ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ

Пояснювальна записка

до магістерської роботи
на тему:

«ТЕХНОЛОГІЯ УПРАВЛІННЯ СЕРЕДОВИЩЕМ КОРИСТУВАЧІВ ВЕБ-ПОРТАЛУ ОРГАНІЗАЦІЇ»

Виконав студент 6 курсу, групи БСДМ-63
спеціальності 125 Кібербезпека
освітньо-професійної програми «Інформаційна та кібернетична безпека»
(шифр і назва спеціальності)
Єльський К.В.
(прізвище та ініціали)
Керівник Гайдур Г.І.
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)
Нормоконтролер Чумак Н.С.
(прізвище та ініціали)

КИЇВ – 2023

ЗМІСТ

Стор.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ…….. 10

ВСТУП ……..……. 11

1 АНАЛІЗ ПРОБЛЕМИ ЗАБЕЗПЕЧЕННЯ КІБЕРБЕЗПЕКИ У СЕРЕДОВИЩАХ УПРАВЛІННЯ КОРИСТУВАЧАМИ ОРГАНІЗАЦІЇ……. 13

1.1. Призначення, структура, функції та умови функціонування середовища користувачів веб-порталу організації……. 13
1.2. Аналіз проблеми забезпечення захисту середовища користувачів веб-порталу організації…….……. 20
1.3. Мета та завдання управління захисту середовища користувачів веб-порталу організації…….. 29
1.4. Аналіз існуючих технологій управління захисту середовища користувачів веб-порталу організації……... 37

2 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ УПРАВЛІННЯ ЗАХИСТОМ СЕРЕДОВИЩА КОРИСТУВАЧІВ ВЕБ-ПОРТАЛУ ОРГАНІЗАЦІЇ……..……..……. 42

2.1. Призначення та функції протоколу OAuth 2.0, JWT та OpenID… 42
2.2. Компоненти та архітектура рішення OAuth 2.0, JWT та OpenID..…… 47
2.3. Можливості щодо управління захистом середовища користувачів веб-порталу організації на базі OAuth 2.0, JWT та OpenID ……. 53

3 РОЗРОБЛЕННЯ ВАРІАНТА ЗАХИСТУ ТЕХНОЛОГІЇ УПРАВЛІННЯ СЕРЕДОВИЩЕМ КОРИСТУВАЧІВ ВЕБ-ПОРТАЛУ ОРГАНІЗАЦІЇ…… 63

3.1. Розроблення варіанта створення та управління, для захищеної системи управління користувачів веб-порталу організації ……. 63
3.2. Технологія управління та розробки захищеної системи управління користувачами веб-порталу організації…… 69
3.3. Розроблення рекомендацій щодо застосування, управління та розробки захищеної системи управління користувачами веб-порталу організації……..…….. 75

ВИСНОВКИ……. 77

ПЕРЕЛІК ПОСИЛАНЬ..... 79

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)..... 81

ВІДГУК РЕЦЕНЗЕНТА
на магістерську роботу

студента Єльського Костянтина Вячеславовича
на тему: «Технологія управління середовищем користувачів веб-порталу організації»

Актуальність:
Управління середовищем адміністрування корпоративної інформаційної системи стає все більш важливою темою. Щодня створюються веб-сайти, веб-портали та інші веб-ресурси. Вони всі потребують систему управління контентом веб-ресурсу, а також середовище управління користувачами. Уніфіковане управління внутрішнім середовищем веб-порталу є комплексним підходом для створення правильної моделі адміністрування всього веб-порталу. Починаючи з публікації різного контенту та завершуючи управлінням, спілкуванням із користувачами веб-ресурсу. Реалізація технології захисту та управління внутрішнім середовищем веб-порталу, створює умови забезпечення багатьох сучасних функцій захисту. Тому тема магістерської роботи є актуальною та своєчасною.

Позитивні сторони:
1. На основі проведеного аналізу, в роботі встановлено зміст проблеми забезпечення захисту веб-порталів та внутрішніх систем керуванням веб-порталами від новітніх загроз, визначена мета та завдання управління захистом корпоративної інформаційної системи.
2. Досліджено методи та засоби управління захистом корпоративної інформаційної системи з використанням технологій, та протоколів OAuth 2.0, JWT та OpenID.
3. Запропоновано варіант технології застосування системи управління захистом кінцевих точок корпоративної інформаційної системи з використанням технологій, та протоколів OAuth 2.0, JWT та OpenID та рекомендації щодо їх застосування.
4. Текст викладено достатньо грамотно, послідовно. Сформульовано чіткі та змістовні висновки. Графічний матеріал оформлено якісно. Список науково-технічної літератури свідчить про вміння користуватись матеріалами за темою магістерської роботи.

Недоліки:
1. У магістерській роботі бажано було б провести саме порівняльний аналіз рішень захисту корпоративної інформаційної системи.
2. Запропонований варіант розробленої системи управління захистом кінцевих точок корпоративної інформаційної системи на базі технологій, та протоколів OAuth 2.0, JWT та OpenID доцільно було б показати на прикладі конкретної організації.

Висновок: Враховуючи недоліки, магістерська робота заслуговує оцінку «**відмінно»**, а студент **Єльський Костянтин Вячеславович** - присвоєння кваліфікації магістр з кібербезпеки за спеціалізацією інформаційна та кібернетична безпека..

Якість роботи	
Виконано на замовлення підприємства	
Виконано за тематикою НДР	
Виконано з макетом	
Виконано з застосуванням ЕОМ та МПТ	√
Має практичну цінність	√
Проект-частина комплексної теми	

Підпис рецензента (_____)

_____

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОС – операційна система

ІБ – інформаційна безпека

ПК – персональний комп’ютер

ІТ – інформаційні технології

ОС – операційна система

БД – база даних

ІС – інформаційна система

ПЗ – програмне забезпечення

ТЗ – технічне завдання

БД – база даних

CMS (Content Management System) – система керування контентом, набір модулів для створення, редагування і управління контентом сайту

VCS – Version Control System

IDE – Integrated Development Environment

CI/CD – Continuous Integration/Continuous Delivery

DDOS – Distributed Denial of Service ("відмова в обслуговуванні")

SPA – Single Page Application

ВСТУП

Актуальність дослідження. На сьогоднішній день дуже велика кількість компаній, організацій, державних та приватних установ мають бажання створювати, розвивати власні веб-портали, веб-сайти та інші веб-ресурси. Це бажання надає велику кількість позитивних аспектів, таких як: швидкість отримання інформації, ознайомлення з послугами, сервісами компаній чи державних установ, цифровізація та інше. Водночас з процесом цифровізації виникають проблеми із забезпеченням безпеки веб-порталів та середовищ управління користувачами.

Створення організацією веб-порталів та інших схожих веб-ресурсів призводить також до проблем із забезпеченням захисту середовищ керуванням користувачами, контентом та інформацією. Незважаючи на це, правильно захищений веб-портал надає можливість для спрощення робочих процесів і їх оптимізації, приводячи до значного збільшення продуктивності і ефективності праці організації, підвищуючи конкурентні переваги даної організації.

Вищенаведені аргументи актуалізують тему даної магістерської роботи, зміст якої становлять дослідження щодо технології управління середовищем користувачів веб-порталу організації.

Об'єкт дослідження – процес управління середовищем користувачів веб-порталу організації.

Предмет дослідження – технології управління середовищем користувачів веб-порталу організації.

Мета роботи – розробити варіант поетапної розробки, захисту та технічного супроводження захищеного веб-порталу та середовища управління користувачів веб-порталу організації.

Наукові завдання:

дослідити сутність проблеми забезпечення захисту управління середовищем користувачів веб-порталу організації;

встановити сутність завдань захисту та розробки захищеного середовища

управління користувачами веб-порталу організації;

проаналізувати існуючі технології захисту та розробки захищеного середовища управління користувачами веб-порталу організації;

проаналізувати методи та засоби забезпечення захисту та розробки захищеного середовища управління користувачами веб-порталу організації;

проаналізувати основні функції та принципи реалізації захисту середовища управління користувачами веб-порталу організації.

Методи дослідження – опрацювання літератури за даною темою, аналіз процесів розробки веб-порталу, аналіз прикладів створення захищених систем управління та розробки веб-порталу, аналіз експлуатаційної документації, міжнародних стандартів та їх порівняння, моделювання топології системи управління середовищем користувачів веб-порталу організації.

Практичне значення одержаних результатів полягає в розробці варіанта топології поетапної розробки, захисту та технічного супроводження захищеного веб-порталу та середовища управління користувачів веб-порталу організації, а також у розробці рекомендацій щодо застосування технології захисту веб-порталу організації.

Апробація результатів магістерської роботи було оприлюднено на Всеукраїнській науковій конференції «Актуальні проблеми кібербезпеки», яка відбулася 27 жовтня 2022 року в Державному університеті телекомунікацій, м. Київ.

1 АНАЛІЗ ПРОБЛЕМИ ЗАБЕЗПЕЧЕННЯ КІБЕРБЕЗПЕКИ У СЕРЕДОВИЩАХ УПРАВЛІННЯ КОРИСТУВАЧАМИ ОРГАНІЗАЦІЇ

1.1. Призначення, структура, функції та умови функціонування середовища користувачів веб-порталу організації

Веб-портал — вебсайт, що надає користувачеві Інтернету різні інтерактивні сервіси (Інтернет-сервіси). Портали працюють як точки доступу до інформації у Інтернеті або як сайти, що допомагають користувачам у пошуку потрібної інформації. Такі портали представляють інформацію з різних джерел або тем об'єднаним способом їх також називають навігаційними сайтами. Всі портали виконують функції пошуку, а також надають Інтернет-сервіси, наприклад: календар подій, стрічка новин, медіа галерея тощо. Ідея роботи порталу — створення або представлення критичної (найбільшої) маси Інтернет-сервісів, якими б можна було залучити до себе таку кількість користувачів-відвідувачів, яка буде постійно поповнюватися та збільшуватися. Більшість сучасних інформаційних систем створюються у вигляді веб-сайтів, тому безпеці необхідно приділити особливу увагу. Як тільки з'являється пристрій з операційною системою і програмним додатком, він відразу ж привертає інтерес зловмисників. Вони починають його вивчати і зламувати. В останній рік цієї тенденції було присвячено багато доповідей на великих хакерських конференціях. Розробники не завжди приділяють особливу увагу повному захисту та безпеці створених сайтів, хоча вона і відіграє дуже важливу роль, тому що кожен сайт так чи інакше може бути підданий злому хакерів, якщо враховувати те, що сайти можуть бути різноманітними: електронні бібліотеки, соціальні мережі, веб-портали міст, веб-портали різних навчальних закладів, онлайн-магазини, офіційні сайти великих організацій, банків та інше. В першу чергу, необхідно звернути увагу на безпеку веб-сервера, так як саме він відповідає за прийом і обробку HTTP-запитів від

клієнтів до веб-сайту. Саме він забезпечує функціонування численних веб-сайтів по всьому світу, а так само відповідає за базові послуги і зберігає персональні дані, як користувачів, так і відвідувачів сайту. Необхідність захисту серверів - це одна з найважливіших завдань для будь-якої організації. [1]

Веб-портали можуть бути декількох видів:

- тематичні - присвячені конкретній темі, а отже, залучають зацікавлену в ній аудиторію (фінансовий, державний, технічний, медичний тощо);
- інформаційні - новинні ресурси, ЗМІ;
- навчальні - проекти навчальних закладів, онлайн-курсів;
- корпоративні - ресурси, що об'єднують співробітників організації онлайн і надають доступ до корпоративних даних.

Сайти, своєю чергою, включають блоги, візитки, лендінги, портфоліо, промо-сайти, прості інтернет-магазини або сайти-вітрини. Вони не вирізняються широким функціоналом, а тому до них застосовуються інші методи веб-розробки та програмування. [17]

Програмування порталу включає в себе кілька етапів:

- розробка прототипу;
- створення функціональної складової (сервісів, блоків управління, скриптів, розмежування доступів та взаємодії з базами даних);
- тестування;
- налаштування взаємодії систем між собою.

Оскільки веб-портали розробляються під більш високе навантаження і з великою кількістю інтерактивних сервісів, для їхнього програмування використовуються:

- Редактори коду - доволі прості програми, що застосовуються для створення та зміни файлів коду;
- Багатофункціональні інтегровані середовища (IDE) (Наприклад IDE від JetBrains) - більший інструментарій і широкий прикладний функціонал;
- Хмарні IDE (Cloud9, Eclipse Che, Orion) - функціонал програмних додатків для веб-розробки, реалізований у вигляді інтернет-сервісу.

- Сервіси для контролю та управління версіями, процесу деплою, веденням та керуванням задач для розробки. Найбільш популярними та зручними рішеннями є: GitLab, GitHub та BitBucket.

Прототипування порталу це один із основних етапів на шляху до початку розробки веб-порталів і візуалізації кожного проекту та системи управління користувачами веб-порталу.

Прототип - схема майбутнього ресурсу або його окремих сторінок. Прототипування, зі свого боку, передбачає процес створення схеми.

Добре продуманий прототип стане міцним фундаментом для наступних етапів розробки веб-порталу організації. Крім того, витрачений на опрацювання схеми порталу час обов'язково заощадить години в майбутньому. Важливість прототипування не тільки в тому, що ви побачите потенційний образ майбутнього веб-ресурсу, а й зможете обміркувати розташування елементів на кожній сторінці та їхню взаємодію, а також внести правки в схему ще до початку створення порталу. В іншому разі, якщо на етапі розробки виявиться, що деякі елементи працюють не так, як потрібно, переробляти доведеться в рази більше, а на це знадобляться додатковий час і фінансові витрати. У проекту будь-якого розміру повинен бути прототип, тому важливо лише вибрати сервіс завдяки якому можна створити детальний прототип.

За глибиною опрацювання виділяють:

- Ескіз (низька деталізація) - підходить для односторінкових сайтів. Намалювати його можна на папері або в найпростішій програмі для створення схем. Такий прототип не підходить для реалізації масштабних проектів.
- Висока деталізація - для багатосторінкових сайтів і об’ємних веб-порталів організацій. Створюється за допомогою спеціальних сервісів.

За місцем зберігання:

- аналоговий (малюнок);
- цифровий (створений у програмі, з доступ до редагування тільки у розробника);
- хмарний (прототип перебуває в хмарі, тому доступ до редагування

мають кілька учасників проекту).

За можливістю взаємодії:

- статичний - графічне зображення;
- інтерактивний - включає елементи взаємодії, такі як клікабельні посилання, слайдери, спливаючі форми тощо.

Рис. 1.1. Тестування веб-ресурсів

Тестування програмного коду вкрай важливий етап для будь-якого інтернет-ресурсу. Тестування - ретельна перевірка сайту на предмет потенційних помилок. Воно проводиться як у процесі розробки порталу, так і перед самим запуском. Під час перевірки виявляються не тільки помилки і баги, а й оцінюється привабливість сервісу для користувачів, зручність і безпека використання. Крім того, тестування

визначає, чи досягаються поставлені задачі, які були поставленні на початку роботи над створення веб-порталу для організації.

Можна виділити декілька основних типів тестування веб-порталу та системи управління користувачами:

- Функціональне тестування перевіряє роботу кожної функції веб-порталу, його можливість вирішити поставлені користувачем завдання. Його сенс полягає у тому, щоб встановити відповідність ресурсу вихідним функціональним вимогам клієнта. Включає в себе перевірку роботи посилань, користувацьких форм, перевірку HTML/CSS/JS-кодів, перевірку процесів і негативних сценаріїв (наприклад, виведення повідомлень про помилку при введенні некоректних даних).
- Тестування зручності використання. Виявляє характер взаємодії людини з порталом. Наголос робиться на простоту навігації та суб'єктивне задоволення відвідувача від використання сервісу.
- UI-тестування (інтерфейсу користувача). Перевірка графічного інтерфейсу користувача. Допомагає зрозуміти, чи зручно натискати на кнопки, чи зрозумілі іконки, визначає, наскільки читабельним є обраний формат тексту і шрифти. Також допомагає визначити, чи правильно розташовані акценти і до чого варто привернути увагу. Крім того, UI-тестування визначає, який вигляд сайт матиме в інших браузерах або на інших девайсах. Веб-портал може виглядати по-різному на Android чи iOS девайсах, функціональність може відрізнятися на Windows, macOS та Linux операційних системах. Усі процеси та етапи роботи веб-порталу повинні бути ретельно перевірені тестувальником.
- Тестування продуктивності. Веб-портали створюються під велике навантаження, тому тестування продуктивності обов'язково включає:
    - навантажувальне тестування, що перевіряє, скільки одночасних користувацьких запитів витримає сайт або сторінка;
    - стрес-тестування - оцінка роботи сайту в екстремальних умовах;
    - тестування на різній швидкості інтернет-з'єднання, яке перевіряє, як швидко відгукується портал у різних умовах.
    - Тестування безпеки. Необхідно для виявлення потенційних

вразливостей, слабких місць і лазівок, які можуть призвести до втрати даних, доходу або репутації організації.

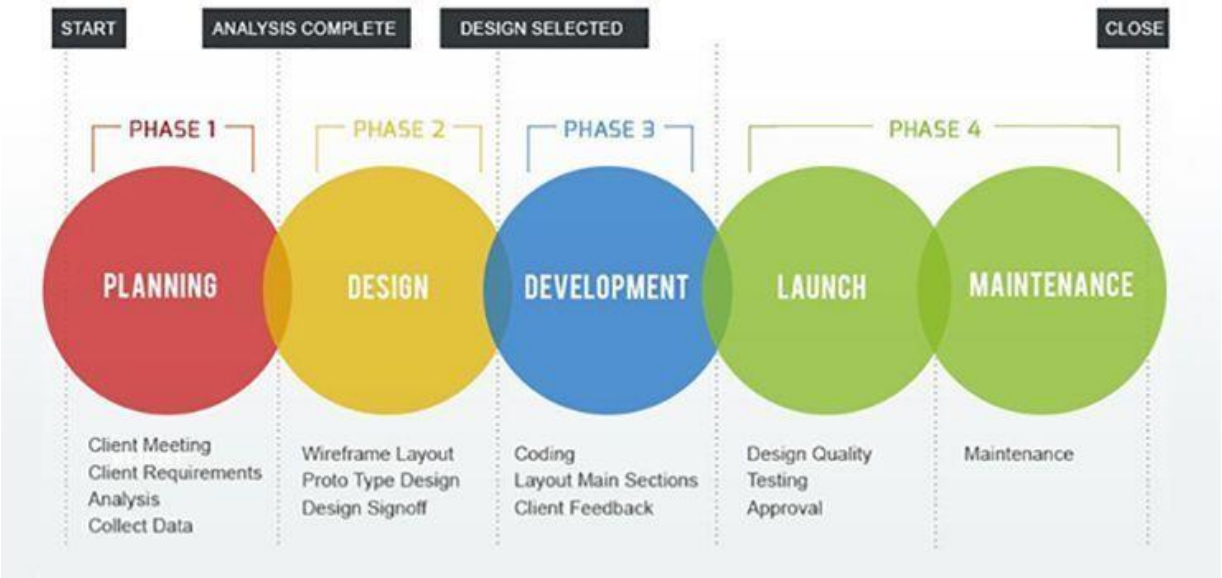

Рис. 1.2. Базові кроки створення веб-ресурсу.

