МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КІБЕРБЕЗПЕКИ

Кваліфікаційна бакалаврська робота

на тему:

«Захист бази даних при розробці вебдодатку»

Завідувач
випускаючої кафедри **Любчак В. О.**

Керівник роботи **Лаврик Т. В.**

Студента групи КБ – 81 **Петленко М. С.**

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра кібербезпеки

Затверджую_____

Зав. кафедрою Любчак В. О.

“_____” _____2022 р.

ЗАВДАННЯ до бакалаврської роботи

Студента четвертого курсу, групи КБ-81 спеціальності «Кібербезпека» денної форми навчання Петленка Максима Сергійовича.

Тема: «Захист бази даних при розробці вебдодатку»

Затверджена наказом по СумДУ

№_____ від _____2022 р.

Зміст пояснювальної записки: 1) аналіз сучасних вразливостей баз даних та відповідних способів захисту; 2) огляд існуючих систем керування базами даних, виявлення їх переваг та недоліків; 3) побудова моделі вебдодатку з урахуванням типів даних та розроблення моделі захисту бази даних; 4) вибір програмних засобів та опис програмної реалізації; 5) опис методів тестування вебдодатку на вразливості; 6) результати тестування та опис протидії виявленим вразливостям.

Дата видачі завдання “_____” _____2022г.

Керівник бакалаврської роботи _____ Лаврик Т. В.

Завдання приняв до виконання _____ Петленко М. С.

РЕФЕРАТ

Записка: 58 стор., 31 рис., 1 додаток, 12 джерел.

Об’єкт дослідження — способи захисту баз даних вебдодатків.

Мета роботи — розроблення моделі захисту бази даних вебдодатку, її програмна реалізація та тестування безпеки.

Метод дослідження — метод аналітичного огляду, метод порівняння, метод моделювання, гнучкий метод розробки (Agile model).

Результати — проаналізовано найпоширеніші вразливості баз даних та способи їх захисту; розроблено модель вебдодатку з урахуванням типів даних та модель захисту бази даних; здійснено програмну реалізацію вебдодатку на мові PHP 8.1, в якості системи керування базами даних обрано MySql. Описано методи тестування безпеки даних та проведено аналіз результатів тестування.

БАЗА ДАНИХ, ВЕБДОДАТОК, ЗАХИЩЕНА МОДЕЛЬ ВЕБДОДАТКУ, АНАЛІЗ, ВРАЗЛИВІСТЬ, ТЕСТУВАННЯ БЕЗПЕКИ, MYSQL

ЗМІСТ

ВСТУП

З плином часу інформаційні технології стали займати все більше місця у повсякденному житті людини. Тож, можна сказати, що інформація стала одним із головних елементів розвинутого суспільства, а інформаційну безпеку як головну складову у будь-якій сфері життєдіяльності людини, яку необхідно вдосконалювати у першу чергу, через можливість втратити чутливі дані при слабкій захищеності інформаційної системи.

На сьогоднішній день 3,6 мільярдів людей користуються соціальними мережами, що стали не лише місцем для спілкування, а й майданчиком для продажу різних речей або послуг, якому користувачі передають таку конфіденційну інформацію, як місце проживання, дата народження та інше. Окрім соціальних мереж були створені вебресурси, які мають свою спеціалізацію, а саме: вебсайти для продажу товарів, блоги, новинні та сайти-візитівки. Усі ці інформаційні системи містять у собі бази даних користувачів і, тим самим, актуалізують питання щодо вдосконалення систем захисту інформації у мережі.

З кожним днем створюється все більше різноманітних інформаційних систем, що використовують різні типи баз даних. Кожна з них потребує спеціального захисту, необхідність якого підтверджується тим, що мережа Інтернет має гарні перспективи для подальшого розвитку та, при цьому, і великі ризики стати жертвою кіберзлочинця. Тим самим, актуальність питання захисту інформаційних систем постійно є нагальним питанням.

Метою роботи є розгляд можливих загроз, які можуть бути властиві базам даних вебресурсів, та створення механізму систем захисту баз даних.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Аналіз вразливостей та загроз для баз даних

Існує багато різноманітних систем керування базами даних (СКБД), кожна з яких має свої плюси та мінуси. Усі системи управління базами даних поділяються на два напрямки: 1) SQL – мова структурованих запитів; 2) NoSQL – неструктурована мова запитів. Вони також відомі, як реляційні та нереляційні системи керування базами даних. Основні відмінності між цими СКБД полягають у тому, як ці бази даних спроєктовані, які типи даних вони можуть містити в собі та спосіб збереження інформації. Реляційні системи керування базами даних призначені для зберігання або модифікування даних, які представляють собою реальні об’єкти. До таких об’єктів відносяться персональні дані людини, вміст корзини в інтернет-магазині та інші. Формат цих даних задається на етапі проєктування, які потім об’єднуються у таблиці. В той час, як нереляційні системи керування базами даних, які орієнтовані на збереження документів, зберігають інформацію у вигляді ієрархічних структур. Такі СКБД можуть мати об’єкти, які містять довільний набір атрибутів. Тож можна сказати, що одні й ті самі об’єкти будуть по-різному представлені у кожному типі. Наприклад, об’єкт, який знаходиться у реляційній СКБД буде розбитий на декілька пов’язаних між собою таблиць, в той час, як у нереляційній СКБД той самий об’єкт може бути представлений у вигляді однієї цілісної сутності. Як приклад, можна навести базу даних, яка містить відомості про взаємовідносини між людьми [1].

На рис. 1 показано структуру нереляційної бази даних. Її суть полягає у відношенні одних обєктів-сутностей до інших тим або іншим способом.

На рис. 2 показано інший тип СКБД – реляційний. У таких базах даних інформація представлена в табличному вигляді і являє собою альтернативний варіант реалізації структури, що зображена на рис. 1.

