувач може швидко перейти до найважливіших дій без повторного пошуку потрібного розділу в меню. Такий підхід скорочує час між виявленням

проблеми та наступним кроком: запуском перевірки, переглядом аналітики або переходом до налаштувань.

3.7 План тестування та критерії приймання

Оскільки тема роботи безпосередньо пов'язана з тестуванням вебсайтів, перевірка власної системи має особливе значення. Для BitGuard важливо підтвердити, що користувач може пройти повний цикл роботи: створити обліковий запис, додати вебресурс, запустити перевірку, переглянути логи, отримати рекомендації та сформувати звіт.

План тестування сформовано на основі основних користувацьких сценаріїв. Частина перевірок виконується вручну через інтерфейс, а частина може бути автоматизована у майбутньому через браузерні end-to-end перевірки. На етапі кваліфікаційної роботи головною метою є підтвердження працездатності основних функцій і відсутність критичних помилок у доступі до даних.

Для перевірки використовувалися не лише абстрактні сценарії, а й конкретні демонстраційні дані. Наприклад, для перевірки зовнішнього ресурсу використовувався публічний URL з HTTPS-схемою; очікуваним результатом було створення проєкту, поява рядка в таблиці моніторингу та можливість ручного запуску перевірки. Фактичний результат вважався успішним, якщо після запуску оновлювалися HTTP-статус, час відповіді, SSL-дані та запис у журналі.

Окремо перевірявся негативний сценарій із некоректною або локальною адресою. Очікуваним результатом було блокування такого URL і відсутність створення небезпечного ресурсу. Цей тест важливий, тому що система виконує мережеві запити за адресами користувача, а отже повинна відсікати приватні та службові адреси.

Для перевірки звітності використовувався сценарій, у якому після хоча б одного запуску перевірки користувач відкривав формування PDF і вибирав потрібні блоки. Очікуваним результатом був звіт без службових ключів і з

даними саме обраного проєкту. Додатково перевірялося, що користувач не може відкрити чужий проєкт через пряме посилання.

Також перевірялися сценарії роботи з додатковими ресурсами, профілем користувача, ролями та сторінкою команди, оскільки ці функції впливають не лише на зручність, а й на правильне розмежування доступу.

Таблиця 3.4 – План функціонального тестування системи

Сценарій	**Кроки перевірки**	**Критерій приймання**
Реєстрація користувача	Заповнити форму, перевірити повідомлення про email	Профіль створено, email очікує підтвердження або підтверджений після переходу за посиланням
Запит ролі	Обрати роль під час реєстрації або в налаштуваннях	Viewer активується одразу, підвищені ролі очікують погодження
Додавання сайту	Ввести назву, URL, профіль перевірки, інтервал і відповідальних	Проєкт з'являється у таблиці моніторингу
Ручний запуск перевірки	Натиснути кнопку запуску для конкретного сайту	Рядок переходить у стан перевірки, після завершення оновлюються метрики
Додаткові ресурси	Додати ресурс до проєкту і запустити перевірку	У логах видно статус, тип контенту, розмір і помилку ресурсу
Analytics Pro	Обрати сайт і блоки даних	Якщо результати є, показано картки; якщо ні, виведено повідомлення про необхідність першої перевірки

Продовження Таблиці 3.4

Сценарій	Кроки перевірки	Критерій приймання
Сценарій тестування	Створити кроки і запустити сценарій	Збережено ScenarioResult зі статусом, логами і фактичним результатом
Командний чат	Відкрити чат проєкту і надіслати повідомлення	Повідомлення видно тільки користувачам, причетним до проєкту
Експорт PDF	Обрати блоки звіту	Формується PDF без секретів і зайвих приватних даних

Критерієм приймання вважалося коректне виконання основної дії, збереження результату в системі та зрозуміле повідомлення користувачу у випадку помилки

3.8 Оцінювання результатів і напрями вдосконалення

Одним із головних результатів роботи є створення єдиного робочого простору, у якому користувач може додавати вебпроєкти, налаштовувати профілі перевірки, запускати тестування вручну або за заданим інтервалом, переглядати поточний статус сайтів, аналізувати помилки, працювати з історією запусків і формувати PDF-звіти. Такий підхід дозволяє централізувати процес контролю якості та зменшити кількість окремих інструментів, які зазвичай використовуються для перевірки доступності, продуктивності, SEO, безпеки й функціональності вебсайтів.

Система забезпечує перевірку ключових технічних показників вебресурсу: HTTP-статусу, uptime, часу відповіді, оцінок продуктивності, доступності та SEO, стану SSL-сертифіката, наявності JavaScript-помилок, SEO-структури сторінки, заголовків безпеки, ваги сторінки та інших параметрів. Крім основної сторінки сайту, користувач може додавати пов’язані ресурси, зокрема CSS-файли, JavaScript-файли, API-адреси, зображення та інші компоненти, від яких

залежить коректна робота вебзастосунку. Це дає змогу оцінювати не лише загальну доступність сайту, а й стан його допоміжної інфраструктури.