Виходячи із вищенаведених прикладів, хотів би зазначити велику цінність таких продуктів-сервісів як GitLab та GitHub. Ці два сервісі є невід'ємною частиною під час планування та при розробці веб-порталу, а також під час впровадження нових рішень та підтримки написаного коду.

Git - це система контролю версій, яка застосовується для відстеження змін у файлах комп'ютера. Його основне завдання - керувати будь-якими змінами, внесеними в один або кілька проектів за певний період часу. Git допомагає координувати роботу учасників проектної групи та відстежувати прогрес у часі, а також дає їм змогу працювати спільно, не впливаючи один на одного. Також система дає можливість локально зберегти проект у файл невеликого розміру.

GitHub і GitLab - це онлайн-сервіси, які служать для розміщення репозиторіїв, дистанційного керування та інших завдань, пов'язаних із розробкою. У них є функціонал відстеження помилок, Wiki для всіх проектів, історія змін і багато іншого.

Сервіси мають схоже призначення, тому і функції у них приблизно однакові.

Однак, є й відмінності.

- У GitLab передбачена безкоштовна безперервна інтеграція. У GitHub замість цього є інструмент Actions, який дає змогу запускати подібні інтеграції в загальнодоступних репозиторіях без оплати, а у приватних репозиторіях - платно.
- GitLab, на відміну від GitHub, має вбудовану платформу розгортання Kubernetes.
- У GitLab доступні безкоштовні репозиторії для проектів із загальнодоступним вихідним кодом. У GitHub такого функціоналу не передбачено.

Як йшлося вище, GitLab є сервісом для спільної розробки. Він допомагає командам розробників скоротити життєвий цикл продукту і підвищити продуктивність. Тут не потрібна авторизація для кожного окремого інструменту. Дозволи встановлюються та налаштовуються один раз, і всі залучені у процес розробки працівники мають доступ до кожного компонента.

Спочатку GitLab був повністю безкоштовним програмним забезпеченням з відкритим вихідним кодом, який поширювався за ліцензією MIT. У 2013 році його розбили на дві окремі версії - GitLab CE (Community Edition) і GitLab EE (Enterprise Edition). GitLab CE залишилася колишньою, а GitLab EE отримав обмежену ліцензію, хоча вихідний код так і залишився загальнодоступним.

GitLab дає нижчеописані можливості:

- Створення. Написання коду, управління ним і внесення правок, синхронізація файлів.
- Перевірка. Включені інструменти CI/CD, доступна інтеграція з GitHub, є перевірка якості написання коду, навантажувальні тести.
- Управління. Для доступу до проекту потрібно пройти двофакторну аутентифікацію. Також доступна підтримка токенів. Сервіс здійснює аналітику продуктивності учасників, надає трекінг завдань, збирає та зберігає усі зміни у коді та завдань проекту.
- Моніторинг. Відстеження продуктивності додатків, управління інцидентами, надсилання логів.
- Планування. Відстеження проблем і контроль витраченого часу на

вирішення задач або впровадження нової функціональності для веб-порталу. Можна використовувати estimates, для того, щоб менеджер проекту мав приблизне уявлення по те, за скільки часу розробник або тестувальник вирішать проблему та задачу.

- Пакування. Управління контейнерами та репозиторіями. Робота GitLab з Docker.
- Безпека. Сканування вразливостей, підтримка SAST і DAST.
- Релізи. Є можливість запускати CI/CD у різних оточеннях, доступні канаркові релізи (безпечне тестування коду або частковий випуск), оркестровка релізів.
- Конфігурування. Підтримка Kubernetes, безсерверні обчислення.
- Захист. Забезпечення мережевої безпеки проектів.

Основна перевага використання GitLab у тому, що він дає змогу всім членам команди взаємодіяти на кожному етапі проекту. Від відстеження процесу планування, до створення проекту. Також GitLab дає змогу розробникам та менеджерам проектів автоматизувати весь життєвий цикл проекту і добитися найкращих результатів.

1.2. Аналіз проблеми забезпечення захисту середовища користувачів веб-порталу організації

Забезпечити безпеку веб-порталу організації та системи управління користувачами – основне завдання не тільки системного адміністратора або людини, що виконує його роль, але і безпосереднього організації, яка є власником інтернет-ресурсу.

Одним із основних етапів створення веб-порталу та системи керуванням користувачами є налаштування та інтеграція із різноманітними модулями які будуть задовільняти потреби організації (власника) веб-порталу, ієрархію користувачів, а також інтегрувати його зі сторонніми сервісами.

Веб-портал – це достатньо великий проект із великою кількістю інформації, яка зберігається на різних модулях, тому важливо розмежувати доступ усіх основних користувачів. Так, пересічним користувачам та відвідувачам можна надати можливість коментувати публікації, переглядати контент, та взаємодіяти із контентом оприлюдненим на веб-порталі. Тоді як адміністрація веб-порталу може як публікувати контент, так і коментувати його, редагувати, видаляти, впроваджувати різноманітні модулі для підвищення якості веб-порталу організації, у такий спосіб комунікувати із відвідувачами порталу.

Найпоширенішими є такі ролі:

- адміністратори та модератори - мають найвищий доступ;
- редактори та контент-менеджери - керують записами, публікують і редагують їх, часто займаються і модерацією коментарів;
- автори, користувачі - можуть створювати свої публікації та керувати ними;
- гість - може переглядати контент, у деяких випадках - коментувати.
- можливості кожного рівня користувача потрібно враховувати під час розроблення веб порталу.

Крім створення ієрархії доступу, потрібно приділити увагу інтеграції ресурсу зі сторонніми сервісами. Хороший проект не може бути ізольованим: зовнішня інтеграція дає змогу вирішити багато важливих бізнес-завдань і підвищити юзабіліті сайту.

Можна виділити такі найпопулярніші системи для інтеграції:

- e-mail - комунікація між користувачем і адміністрацією за допомогою електронної пошти;
- SMS - у вигляді такого повідомлення користувач може отримати логін і пароль для входу на сайт, або тільки тимчасовий пароль, код підтвердження оплати, трек-номер та інформацію про стан доставки тощо;
- Інтеграція з сервісами доставки товарів - вибір адреси або відділення для доставки та розрахунок її вартості, прямо на веб-порталі;
- платіжні системи - оплата послуг і покупок онлайн;

- системи обліку товарів, бухгалтерський облік - інтеграція порталу зі сховищем даних;
- CRM - система управління відносинами з клієнтом (збір даних про клієнтів, управління угодами, аналітика, програмування).

Портал можна інтегрувати і з іншими типами ПЗ, яке розробляється індивідуально під окремі проекти або потреби.

Безпека сайту вимагає пильності у всіх аспектах дизайну та використання сайту. Інтернет є досить небезпечним місцем. Ми регулярно чуємо про те, що веб-сайти стають недоступними через атаки на кшталт «відмовлено в обслуговуванні», або відображення зміненої (і часто пошкодженої) інформації на їхніх сторінках. В інших випадках мільйони паролів, адрес електронної пошти та дані кредитних карток ставали загальнодоступними, наражаючи користувачів веб-сайту на небезпеку або на фінансові ризики.

Мета веб-безпеки полягає в запобіганні цим (або іншим) видам атак. Більш формальним визначенням веб-безпеки є: способи захисту веб-сайтів від несанкціонованого доступу, використання, зміни, знищення або порушення роботи.

Для ефективної безпеки веб-сайту необхідно приділяти особливу увагу до розроблення всього веб-сайту: до вашого веб-додатку, конфігурації веб-сервера, під час написання політик створення та оновлення паролів, а також коду на стороні клієнта. Хоча все це звучить дуже зловісно, хороша новина полягає в тому, що якщо ви використовуєте веб-фреймворк для серверної частини, то він майже напевно забезпечить "за замовчуванням" надійні та продумані механізми захисту від низки найпоширеніших атак. Інші атаки можна пом'якшити за допомогою конфігурації вашого веб-сервера, наприклад, увімкнувши HTTPS. Нарешті, є загальнодоступні інструменти для сканування вразливостей, які можуть допомогти вам визначити, якщо ви припустилися якихось очевидних помилок.

Потрібно звернути увагу на те, наскільки успішними є загрози сайту, коли веб-додаток довіряє або недостатньо «параноїдально» ставиться до даних, що надходять із браузера.

XSS (Cross-Site Scripting - Міжсайтовий скриптинг) - це термін, який використовується для опису типу атак, що дають змогу зловмиснику впроваджувати шкідливий код через веб-сайт у браузери інших користувачів. Оскільки впроваджений код надходить у браузер із сайту, він є довіреним і може виконувати такі дії, як надсилання авторизаційного файлу cookie користувача зловмиснику. Коли у зловмисника є файл cookie, він може увійти на сайт, наче він був користувачем, і зробити все, що може користувач, наприклад, отримати доступ до даних кредитної картки, переглянути контактні дані або змінити паролі.

Уразливості XSS поділяються на відображені та збережені, залежно від того, як сайт повертає впроваджений код у браузер.

Уразливості XSS історично зустрічалися частіше, ніж будь-які інші види загроз безпеки.

- Відображена XSS-вразливість виникає, коли призначений для користувача контент, який передається на сервер, негайно і без змін повертається для відображення в браузері. Будь-який скрипт у вихідному користувацькому контенті запуститься під час завантаження нової сторінки. Наприклад, розглянемо рядок пошуку на сайті, в якому пошукові слова закодовані як параметри URL, і ці слова відображаються разом із результатами. Зловмисник може створити пошукове посилання, яке міститиме шкідливий скрипт як параметр (наприклад: http://mysite.com?q=javascript<script%20src="http://testwebsite.com/testjsfile.js"></script>) і переслати його іншому користувачеві електронною поштою. Якщо цільовий користувач клацне по цьому "цікавому посиланню", то скрипт виконається під час відображення результатів пошуку. Як ми вже говорили, зловмисник у такий спосіб отримує всю інформацію, необхідну йому для входу на сайт як цільовий користувач, потенційного здійснення покупок від імені користувача або отримання його контактної інформації.

- Постійна вразливість XSS виникає, коли шкідливий скрипт зберігається на веб-сайті, а потім знову відображається без змін, щоб інші користувачі могли виконувати його мимоволі. Наприклад, дошка обговорень, яка приймає коментарі, що містять незмінений HTML, може зберігати шкідливий скрипт від зловмисника. Коли коментарі відображаються, скрипт виконується і може надіслати зловмиснику інформацію, необхідну для доступу до облікового запису користувача. Атака такого роду надзвичайно популярна та потужна, тому що зловмисник може навіть не мати прямого відношення до жертв. Хоча дані із запитів POST або GET є найбільш поширеним джерелом вразливостей XSS, будь-які дані з браузера потенційно вразливі, такі як дані cookie, що відображаються браузером, або файли користувача, які завантажуються і відображаються. Найкращим захистом від вразливостей XSS є видалення або відключення будь-якої розмітки, яка потенційно може містити інструкції щодо запуску коду. Для HTML це включає такі елементи, як <script>, <object>, <embed> та <link>. Процес зміни даних користувача, щоб їх не можна було використовувати для запуску сценаріїв або іншим чином впливати на виконання серверного коду, називається «очищенням введення». Багато веб-фреймворків автоматично очищають введення користувача від HTML-форм за замовчуванням.

Уразливості SQL-ін'єкцій дають змогу зловмисникам виконувати довільний код SQL у базі даних, даючи змогу отримувати, змінювати або видаляти дані незалежно від дозволів або ролі користувача. Успішна ін'єкційна атака може підробити права, створити нові права з правами адміністратора, отримати доступ до всіх даних на сервері або знищити / змінити дані, щоб зробити їх непридатними для використання. [8]

Ця вразливість присутня, якщо користувацьке введення, яке передається в базовий оператор SQL, може змінити сенс оператора. Наприклад, наступний код призначений для перерахування всіх користувачів із певним ім'ям (userName), яке було надано з форми HTML:

```
statement = "SELECT * FROM users WHERE name = '" + userName + "';"
```

Якщо користувач вказує реальне ім'я, оператор працюватиме так, як задумано. Однак зловмисний користувач може повністю змінити поведінку цього оператора SQL на новий оператор у наступному прикладі, просто вказавши текст напівжирним шрифтом для userName.

SELECT * FROM users WHERE name = 'a';DROP TABLE users; SELECT * FROM userinfo WHERE 't' = 't';

Модифікований оператор створює дійсний оператор SQL, який видаляє таблицю користувачів і вибирає всі дані з таблиці userinfo (яка розкриває інформацію про кожного користувача). Це працює, тому що перша частина введеного тексту (a ';) завершує вихідне твердження.

Щоб уникнути такого роду атак, ви маєте переконатися, що будь-які призначені для користувача дані, які передаються в запит SQL, не можуть змінити логіку запиту. Один зі способів зробити це - екранувати всі символи користувацького введення, які мають особливе значення в SQL. [11]

Інструкція SQL обробляє символ ' як початок і кінець рядкового літерала. Помістивши зворотну косу риску перед цим символом (\ '), ми екрануємо символ і говоримо SQL замість цього трактувати його як символ (тільки частина рядка). [9]

У наступному прикладі приведено запит з екрануванням символа '. Тепер SQL інтерпретуватиме ім'я як весь рядок:

SELECT * FROM users WHERE name = 'a\';DROP TABLE users; SELECT * FROM userinfo WHERE \'t\' = \'t';

Веб-фреймворки завжди піклуються про зарезервовані символи за вас. Розробники веб-фреймворків намагаються додавати максимальну кількість корисних можливостей та функцій, задля того, щоб розробникам було більш

комфортніше розробляти веб-ресурси та щоб результат максимально задовільнив замовника по багатьом параметрам.

Django, наприклад, гарантує, що будь-які призначені для користувача дані, що передаються в набори запитів (модельні запити), будуть екрановані.

CSRF-атаки дають змогу зловмиснику виконувати дії, використовуючи облікові дані іншого користувача, без його відома або згоди.

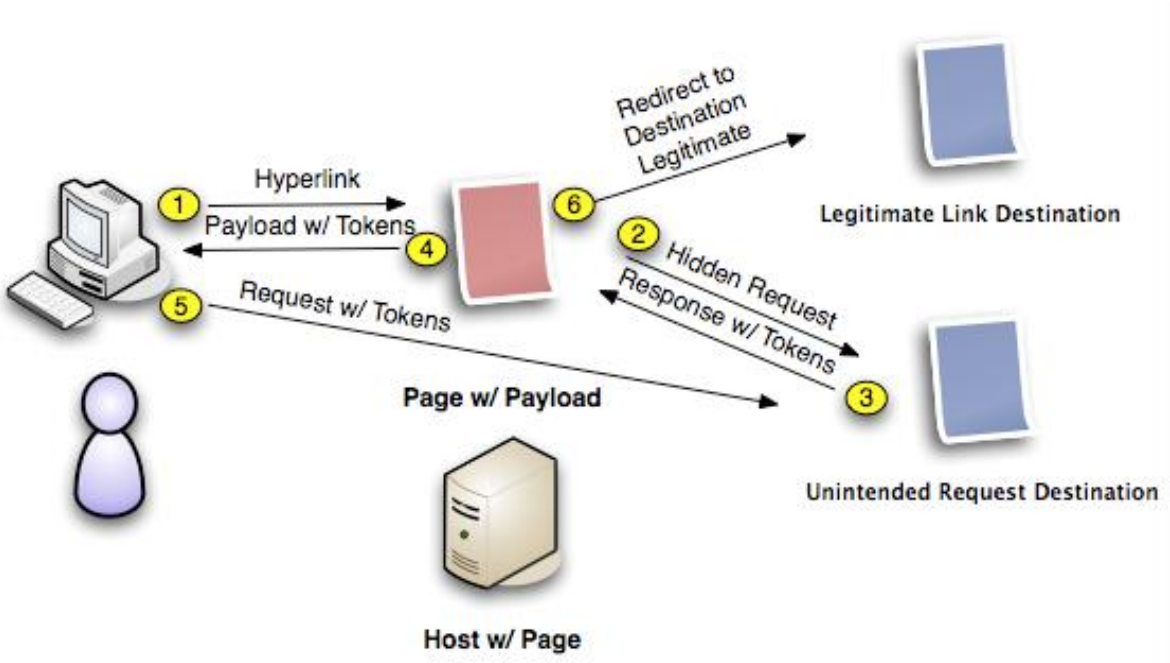

Рис. 1.3. Приклад CSRF атаки

Цей тип атаки найкраще пояснити на прикладі. Джон - зловмисник, який знає, що певний сайт дає змогу користувачам, які ввійшли в систему, надсилати гроші на вказаний обліковий запис, використовуючи HTTP-запит POST, що містить ім'я облікового запису та суму грошей. Джон створює форму, яка містить його банківські реквізити і суму грошей у вигляді прихованих полів, і надсилає її електронною поштою іншим користувачам сайту (з кнопкою "Відправити", замаскованою під посилання на сайт "швидкого збагачення"). [7]

Якщо користувач натискає кнопку відправки, на сервер буде надіслано HTTP-запит POST, який містить відомості про транзакцію та будь-які файли cookie

на боці клієнта, які браузер пов'язав із сайтом (додавання пов'язаних файлів cookie сайту в запити є нормальною поведінкою браузера). Сервер перевірить файли cookie і використовує їх, щоб визначити, чи увійшов користувач у систему і чи має дозвіл на здійснення транзакції.

У результаті будь-який користувач, який натискає кнопку Відправити під час входу на торговий сайт, здійснює транзакцію. Джон стає багатим. [6]

Один зі способів запобігти цьому типу атаки - запросити сервером запити POST, що містять секрет, створений користувачем для конкретного сайту. Секрет буде надано сервером під час надсилання веб-форми, використовуваної для переказів. Такий підхід не дає змоги Джону створити свою власну форму, бо він має знати секрет, який сервер надає користувачеві. Навіть якщо він дізнається секрет і створить форму для конкретного користувача, він більше не зможе використовувати ту саму форму для атаки на кожного користувача.