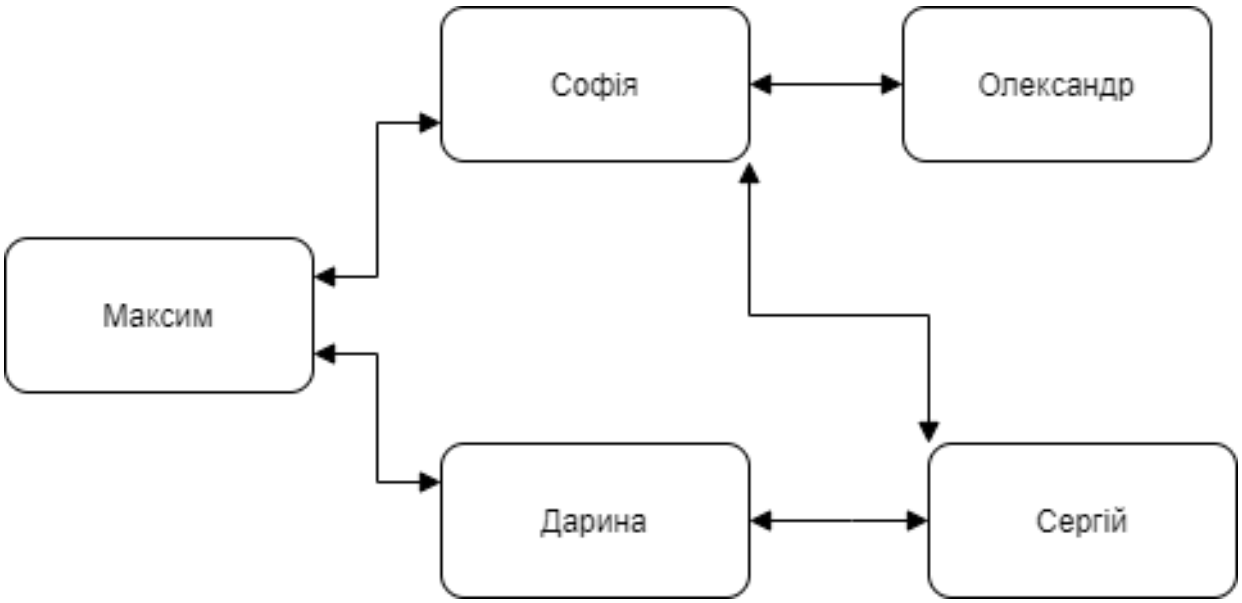


Рисунок 1 – Структура нереляційних баз даних

Персона	Друзі
Максим	{Софія, Дарина}
Софія	{Максим, Сергій, Олександр}
Дарина	{Максим, Сергій}
Олександр	{Софія}
Сергій	{Софія, Дарина}

Рисунок 2 – Структура реляційних баз даних

До реляційних СКБД відносяться: MySQL, Oracle, Microsoft SQL Server, PostgreSQL.

MySQL містить в собі такі властивості:

- Безкоштовна;
- Стабільна та швидко розвивається;
- Доступна на всіх платформах;
- Покриває велику кількість варіантів використання.

Oracle має такі властивості:

- Комерційна СКБД з постійними оновленнями;
- Одна з найдорожчих СКБД;

- Має можливість працювати з великою кількістю даних;
- Проста в оновлені;
- Контроль транзакцій;
- Сумісна з усіма операційними системами.

Властивості Microsoft SQL Server:

- Працює лише на Windows або Linux;
- Дружелюбний до користувачів;
- Важко вносити правки;
- Прекрасно підходить для малих та середніх організацій.

Властивості PostgreSQL:

- Гібридна СКБД;
- Безкоштовна;
- Сумісна з усіма операційними системами;
- Використовує чистий SQL.

До нереляційних СКБД відноситься MongoDB.

MongoDB має такі властивості:

- Безкоштовна;
- Динамічна схема;
- Відмінна продуктивність з простими запитами [2].

Хоча усі вище зазначені СКБД різні між собою та методи їх зламу мало чим відрізняється один від одного. Наприклад, кіберзлочинець може використовувати метод SQL ін’єкцій або інші вразливості. Розглянемо деякі найпоширеніші з них.

SQL ін'єкція

Цей тип атаки впровадження SQL-коду відбувається, коли шкідливий код впроваджується через front-end веб-додатків і потім передається в back-end. Цей процес дозволяє зловмисникові отримати абсолютний доступ до інформації, що зберігається в базі даних.

Метою використання такої вразливості зазвичай є крадіжка даних або їх пошкодження. SQL ін'єкція націлена на традиційні бази даних, а впровадження NoSQL коду – на бази BIG Data. Для того, щоб отримати доступ до бази даних за допомогою SQL ін’єкцій необхідно знайти на сайті будь-яку форму, яка працює з базою даних (це може бути, наприклад, форма реєстрації). Після знаходження такої форми необхідно ввести в поле з логіном будь-які дані, які схожі на логін або адресу електронної пошти, та в поле з паролем ввести умову, яка завжди буде істинною, наприклад, xxx’) OR 1=1 --]. Таким чином, можна отримати доступ до інформації [8].

На рис. 3 та рис. 4 наведено приклад такої атаки, яка було виконана на спеціально створеному сайті (http://www.techpanda.org/).