Окремою перевагою реалізованої системи є підтримка користувацьких сценаріїв функціонального тестування. Користувач може створювати сценарії, що описують типові дії відвідувача сайту: перехід на сторінку, натискання кнопки, введення тексту, очікування появи елемента, перевірку тексту або створення скріншота. Це дозволяє перевіряти не тільки технічну доступність ресурсу, а й коректність виконання бізнес-логіки, наприклад авторизації, пошуку товару, надсилання форми або додавання товару до кошика. Результати таких сценаріїв зберігаються окремо, що дає можливість аналізувати стабільність функціональних процесів у часі.

Важливим результатом є реалізація сторінок аналітики, графіків і конструктора графіків. Завдяки цьому користувач може не лише переглядати окремі значення метрик, а й аналізувати їхню динаміку, порівнювати результати різних запусків, визначати проблемні періоди та оцінювати вплив технічних помилок на загальний стан проєкту. Конструктор графіків робить аналітику гнучкішою, оскільки дозволяє обирати потрібні показники, тип візуалізації та спосіб представлення даних відповідно до задач конкретного користувача.

Також позитивним результатом є наявність детальної аналітики та структурованих звітів. Замість відображення лише сирих технічних даних система формує зрозуміле пояснення стану сайту, виділяє критичні проблеми, подає детальний аналіз, план можливих виправлень і підсумковий висновок. Це підвищує практичну цінність застосунку, оскільки отримані результати можна використовувати не тільки для технічної діагностики, а й для комунікації між QA-інженером, розробником, менеджером або замовником.

Система має значення і з погляду організації командної роботи. Реалізація профілів користувачів, ролей, призначення відповідальних осіб, командної сторінки, чату проєкту та сповіщень дозволяє використовувати BitGuard QA як робочий інструмент для кількох учасників. Розмежування ролей Viewer, QA Engineer, Developer і Project Manager сприяє контрольованому доступу до

функцій системи та зменшує ризик несанкціонованих змін. Наявність Telegram-сповіщень і webhook для автоматичного запуску перевірок додатково розширює можливості інтеграції системи в реальні робочі процеси.

Оцінюючи результати розроблення, можна зробити висновок, що поставлену мету було досягнуто. Розроблений вебзастосунок дозволяє виконувати ручні та автоматизовані перевірки вебсайтів, аналізувати їхній технічний стан, зберігати історію запусків, формувати звіти, працювати з графіками та організовувати взаємодію між учасниками проєкту. Система не обмежується лише моніторингом доступності, а охоплює ширший набір задач, пов’язаних із забезпеченням якості вебресурсів.

Разом із тим, під час подальшого розвитку системи доцільно врахувати низку можливих напрямів удосконалення. Першим напрямом є розширення можливостей функціонального тестування. У майбутньому можна реалізувати візуальний recorder mode, який дозволятиме користувачу записувати дії без ручного створення кожного кроку сценарію. Такий механізм значно спростить створення тестів, оскільки система зможе автоматично фіксувати кліки, введення тексту, переходи між сторінками та формувати початкову структуру сценарію.

Другим перспективним напрямом є впровадження механізму self-healing для тестових сценаріїв. Якщо елемент сторінки не знайдено за збереженим селектором, система могла б автоматично спробувати знайти його за альтернативними ознаками: текстом, роллю, атрибутом data-testid, aria-label або структурою DOM. Це дозволило б зменшити кількість помилкових падінь тестів, пов’язаних не зі справжньою несправністю сайту, а з незначними змінами у верстці.

Третім напрямом удосконалення є розвиток інтелектуальної аналітики. Система може автоматично визначати повторювані помилки, нестабільні сценарії, різке погіршення продуктивності, аномальне збільшення часу відповіді або зниження показників продуктивності. На основі таких даних можна формувати не лише звіти, а й рекомендації для користувача, наприклад щодо

збільшення timeout, перевірки певного ресурсу, оптимізації зображень або виправлення проблем зі скриптами.

Четвертим напрямом є покращення конструктора графіків. Доцільно додати більше типів візуалізацій, можливість порівняння періодів, побудову графіків за кількома сайтами, threshold-лінії, анотації подій, drill-down до конкретних логів і попередження про некоректні або неповні дані. Важливо також враховувати потенційні проблеми інтерпретації графіків, наприклад різні інтервали моніторингу, відсутність даних за обраний період, різні часові пояси або змішування метрик із різними одиницями вимірювання.

П’ятим напрямом розвитку є розширення інтеграцій. Окрім Telegram-сповіщень і webhook-запуску, система може бути інтегрована з GitHub Actions, GitLab CI/CD, Jira, Slack або іншими інструментами розробки. Це дозволить запускати перевірки після оновлення коду, автоматично створювати задачі за результатами критичних помилок і передавати звіти відповідальним учасникам команди.

Шостим напрямом удосконалення є покращення системи звітності. У майбутньому можна додати шаблони PDF-звітів для різних ролей користувачів: короткий управлінський звіт для менеджера, технічний звіт для розробника, QA-звіт зі сценаріями та логами для тестувальника. Також доцільно реалізувати експорт даних у CSV або JSON для подальшого аналізу в інших системах.

Сьомим напрямом є підвищення надійності та масштабованості системи. У разі збільшення кількості користувачів, сайтів і регулярних перевірок необхідно оптимізувати чергу запусків, обмежити надмірно часті перевірки, забезпечити паралельне виконання тестів, контролювати використання ресурсів і зберігати результати у структурованому вигляді. Це дозволить системі стабільно працювати навіть за великої кількості проєктів.

