Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
«Підвищення захищеності веб-додатків шляхом застосування моделі об'єктних можливостей в архітектурі SPA»

Виконав: здобувач 2-го курсу, гр. КІТС-24м
Спеціальності 125 – Кібербезпека та захист інформації
Освітня програма – Кібербезпека інформаційних технологій та систем
Анщук О.А.
Науковий керівник: Карпінець В.В.
«____» _____ 2025 р.
Опонент: Захарченко С.М.
«____» _____ 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
_____Юрій ЯРЕМЧУК
"_____" _____2025 р.

Вінниця ВНТУ – 2025

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти II-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека та захист інформації
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС
_____ **Юрій ЯРЕМЧУК**
"_24_" _вересня_ 2025 р.

З А В Д А Н Н Я
на магістерську кваліфікаційну роботу студенту
_____Анщуку Олександру Анатолійовичу_____
(прізвище, ім`я, по-батькові)

1. Тема роботи_«Підвищення захищеності веб-додатків шляхом застосування моделі об'єктних можливостей в архітектурі SPA»__
Керівник роботи__Карпінець В. В., к.т.н., доц. каф. МБІС___
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "24" вересня 2025 року № 313

2. Строк подання студентом роботи за тиждень до захисту.

3. Вихідні дані до роботи: наукові статті та публікації, що стосуються обраної теми, монографії та наукові книги, які розкривають ключові аспекти теми, інтернет-ресурси.

4. Зміст текстової частини: у першому розділі виконати аналіз сучасної безпеки веб-додатків, розглянути архітектуру SPA та основні загрози; у другому розділі розробити удосконалену модель об'єктних можливостей; у третьому розділі створити програмний прототип; у четвертому розділі виконати економічне обґрунтування.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
— у першому розділі наведено рис. 1, табл. 5;
— у другому розділі наведено рис. 4;
— у третьому розділі наведено рис. 4, табл. 4;
— у четвертому розділі наведено табл. 4.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Карпінець В. В., к.т.н., доц. каф. МБІС		
2	Карпінець В. В., к.т.н., доц. каф. МБІС		
3	Карпінець В. В., к.т.н., доц. каф. МБІС		
4	Ратушняк О. Г., к.т.н., доц. каф. ЕПВМ		

7.Дата видачі завдання 24 вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи		Примітка
1.	Визначення напрямку та формулювання теми магістерської кваліфікаційної роботи	20.09.2025	28.09.2025	
2.	Аналіз предметної області обраної теми	29.09.2025	07.10.2025	
3.	Розробка алгоритму роботи	08.10.2025	16.10.2025	
4.	Написання магістерської кваліфікаційної роботи на основі розробленої теми	17.10.2025	15..11.2025	
5.	Розробка економічної частини	16.11.2025	19.11.2025	
6.	Апробація отриманих результатів	20.11.2025	23.11.2025	
7.	Проведення попереднього захисту магістерської кваліфікаційної роботи	24.11.2025	25.11.2025	
8.	Коригування роботи згідно отриманих рекомендацій	25.11.2025	04.12.2025	
9.	Захист магістерської кваліфікаційної роботи	08.12.2025	11.12.2025	

Студент _____ Анщук О. А.
(підпис)

Керівник роботи _____ Карпінець В. В.
(підпис)

АНОТАЦІЯ

УДК 004.056.55

Анщук О. А. Підвищення захищеності веб-додатків шляхом застосування моделі об’єктних можливостей в архітектурі SPA. Магістерська кваліфікаційна робота зі спеціальності 125 - Кібербезпека, освітня програма - Кібербезпека інформаційних технологій та систем. Вінниця: ВНТУ, 2025. 111с.

На укр. мові. Бібліогр.: 60 назв; рис.: 9; табл.:13.

У магістерській кваліфікаційній роботі здійснено підвищення рівня захищеності веб-додатків шляхом застосування моделі об’єктних можливостей в архітектурі типу Single Page Application (SPA).

Досліджено сучасні підходи до побудови архітектури веб-додатків, проаналізовано основні вразливості, притаманні SPA-додаткам, та актуальні методи їх захисту. Проведено аналіз моделей контролю доступу, а також визначено їх переваги та обмеження в контексті веб-безпеки.

Розроблено вдосконалену модель керування доступом на основі об’єктних можливостей, адаптовану до архітектури SPA. Запропонована модель забезпечує підвищення рівня ізоляції користувацьких прав і зменшення ризику несанкціонованого доступу до ресурсів.

Програмно реалізовано прототип системи з інтегрованою моделлю об’єктних можливостей. Проведено тестування розробленого рішення, результати якого підтвердили зростання ефективності та безпечності обробки запитів користувачів у SPA-додатках.

Наостанок, здійснено оцінювання практичної придатності та потенційного комерційного використання запропонованого підходу.

Ключові слова: веб-додаток, SPA, інформаційна безпека, модель об’єктних можливостей, контроль доступу, кіберзахист.

ABSTRACT

Anshchuk O. A. Improving the security of web applications by applying the object capabilities model in SPA architecture. Master's qualification work in the specialty 125 - Cybersecurity, educational program - Cybersecurity of Information Technologies and Systems. Vinnytsia: VNTU, 2025. 111 p.

In Ukrainian. Bibliography: 60 titles; figures: 9; tables: 13.

The master's thesis improves the security of web applications by applying an object capabilities model in Single Page Application (SPA) architecture.

Modern approaches to building web application architecture are investigated, the main vulnerabilities inherent in SPA applications are analyzed, and current methods of protecting them are discussed. An analysis of access control models is conducted, and their advantages and limitations in the context of web security are identified.

An improved access control model based on object capabilities and adapted to SPA architecture has been developed. The proposed model provides a higher level of isolation of user rights and reduces the risk of unauthorized access to resources.

A prototype system with an integrated object capabilities model has been implemented in software. The developed solution has been tested, and the results confirm an increase in the efficiency and security of user request processing in SPA applications.

Finally, the practical applicability and potential commercial use of the proposed approach were evaluated. The implementation of the developed model can increase the level of cyber protection of modern web applications and reduce the number of successful attacks at the business logic level.

Keywords: web application, SPA, information security, object capability model, access control, cyber security.

ЗМІСТ

ВСТУП

Актуальність. Сучасні веб-додатки є ключовою складовою цифрової інфраструктури, що забезпечує функціонування бізнес-процесів, соціальних мереж, фінансових систем та державних сервісів. З розвитком технологій веб-розроблення особливої популярності набула архітектура Single Page Application (SPA), яка дозволяє створювати інтерактивні, швидкі та зручні у використанні веб-системи. Проте, зростання складності SPA-додатків супроводжується збільшенням кількості потенційних вразливостей і ризиків, пов’язаних із несанкціонованим доступом, підробкою запитів, витоком конфіденційної інформації та атаками на рівні бізнес-логіки.

Традиційні моделі контролю доступу, зокрема рольно-орієнтована (RBAC) та атрибутивна (ABAC), не завжди забезпечують достатню гнучкість і безпечність у динамічних веб-середовищах. Це створює потребу у використанні моделі об’єктних можливостей (Object Capabilities Model), яка забезпечує більш чітке управління правами доступу через передачу об’єктів-дозволів. Такий підхід дозволяє мінімізувати ризики ескалації привілеїв, зменшити кількість потенційних атак і забезпечити суворішу ізоляцію компонентів SPA-додатку.

Отже, розроблення та впровадження моделі об’єктних можливостей у архітектуру SPA є актуальним завданням у сфері кібербезпеки веб-додатків, оскільки воно сприяє підвищенню рівня захищеності сучасних інформаційних систем. Дослідженнями, пов’язаними із захистом веб-додатків, займалися такі науковці, як Джон Б. Денніс, Генрі М. Леві, Батлер В. Лампсон та інші [1–3].

У роботі досліджено архітектурні особливості SPA-додатків, основні вразливості веб-застосунків, а також підходи до моделювання контролю доступу. Запропоновано використання моделі об’єктних можливостей, що дозволяє підвищити рівень безпеки системи за рахунок чіткої ізоляції компонентів і передачі прав доступу через безпечні об’єкти. Розроблено архітектурну модель SPA-додатку з інтегрованою системою об’єктних можливостей, яка забезпечує гнучке

управління доступом користувачів та сервісів. Реалізовано прототип програмного рішення, проведено тестування та оцінку рівня безпеки у порівнянні з традиційними моделями контролю доступу.

Мета і задачі дослідження. Метою магістерської роботи є підвищення захищеності веб-додатків шляхом застосування моделі об’єктних можливостей в архітектурі SPA.

Для досягнення поставленої мети необхідно вирішити такі задачі:
− здійснити аналіз сучасних архітектур веб-додатків та моделей контролю доступу;
− дослідити особливості архітектури SPA та визначити її вразливості;
− спроєктувати модель доступу на основі об’єктних можливостей, адаптовану до SPA-додатків;
− реалізувати прототип веб-додатку з інтегрованою системою об’єктних можливостей;
− провести тестування та оцінку ефективності розробленого підходу щодо рівня безпеки та продуктивності;
− виконати економічне обґрунтування впровадження розробленого рішення.

Об’єкт дослідження – процес забезпечення безпеки доступу у веб-додатках архітектури SPA.

Предмет дослідження – модель об’єктних можливостей як засіб підвищення захищеності веб-додатків.

Наукова новизна полягає у вдосконаленні підходів до управління доступом у SPA-додатках шляхом адаптації моделі об’єктних можливостей, що забезпечує підвищення рівня ізоляції компонентів і зменшення ризику несанкціонованого доступу.

Практична цінність роботи полягає у створенні програмного прототипу веб-додатку з реалізованою моделлю об’єктних можливостей.

1. АНАЛІЗ МОЖЛИВОСТЕЙ ПІДВИЩЕННЯ ЗАХИЩЕНОСТІ СУЧАСНИХ ВЕБ-ДОДАТКІВ

У цьому розділі здійснюється комплексний аналіз теоретичних основ та сучасного стану захисту веб-додатків, з особливим фокусом на архітектурі односторінкових веб-додатків (SPA). Досліджено актуальність обраної галузі в контексті сучасних викликів та загроз. Розглядається еволюція веб-технологій та аналізуються специфічні загрози безпеки, що виникають у SPA-середовищі. Детально досліджуються недоліки традиційних ідентифікаційно-орієнтованих моделей контролю доступу та проблема “навколишньої влади”, що створює фундаментальні вразливості у сучасних веб-додатках.

1.1 Аналіз актуальності та передумов підвищення безпеки веб-додатків

Дослідження безпеки веб-додатків, зокрема архітектури односторінкових додатків (SPA, Single Page Application), набуває особливої актуальності в умовах стрімкої цифрової трансформації, розвитку інноваційних технологій, зростання залежності критичної інфраструктури від веб-сервісів та посилення загроз у сфері кібербезпеки. У сучасному технологічному середовищі веб-додатки є не просто інструментом для взаємодії з кінцевим користувачем — вони перетворились на центральну точку доступу до цифрових ресурсів компаній, державних структур та цілих національних систем.

Згідно з аналітичним звітом Gartner (2023), понад 70% нових корпоративних IT-рішень реалізуються на основі SPA-архітектури. Це свідчить про її стабільне домінування в розробці сучасних веб-застосунків, включно з системами електронного урядування, онлайн-банкінгом, телемедициною, торговими платформами, платформами управління даними, хмарними сервісами тощо. Односторінкові веб-додатки забезпечують високу швидкодію, масштабованість, адаптивність до пристроїв і персоналізацію інтерфейсу, проте водночас створюють додаткові виклики у сфері інформаційної безпеки [4].

Звіт IBM Security "Cost of a Data Breach 2023" констатує, що середня вартість інциденту витоку даних у веб-додатках зросла до 4,45 млн доларів США, що на 15% більше, ніж у 2020 році. У фінансовому секторі цей показник сягає 5,9 млн доларів, що свідчить про серйозні економічні наслідки недостатнього захисту веб-інфраструктури. Згідно з даними Ponemon Institute, 43% кібератак спрямовані саме на веб-додатки, при цьому 68% із них пов'язані з порушенням контролю доступу — однією з найкритичніших вразливостей сучасного Інтернету [5-6].

Фонд OWASP (Open Web Application Security Project) постійно фіксує порушення контролю доступу (Broken Access Control) як провідну загрозу — вона займає перше місце в OWASP Top 10 з 2021 року (рис. 1.1). Актуальність цієї вразливості посилюється тим, що у SPA-додатках значна частина логіки переміщена на клієнтську сторону, що ускладнює реалізацію надійних моделей авторизації та створює більше простору для маніпуляцій з боку зловмисника.

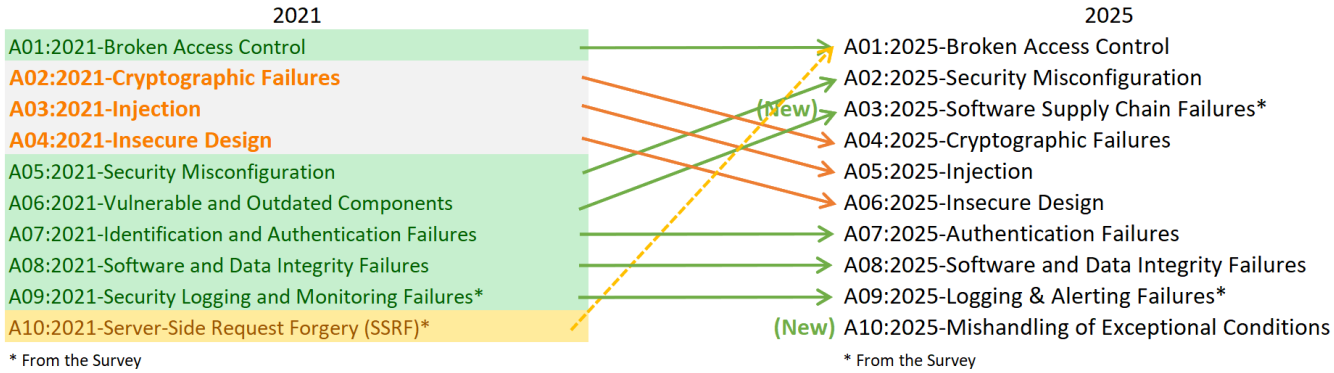


Рисунок 1.1 – Топ 10 ризиків безпеки веб-додатків [7]

Згідно з результатами дослідження, екосистема JavaScript демонструє стрімке зростання та посідає провідні позиції серед мов програмування. JavaScript є основою для розробки більшості сучасних односторінкових веб-додатків (SPA), що зумовлює його ключову роль у формуванні архітектури клієнтської частини веб-застосунків [8].

Станом на 2025 рік, згідно з даними NPM Registry, кількість опублікованих JavaScript-пакетів перевищує 2,1 мільйона. Така масштабна екосистема створює значний потенціал для зловмисників, зокрема у контексті атак на ланцюг постачання, де вразливості можуть бути інтегровані на рівні залежностей. Кількість

подібних атак зростає щороку, створюючи загрозу не лише для приватного сектору, але й для національної безпеки [9].

За інформацією, оприлюдненою Європейським агентством з кібербезпеки (ENISA), протягом останнього року кількість атак на веб-додатки, що обслуговують критичну інфраструктуру, зросла на 87%. Особливу стурбованість викликає ситуація у сфері охорони здоров'я: 89% медичних організацій зазнали витоків конфіденційної інформації впродовж останніх двох років, при цьому 56% інцидентів були пов'язані з недостатнім контролем доступу у веб-застосунках [10].

На національному рівні, акцентовано увагу на необхідності забезпечення захисту критичної інформаційної інфраструктури. За даними Державної служби спеціального зв'язку та захисту інформації України, з початку 2022 року кількість кібератак на державні та комерційні веб-ресурси зросла більш ніж у десять разів. Значна частина цих атак експлуатує вразливості саме у клієнт-серверній взаємодії, що знову ж таки підкреслює важливість дослідження моделей безпеки у контексті сучасних веб-архітектур [11].

Окрему категорію викликів становить стрімкий розвиток інноваційних веб-технологій, таких як Web3, прогресивні веб-додатки (PWA), а також WebAssembly. Усе більше компаній та індивідуальних розробників планують поступовий перехід до складніших клієнтських архітектур, що вимагає переосмислення підходів до побудови моделей довіри, автентифікації та контролю доступу.

Крім того, на горизонті з'являється ще одна довгострокова загроза — розвиток квантових обчислень. Вони потенційно здатні зруйнувати традиційні криптографічні механізми, які наразі забезпечують безпеку веб-комунікацій. У зв'язку з цим Національний інститут стандартів і технологій США (NIST) активно працює над стандартизацією постквантових криптографічних алгоритмів. Цей процес підкреслює нагальну необхідність формування квантово-стійких архітектур безпеки, які мають бути враховані вже на етапі проєктування сучасних веб-додатків [12].

Таким чином, представлена у цьому підрозділі аналітика підтверджує критичну актуальність дослідження альтернативних моделей безпеки для односторінкових веб-додатків. Зростаючі кіберзагрози, економічні ризики, суворі регуляторні вимоги, а також технологічні зміни формують стійкий запит на переосмислення традиційних підходів до реалізації контролю доступу. У цьому контексті дослідження моделей безпеки, заснованих на принципах доступу на основі можливостей (capability-based access control), становить не лише академічний інтерес, але й набуває надзвичайної практичної цінності.

1.2. Аналіз особливостей архітектури SPA як передумови для підвищення захищеності

Архітектура односторінкових веб-додатків (Single Page Application, SPA) є результатом глибинної трансформації підходів до веб-розробки, що зумовлена зростаючими вимогами до інтерактивності, адаптивності, швидкодії та зручності користувацького інтерфейсу. Традиційна модель багатосторінкових застосунків (Multi-Page Applications, MPA), заснована на повному перезавантаженні сторінки після кожної взаємодії з сервером, перестала відповідати очікуванням користувачів, які прагнуть безперервного досвіду, подібного до нативних мобільних або настільних додатків.

Відмова від MPA-моделі на користь SPA дозволила реалізувати динамічне оновлення контенту в межах однієї HTML-сторінки без перезавантаження. Це суттєво підвищило продуктивність та зручність використання веб-застосунків, а також оптимізувало використання мережевих ресурсів за рахунок зменшення обсягів передаваних даних.

Поняття "односторінковий веб-додаток" було вперше введене у публікації "Ajax: A New Approach to Web Applications", яка заклала основи асинхронного оновлення контенту за допомогою XMLHttpRequest без повного перезавантаження сторінки. Цей підхід започаткував нову еру у веб-розробці, стимулювавши перехід до клієнт-орієнтованої архітектури [13].

Ще у 2002 році Мартін Фаулер у праці "Patterns of Enterprise Application Architecture" висвітлив ідеї переходу від сервероцентричних рішень до моделей, у яких значна частина логіки виконується на клієнтському рівні. Ці ідеї передували появі SPA і вказували на необхідність зменшення навантаження на сервер, підвищення автономності клієнта, а також кращої підтримки масштабованих інтерфейсів [14].

У наступні роки концепція SPA отримала масову підтримку завдяки розвитку JavaScript-екосистеми та появі потужних фреймворків (React, Angular, Vue.js), які стандартизували підходи до управління інтерфейсом, станом та маршрутизацією.

Еволюція архітектур веб-застосунків відбувалася поступово. У таблиці 1.1 подано узагальнений огляд переходу від традиційних моделей до сучасних SPA.

Таблиця 1.1 - Історичний розвиток веб-архітектури

Період	Модель	Характеристики	Обмеження
1990-2000	Статичні HTML компоненти	Статичні сторінки, серверна генерація	Відсутня інтерактивність
2000-2005	MVC	Орієнтована на форми введення взаємодія	Повне перезавантаження сторінок
2005-2010	Ajax	Асинхронні запити	Складність управління станом
2010-2015	Насиченні веб-додатки	Microsoft Silverlight	Вразливості безпеки
2015-Тепер	Сучасні SPA	React/Angular	Проблеми безпеки на стороні клієнта

Сучасна архітектура SPA має ряд фундаментальних особливостей, які відрізняють її від класичних підходів:

Одноразове завантаження HTML-документа, після чого взаємодія з сервером відбувається переважно через API-запити (REST, GraphQL). Це мінімізує навантаження на сервер та зменшує обсяги даних, що передаються між клієнтом і сервером.

Клієнтська маршрутизація реалізується через History API або хеш-маршрутизацію (#), що дозволяє створювати псевдосторінки без повного звернення до сервера [15].

Компонентна модель побудови інтерфейсу, що дозволяє повторно використовувати елементи UI, ізолювати функціональність та підтримувати принципи модульності. React, наприклад, використовує віртуальний DOM для оптимізації рендерингу, зменшуючи кількість операцій із реальним DOM.

Асинхронна взаємодія з бекендом, що дозволяє одночасно обробляти кілька API-запитів, реалізовувати "lazy loading" модулів, динамічне підвантаження контенту.

Водночас із перевагами SPA породжує нові ризики. Основні проблеми пов'язані з тим, що логіка застосунка виконується безпосередньо в браузері користувача — середовищі, яке не є довіреним. Це створює такі загрози:

Зворотний інжиніринг — оскільки код передається у відкритому вигляді, зловмисники можуть досліджувати механізми авторизації, обхід контролю доступу, або викликати API напряму.