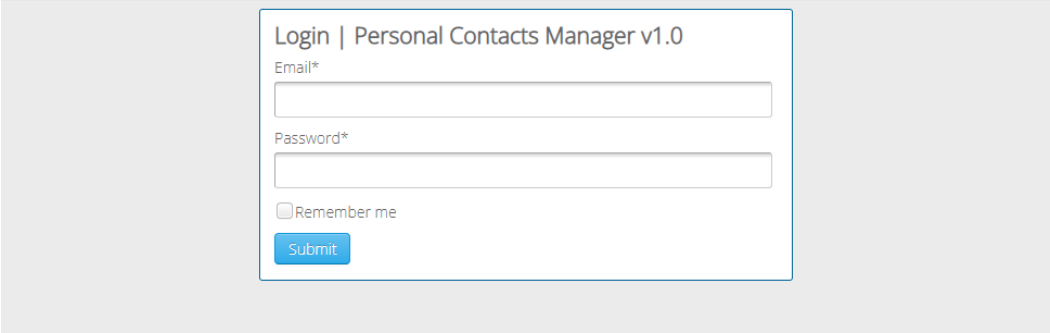


Рисунок 3 – Сторінка логіну на тестовому сайті

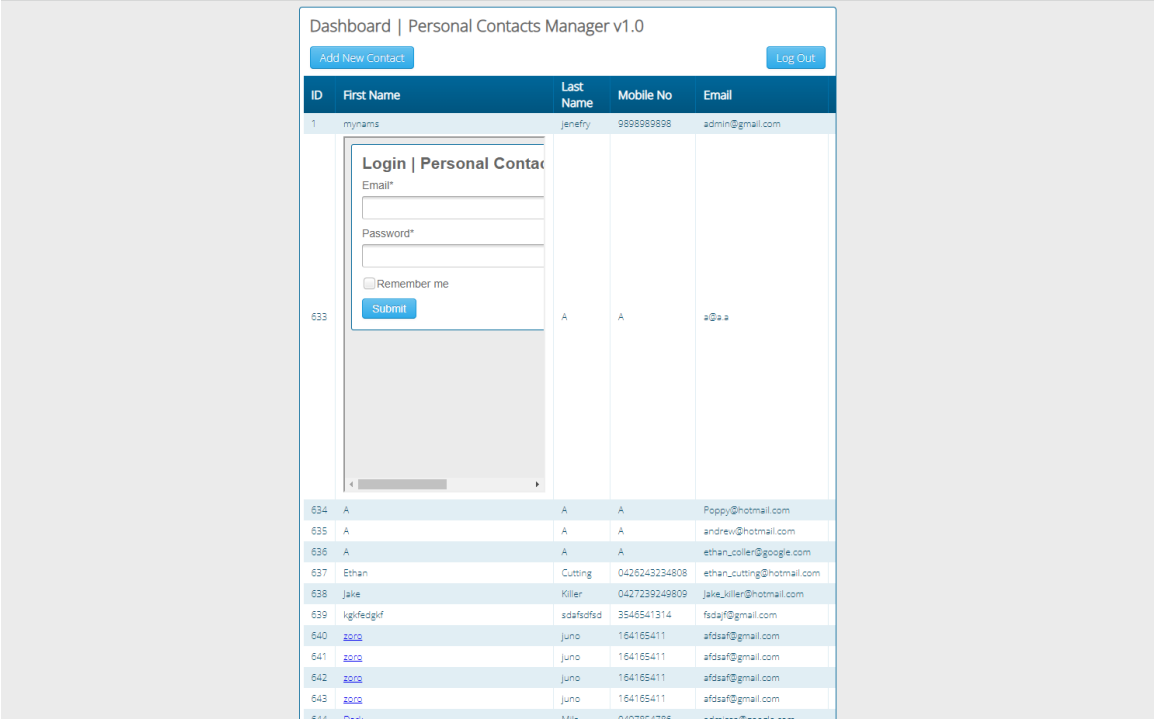


Рисунок 4 – Сторінка з результатами атаки

Окрім перевірки в ручному режимі на вразливість бази даних до SQL ін’єкцій, кіберзлочинець може використати спеціальні утиліти для знаходження вразливостей та їх подальшої експлуатації. Прикладами таких утиліт є:

- Netsparker;
- Acunetix;
- SLOB
- Orion
- Se Lite
- NoSQL Lite
- Intruder;
- Solarwind;
- OpenVAS;
- Nexpose;
- Tripwire;
- Wireshark.

Необмежені привілеї бази даних

Зазвичай це відбувається, коли користувачам бази даних надаються численні привілеї в системі, що призводить до зловживання привілеями, яке може бути надмірним, законним чи невживаним. Такі дії можуть бути вчинено як діючими, так і колишніми співробітниками компанії.

Для запобіганню цьому типу атак необхідно запровадити суворий контроль доступу та привілеїв, а також переконатися, що нікому зі співробітників не було надано надмірні привілеї, і завчасно деактивувати застарілі привілеї та робочі профілі [9].

Поганий аудиторський слід

Згідно з деякими стандартами безпеки кожна подія в базі даних має бути записано для цілей аудиту. Якщо не можна буде надати суду докази наявності журналу аудиту бази даних, то це може являти собою дуже серйозний ризик

для безпеки, оскільки в разі вторгнення неможливо буде провести розслідування.

Відкриті резервні копії баз даних

Кожній організації необхідно слідкувати за планом резервного копіювання. Але коли резервні копії доступні, вони стають відкритими для компрометації та крадіжки. Багато випадків витоку конфіденційної інформації виникало саме через те, що резервна копія бази даних не була зашифрована.

Шифрування і аудит виробничих баз даних і резервних копій – найкраща форма захисту корпоративних конфіденційних даних.

Неправильна конфігурація бази даних

Деякі з загроз, що зустрічаються в базі даних, є результатом їх неправильної конфігурації. Зловмисники зазвичай користуються базою даних, яка має стандартний обліковий запис і налаштування конфігурації.

Це тривожний сигнал, що при налаштуванні бази даних не повинно бути нічого схожого на обліковий запис за замовчуванням, а параметри повинні бути налаштовані таким чином, щоб зловмисникові було складно що-небудь зробити.

Відмова в обслуговуванні (DoS)

Це тип атаки, який впливає на доступність сервісу. Він впливає на продуктивність сервера бази даних і робить сервіс бази даних недоступним для користувачів.

Наприклад, якщо є запит на дуже важливі фінансові дані, а база даних недоступна через DoS, то це може привести до втрати фінансових ресурсів [10].

Погане управління даними

Деякі організації не вміють правильно керувати своїми конфіденційними даними, вони не ведуть їх точну інвентаризацію, і таким чином деякі з цих конфіденційних даних можуть потрапити до рук зловмисників. Якщо не провести належну інвентаризацію нових даних,

доданих до бази даних, то вони можуть стати вразливими. Тому дуже важливо шифрувати дані і застосовувати необхідні дозволи та засоби контролю.

Тестування бази даних проводиться для виявлення будь-яких слабких місць або вразливостей в конфігурації безпеки бази даних і для пом'якшення наслідків будь-якого небажаного доступу до бази даних.

Усі конфіденційні дані повинні бути захищені від зловмисників, тому регулярні перевірки безпеки дуже важливі й мають бути обов'язковими.

Нижче перераховані основні властивості даних, з яких тестування безпеки бази даних є обов'язковим:

- аутентифікація;
- авторизація;
- облік;
- конфіденційність;
- цілісність;
- доступність;
- стійкість.

Заходи протидії загальним ризикам та вразливостям

Для протидії описаним вище ризикам та вразливостям є різні методології. Кожна з них забазпечує в тій чи іншій мірі можливість для зменшення ризиків різними методами.

Перша методологія це тест на проникнення. Це навмисна атака на систему з метою знайти уразливості в системі безпеки, завдяки яким зловмисник може отримати доступ до всієї системи, включно з базою даних. Якщо вразливість знайдена, то важливо негайно усунути будь-яку загрозу, яка може викликати таку вразливість.

Другий метод протидії є процес проведення оцінки ризику для визначення рівня ризику, пов'язаного з типом впровадженої конфігурації безпеки бази даних, і можливості виявлення вразливості. Така оцінка зазвичай проводиться експертами з безпеки, які можуть проаналізувати ступінь ризику, пов'язаного з тим чи іншим процесом.