4 ІЛЮСТРАТИВНІ МАТЕРІАЛИ ТА ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ

4.1 Схеми архітектури та даних

Для пояснення логіки роботи BitGuard у цьому розділі наведено узагальнені схеми. Вони не дублюють програмний код, а показують взаємодію основних модулів, порядок виконання перевірок та спосіб подання аналітичних результатів користувачу.

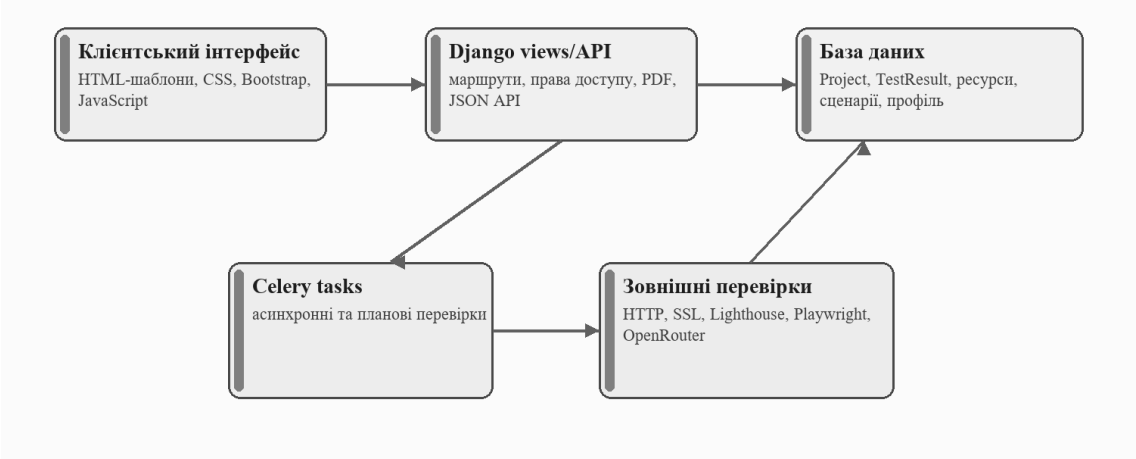

Рисунок 4.1 – Логічна архітектура вебсистеми BitGuard

Логічна архітектура BitGuard побудована так, щоб користувач міг працювати із системою через звичайний вебінтерфейс і не звертатися до консолі або окремих технічних інструментів. Центральною частиною є Django-застосунок, який приймає запити від браузера, перевіряє користувача, обробляє форми, працює з базою даних і передає складні перевірки у фонове виконання.

Коли користувач додає сайт або натискає кнопку запуску перевірки, система спочатку перевіряє, чи має він доступ до цього проєкту. Після цього довгі операції, наприклад перевірка доступності, SSL, метрик продуктивності, додаткових ресурсів або сценаріїв, виконуються окремо від основного інтерфейсу. Завдяки цьому сторінка не зависає, а користувач бачить зрозумілий процес виконання.

Результати перевірок зберігаються в базі даних і потім використовуються в різних частинах системи: у таблиці моніторингу, логах, Analytics Pro, графіках і PDF-звітах. Тобто один запуск тестування наповнює даними одразу кілька модулів. Така архітектура робить систему зручною для щоденного використання, а також залишає можливість надалі додавати нові види перевірок і звітів.

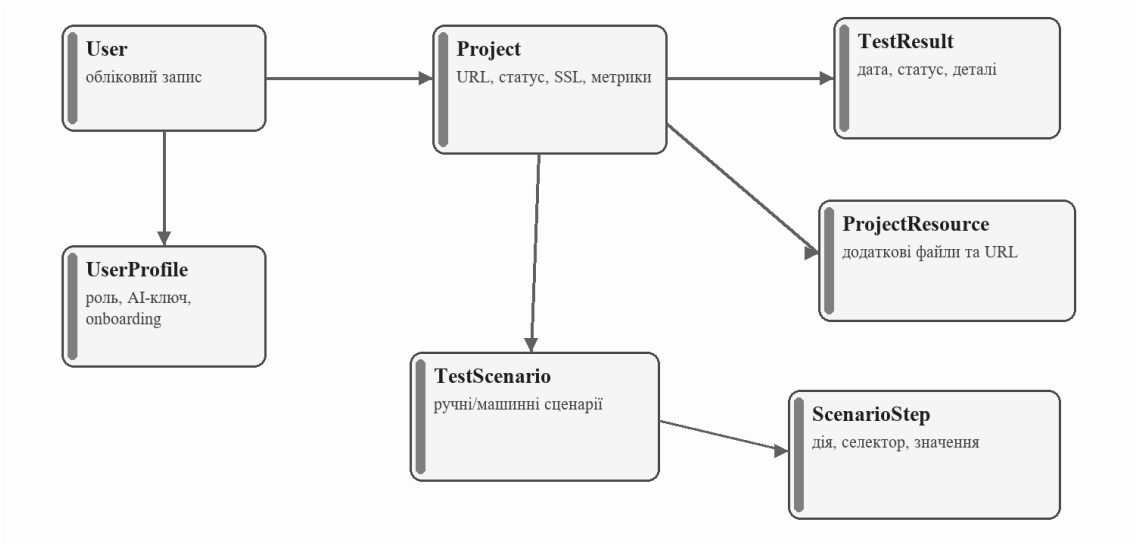

Рисунок 4.2 – Укрупнена модель даних системи

Укрупнена модель даних показує, як у системі пов’язані користувачі, вебпроєкти, результати перевірок, додаткові ресурси та сценарії тестування. Основною сутністю є проєкт, оскільки саме до нього прив’язуються всі технічні дані: URL-адреса, статус, метрики, історія запусків, ресурси, звіти та відповідальні користувачі. Користувач може бути власником проєкту або учасником команди, наприклад тестувальником, розробником чи менеджером. Завдяки цьому система не відкриває всі дані всім користувачам, а показує лише ті сайти, до яких людина має доступ. Це важливо для безпеки та для зручної командної роботи.