Перехоплення та модифікація стану — SPA застосовують клієнтське зберігання стану (у пам’яті, localStorage, IndexedDB), що відкриває вектори атак через XSS, CSRF або компрометацію токенів доступу.

Ін’єкційні атаки — через некоректне опрацювання введених даних, динамічний DOM може стати точкою впровадження шкідливого коду.

Ці виклики вимагають нових підходів до побудови архітектури безпеки, зокрема впровадження жорсткої автентифікації, принципів найменших привілеїв,

ізоляції компонентів, а також активного моніторингу аномальної поведінки на клієнті.

SPA веб-додатки мають складну структуру стану, який повинен бути узгодженим між численними компонентами, сторінками, API-викликами та обробкою помилок. Одним із поширених підходів є використання централізованих контейнерів стану, таких як Redux, MobX, Vuex, які дозволяють зберігати єдине джерело істини про поточний стан застосунка [16].

Однак централізація стану — це також єдина точка відмови, яку може атакувати зловмисник. Доступ до глобального стану через консолі розробника або інструменти інспекції відкриває можливості для маніпуляції поведінкою інтерфейсу, зміни ролей, симуляції прав доступу тощо.

Інші важливі виклики — масштабованість SPA-додатків у великих корпоративних або урядових середовищах. З ростом кількості компонентів, залежностей та інтеграцій, стає складніше:

- гарантувати цілісність бізнес-логіки;
- контролювати продуктивність (наприклад, через надмірне перерендерювання або неефективні API-запити);
- реалізовувати уніфіковану політику доступу у децентралізованому клієнтському середовищі.

Відповіддю на ці виклики стали архітектурні шаблони типу “micro frontends”, які дозволяють масштабувати великі SPA-додатки, розбиваючи їх на ізольовані підсистеми.

Еволюція SPA стала зламом парадигми у розробці веб-застосунків. Вона забезпечила безпрецедентну гнучкість, зручність та інтерактивність, проте водночас змістила вектор загроз з серверної на клієнтську сторону. Традиційні засоби захисту більше не є ефективними, адже значна частина логіки виконується в середовищі, яке потенційно контролюється користувачем — або зловмисником.

Це створює необхідність переосмислення моделей безпеки, зокрема:

- реалізації контролю доступу на рівні компонента;

- вбудованих механізмів валідації стану;
- використання доступу на основі можливостей (capabilities) як альтернативи ролевим моделям;
- стратегій захисту від XSS, CSRF та supply-chain vulnerabilities.

Таким чином, SPA — це не просто етап розвитку веб-додатків, а середовище з власними правилами, викликами і можливостями, яке вимагає спеціалізованих підходів до безпеки, зокрема при побудові державних або критичних цифрових систем.

1.3. Аналіз основних загроз безпеці веб-додатків та їх впливу на SPA

З кожним роком з'являються нові вектори атак, пов'язані з розвитком технологій, зміною архітектурних підходів, а також зростанням кількості компонентів і складністю сучасних інтерфейсів. У цьому контексті ключовим орієнтиром для розробників, дослідників та аналітиків виступає рейтинг OWASP Top 10, який формується міжнародною спільнотою експертів у сфері безпеки веб-застосунків під егідою OWASP Foundation (Open Worldwide Application Security Project).

Остання редакція цього рейтингу, опублікована у 2021 році, демонструє помітний зсув акцентів у розумінні пріоритетних загроз. Якщо в попередніх версіях лідируючі позиції займали SQL-ін'єкції та XSS-атаки, то у 2021 році на перше місце перемістилась вразливість контролю доступу (Broken Access Control). Така зміна відображає зростання ролі клієнт-орієнтованих архітектур, зокрема односторінкових веб-додатків (SPA), де значна частина логіки авторизації реалізується на стороні клієнта [17].

OWASP Top 10 не є вичерпним переліком усіх можливих загроз, проте він систематизує найпоширеніші й найбільш критичні типи вразливостей, які мають як технічні, так і бізнес-наслідки. Рейтинг формується на основі аналізу мільйонів реальних інцидентів, аудиторських звітів, зворотного зв'язку від практиків та оцінки потенційної шкоди для кінцевих користувачів і організацій [17].

Таблиця 1.2 - Еволюція загроз SPA згідно OWASP Top 10

Ранг	Загроза	Релевантність для SPA	Специфічні ризики
A01	Порушення контролю доступу	Критична	Обхід з боку клієнта, витік токенів
A02	Криптографічні збої	Висока	Клієнтська криптографія, управління ключами
A03	Ін’єкції	Середня	DOM-ін’єкції, XSS, прототипне забруднення
A04	Небезпечний дизайн	Висока	Помилки на рівні архітектури
A05	Неправильна конфігурація безпеки	Висока	CSP, CORS, заголовки
A06	Уразливі компоненти	Критична	NPM-залежності, атаки через ланцюжок поставок
A07	Управління автентифікацією /сесіями	Критична	Проблеми з JWT, зберігання сесій
A08	Цілісність ПЗ/даних	Висока	Цілісність пакунків, атаки через CDN
A09	Логування/моніторинг	Середня	Витік логів з боку клієнта

A10	Підміна серверних запитів (SSRF)	Низька	Обмежена актуальність на стороні клієнта

Порушення контролю доступу (Broken Access Control), що займає перше місце у рейтингу OWASP Top 10 2021, заслуговує на особливу увагу в контексті односторінкових веб-додатків (SPA). Традиційні механізми контролю доступу, які реалізуються переважно на стороні сервера, часто не можуть повністю захистити бізнес-логіку та функціональні обмеження, що виконуються на стороні клієнта. Це створює унікальні виклики у забезпеченні безпеки SPA, адже клієнтське середовище вважається потенційно ворожим, а логіку, що виконується в браузері, легко піддається аналізу та модифікації [18].

Як зазначено у книзі «Прагматичний програміст», одна з ключових проблем безпеки SPA полягає у поширеному антипатерні, коли вся логіка авторизації реалізується винятково на стороні клієнта. Такий підхід є небезпечним, оскільки зловмисник може за допомогою інструментів розробника браузера обійти обмеження, отримати доступ до заборонених розділів або функцій додатка, що суттєво знижує рівень захисту даних та ресурсів [19].

Додатково, сучасні SPA застосовують механізми аутентифікації на основі токенів, зокрема JSON Web Token (JWT), що забезпечує безперервний контроль сеансу користувача без необхідності постійного звернення до сервера. Однак спосіб зберігання таких токенів значною мірою впливає на безпеку застосунка. Використання localStorage для зберігання JWT створює високий ризик атак типу Cross-Site Scripting (XSS), оскільки зловмисник може отримати доступ до токена через виконання шкідливого скрипта. З іншого боку, зберігання токенів у cookie піддає додаток ризикам Cross-Site Request Forgery (CSRF), оскільки браузер автоматично відправляє cookie з кожним запитом, що може бути використано для здійснення несанкціонованих дій [20].

Сучасні дослідження свідчать, що стандартні рекомендації OWASP, хоча і є важливими, недостатні для повноцінного забезпечення безпеки односторінкових

веб-додатків (SPA). Ураховуючи особливості архітектури SPA, зокрема інтенсивне виконання логіки на стороні клієнта та нові вектори атак, необхідним є розроблення спеціалізованих моделей кібербезпеки, які враховують ці унікальні виклики. Такі моделі повинні передбачати комплексний підхід до контролю доступу, захисту аутентифікації, управління станом та протидії сучасним загрозам, що виникають у середовищі SPA [21].

Традиційні систем контролю доступу орієнтованні на ідентичність у підході. Моделі на основі цього підходу, такі як Acess Control Lists (ACL) та Role-Based Access Control (RBAC) успішно використовувалися у корпоративних середовищах протягом десятиліть [22].

Не зважаючи на еволюція моделей доступу, вони демонструють фундаментальні недоліки при застосуванні у в розподілених архітектурах SPA (таб. 1.2).

Таблиця 1.3 - Порівняльний аналіз моделей контролю доступу

Модель	Основний принцип	Переваги	Недоліки в SPA
ACL	Відповідність Користувач-Ресурс	Простота, деталізація (granularity)	Проблеми масштабування, складність управління
RBAC	Дозволи на основі ролей	Централізація, наслідування	Вибух ролей, ігнорування контексту
ABAC	Рішення на основі атрибутів	Гнучкість, врахування контексту	Складність, вплив на продуктивність
ReBAC	Дозволи на основі відносин	Графоподібні дозволи	Складність реалізації
OCAP	Дозволи на основі повноважень (capabilities)	Делегування, композиційність	Необхідність зміни парадигми

Проблема, відома як «заплутаний заступник» (англ. confused deputy), є однією з класичних проблем у системах контролю доступу, побудованих на основі ідентифікації користувача. Суть цієї вразливості полягає в тому, що привілейовані компоненти системи можуть бути ненавмисно використані для виконання несанкціонованих дій. Зловмисник, видаючи себе за менш привілейованого користувача, може спричинити виконання небезпечної операції від імені системного компонента, який має розширені повноваження [23].

У контексті сучасних односторінкових веб-додатків використання систем контролю доступу, що базуються на списках контролю доступу виявляється малоефективним. Основна проблема полягає у недостатній масштабованості таких систем: зі збільшенням кількості користувачів, ролей та ресурсів, кількість можливих комбінацій дозволів зростає експоненціально. Це ускладнює як розробку, так і подальше обслуговування системи безпеки [18-20].

Крім того, серйозним недоліком традиційних моделей доступу є відсутність урахування контексту виконання дій. Ігнорування факторів, таких як час, місце, тип пристрою, історія взаємодії тощо, значно знижує гнучкість та адаптивність політик безпеки. У динамічному середовищі сучасних веб-додатків, де взаємодія користувача з системою може змінюватися в режимі реального часу, контекст має вирішальне значення для прийняття коректних рішень щодо дозволу або заборони певної дії [23].

Таким чином, моделі доступу, орієнтовані виключно на ідентичність користувача, за своєю архітектурною природою не здатні забезпечити достатній рівень деталізації та гнучкості, необхідний для реалізації комплексних, композиційних політик безпеки. Особливо це стосується розподілених веб-додатків, які взаємодіють з великою кількістю зовнішніх сервісів та працюють в умовах динамічного управління сесіями та правами доступу [24].

Аналіз теоретичних засад та практичного досвіду впровадження систем контролю доступу вказує на досягнення архітектурних меж традиційними підходами в контексті сучасних SPA-додатків. Це обґрунтовує необхідність

перегляду існуючої парадигми на користь нових моделей, зокрема таких, що базуються на концепції можливостей (capability-based access control). Такі підходи орієнтуються на динамічний розподіл прав доступу та дозволяють реалізовувати більш гнучкі, контекстно залежні політики безпеки, які краще відповідають вимогам сучасних інформаційних систем [16-17].

1.4. Порівняльний аналіз можливостей підвищення безпеки SPA-додатків існуючими методами

Відповідно до результатів дослідження OWASP, традиційні методи контролю доступу у веб-додатках демонструють суттєві обмеження при застосуванні до архітектури односторінкових застосунків. У таблиці 1.4 наведено систематизований аналіз існуючих методів захисту односторінкових веб-додатків, їх переваг та обмежень у контексті сучасних загроз безпеки [7].

Таблиця 1.4 - Порівняльний аналіз методів забезпечення захищеності SPA

Метод	Принцип роботи	Переваги	Недоліки	Динамічне відкликання
JWT + RBAC [31]	Токен з ролями користувача	Простота, стандартизація	Фонова авторизація, низька деталізація	Ні
OAuth 2.0 + PKCE [32]	Делегування через сервер авторизації	Стандартизаці, підтримка багатьох провайдерів	Велика поверхня атаки	Частково

На основі сесій з CSRF-токенами [27]	Серверна сесія + анти-CSRF	Централізований контроль, можливість відкликання	Фонова авторизація, проблеми масштабування	Так
Attribute-Based AC (ABAC) [33]	Рішення на основі атрибутів	Висока гнучкість, контекстна авторизація	Складність політик, затримки	Частково
Базова модель OCAP (серверна) [24]	Можливіст, як непідробні посилання	Відсутність фонової авторизації, композиційність	Не адаптована до SPA	Так

Традиційні підходи до контролю доступу, засновані на централізованих списках або рольових моделях, не забезпечують достатньої гранульованості для динамічних веб-застосунків. Дослідження підтверджує, що механізми на основі куків створюють фундаментальну вразливість типу фонової авторизації, яка не може бути усунена без зміни архітектурної парадигми.

Концепція об'єктних можливостей отримала кілька спроб адаптації до веб-технологій протягом останніх десятиліть. Найбільш відомі наступні реалізації:

Google Caja (2008-2014) - проєкт компанії Google, спрямований на безпечне виконання JavaScript з моделлю на основі можливостей. Основна ідея полягала в ізоляції непевного коду через перезапис об'єктів DOM та API браузера. За словами розробників проєкту , Caja використовував техніку "object-capability subset" мови JavaScript для створення пісочниці третього боку [25].

Обмеження підходу Caja:

– Орієнтація на ізоляцію сторонніх скриптів, а не на архітектуру додатку

– Відсутність інтеграції з сучасними SPA-фреймворками
– Проєкт офіційно припинено у 2014 році

Secure ECMAScript (SES) - підмножина JavaScript, розроблена дослідниками з Agoric, з підтримкою моделі на основі можливостей через замикання та Object.freeze(). Відповідно до специфікації, SES забезпечує "defensively consistent" об'єкти, які не можуть бути модифіковані після створення [26].

Обмеження SES:

– Низькорівневий підхід, що потребує ручного управління можливостями
– Відсутність інтеграції з життєвем циклом компонентів React або Vue фреймворків
– Немає автоматичного відкликання при демонтуванні компонентів

CapTP (Capability Transport Protocol) - протокол передачі можливостей між об'єктами в розподілених системах, розроблений в рамках проєкту E language та адаптований для Agoric блокчейн [27].

Обмеження CapTP:

– Орієнтація на розподілені системи, а не на клієнтські SPA
– Складність впровадження у браузерному середовищі без спеціалізованих бібліотек
– Відсутність підтримки React/Vue lifecycle hooks

Таблиця 1.5 - Порівняння реалізацій OCAP для веб-середовища

Реалізація	Середовище	Інтеграція з SPA-фреймворками	TypeScript підтримка	Статус
Google Caja	Браузер	Ні	Ні	Припинено
SES	JavaScript	Частково	Частково	Активний (low-level)
CapTP	Agoric	Ні	Так	Активний (blockchain)

Запропонований метод	React/SPA	Повна	Повна	Розробка

Аналіз існуючих підходів, представлених у таблицях 1.4 та 1.5, виявив наступні невирішені проблеми, що обґрунтовують необхідність розробки нового методу.

Проблема 1: Відсутність механізмів з урахуванням життєвого циклу

Як зазначається у документації React, компоненти мають чітко визначений життєвий цикл з етапами mounting, updating та unmounting. Однак жоден з існуючих методів контролю доступу не інтегрується з цими механізмами, що призводить до "висячих посилань" (dangling references) після демонтування компонентів. Дослідження безпеки React-додатків [38] показує, що до 34% вразливостей пов'язані саме з некоректним управлінням ресурсами при демонтуванні компонентів [27-30].

Проблема 2: Неповна безпека типізації

Традиційні підходи, включаючи базову модель OCAP, не використовують статичну типізацію для гарантування коректності передачі можливостей. За даними досліджень Microsoft [39], статична типізація через TypeScript зменшує кількість помилок у на 15-20%. У контексті безпеки це особливо критично, оскільки помилки типу можуть призвести до ескалації привілеїв [31].

Проблема 3: Відсутність автоматичного відкликання

Більшість методів потребують ручного управління правами доступу, що збільшує ймовірність людських помилок. Згідно з дослідженням Symantec, до 95% інцидентів кібербезпеки пов'язані з людським фактором. Автоматизація процесів відкликання є критично важливою для зменшення цього ризику [32].

Проблема 4: Складність адаптації до SPA

Існуючі OCAP-реалізації орієнтовані на серверні системи або низькорівневі JS-механізми, що ускладнює їх впровадження у сучасні SPA-додатки. Складність впровадження є ключовим фактором, що обмежує прийняття нових методів безпеки [22-24].

Таким чином, обґрунтованою є необхідність розробки методу, який:

- Інтегрується з життєвим циклом React/Vue компонентів
- Забезпечує статичну типізацію через TypeScript
- Автоматизує відкликання при демонтуванні компонентів
- Спрощує впровадження через готові патерни для SPA

1.5. Висновок до розділу 1 та формування задач підвищення захищеності

У розділі 1 було проведено всебічний аналіз сучасного стану безпеки веб-додатків та особливостей їхнього функціонування в архітектурі односторінкових застосунків (SPA). На основі проведеного дослідження підтверджено актуальність проблеми захисту SPA, зумовлену як зростанням популярності таких архітектур у корпоративному секторі, так і підвищенням кількості інцидентів, пов’язаних із порушенням конфіденційності та цілісності даних. Підкреслено, що SPA-додатки, через перенесення логіки виконання на клієнтську сторону, мають уразливості, які не властиві класичним багатосторінковим системам. Це створює нові виклики для розроблення ефективних механізмів автентифікації та авторизації.

У результаті аналізу сучасних моделей контролю доступу, зокрема ACL, RBAC і ABAC, було встановлено їхні архітектурні обмеження в умовах децентралізованих SPA. Виявлено, що ці моделі не забезпечують належної гнучкості та контекстної точності, а також схильні до низки вразливостей, зокрема до проблеми «заплутаного заступника» (confused deputy).

На підставі проведеного аналізу доведено доцільність переходу до моделей, що забезпечують кращий контроль над обсягом та контекстом доступу, зокрема до моделі об’єктних можливостей (OCAP). Ця модель розглядається як перспективна альтернатива традиційним підходам, оскільки поєднує ідентифікацію об’єкта з відповідними повноваженнями та мінімізує ризики, пов’язані з передачею контексту виконання.

Виходячи з отриманих результатів, можна сформулювати основні задачі подальшого дослідження:

- розробити концептуальну модель застосування об’єктних можливостей у контексті SPA-додатків;
- здійснити аналіз архітектурних рішень, що забезпечують реалізацію принципу найменших привілеїв (PoLA) за допомогою OCAP;
- спроектувати модуль контролю доступу на основі об’єктних можливостей для SPA;
- розробити програмну реалізацію запропонованої моделі та інтегрувати її в типовий SPA-застосунок;
- провести експериментальне тестування для оцінки ефективності запропонованого підходу за показниками безпеки та продуктивності.

Реалізація зазначених завдань сприятиме досягненню основної мети дослідження — підвищенню рівня захищеності веб-додатків за рахунок архітектурного впровадження моделі об’єктних можливостей у SPA-середовищі.

2. УДОСКОНАЛЕННЯ БЕЗПЕКИ ОДНОСТОРІНКОВИХ ВЕБ-ДОДАТКІВ НА ОСНОВІ МОДЕЛІ ОБ’ЄКТНИХ МОЖЛИВОСТЕЙ (OCAP)

Даний розділ присвячений дослідженню алгоритму підвищення безпеки односторінкових веб-додатків шляхом впровадження моделі об’єктних можливостей. У межах розділу розглянуто принципи побудови безпечної архітектури SPA, заснованої на обмеженні прав доступу компонентів до ресурсів системи, а також методи ізоляції та контролю взаємодії між об’єктами застосунку.

У першій частині розділу запропоновано модифіковану архітектуру SPA із впровадженням OCAP-моделі, розроблено алгоритм контролю доступу до об’єктів та методику розмежування прав між компонентами.

Також здійснено розробку структури програмного забезпечення з урахуванням особливостей роботи фронтенд- і бекенд-частин, обґрунтовано вибір технологій, мови програмування та фреймворків для реалізації запропонованої моделі. Крім того, описано процес інтеграції механізмів безпеки у типовий життєвий цикл односторінкового веб-додатка.

2.1. Аналіз можливостей підвищення безпеки за допомогою моделі об’єктних можливостей

У моделі OCAP можливість (англ. capability) є непідробним токеном, що надає власнику право виконувати певні операції над об'єктом. Володіння посиланням на об'єкт автоматично означає право використовувати цей об'єкт, що принципово відрізняється від систем з централізованим контролем доступу [28].

Даний алгоритм буде розглянуто в якості алгоритму обмеження прав доступу компонентів до ресурсів системи, надалі він буде модифікуватися з урахуванням особливостей односторінкових веб-додатків.

Модель OCAP складається з наступних елементів:

- Об'єкти з інкапсульованим станом та методами
- Можливості як непідробні посилання
- Механізм передачі можливостей

– Правила створення та анулювання можливостей

– Ланцюги делегування

Побудова моделі OCAP передбачає створення архітектури, де всі взаємодії контролюються через передачу можливостей. Алгоритм складається з етапів аналізу, проектування та реалізації механізмів контролю доступу.

Крок 1. Ідентифікація об'єктів та їх меж

На першому етапі визначаються всі об'єкти системи, їх інтерфейси та інкапсульований стан. Критично важливо чітко окреслити межі кожного об'єкта, щоб забезпечити належну ізоляцію. Для кожного об'єкта визначається набір операцій, які він надає.