Третім методом є проведення аудиту безпеки, щоб оцінити політику безпеки і з'ясувати, чи дотримуються стандарти чи ні. Існують різні підприємства зі своїми особливими стандартами безпеки, і якщо ці стандарти встановлені, то від них уже не можна відмовитися. Якщо хтось не дотримується будь-яких з цих стандартів, то це буде вважатися серйозним порушенням [3].

Як приклад незахищеності інформаційної системи можна навести випадок з компанією UnderArmour (MyFitnessPal) у 2018 році, який призвів до витоку конфіденційної інформації понад 147 мільйонів користувачів. Також одним із найбільш скандальних випадків було повідомлення у 2016 році компанією Yahoo, що у 2013 році їх база даних була зламана і це призвело до витоку інформації понад 500 мільйонів акаунтів, які потім можна було знайти на «чорному» ринку [4].

Описані вище методи є загальними і можуть дещо відрізнятись від практичних методологій, оскільки вони повинні обирати для кожної конкретної ситуації.

1.2 Існуючі способи захисту баз даних

Перш за все необхідно зрозуміти, що немає стовідсоткового способу захисту від атак кіберзлочинців. Але дотримання та виконання усіх описаних нижче дій дозволяє знизити ризики для компаній бути зламаними.

Для того, щоб забезпечити надійний захист бази даних, необхідно дотримуватися таких основних правил:

- встановлення надійних паролів на свої облікові записи;
- створення резервних копій;
- використання безпечних і перевірених додатків;
- своєчасне оновлення системи, бо у системі можуть знаходитися експлойти, які надають доступ до бази даних.

Для посилення безпеки бази даних необхідно контролювати тих користувачів, хто має до неї доступ. Такі обмеження допоможуть знизити

можливість того, що доступ до інформації буде наданий кіберзлочинцю. Крім базових дозволів для користувачів слід застосувати таке:

- Обмежити доступ до чутливих даних для користувачів, які можуть робити запити, що пов’язані з конфіденційною інформацією.
- Уникати надання доступу до бази даних у неробочий час.
- Вимкнути усі служби та процедури, які не використовуються на даний час.
- Необхідно розміщувати базу даних на віддаленому сервері, який не має доступу до мережі Інтернет.

Серед основних методів захисту є шифрування інформації всередині бази даних надійним криптографічним алгоритмом [5]. Також безпека даних у СКБД забезпечується за допомогою перевірки відповідності запитів правилам безпеки, що зберігаються в системному каталозі. У більшості СКБД підтримується вибіркове управління доступом. При вибірковому управлінні даними користувачі мають доступ тільки до даних у межах свого представлення. Користувачі можуть застосовувати обмежений набір операцій до елементів даних. Користувачі мають різні повноваження при роботі з об'єктами. При доступі до об’єкта користувачі можуть мати різні повноваження. Вибіркове управління складається з таких компонентів:

- найменування правила;
- привілеї;
- діапазон застосування правил;
- дані користувачів, яким доступні ці привілеїв;
- реакція на порушення системних правил.

Виконуючи важливі операції, база даних повинна реєструвати контрольний слід цих операцій. Якщо існує підозра, що було виконано несанкціонований доступ до бази даних, тоді цей слід зможе допомогти при виявлені порушника. Контрольний файл містить в собі такі дані:

- Текст запиту;
- Термінал, з якого було надіслано запит;

- Дані користувача;
- Час і дата виконання запиту;
- Нові дані;
- Попередні дані.

Усі спроби доступу до бази даних повинні фіксуватися у системі та бути записаними до відповідного файлу.

При управлінні доступом кожен об'єкт в базі даних має класифікацію, наприклад: цілком таємно, таємно, для службового користування тощо. Кожен користувач у системі має рівень допуску, який відповідає його обов’язкам з такими ж градаціями, що і на рівні класифікації. При цьому підході доступ до даних певних об'єктів мають лише користувачі з необхідним рівнем допуску. При такому виді управлінні виконуються такі правила безпеки:

– користувач має право доступу до об'єкту лише у випадку, якщо його рівень доступу відповідає рівню класифікації об'єкта [6].

1.3 Постановка задачі

На основі проведеного аналізу було сформульовано наступні задачі:

1. Розробити механізм захисту для бази даних. Включити до уваги всі можливі вразливості та способи їх подолання.

2. Створити базу даних на обраній СКБД для вебдодатку. Протестувати безпеку та продуктивність при роботі створеної бази даних.

3. Провести більш детальне тестування вебдодатку (перед-релізне) на виявлення усіх, навіть найменших загроз цілісності, доступності та конфіденційності.

4. Виправити усі знайдені помилки, вразливості та реліз додатку.

2. ХАРАКТЕРИСТИКА МЕХАНІЗМІВ ЗАХИСТУ БАЗИ ДАНИХ

2.1 Системи керування базами даних та їх моделі

Відомо, що існує два типи баз даних: реляційні та нереляційні. Розглянемо більш детальніше особливості кожного типу.

Реляційний тип СКБД реалізує дані у вигляді таблиць, котрі пов’язані один з одним. У таких СКБД мова запитів є ядром системи, бо вона використовується для зв’язку з базою даних та керування нею. Дані в таблицях розташовані у строках та стовбцях зі строгою структурою. Зазвичай кожен запис має значення для кожного атрибуту, що призводить до чітких залежностей між таблицями.

У реляційних базах даних зазвичай дані зберігаються на одному пристрої, а масштабування здійснюється шляхом додавання додаткової потужності до цього одного сервера. Однак перехід від невеликих машин до більших часто призводить до простоїв. Масштабування бази даних SQL між кількома серверами може бути складним завданням, оскільки вимагає змін у структурі даних і додаткових інженерних зусиль. Також це сильно впливає на безпеку, оскільки потрібно захищати одразу 2 сервери.

Реляційні бази даних показують високу продуктивність завдяки інтенсивним операціям читання/запису для достатніх наборів даних. Також в таких базах можна покращити швидкість пошуку даних використовуючи індекси та об’єднання таблиць. Але варто пам’ятати, коли кількість даних і запитів зростає, швидкість будь-яких операцій може погіршитися.

Завдяки інтегрованій структурі та системі зберігання даних бази даних SQL не вимагають особливих інженерних зусиль, щоб забезпечити їх належний захист. Вони є хорошим вибором для створення та підтримки складних програмних рішень, де будь-яка взаємодія має ряд наслідків. Зазвичай усі реляційні СКБД мають надійну систему аутентифікації та розмежування прав доступу, що лише закріплює безпечність таких систем.