Веб-фреймворки часто включають такі механізми запобігання CSR

Інші поширені атаки / вразливості включають:

Clickjacking. У цій атаці зловмисник перехоплює кліки, призначені для видимого сайту верхнього рівня, і направляє їх на приховану нижче сторінку. Цей метод можна використовувати, наприклад, для відображення законного сайту банку, але захоплення облікових даних для входу в невидимий <iframe> (en-US), контрольований зловмисником. Clickjacking також можна використовувати для того, щоб змусити користувача натиснути кнопку на видимому сайті, але при цьому насправді мимоволі натискати зовсім іншу кнопку. Як захист ваш сайт може запобігти вбудовуванню себе в iframe на іншому сайті, встановивши відповідні заголовки HTTP. [12]

Denial of Service (DoS). DoS зазвичай досягається за рахунок заспамленням цільового сайту підробленими запитами, так що доступ до сайту порушується для законних користувачів. Запити можуть бути просто численними або окремо споживати великі обсяги ресурсів (наприклад, повільне читання або завантаження великих файлів). Захист від DoS зазвичай працює, виявляючи і блокуючи "поганий" трафік, пропускаючи при цьому легітимні повідомлення. Ці засоби захисту

зазвичай розташовані перед веб-сервером або на ньому (вони не є частиною самого веб-додатка).

Directory Traversal (Файл і розкриття). У цій атаці зловмисник намагається отримати доступ до частин файлової системи веб-сервера, до яких у нього не повинно бути доступу. Ця вразливість виникає, коли користувач може передавати імена файлів, що містять символи навігації файлової системи (наприклад, ../../). Рішення полягає в тому, щоб очищати введення перед його використанням.

File Inclusion. У цій атаці користувач може "випадково" вказати файл для відображення або виконання в даних, що передаються на сервер. Після завантаження цей файл може виконуватися на веб-сервері або на стороні клієнта (що призводить до XSS-атаки). Рішення полягає в тому, щоб дезінфікувати введення перед його використанням.

Впровадження команд. Атаки з впровадженням команд дають змогу зловмиснику виконувати довільні системні команди в операційній системі хоста. Рішення полягає в тому, щоб дезінфікувати дані, що вводяться користувачем, до того, як їх можна буде використовувати в системних викликах. [10]

Майже всі експлойти безпеки, описані в попередніх розділах, успішні, коли веб-додаток довіряє даним із браузера. Що б ви не робили для підвищення безпеки свого веб-сайту, ви маєте дезінфікувати всі дані, які надходять від користувачів, перш ніж їх відображатимуть у браузері, використовуватимуть у запитах SQL або передаватимуть у виклик операційної системи або файлової системи.

Найважливіший урок, який ви можете винести про безпеку веб-сайтів: ніколи не довіряйте даним із браузера. Це включає, крім іншого, дані в параметрах URL-адрес запитів GET, запитів POST, заголовків HTTP і файлів cookie, а також файлів, завантажених користувачем. Завжди перевіряйте і дезінфікуйте всі вхідні дані. [2]

Необхідно зробити низку інших конкретних кроків:

- Використовуйте більш ефективне управління паролями. Заохочуйте регулярну зміну надійних паролів. Розгляньте можливість двофакторної автентифікації для вашого сайту, щоб на додачу до пароля користувач мусив

ввести інший код автентифікації (зазвичай той, що доставляється через якесь фізичне обладнання, яке матиме лише користувач, наприклад, код у SMS, надісланому на його телефон).

- Налаштуйте свій веб-сервер для використання HTTPS і HTTP Strict Transport Security (en-US) (HSTS). HTTPS шифрує дані, що передаються між вашим клієнтом і сервером. Це гарантує, що облікові дані для входу, файли cookie, дані запитів POST та інформація заголовка не будуть легко доступні для зловмисників.
- Слідкуйте за найпопулярнішими загрозами і насамперед усувайте найпоширеніші вразливості.
- Використовуйте інструменти сканування вразливостей, щоб виконати автоматичне тестування безпеки на вашому сайті. Пізніше ваш дуже успішний веб-сайт може також виявляти помилки, пропонуючи винагороду за виявлення помилок, як це робить тут Mozilla.
- Зберігайте і відображайте тільки ті дані, які вам потрібні. Наприклад, якщо ваші користувачі мають зберігати конфіденційну інформацію, як-от дані кредитної картки, відображайте частину номера картки тільки для того, щоб його міг ідентифікувати користувач, і недостатньо, щоб його міг скопіювати зловмисник і використати на іншому сайті. Найпоширеніший шаблон нині - відображення тільки останніх 4 цифр номера кредитної картки.
- Веб-фреймворки можуть допомогти пом'якшити багато з найпоширеніших вразливостей.

1.3. Мета та завдання управління захисту середовища користувачів веб-порталу організації

Розробка веб-порталів та середовищ управління користувачами для організацій це досить масивний та важкий процес, який включає в себе дуже багато факторів та етапів, для того, щоб отримати повністю готовий продукт.

Абсолютно всі веб-портали повинні бути максимально захищені від загроз, злому та викрадення даних. Програмна частина повинна працювати безперебійно, злагоджено та без нарікань. Також усі можливі модулі, функції, сторінки, форми та інтеграції повинні бути протестовані, перед повним запуском продукту та при подальших змінах коду або нових інтеграціях.

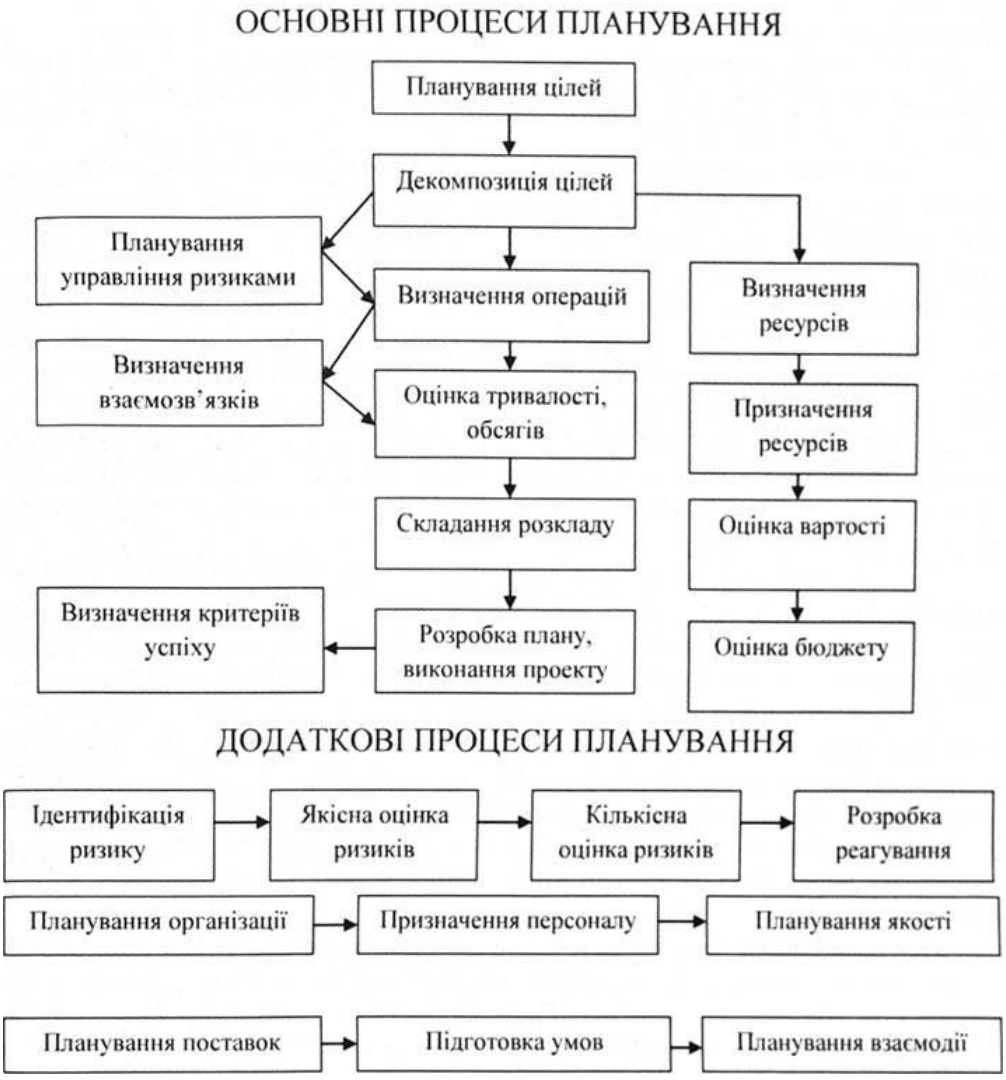

Рис. 1.4. Загальні процеси планування проектів

Нижче виділено 8 основних етапів розробки надійних, повноформатних та сучасних веб-порталів для міських рад:

1. Дослідження:
   - Аналіз бізнес-процесів замовника;
   - Аналіз існуючих видів комунікації та виявлення «слабких місць»;
   - Проведення інтерв'ювання громадян, громадських активістів та організацій, керівництва, відповідальних за комунікації;
   - Аналізуємо статистику використання ресурсу;
   - Дослідження зарубіжних практик і аналогічних успішних кейсів.
2. Проектування і прототипування:
   - Виявлення ключових цільових груп користувачів;
   - Моделювання поведінки різних груп користувачів;
   - Створення найзручнішого інтерфейсу для основної цільової групи;
   - Розробка прототипів сторінок.
3. Тестування:
   - Тестування прототипів у фокус-групах;
   - Доопрацювання прототипу згідно з результатами тестування.
4. Дизайн:
   - Створення візуальної концепції сайту;
   - Розробка графічних макетів сторінок.
5. Верстка та програмування:
   - Верстка дизайн-макетів;
   - Адаптація сайту для всіх пристроїв від широкоформатних моніторів до смартфонів та планшетів;
   - Програмування функціоналу ресурсу;
   - Налаштування зручного інструменту адміністрування;
   - Інтеграція ресурсу в існуючу інфраструктуру замовника.
6. Контент:

- Формування структури і списку актуальних питань;
- Створення простого, доступного і зручного для читання контенту.

7. Запуск:

- Розміщення ресурсу на сервері і налаштування доменного ім’я;
- Проведення бета-тестування;
- Навчання персоналу користуванню ресурсом.

8. Супровід:

- Консультація користувачів по роботі системи;
- Забезпечення кіберзахисту 24/7;
- У разі знаходження помилок – безкоштовне їх виправлення;
- Підтримка актуальності контенту. [18]

Для розробки ресурсу, окрім підготовки, тестування та аналізу бізнес-процесів замовника, потрібно розуміти які технології та підходи можуть бути використані для того чи іншого проекту. Існує велика кількість мов програмування для, підходів та практик.

Найпопулярніші мови для Front-end та Back-end частин:

- JavaScript;
- Python;
- Swift;
- Go;
- Python;
- PHP.

Для написання Back-end частини веб-порталів можна використовувати:

- Мови Node.js, PHP та C# із застосуванням паттернів проектування;
- Фреймворк Yii2, Laravel для великих проектів:
  - застосовується компонентна, модульна архітектура;
  - за допомогою інтерналізації i18n можливе додавання багатомовності;

- присутня можливість інтеграції з різними сторонніми сервісами, через API.

- В якості БД є можливість використовувати Mysql, MongoDB, PostgreSQL.

Для розробки Front-end частини можна використовувати такі методи та технології:

- Верстка макету з використанням HTML, CSS, JavaScript:
  - принципи абсолютно незалежних блоків з БЕМ;
  - анімації і JS інтерактив;
  - адаптивність;
  - кроссбраузерність;
  - Bootstrap 4 (5) для швидкого прототипування візуальної частини веб-порталу;
  - налаштування гнучкого, модульного JavaScript з Webpack, з можливістю писати на новому стандарті ES6, транслюючи код в ES5 за допомогою Babel;
  - автоматична оптимізація графіки з побудовою системи іконок на PNG та SVG.
- Розробка SPA з використанням React + Redux;
- Gulp, Webpack, Babel, ESLint.

Під час та після завершення розробки продукту, потрібно оптимізувати продуктивність проектів:

- скорочення кількості запитів до сервера, шляхом правильного складання залежностей;
- поступове завантаження не критичних компонентів сторінки;
- управління відображенням вмісту в процесі завантаження сторінки;
- правильне використання різних форматів графіки і її оптимізація.

Для того, щоб додати новий функціонал для користувачів веб-порталу, або для адміністрації середовища управління користувачами, організації часто

використовують інтеграції із різними сервісами. Це рішення надає можливість використовувати модулі написані іншою компанією з продажу та створення веб-плагінів або окремою командою розробників яка спеціалізується на створенні, тестуванні та впровадженні віджетів, інтеграцій, веб-інтеграцій, на веб-ресурсах та не витрачати значні кошти на самостійну розробку, тестування та впровадження потрібного функціоналу.

Проте цей, з першого погляду простий процес, повинен бути безпечним у першу чергу для організації, яка впроваджує функціонал зроблений сторонніми людьми. Завдяки інтеграціям, використовуючи шкідливий програмний код стороння організація може отримати змогу впровадити та застосувати небажані шкідливі функції або процеси.

Це можуть бути корисні веб-модулі які стосуються наступної функціональності:

- інтеграція з платіжними сервісами (Visa, MasterCard, PayPal, Payoneer, Portmone);
- сервіси доставки (Нова Пошта, УкрaПошта, Meest Express);
- CAPTCHA для захисту від атаки ботів або спаму (Google Captcha, графічна капча, логічна капча, звукова, математична або візуальна);
- Інтеграція з CRM-системами.

CRM розшифровується як Customer Relationship Management, що означає «управління взаємовідносинами з клієнтами» і стосується всіх стратегій, методів, інструментів та технологій, які використовує бізнес для розвитку, утримання та залучення клієнтів. Customer Relationship Management - це особливий підхід до ведення бізнесу, при якому в основу діяльності компанії ставиться клієнт.

Основна мета впровадження CRM-стратегії — створення єдиної екосистеми із залучення нових та розвитку існуючих клієнтів. Управляти взаємовідносинами означає залучати нових клієнтів, нейтральних покупців перетворювати на лояльних клієнтів, з постійних клієнтів формувати бізнес-партнерів.

Основна перевага CRM-системи полягає в тому, що вона може принести користь практично будь-якому організаційному підрозділу — від продажу та

обслуговування клієнтів до рекрутингу, маркетингу та розвитку бізнесу.

Зберігання всієї інформації про клієнтів в одному місці, реєстрація проблем з обслуговуванням, визначення можливостей продажу, управління маркетинговими кампаніями - це лише кілька можливостей, які надає CRM.

Оскільки CRM забезпечує швидкий доступ до даних, користувачам стає набагато простіше співпрацювати між собою — як наслідок вирішуються питання внутрішньокомандної взаємодії та підвищується продуктивність.

Ще один вагомий аргумент на користь CRM полягає в тому, що система підходить для компаній будь-якого розміру та будь-якої галузі - банків, агентств нерухомості, великих виробничих підприємств, транспортних компаній, дистриб'юторів, телекомунікаційних компаній, державних установ та багатьох інших.

Підключення сторонніх сервісів та інтегрування із різними віджетами або модулями на користь веб-порталу організації може відбуватися за допомогою: API, iframe або із використанням JavaScript коду, HTML розмітки, розміщенням та впровадженням їх на веб-портал організації.

Цей спосіб є дуже зручним з точки зору розробників, тестувальників та навіть зі сторони власника веб-порталу організації, але цей спосіб також може мати перелік проблем та небезпечних.

Розглянемо проблему із iframe:

Елемент IFRAME може бути загрозою безпеці, якщо ваш сайт вбудований всередині IFRAME на ворожий сайт. Google "clickjacking" для більш детальної інформації. Зверніть увагу, що немає значення, використовуєте ви <iframe> чи ні. Єдиний реальний захист від цієї атаки – додати HTTP-заголовок X-Frame-Options: DENY і сподіватися, що браузер дізнається про його роботу.

Крім того, елемент IFRAME може становити загрозу безпеці, якщо будь-яка сторінка вашого сайту містить вразливість XSS, яка може бути використана. У цьому випадку зловмисник може розгорнути атаку XSS на будь-яку сторінку в тому ж домені, яку можна переконати завантажити в <iframe> на сторінці з уразливістю XSS. Це пов'язано з тим, що контент з одного джерела (той самий домен)

дозволений для доступу до батьківського вмісту DOM (практично виконується JavaScript в документі "host"). Єдині реальні способи захисту від цієї атаки - додати HTTP-заголовок X-Frame-Options: DENY і/або завжди правильно кодувати всі дані користувача (тобто. Ніколи не мати вразливості XSS на вашому сайті - простіше сказати, чим зробити).

Це технічний бік проблеми. Крім того, існує проблема з інтерфейсом користувача. Якщо ви навчите своїх користувачів покладатися на те, що рядок URL не повинен змінюватися при натисканні посилань (наприклад, ваш сайт використовує великий iframe з усім фактичним контентом), то користувачі не помітять нічого у майбутньому або у разі реальної вразливості системи безпеки. Наприклад, у вас може бути вразливість XSS на вашому сайті, яка дозволяє зловмиснику завантажувати контент із недружнього джерела у вашому iframe. Ніхто не може сказати різниці, тому що рядок URL, як і раніше, виглядає ідентично попередньому поведінці (ніколи не змінюється), а контент "виглядає" дійсним, навіть незважаючи на те, що він з недружнього домену запитує облікові дані користувача.

Якщо хтось стверджує, що використання елемента <iframe> на вашому сайті небезпечне та створює загрозу безпеці, він не розуміє, що робить елемент <iframe>, або він говорить про можливість <iframe> пов'язаних вразливостей у браузерах. Безпека тега <iframe src="..."> дорівнює <img src="..." або <a href="...">, оскільки в браузері немає вразливостей. І якщо є вразливість, можливо, її можна буде запустити навіть без використання елементів <iframe>, <img> або <a>, тому не варто розглядати цю проблему.

Однак слід попередити, що контент із <iframe> може ініціювати навігацію верхнього рівня за умовчанням. Тобто вміст всередині <iframe> дозволяється автоматично відкривати посилання на поточне розташування сторінки (нове розташування буде відображатися в адресному рядку). Єдиний спосіб уникнути цього – додати атрибут sandbox без значення allow-top-navigation. Наприклад, <iframe sandbox="allow-forms allow-scripts"...>. На жаль, пісочниця також відключає усі плагіни. Жоден браузер не підтримує плагіни та одночасно забороняє

навігацію на верхньому рівні.