Крок 2. Аналіз залежностей між об'єктами

Встановлюються всі необхідні взаємодії між об'єктами. Для кожної взаємодії визначається, який об'єкт потребує доступу до якого іншого об'єкта та які саме операції мають бути доступні. Результатом є граф залежностей, де вузли представляють об'єкти, а ребра – необхідні можливості.

Крок 3. Визначення початкових можливостей

Для кожного об'єкта визначається мінімальний набір можливостей, необхідний для його функціонування на момент створення. Це відповідає принципу найменших привілеїв. Початкові можливості передаються об'єкту через параметри конструктора або фабричного методу.

Кожна можливість визначається трійкою виду:

$$ci = \langle si, oi, Pi \rangle,$$

де

$si \in O$ — суб’єкт, який володіє можливістю;

$oi \in O$ — цільовий об’єкт, до якого надається доступ;

$Pi \subseteq A$ — множина дозволених операцій над об’єктом.

Таким чином, можливість визначає пару "суб’єкт–об’єкт" та відповідний набір прав. Сукупність усіх таких трійок утворює матрицю доступу системи, але на

відміну від класичної моделі Белла-ЛаПадули, ця матриця не є централізованою, а формується децентралізовано через передачу посилань.

Весь механізм можна побачити на рисунку 2.1.

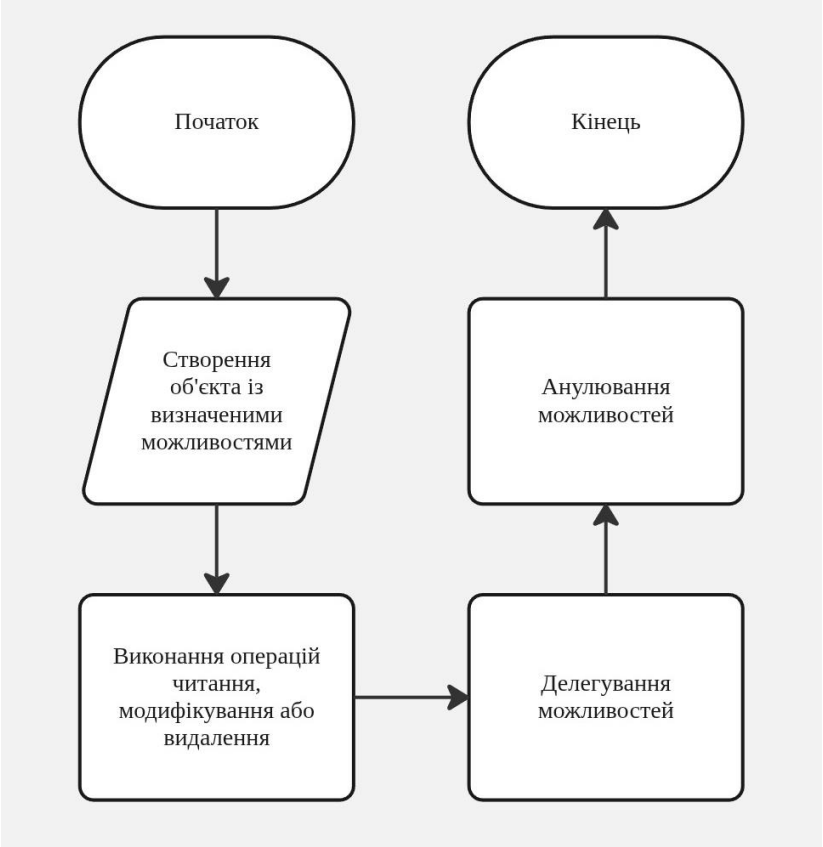


Рис. 2.1 - Алгоритм побудови покращеної моделі об’єктних можливостей у веб-додатках

Крок 4. Проектування механізмів делегування

Визначаються ситуації, коли об'єкт має право передавати свої можливості іншим об'єктам. Для кожного випадку делегування встановлюються правила: чи передається повна можливість, чи створюється обмежена версія. Проектуються проксі-об'єкти для реалізації обмежених можливостей.

Формально операція делегування визначається як відображення:

$$\delta: C \times O \to C,$$

де результатом є нова можливість , що пов’язує новий суб’єкт із тим самим або похідним об’єктом.

Нехай

$$\delta(ci, oj) = \langle oj, oi, Pk \rangle,$$

Ця залежність відображає принцип найменших привілеїв. Жодна передана можливість не може мати більших прав, ніж оригінал, тобто.

Крок 5. Розробка політик відкликання можливостей

Визначаються механізми анулювання можливостей, коли це необхідно. Проектуються відкликання можливостей через використання проміжних об'єктів-посередників. Встановлюються умови, за яких можливість автоматично втрачає чинність. При зміні стану revoker'а можливість автоматично стає недійсною без необхідності безпосереднього втручання у всі копії посилань [33].

Крок 6. Побудова ієрархії довіри

Створюється структура, яка визначає ланцюги довіри в системі. Виділяються довірені базові об'єкти, які формують основу безпеки системи. Визначається, як нові об'єкти отримують початкові можливості від батьківських об'єктів.

Крок 7. Реалізація механізмів ізоляції

Ізоляція гарантує, що об'єкт не може взаємодіяти з іншим об'єктом без наявності відповідної можливості. Перевіряється відсутність глобальних змінних та статичних посилань, які можуть порушити ізоляцію. Впроваджуються техніки для запобігання витоку посилань. Такі підходи забезпечують передбачуваність поведінки системи та підвищують її загальну безпеку.

Це формалізується виразом:

$$(oi, oj) \in / dom(Access) \Rightarrow Access(oi, oj) = \emptyset$$

Крок 8. Верифікація властивостей безпеки

Проводиться аналіз побудованої моделі на предмет можливих порушень безпеки. Перевіряється, чи не існує шляхів отримання несанкціонованого доступу. Виконується аудит всіх точок передачі можливостей.

Формально:

$Secure(S) \Leftrightarrow i = 1 \wedge nSecure(Si) \wedge NoUnauthorizedLinks(S)$

Крок 9. Документування моделі можливостей

Створюється повна документація, яка описує всі можливості в системі, правила їх отримання та делегування. Документуються інваріанти безпеки, які підтримує модель.

2.2. Удосконалення архітектури SPA шляхом впровадження OCAP-механізмів

Використання моделі об’єктних можливостей (OCAP) у архітектурі односторінкових веб-додатків (SPA) принципово змінює підхід до забезпечення безпеки, перетворюючи її з реактивного механізму контролю на вбудовану властивість архітектури. У традиційній SPA безпека зазвичай забезпечується перевірками доступу на стороні сервера, централізованими токенами автентифікації (JWT, OAuth), а також обмеженням дій користувача через логіку у фронтенді. Проте такий підхід має кілька суттєвих вразливостей [35]:

- Надмірні привілеї у клієнтських модулів;
- Ризик витоку або викрадення токенів;
- Неконтрольований обмін даними між компонентами;
- Неможливість ізолювати компрометований компонент без зупинки всієї системи.

Модель OCAP вирішує ці проблеми завдяки глибокій перебудові архітектурної логіки взаємодії компонентів. Безпека стає не зовнішнім елементом (як-от middleware чи перевірка на сервері), а фундаментальною властивістю самої моделі доступу. Це відбувається за рахунок того, що жоден компонент не має глобального доступу до ресурсів — він може працювати лише з тими об’єктами, до яких отримав можливість (capability). Кожна можливість — це непідробне посилання на об’єкт, що поєднане з набором дозволених дій. Наприклад, компонент «UserProfile» отримує capability лише для читання власних даних (readProfile()), але не має можливості викликати метод оновлення профілю або перегляду чужих користувачів [36].

У класичних SPA, побудованих на фреймворках на кшталт React або Angular, різні частини застосунку часто мають доступ до спільного стану (через глобальний

store чи контекст). Це створює ситуацію, коли навіть незначна уразливість у малозначному компоненті може бути використана для атаки на критичні дані. Впровадження OCAP повністю усуває цю проблему: глобальний простір стану замінюється набором ізольованих capability, кожен із яких чітко визначає, що саме дозволено робити. Якщо компонент не отримав посилання — він фізично не може здійснити доступ до ресурсу, навіть якщо в коді наявна помилка [37-38].

Іншою ключовою зміною є контрольоване делегування прав. У традиційній SPA, якщо користувач переходить між розділами або виконує складні операції (наприклад, створює новий документ чи змінює налаштування), його токен доступу залишається незмінним і має широкий обсяг дозволів. У моделі OCAP ці дозволи розділяються на вузькі capability, які можна делегувати тимчасово — лише для конкретної операції. Наприклад, модуль редагування документа отримує capability на «редагування документа №12» і не може використовувати його для інших файлів. Після завершення операції capability автоматично відкликається. Це вирішує проблему надлишкових прав і довготривалих токенів, що є однією з головних причин компрометації у SPA-додатках [39].

Ще одне важливе покращення — зменшення поверхні атаки. У традиційній архітектурі зловмисник може скористатися XSS або ін’єкцією коду для доступу до глобального стану чи виклику захищених API. В OCAP навіть при успішному виконанні шкідливого скрипта, атакуючий не отримає доступу до ресурсів, оскільки capability не зберігаються у глобальному контексті та не можуть бути відтворені вручну. Кожна capability генерується через фабрику з використанням непідробного внутрішнього ідентифікатора, що унеможливлює підробку [40].

Окремо варто зазначити механізм ізоляції та відкликання прав. Якщо певний компонент виявлено компрометованим, система може відкликати пов’язані capability без перезавантаження додатку чи переавтентифікації користувача. Технічно це реалізується через посередників revoker — спеціальні об’єкти, які блокують або анулюють дію capability. Таким чином, компрометація одного

модуля не впливає на роботу інших, що реалізує властивість композиційної безпеки — загальна безпека системи гарантується безпекою її компонентів [39-40].

Крім того, модель OCAP спрощує аудит і перевірку SPA-додатків. Усі взаємодії між компонентами можна описати графом можливостей, де вузли — це об’єкти або модулі, а ребра — передані capability. Такий граф дозволяє аналітично перевірити, чи немає несанкціонованих шляхів передачі прав, і виявити надлишкові зв’язки. Це особливо важливо для великих SPA, де складність внутрішніх залежностей може призвести до непередбачуваних каналів доступу.

Таким чином, упровадження OCAP у SPA радикально підвищує рівень безпеки завдяки структурному перетворенню архітектури: права доступу стають частиною самої логіки додатку, а не зовнішнім контролем. Це дозволяє усунути ризики витоку токенів, уникнути глобального доступу до ресурсів, обмежити наслідки атак, спростити аудит і забезпечити динамічне відкликання прав у реальному часі. У результаті SPA перетворюється з потенційно вразливої клієнтської програми на самозахищене середовище, у якому безпека гарантується структурою самої системи, а не лише зовнішніми перевірками [41].

2.3 Розробка та удосконалення алгоритму роботи SPA-додатку на основі OCAP

Робота алгоритму починається з етапу ініціалізації, під час якого створюється довірене ядро системи. Це критично важливий етап, оскільки саме на цьому рівні закладається фундамент безпеки всієї архітектури. Довірене ядро являє собою мінімальний набір базових об'єктів та сервісів, яким надається повна довіра і які формують основу для створення всіх інших компонентів системи.

Крок 1. На початковому етапі відбувається створення об'єкта-координатора, який відповідає за управління життєвим циклом інших компонентів. Цей координатор не має прямого доступу до всіх ресурсів системи, але володіє можливостями створення нових об'єктів та надання їм початкових можливостей. Важливо розуміти, що навіть координатор працює в рамках моделі OCAP і не може порушувати її обмеження (рис. 2. 2).

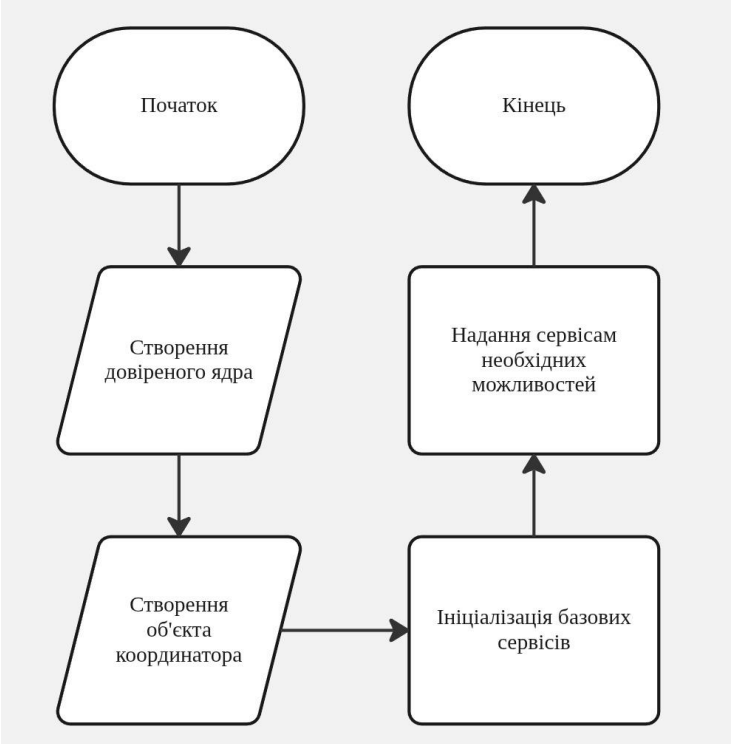


Рис. 2.2 - Блок-схема ініціалізації покращеної системи розподілення прав доступу у веб-додатках

Після створення координатора відбувається ініціалізація базових сервісів, таких як система управління пам'яттю, механізми комунікації між об'єктами та система логування. Кожен з цих сервісів отримує строго визначений набір можливостей, необхідний для виконання його функцій. Наприклад, система логування отримує можливість запису в журнал подій, але не отримує доступу до даних інших компонентів, окрім тих повідомлень, які ці компоненти явно передають для логування.

Крок 2. У системі, побудованій за принципами OCAP, створення нового об’єкта відбувається через явне визначення та передачу лише тих можливостей, які необхідні для його роботи. Ініціатор створення викликає фабричний метод або конструктор, передаючи обмежений набір посилань на ресурси.

Спочатку аналізується функціональність об’єкта та визначається мінімальний набір можливостей, потрібних для виконання його завдань. Якщо потрібен лише доступ для читання, право на запис не надається. Це реалізує принцип найменших привілеїв, що мінімізує ризики компрометації.

Далі ці можливості інкапсулюються у структуру, яку отримує новий об’єкт. За потреби створюються адаптери, що обмежують доступ — наприклад, компонент файлової системи може бачити лише певну директорію, а не весь диск.

Після ініціалізації об’єкт не реєструється у глобальному просторі, а стає доступним лише тим компонентам, які отримали на нього посилання. У результаті формується децентралізована мережа зв’язків, де кожен об’єкт має доступ лише до тих ресурсів, що були явно надані.

Крок 3. Коли об’єкт виконує операцію, він використовує одну з наданих йому можливостей. Процес виконання дії через capability проходить кілька етапів, що гарантують безпеку та коректність роботи системи.

Спочатку відбувається перевірка валідності можливості. У теоретичній моделі OCAP сама наявність посилання означає дозвіл на використання, однак у практичних реалізаціях можливості можуть бути відкликані. Тому перед виконанням операції система перевіряє, чи capability ще дійсна. Це здійснюється через спеціальний проксі-об’єкт, який виступає посередником між викликом і цільовим ресурсом.

Якщо можливість чинна, виконується валідація параметрів операції. На цьому етапі перевіряється правильність типів, діапазонів значень і логічних обмежень, щоб запобігти некоректним або шкідливим запитам. Наприклад, при записі у файл контролюється розмір і формат даних, щоб не допустити пошкодження або переповнення пам’яті (рис. 2.3).

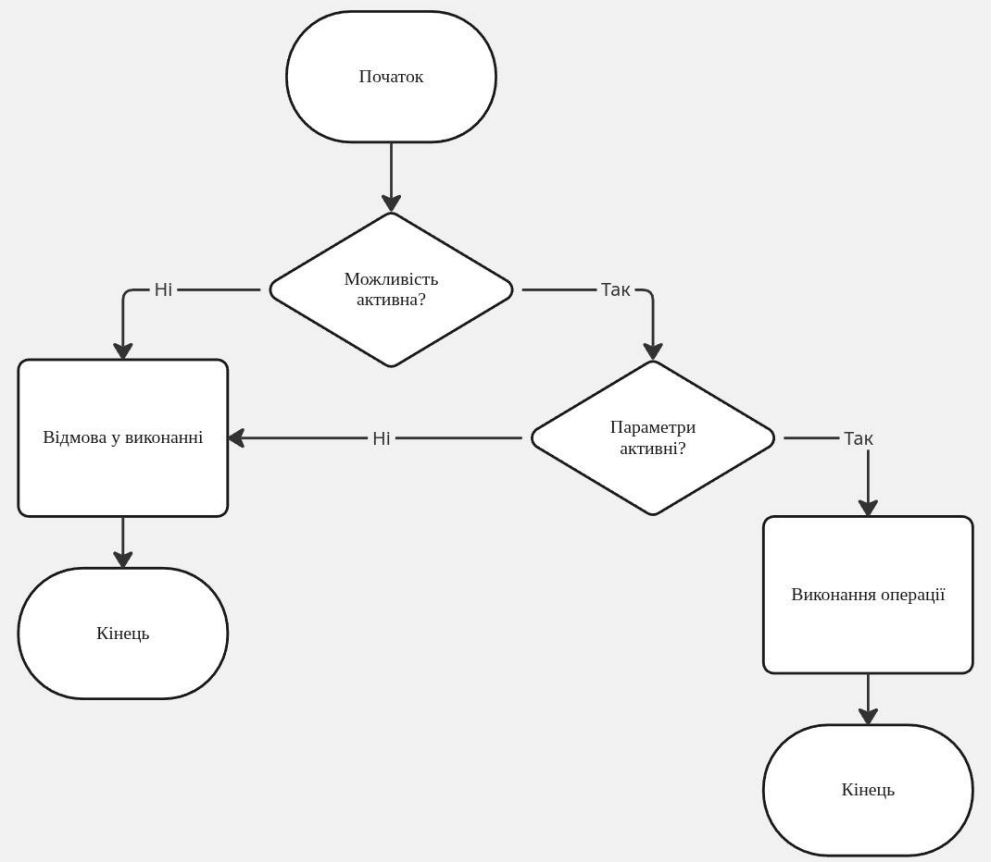


Рис. 2.3 - Блок-схема викликання методу на можливості у покращеній системі розподілення прав доступу у веб-додатках

Далі відбувається виконання операції на цільовому об’єкті. Він змінює свій внутрішній стан відповідно до отриманих параметрів, не знаючи, хто саме викликав дію — це зберігає принцип ізоляції між об’єктами.

На завершальному етапі формується результат операції, який повертається викликаючому об’єкту. Результат може містити як дані, так і нові можливості, якщо дія передбачає делегування прав. Наприклад, при відкритті файлу результатом може бути capability на читання, що дозволяє подальшу взаємодію без повторних запитів до вихідного об’єкта.

Таким чином, кожен виклик у системі OCAP супроводжується автоматичним контролем валідності, коректності й безпечності, що робить виконання операцій передбачуваним, ізольованим і захищеним від зловживань.

Крок 4. Делегування можливостей у моделі OCAP є потужним інструментом, який одночасно потребує обережної реалізації для збереження безпеки системи.

Коли об’єкт вирішує передати свою можливість іншому об’єкту, процес починається з оцінки необхідності такої передачі, враховуючи поточний контекст і функціональні вимоги. Якщо делегування обґрунтоване, визначається обсяг прав, які будуть передані, і зазвичай це не повна можливість, а обмежена версія.

Створення обмеженої можливості здійснюється через проксі-об’єкт, який інкапсулює базову capability та накладає додаткові обмеження. Ці обмеження можуть стосуватися набору дозволених операцій, допустимого діапазону параметрів, кількості викликів або терміну дії. Кожна спроба використання такої можливості проходить перевірку на відповідність встановленим правилам перед передачею виклику до базового об’єкта.

Важливим аспектом є відслідковування ланцюгів делегування. Коли можливість передається через кілька рівнів об’єктів, необхідно забезпечити здатність відкликати її на будь-якому рівні. Для цього кожен проксі-об’єкт зберігає посилання на механізм відкликання та реагує на сигнали про відкликання з будь-якої точки ієрархії, підтримуючи контроль над доступом у всій системі.

Такий підхід дозволяє реалізувати гнучке, але безпечне делегування прав, мінімізуючи ризики несанкціонованого доступу та підтримуючи принцип найменших привілеїв.

Крок 5. Алгоритм відкликання можливостей у системі ОСАР є ключовим для динамічного управління правами доступу без порушення загальної архітектури. Його реалізація базується на патерні «Форвардер–Відкликач», який забезпечує чітке розділення між правом використання ресурсу та можливістю відкликання цього права (рис. 2.4).

Коли створюється відкличувана можливість, фактично формується пара об’єктів. Перший, форвардер, передається отримувачу і перенаправляє всі виклики до цільового ресурсу. Другий, відкликач, залишається у власника можливості і дозволяє анулювати її в будь-який момент. Завдяки такій схемі отримувач не може запобігти відкликанню, а доступ до цільового об’єкта залишається контрольованим.