Нереляційний тип СКБД реалізує різні нетабличні моделі даних для зберігання те керування даними. Найпоширенішими є наступні моделі: документно-орієнтовний (для зберігання документів, наприклад json), ключ-значення (унікальний ключ, по якому знаходиться значення), graph (зберігання у структурі вузол-ребро-вузол), широкий стовпець (коли для кожного рядка можуть бути свої стовбці). Такі бази даних дозволяють зберігати неструктуровані дані. Наприклад: статичний текст, фотографії, відео, та інші файли. Тобто дані легко отримати, але структурності як у реляційних БД немає.

Зазвичай, коли кількість даних збільшується, нереляційні СКБД масштабуються горизонтально, тобто додаючи нові сервери в центр обробки даних. В такому випадку кожен сервер містить лише частину спільних даних, а тому і швидкість запитів буде гіршою. Також, треба зауважити що горизонтальне масштабування негативно впливає на безпеку.

Нереляційні бази даних відомі своєю високою продуктивністю при роботі з невеликим обсягом даних: вони мають розподілену конструкцію, що знижує навантаження на продуктивність системи та надає великій кількості користувачів одночасний доступ. Такі бази даних можуть зберігати необмежену кількість типів і форм самих даних. А ще вони досить гнучкі, коли справа доходить до заміни типів даних. Але, як сказано у попередньому абзаці, вся продуктивність погіршується при масштабуванні бази даних, тому робимо висновок, що при великих обсягах даних нереляційний тип використовувати не потрібно.

Нереляційні бази даних мають досить слабкий захист, і це є досить серйозною проблемою для багатьох компаній. Звичайно, деякі СКБД можуть надавати гарантії безпеки, але як показують відгуки, безпека все ж гірша, ніж у реляційного типу.

Проаналізувавши типи СКБД, можемо зробити висновок, що в нашому випадку ми будемо використовувати реляційний тип СКБД. По-перше, так буде краще з міркувань безпеки, а по-друге – з міркувань перспективної

продуктивності, оскільки база даних буде зберігати структуровані дані великого обсягу [11].

Проаналізуємо найпопулярніші СКБД, які в даний момент використовує переважна кількість IT-компаній та виділимо їх переваги та недоліки:

1) **MySql** – є найпоширенішою реляційною СКБД серед розробників. Вона є безкоштовною якщо ми говоримо про базовий набор інструментів, та має більш-менш просту та зрозумілу документацію. Оскільки це реляційна СКБД, то з гарантіями безпеки тут теж все добре. Також MySql ідеально підходить для будь-яких невеликих вебдодатків, оскільки мова php найкраще працює з даною СКБД. Недоліками є погана горизонтальна масштабованість, та ліцензування коду приватною компанією Oracle.
2) **MariaDB** – доопрацьована варіація реляційної СКБД MySql, яка працює під загальнодоступною ліцензією і має API подібне до MySql. Із плюсів можна виділити надійне шифрування даних, широка функціональність, якої іноді недостатньо в MySql, дуже висока продуктивність у порівнянні з батьківською розробкою. Із мінусів можно виділити лише низький рівень спільноти, що призводить до складного вирішення задач (оскільки складно знайти людину, яка уже вирішила певну задачу), та проблеми з міграційною сумісністю з MySql.
3) **Oracle** – одна з найпоширеніших комерційних реляційних СКБД. Вона повністю керується однойменною компанією, та має потужну документацію та технічну підтримку. Також Oracle є дуже ефективною та дозволяє обробляти величезну кількість даних. Із мінусів можна виділити високу вартість на послуги, а безкоштовні версії дуже обмежені по функціоналу. Також СКБД потребує дуже потужне обладнання, і не буде швидко працювати на слабких пристроях. І напевно найбільший мінус даної СКБД – це дуже складна документація, через що для розгортання бази через Oracle потрібні сертифіковані спеціалісти, які довго вивчали принципи роботи з даною СКБД.

4) **MongoDB** – є найпопулярнішою нереляційною СКБД. Вона безкоштовна (також є комерційна версія) та має відкритий вихідний код. Із плюсів можна виділити легкий та швидкий доступ до даних, оскільки дані дуже часто зберігаються в оперативній пам’яті. MongoDB має потужну сумісність з різними СКБД, як реляційними, так і нереляційними. До мінусів можна віднести дуже велике споживання оперативної пам’яті, відсутність внутрішньої авторизації для доступу до даних.
5) **Redis** – зазвичай використовується для тимчасових даних оскільки записує дані у вигляді ключ-значення. Серед переваг є швидкість та массовість обробки запитів, оскільки дані можуть зберігатися в оперативній пам’яті. Але це також є і недоліком, бо велика база даних буде потребувати великого об’єму оперативної пам’яті. Також СКБД не має підтримки мови запитів, що робить її несумісною з реляційними БД [12].

2.2 Модель вебдодатку та типи даних

Оскільки планується передбачати захист даних вебдодатку, який буде використовувати для взаємодії з базою даних мову PHP, а самі дані будуть реляційної моделі, прийнято рішення використовувати СКБД MySql. Обрана СКБД є найпопулярнішою у світі та має досить надійну внутрішню систему безпеки. Також не виникатиме проблем з ліцензуванням, оскільки вона є безкоштовною.

Кожен вебдодаток має як публічні, так і конфіденційні дані. В нашому випадку вебдодаток буде мати наступні дані:

Публічні:

1) Статичні дані, які використовуються в логіці побудови html сторінок.
2) Пости в глобальній стрічці, їх вміст, нікнейм автора та картинки.
3) Нікнейми користувачів для пошуку профілів.

Конфіденційні:

1) Паролі користувачів.

2) Справжні прізвище та ім’я користувача, його місто проживання та дата народження.
3) Налаштування конфіденційності.
4) IP-адреси реєстрації та авторизації.
5) Дата реєстрації та авторизації.

Варто зауважити, що деякі дані можуть зберігатися у вигляді хешу, що унеможливлює використання даних навіть після кібератаки. Тому паролі користувачів зберігаються у вигляді хешу.

2.3 Механізми захисту бази даних

Грунтуючись на переваги обраної СКБД та можливий функціонал, розробимо певні механізми захисту для бази даних, що створюватиметься.