Clickjacking - проблема, якщо ваш сайт включений як iframe.

Скомпрометований iframe може відображати шкідливий контент (припустимо, що iframe відображає вікно входу замість оголошення).

Увімкнений iframe може виконувати певні JS-функції, такі як попередження та підказка, які можуть дратувати вашого користувача.

1.4. Аналіз існуючих технологій управління захисту середовища користувачів веб-порталу організації

На сьогоднішній день проблема захисту середовища користувачів веб-порталу є дуже актуальною та важливою. У першу чергу повинен бути максимальний захист середовища управління користувачами веб-порталу і тільки після цього повинен бути максимально захищений сам веб-портал, створена та налагоджена повністю безпечна інтеграція із різноманітними веб-сервісами та платформами. Для того, щоб зберегти цілісність середовища управління користувачами веб-порталу організації та самого веб-порталу організації. Щоб результат роботи веб-порталу задовольняв замовника та організацію в цілому.

Потрібно налаштувати процеси:

- планування розробки проекту;
- розробки самого веб-порталу;
- тестування;
- технічної підтримки;
- планування впровадження на веб-портал нової функціональності, блоків, секцій та інтеграцій у майбутньому.

Але одним аналізом, проектуванням, розробкою та налаштуваннями процесів веб-безпека не може бути створена. Для забезпечення цілісності веб-порталу організації можна та потрібно використовувати велику кількість методів, засобів, підходів та технологій. Серед найпопулярніших та найдієвіших можна відмітити наступне:

- забезпечення цілісності сервера;
- нагальна розробка веб-порталу організації із використанням актуальних підходів та паттернів розробки;
- кропітливе тестування візуальної частини веб-порталу та середовища управління користувачами веб-порталу організації;
- технології аутентифікації користувача;
- технології авторизації користувача.

Якщо вхід до панелі управління користувачами, контентом, даними веб-порталу не буде захищена належним чином, то у зловмисника може з’явитися можливість побачити, редагувати та видаляти данні із закритої частини веб-порталу.

Щоб користування веб-порталом було безпечне для користувача та також безпечне для закритої частини веб-порталу, потрібно використовувати наступні технології:

- OAuth 2.0;
- OpenID;
- JWT (JSON Web Tokens).

OAuth 2.0 є відкритим фреймворком авторизації, що дає змогу отримати стороннім додаткам обмежений доступ до ресурсів HTTP-сервісу.

OAuth з'явився в листопаді 2006 року, під час розробки Блейном Куком протоколу OpenID для сервісу мікроблогів Twitter. Спільно з Крісом Мессіною він шукав спосіб використання OpenID для доступу до Twitter API без надання сервісу пароля. У співпраці з одним із творців OpenID Девідом Рекордоном вони провели аналіз функціональності OpenID, а також кількох інших пропрієтарних протоколів авторизації, і дійшли висновку про необхідність нового, універсального та відкритого протоколу.

У квітні 2007 року утворилася група інженерів, які працювали над його створенням. У її роботі взяли участь співробітники компаній Google і AOL. Фінальна версія ядра протоколу OAuth 1.0 була представлена 4 грудня 2007 року. 15 квітня 2009 року Twitter запропонував користувачам рішення, що дає змогу

делегувати стороннім сайтам і сервісам доступ до своїх акаунтів. Воно було названо "Увійти через Twitter" і засноване на OAuth.

У квітні 2010 року було випущено інформаційний документ RFC 5849, присвячений стандарту OAuth. У 2010 році почалася робота над новою версією протоколу OAuth 2.0. У жовтні 2012 року структуру OAuth 2.0 було опубліковано в RFC 6749, а використання носія токена регламентовано в RFC 6750. Друга версія була несумісна з першою.

Для створення OAuth 2.0 була низка підстав. По-перше, потрібно було спростити розробку клієнтських додатків. По-друге, незважаючи на заявлену в стандарті реалізацію трьох методів отримання токена (унікального ідентифікатора) для авторизації - для веб-застосунків, настільних клієнтів і мобільних клієнтів - фактично всі три способи були злиті в один. По-третє, протокол виявився погано масштабованим. [5]

У результаті до нового протоколу було внесено кілька важливих змін:

- Спрощений підпис. Підпис було значно спрощено, щоб усунути необхідність у спеціальному аналізі, кодуваннях і сортуваннях параметрів.
- Короткоживучі токени з довготривалою авторизацією. Замість видачі токена у якого «довгий термін життя» (який за тривалий час може бути скомпрометований), сервер надає короткочасний доступ і довготривалу можливість оновлювати токен без участі користувача.
- Розділення ролей. За авторизацію і за надання доступу до API можуть відповідати різні сервери.

На даний момент OAuth 2.0 використовується великою кількістю провідних сервісів, таких як Google, Instagram, Facebook, Twitter та інші.

Відмінність від OpenID

Часто можна почути таке запитання. А навіщо потрібен OAuth, якщо існує OpenID? Хоча OAuth і OpenID мають багато спільного, між ними є принципова різниця:

OAuth - протокол авторизації, тобто дає змогу надати права на використання

деякого ресурсу (наприклад, API будь-якого сервісу). При цьому в загальному випадку не можна визначити, хто зараз користується правами.

OpenID є засобом аутентифікації: за допомогою цієї системи можна упевнитися, що користувач - саме той, за кого себе видає. Якими правами володіє користувач, який пройшов аутентифікацію, визначає сторона, що проводить аутентифікацію.

OpenID - це відкрита децентралізована система, яка дає змогу користувачеві використовувати єдиний обліковий запис для автентифікації на безлічі не пов'язаних один з одним сайтів, порталів, блогів і форумів. Якщо веб-сайт передбачає використання аутентифікації або авторизації користувача, то можливі варіанти реалізації цього наведено нижче:

- Використовувати стандартну реєстрацію через введення email і заповнення необхідних даних.
- Використовувати аутентифікацію через зазначення особистого ідентифікатора OpenID провайдера.
- Використовувати авторизацію через відкритий протокол OAuth

Підтримка другого і третього способу є однією з переваг, яка має підвищити кількість зареєстрованих користувачів.

JSON Web Token (JWT) - це JSON об'єкт, який визначено у відкритому стандарті RFC 7519. Він вважається одним із безпечних способів передачі інформації між двома учасниками. Для його створення необхідно визначити заголовок (header) із загальною інформацією щодо токена, корисні дані (payload), як-от id користувача, його роль тощо та підписи (signature).

Останнім часом автентифікація через JWT стала неймовірно популярною, тим часом багато програмістів-початківців не до кінця розуміють, що, окрім очевидних плюсів, у цього підходу є і мінуси.

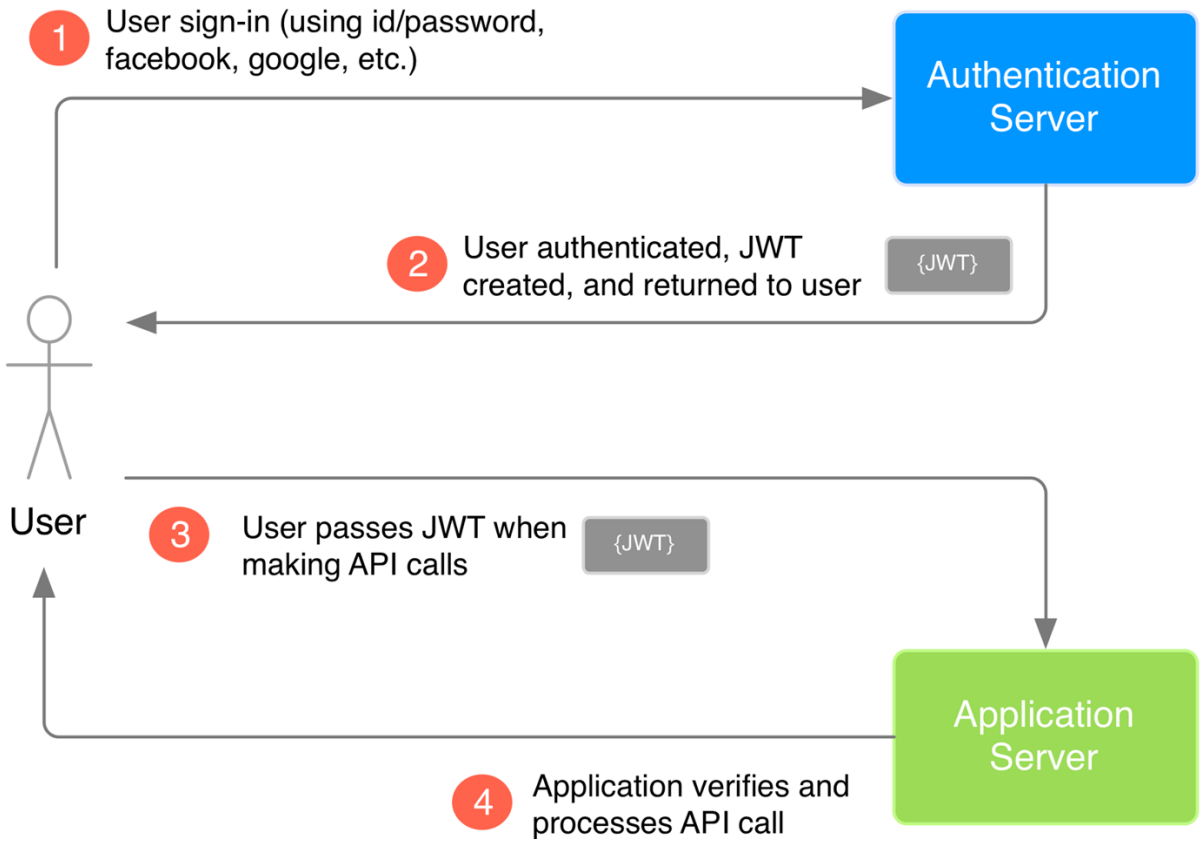

Рис. 1.5. Приклад роботи з JWT

JWT стають дедалі більш усюдисущим. Постачальники послуг з управління ідентифікацією та доступом клієнтів (CIAM) повсюдно просувають JWT як срібну кулю для всього. JWT - це дуже здорово, але хотілося б зазначити деякі недоліки JWT і альтернативні рішення, які також можуть бути розглянуті.

Один зі способів опису JWT полягає в тому, що він є "переносними одиницями ідентифікації". Це означає, що токени містять інформацію про ідентифікацію у форматі JSON і можуть універсальним способом передаватися службам і додаткам. Будь-який сервіс або додаток може самостійно перевірити JWT. Службі/додатку, що отримує JWT, не потрібно запитувати постачальника ідентифікаційних даних, який створив JWT, чи дійсний він. Після перевірки JWT - служба або додаток може використовувати дані, що містяться в ньому, для виконання дій від імені користувача. [14]

2 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ УПРАВЛІННЯ ЗАХИСТОМ СЕРЕДОВИЩА КОРИСТУВАЧІВ ВЕБ-ПОРТАЛУ ОРГАНІЗАЦІЇ

2.1. Призначення та функції протоколу OAuth 2.0, JWT та OpenID

Враховуючи вищенаведені технологій OAuth 2.0, OpenID та JWT, вважаю доцільним навести приклади реального призначення та функцій для веб-порталів організацій, та для систем управління користувачами веб-порталів.

Стандарт OAuth 2.0 визначає такі чотири ролі:

- власник ресурсу - сутність, що має право на видачу доступу до захищених ресурсів. У разі якщо власник є людиною, його називають кінцевим користувачем;
- сервер ресурсів - сервер, що містить ресурси, які захищаються, і володіє можливістю отримання та формування відповіді на запити до ресурсів, які захищаються, за допомогою використання маркера доступу;
- клієнт - додаток, що здійснює доступ до захищених ресурсів від імені власника. Термін "клієнт" явно не визначає будь-яке конкретне виконання (чи то сервер, персональний комп'ютер, чи то мобільний додаток);
- сервер авторизації - сервер, що здійснює випуск маркерів доступу для клієнтських застосунків після успішної автентифікації та авторизації власника ресурсів.

Загальна схема взаємодії має такий вигляд:

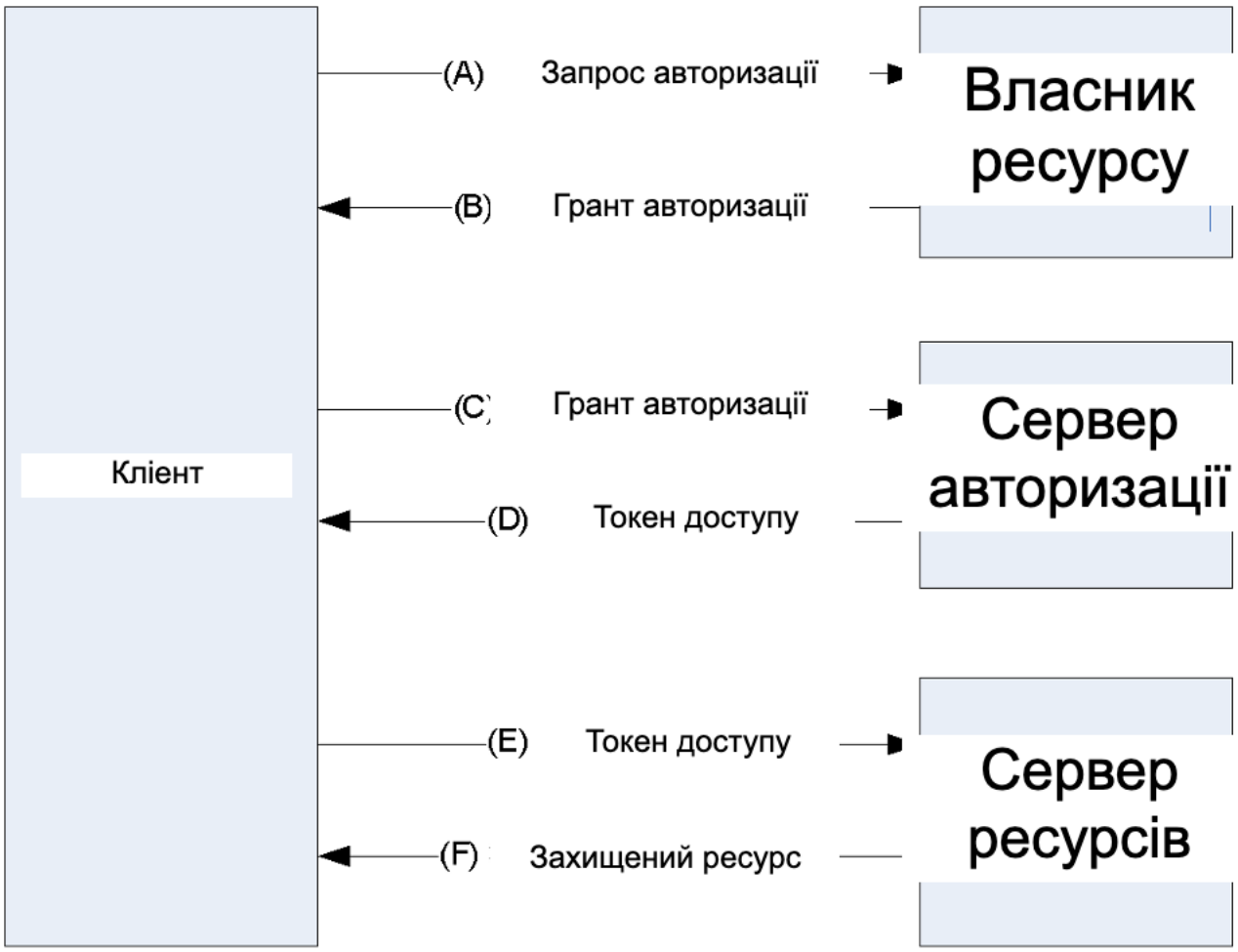

Рис 2.1. Схема взаємодії додатків з користувачем

В описаному вище прикладі взаємодіють два додатки (клієнт, сервер авторизації, поєднаний із сервером ресурсів) і кінцевий користувач. [4]

Таким чином є:

- Кінцевий користувач (ви).
- Клієнт (додаток сайту на якому ми хочемо авторизуватися coursera.org).
- Сервер авторизації (застосунок соціальної мережі Facebook, акаунт, який ми хочемо використовувати). У цьому випадку він же буде і сервером ресурсів, але взагалі це можуть бути різні додатки.

Виходячи із цього, може бути наступна послідовність дій:

- Клієнт запитує в кінцевого користувача проходження авторизації на сервері авторизації (у наведеному прикладі, це коли користувача перенаправляють на сторінку логіна Facebook, потім запитують дозвіл на доступ із застосунку клієнта).
- Після того як кінцевий користувач авторизувався, клієнт отримує грант авторизації. (Фактично грант авторизації клієнту видає сервер авторизації після того, як кінцевий користувач авторизувався і підтвердив видачу запитуваних клієнтом прав).
- Клієнт запитує у сервера токен доступу. При цьому клієнт надає деякі ідентифікаційні дані про себе і грант авторизації від користувача. Токен доступу є альтернативою логіну і паролю, має обмежений час дії і пов'язаний з певними обмеженнями прав. Зазначимо, що клієнт має бути попередньо зареєстрований на сервері авторизації, щоб можна було його ідентифікувати.
- Якщо автентичність клієнта підтверджено і дозвіл на авторизацію дійсний, сервер авторизації створює токен доступу для клієнта і передає його. Авторизацію завершено.
- Клієнт використовує токен доступу для аутентифікації на сервері авторизації.
- Клієнт отримує доступ до необхідних ресурсів (читає дані акаунта користувача Facebook і створює на її основі свій обліковий запис).

OAuth 2.0 розроблено тільки для авторизації - для надання доступу до даних і функцій від одного додатка іншому. OpenID Connect (OIDC) - це тонкий шар поверх OAuth 2.0, що додає відомості про логін і профіль користувача, який увійшов в обліковий запис. Організацію логін-сесії часто називають автентифікацією, а інформацію про користувача, який увійшов у систему (тобто про Resource Owner'a), - особистими даними. Якщо Authorization Server підтримує OIDC, його іноді називають постачальником особистих даних, оскільки він надає Client'у інформацію про Resource Owner'a. [3]

OpenID Connect дає змогу реалізовувати сценарії, коли єдиний логін можна

використовувати в безлічі додатків, - цей підхід також відомий як single sign-on (SSO). Наприклад, застосунок може підтримувати SSO-інтеграцію із соціальними мережами, як-от Facebook або Twitter, даючи змогу користувачам використовувати обліковий запис, який вони вже мають і який вони вважають за краще використовувати.