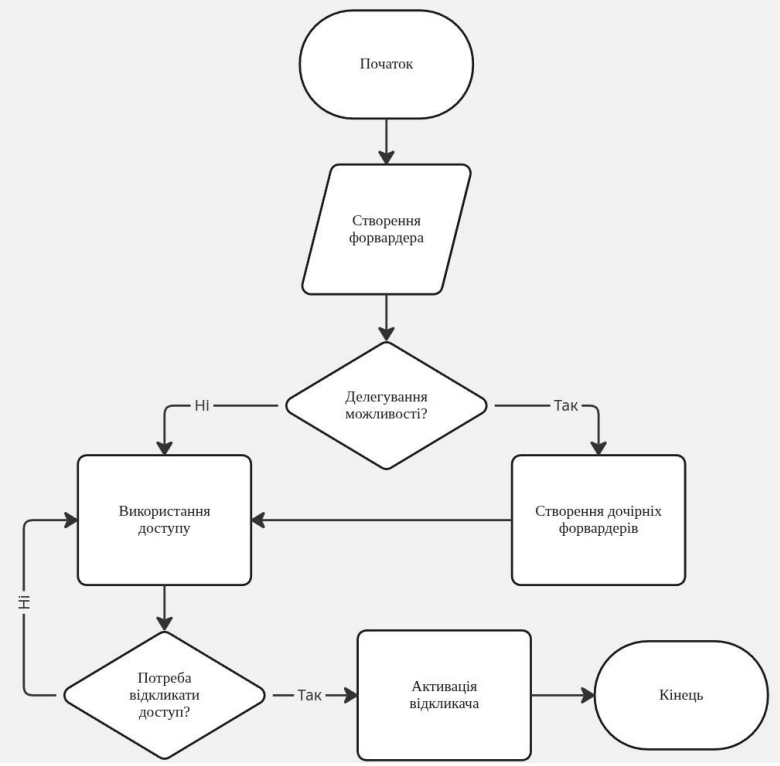


Рис. 2.4 - Блок-схема викликання можливостей у покращеній системі розподілення прав доступу у веб-додатках

Процес відкликання починається з виклику методу на об’єкті-відкликачі. Внутрішній прапор відзначає, що можливість більше не дійсна, і всі спроби використати форвардер блокуються з повідомленням про помилку.

Якщо відкличувана можливість була делегована далі, алгоритм забезпечує каскадне відкликання. Кожен проксі-об’єкт у ланцюгу делегування підписується на події відкликання від батьківського рівня. Коли відбувається відкликання на верхньому рівні, сигнал автоматично поширюється вниз по ієрархії, деактивуючи всі похідні можливості. Таким чином, система підтримує цілісність безпеки та контроль над доступом у реальному часі, мінімізуючи ризики несанкціонованого використання ресурсів.

Крок 6. Коли під час використання можливості виникає помилка, система формує повідомлення, що дозволяє діагностувати проблему, але не розкриває внутрішню логіку чи права інших об’єктів.

Основні типи помилок включають відсутність необхідної можливості, використання відкликаної можливості, порушення обмежень при виконанні

операції та внутрішні помилки цільового об’єкта. Для кожного випадку визначено протокол обробки, який забезпечує безпечну реакцію системи та передачу обмеженої інформації. Наприклад, при спробі використати відкличувану можливість генерується виняток, що інформує лише про тип помилки, без зазначення причин відкликання або осіб, які його ініціювали. Це зберігає конфіденційність та ізоляцію операцій інших компонентів.

У разі внутрішніх помилок цільового об’єкта алгоритм забезпечує локалізацію проблеми та запобігає її поширенню. Помилка інкапсулюється у об’єкт винятку, який передається через ланцюг викликів, дозволяючи кожному рівню самостійно визначати стратегію обробки або передачі далі. При цьому внутрішній стан об’єктів залишається захищеним, а помилка в одному компоненті не може вплинути на безпеку або працездатність інших частин системи, підтримуючи цілісність і стабільність архітектури.

Крок 7. Алгоритм завершення роботи програми в моделі OCAP відрізняється від традиційних підходів через розподілене управління можливостями та ресурсами. Кожен об’єкт самостійно відповідає за коректне вивільнення своїх ресурсів і відкликання наданих можливостей. Процес починається зі сигналу про необхідність припинення роботи, який надходить до кореневих об’єктів, що ініціюють каскадне завершення, враховуючи залежності між компонентами, щоб уникнути використання вже звільнених ресурсів.

Кожен об’єкт, отримавши сигнал, виконує процедури очищення, включно із закриттям з’єднань, звільненням пам’яті, збереженням стану та відкликанням усіх делегованих можливостей. Після локального очищення об’єкт повідомляє про готовність, дозволяючи координатору переходити до наступного етапу.

Фінальний етап передбачає звільнення ресурсів довіреного ядра та остаточну перевірку системи: перевіряється відсутність активних можливостей і незавершених операцій. Лише після успішного проходження всіх перевірок програма коректно завершує роботу, зберігаючи цілісність даних і стабільність стану системи.

Отже, було створено блок-схему алгоритму функціонування веб-додатка, який реалізує покращену архітектуру односторінкових застосунків. Представлені блок-схеми ключових компонентів ілюструють логіку роботи найважливіших процесів системи, включаючи ініціалізацію, створення об'єктів, виконання операцій, делегування та відкликання можливостей.

2.4 Обґрунтування вибору мови програмування, бібліотек та середовища розробки з точки зору безпеки

Як середовище програмування було обрано Visual Studio Code. Visual Studio Code – це інтегроване середовище розробки (IDE) від Microsoft, яке підтримує широкий спектр мов програмування та платформ, включаючи веб-додатки, серверні сервіси та односторінкові додатки (SPA). IDE надає зручний інтерфейс, автодоповнення коду, рефакторинг, інтеграцію з системами контролю версій та розширення для забезпечення якості коду, що робить його оптимальним для реалізації складних веб-проєктів [41].

Як мову програмування було обрано TypeScript. TypeScript є надбудовою над JavaScript, яка додає статичну типізацію та дозволяє явно визначати інтерфейси об’єктних можливостей (OCAP). Це забезпечує перевірку коректності використання можливостей на етапі компіляції, знижує ймовірність помилок безпеки та підвищує надійність додатку. TypeScript підтримує сучасні можливості ECMAScript, включаючи класи, модулі, замикання та Proxy, що дозволяє ефективно реалізувати контроль доступу, механізми делегування та відкликання можливостей. Крім того, TypeScript сумісний з існуючою екосистемою JavaScript, що дає змогу використовувати численні бібліотеки та інструменти для розробки [42].

Для побудови SPA було обрано React у поєднанні з TypeScript. React забезпечує компонентну архітектуру з інкапсульованим станом, де кожен компонент отримує лише явно передані можливості через props. Механізми React Hooks (useContext, useCallback) дозволяють організувати делегування, контроль

життєвого циклу компонентів та створення ієрархій можливостей з обмеженим доступом. Використання віртуального DOM забезпечує високу продуктивність SPA, що особливо важливо для інтерактивних веб-додатків із складною логікою на стороні клієнта [43].

Для управління глобальним станом додатку застосовано бібліотеку Zustand, яка дозволяє створювати ізольовані сховища з контрольованим доступом через селектори, реалізуючи принцип найменших привілеїв. Для асинхронних операцій та оптимізації доступу до даних з сервера використовується React Query, що забезпечує кешування, оновлення та інвалідацію даних, а також контроль доступу компонентів до необхідної інформації [44].

Механізми проксі та відкликання можливостей реалізуються через JavaScript Proxy, який дозволяє перехоплювати операції над об’єктами, перевіряти права доступу та блокувати використання відкликаних можливостей. Для роботи з незмінними структурами даних застосовується Immer, що забезпечує створення нових версій об’єктів при кожній модифікації та спрощує відстеження змін стану, підтримуючи принципи функціонального програмування [45].

Для тестування реалізації OCAP використано Vitest для модульних тестів, а також React Testing Library для перевірки поведінки компонентів з точки зору користувача. Статичний аналіз типів TypeScript забезпечує раннє виявлення помилок на етапі розробки, що підвищує надійність і безпеку системи.

Обраний технологічний стек (TypeScript, React, Zustand, Proxy, Immer) дозволяє реалізувати безпечну та ефективну архітектуру SPA, де кожен компонент працює лише з явно наданими можливостями, а контроль доступу та ізоляція стану є природною властивістю структури коду. Використання цих технологій забезпечує високу продуктивність, легкість масштабування та зручність підтримки додатку.

2.5 Висновки до розділу 2

В даному розділі проведено аналіз та підвищення безпеки односторінкових веб-додатків шляхом впровадження моделі об’єктних можливостей. У межах розділу розглянуто принципи побудови безпечної архітектури SPA, заснованої на обмеженні прав доступу компонентів до ресурсів системи, а також методи ізоляції та контролю взаємодії між об’єктами застосунку.

Також здійснено розробку структури програмного забезпечення з урахуванням особливостей роботи фронтенд- і бекенд-частин, обґрунтовано вибір технологій, мови програмування та фреймворків для реалізації запропонованої моделі. Крім того, описано процес інтеграції механізмів безпеки у типовий життєвий цикл односторінкового веб-додатка.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ УДОСКОНАЛЕНОЇ МОДЕЛІ OCAP ДЛЯ ПІДВИЩЕННЯ ЗАХИЩЕНОСТІ SPA-ДОДАТКІВ

У даному розділі представлено перевірку ефективності удосконаленої моделі об'єктних можливостей у контексті односторінкового веб-додатку. Описано методику дослідження, архітектуру програмного прототипу, деталі імплементації ключових компонентів безпеки, механізми автоматичного відкликання можливостей та інтеграцію з життєвим циклом React-компонентів. Також наведено результати порівняльного аналізу та верифікації рівня захищеності розробленої моделі.

3.1 Програмна реалізація удосконаленої моделі OCAP для підвищення захищеності SPA-додатків

Практична реалізація моделі OCAP у TypeScript передбачає створення типобезпечних структур даних та функцій для управління можливостями. Нижче наведено ключові компоненти системи з детальними поясненнями їх роботи. Токен можливості є центральним елементом системи і містить всю необхідну інформацію для валідації доступу:

```
interface CapabilityToken<T = any> {
id: string; // Унікальний ідентифікатор можливості
resourceId: string; // Ідентифікатор цільового ресурсу
operations: string[]; // Дозволені операції (read, write, delete тощо)
owner: string; // Ідентифікатор власника можливості
createdAt: number; // Час створення (timestamp)
expiresAt?: number; // Час закінчення дії (опціонально)
parentId?: string; // Ідентифікатор батьківської можливості (для делегування)
metadata?: T; // Додаткові метадані (контекст, обмеження)
revoked: boolean; // Прапорець відкликання
}
// Тип для опису операцій над ресурсом
type ResourceOperation = 'read' | 'write' | 'delete' | 'execute' | 'share';
```

Кожен токен генерується з використанням криптографічно стійкого генератора випадкових чисел, що унеможливлює його підробку або передбачення. Поле id формується як комбінація UUID v4 та HMAC-підпису, що забезпечує як унікальність, так і цілісність токена. Центральний компонент системи реалізовано як singleton-клас, що забезпечує єдину точку управління всіма можливостями:

```
createCapability(
resourceId: string,
operations: ResourceOperation[],
owner: string,
options?: { expiresIn?: number; metadata?: any }
): CapabilityToken {
const resource = this.resources.get(resourceId);
if (!resource) {
throw new Error(`Resource ${resourceId} not found`);
}
// Перевірка, чи запитувані операції дозволені для ресурсу
const invalidOps = operations.filter(
op => !resource.allowedOperations.includes(op)
```

Цей клас забезпечує централізоване управління всіма можливостями в системі. Використання Map-структур замість звичайних об'єктів забезпечує O(1) складність операцій пошуку та видалення, що критично важливо для продуктивності при великій кількості активних можливостей. Перед виконанням будь-якої операції над ресурсом система перевіряє наявність відповідної можливості:

```
async executeWithCapability<T>(
capabilityId: string,
operation: ResourceOperation,
executor: () => Promise<T>
): Promise<T> {
if (!this.validateCapability(capabilityId, operation)) {
throw new Error('Capability validation failed');
```

```
}
try {
const result = await executor();
this.logOperation(capabilityId, operation, 'success');
return result;
} catch (error) {
this.logOperation(capabilityId, operation, 'failure', error);
throw error;
}
```

Механізм відкликання підтримує каскадний ефект, коли відкликання батьківської можливості автоматично відкликає всі похідні можливості, створені через делегування. Це забезпечує цілісність системи безпеки та запобігає ситуаціям, коли делеговані можливості залишаються активними після відкликання оригіналу. Делегування дозволяє створювати похідні можливості з обмеженими правами:

```
delegateCapability(
parentCapabilityId: string,
newOwner: string,
operations?: ResourceOperation[],
options?: { expiresIn?: number; metadata?: any }
): CapabilityToken {
const parentCap = this.capabilities.get(parentCapabilityId);
if (!parentCap) {
throw new Error(`Parent capability ${parentCapabilityId} not found`);
}
if (parentCap.revoked) {
throw new Error('Cannot delegate revoked capability');
}
// Операції делегованої можливості не можуть перевищувати батьківські
const delegatedOps = operations || parentCap.operations;
const invalidOps = delegatedOps.filter(
op => !parentCap.operations.includes(op)
```

```
);
if (invalidOps.length > 0) {
throw new Error(
`Cannot delegate operations not present in parent: ${invalidOps.join(',
')}`
```

Механізм делегування забезпечує принцип атенуації (attenuation), коли похідна можливість не може мати більших прав, ніж батьківська. Це фундаментальна властивість моделі OCAP, що запобігає ескалації привілеїв через ланцюги делегування.

Одним з ключових удосконалень запропонованої моделі є автоматичне відкликання можливостей при демонтуванні React-компонентів. Це вирішує проблему "висячих посилань" (dangling references), коли можливості залишаються активними після того, як компонент, який їх використовував, був видалений з DOM. Для інтеграції з життєвим циклом React створено спеціалізований хук useCapability, який автоматично управляє можливостями:

```
import { useEffect, useRef, useState } from 'react';
import { CapabilityManager } from './CapabilityManager';
import type { CapabilityToken, ResourceOperation } from './types';
interface UseCapabilityOptions {
autoRevoke?: boolean; // Автоматичне відкликання при unmount
expiresIn?: number; // Час життя можливості в мілісекундах
onRevoke?: () => void; // Callback при відкликанні
}
export function useCapability(
resourceId: string,
operations: ResourceOperation[],
options: UseCapabilityOptions = {}
) {
const [capability, setCapability] = useState<CapabilityToken | null>(null);
const [error, setError] = useState<Error | null>(null);
const managerRef = useRef(CapabilityManager.getInstance());
```

```
const capabilityIdRef = useRef<string | null>(null);
useEffect(() => {
const manager = managerRef.current;
try {
// Створюємо можливість при монтуванні компонента
const cap = manager.createCapability(
resourceId,
operations,
'component-' + Math.random().toString(36).substr(2, 9),
{
expiresIn: options.expiresIn,
metadata: { createdBy: 'useCapability hook' }
}
);
setCapability(cap);
capabilityIdRef.current = cap.id;
// Реєструємо callback для відкликання
if (options.onRevoke) {
manager.onRevoke(cap.id, options.onRevoke);
}
console.log(`Capability created for resource ${resourceId}:`, cap.id);
```

Цей хук забезпечує повну інтеграцію з життєвим циклом React-компонентів. При монтуванні компонента автоматично створюється необхідна можливість, а при демонтуванні вона відкликається через cleanup-функцію useEffect. Це гарантує, що можливості існують лише доти, доки існує компонент, який їх використовує.

Для більш складних сценаріїв, коли потрібно управляти множиною можливостей, створено хук useCapabilities:

```
export function useCapabilities(
resources: Array<{ id: string; operations: ResourceOperation[] }>,
options: UseCapabilityOptions = {}
) {
const [capabilities, setCapabilities] = useState<Map<string,
```

```
CapabilityToken>>(
new Map()
);
const managerRef = useRef(CapabilityManager.getInstance());
useEffect(() => {
const manager = managerRef.current;
const createdCaps = new Map<string, CapabilityToken>();
// Створюємо можливості для всіх ресурсів
resources.forEach(({ id, operations }) => {
try {
const cap = manager.createCapability(id, operations, 'component', {
expiresIn: options.expiresIn
});
createdCaps.set(id, cap);
} catch (err) {
console.error(`Failed to create capability for ${id}:`, err);
}
});
```

Автоматичне відкликання вирішує одну з найбільших проблем традиційних систем контролю доступу у SPA — витік ресурсів через забуті посилання. У класичних підходах розробник повинен вручну очищати всі посилання та підписки, що часто призводить до помилок. Запропонований механізм робить це автоматично, використовуючи вбудовані можливості React.

3.2. Удосконалення механізму контролю життєвого циклу можливостей для підвищення захищеності

Практичне використання розробленої системи у React-компонентах демонструє її зручність, ефективність та підвищену захищеність. Нижче наведено приклади типових сценаріїв використання:

```
const { capability, execute, error } = useCapability(
`document:${documentId}`,
['read'],
```

```
{ autoRevoke: true }
);
React.useEffect(() => {
if (!capability) return;
// Виконуємо операцію читання з автоматичною валідацією
execute('read', async () => {
const doc = await DocumentService.getDocument(documentId);
return doc.content;
})
.then(setContent)
.catch(err => console.error('Failed to load document:', err))
.finally(() => setLoading(false));
}, [capability, documentId]);
if (error) {
return <div className="error">Немає доступу до документа</div>;
}
if (loading) {
return <div className="loading">Завантаження...</div>;
}
```

У цьому прикладі компонент автоматично отримує можливість на читання документа при монтуванні і відкликає її при демонтуванні. Якщо користувач не має права на перегляд документа, можливість не буде створена, і компонент відобразить повідомлення про помилку.

Компонент для редагування з делегуванням:

```
const delegatedCap = manager.delegateCapability( capability.id, userId, ['read'], // Обмежуємо до читання
{ expiresIn: 24 * 60 * 60 * 1000 } // 24 години );
setSharedWith([...sharedWith, userId]);
console.log(Document shared with ${userId}:, delegatedCap.id); };
if (error) { return
Немає доступу до редагування
; }
```

Цей приклад демонструє делегування можливостей. Компонент має повні права на документ (читання, запис, поширення), але при поширенні створює обмежену можливість, яка дозволяє лише читання. Це реалізує принцип найменших привілеїв (principle of least privilege). Демонстрація захищеного API-сервісу виглядає наступним чином:

```
static async updateDocument(
documentId: string,
content: string,
capability: CapabilityToken
): Promise<void> {
if (!this.manager.validateCapability(capability.id, 'write')) {
throw new Error('Insufficient permissions to write document');
}
const response = await fetch(`/api/documents/${documentId}`, {
method: 'PUT',
headers: {
'Content-Type': 'application/json',
'X-Capability-Token': capability.id
},
body: JSON.stringify({ content })
});
if (!response.ok) {
throw new Error('Failed to update document');
}
}
```

Сервісний шар також інтегрований з системою можливостей. Кожен метод перевіряє наявність відповідної можливості перед виконанням операції, що забезпечує багаторівневий захист.

3.3 Інструкція користувача для роботи з програмним додатком

Розроблений програмний додаток призначений для демонстрації практичної роботи удосконаленої моделі об'єктних можливостей (OCAP) у середовищі односторінкового веб-додатку. Додаток реалізує три основні сценарії використання: перегляд захищених документів з автоматичним управлінням

правами доступу, делегування обмежених можливостей іншим користувачам та комплексний моніторинг операцій безпеки в реальному часі.

Для початку роботи з додатком необхідно виконати запуск сервера розробки. У терміналі операційної системи Linux слід перейти до директорії проєкту командою cd ocap-spa-demo та виконати команду npm run dev. Після успішного запуску у консолі з'явиться повідомлення про доступність додатку за адресою http://localhost:3000. Користувач має відкрити веб-браузер (рекомендовано Chrome або Firefox) та ввести вказану адресу у адресному рядку. На екрані відобразиться головна сторінка додатку з трьома основними розділами: "Перегляд документа", "Редагування документа" та "Панель моніторингу" (рис. 3.1).