Захист від зовнішніх підключень

Як відомо, для того щоб кіберзлочинцю отримати будь-які дані безпосередньо з бази даних, необхідно пройти аутентифікацію. Захист паролем це типове рішення, але не є достатньо надійним. Тому першочергово забезпечимо доступ до бази даних лише з локальної мережі. У даному прикладі будемо розглядати команди для сервера на базі операційної системи Linux (Ubuntu 20.04).

По-перше, нам потрібно закрити доступ до порту MySQL сервера – за замовчуванням 3306. Зробити ми це можемо через стандартний інструмент ubuntu – UWF. Наведені нижче команди (рис. 5) забороняють доступ до порту 3306 від усіх джерел.

```
jekmant@jekmant:~$ sudo ufw deny 3306/tcp
[sudo] password for jekmant:
Rules updated
Rules updated (v6)
jekmant@jekmant:~$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
```

Рисунок 5 – Заборона усіх зовнішніх підключень до бази даних

Застосування цих команд обмежить доступ для адміністраторів бази, тому зробимо наступне: розгорнемо корпоративний VPN (Virtual Private Network) зі статичною IP-адресою, якій і надамо доступ для підключення до mysql серверу. Щоб надати доступ для адреси вставимо нове правило, як показаною на рис. 6.

```
jekmant@jekmant:~$ sudo ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] Nginx HTTP                 ALLOW IN    Anywhere
[ 2] 3306/tcp                   DENY IN     Anywhere
[ 3] Nginx HTTP (v6)            ALLOW IN    Anywhere (v6)
[ 4] 3306/tcp (v6)              DENY IN     Anywhere (v6)

jekmant@jekmant:~$ sudo ufw insert 2 allow from 192.168.50.241 to any port 3306
Rule inserted
jekmant@jekmant:~$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
```

Рисунок 6 – Додаємо адресу корпоративного VPN у білий список

Завдяки даному способу ми унеможливили підключення хакеру до MySQL серверу. Перед тим як він зможе це зробити, йому потрібно отримати дані для авторизації до корпоративної VPN мережі. А в процесі авторизації до неї ми можемо легко використовувати будь-які методи двухфакторної аутентифікації, що ще більше знизить ризик несанкціонованого доступу.

Аудит та регулярний моніторинг

Завдяки тому, що всі дії можуть виконуватись лише через VPN, ми можемо легко відслідковувати який користувач в який час входив до мережі, та навіть відслідковувати останній трафік до 3306 порту.

Резервне копіювання даних

Регулярне збереження даних на інший носій або спеціалізований хмарний сервер забезпечить від втрати інформації у випадку будь-яких інцидентів.

Жорсткий контроль доступу

Для реалізації даного механізму нам потрібно при створенні користувача в СКБД видавати йому лише ті права, які потрібні. Так наприклад: для користувача, дані якого буде використовувати лише вебдодаток ми надамо права лише на команди SELECT, INSERT, UPDATE, DELETE, як показано на рис. 7.

```
mysql> grant select, update, insert, delete on univer.* to 'website'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

Рисунок 7 – Надання лише потрібних прав користувачу

Захист вебдодатку

Для захисту бази даних нам потрібно забезпечити і захист самого додатку, який взаємодіє з базою. Для цього в коді використано наступні механізми:

- Максимальний захист від SQL ін’єкцій. В коді вебдодатку часто використовується функція real_escape_string, яка екранує усі спеціальні вхідні символи. На рис. 8 показано код, який протидіє даній вразливості.

```
$sql = sprintf("SELECT * FROM users WHERE user_login = '%s' LIMIT 1",
    $this -> db -> real_escape_string($user_login));
```

Рисунок 8 – Захист від sql ін’єкцій

- Захист від XSS ін’єкцій та інших ризиків, пов’язаних зі шкідливими даними, які передаються від користувача. На рис. 9 показано фрагмент коду, який перевіряє вхідні дані.