Потік (flow) OpenID Connect має такий самий вигляд, як і в разі OAuth. Єдина різниця в тому, що при первинному запиті використовується конкретний scope - openid, - а Client у підсумку отримує як Access Token, так і ID Token.

Так само, як і в потоці OAuth, Access Token в OpenID Connect - це якесь значення, не зрозуміле Client'у. З точки зору Client'а Access Token представляє якийсь рядок із символів, який передається разом із кожним запитом до Resource Server'а, а той визначає, чи дійсний токен.

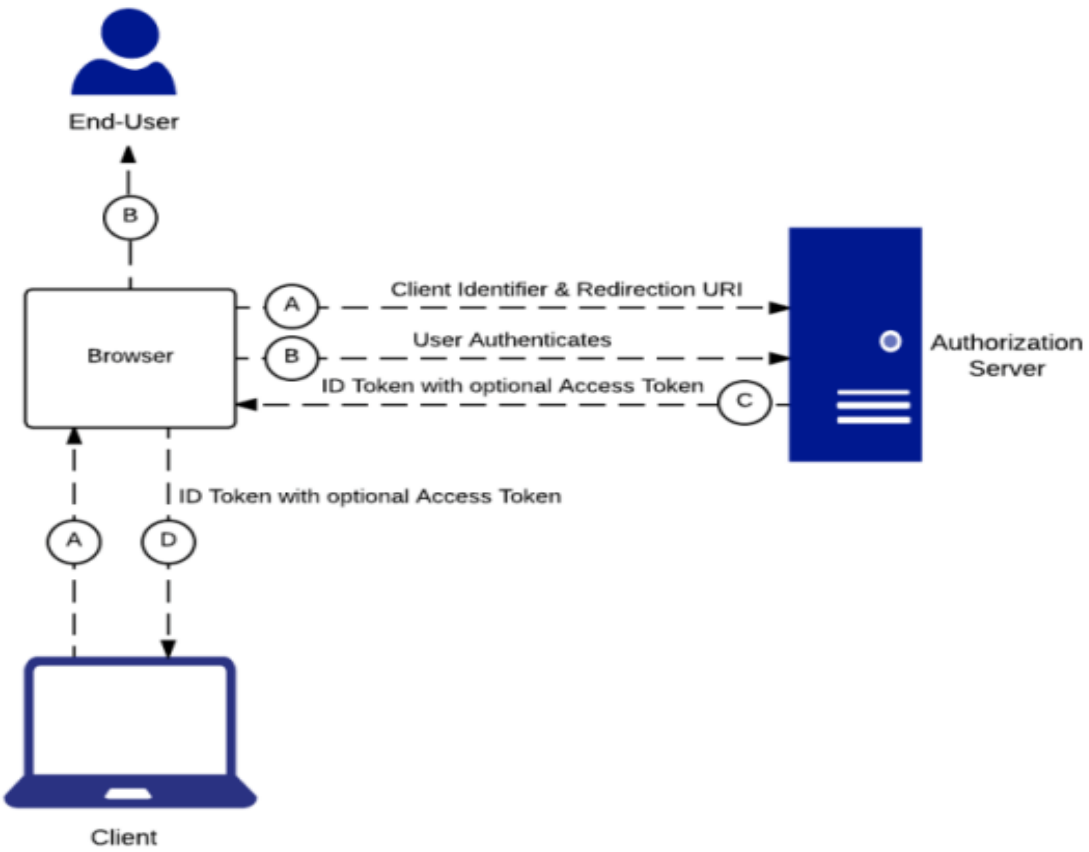

Рис. 2.2. Авторизація користувача

Знизу приведена (рис. 2.3.), на якому показано як постачальник ідентифікаційних даних створює JWT і як служба може використовувати JWT, не звертаючись до постачальника ідентифікаційних даних

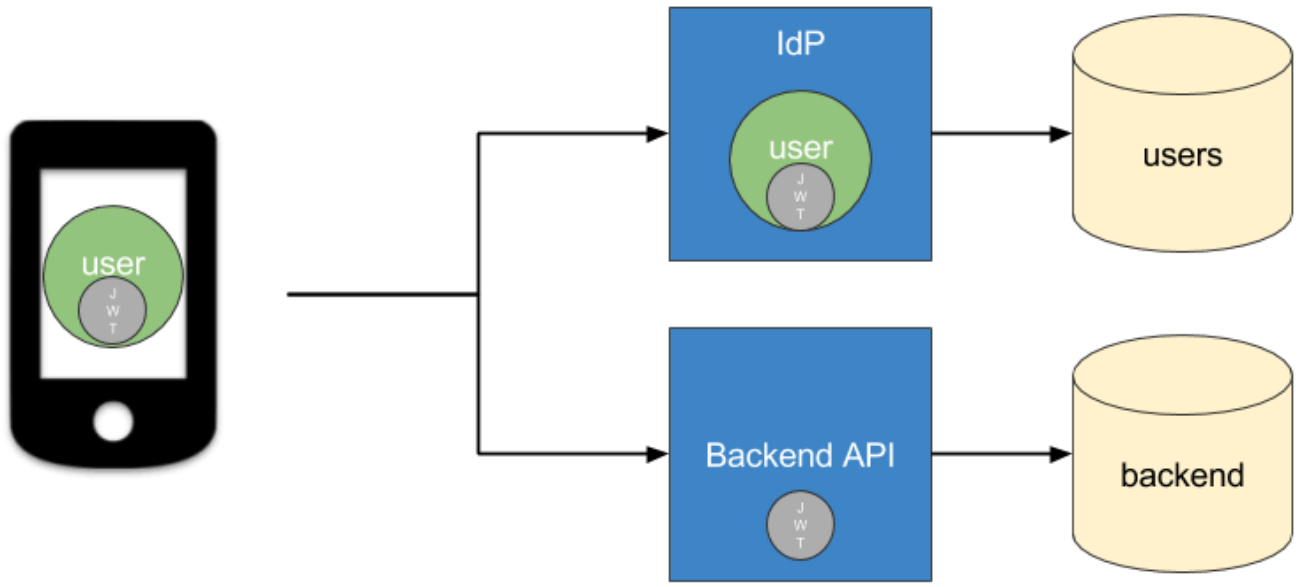

Рис. 2.3. Ідентифікаційні дані JWT

Коли ви порівняєте це з непрозорим токеном, ви зрозумієте, чому так багато розробників використовують JWT. Непрозорі токени - це просто великий рядок символів, що не містить жодних даних. Такий токен має бути перевірений шляхом додаткового запиту в постачальника ідентифікаційних даних, чи дійсний він ще і чи повертає він дані користувача, необхідні службі.

На рис. 2.4 , показано як викликається постачальник ідентифікаційних даних для перевірки непрозорого токена і для отримання даних користувача:

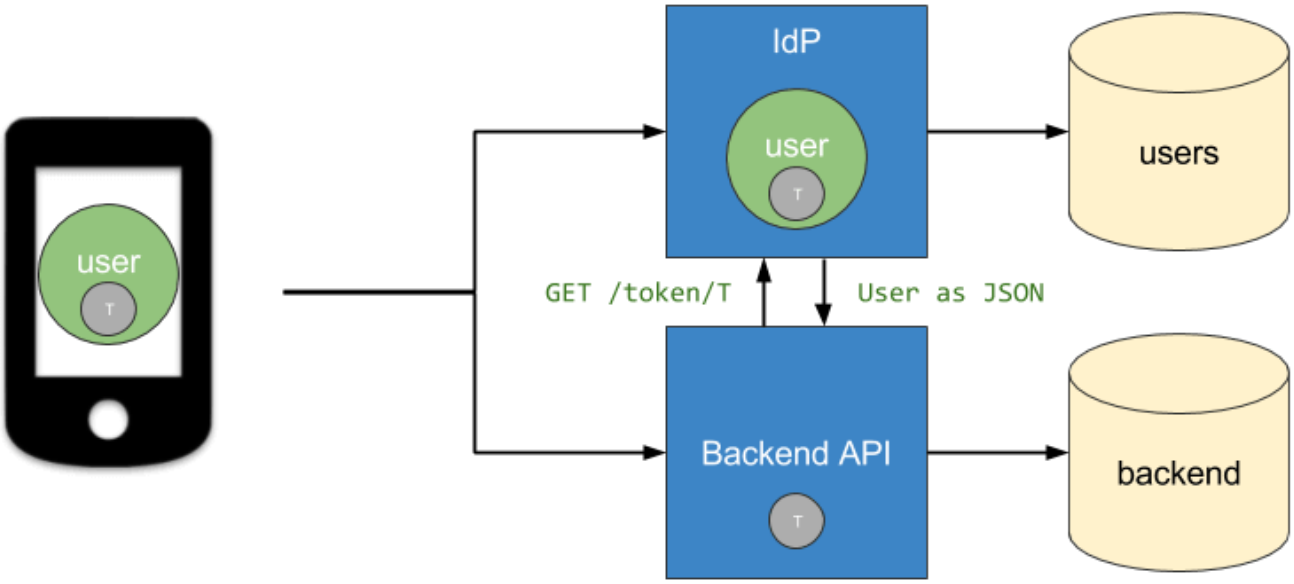

Рис. 2.4. Виклик постачальника ідентифікаційних даних

Цей метод перевірки та обміну токенами може бути дуже "балакучим", створюючи певний оверхед. Крім того, він також вимагає методу збереження і завантаження токенів всередині провайдера ідентифікації. JWT-токени, з іншого боку, не потребують збереження або специфічної логіки у провайдера ідентифікації, оскільки вони переносяться та є універсальними. [16]

2.2. Компоненти та архітектура рішення OAuth 2.0, OpenID та JWT

На сьогоднішній день використання рішення OAuth 2.0 для авторизації є одним із найкомфортніших та безпечних способів. Ця технологія зарекомендувала себе як стабільна та зручна у використанні. Саме тому, топ організації і компанії використовують її на своїх веб-порталах, сайтах та застосунках. Впровадження OAuth 2.0 на веб-ресурс є достатньо корисним та не важким процесом. Розробники цієї технології створили достатню кількість документації, гайдів, прикладів для впровадження її на веб-ресурси. Також розробники написали та створили велику кількість матеріалу по налаштуванню та використанню OAuth 2.0. Завдяки їм процеси стали значно простіше та комфортніше.

Розглянемо компоненти та архітектуру рішення із використанням OAuth 2.0.

Щоб запросити токен доступу, клієнт отримує авторизацію від власника ресурсу. Дозвіл виражається у вигляді гранту авторизації (authorization grant), який клієнт використовує для запиту токена доступу (access token). OAuth визначає чотири типи надання:

- код авторизації (authorization code),
- неявний (implicit),
- облікові дані власника ресурсу,
- облікові дані клієнта.

Також надається механізм розширення для визначення додаткових типів грантів. Розглянемо їх детальніше.

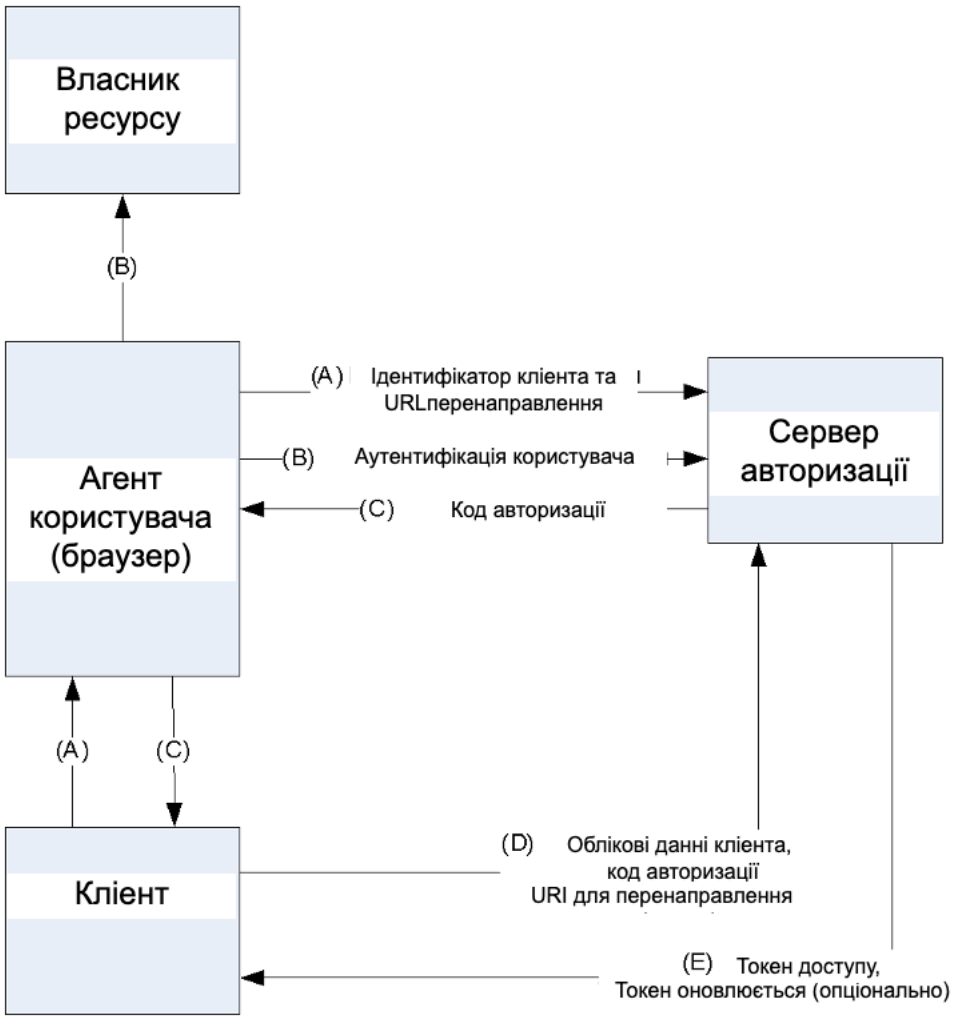

Рис 2.5. Код авторизації (авторизація для додатків, що мають серверну частину).

- Клієнт ініціює потік, направляючи агента користувача (інтернет-браузер) до кінцевої точки авторизації. Клієнт включає свій ідентифікатор клієнта, запит прав, локальний стан і URI переспрямування, на який сервер авторизації відправить призначеного для користувача агента, після того, як тільки доступ буде надано (або заборонено).
- Сервер авторизації автентифікує власника ресурсу (через користувацький агент) і надає або відхиляє запит клієнта на доступ.
- Припускаючи, що власник ресурсу надає доступ, сервер авторизації перенаправляє призначений для користувача агент назад клієнту, використовуючи URI перенаправлення, наданий раніше (у запиті або під час реєстрації клієнта). В URI перенаправлення сервер включає код авторизації.
- Клієнт запитує токен доступу у сервера авторизації, відправляючи код авторизації отриманий на попередньому кроці. Роблячи запит, клієнт аутентифікується на сервері авторизації. Раніше клієнт повинен був зареєструватися на цьому сервері авторизації. Сервер авторизації під час реєстрації видає клієнту секретний ключ. Клієнт включає цей ключ у запит токена доступу, щоб пройти аутентифікацію на сервері.
- Сервер авторизації автентифікує клієнта, перевіряє код авторизації та гарантує, що отриманий URI перенаправлення відповідає URI, використовуваному для перенаправлення клієнта в кроці (3). Якщо це справді так, сервер авторизації надає токен доступу і за необхідності токен оновлення.

Тепер розглянемо компоненти та архітектуру рішення із використанням технології JWT.

Однією із головних функцій JWT є те, шо термін дії JWT спливає через певні проміжки часу.

Коли створюється JWT, йому призначається певний термін дії. Термін дії JWT є фіксованим, також рекомендується, щоб він був невеликим (думайте про хвилини, а не про години). Якщо у вас є досвід роботи з традиційними сесіями, то JWT у цьому плані сильно відрізняється від них.

Традиційні сесії авторизації завжди мають певну тривалість з моменту

останньої взаємодії з користувачем. Це означає, що якщо користувач натискає на кнопку, його сесія продовжується. Якщо згадати більшість застосунків, якими ви користуєтеся, то це досить поширене явище. Ви автоматично виходите з програми після певного часу бездіяльності.

Токени JWT, з іншого боку, не поновлюються під час взаємодії з користувачем. Замість цього вони програмно замінюються шляхом створення нового JWT для користувача, причому все відбувається прозоро і у фоновому режимі.

Щоб вирішити цю проблему, більшість додатків використовують маркери оновлення. Маркери оновлення - це непрозорі токени, які використовуються для генерації нових JWT. Термін дії маркерів оновлення також повинен закінчитися в певний момент, але вони можуть бути більш гнучкими в цьому механізмі, оскільки зберігаються у провайдера ідентифікації. Це також робить їх схожими на стандартні непрозорі токени, описані вище.

Оскільки JWT підписані криптографічно, для їхньої перевірки потрібен аналогічний криптографічний алгоритм. Криптографічні алгоритми спеціально розробляються так, щоб бути повільними. Що повільніший алгоритм, то вища складність, і то менша ймовірність того, що алгоритм можна зламати методом перебору (brute force).

На чотирьохядерному MacBook Pro можна створити і підписати близько 200 JWT на секунду за допомогою RSA з відкритим і закритим ключем. Це число різко знижується на віртуалізованому обладнанні, такому як Amazon EC2s. Підписання HMAC набагато швидше, але не володіє такою ж гнучкістю і характеристиками безпеки.

Зокрема, якщо постачальник ідентифікаційних даних використовує HMAC для підписання JWT, то всі служби, які хочуть перевірити JWT, повинні мати секрет HMAC. Це означає, що всі служби тепер також можуть створювати і підписувати JWT. Це робить JWT менш переносимими (зокрема, для публічних служб) і менш безпечними.

Щоб дати уявлення про характеристики продуктивності JWT і

використовувані криптографічні алгоритми, було проведено декілька тестів на чотирьохядерному MacBook. Ось виміри, які були отримані під час масового створення JWT (у форматі метрика/таймінг):

- Серіалізація JSON + кодування Base64 - 400,000/s
- Серіалізація JSON + кодування Base64 + підписання HMAC - 150,000/s
- Серіалізація JSON + кодування Base64 + підписання RSA - 200/c
- Декодування Base64 + парсинг JSON - 400,000/s
- Декодування Base64 + парсинг JSON + перевірка HMAC - 130,000/s
- Декодування Base64 + парсинг JSON + верифікація RSA - 6,000/s

JWT не можна легко відкликати

Це означає, що JWT може бути дійсним, навіть якщо обліковий запис користувача було призупинено або видалено. Є кілька способів обійти це, включно з "подією відкликання оновленого токена" у поєднанні з webhook. Це рішення, наприклад, доступне у FusionAuth.