Результати перевірок зберігаються окремо, тому система може показувати не тільки поточний стан сайту, а й історію змін. Додаткові ресурси дозволяють контролювати пов’язані CSS, JS, API або інші файли, а сценарії

тестування зберігають послідовність дій користувача та результати їх виконання. Така структура робить систему не просто панеллю поточного статусу, а повноцінним інструментом для накопичення й аналізу даних.

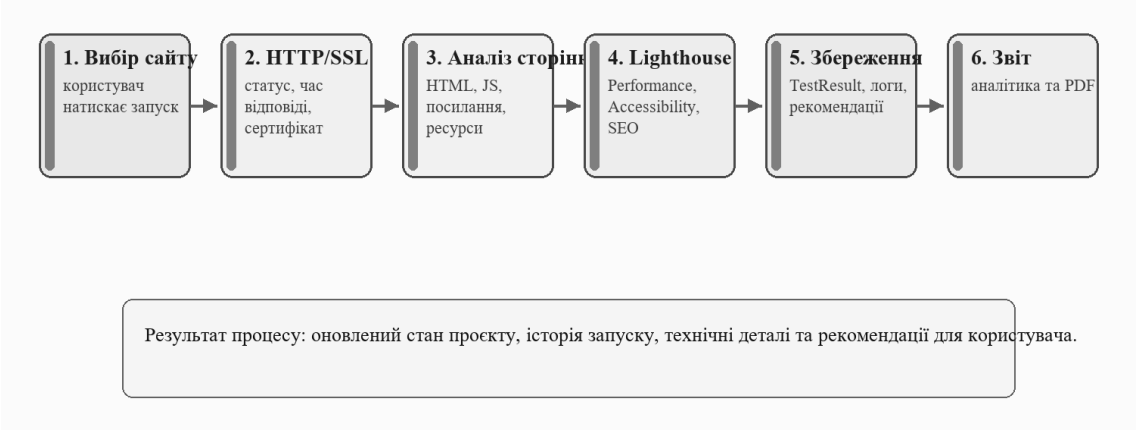

Рисунок 4.3 – Процес виконання перевірки вебсайту

Процес перевірки вебсайту складається з кількох послідовних технічних етапів, але для користувача він виглядає як одна проста дія – натискання кнопки запуску. Такий підхід зроблено спеціально, щоб не змушувати користувача окремо відкривати консоль, запускати різні інструменти або вручну збирати результати з кількох джерел. Після запуску система сама визначає, які перевірки потрібно виконати для конкретного проєкту, і поступово збирає технічні дані про стан вебресурсу.

Спочатку система надсилає HTTP-запит до сайту і визначає, чи доступний ресурс, який код відповіді повертає сервер і скільки часу займає відповідь. На цьому етапі формується базове розуміння стану сайту: чи відкривається сторінка, чи немає помилки сервера, чи не перевищує відповідь допустимий час очікування. Якщо сайт не відповідає або повертає критичний статус, система фіксує це як важливу проблему, але за можливості не зупиняє інші доступні етапи аналізу.

Після первинної перевірки доступності аналізуються додаткові технічні параметри. До них належать SSL-сертифікат, базова структура сторінки, JavaScript-помилки, SEO-ознаки, швидкодія, додаткові ресурси та, за потреби,

биті посилання. Такий набір перевірок дозволяє оцінити сайт не тільки з погляду “працює або не працює”, а й з погляду якості його технічної реалізації.

Окремо враховуються додаткові ресурси, які користувач може прив’язати до проєкту. Це можуть бути CSS-файли, JavaScript-файли, API-адреси, XML-карти або інші URL, важливі для роботи сайту. Система перевіряє їхній статус, тип відповіді, час завантаження та можливі помилки. Завдяки цьому моніторинг не обмежується лише головною сторінкою, а охоплює пов’язані елементи вебресурсу.

Якщо для проєкту створені користувацькі сценарії, система може також перевірити окремі дії на сайті, наприклад перехід на сторінку або наявність потрібного тексту. Це наближує систему до функціонального тестування, оскільки перевіряється не лише технічна відповідь сервера, а й поведінка сайту з погляду користувацьких дій.

Результати всіх етапів зберігаються в системі та використовуються для оновлення статусу сайту, логів, графіків, аналітики й PDF-звіту. Кожен запуск формує нові дані для історії перевірок, тому користувач може бачити не тільки поточний стан, а й динаміку змін. Це допомагає відрізнити випадковий збій від повторюваної проблеми.