```
$param = trim($param);
$param = htmlspecialchars($param, ENT_QUOTES);
$param = htmlentities($param, ENT_QUOTES);
$param = strip_tags($param);
$param = addslashes($param);
```

Рисунок 9 – Захист від XSS

- Захист від підміни сесії. Даний метод дозволить уникнути ситуацій, коли жертву «піймали на гачок» та дізнались ідентифікаційний номер її сесії. Тобто підміна сесії через будь-які редактори cookie буде неактуальною.

3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО РІШЕННЯ

3.1 Вибір програмних засобів і опис програмної реалізації

Програмні засоби

Для реалізації вебдодатку обрано серверний дистрибутив Ubuntu 20.04, який заснований на операційній системі GNU/Linux. В самій системі встановлено Nginx, PHP версії 8.1 та MySQL-server. Для зручності сервером виступає віртуальна машина, яка запущена програмою Oracle Virtual Box.

При налаштуванні MySQL-server встановлюємо політику безпеки паролів на максимальний рівень. Це забезпечить надійність паролів користувачів, оскільки буде потребувати пароль з малими та великими літерами, цифрами та спеціальними символами.

Далі налаштовуємо мережеву безпеку. Закриваємо зовнішній доступ до порту MySQL сервера (3306), після чого розгортаємо корпоративний VPN сервер та надаємо доступ для підключення з IP-адреси даного VPN. Для самого VPN-серверу налаштовуємо двухфаторну аутентифікаію за допомогою сервісу Google Authenticator.

Після налаштувань мережевої безпеки починаємо створювати профілі для MySQL серверу. Першим створюємо профіль розробника БД, якому надаються усі права доступу, а вже після – профіль для вебдодатку, до прав якого входить лише редагування даних та читання таблиць. В третю чергу створюються профілі для інших розробників та надаються ті привілеї, які їм потрібні для роботи.

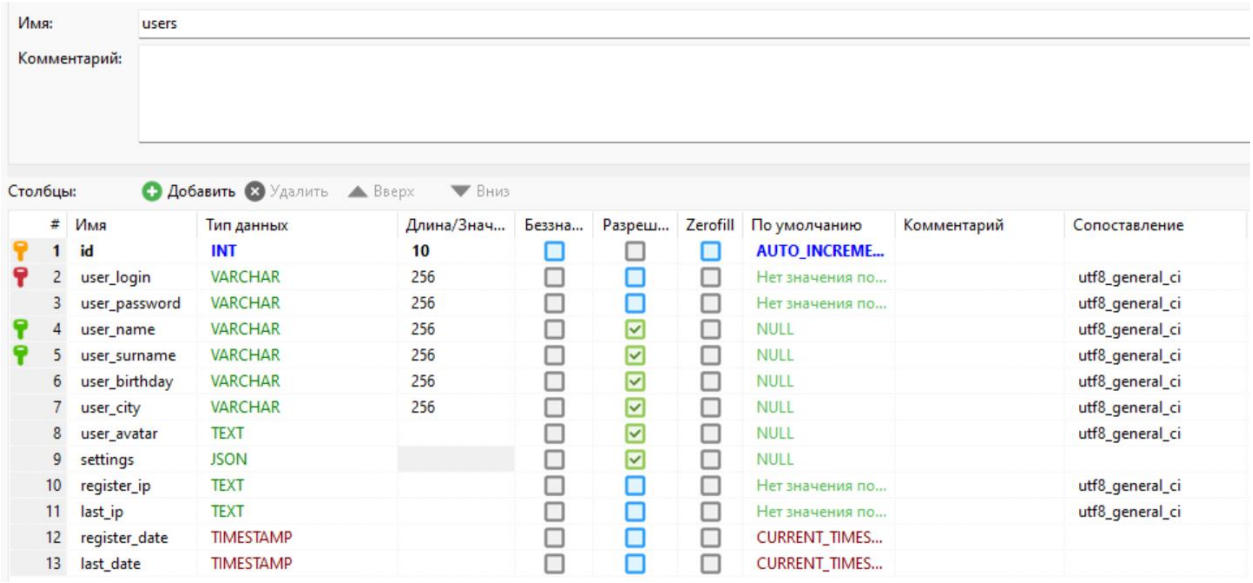
Для адміністрування базою даних обрана програма HeidiSQL. Але використовувати можна будь-який інший спеціалізований додаток, оскільки йде підключення до SQL серверу.

Опис програмної реалізації

Сутність вебдодатку – невелика соціальна мережа з можливістями писати публічні повідомлення, переглядати профілі інших користувачів мережі. Також є функція для пошуку профілів, щоб було зручніше знайти

потрібних користувачів. Обов'язково передбачена функція авторизації та реєстрації користувачів. Враховуючи, що це соціальна мережа, кожен користувач може вказати потенційно конфіденційні дані та налаштувати, що саме буде відображатись для інших користувачів. Саме тому в даному вебдодатку ми маємо публічну інформацію (повідомлення користувачів), конфіденційну інформацію (паролі, IP-адреси, налаштування) та умовно конфіденційну (залежить від налаштувань кожного користувача).