JWT мають уразливості і це скоріше питання поганого кодування, ніж недоліків, притаманних JWT. Алгоритм "none" і злом "HMAC" - добре відомі випадки використання JWT.

Обидві ці атаки мають прості виправлення. Зокрема, ви ніколи не повинні дозволяти використовувати JWT-токени, які створені з використанням алгоритму "none". Також не слід сліпо завантажувати ключі/підписи, використовуючи заголовок "kid" у JWT. Натомість слід додатково перевірити, що ключ справді є правильним ключем для алгоритму, зазначеного у заголовку.

Замість використання JWT або непрозорих токенів у вас завжди є можливість використовувати як альтернативу - сесії. Сесії існують уже понад два десятиліття і є перевіреною технологією. Сесії зазвичай працюють за допомогою файлів cookie і стану, який зберігається на стороні сервера. Cookie містить рядок символів, який є ідентифікатором сесії. Сервер містить великий хеш, у якому зберігаються довільні дані. [15]

Коли користувач входить у систему, об'єкт користувача зберігається в сесії, а сервер надсилає назад сесійний файл cookie, що містить ідентифікатор сесії. Кожен наступний запит до сервера включає індивідуальний файл cookie для цієї сесії. Сервер використовує файл cookie для завантаження об'єкта користувача із сесії Hash. Потім об'єкт користувача використовується для ідентифікації користувача, який зробив запит.

Рис. 2.6. та 2.7 ілюструють цю концепцію:

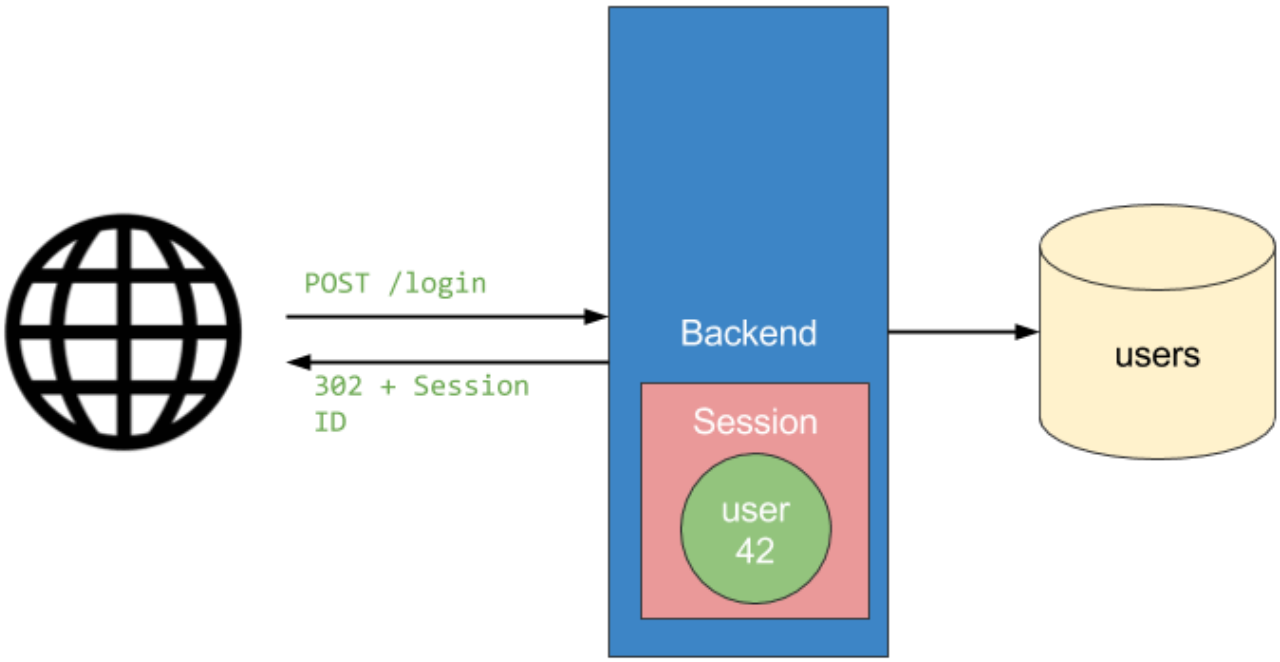

Рис 2.6. Вхід у систему - приклад входу в систему для сесій

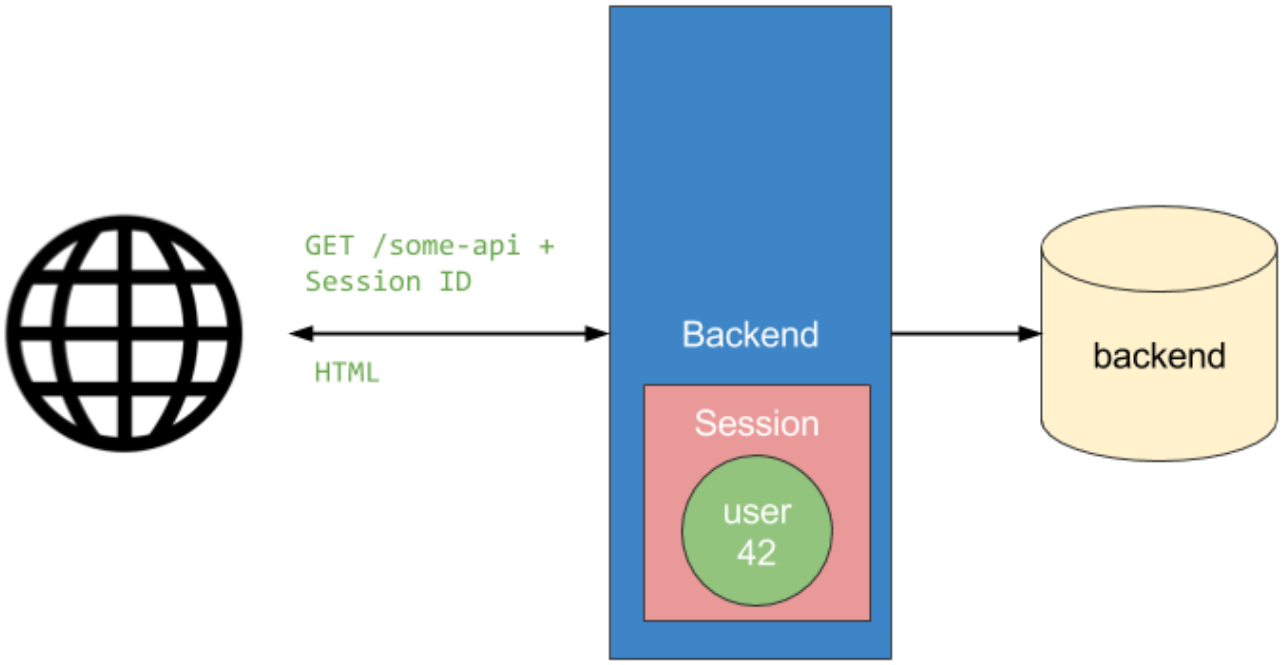

Рис 2.7. Приклад виклику API із сесією

Якщо у організації невеликий веб-портал, що використовує один бекенд, сесії будуть працювати добре. Щойно веб-портал почне масштабуватися або використовувати мікросервіси, сеанси можуть стати складнішим завданням.

Більші архітектури вимагають балансування навантаження і прив'язки сесій, коли кожен клієнт прив'язується до певного сервера, на якому зберігається його сесія. Для забезпечення відмовостійкості або оновлення без простою, може знадобитися реплікація сеансів або розподілений кеш. Навіть з урахуванням цієї додаткової складності, сесії все ще можуть бути дійсно гарним варіантом.

2.3. Можливості щодо управління захистом середовища користувачів веб-порталу організації на базі OAuth 2.0, JWT та OpenID

Розглянемо приклад авторизації із використанням OAuth 2.0. Приклади наводяться для API Facebook.

Перенаправляємо браузер користувача на сторінку авторизації:

```
> GET /oauth/authorize?response_type=code&client_id=464119&
redirect_uri=http%3A%2F%2Fexample.com%2Fcb%2F123 HTTP/1.1
> Host: connect.facebook.com
```

client_id і client_secret - значення, отримані під час реєстрації програми на платформі. Після того, як користувач видасть права, відбувається перенаправлення на вказаний redirect_uri:

```
< HTTP/1.1 302 Found
< Location: http://example.com/cb/123?code=DoRieb0y
```

Використовуємо отриманий code (код авторизації) для отримання токена доступу, виконуючи запит із сервера:

```
> POST /oauth/token HTTP/1.1
> Host: connect.facebook.com
> Content-Type: application/x-www-form-urlencoded
>
>
grant_type=authorization_code&client_id=464119&client_secret=deadbee
f&code=DoRieb0y&redirect_uri=http%3A%2F%2Fexample.com%2Fcb%2F123

< HTTP/1.1 200 OK
< Content-Type: application/json
<
< {
< "access_token":"SlAV32hkKG",
< "token_type":"bearer",
< "expires_in":86400,
< "refresh_token":"8xLOxBtZp8",
< }
```

Зверніть увагу, що в останньому запиті використовується поле client_secret, яке в даному випадку зберігається на сервері додатка, і підтверджує, що запит не підроблено.

У результаті останнього запиту отримуємо токен доступу (access_token), час його життя (expires_in), тип ключа, що визначає як його треба використовувати, і токен оновлення. Отримані дані можна використовувати для доступу до захищених ресурсів.

```
> GET
/platform/api?oauth_token=SlAV32hkKG&client_id=464119&format=json&me
thod=users.getInfo&sig=... HTTP/1.1
> Host: appsfacebook.com
```

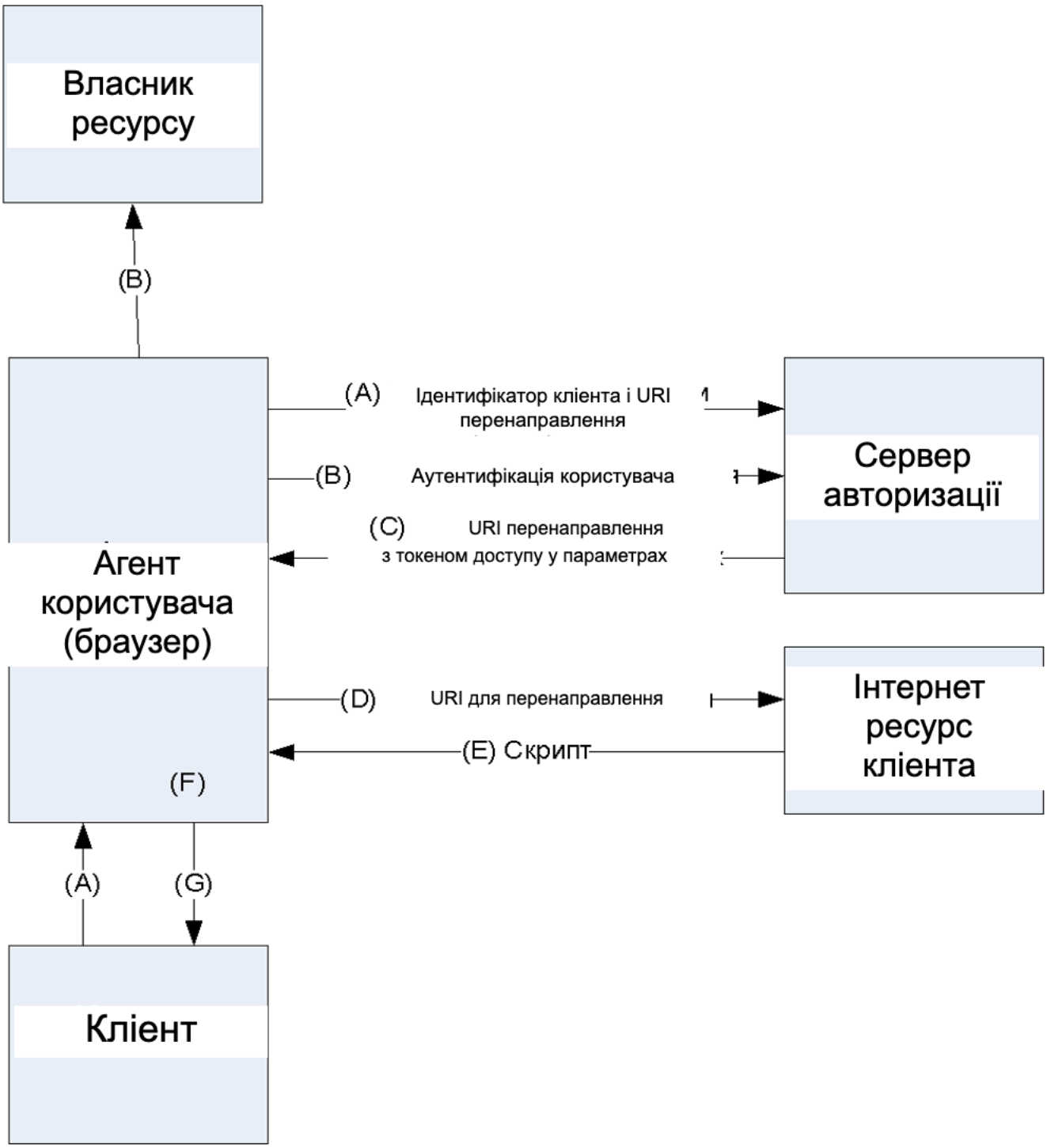

Рис 2.8. Неявний (авторизація повністю клієнтських додатків)

Цей варіант вимагає підняття в застосунку вікна браузера, але не вимагає серверної частини і додаткового виклику сервер-сервер для обміну коду авторизації на токен доступу.

- Клієнт ініціює потік, направляючи агента користувача до кінцевої точки авторизації. Клієнт включає до запиту свій ідентифікатор клієнта, права, локальний стан і URI перенаправлення, на який сервер авторизації надішле користувацький агент, щойно доступ буде надано (або заборонено).
- Сервер авторизації аутентифікує власника ресурсу (через призначений для користувача агент) і надає або відхиляє запит клієнта на доступ.
- Коли доступ надано власником ресурсу, сервер авторизації перенаправляє призначений для користувача агент назад клієнту, використовуючи URI перенаправлення, зазначений раніше. URI перенаправлення включає токен доступу в параметрах.
- Агент користувача слідує інструкціям перенаправлення, створює запит до клієнтського сервісу, розміщеного в інтернеті, а дані параметрів зберігає локально.
- Клієнтський ресурс, розміщений в інтернеті, повертає веб-сторінку (зазвичай документ HTML із вбудованим скриптом), здатний отримати повний доступ до URI переспрямування, включно з даними параметрів, і витягти токен доступу.
- Користувацький агент виконує скрипт, наданий клієнтським ресурсом, локально і витягує токен доступу.
- Користувацький агент передає токен доступу клієнту.

Розглянемо приклад:

Відкриваємо браузер зі сторінкою авторизації:

```
> GET /oauth/authorize?response_type=token&client_id=464119 HTTP/1.1
> Host: connect.facebook.com
```

Після того, як користувач видасть права, відбувається перенаправлення на стандартну сторінку-заглушку.

```
< HTTP/1.1 302 Found
< Location:
http://connect.facebook.com/oauth/success.html#access_token=FJQbwq9&
token_type=bearer&
expires_in=86400&refresh_token=yaeFa0gu
```

Тут застосунок має перехопити останній редирект, отримати з адреси acess_token і використовувати його для звернення до захищених ресурсів.

Облікові дані власника ресурсу (Авторизація за логіном і паролем)

Авторизація за логіном і паролем являє собою простий POST-запит, у результаті якого повертається токен доступу. Цю схему вставлено до стандарту для спільності та рекомендовано до застосування тоді, коли інші варіанти авторизації не доступні.

```
> POST /oauth/token HTTP/1.1
> Host: connect.facebook.com
> Content-Type: application/x-www-form-urlencoded
>
>
grant_type=password&client_id=31337&client_secret=deadbeef&username=
api@corp.facebook.com&password=qwerty
< HTTP/1.1 200 OK
< Content-Type: application/json
<
< {
< "access_token":"SlAV32hkKG",
< "token_type":"bearer",
< "expires_in":86400,
< "refresh_token":"8xLOxBtZp8",
< }
```

Клієнт може запросити токен доступу, використовуючи тільки свої облікові дані, наприклад, коли запитує доступ до захищених ресурсів, що перебувають під його контролем. Цей варіант тривіальний і детально розглядатися не буде.

Зазвичай, токен доступу має обмежений термін придатності. Щоб не змушувати користувача проходити авторизацію після закінчення терміну його дії, у всіх перерахованих вище варіантах на додаток до токена доступу може повертатися ще токен оновлення. За ним можна отримати новий токен доступу за допомогою HTTP-запиту, аналогічно авторизації за логіном і паролем.

```
> POST /oauth/token HTTP/1.1
> Host: connect.facebook.com
> Content-Type: application/x-www-form-urlencoded
>
>
grant_type=refresh_token&client_id=31337&client_secret=deadbeef&refr
esh_token=8xLOxBtZp8


< HTTP/1.1 200 OK
< Content-Type: application/json
<
<
< {
< "access_token":"Uu8oor1i",
< "token_type":"bearer",
< "expires_in":86400,
< "refresh_token":"ohWo1ohr",
< }
```

Виникає питання: чому б одразу не віддати клієнту токен доступу, з яким можна звертатися до ресурсів. Навіщо спочатку отримувати код авторизації, а потім його обмінювати на токен доступу, це начебто зайвий крок.

Річ у тім, що це безпечніше. Канал між додатками (клієнтом і сервером ресурсів) безпечний (back channel). А канал запитів, що проходять через браузер (front channel), - ні. Код авторизації проходить через браузер: він повертається в url під час перенаправлення назад на клієнт. Для звернення до ресурсів код авторизації, що пройшов через браузер, не дуже підходить. Його відносно легко може перехопити зловмисник. Тому його замінюють на токен доступу, що пересилається через безпечний канал (back channel). Крім того, без секрету клієнта (client-secret, який також передається back channel) токен доступу не отримати, що забезпечує додаткову безпеку.

Refresh токен потрібен для: Припустимо, хтось заволодів вашим токеном доступу і отримав доступ до захищених даних. Саме тому в токенів є термін придатності. У токена доступу він зазвичай невеликий - від кількох секунд до кількох днів, у токена оновлення - набагато більший. Так ось: доступ до даних у зловмисника буде доти, доки токен доступу не "закінчить термін життя", тобто недовго.

