Багаторівневі підходи щодо безпеки веб-орієнтованих систем

УДК 004.056

Александрос Фотинос[1], Лариса Шумова[2]

Східноукраїнський національний університет імені Володимира Даля,
[1]photinosaleksandros12@gmail.com, [2]shumova@snu.edu.ua

Одним із критично важливих факторів, що визначають успішність веб-системи, є її безпека. Механізми безпеки сучасних веб-орієнтованих систем реалізовані на основі багаторівневого підходу, що охоплює всі компоненти архітектури: клієнтську частину, серверну частину та рівень зберігання даних.

Схема на рисунку 1 демонструє три рівні захисту: клієнтська частина (React 18), серверна частина (NestJS) та рівень зберігання даних (PostgreSQL). Кожен рівень реалізує власні механізми безпеки, що забезпечують комплексний захист від основних векторів атак.

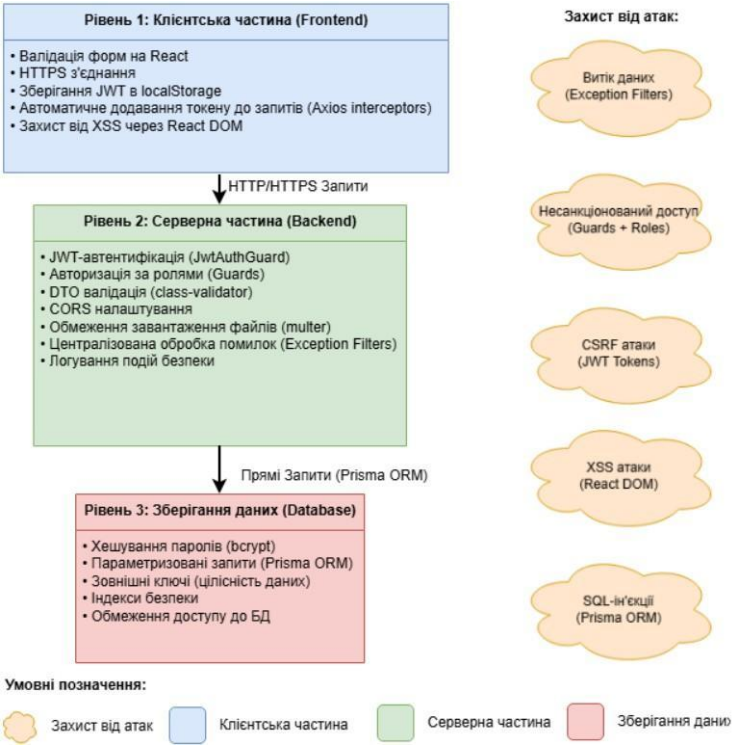

Рис.1. Багаторівневий підхід до безпеки веб-орієнтованої системи

Основним механізмом автентифікації є JWT-токени, які генеруються сервером після успішної перевірки облікових даних користувача. Токен містить ідентифікатор користувача, електронну пошту та роль, має обмежений час

життя двадцять чотири години та автоматично додається до всіх захищених запитів через інтерсептори HTTP-клієнта Axios. Такий підхід забезпечує stateless-автентифікацію, що спрощує масштабування системи та усуває необхідність зберігання стану сесій на сервері.

Ще одним рівнем безпеки є авторизація та розмежування прав доступу, що реалізовано за допомогою механізму Guards у фреймворку NestJS. Guard JwtAuthGuard перевіряє наявність та валідність JWT-токена для всіх захищених маршрутів, а додаткові перевірки ролі користувача обмежують доступ до функціоналу відповідно до призначення облікового запису. Розмежування реалізовано на рівні серверної логіки, що унеможливлює обхід обмежень через маніпуляції з клієнтської сторони.

Окремим рівнем безпеки є захист паролів користувачів, що забезпечується за допомогою алгоритму bcrypt. Паролі не зберігаються у відкритому вигляді в базі даних, а перед записом проходять процедуру хешування. Під час автентифікації введений пароль порівнюється зі збереженим хешем, що гарантує неможливість відновлення оригінального пароля навіть у випадку компрометації бази даних.

Захист від ін'єкційних атак забезпечується за рахунок використання Prisma ORM для взаємодії з базою даних PostgreSQL. Усі запити до бази даних формуються через типобезпечний API Prisma, що автоматично екранує вхідні параметри та використовує параметризовані запити. Це повністю усуває можливість SQL-ін'єкцій, оскільки ручне формування SQL-рядків у коді відсутнє.

Механізм CORS налаштований на серверній стороні для обмеження джерел, з яких дозволені запити до API. У конфігурації вказано конкретні дозволені origin, що запобігає міжсайтовим атакам та несанкціонованому використанню ендпоінтів з сторонніх доменів. Завантаження файлів реалізовано через бібліотеку multer з обмеженням максимального розміру файлу та збереженням у спеціальній директорії з унікальними іменами, що запобігає перезапису існуючих файлів та виконанню шкідливого коду через завантажені ресурси.

Обробка помилок реалізована централізовано через Exception Filters фреймворку NestJS. Користувачу не передаються деталі внутрішніх помилок сервера, що запобігає витоку інформації про структуру системи або версії використаних бібліотек. Помилки логуються на сервері для подальшого аналізу розробником, але клієнт отримує лише узагальнені повідомлення без технічного контексту. Такий підхід відповідає принципам безпеки через невідомість та мінімізує поверхню атаки.

Усі описані заходи багаторівневого підходу щодо безпеки веб-орієнтованої системи реалізовано й перевірено в процесі функціонального та інтеграційного тестування. Комплексний підхід до захисту даних, автентифікації, авторизації та валідації вхідних даних забезпечує відповідність системи сучасним вимогам до веб-додатків та гарантує безпеку веб-орієнтованої системи в онлайн-середовищі.