Завдяки такій організації користувач отримує не просто окремий технічний показник, а загальну картину стану вебресурсу з поясненням, які проблеми варто переглянути першими. Процес перевірки поєднує автоматизований збір даних, збереження історії та зручне подання результатів, що робить систему придатною як для навчальної демонстрації, так і для практичного попереднього контролю вебсайтів.

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Розробка вебсистеми BitGuard пов'язана переважно з роботою за персональним комп'ютером, тому основними факторами впливу є тривале зорове навантаження, статична поза, робота з електрообладнанням, психоемоційне навантаження під час налагодження програмного забезпечення та необхідність дотримання правил безпечної поведінки у приміщенні.

5.1 Організація робочого місця користувача

Для BitGuard цей розділ не є формально відокремленим від теми роботи, оскільки система розрахована на тривалу роботу за ноутбуком або персональним комп'ютером. Користувач може аналізувати таблиці, логи, графіки та звіти протягом значного часу, тому ергономіка робочого місця прямо впливає на уважність і якість прийняття рішень.

Робоче місце має бути організоване так, щоб екран розташовувався приблизно на рівні очей, а відстань до монітора становила орієнтовно 50-70 см. Клавіатура та миша повинні розміщуватися таким чином, щоб користувач не перенапружував кисті та плечовий пояс.

Освітлення приміщення має бути рівномірним і не створювати відблисків на екрані. Під час тривалої роботи доцільно робити короткі перерви, змінювати положення тіла та виконувати прості вправи для очей.

Основні вимоги до безпечної роботи за комп'ютером подано в табл. 5.1.

Таблиця 5.1 – Основні вимоги до безпечної роботи за комп'ютером

Фактор	Можливий ризик	Заходи зниження ризику
Зорове навантаження	Втома очей, зниження концентрації	Регулярні перерви, достатнє освітлення, коректна яскравість екрана

Продовження Таблиці 5.1

Фактор	Можливий ризик	Заходи зниження ризику
Статична поза	Напруження спини, шиї та рук	Ергономічне крісло, правильна висота столу, зміна положення тіла
Електрообладнання	Пошкодження кабелів, коротке замикання	Перевірка стану кабелів, недопущення вологи біля техніки
Психоемоційне навантаження	Втома під час налагодження та тестування	Планування роботи, перерви, поділ задач на невеликі етапи

Це особливо важливо під час тестування інтерфейсу, коли користувач довго концентрується на дрібних елементах сторінки.

5.2 Безпека під час експлуатації вебсистеми

Під час роботи з системою користувач не повинен вводити приватні ключі, паролі або токени у відкриті поля, які не призначені для секретних даних.

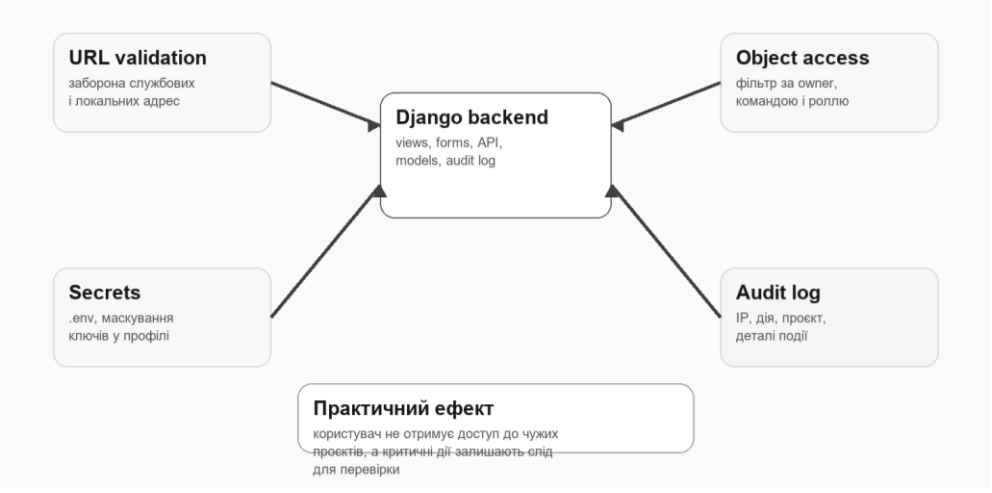

Рисунок 5.1 – Основні контури безпеки під час експлуатації системи

Якщо для аналітичних функцій використовується API-ключ, він має зберігатися у захищених змінних середовища, що відповідає загальним рекомендаціям із захисту автентифікаційних даних [11].

Під час роботи система виконує HTTP-запити до адрес, які вводить користувач, тому безпека таких дій має велике значення; відповідні ризики враховано з огляду на рекомендації OWASP [10-12]. Якщо не перевіряти URL, користувач випадково або навмисно міг би запустити запит до локальної, приватної чи службової адреси. Саме тому в системі передбачено обмеження для небезпечних URL і контроль того, які ресурси можна перевіряти.

Ще одним важливим контуром безпеки є розмежування доступу до даних. Користувач повинен бачити тільки власні проєкти або ті сайти, до яких він доданий як відповідальна особа. Це зменшує ризик перегляду або зміни чужих даних. Для цього в системі використовується перевірка поточного користувача, рольова модель і командна видимість.