Припустімо, цей хтось заволодів також і токеном оновлення і отримав новий токен доступу. Вам буде достатньо розлогінитися і залогінитися заново. Усі старі токени стануть недійсними або видаляться (залежить від реалізації). Після цієї процедури ви отримуєте нові токени і можете спокійно продовжити роботу, а зловмисник зі старими токенами залишився з носом.

Переваги та недоліки OAuth 2.0

З переваг протоколу OAuth 2.0 можна виділити таке:

- Звернення до ресурсів відбувається по HTTP/HTTPS із зазначенням токена в заголовках. Це дає змогу використовувати OAuth практично в будь-яких рішення: мобільних і десктоп-додатках, сайтах і навіть у плагінах для браузерів.
- Можливість авторизації користувача.

- Популярність - більшість компаній використовують його у своїх API.
- Простота реалізації та велика кількість літератури.
- Наявність готових рішень, які можна змінювати під свої потреби.

З недоліків:

- Немає єдиного встановленого формату, внаслідок чого на кожен сервіс потрібно мати окрему реалізацію.
- Під час аутентифікації іноді доводиться робити додаткові запити для отримання навіть мінімальної інформації про користувача. Вирішується використанням jwt-токена, але далеко не всі сервіси його підтримують.
- У разі крадіжки токена у зловмисника на якийсь час з'являється доступ до захищених даних. Для мінімізації цього варіанту можна використовувати токен із підписом.

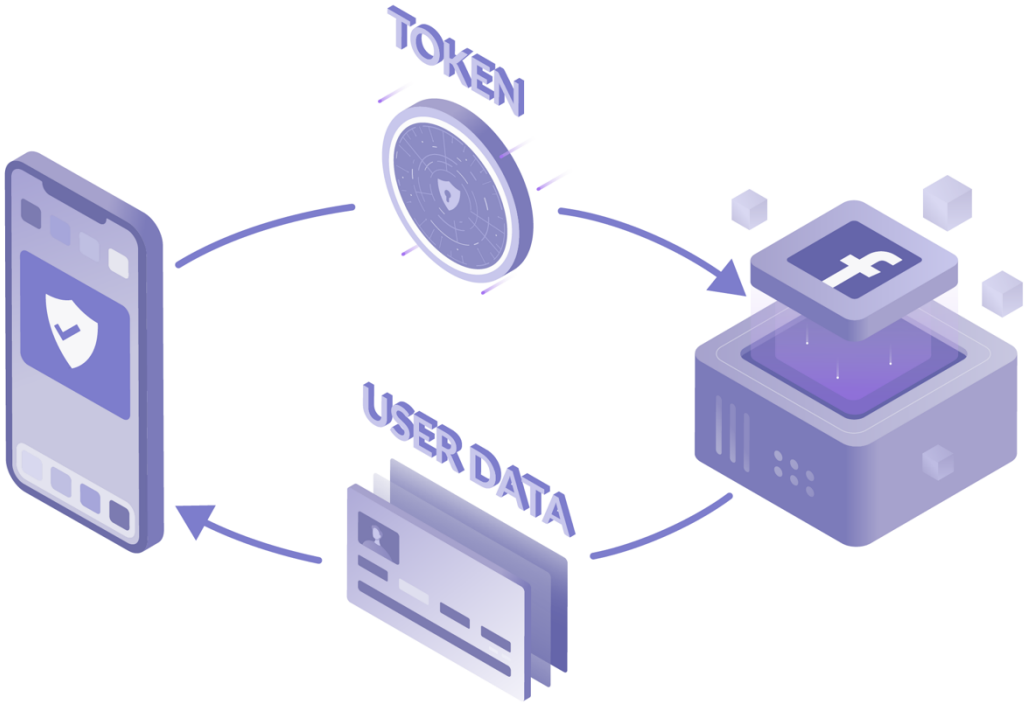

Рис. 2.9. Найпростіший візуальний приклад роботи OAuth 2.0

Отже, OAuth 2.0 - це гнучка технологія для делегування прав доступу до додатків. Сценаріїв використання OAuth 2.0 величезна кількість, це може бути як

спрощений вхід на сторонні сайти, так і автоматизація читання статистики з соцмережі або виконання віддалених обчислень. Що завгодно, що вимагає наскрізної авторизації для доступу до своїх ресурсів.

Загалом, OAuth 2.0 виправляє недоліки OAuth 1.0, але має низку своїх недоліків. Наразі він усе ще перебуває в розвитку. Наступна очікувана версія стандарту - OAuth 2.1. [4]

3 РОЗРОБЛЕННЯ ВАРІАНТА ЗАХИСТУ ТЕХНОЛОГІЇ УПРАВЛІННЯ СЕРЕДОВИЩЕМ КОРИСТУВАЧІВ ВЕБ-ПОРТАЛУ ОРГАНІЗАЦІЇ

3.1. Розроблення варіанта створення та управління, для захищеної системи управління користувачів веб-порталу організації

Для того, щоб створити та управляти веб-порталом, та системою управління користувачами організації, у першу чергу потрібно детально спланувати усі можливі етапи розробки та тестування. На сьогодні створено велику кількість технологій для розробки веб-порталів та інших веб-ресурсів. Великі компанії заохочені у створенні технологій, які будуть популярні та будуть використовуватись великою кількістю розробників та на багатьох великих та навантажених проектах. З часом технології, принципи та паттерни розробки стають все більш комфортними та «продуманими» для користування та використання при розробці. Google, Microsoft та Meta створили одні із найпопулярніших рішень для веб-розробки:

- Google – Golang (Back-end мова програмування. Використовуючи цю мову програмування у розробника є можливість створювати застосунки для Linux, Android, Mac OS X, Windows та веб-застосунки);
- Microsoft - .NET Framework (програмна технологія, яка використовується як платформа для звичайних програм, так і для веб-застосунків);
- Meta – React.js (бібліотека для створення користувацьких інтерфейсів, основою якої є мова програмування JavaScript).

В основному створення веб-порталу та його системи управління користувачами починається із планування процесів. Організація, розробники, тестувальники та технічні письменники повинні бути обов’язково залучені у цей важкий, але дуже потрібний процес. Тому що якість та безпека веб-порталу дуже сильно залежить від правильно побудованої організації та планування. Замовник

або організація повинні бути заохочені у контролюванні процесів розробки для того щоб отримати повністю готовий, захищений та працюючий веб-портал або інший веб-ресурс.

Після або під час планування, починається етап вибору мов програмування для Front-end та Back-end частин. Я маю досвід в розробці великих навантажених веб-проектів, та можу сказати, що найчастіше для сучасних веб-порталів, інтернет магазинів або інших веб-ресурсів для Front-end частини використовують наступний стек технологій:

- Верстка та візуальна складова – звичний HTML + CSS (SCSS) або готові рішення завдяки використанню шаблонів та компонентів Bootstrap 4/5, Material UI;
- Для роботи із даними, виводити данні із API та для обробки запитів, найчастіше використовують мову програмування JavaScript. Але на сьогодні великі, навантажені та із важкою бізнес логікою проекти все рідше і рідше розробляють на чистому JavaScript. Розробники віддають перевагу бібліотекам та фреймворкам. Тому що вони можуть містити в собі модулі, функції та методи, завдяки яким у розробника відкривається можливість обробляти, виводи, змінювати данні швидше та гнучкіше, аніж при використанні звичайного, «чистого» JavaScript.

У розробників Back-end частин веб-ресурсів, як і у Front-end частині, також є улюблені інструменти, мови програмування та паттерни, які зарекомендували себе якістю, роками та тисячами розроблених проектів. Поспілкувавшись із Back-end розробниками, провівши аналіз проектів, можу сказати, що лідерами у Back-end мовах програмування є:

- PHP (скриптова мова загального призначення, яку часто застосовують для розроблення веб-сайтів);
- Node.js (середовище виконання JavaScript, оточення Node.js включає все, що вам потрібно для виконання програми, написаної на JavaScript.);
- Golang (компільована багатопотокова мова програмування, розроблена

компанією Google);

Одним із найважливіших етапів розробки веб-порталів та систем управління користувачами веб-порталів є тестування. Без правильного та послідовного тестування ніякий веб-портал або веб-ресурс не буде працювати належним чином тим самим побажання організації або замовника не будуть виконані належним чином. Тестування веб-ресурсів може бути дуже різним та варіативним. Напрям тестування може визначати сам тестувальник або замовник.

У кваліфікованого тестувальника повинен бути спеціальний список, “check-list”, в якому будуть виписані та упорядковані по пріоритету перевірки, усі сторінки, модулі, секції та інтеграції. Також під час тестування у кожного тестувальника будуть виникати питання щодо того, чи правильно працює той чи інший модуль або секція, також будуть питання по візуальній частині, якщо вони не занотовані у дизайні проекту.

Тестування веб-ресурсів може поділятися на два основні типи:

- Мануальне;
- Автоматизоване.

Мануальне тестування це частина процесу тестування на етапі контролю якості в процесі розроблення програмного забезпечення. Воно проводиться тестувальниками або звичайними користувачами шляхом моделювання можливих сценаріїв дії користувача.

Завдання тестувальника полягає в пошуку найбільшої кількості помилок. Він повинен добре знати помилки, яких найчастіше припускаються, і вміти їх знаходити за мінімально короткий період часу. Решта помилок, які не є типовими, виявляються тільки ретельно створеними наборами тестів, які можуть бути сформовані та описані у “check-list”.

Library Website Checklist

Library Group Name:

Description	Yes	No	n/a
1. Accessibility features are easy to find			
2. Prominent 'contact us' link with details			
3. Clear text resizing controls at top of the page			
4. Clearly marked home link on every page			
5. Homepage lists key tasks			
6. It is easy to find shire and community information			
7. It is easy to find general library information			
8. It is easy to find out how to join the library			
9. It is easy to find a catalogue link			
10. It is easy to order an item			
11. There is a simple site map			
12. There is prominent search feature			
13. The search feature is easy to use			
14. Search results are simple to interpret and useful			
15. Low bandwidth version of the page is available			
16. Language translation link is available			
17. ALT tags are present for all images			
18. Presentation – well laid out and inviting			
Score /18			

Рис. 3.1. Check-list для тестування веб-ресурсу

Автоматизоване тестування це процес верифікації програмного забезпечення, під час якого основні функції та кроки тесту, наприклад, запуск, ініціалізація, виконання, аналіз і видача результату, виконуються автоматично за допомогою інструментів для автоматизованого тестування.

Також, одним із найважливіших етапів розробки є деплой. Деплой це процес, завдяки якому веб-портал потрапляє у «світ». На етапі розробки та тестування деплой є одним із найважливіших аспектів. Без добре налаштованого процесу деплою коду на сервер, для подальшого тестування продукту – тестування буде уповільнене. Помилки, баги та неточності тестувальником будуть знайдені не так швидко як при добре працюючому процесі деплою. Цей процес повинен буду повністю злагодженим та обговореним із розробниками. Виходячи із свого досвіду, DevOps'и при роботі з GitLab практикують наступний принцип деплою коду на

сервер.

Наприклад, під час створення нового функціоналу, розробник працює у GitLab в задачі під номером #10378. Локально у IDE потрібно створити git-гілку з таким же номером, як і у задачі. Після завершення роботи по задачі розробник повинен виконати декілька команд:

- *git commit ...* – написати короткий та змістовний коментар та вибрати які файли повинні бути «запушені» до git-гілки задачі.
- *git pull origin master* або *git pull origin main* – ця git-команда оновлює проект, та може запобігти конфлікти змін в процесі розробки;
- *git push origin master* або *git push origin main* після успішного виконання описаних вище команд – розробник має право та можливість «запушити» свій «комміт (або комміти)».

Після завершення задачі, зміни від розробника потрапляють у його гілку #10378 (як в прикладі) та можуть знаходитись там скільки завгодно часу. Але якщо вони там будуть знаходиться, на тестову середу вони ніколи не потраплять. Тому, для того, щоб «задеплоїти» зміни на тестову середу потрібно створити merge request із гілки розробника у гілку master (або main).

Завдяки merge request'ам розробники, тестувальники та менеджери проектів мають змогу перевірити код та подивитися на усі зміни в файлах, шо були зроблені в рамках конкретної задачі. Це дуже важливий елемент з точки зору безпеки написаного коду, та у цілому безпеки всього проекту. Тому що, недобросовісний розробник може додати у merge request зміни які містять шкідливий код, код який може зламати або порушити роботу та цілісність веб-порталу організації. Саме тому, дозвіл на прийняття merge request'ів повинен бути у перевірених розробників із достатнім досвідом та у менеджерів проектів. Тому, я вважаю, що усі merge request'и повинні перевірятися задля безпеки веб-ресурсів, та у цілому безпеки організації.

Будь-який злом, витік інформації із системи управління користувачами, порушення цілісності веб-системи може мати дуже негативні наслідки для усієї компанії. Це можуть бути репутаційні збити, фінансові або навіть кадрові.

Замовники можуть зупинити багаторічну співпрацю, фонди можуть перестати фінансувати проекти та розробки модулів або інтеграцій.

Отже, після створення merge request'а на мердж із гілки "main" у гілку "stage", його повинен підтвердити конкретний технічний спеціаліст або менеджер проекту. Після підтвердження merge request'у, зміни із задачі #10378 які опинилися у гілці main, потрапляють у stage гілку. На цьому етапі зміни із гілки stage потрапляють на тестове середовище завдяки процесу deploy. І вже, на тестовому середовищі, тестувальники, розробники, менеджери проектів або навіть замовники можуть переглянути тестову версію та побачити процес розробки, готовність тих чи інших модулів, сторінок або секцій.

Після того, як модулі, сторінки і секції були протестовані на тестовому середовищі, та від тестувальників не було створено ніяких додаткових задач на виправлення, менеджери проекту можуть узгоджувати із командою та замовником процес деплою на prod середовище замовника.

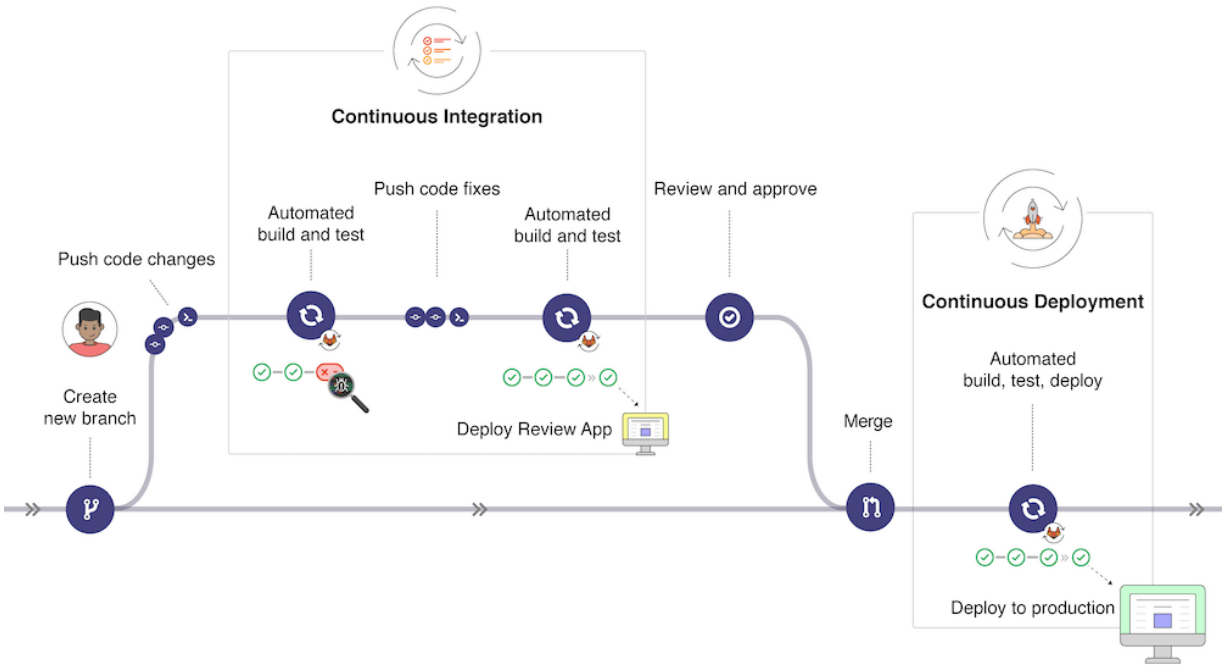

Рис 3.2. Схема деплою ресурсу через інструменти GitLab

Prod середовище це фінальна версія готового веб-ресурсу. Код деплоїться на prod середовище виключно після завершення повного тесту однієї таски на тестовому (stage) середовище. Дивлячись на практику розробки веб-порталів для організацій, можу стовідсотково сказати, що деплой коду на prod середовище відбувається тільки коли набереться стек задач які вже можуть виливатися на prod та протестовані на тестовому середовищі. Також є різні випадки, коли тестувальник або користувач помітив критичний баг на веб-порталі, то потрібно його якнайшвидше виправити, задеплоїти на тестове середовище, тестувальник повинен впевнитися, що проблема повністю вирішена та тільки після цього потрібно у пришвидшеному темпі ініціювати деплой коду на prod середовище. Такий структурований процес дозволяє надавати замовнику або організації якісну підтримку та виправлення критичних помилок без загострення ситуація. Я бачив ряд випадків, коли за проханням замовника виправлення критичної помилки на порталі деплоїлось напряму до prod середовища, без нагального тесту. Після такого наслідки були навіть гірше аніж до «виправлення» критичної помилки.

Тому, виходячи із свого досвіду, я рекомендую використовувати метод який містить в собі повне тестування виправлення або технічного покращення, тому що це може зберегти як нерви так и робочий ресурс, тому що краще спокійно виправити одну критичну помилку, аніж через недбалість розбиратися із нею годинами.

3.2. Технологія управління та розробки захищеної системи управління користувачами веб-порталу організації

Перш за все, для роботи з системою управління користувачами веб-порталу організації, адміністратор повинен мати змогу заходити у систему. Для цього потрібно мати логін та пароль, який повинен бути виданий старшим адміністратором, або DevOps’ом організації.