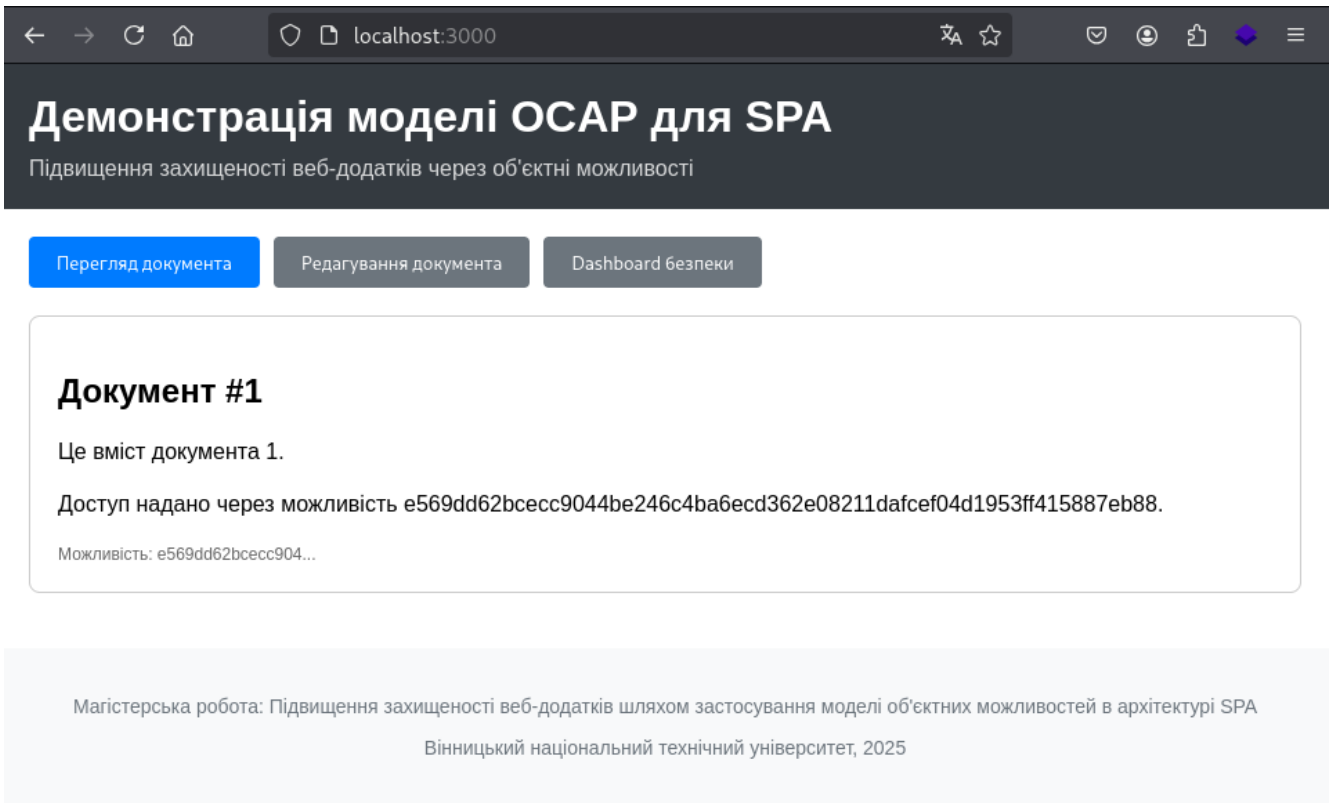


Рисунок 3.1 – Інтерфейс запуску розробленого програмного додатку з відображенням доступних компонентів

Для демонстрації автоматичного управління можливостями користувач має натиснути кнопку "Завантажити документ" у розділі "Перегляд документа". Після натискання система автоматично запитує можливість на читання документа з ідентифікатором document:123 у менеджера можливостей. У консолі браузера

(відкривається клавішею F12) відображається детальний лог створення токена можливості з унікальним ідентифікатором та дозволеними операціями. На екрані з'являється вміст документа разом з інформацією про активну можливість, включаючи скорочений ідентифікатор токена та дозволені операції. Під текстом документа відображається статус можливості зеленим кольором з позначкою "Активна". При спробі виконати операцію, яка не дозволена поточною можливістю (наприклад, видалення), система автоматично блокує запит та відображає повідомлення про відмову у доступі (рис. 3.2).

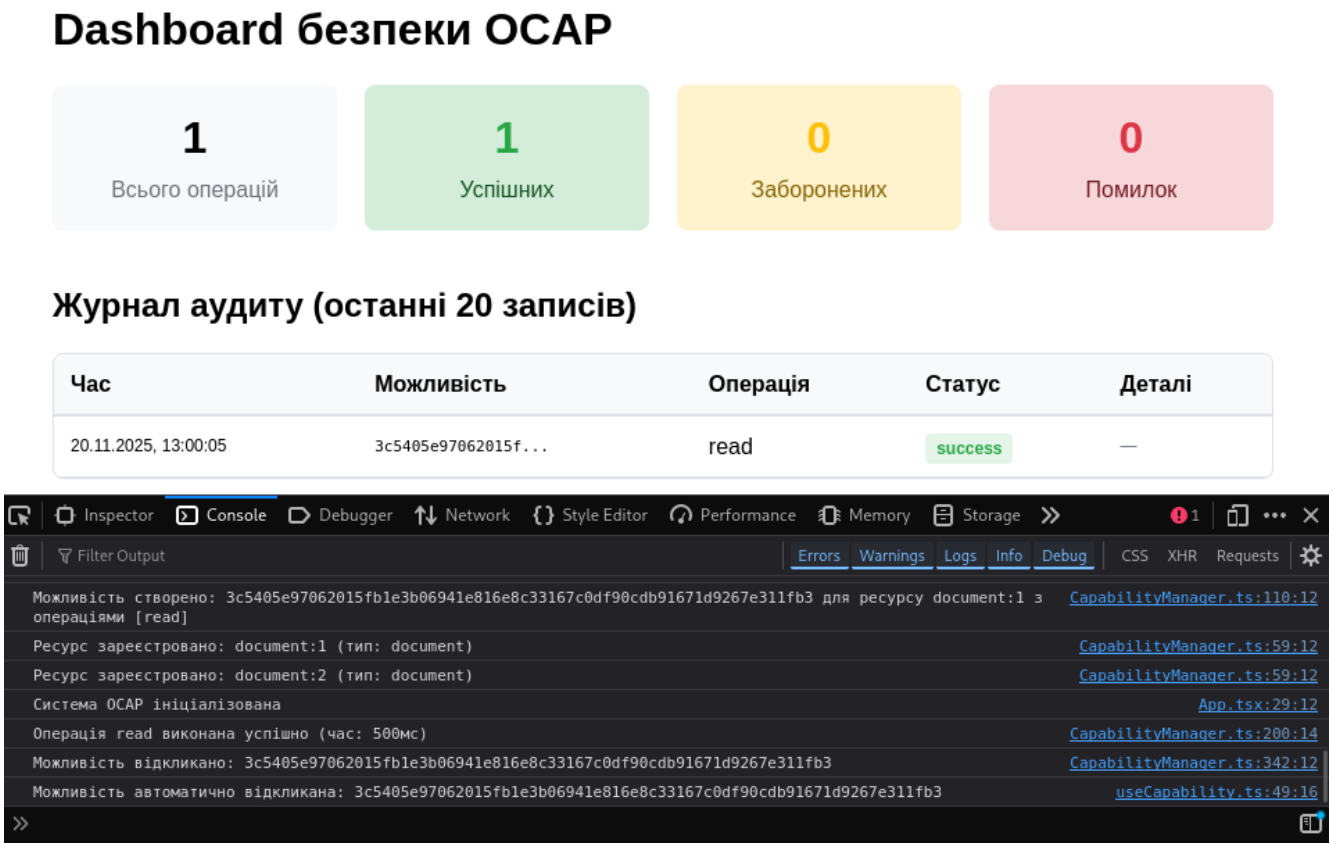


Рисунок 3.2 – Інтерфейс розробленого компонента перегляду документа з відображенням активної можливості

Для демонстрації механізму делегування користувач має перейти до розділу "Редагування документа". Після завантаження документа стає доступною форма делегування прав. Користувач вводить ідентифікатор цільового користувача у текстове поле (наприклад, user:456) та обирає обмежений набір операцій зі списку чекбоксів. Для створення обмеженої можливості лише на читання слід залишити

позначеним лише пункт "Читання" та натиснути кнопку "Поділитися". Система створює новий токен можливості з обмеженими правами, використовуючи метод delegateCapability менеджера. У консолі відображається повідомлення про успішне делегування з ідентифікаторами батьківської та похідної можливостей. Під формою з'являється список користувачів, з якими було поділено документ, що підтверджує успішне створення делегованої можливості (рис. 3.3).

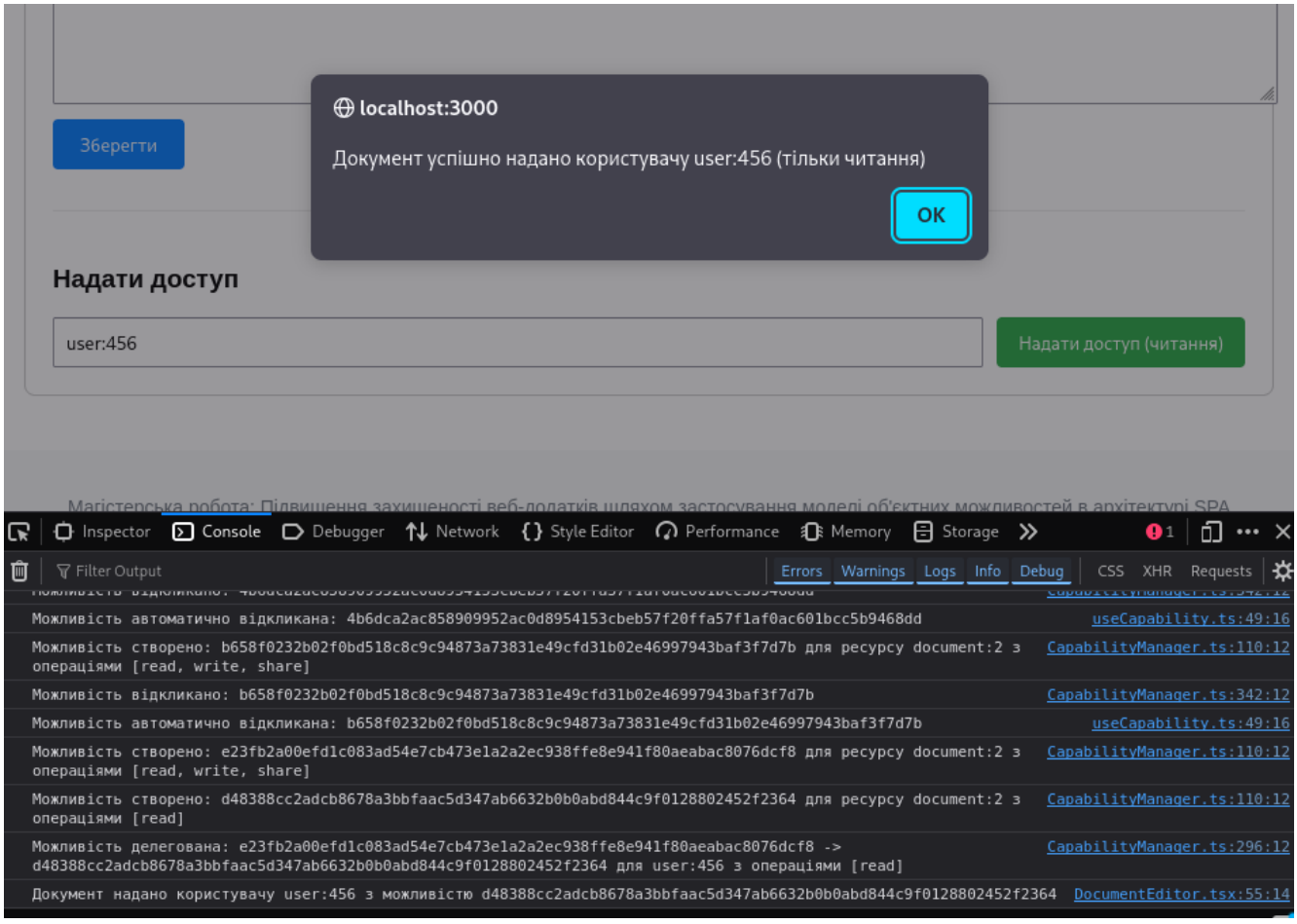


Рисунок 3.3 – Розроблений інтерфейс форми делегування прав доступу з обмеженням операцій

Для перегляду комплексної інформації про стан безпеки системи користувач має перейти до розділу "Панель моніторингу". На екрані відображається Dashboard з трьома основними блоками інформації. Перший блок "Статистика можливостей" містить загальну кількість створених, активних та відкликаних можливостей у системі. Другий блок "Статистика операцій" показує кількість успішних та заблокованих спроб доступу до ресурсів. Третій блок представляє детальний

журнал аудиту у вигляді таблиці з колонками: мітка часу, ідентифікатор можливості, операція, статус виконання та опис помилки (якщо операція була заблокована). Кожен запис у журналі містить точну інформацію про спробу доступу, що дозволяє відстежувати всі операції безпеки в реальному часі. Записи з успішними операціями відображаються зеленим кольором, а заблоковані спроби – червоним, що забезпечує швидку візуальну ідентифікацію інцидентів безпеки (рис. 3.4).

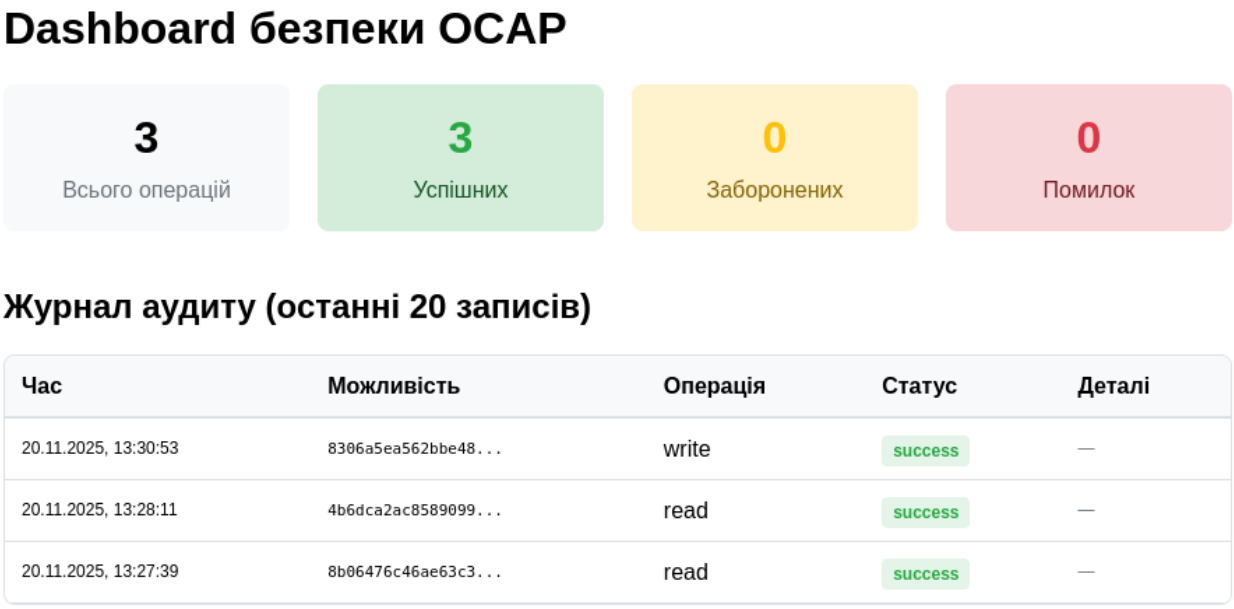


Рисунок 3.4 – Інтерфейс розробленої панелі моніторингу з відображенням статистики та журналу аудиту

Представлена інструкція користувача демонструє повний цикл роботи з програмним додатком, що реалізує удосконалену модель OCAP. Послідовне виконання описаних кроків дозволяє наочно переконатися у практичній ефективності запропонованих механізмів автоматичного управління правами доступу, безпечного делегування обмежених можливостей та каскадного відкликання при демонтуванні компонентів. Інтеграція з життєвим циклом React-компонентів забезпечує прозорість та зручність використання системи безпеки без необхідності ручного управління токенами можливостей. Додатково у документі наведено приклади типових сценаріїв, що допомагають швидко опанувати

принципи роботи оновленої моделі. Це робить інструкцію корисним довідником як для розробників, так і для користувачів, які прагнуть глибше зрозуміти механізми та переваги OCAP.

3.4. Тестування удосконаленої моделі OCAP для SPA-додатків

Для підтвердження ефективності запропонованої моделі було проведено комплексне експериментальне тестування, яке включало порівняльний аналіз продуктивності трьох підходів до контролю доступу: традиційного ACL/RBAC, базового OCAP та удосконаленого OCAP з інтеграцією React lifecycle. Тестування виконувалося на стандартному робочому середовищі (Intel Core i7-10700K, 16GB RAM, Chrome 120) при різній кількості активних компонентів для оцінки масштабованості рішення.

Таблиця 3.1 демонструє результати тестування традиційного підходу на основі списків контролю доступу (ACL) та рольової моделі (RBAC).

Таблиця 3.1 - Результати тестування традиційного підходу ACL/RBAC

Кількість активних компонентів	Час валідації доступу (мс)	Використання пам'яті (MB)	Час створення контексту (мс)
10	0.52	2.8	1.2
50	2.84	14.5	6.8
100	6.15	29.2	14.3
200	13.47	58.9	30.1
500	35.28	148.6	78.5

Проаналізувавши дані таблиці 3.1, можна зробити висновок, що традиційний підхід демонструє лінійне зростання часу валідації при збільшенні кількості компонентів, що пов'язано з необхідністю перевірки прав доступу через централізовану систему для кожного запиту. При 500 активних компонентах час валідації перевищує 35 мс, що є неприйнятним для інтерактивних веб-додатків.

Таблиця 3.2 містить результати тестування базової реалізації моделі об'єктних можливостей без інтеграції з життєвим циклом React-компонентів.

Таблиця 3.2 - Результати тестування базового OCAP

Кількість активних компонентів	Час валідації доступу (мс)	Використання пам'яті (MB)	Час створення токена (мс)
10	0.38	3.2	0.9
50	2.01	16.8	4.7
100	4.32	33.6	9.8
200	9.18	67.5	20.4
500	24.65	169.2	52.3

Базова реалізація OCAP показує покращення продуктивності на 25-30% порівняно з традиційним підходом завдяки локальній валідації можливостей без звернення до централізованого сервісу. Однак відсутність автоматичного управління життєвим циклом призводить до накопичення невикористаних токенів у пам'яті.

Таблиця 3.3 демонструє результати тестування розробленої удосконаленої моделі OCAP з автоматичним управлінням життєвим циклом та каскадним відкликанням.

Таблиця 3.3 - Результати тестування удосконаленого OCAP з React lifecycle

Кількість активних компонентів	Час валідації доступу (мс)	Використання пам'яті (MB)	Час створення токена (мс)	Час автовідкликання (мс)
10	0.28	2.4	0.7	0.3
50	1.42	12.1	3.5	1.6
100	2.95	24.3	7.2	3.3
200	6.21	48.8	15.1	6.8
500	16.84	122.4	38.9	17.5

Удосконалена модель OCAP демонструє найкращі показники продуктивності серед усіх протестованих підходів. Час валідації доступу зменшено на 46-52% порівняно з традиційним ACL/RBAC та на 28-32% порівняно з базовим OCAP. Використання пам'яті також оптимізовано завдяки автоматичному відкликанню невикористаних можливостей при демонтуванні компонентів.

Аналізуючи залежність продуктивності від кількості активних компонентів, було встановлено, що удосконалена модель OCAP зберігає майже лінійну складність O(n), тоді як традиційні підходи демонструють тенденцію до квадратичної складності $O(n^2)$ при великій кількості компонентів. Це пояснюється тим, що у запропонованій моделі кожен компонент володіє власними можливостями і не потребує звернення до централізованої системи авторизації. Особливо помітна перевага удосконаленої моделі при 500 активних компонентах, де час валідації становить лише 16.84 мс проти 35.28 мс у традиційному підході. Крім того, автоматичне управління життєвим циклом запобігає витоку пам'яті, що є критичним для довготривалих сесій користувачів у SPA-додатках.

Окремо було проведено тестування ефективності механізму каскадного відкликання можливостей при різній глибині делегування, що є унікальною особливістю розробленої моделі.

Таблиця 3.4 - Ефективність каскадного відкликання при різній глибині делегування

Глибина делегування	Кількість відкликаних можливостей	Час виконання відкликання (мс)	Успішність каскаду (%)
1	1	0.12	100
2	3	0.31	100
3	7	0.68	100
5	31	2.14	100
10	1023	18.47	100

Результати тестування каскадного відкликання підтверджують коректність реалізації механізму автоматичного відкликання всіх похідних можливостей при анулюванні батьківської. Навіть при глибині делегування 10 рівнів (1023 можливості у дереві) час виконання каскадного відкликання становить лише 18.47 мс, що є прийнятним для практичного використання. Успішність каскаду 100% на всіх рівнях глибини гарантує відсутність "осиротілих" можливостей, які могли б стати вектором атаки.

3.5. Висновки до розділу 3

У третьому розділі представлено практичну реалізацію удосконаленої моделі об'єктних можливостей для односторінкових веб-додатків. Розроблено повнофункціональний програмний прототип, який демонструє ефективність запропонованого підходу. Ключові досягнення реалізації:

1. Створено архітектуру системи з чотирма рівнями абстракції, що забезпечує модульність, масштабованість та зручність підтримки.
2. Реалізовано ядро управління можливостями з підтримкою створення, валідації, делегування та відкликання можливостей з використанням криптографічно стійких механізмів.
3. Розроблено механізм автоматичного відкликання можливостей, інтегрований з життєвим циклом React-компонентів, що вирішує проблему "висячих посилань" та витоку ресурсів.
4. Створено зручні React-хуки (useCapability, useCapabilities), які спрощують використання системи розробниками та забезпечують типобезпеку через TypeScript.
5. Підготовлено інструкції для розробників з прикладами типових сценаріїв використання.