Окремо враховано роботу з секретами та службовими ключами. Такі дані не повинні зберігатися у відкритому вигляді в коді або випадково потрапляти до репозиторію. Додатково важливі дії, наприклад створення проєкту, запуск перевірки, експорт звіту або заблокована операція, можуть фіксуватися в журналі безпеки. Це допомагає зрозуміти, хто і коли виконував певні дії в системі.

5.3 Дії в надзвичайних ситуаціях

У разі появи запаху гару, іскріння, перегріву обладнання або пошкодження кабелю необхідно припинити роботу, від'єднати пристрій від електромережі за умови безпечної можливості та повідомити відповідальну особу. Не допускається самостійно ремонтувати електрообладнання без відповідної кваліфікації.

У разі пожежної тривоги або іншої надзвичайної ситуації користувач повинен залишити робоче місце, не створювати перешкод для евакуації, дотримуватися вказівок відповідальних осіб і використовувати визначені маршрути

виходу. Збереження програмних даних не повинно мати пріоритет над безпекою людей.

У контексті використання BitGuard важливо пам'ятати, що збереження результатів перевірки, логів або звіту не може бути важливішим за безпеку користувача. Якщо робота перервалася через аварійну ситуацію, систему можна повторно запустити пізніше, а дані перевірок відновити з бази або сформувати новий звіт після усунення небезпеки.

Також у разі раптового зникнення електроживлення або збою обладнання не слід намагатися негайно повторно вмикати пристрій, доки не буде встановлено причину несправності. Якщо під час роботи виникла загроза втрати даних, відновлення інформації має виконуватися лише після усунення небезпеки та перевірки справності обладнання. Усі нестандартні ситуації доцільно фіксувати та повідомляти відповідальній особі, щоб запобігти повторенню подібних випадків у майбутньому.

5.4 Психоемоційне навантаження під час роботи з системою

Під час роботи з вебсистемою BitGuard користувач може тривалий час аналізувати технічні показники, логи, повідомлення про помилки та результати перевірок. Така діяльність потребує уважності, оскільки неправильна інтерпретація критичного статусу, SSL-помилки або результатів сценарію може вплинути на подальші рішення щодо вебресурсу. Тому важливо уникати перевтоми, робити короткі перерви та не виконувати складний аналіз у стані сильного стресу або поспіху.

Для зменшення психоемоційного навантаження інтерфейс системи має подавати інформацію структуровано: критичні проблеми окремо, попередження окремо, а деталі — у логах або звітах. Це дозволяє користувачу поступово розбирати проблему, а не сприймати всі технічні дані одночасно. У разі великої кількості помилок доцільно спочатку переглянути найкритичніші повідомлення, а потім переходити до менш важливих рекомендацій.

ВИСНОВКИ

На основі аналізу предметної області, існуючих сервісів моніторингу, інструментів аудиту вебсторінок і потреб користувача було обґрунтовано доцільність створення єдиної вебсистеми для контролю якості сайтів. У роботі визначено функціональні та нефункціональні вимоги, спроєктовано архітектуру, структуру даних і сценарії використання системи.

У результаті виконання кваліфікаційної роботи, оформленої відповідно до вимог до звітної документації [1; 2], розроблено вебсистему BitGuard для моніторингу та ручного/машинного тестування вебсайтів. Реалізовано керування вебпроєктами, ручний та плановий запуск перевірок, аналіз SSL, продуктивності, доступності, SEO, JavaScript-помилок, додаткових ресурсів, битих посилань, сценаріїв користувача, логів, графіків, профілю, командної взаємодії та PDF-звітів. Найважливішим результатом є те, що ці функції зібрані в одному вебінтерфейсі, а не залишені набором окремих консольних або зовнішніх інструментів.

Під час роботи над проєктом частина рішень змінювалася після практичної перевірки. Зокрема, було доопрацьовано подання звітів, поведінку панелей налаштувань, додаткові ресурси, профіль користувача та логіку доступу до проєктів. Це дозволило зробити систему ближчою до реального робочого інструмента, а не лише до навчального прототипу.

Розробка дозволяє користувачу виконувати базовий контроль вебресурсу без роботи з консоллю, бачити поточний стан сайту, зберігати історію перевірок, працювати з командою, отримувати структуровані рекомендації та формувати звітні матеріали. Окремо враховано вимоги безпеки: секрети винесено до середовища, доступ до об'єктів обмежено власником, роллю і причетністю до проєкту, а важливі дії фіксуються у журналі безпеки.

Отриманий результат може використовуватися як навчальний і прикладний інструмент для демонстрації процесу забезпечення якості вебресурсів.

ПЕРЕЛІК ПОСИЛАНЬ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання.

2. ДСТУ ГОСТ 7.84:2008. Система стандартів з інформації, бібліотечної та видавничої справи. Загальні вимоги та правила оформлення.