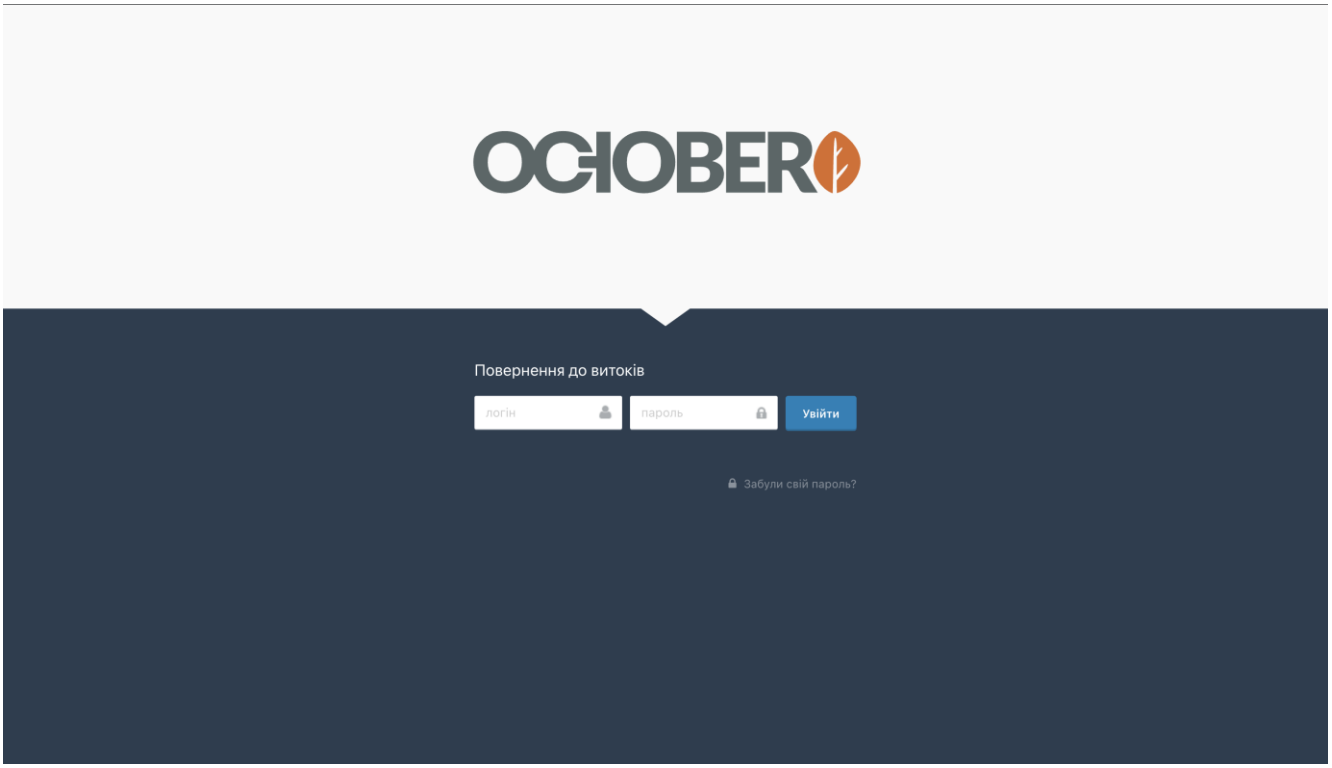

Рис. 3.3. Вхід у панель адміністрування веб-порталу

Панель адміністрування веб-порталу повинна мати усі необхідні модулі для налаштування кожної секції, сторінки, поля. Також для публікації контенту, надання прав користувачам та тощо. Невід'ємним пунктом при розробці веб-порталу та панелі адміністрування веб-порталу є детальне планування структури роботи панелі адміністрування, її вигляд, функції та можливості. Цьому етапу потрібно виділити достатньо уваги, щоб у майбутньому не займатись важкою роботою з перероблення або впровадження нових модулів та функцій.

Навігація повинна бути абсолютно зрозумілою людині, яка не є розробникам, тестувальником або навіть не має великого досвіду у користуванні веб-систем та веб-ресурсів. Для цього і потрібне детальне планування та обговорення із дизайнерами, розробниками та тестувальника. Усі процеси під час розробки повинні будуватися та створюватися задля комфорту користування веб-порталом та системою управління користувачами.

Перш за все, усі адміністратори повинні мати свою роль, для тих цілей які кожен з них буде виконувати. Для цього використовуються: розмежування прав ролі та доступи.

Головний адміністратор (супер-адміністратор) має не обмежений дозвіл на всі дії по веб-порталу. Він має право додавати або видаляти користувачів, змінювати права користування для різних ролей, має доступ до Google Analytics та інше. Зазвичай, на проектах всього один супер-адміністратор. Це зроблено для того, щоб тільки одна людина мала необмежені права керування панеллю адміністрування.

Розмежування ролей це один із найважливіших пунктів, під час організації роботи адміністрації веб-порталу. Наприклад, супер-адміністратор має змогу:

- Надати індивідуальний доступ до певного розділу (модулю);
- Успадкувати - наслідування прав згідно тієї ролі (групи), яку обрали раніше при створенні користувача;
- Заборонити - закрити доступ до розділу (створювати сторінки, пункти меню, редагувати інформацію в новинах або взагалі приховати модулі).

Таблиця 3.1

Розмежування прав та доступів для адміністраторів

Роль	**Модуль**	**Права**
Головний-адміністратор	Новини (“Blog”)	Має повний доступ
Адміністратор, який повинен займатися написанням та публікаціями новин	Новини (“Blog”)	Має доступ тільки для написання та публікації.
Адміністратор, який повинен займатися заповненням модулю медіа-галереї	Новини (“Blog”)	Не має доступу

Адміністратор, який займається заповненням та редагуванням модулю	Новини (“Blog”)	Не має доступу

Усі розмежування, права та доступи зроблені для того, щоб кожен адміністратор мав змогу займатись тільки своїм модулем, тобто модулем або секцією на які він має права.

В першу чергу це зроблено для забезпечення безпеки модулів в яких адміністратор не має достатньо знань та навичок. Наприклад, адміністратор який декілька років займається публікацією новин, захоче зайти до модулю НПА, то він через те, що не має повного розуміння роботи функціоналу, не знайомого для нього модулю, може зробити не правильні дії: видалити, відреагувати, перемістити НПА та інше, та цим порушити злагоджені процеси роботи усієї адміністрації веб-порталу організації. [13]

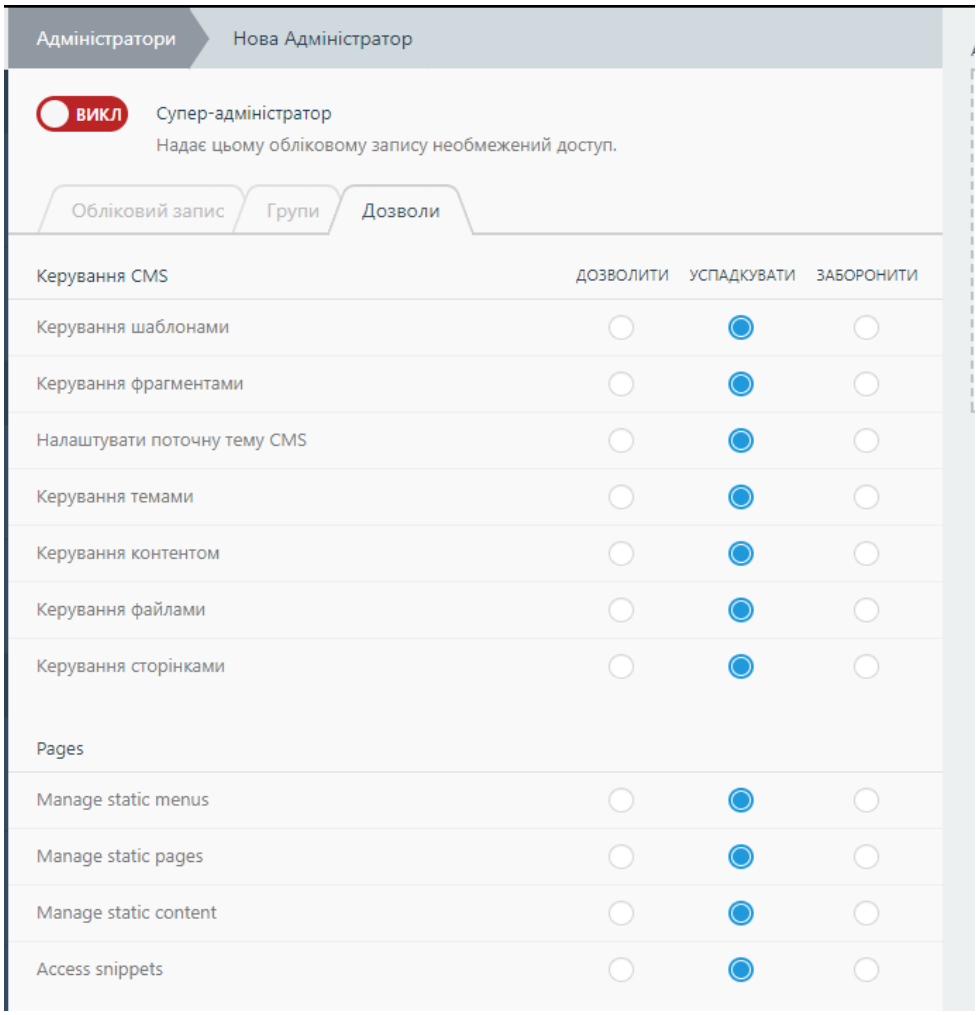

Рис. 3.4. Інтерфейс редагування прав та доступів для адміністратора.

Тільки один адміністратор має право змінювати, додавати або видаляти доступи до взаємодії с модулями. Це все можна зробити через панель адміністрування, за декілька кліків:

- Дозволити означає надати індивідуальний доступ до певного розділу (модулю);
- Успадкувати - наслідування прав згідно тієї ролі (групи), яку обрали раніше при створенні користувача;
- Заборонити - закрити доступ до розділу (створювати сторінки, пункти меню, редагувати інформацію в новинах або взагалі приховати модулі).

Якщо у адміністрації виникають проблеми з роботою по веб-порталу, потрібно проводити зум-конференції або скайп-конференції та допомагати у налагодженні процесів роботи. Якщо проблема існує тільки у одного адміністратора, то її можна вирішити за допомогою зручних месенджерів, таких як Telegram або Viber.

Кожен адміністратор має персональні данні для входу у панель адміністрування веб-порталу. Зазвичай логін це перша літера ім’я, а пароль набір згенерованих символів випадковим способом.

Наприклад: адміністратор модулю Новини – Маненська Галина. У такому випадку її логін буде мати такий вигляд: m.galyna. Також, для максимальної надійності пароль повинен складатися із: заголовних та маленьких літер, цифр, спеціальних символів, та мати довжину 8-12 символів.

Існують спеціальні сервіси, які можуть згенерувати пароль згідно з вашими потребами. Завдяки налаштуванням та вибору наступних параметрів:

- Великі літери
- Малі літери
- Числа
- Символи

Рис. 3.5. Сервіс генерації паролів.

У даному випадку, після генерації паролю сервіс покаже інформацію щодо того, надійний пароль чи ні. Паролі завжди повинні мати статус «Надійний» для безпеки самого працівника, системі керуванням користувачами і контентом веб-порталу.

3.3. Розроблення рекомендацій щодо застосування, управління та розробки захищеної системи управління користувачами веб-порталу організації

Веб-портали або інші веб-ресурси час від часу повинні оновлюватись. Застарілі плагіни, секції або модулі повинні бути відредаговані за новітніми практиками, прикладами із документації та вимогами, або за вимогою організації або замовника. Теж саме стосується і середовищ управління користувачами веб-порталів. Потрібно регулярно перевіряти те, як працює система, збирати відгуки від користувачів та працівників. Також потрібно не забувати про оновлення версій

фреймворків, бібліотек, мов програмування та CMS оболонок, тому що велика ймовірність того, що у нових версіях можуть бути виправлені недоліки, завдяки своєчасному оновленню можна зробити веб-портал організації більш захищеним. Нові частини та модулі веб-порталу мають інтегруватися поетапно, та мати три основних етапи:

1. Обробка задачі.
    - Детальне планування та розробка чіткого ТЗ;
    - Обговорення задачі із розробниками та тестувальниками;
    - Створення та повний опис задачі у вигляді «задач» у GitLab.
2. Розробка та тестування нової функціональності веб-порталу.
    - Виконання усіх вимог, попередньо поставлених у задачі;
    - Перевірка написаного коду;
    - Розгортання оновлення на stage-середовищі;
    - Повне тестування нової функціональності.
3. Впровадження нового функціоналу на prod-середовище.
    - Повторна перевірка написаного коду;
    - Розгортання оновлення на prod-середовищі;
    - Повне тестування.

Для того, щоб нові модулі веб-порталу впроваджувались безпечно - потрібно використовувати налагодженні deploy-процеси, stage-середовище для повного тестування перед розгортанням на prod-середовищі, та повний цикл тестування усіх компонентів, секцій та сторінок та закритої захищеної системи управління користувачами веб-порталу.

ВИСНОВКИ

Проведено аналіз створення захищеного веб-порталу та захисту закритої системи управління користувачами. Ця тема є досить актуальною та важливою у сучасних реаліях. Щодня створюються веб-сайти, веб-портали та інші веб-ресурси. Вони всі потребують систему управління контентом веб-ресурсу, а також середовище управління користувачами. Уніфіковане управління внутрішнім середовищем веб-порталу є комплексним підходом для створення правильної моделі адміністрування всього веб-порталу. Починаючи з публікації різного контенту та завершуючи управлінням, спілкуванням із користувачами веб-ресурсу. Реалізація технології захисту та управління внутрішнім середовищем веб-порталу, створює умови забезпечення багатьох сучасних функцій захисту.

У роботі був розроблений варіант топології поетапної розробки, захисту та технічного супроводження захищеного веб-порталу та середовища управління користувачів веб-порталу, а також були сформовані рекомендацій щодо розробки захищеного веб-порталу, закритої частини веб-порталу та застосування технології захисту веб-порталу організації. Були розглянуті та використані рішення із використанням технологій та протоколів OAuth 2.0, JWT та OpenID. Наведено детальні приклади усіх етапів розробки. Починаючи від планування створення веб-ресурсу та прототипування, закінчуючи модернізуванням вже працюючої системи.

Враховуючи описані принципи, методи та технології та також спираючись на свій досвід, можу рекомендувати описані приклади захисту веб-порталу та системи управління користувачами та контенту веб-порталу. При розробці веб-порталу та системи управління користувачами перш за все потрібно виділяти велику кількість часу на планування розробки. Розробка проекту повинна обговорюватися з усіма потрібними членами команди, наприклад; менеджери проектів, розробники, тестувальники та DevOps’и. Завдяки цьому можна продумати безліч важливих аспектів для якісної розробки веб-ресурсу та його подальшого тестування.

У роботі проведено аналіз проблеми забезпечення кібербезпеки корпоративної інформаційної системи та визначено мета та завдання управління веб-порталом та середовищем користувачів веб-порталу організації. Проаналізовано існуючі технології управління захисту корпоративної інформаційної системи.

Також у роботі була досліджена сутність проблеми забезпечення захисту управління середовищем користувачів веб-порталу організації. Встановлена сутність завдань захисту та розробки захищеного середовища управління користувачами веб-порталу організації. Проаналізовані існуючі технології захисту та розробки захищеного середовища управління користувачами веб-порталу організації, методи та засоби забезпечення захисту, та також розробки захищеного середовища управління користувачами веб-порталу організації та основні функції, принципи реалізації захисту середовища управління користувачами веб-порталу організації.

ПЕРЕЛІК ПОСИЛАНЬ

1. Arthur Tatnall. Web Portals: The New Gateways to Internet Information and Services – 2004. 364 / 381 pages. [Електронний ресурс]. – Режим доступу до ресурсу: https://epdf.tips/web-portals-the-new-gateways-to-internet-information-and-services.html.

2. Website security. MDN docs article [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Website_security.

3. An Introduction to OAuth 2.0 By Mitchell Anicas (Developer and author at DigitalOcean) [Електронний ресурс]. – Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2.

4. The OAuth 1.0 Protocol. [Електронний ресурс]. – Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc5849.

5. Structure of OAuth 2.0 Protocol. [Електронний ресурс]. – Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc5849

6. Методы защиты от CSRF-атаки [Електронний ресурс]. – Режим доступу до ресурсу до ресурсу: https://habr.com/ru/post/318748/.

7. Cross-Site Request Forgery (CSRF) [Електронний ресурс]. – Режим доступу до ресурсу: https://www.owasp.org/index.php/CrossSite_Request_Forgery_(CSRF).

8. Choudhary A. SQL Injection Attacks: Know How to Prevent Them [Електронний ресурс] / Archana Choudhary // Security Zone. – Режим доступу до ресурсу: https://dzone.com/articles/sql-injection-attacks-know-how-to- prevent-them.

9. How to Protect Against SQL Injection Attacks [Електронний ресурс] // UC Berkeley. – Режим доступу до ресурсу: https://security.berkeley.edu/educationawareness/best-practices-how-articles/system-application-security/how-protect-againstsql.

10. 6 Common Website Security Vulnerabilities [Електронний ресурс] // Common places. – Режим доступу до ресурсу: https://www.commonplaces.com/blog/6- common-website-security-vulnerabilities/.

11. Web Server Vulnerabilities Attacks: How to Protect Your Organization [Електронний ресурс] // Tech Funnel. – Режим доступу до ресурсу: https://www.techfunnel.com/information-technology/web-server-vulnerabilities-attackshow-to-protect-your-organization/.

12. Уязвимости веб-приложений. [Електронний ресурс] – Режим доступу до ресурсу: https://www.ptsecurity.com/upload/corporate/ru-ru/analytics/WebVulnerabilities-2019-rus.pdf.

13. Users & Permissions. OctoberCMS documentation. [Електронний ресурс] – Режим доступу до ресурсу: https://docs.octobercms.com/2.x/backend/users.html

14. Безопасность JSON Web Tokens (JWT) [Електронний ресурс] – Режим доступу до ресурсу: https://cyberpolygon.com/ru/materials/security-of-json-web-tokens-jwt/

15. Плюсы и минусы JWT: краткий обзор тонкостей этой технологии. Игорь Грегорченко. [Електронний ресурс] – Режим доступу до ресурсу: https://highload.today/jwt-auth-json/

16. OAuth 2.0 — основы понятным языком. Сергей Рыльков. [Електронний ресурс] – Режим доступу до ресурсу: https://highload.today/oauth-2-0-osnovy/

17. Веб-портал. [Електронний ресурс] – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Веб-портал

18. Етапи створення сайту. [Електронний ресурс] – Режим доступу до ресурсу: https://webcase.com.ua/blog/iz-chego-sostoit-razrabotka-sajta/

КОПІЇ ОБОВ’ЯЗКОВИХ АРКУШІВ (ПРЕЗЕНТАЦІЯ)