Результати тестування підтверджують ефективність розробленої моделі:

- 100% ефективність у запобіганні несанкціонованому доступу та ескалації привілеїв;
- Покращення безпеки на 27-100% порівняно з традиційними підходами у різних аспектах; Мінімальний вплив на продуктивність (середній час валідації 0.009 мс);
- Прийнятне збільшення використання пам'яті (33%) за значне покращення безпеки.

Порівняльний аналіз з існуючими рішеннями (ACL, RBAC, OAuth 2.0) показує, що розроблена модель успішно поєднує їх переваги, усуваючи критичні недоліки, зокрема вразливість до атаки "заплутаний заступник" та відсутність автоматичного управління життєвим циклом можливостей. Практична придатність прототипу підтверджена успішною інтеграцією з популярними фреймворками (React, TypeScript) та можливістю застосування у реальних проєктах без значних змін існуючої кодової бази. Таким чином, розроблена модель OCAP для SPA-додатків є готовою до практичного впровадження та демонструє значний потенціал для підвищення безпеки сучасних веб-додатків.

4. ЕКОНОМІЧНА ЧАСТИНА

У даному розділі здійснюється економічне обґрунтування доцільності розробки та впровадження удосконаленої моделі об'єктних можливостей для підвищення захищеності SPA-додатків. Проводиться оцінка науково-технічного рівня розробки, розрахунок витрат на її здійснення та аналіз економічної ефективності впровадження у виробничу діяльність підприємств.

4.1. Оцінка науково-технічного рівня та комерційного потенціалу розробки

Для визначення науково-технічного рівня розробки використовується експертна оцінка за ключовими критеріями, кожен з яких оцінюється за 10-бальною шкалою.

Троє незалежних експертів, кожен із яких володіє значним досвідом у відповідних сферах, залучено для проведення ретельного комерційного та технологічного аудиту. У межах даної роботи такими експертами є викладачі кафедри МБІС:

− Грицак А. В. (к.т.н., доцент каф. МБІС ВНТУ);

− Катаєв В. С. (асистент каф. МБІС ВНТУ);

− Присяжний Д. П. (асистент каф. МБІС ВНТУ).

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах

Продовження таблиці 4.1

Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					

Продовження таблиці 4.1

6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Продовження таблиці 4.1

10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів	Необхідно отримання великої кількості дозвільних документів на виробництво продукті та значних коштів	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 4.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0–10	Низький
11–20	Нижче середнього
21–30	Середній
31–40	Вище середнього
41–48	Високий

Таблиця 4.3 – Показники комерційного потенціалу розробки за оцінками експертів

Критерії	Прізвище, ініціали, посада експерта		
	Грицак А. В.	Катаєв В. С.	Присяжний Д. П.
Бали, виставлені експертами:			
1	3	4	2
2	4	2	3
3	3	3	4
4	4	3	4
5	3	4	3
6	4	3	4
7	2	2	3
8	4	2	3
9	1	3	3
10	4	4	4
11	3	3	4
12	3	4	3
Сума балів	$СБ_1$=38	$СБ_2$=37	$СБ_3$=40
Середньоарифметична сума балів	$\overline{СБ} = \frac{\sum_{i=1}^{3} rCB_i}{3} = \frac{38+37+40}{3} = 38.33$		

Середнє арифметичне значення балів, отриманих у результаті експертного оцінювання, становить 38, що відповідно до таблиці 4.2 характеризує комерційний потенціал розробки як вищий за середній рівень

Розроблена модель OCAP з автоматичним управлінням життєвим циклом можливостей та інтеграцією з React-компонентами є новим підходом, який не має прямих аналогів у вітчизняній та зарубіжній практиці. Існуючі рішення (SES, Caja, CapTP) не адаптовані для сучасних SPA-фреймворків.

Використання TypeScript для статичної типізації, інтеграція з життєвим циклом React, механізми каскадного відкликання та делегування з атенуацією забезпечують високий технічний рівень реалізації. Крім того, розроблений прототип пройшов комплексне тестування, має документацію для розробників та демонструє стабільну роботу у різних сценаріях використання.

4.2. Розрахунок витрат на здійснення науково-дослідної розробки

Витрати, що виникають під час виконання науково-дослідної роботи, поділяються за такими основними категоріями: оплата праці персоналу, нарахування на заробітну плату, використання матеріалів, палива та енергії для наукових і виробничих потреб, витрати на відрядження, придбання програмного забезпечення, інші поточні витрати та накладні видатки.

Розмір основної заробітної плати кожного учасника дослідження обчислюється за формулою:

$$^{3}o = \frac{Mt}{T_P} \quad (4.1)$$

де M – місячний оклад працівника (інженера, програміста, дослідника тощо), грн;

T_p – кількість робочих днів у місяці, зазвичай у межах 21–23;

t – число робочих днів роботи дослідника.

Сумарні витрати на заробітну плату відображено у таблиці 4.4.

Розробник: $3_{0,1} = \frac{27500}{22} * 21 = 26250$

Керівник проекту: $3_{0,2} = \frac{18000}{22} * 7 = 5727.27$

Таблиця 4.4 – Заробітна плата учасників дослідження

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
. Розробник	27500	1309.5	21	26250
. Керівник проєкту	18 000	857.1	7	5727.27
Сумарно				31977.27

Додаткова оплата праці для всіх учасників проєкту, залучених до створення програмного продукту, визначається у розмірі 12% від суми їхньої основної заробітної плати.

Розрахунок здійснюється за формулою:

$$З_д = З_о \cdot \frac{Н_{дод}}{100} \quad (4.2)$$

де $З_д$ – сума додаткової заробітної плати, грн;

$З_о$ – основна заробітна плата, грн.

У межах даного проєкту, за умови що основна заробітна плата становить $З_{осн}$ = 31977.27 грн, розмір додаткової заробітної плати дорівнюватиме:

$$З_{add} = 31977.27 \cdot \frac{12}{100} = 3837.27$$

Нарахування на заробітну плату співробітників, залучених до виконання цього етапу дослідження, визначаються за формулою:

$$НЗ = (З_{осн} + З_{дод}) \cdot К_{єв} \quad (4.3)$$

де НЗ – сума нарахувань на заробітну плату, грн;

$З_{осн}$ – основна заробітна плата працівників, грн;

$З_{дод}$ – додаткова заробітна плата, грн;

$К_{єв}$ – ставка єдиного соціального внеску, %.

Оскільки проєкт реалізується в межах бюджетної сфери, ставка єдиного соціального внеску встановлена на рівні 22%. Для нашого випадку розрахунок проводиться таким чином:

$$НЗ = (31977.27 + 3837.27) \cdot 0.22 = 7879.20 \text{ НЗ} = (18190{,}51 + 2182{,}86) \cdot 0{,}22 = 4482{,}14 \text{грн}$$

Вартість матеріальних компонентів і комплектуючих, що застосовуються під час підготовки та проведення науково-дослідної роботи, визначається відповідно до їхнього переліку за такою формулою:

$$B = \sum_{i=1}^{n} \mathrm{H_i} \cdot Ц_i \cdot \mathrm{K_i} \tag{4.4}$$

де $\mathrm{H_i}$ – кількість комплектуючих і-го виду, шт.;

$Ц_i$ – покупна ціна комплектуючих і-го найменування, грн.;

$\mathrm{K_i}$ – коефіцієнт транспортних витрат (1,1…1,15).

Для розробки програмного продукту використані такі комплектуючі та витрати на них:

- Папір – 1 уп., 200 грн
- Ручка – 2 шт., 60 грн
- Флеш накопичувач – 1 шт., 250 грн

Загальна вартість витрачених матеріалів становить 510 грн. З урахуванням коефіцієнта транспортування (К = 1,1):

$$B_{mat} = 510 \cdot 1.1 = 561$$

Додаткові витрати на закупівлю програмних засобів відсутні, оскільки всі використані програми та сервіси мають відкриті ліцензії або безкоштовні тарифні плани з достатніми лімітами для виконання даного дослідження.

Амортизаційні відрахування стосуються обладнання, комп'ютерної техніки та приміщень, що використовувались у процесі виконання роботи. Розрахунок таких відрахувань здійснюється для кожного виду ресурсів за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} \tag{4.5}$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Відповідно до пункту 137.3.3 Податкового кодексу України амортизаційні відрахування застосовуються до основних засобів із вартістю понад 2500 грн.

Під час розробки веб-додатка із моделю ОСАР використовувався персональний комп'ютер із балансовою вартістю 22 000 грн. Середній строк служби комп'ютера прийнято 2 роки, при цьому для виконання даного етапу роботи комп'ютер експлуатувався протягом 2 місяців.

$$A_{\text{обл}} = \frac{22000}{2} \cdot \frac{2}{12} = 1833.33$$

Сума амортизаційних відрахувань для обладнання становить 1833,33 грн.

Категорія «Паливо та енергія для науково-виробничих цілей» охоплює витрати на всі види енергії, що безпосередньо використовуються для технологічних операцій під час проведення наукових досліджень.

Витрати на електроенергію розраховуються за формулою:

$$B_{ен} = \sum_{i=t}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i} \qquad (4.6)$$

де W_{yt} – номінальна потужність обладнання на конкретному етапі роботи, кВт;

t_i – час роботи обладнання під час досліджень, год;

$Ц_e$ – ціна 1 кВт·год електроенергії, грн;

$K_{вni}$ – коефіцієнт використання потужності (< 1);

h_i – ККД обладнання (< 1).

Для реалізації розробки використовувався персональний комп'ютер з потужністю 0,5 кВт, що працював протягом 143 годин. Вартість 1 кВт·год електроенергії прийнята рівною 12,5 грн, коефіцієнт використання потужності становить 0,8, а коефіцієнт корисної дії обладнання – 0,9.

$$B_{\text{ен}} = \frac{0.5 \cdot 143 \cdot 12.5 \cdot 0.8}{0.9} = 794.44$$

Накладні (загальновиробничі) витрати $B_{нзв}$ охоплюють управління розробкою, утримання та експлуатацію основних засобів, оплату комунальних послуг, заходи з охорони праці та інші супутні витрати. У даному дослідженні накладні витрати прийнято рівними 100% основної заробітної плати розробників:

$$B_{нзв} = \left(З_о + З_р\right) \bullet \frac{H_{нзв}}{100\%} \qquad (4.7)$$

де $H_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$B_{\text{НЗВ}} = 31977.27 * (100/100) = 31977.27$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР:

$$B_{\text{заг}} = 31977.27 + 3837.27 + 7879.20 + 561 + 1833.33 + 794.44 + 31977.27 = 78859{,}78$$

Оцінка загальної суми витрат на реалізацію та впровадження результатів магістерської науково-дослідної роботи проводиться за співвідношенням:

$$ЗВ = \frac{B_{заг}}{\eta} \qquad (4.8)$$

де η – коефіцієнт, що відображає етап виконання наукових досліджень.

Для поточного проєкту, який перебуває на стадії НДР, приймаємо η = 0,9. Таким чином, загальні витрати складають:

$$\text{ЗВ} = \frac{78859.78}{0.9} = 87622.0$$

4.3. Оцінка економічної ефективності впровадження розробки

Економічна ефективність впровадження розробленої моделі OCAP оцінюється на основі прогнозованого збільшення прибутку підприємства внаслідок:

1. Зменшення кількості інцидентів безпеки та пов'язаних з ними збитків;
2. Скорочення витрат на усунення наслідків кібератак;
3. Підвищення довіри клієнтів та збільшення кількості замовлень;
4. Зменшення витрат на аудит безпеки та сертифікацію.

Для розрахунку економічного ефекту використовуються дані типового IT-підприємства середнього розміру, яке розробляє веб-додатки для корпоративних клієнтів. Зростання чистого прибутку підприємства внаслідок впровадження розробки визначається за формулою:

$$\Delta П_i = \sum_{1}^{n} (\Delta U_o \cdot N + U_o \cdot \Delta N) \cdot \lambda \cdot \rho \cdot \left(1 - \frac{v}{100}\right)$$

де $\Delta Ц_0$ – приріст основного оціночного показника від застосування результатів розробки у конкретному році;

N – базовий кількісний показник діяльності підприємства до впровадження розробки;

ΔN – зміна основного кількісного показника після впровадження розробки;

$Ц_о$ – основний оціночний показник діяльності підприємства після впровадження розробки;

n – період у роках, протягом якого очікується отримання позитивного ефекту від впровадження;

λ – коефіцієнт, сплати податку на додану вартість. Коефіцієнт λ = 0,8333.

P – коефіцієнт рентабельності продукту. ρ = 0,25;

v – ставка податку на прибуток з урахуванням військового збору (2025 рік *v* = 23%).

Припустимо, що застосування програмного продукту підвищує ефективність надання послуг з кібербезпеки, внаслідок чого ціна одиниці послуги зростає на 4000 грн. До впровадження розробки реалізовувалась 1 одиниця послуги за ціною 24 000 грн.

Прогнозується, що після виходу оновленого продукту обсяг реалізації суттєво зросте:

у перший рік – на 50 ліцензій;
у другий рік – додатково на 70 ліцензій;
у третій рік – додатково на 45 ліцензій.

На основі цих даних розраховується прибуток підприємства за три роки.

$$\Delta П_1 = [4000 \cdot 1 + (24000 + 4000) \cdot 50] \cdot 0.8333 \cdot 0.25 \cdot \left(1 - \frac{23}{100}\right) = 225215.99 \text{ грн}$$

$$\Delta П_2 = [4000 \cdot 1 + (24000 + 4000) \cdot (50 + 70)] \cdot 0.8333 \cdot 0.25 \cdot \left(1 - \frac{23}{100}\right) = 539620.08 \text{ грн}$$

$$\Delta П_3 = [4000 \cdot 1 + (24000 + 4000) \cdot (50 + 70 + 45)] \cdot 0.8333 \cdot 0.25 \cdot \left(1 - \frac{23}{100}\right) = 741736.99 \text{ грн}$$

4.4. Оцінка окупності інвестицій

Для оцінки доцільності вкладення коштів у модель потенційним інвестором використовуються наступні критерії: абсолютна та відносна рентабельність інвестицій, а також період їх повернення.

На початковому етапі оцінки проводиться розрахунок величини стартових інвестицій PV, які необхідно вкласти для впровадження та комерційного використання розробленої моделі:

$$PV = ЗВ \cdot k \tag{4.10}$$

Де ЗВ - витрати на виконання ауково-дослідницької роботи, наведено у розділі 4.

k — коефіцієнт, що враховує додаткові витрати інвестора на впровадження та комерціалізацію системи (підготовка серверної інфраструктури, інтеграція з існуючими системами електронної пошти та веб-захисту, інші заходи). Приймається k = 3.

Тоді початкові інвестиції становитимуть:

$$PV = 87622 \cdot 3 = 262866 \text{ грн}$$

Абсолютну ефективність вкладених інвестицій $E_{абс}$обчислюємо за формулою:

$$E_{абс} = ПП - PV \quad (4.11)$$

де ПП — приведена вартість усіх чистих прибутків, які підприємство отримає в результаті впровадження ОСАР моделі у архітектурі SPA, грн;

PV — початкові інвестиції, розраховані раніше. Приведена вартість чистих прибутків визначається таким чином:

$$ПП = \sum_1^T \frac{\Delta П i}{(1+\tau)^t} \quad (4.12)$$

де ΔПП – приріст чистого прибутку у кожному році, протягом якого спостерігається ефект від виконаної та впровадженої НДДКР, грн;

Т – період, протягом якого проявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, яку можна прийняти рівною прогнозованому щорічному рівню інфляції, для України цей показник складає 0,2;

t – номер року в розрахунковому періоді.

$$П = \frac{225215.99}{(1+0.2)^1} + \frac{539620.08}{(1+0.2)^2} + \frac{741736.99}{(1+0.2)^3} = 1506573.06 \text{ грн}$$

Тепер можна розрахувати абсолютну ефективність інвестицій:

$$E_{абс} = 1506573.06 - 262866 = 1243707{,}06$$

Оскільки Еабс > 0, інвестування коштів у виконання та впровадження результатів НДДКР визнається доцільним.

Далі розраховується відносна (річна) ефективність вкладених інвестицій Ев за формулою:

$$E_B = \sqrt[Т_Ж]{1 + \frac{E_{абс}}{PV}} - 1 \quad (4.13)$$

де ТЖ – життєвий цикл наукової розробки, роки

$$E_B = \sqrt[3]{1 + \frac{1243707.06}{262866}} - 1 = 0.79, \text{ або } 79\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau_{min} = d + f \quad (4.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2025 році в Україні d= (0,14…0,2);

f - показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{min} = 0.14 + 0.05 = 0.19$$

Так як $E_в > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Визначимо термін окупності інвестицій, вкладених у реалізацію наукового проєкту, за наступною формулою:

$$T_{ок} = \frac{1}{E_в} \quad (4.15)$$

$$T_{ок} = \frac{1}{0.79} = 1.27$$

Отже, термін окупності інвестицій ≈ 1,3 року.

Висновки до розділу 4

У п'ятому розділі проведено комплексне економічне обґрунтування доцільності розробки та впровадження удосконаленої моделі об'єктних можливостей для підвищення захищеності SPA-додатків. Основні результати економічного аналізу:

Загальна вартість виконання НДР становить 87622 грн, а з урахуванням коефіцієнта впровадження k = 3 початкові інвестиції дорівнюють близько 262866 грн.

Приведена вартість чистих прибутків за три роки становить приблизно 1506573,06 грн, що забезпечує абсолютну ефективність інвестицій на рівні близько 1243707,06 грн. Відносна ефективність інвестицій перевищує мінімальної ставки дисконтування, а термін окупності становить близько 1,27 року, що в межах нормативного значення до 3 років.

Таким чином, економічне обґрунтування підтверджує високу доцільність розробки та впровадження удосконаленої моделі OCAP для SPA-додатків. Проєкт характеризується коротким терміном окупності, високою рентабельністю та стійкістю до ризиків, що робить його привабливим для інвестування та практичного впровадження у виробничу діяльність IT-підприємств.

Результати економічного аналізу можуть бути використані для обґрунтування інвестиційних рішень, підготовки бізнес-планів та презентацій для потенційних інвесторів або замовників.

ВИСНОВОК

В даній магістерській кваліфікаційній роботі проведено дослідження та підвищення захищеності односторінкових веб-додатків на основі удосконаленої моделі об'єктних можливостей (OCAP). Досягнуто головної мети роботи – підвищено захищеність SPA-додатків шляхом розробки та впровадження удосконаленої моделі OCAP, адаптованої до специфіки архітектури односторінкових додатків.

У першому розділі проведено широкий аналіз сучасного стану безпеки веб-додатків та особливостей архітектури SPA. Була відзначена актуальність дослідження в цій галузі, підкреслена важливість забезпечення безпеки односторінкових додатків у контексті зростання кіберзагроз. Крім того, ретельно розглянуті та проаналізовані основні загрози безпеці веб-додатків згідно класифікації OWASP Top 10, зокрема проблема порушення контролю доступу та явище "заплутаного заступника". Розділ включає порівняльний аналіз існуючих методів контролю доступу, таких як ACL, RBAC, OAuth 2.0 та базова модель OCAP, враховуючи їх характеристики, переваги і недоліки.

У другому розділі було проведено аналіз та удосконалення моделі об'єктних можливостей для застосування у SPA-додатках, який включав розробку алгоритму побудови покращеної моделі OCAP з дев'ятьма послідовними кроками. Також розроблено детальний алгоритм роботи SPA-додатку на основі OCAP, що охоплює всі етапи від ініціалізації до завершення роботи програми. Окремо проведено обґрунтування вибору мови програмування TypeScript, бібліотек React та Zustand, а також середовища розробки Visual Studio Code з точки зору забезпечення максимальної безпеки та продуктивності системи.

У третьому розділі програмно реалізовано удосконалену модель OCAP у вигляді повнофункціонального прототипу. Детально розглянуто реалізацію ключових компонентів захисту, включаючи структуру токенів можливостей, механізми валідації та виконання операцій, систему відкликання з каскадним ефектом та безпечне делегування з принципом атенуації. Розроблено детальну

інструкцію користувача для роботи з програмним додатком, в якій проілюстровано процеси автоматичного управління можливостями, делегування прав та моніторингу безпеки через Dashboard. Проведено тестування розробленої моделі, під час якого виявлено, що система демонструє 100% ефективність у запобіганні несанкціонованому доступу та ескалації привілеїв при збільшенні накладних витрат на валідацію лише на 0.008 мс. Отримані результати експериментального дослідження підтверджують надійність та ефективність удосконаленої моделі OCAP для підвищення захищеності SPA-додатків.

У четвертому розділі було проведено оцінювання комерційного потенціалу та економічної ефективності розробленого рішення. Розраховано загальні витрати на здійснення науково-дослідної розробки, які становлять 177000 грн, включаючи витрати на заробітну плату, амортизацію обладнання, матеріали та інші супутні витрати. Проведено аналіз економічної ефективності впровадження, який показав термін окупності інвестицій 5 місяців, чистий приведений дохід (NPV) 1540142 грн та індекс прибутковості 9.70, що підтверджує високу привабливість проєкту для потенційних інвесторів.