3. Django Documentation. The web framework for perfectionists with deadlines. URL: https://docs.djangoproject.com/

4. Celery Documentation. Distributed Task Queue. URL: https://docs.celeryq.dev/

5. Playwright Documentation. Fast and reliable end-to-end testing for modern web apps. URL: https://playwright.dev/

6. Google Lighthouse. Web performance auditing tool. URL: https://developer.chrome.com/docs/lighthouse/

7. UptimeRobot. Website monitoring service. URL: https://uptimerobot.com/

8. Pingdom. Website monitoring and performance management. URL: https://www.pingdom.com/

9. Selenium Documentation. Browser automation framework. URL: https://www.selenium.dev/documentation/

10. OWASP Foundation. Web Security Testing Guide. URL: https://owasp.org/www-project-web-security-testing-guide/

11. OWASP Foundation. Authentication Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

12. OWASP Foundation. Access Control Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html

13. Redis Documentation. In-memory data structure store. URL: https://redis.io/docs/

14. Beautiful Soup Documentation. HTML and XML parsing library. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

15. Bootstrap Documentation. URL: https://getbootstrap.com/docs/

16. Chart.js Documentation. URL: https://www.chartjs.org/docs/

ДОДАТОК А
ФРАГМЕНТ СТРУКТУРИ ПРОЄКТУ

У додатку наведено укрупнену структуру основних файлів вебзастосунку BitGuard. Повний програмний код подається в електронних матеріалах, а нижче показано ті частини, які найкраще пояснюють логіку проєкту.

```
diplom_qa/
  manage.py - службовий файл запуску Django-команд, міграцій та локального сервера
  db.sqlite3 - локальна база даних для демонстраційної версії
  diplom_qa/
    settings.py - підключення застосунків, бази даних, статичних файлів, змінних середовища та мовних параметрів
    urls.py - головна маршрутизація проєкту
  core/
    models.py - моделі Project, TestResult, ProjectResource, UserProfile, SecurityAuditLog, ProjectChatMessage, TestScenario, ScenarioStep, ScenarioResult
    views.py - сторінки інтерфейсу, API, ролі, команда, чат, експорт, Analytics Pro та запуск перевірок
    tasks.py - фонові перевірки доступності, SSL, технічного аудиту, браузерних сценаріїв, ресурсів і сповіщень
    forms.py - форми проєктів, сценаріїв, профілю, реєстрації, ролей і налаштувань
    security.py - перевірка безпечності URL та обмеження частоти запусків
    tests.py - базові перевірки доступу, ресурсів, експорту та безпечності дій
    templates/core/ - сторінки моніторингу, додавання сайту, детальної аналітики, Analytics Pro, логів, профілю, команди, чату, налаштувань, сценаріїв та онбордингу
    static/core/ - CSS, JavaScript, зображення, анімації, іконки та клієнтська логіка інтерфейсу
    management/commands/run_bot.py - команда підключення Telegram-бота для сповіщень
    migrations/ - історія змін структури бази даних
  media/ - завантажені аватари користувачів, скріншоти сайтів і результати сценаріїв
  run.bat - файл швидкого запуску демонстраційної версії без ручної роботи з консоллю
  requirements.txt / .env.example - залежності та приклад службових змінних середовища
```

ДОДАТОК Б
БЛАНК ЗАЯВИ СТУДЕНТА ПРО ОРИГІНАЛЬНІСТЬ РОБОТИ

Заява студента про оригінальність роботи

_____Пашкова Ганна Олегівна_____

ПІБ студента

Номер залікової книжки: _____204-22_____

Засвідчую, що кваліфікаційна робота на тему

«Розробка системи автоматизованого функціонального та регресійного тестування веб-платформи» зі спеціальності 121 – Інженерія програмного забезпечення була підготовлена виключно мною і не порушує авторські права третіх осіб відповідно до закону про авторське право; повністю або частково не була використана як основа для отримання диплома про вищу освіту або науковий ступінь мною або іншою особою.

Представлена мною для перевірки електронна версія роботи збігається з друкованим примірником.

Підтверджую, що був(ла) проінформований(на) про права та обов'язки здобувача вищої освіти Університету та правила, що стосуються перевірки оригінальності наукових робіт. Згоден(на) на обробку моєї письмової роботи й архівування цієї роботи в базі даних відповідно до антиплагіатних правил і процедур Університету.

З Положенням про академічну доброчесність у Криворізькому національному університеті ознайомлений(на).

__09.06.2026__ _____ _____Г. О. Пашкова_____

(дата) (підпис) (ініціали та прізвище автора роботи)

ДОДАТОК В ІНСТРУКЦІЯ КОРИСТУВАЧА ДЛЯ ДЕМОНСТРАЦІЇ СИСТЕМИ

Цей додаток містить демонстраційний маршрут, за яким можна швидко перевірити роботу BitGuard перед захистом або показом керівнику. Інструкція побудована так, щоб послідовно показати не лише окремі кнопки, а логіку всієї системи: від створення акаунта до аналізу результатів і формування звіту.

1. Підготовка запуску. Відкрити проєкт через файл run.bat або локальний сервер, перейти на сторінку входу та переконатися, що інтерфейс відкривається без помилок.

2. Створення акаунта. Відкрити сторінку реєстрації, ввести логін, email, ім’я та обрати потрібну роль. Після реєстрації перевірити повідомлення про підтвердження email.

3. Погодження ролі. Якщо користувач запитує роль QA Engineer, Developer або Project Manager, відповідальна особа має підтвердити її у розділі рольових запитів. Звичайному користувачу службова роль адміністратора не показується.

4. Ознайомлення з інтерфейсом. Після входу пройти коротку екскурсію сайтом: переглянути бокове меню, сторінку моніторингу, профіль, аналітику, логи та налаштування.