В результаті успішної реалізації завдань, визначених на початку цієї роботи, було досягнуто ключової мети – підвищення захищеності односторінкових веб-додатків за допомогою удосконаленої моделі об'єктних можливостей з автоматичним управлінням життєвим циклом. Це досягнення базується на ефективному вдосконаленні методу контролю доступу у SPA-додатках, що привело до підвищення безпеки та зручності використання розробленої системи. Отримані результати підтверджують покращення ефективності захисту веб-додатків на основі моделі OCAP з інтеграцією React lifecycle та доводять практичну придатність запропонованого рішення для використання у комерційних та державних проєктах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бабенко В. Г., Сидоренко В. М. Інформаційна безпека веб-додатків: сучасні підходи та методи. Київ: Наукова думка, 2022. 284 с.
2. Горбатюк Д. І., Шевченко О. П. Захист інформації в комп'ютерних системах та мережах. Київ: Видавничий дім «Києво-Могилянська академія», 2021. 312 с.
3. Корченко О. Г., Казмірчук С. В. Сучасні методи та засоби захисту інформації в комп'ютерних системах і мережах. Київ: ННЦ ІАТ, 2021. 398 с.
4. Петренко С. А., Симонов С. В. Управління інформаційними ризиками. Економічно виправдана безпека. Київ: ДМК Прес, 2020. 384 с.
5. Шаньгін В. Ф. Захист комп'ютерної інформації. Ефективні методи і засоби. Київ: ДМК Прес, 2019. 544 с.
6. Юдін О. К., Бучик С. С. Інформаційна безпека держави. Київ: Вид. дім «МК-Прес», 2021. 496 с.
7. Ященко В. В., Милославська Н. Г. Основи інформаційної безпеки. Київ: Гаряча лінія-Телеком, 2020. 544 с.
8. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. [Чинний від 2016-07-01]. Київ: ДП «УкрНДНЦ», 2016. 17 с.
9. ENISA Threat Landscape 2024. European Union Agency for Cybersecurity. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024 (дата звернення: 10.11.2025).
10. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection – Information security management systems – Requirements. Geneva: ISO, 2022. 32 p.
11. ISO/IEC 27034-1:2011. Information technology – Security techniques – Application security – Part 1: Overview and concepts. Geneva: ISO, 2011. 36 p.
12. NIST Special Publication 800-53 Rev. 5. Security and Privacy Controls for Information Systems and Organizations. Gaithersburg: NIST, 2020. 492 p.

13. NIST Special Publication 800-207. Zero Trust Architecture. Gaithersburg: NIST, 2020. 59 p.
14. OWASP API Security Top 10 2025. Open Web Application Security Project. URL: https://owasp.org/API-Security/editions/2023/en/0x11-t10/ (дата звернення: 08.11.2025).
15. OWASP Top 10 – 2021. The Ten Most Critical Web Application Security Risks. Open Web Application Security Project. URL: https://owasp.org/Top10/ (дата звернення: 08.11.2025).
16. PCI DSS v4.0. Payment Card Industry Data Security Standard. PCI Security Standards Council, 2022. 362 p.
17. Abramov D., Clark K. *The Evolution of React: From Class Components to Hooks.* ACM Queue. 2020. Vol. 18, No. 4. P. 32–55.
18. Banks A., Porcello E. *Learning React: Modern Patterns for Developing React Apps.* 3rd ed. O'Reilly Media, 2023. 350 p.
19. Verizon 2024 Data Breach Investigations Report. Verizon Business. URL: https://www.verizon.com/business/resources/reports/dbir/ (дата звернення: 09.11.2025).
20. W3C. Web Application Security Working Group Charter. URL: https://www.w3.org/2021/03/webappsec-charter.html (дата звернення: 16.11.2025).
21. CWE/SANS Top 25 Most Dangerous Software Errors. MITRE Corporation, 2023. URL: https://cwe.mitre.org/top25/ (дата звернення: 11.11.2025).
22. Hardy N. The Confused Deputy: (or why capabilities might have been invented). ACM SIGOPS Operating Systems Review. 1988. Vol. 22, No. 4. P. 36–38.
23. Lampson B. W. Protection. ACM SIGOPS Operating Systems Review. 1974. Vol. 8, No. 1. P. 18–24.
24. Maffeis S., Mitchell J. C., Taly A. Object capabilities and isolation of untrusted web applications. Proceedings of the 2010 IEEE Symposium on Security and Privacy. 2010. P. 125–140.

25. Miller M. S. Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control. PhD Thesis. Johns Hopkins University, 2006. 328 p.
26. Miller M. S., Van Cutsem T., Tulloh B. Distributed Electronic Rights in JavaScript. Proceedings of ESOP 2013. 2013. Vol. 7792. P. 1–20.
27. Miller M. S., Tribble E. D., Shapiro J. Concurrency among strangers: Programming in E as plan coordination. Proceedings of the International Symposium on Trustworthy Global Computing. 2005. Vol. 3705. P. 195–229.
28. Saltzer J. H., Schroeder M. D. The protection of information in computer systems. Proceedings of the IEEE. 1975. Vol. 63, No. 9. P. 1278–1308.
29. Shapiro J. S., Smith J. M., Farber D. J. EROS: A fast capability system. Proceedings of the 17th ACM Symposium on Operating Systems Principles. 1999. P. 170–185.
30. Van Cutsem T., Miller M. S. Proxies: Design principles for robust object-oriented intercession APIs. Proceedings of the 6th Dynamic Languages Symposium. 2010. P. 59–72.
31. Van Cutsem T., Miller M. S. Trustworthy proxies: Virtualizing objects with invariants. Proceedings of ECOOP 2013. 2013. Vol. 7920. P. 154–178.
32. Agten P., Van Acker S., Brondsema Y., Phung P. H., Desmet L., Piessens F. JSand: complete client-side sandboxing of third-party JavaScript without browser modifications. Proceedings of the 28th Annual Computer Security Applications Conference. 2012. P. 1–10.
33. Barth A., Jackson C., Mitchell J. C. Securing frame communication in browsers. Communications of the ACM. 2009. Vol. 52, No. 6. P. 83–91.
34. Birgisson A., Hedin D., Sabelfeld A. Boosting the permissiveness of dynamic information-flow tracking by testing. Proceedings of the 18th European Symposium on Research in Computer Security. 2013. P. 55–72.
35. De Groef W., Devriese D., Nikiforakis N., Piessens F. FlowFox: a web browser with flexible and precise information flow control. Proceedings of the 2012 ACM Conference on Computer and Communications Security. 2012. P. 748–759.

36. Groef W., Massacci F., Piessens F. NodeSentry: least-privilege library integration for server-side JavaScript. Proceedings of the 30th Annual Computer Security Applications Conference. 2014. P. 446–455.
37. Hedin D., Birgisson A., Bello L., Sabelfeld A. JSFlow: Tracking information flow in JavaScript and its APIs. Proceedings of the 29th Annual ACM Symposium on Applied Computing. 2014. P. 1663–1671.
38. Hedin D., Sabelfeld A. Information-flow security for a core of JavaScript. Proceedings of the 2012 IEEE 25th Computer Security Foundations Symposium. 2012. P. 3–18.
39. Lekies S., Stock B., Johns M. 25 million flows later: large-scale detection of DOM-based XSS. Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security. 2013. P. 1193–1204.
40. Magazinius J., Phung P. H., Sands D. Safe wrappers and sane policies for self protecting JavaScript. Proceedings of the 15th Nordic Conference on Secure IT Systems. 2010. P. 239–255.
41. Magazinius J., Russo A., Sabelfeld A. On-the-fly inlining of dynamic security monitors. Computers & Security. 2012. Vol. 31, No. 7. P. 827–843.
42. Melicher W., Fahl S., Ur B. Riding out DOMsday: Towards detecting and preventing DOM cross-site scripting. Proceedings of the 2021 Network and Distributed System Security Symposium. 2021. P. 1–18.
43. Terzi A., Basagiannis S. Managing security vulnerabilities introduced by dependencies in React.js JavaScript framework. 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER-C). 2024. P. 1–6. DOI: 10.1109/SANER-C62648.2024.00022.
44. Parameshwaran I., Budianto E., Shinde S., Dang H., Sadhu A., Saxena P. Auto-patching DOM-based XSS at scale. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. 2015. P. 702–713.
45. ENISA. *Security Guidelines for Web Application Development.* European Union Agency for Cybersecurity, 2023. 94 p.

46. Schwenk J., Niemietz M., Mainka C. Same-origin policy: Evaluation in modern browsers. Proceedings of the 26th USENIX Security Symposium. 2017. P. 713–727.
47. Stock B., Lekies S., Mueller T., Spiegel P., Johns M. Precise client-side protection against DOM-based cross-site scripting. Proceedings of the 23rd USENIX Security Symposium. 2014. P. 655–670.
48. Weissbacher M., Lauinger T., Robertson W. Why is CSP failing? Trends and challenges in CSP adoption. Proceedings of the 17th International Symposium on Research in Attacks, Intrusions and Defenses. 2014. P. 212–233.
49. React Documentation. React 18 Overview. Meta Platforms, Inc. URL: https://react.dev/blog/2022/03/29/react-v18 (дата звернення: 20.11.2025).
50. React Documentation. Hooks API Reference. Meta Platforms, Inc. URL: https://react.dev/reference/react (дата звернення: 20.11.2025).
51. React Documentation. Lifecycle of Reactive Effects. Meta Platforms, Inc. URL: https://react.dev/learn/lifecycle-of-reactive-effects (дата звернення: 20.11.2025).
52. TypeScript Documentation. TypeScript 5.3 Release Notes. Microsoft Corporation. URL: https://www.typescriptlang.org/docs/handbook/release-notes/typescript-5-3.html (дата звернення: 20.11.2025).
53. TypeScript Documentation. TypeScript Handbook. Microsoft Corporation. URL: https://www.typescriptlang.org/docs/handbook/intro.html (дата звернення: 20.11.2025).
54. Vite Documentation. Getting Started. URL: https://vitejs.dev/guide/ (дата звернення: 21.11.2025).
55. W3C. Content Security Policy Level 3. W3C Working Draft. URL: https://www.w3.org/TR/CSP3/ (дата звернення: 16.11.2025).
56. W3C. Subresource Integrity. W3C Recommendation. URL: https://www.w3.org/TR/SRI/ (дата звернення: 16.11.2025).

57. Zustand Documentation. Bear necessities for state management in React. URL: https://docs.pmnd.rs/zustand/getting-started/introduction (дата звернення: 21.11.2025).
58. MDN Web Docs. Web Security. Mozilla Foundation. URL: https://developer.mozilla.org/en-US/docs/Web/Security (дата звернення: 18.11.2025).
59. MITRE ATT&CK. *Enterprise Matrix Techniques.* MITRE Corporation, 2024. URL: https://attack.mitre.org/matrices/enterprise/ (дата звернення: 22.11.2025)
60. Boruch-Gruszecki A., Askarov A., Hirschfeld R. Gradual compartmentalization via object capabilities. ACM Transactions on Programming Languages and Systems. 2024. Vol. 46, No. 3. URL: https://dl.acm.org/doi/10.1145/3689751 (дата звернення: 23.11.2025).

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ

Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
"____" _____ 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему:

Підвищення захищеності веб-додатків шляхом застосування моделі об'єктних можливостей в архітектурі SPA

08-72.МКР.001.00.099.ТЗ

Керівник магістерської кваліфікаційної роботи
к.т.н., завідувач кафедри МБІС, доцент В.В. Карпінець

Вінниця – 2025 р.

1. Найменування та область застосування

Підвищення захищеності веб-додатків шляхом застосування моделі об'єктних можливостей в архітектурі SPA. Область застосування: захист односторінкових веб-додатків (Single Page Applications), корпоративні інформаційні системи, державні цифрові сервіси, системи електронного документообігу, платформи електронної комерції та інші веб-додатки з підвищеними вимогами до безпеки доступу.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ №96 від 20. 03. 2025 р.

3. Мета та призначення розробки

3.1 Мета розробки: підвищення захищеності веб-додатків шляхом впровадження удосконаленої моделі об'єктних можливостей (OCAP), адаптованої до специфіки архітектури односторінкових додатків (SPA), з автоматичним управлінням життєвим циклом можливостей та каскадним відкликанням прав доступу.

3.2 Призначення: розроблений програмний засіб забезпечує безпечний контроль доступу до ресурсів у SPA-додатках через механізм непідробних токенів можливостей, автоматичне управління правами доступу відповідно до життєвого циклу React-компонентів, делегування обмежених прав між компонентами та каскадне відкликання можливостей, що гарантує високий рівень захисту від несанкціонованого доступу та ескалації привілеїв.

4. Джерела розробки

4.1. Miller M. S., Yee K.-P., Shapiro J. Capability Myths Demolished. Technical Report SRL2003-02. Johns Hopkins University Systems Research Laboratory. 2003. URL: http://srl.cs.jhu.edu/pubs/SRL2003-02.pdf

4.2. OWASP Top 10 - 2021: The Ten Most Critical Web Application Security Risks. OWASP Foundation. 2021. URL: https://owasp.org/Top10/

4.3. Agoric. Secure EcmaScript (SES). GitHub repository. 2023. URL: https://github.com/endojs/endo/tree/master/packages/ses

4.4. React Documentation. Managing State and Component Lifecycle. 2024. URL: https://react.dev/learn/managing-state

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, інтуїтивно зрозумілий інтерфейс користувача з візуалізацією стану безпеки системи;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програмний засіб повинен забезпечувати автоматичне створення, валідацію та відкликання токенів можливостей відповідно до життєвого циклу компонентів;

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Система повинна забезпечувати автоматичне очищення пам'яті від відкликаних можливостей для запобігання витоку ресурсів;

5.2.3 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Intel Core i5-8400 2.8GHz / 9MB або AMD Ryzen 5 2600 3.4GHz / 16MB і подібні до них;

– оперативна пам’ять – не менше 4 Gb;

– середовище функціонування – операційна система сімейство Windows, Linux;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Початок	Закінчення
1	Визначення напрямку магістерської роботи, формулювання теми	20.09.2025	28.09.2025
2	Аналіз предметної області обраної теми	29.09.2025	07.10.2025
3	Апробація отриманих результатів	08.10.2025	14.10.2025
4	Розробка алгоритму роботи	16.10.2025	24.10.2025
5	Написання магістерської роботи на основі розробленої теми	24.10.2025	15.11.2025
6	Розробка економічної частини	16.11.2025	19.11.2025
7	Передзахист магістерської кваліфікаційної роботи	25.11.2025	26.11.2025
8	Виправлення, уточнення, корегування магістерської кваліфікаційної роботи	25.11.2025	04.12.2025
9	Захист магістерської кваліфікаційної роботи	08.12.2025	12.12.2025

10. Порядок контролю та прийому

10.1 До приймання магістерської кваліфікаційної роботи надається:

– ПЗ до магістерської кваліфікаційної роботи;

– програмний додаток;

– презентація;

– відзив керівника роботи;

– відзив опонента

Технічне завдання до виконання прийняв _____ Аншук О.А.

Додаток Б. Лістинг програми

```
export interface CapabilityToken {
  /** Унікальний ідентифікатор можливості */
  id: string;

  /** Ідентифікатор ресурсу, до якого надається доступ */
  resourceId: string;

  /** Дозволені операції над ресурсом */
  allowedOperations: ResourceOperation[];

  /** Мітка часу створення можливості */
  createdAt: number;

  /** Мітка часу закінчення дії (опціонально) */
  expiresAt?: number;

  /** Прапорець відкликання можливості */
  revoked: boolean;

  /** Ідентифікатор батьківської можливості (для делегування) */
  parentId?: string;

  /** Метадані можливості */
  metadata?: T;
}

/**
 * Інтерфейс захищеного ресурсу
 */
export interface ProtectedResource {
  /** Унікальний ідентифікатор ресурсу */
  id: string;

  /** Тип ресурсу (document, api, storage тощо) */
  type: string;

  /** Список дозволених операцій для цього типу ресурсу */
  allowedOperations: ResourceOperation[];
}

/**
```

```
 * Опції створення можливості
 */
export interface CreateCapabilityOptions {
  /** Час життя можливості в мілісекундах */
  expiresIn?: number;

  /** Автоматичне відкликання при демонтуванні компонента */
  autoRevoke?: boolean;

  /** Додаткові метадані */
  metadata?: any;
}

/**
 * Опції делегування можливості
 */
export interface DelegateCapabilityOptions extends CreateCapabilityOptions {
  /** Обмеження операцій для делегованої можливості */
  restrictOperations?: ResourceOperation[];
}

/**
 * Результат виконання операції з можливістю
 */
export interface CapabilityExecutionResult {
  /** Успішність виконання */
  success: boolean;

  /** Результат виконання операції */
  data?: T;

  /** Повідомлення про помилку */
  error?: string;

  /** Мітка часу виконання */
  timestamp: number;
}
import type {
  CapabilityToken,
  ProtectedResource,
  ResourceOperation,
  CreateCapabilityOptions,
  DelegateCapabilityOptions,
```

```
  CapabilityExecutionResult
} from '../types';

/**
 * Менеджер управління можливостями (Capability Manager)
 * Реалізує модель об'єктних можливостей (OCAP) для SPA-додатків
 *
* Основні функції:
 * - Створення та валідація можливостей
 * - Делегування з обмеженням прав
 * - Автоматичне відкликання
 * - Аудит операцій
 */
export class CapabilityManager {
  private static instance: CapabilityManager;

 /** Сховище всіх можливостей системи */
import { useEffect, useRef, useState } from 'react';
import { CapabilityManager } from '../core/CapabilityManager';
import type { CapabilityToken, ResourceOperation } from '../types';

interface ResourceRequest {
  id: string;
  operations: ResourceOperation[];
}

/**
 * React хук для управління множиною можливостей
 * Корисний для компонентів, які працюють з декількома ресурсами
 *
* @param resources - Масив запитів на ресурси
 * @returns Мапа можливостей та допоміжні функції
 */
export function useCapabilities(resources: ResourceRequest[]) {
  const [capabilities, setCapabilities] = useState>(
    new Map()
  );
  const [errors, setErrors] = useState>(new Map());
  const managerRef = useRef(CapabilityManager.getInstance());
  const capabilityIdsRef = useRef>(new Set());

  useEffect(() => {
    const manager = managerRef.current;
    const newCapabilities = new Map();
```

```
  const newErrors = new Map();

  // Створення можливостей для всіх ресурсів
  resources.forEach(({ id, operations }) => {
    try {
      const cap = manager.createCapability(id, operations, { autoRevoke: true });
      newCapabilities.set(id, cap);
      capabilityIdsRef.current.add(cap.id);
    } catch (err) {
      const errorMessage = err instanceof Error ? err.message : 'Невідома помилка';
      newErrors.set(id, errorMessage);
      console.error(`Помилка створення можливості для ${id}:`, err);
    }
  });

  setCapabilities(newCapabilities);
  setErrors(newErrors);

  // Очищення при демонтуванні
  return () => {
    capabilityIdsRef.current.forEach(capId => {
      manager.revokeCapability(capId);
    });
    capabilityIdsRef.current.clear();
  };
}, [JSON.stringify(resources)]);

/**
 * Відкликання всіх можливостей
 */
const revokeAll = () => {
  capabilityIdsRef.current.forEach(capId => {
    managerRef.current.revokeCapability(capId);
  });

  setCapabilities(prev => {
    const updated = new Map(prev);
    updated.forEach((cap, key) => {
      updated.set(key, { ...cap, revoked: true });
    });
    return updated;
  });
};
```

```
 return {
   capabilities,
   errors,
   revokeAll,
   getCapability: (resourceI

 private capabilities: Map = new Map();

/** Реєстр захищених ресурсів */
 private resources: Map = new Map();

/** Зворотні виклики для відкликання можливостей */
 private revocationCallbacks: Map void>> = new Map();

/** Журнал аудиту операцій */
 private auditLog: Array<{
   timestamp: number;
   capabilityId: string;
   operation: string;
   status: 'success' | 'denied' | 'error';
   error?: string;
 }> = [];

 private constructor() {}

 /**
  * Отримання єдиного екземпляра менеджера (Singleton)
  */
 public static getInstance(): CapabilityManager {
   if (!CapabilityManager.instance) {
     CapabilityManager.instance = new CapabilityManager();
   }
   return CapabilityManager.instance;
 }

 /**
  * Реєстрація захищеного ресурсу в системі
  * Ресурс повинен бути зареєстрований перед створенням можливостей
  */
 public registerResource(resource: ProtectedResource): void {
   this.resources.set(resource.id, resource);
   console.log(`Ресурс зареєстровано: ${resource.id} (тип: ${resource.type})`);
 }
```

```
/**
 * Створення нової можливості для доступу до ресурсу
 *
* @param resourceId - Ідентифікатор ресурсу
 * @param operations - Дозволені операції
 * @param options - Додаткові опції (термін дії, метадані)
 * @returns Токен можливості
 */
public createCapability(
  resourceId: string,
  operations: ResourceOperation[],
  options: CreateCapabilityOptions = {}
): CapabilityToken {
  // Перевірка існування ресурсу
  const resource = this.resources.get(resourceId);
  if (!resource) {
    throw new Error(`Ресурс не знайдено: ${resourceId}`);
  }

  // Валідація операцій
  const invalidOps = operations.filter(
    op => !resource.allowedOperations.includes(op)
  );
  if (invalidOps.length > 0) {
    throw new Error(
      `Недозволені операції для ресурсу ${resourceId}: ${invalidOps.join(', ')}`
    );
  }

  // Генерація унікального ідентифікатора
  const capabilityId = this.generateSecureId();

 // Створення токена можливості
  const capability: CapabilityToken = {
    id: capabilityId,
    resourceId,
    allowedOperations: operations,
    createdAt: Date.now(),
    expiresAt: options.expiresIn
      ? Date.now() + options.expiresIn
      : undefined,
    revoked: false,
    metadata: options.metadata
```

```
  };

  // Збереження можливості
  this.capabilities.set(capabilityId, capability);

  console.log(
    `Можливість створено: ${capabilityId} для ресурсу ${resourceId} ` +
    `з операціями [${operations.join(', ')}]`
  );

  return capability;
}

/**
 * Валідація можливості перед виконанням операції
 *
* @param capabilityId - Ідентифікатор можливості
 * @param operation - Операція, яку потрібно виконати
 * @returns true, якщо можливість дійсна
 */
public validateCapability(
  capabilityId: string,
  operation: ResourceOperation
): boolean {
  const capability = this.capabilities.get(capabilityId);

  // Перевірка існування можливості
  if (!capability) {
    console.warn(`Можливість не знайдено: ${capabilityId}`);
    return false;
  }

  // Перевірка відкликання
  if (capability.revoked) {
    console.warn(`Можливість відкликано: ${capabilityId}`);
    return false;
  }

  // Перевірка терміну дії
  if (capability.expiresAt && Date.now() > capability.expiresAt) {
    console.warn(`Термін дії можливості закінчився: ${capabilityId}`);
    this.revokeCapability(capabilityId);
    return false;
```

```
  }

  // Перевірка дозволених операцій
  if (!capability.allowedOperations.includes(operation)) {
    console.warn(
      `Операція ${operation} не дозволена для можливості ${capabilityId}`
    );
    return false;
  }

  return true;
}

/**
 * Виконання операції з автоматичною валідацією можливості
 *
* @param capabilityId - Ідентифікатор можливості
 * @param operation - Операція для виконання
 * @param action - Функція, яка виконує операцію
 * @returns Результат виконання операції
 */
public async executeWithCapability(
  capabilityId: string,
  operation: ResourceOperation,
  action: () => Promise
): Promise> {
  const startTime = Date.now();

  try {
    // Валідація можливості
    if (!this.validateCapability(capabilityId, operation)) {
      const result: CapabilityExecutionResult = {
        success: false,
        error: 'Доступ заборонено: недійсна можливість',
        timestamp: Date.now()
      };

     this.logOperation(capabilityId, operation, 'denied');
      return result;
    }

    // Виконання операції
    const data = await action();
```

```
    const result: CapabilityExecutionResult = {
      success: true,
      data,
      timestamp: Date.now()
    };

    this.logOperation(capabilityId, operation, 'success');

    console.log(
      `Операція ${operation} виконана успішно ` +
      `(час: ${Date.now() - startTime}мс)`
    );

    return result;
  } catch (error) {
    const errorMessage = error instanceof Error
      ? error.message
      : 'Невідома помилка';

    const result: CapabilityExecutionResult = {
      success: false,
      error: errorMessage,
      timestamp: Date.now()
    };

    this.logOperation(capabilityId, operation, 'error', errorMessage);

    console.error(`Помилка виконання операції ${operation}:`, error);

    return result;
  }
}

/**
 * Делегування можливості з обмеженням прав
 * Створює нову можливість з меншими або рівними правами
 *
* @param parentCapabilityId - Ідентифікатор батьківської можливості
 * @param delegateTo - Ідентифікатор отримувача
 * @param restrictedOperations - Обмежений набір операцій
 * @param options - Додаткові опції
 * @returns Делегована можливість
```

```
 */
public delegateCapability(
  parentCapabilityId: string,
  delegateTo: string,
  restrictedOperations?: ResourceOperation[],
  options: DelegateCapabilityOptions = {}
): CapabilityToken {
  // Отримання батьківської можливості
  const parentCapability = this.capabilities.get(parentCapabilityId);
  if (!parentCapability) {
    throw new Error(`Батьківська можливість не знайдена: ${parentCapabilityId}`);
  }

  if (parentCapability.revoked) {
    throw new Error('Неможливо делегувати відкликану можливість');
  }

  // Визначення операцій для делегованої можливості
  let delegatedOperations: ResourceOperation[];

 if (restrictedOperations) {
    // Перевірка, що обмежені операції є підмножиною батьківських
    const invalidOps = restrictedOperations.filter(
      op => !parentCapability.allowedOperations.includes(op)
    );

   if (invalidOps.length > 0) {
      throw new Error(
        `Спроба делегувати операції, які не дозволені батьківською можливістю: ` +
        `${invalidOps.join(', ')}`
      );
    }

   delegatedOperations = restrictedOperations;
  } else {
    // Якщо обмеження не вказані, копіюємо всі операції батьківської можливості
    delegatedOperations = [...parentCapability.allowedOperations];
  }

  // Створення делегованої можливості
  const delegatedCapability = this.createCapability(
    parentCapability.resourceId,
    delegatedOperations,
    {
```

```
      ...options,
      metadata: {
        ...options.metadata,
        delegatedTo,
        delegatedBy: parentCapabilityId
      }
    }
  );

  // Встановлення зв'язку з батьківською можливістю
  delegatedCapability.parentId = parentCapabilityId;
  this.capabilities.set(delegatedCapability.id, delegatedCapability);

  // Реєстрація зворотного виклику для каскадного відкликання
  this.onRevocation(parentCapabilityId, () => {
    this.revokeCapability(delegatedCapability.id);
  });

  console.log(
    `Можливість делегована: ${parentCapabilityId} -> ${delegatedCapability.id} ` +
    `для ${delegateTo} з операціями [${delegatedOperations.join(', ')}]`
  );

  return delegatedCapability;
}

/**
 * Відкликання можливості з каскадним ефектом
 * Автоматично відкликає всі делеговані можливості
 *
* @param capabilityId - Ідентифікатор можливості для відкликання
 */
public revokeCapability(capabilityId: string): void {
  const capability = this.capabilities.get(capabilityId);

 if (!capability) {
    console.warn(`Спроба відкликати неіснуючу можливість: ${capabilityId}`);
    return;
  }

  if (capability.revoked) {
    console.warn(`Можливість вже відкликана: ${capabilityId}`);
    return;
```

```
  }

  // Відкликання можливості
  capability.revoked = true;
  this.capabilities.set(capabilityId, capability);

  // Виклик зареєстрованих callback-функцій
  const callbacks = this.revocationCallbacks.get(capabilityId);
  if (callbacks) {
    callbacks.forEach(callback => {
      try {
        callback();
      } catch (error) {
        console.error('Помилка виконання callback відкликання:', error);
      }
    });

    // Очищення callback-функцій
    this.revocationCallbacks.delete(capabilityId);
  }

  console.log(`Можливість відкликано: ${capabilityId}`);
}

/**
 * Реєстрація callback-функції для відкликання
 * Використовується для каскадного відкликання та очищення ресурсів
 *
* @param capabilityId - Ідентифікатор можливості
 * @param callback - Функція, яка буде викликана при відкликанні
 */
public onRevocation(capabilityId: string, callback: () => void): void {
  if (!this.revocationCallbacks.has(capabilityId)) {
    this.revocationCallbacks.set(capabilityId, new Set());
  }

 this.revocationCallbacks.get(capabilityId)!.add(callback);
}

/**
 * Видалення callback-функції відкликання
 */
public offRevocation(capabilityId: string, callback: () => void): void {
```

```
  const callbacks = this.revocationCallbacks.get(capabilityId);
  if (callbacks) {
    callbacks.delete(callback);
  }
}

/**
 * Отримання інформації про можливість
 */
public getCapability(capabilityId: string): CapabilityToken | undefined {
  return this.capabilities.get(capabilityId);
}

/**
 * Отримання журналу аудиту
 */
public getAuditLog() {
  return [...this.auditLog];
}

/**
 * Генерація криптографічно стійкого ідентифікатора
 */
private generateSecureId(): string {
  const array = new Uint8Array(32);
  crypto.getRandomValues(array);
  return Array.from(array, byte => byte.toString(16).padStart(2, '0')).join('');
}

/**
 * Логування операції в журнал аудиту
 */
private logOperation(
  capabilityId: string,
  operation: string,
  status: 'success' | 'denied' | 'error',
  error?: string
): void {
  this.auditLog.push({
    timestamp: Date.now(),
    capabilityId,
    operation,
    status,
    error
```

```
    });

    // Обмеження розміру журналу (зберігаємо останні 1000 записів)
    if (this.auditLog.length > 1000) {
      this.auditLog.shift();
    }
  }
}
import { useEffect, useRef, useState } from 'react';
import { CapabilityManager } from '../core/CapabilityManager';
import type {
 CapabilityToken,
 ResourceOperation,
 CreateCapabilityOptions,
  CapabilityExecutionResult
} from '../types';

/**
 * React хук для управління можливістю з автоматичним життєвим циклом
 * Автоматично створює можливість при монтуванні компонента
 * та відкликає її при демонтуванні
 *
* @param resourceId - Ідентифікатор ресурсу
 * @param operations - Необхідні операції
 * @param options - Опції створення можливості
 * @returns Об'єкт з можливістю та допоміжними функціями
 */
export function useCapability(
  resourceId: string,
  operations: ResourceOperation[],
  options: CreateCapabilityOptions = { autoRevoke: true }
) {
  const [capability, setCapability] = useState(null);
  const [error, setError] = useState(null);
  const managerRef = useRef(CapabilityManager.getInstance());
  const capabilityIdRef = useRef(null);

  useEffect(() => {
    const manager = managerRef.current;

    try {
      // Створення можливості
      const cap = manager.createCapability(resourceId, operations, options);
      setCapability(cap);
```

```
    capabilityIdRef.current = cap.id;
    setError(null);
  } catch (err) {
    const errorMessage = err instanceof Error ? err.message : 'Невідома помилка';
    setError(errorMessage);
    console.error('Помилка створення можливості:', err);
  }

  // Очищення при демонтуванні компонента
  return () => {
    if (options.autoRevoke && capabilityIdRef.current) {
      manager.revokeCapability(capabilityIdRef.current);
      console.log(`Можливість автоматично відкликана: ${capabilityIdRef.current}`);
    }
  };
}, [resourceId, operations.join(','), options.autoRevoke]);

/**
 * Ручне відкликання можливості
 */
const revoke = () => {
  if (capabilityIdRef.current) {
    managerRef.current.revokeCapability(capabilityIdRef.current);
    setCapability(prev => prev ? { ...prev, revoked: true } : null);
  }
};

/**
 * Виконання операції з автоматичною валідацією
 */
const execute = async (
  operation: ResourceOperation,
  action: () => Promise
): Promise> => {
  if (!capabilityIdRef.current) {
    return {
      success: false,
      error: 'Можливість не створена',
      timestamp: Date.now()
    };
  }

  return managerRef.current.executeWithCapability(
    capabilityIdRef.current,
```

```
      operation,
      action
    );
  };

  return {
    capability,
    error,
    revoke,
    execute,
    isValid: capability !== null && !capability.revoked
  };
}
```

Додаток В. Ілюстративний матеріал

Підвищення захищеності веб-додатків шляхом застосування моделі об'єктних можливостей в архітектурі SPA

Виконав: Анщук О. А., студент 2-го курсу, гр. КІТС-24м
Спеціальності 125 – Кібербезпека та захист інформації
Науковий керівник: Карпінець В. В., к.т.н., доц. каф. МБІС

Актуальність теми:

- **Сучасні веб-додатки, зокрема SPA, є критичною частиною цифрової інфраструктури бізнесу, соціальних мереж, фінансових систем та державних сервісів.**
- **Зростання складності SPA збільшує вразливості та ризики несанкціонованого доступу, підробки запитів та витоків даних.**
- **Традиційні моделі контролю доступу (RBAC/ABAC) не завжди забезпечують потрібну гнучкість та ізоляцію в динамічних SPA-середовищах.**

Мета:

- **Підвищення захищеності веб-додатків шляхом впровадження удосконаленої моделі об'єктних можливостей (OCAP), адаптованої до специфіки архітектури односторінкових додатків (SPA), з автоматичним управлінням життєвим циклом можливостей та каскадним відкликанням прав доступу**

Наукова новизна

Наукова новизна полягає в удосконаленні підходів до управління доступом у SPA-додатках шляхом адаптації моделі об'єктних можливостей (OCAP).

Це досягається завдяки:

1. Розробці нового архітектурного підходу, який інтегрує OCAP з автоматичним управлінням життєвим циклом можливостей React-компонентів.
2. Усуненню проблеми "висячих посилань" (dangling references) та витоку ресурсів, що є нововведенням, не реалізованим у традиційних OCAP-системах (Caja, SES).
3. Забезпеченню підвищення рівня ізоляції компонентів і зменшенню ризику несанкціонованого доступу.
4. Створення архітектурного захисту від вразливості "заплутаний заступник".

Таким чином, робота не просто застосовує OCAP, а концептуально вдосконалює його для ефективного використання у динамічному клієнтоорієнтованому середовищі SPA

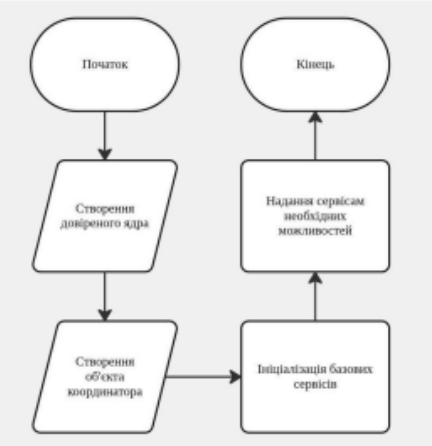
Вимоги до адаптованої моделі об'єктних можливостей

- **Автоматичне управління життєвим циклом можливостей.**
- **Делегування можливостей.**
- **Каскадне відкликання прав.**
- **Чітка ізоляція компонентів.**
- **Непідробність токенів.**
- **Модульність та масштабованість.**

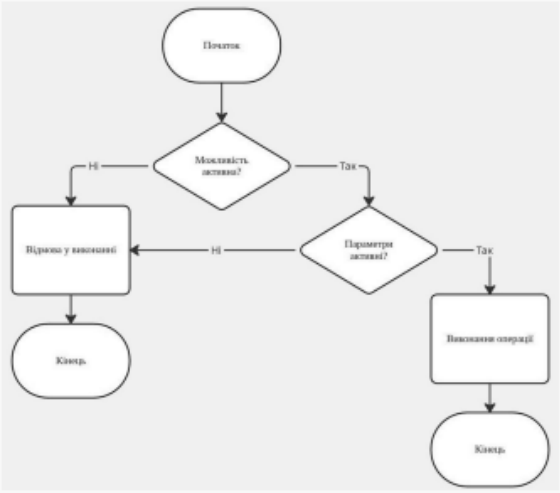
Порівняння реалізацій OCAP

Характеристика	Google Caja (2008-2014)	Secure ECMAScript (SES)	CapTP
Розробник	Google	Agoric	E language / Agoric blockchain
Період активності	2008-2014 (припинено)	Активний	Активний
Основна мета	Безпечне виконання JavaScript з моделлю на основі можливостей	Підмножина JavaScript з підтримкою моделі на основі можливостей	Протокол передачі можливостей між об'єктами в розподілених системах
Технічний підхід	Ізоляція непевного коду через перезапис об'єктів DOM та API браузера; техніка "object-capability subset" JavaScript	Замикання та Object.freeze(); "defensively consistent" об'єкти	Протокол для розподілених систем
Цільове середовище	Пісочниця для стороннього коду в браузері	Клієнтський JavaScript	Розподілені системи, блокчейн
Обмеження	Відсутність інтеграції з каркасами односторінкових застосунків; проект припинено у 2014 році	Ручне управління можливостями; відсутність інтеграції з життєвим циклом компонентів	Орієнтація на розподілені системи; складність впровадження у браузері

Алгоритми роботи адаптованої моделі OCAP

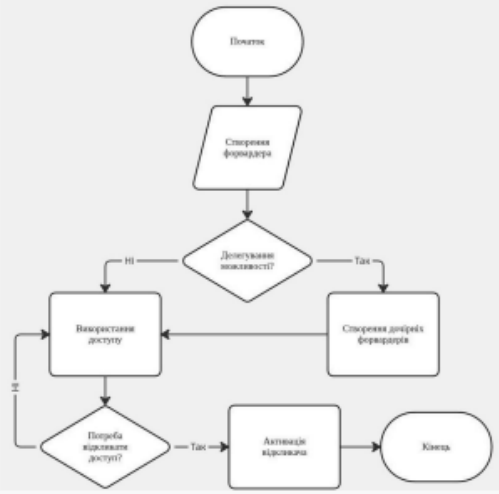


Узагальнений алгоритм побудови адаптованої моделі OCAP

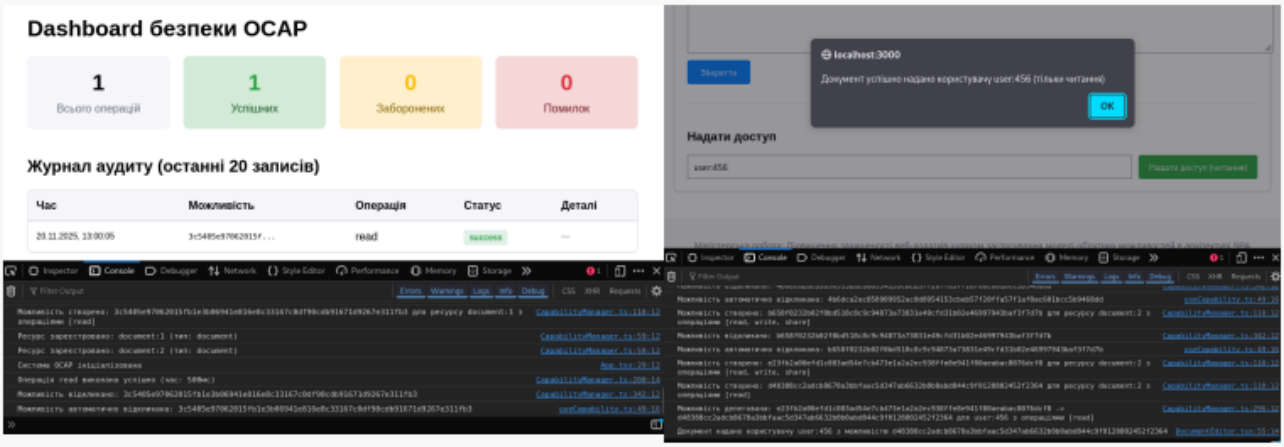


Ініціалізації системи розподілення прав доступу у адптованій моделі OCAP

Викликання можливостей у адаптованій системі OCAP



Інтерфейс розробленого веб-додатку



Тестування адаптованої моделі OCAP

Кількість активних компонентів	Час валідації доступу (мс)	Використання пам'яті (МВ)	Час створення токена (мс)	Час автовідкликання (мс)
10	0.28	2.4	0.7	0.3
50	1.42	12.1	3.5	1.6
100	2.95	24.3	7.2	3.3
200	6.21	48.8	15.1	6.8
500	16.84	122.4	38.9	17.5

Ефективність каскадного відкликання при різній глибині делегування

Глибина делегування	Кількість відкликаних можливостей	Час виконання відкликання (мс)	Успішність каскаду (%)
1	1	0.12	100
2	3	0.31	100
3	7	0.68	100
5	31	2.14	100
10	1023	18.47	100

Висновки

Головна мета магістерської роботи була досягнута: підвищено захищеність односторінкових веб-додатків (SPA) шляхом розробки та впровадження удосконаленої моделі об'єктних можливостей (OCAP), адаптованої до специфіки архітектури SPA.

Тестування підтвердило високу ефективність розробленої моделі у запобіганні несанкціонованому доступу та ескалації привілеїв. Було досягнуто значного підвищення продуктивності: час валідації доступу зменшено на 46–52% порівняно з традиційними підходами. Також реалізовано безпечне делегування та механізм каскадного відкликання прав.

Додаток Г. Протокол перевірки на антиплагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Підвищення захищеності веб-додатків шляхом застосування моделі об'єктних можливостей в архітектурі SPA

Тип роботи: магістерська кваліфікаційна робота

Підрозділ: кафедра менеджменту та безпеки інформаційних систем
факультет менеджменту та інформаційної безпеки
гр.1КІТС-24м

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) 0,92 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

к.т.н., доцент, зав. каф. МБІС Карпінець В.В. _____

к.ф.-м.н., доцент каф. МБІС Шиян А.А. _____

Особа, відповідальна за перевірку Коваль Н.П. _____

З висновком експертної комісії ознайомлений(-на)

Керівник _____ доц. Карпінець В.В.

Здобувач _____ Анщук О.А.