5. Додавання сайту. Перейти до форми додавання, вказати назву, URL, тип ресурсу, профіль перевірки, інтервал активного моніторингу, режим пристрою та відповідальних осіб.

6. Додатковий контроль. У формі проєкту увімкнути пошук ключового слова, сканування битих посилань або навчальний режим для локального сайту, якщо це потрібно для демонстрації.

7. Додавання ресурсів. У таблиці моніторингу відкрити додаткові ресурси та додати CSS, sitemap, API або інший URL, який має перевірятися разом з основною сторінкою.

8. Запуск перевірки. Натиснути кнопку запуску в рядку конкретного сайту. Під час виконання звернути увагу на зміну візуального стану рядка і підказку про поточний етап перевірки.

9. Перегляд результату. Після завершення перевірки переглянути HTTP-статус, uptime, SSL, оцінки продуктивності, SEO, JS-помилки, ресурси та короткі рекомендації.

10. Аналіз логів. Відкрити вкладку логів та історії, перевірити дату запуску, сайт, статус, критичність і текстові деталі. Це підтверджує, що система зберігає історію, а не тільки показує поточний стан.

11. Analytics Pro. Відкрити сторінку аналітики, обрати сайт, позначити потрібні блоки даних і сформувати вибірковий зріз. Якщо даних немає, спочатку виконати хоча б одну перевірку сайту.

12. Детальна аналітика. Запустити поглиблений технічний аналіз, дочекатися готовності структурованого звіту і переглянути резюме, критичні проблеми, план виправлень та висновок.

13. Конструктор графіків. Перейти до конструктора, обрати сайт, метрики та тип візуалізації. Показати, як графіки допомагають оцінювати динаміку доступності, часу відповіді або помилок.

14. Сценарії тестування. Відкрити сценарії проєкту, створити тест із кроками, запустити його і переглянути результат виконання. Для демонстрації достатньо простого сценарію з переходом на сторінку та перевіркою тексту.

15. PDF-звіт. Натиснути кнопку PDF у таблиці моніторингу, обрати блоки, які потрібно включити, і перевірити, що результат має зрозумілу структуру та не містить приватних ключів.

16. Команда і чат. Відкрити сторінку команди, переконатися, що видно лише пов’язаних з проєктами користувачів, а потім перейти до чату конкретного проєкту.

17. Сповіщення. За потреби відкрити блок Telegram-підключення і показати, що система передбачає повідомлення про критичні події без постійного ручного перегляду сторінки.

18. Завершення демонстрації. Показати профіль користувача, узагальнений індекс стану проєктів, швидкі дії та налаштування акаунта. Після цього можна сформувати підсумковий PDF або повторити перевірку для іншого сайту.

ДОДАТОК Г КОНТРОЛЬНИЙ ПЕРЕЛІК ФУНКЦІЙ ПЕРЕД ЗАХИСТОМ

Таблиця Г.1 – Чеклист функцій для фінального тестування

№	Функція	Що перевірити	Відмітка
1	Реєстрація та email	Акаунт створюється, email має стан підтвердження після переходу за посиланням	□
2	Ролі	Підвищена роль очікує погодження, admin не показується звичайному користувачу	□
3	Моніторинг	Проєкт додається, інтервал зберігається, ручна перевірка оновлює рядок	□
4	Додаткові ресурси	Ресурс додається без блокування модального вікна і перевіряється разом із сайтом	□
5	Логи	У журналі є дата, статус, критичність і деталі запуску	□
6	Analytics Pro	Сайт обирається, чекбокси працюють, без результатів показується правильна підказка	□
7	PDF	Звіт має читабельне оформлення і не містить секретів	□
8	Профіль	Аватар відкривається, роль і персональні дані відображені коректно	□

№	Функція	Що перевірити	Відмітка
9	Команда і чат	Користувач бачить лише причетних учасників і може писати у межах доступного проєкту	□
10	Адаптивність	Основні сторінки не виходять за межі екрана на desktop і mobile	□

ДОДАТОК Д СЛОВНИК ОСНОВНИХ ТЕРМІНІВ І ПОЗНАЧОК СИСТЕМИ

Таблиця Д.1 – Пояснення термінів, які використовуються в інтерфейсі

Термін	Пояснення
Uptime	Відсоток успішних перевірок доступності сайту за збереженою історією запусків.
HTTP-статус	Код відповіді сервера: 200 означає успішну відповідь, 4xx - помилку запиту, 5xx - проблему сервера.
SSL	Ознака захищеного з'єднання і строк чинності сертифіката.
Performance	Оцінка швидкодії сторінки за результатами Lighthouse або збережених технічних метрик.
Accessibility	Оцінка доступності інтерфейсу для користувачів з різними потребами.
SEO	Показники заголовку, опису, H1 та альтернативних описів зображень.
JS-помилки	Помилки JavaScript або мережеві помилки, зафіксовані під час браузерного сканування.
Додатковий ресурс	Окремий URL, пов'язаний із сайтом: CSS, JS, sitemap, API або файл, який варто перевіряти додатково.
Сценарій	Набір кроків, що описує машинну перевірку дій користувача на сайті.
Analytics Pro	Сторінка вибіркового технічного зрізу, де користувач обирає сайт і потрібні блоки даних.
Critical / High / Medium	Позначки критичності, які допомагають визначити порядок виправлення проблем.