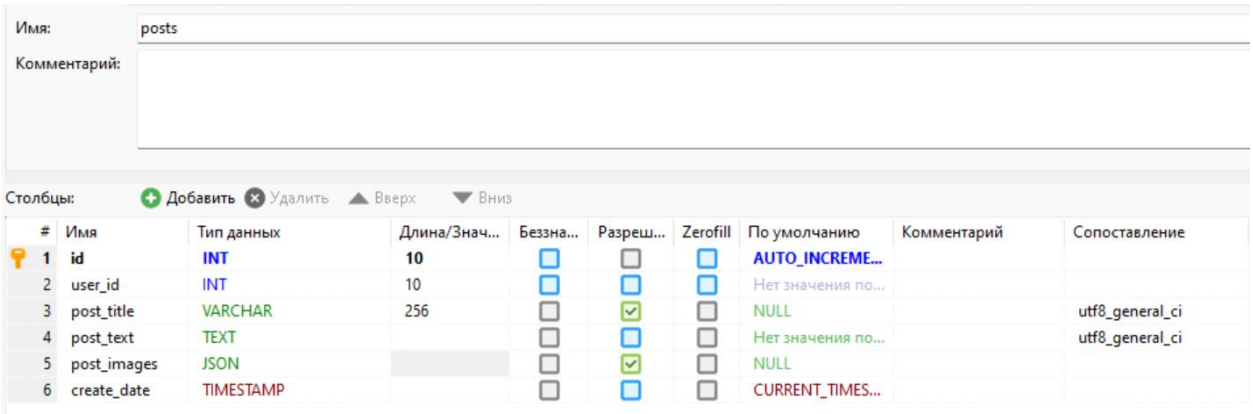
Розглянемо таблиці в базі даних. На рис. 10 показано структуру таблиці *users*. В ній зберігаються усі дані про користувачів.



#	Имя	Тип данных	Длина/Знач...	Беззна...	Разреш...	Zerofill	По умолчанию	Комментарий	Сопоставление
1	id	INT	10	☐	☐	☐	AUTO_INCREME...		
2	user_login	VARCHAR	256	☐	☐	☐	Нет значения по...		utf8_general_ci
3	user_password	VARCHAR	256	☐	☐	☐	Нет значения по...		utf8_general_ci
4	user_name	VARCHAR	256	☐	☑	☐	NULL		utf8_general_ci
5	user_surname	VARCHAR	256	☐	☑	☐	NULL		utf8_general_ci
6	user_birthday	VARCHAR	256	☐	☑	☐	NULL		utf8_general_ci
7	user_city	VARCHAR	256	☐	☑	☐	NULL		utf8_general_ci
8	user_avatar	TEXT		☐	☑	☐	NULL		utf8_general_ci
9	settings	JSON		☐	☑	☐	NULL		
10	register_ip	TEXT		☐	☐	☐	Нет значения по...		utf8_general_ci
11	last_ip	TEXT		☐	☐	☐	Нет значения по...		utf8_general_ci
12	register_date	TIMESTAMP		☐	☐	☐	CURRENT_TIMES...		
13	last_date	TIMESTAMP		☐	☐	☐	CURRENT_TIMES...		

Рисунок 10 – Структура таблиці users

На рис. 11 показана структура таблиці *posts*. В ній зберігаються публічні повідомлення усіх користувачів.



#	Имя	Тип данных	Длина/Знач...	Беззна...	Разреш...	Zerofill	По умолчанию	Комментарий	Сопоставление
1	id	INT	10	☐	☐	☐	AUTO_INCREME...		
2	user_id	INT	10	☐	☐	☐	Нет значения по...		
3	post_title	VARCHAR	256	☐	☑	☐	NULL		utf8_general_ci
4	post_text	TEXT		☐	☐	☐	Нет значения по...		utf8_general_ci
5	post_images	JSON		☐	☑	☐	NULL		
6	create_date	TIMESTAMP		☐	☐	☐	CURRENT_TIMES...		

Рисунок 11 – Структура таблиці posts

Тепер, коли ми маємо уявлення про структуру даних вебдодатку, подивимось які сторінки доступні для користування. На рис. 12 показано головну сторінку – користувач не авторизований. На ній можна побачити шапку, місце для пошуку користувачів, повідомлення користувачів та копірайт.

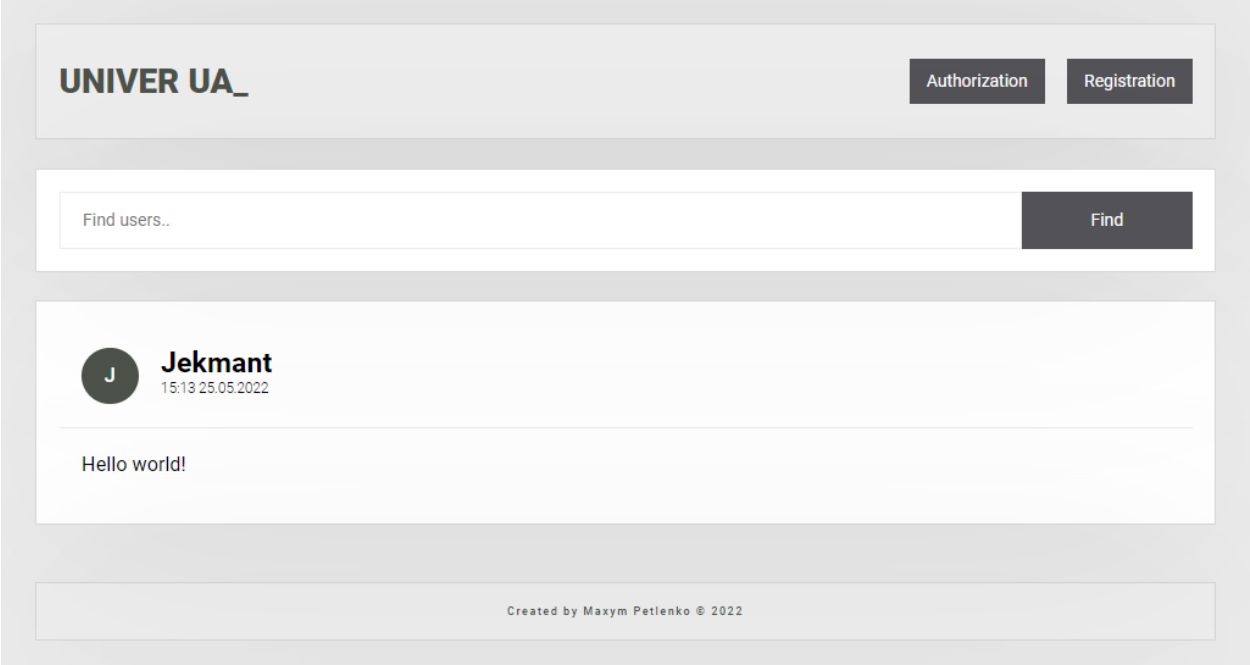


Рисунок 12 – Головна сторінка вебдодатку

Вебдодаток має реєстрацію та авторизацію користувача. На рис. 13 показано сторінку реєстрації.

На рис. 14 продемонстровано сторінку авторизації.

Після того як користувач успішно пройшов процедуру реєстрації та авторизації, відкривається головна сторінка вебдодатку, але дещо інакше виглядає. На рис. 15 показано головну сторінку для авторизованого користувача. Як бачимо, користувач може писати свої глобальні повідомлення, а також вийти з профіля користувача.

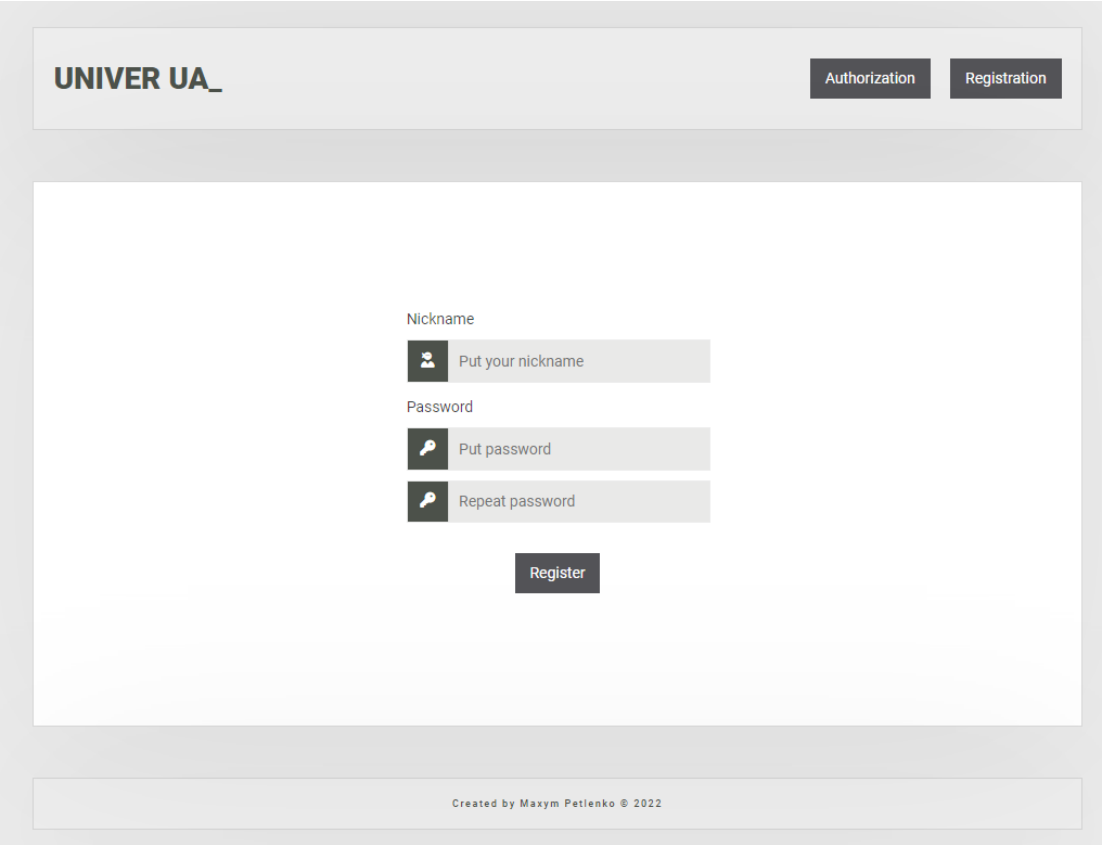


Рисунок 13 – Сторінка реєстрації користувача

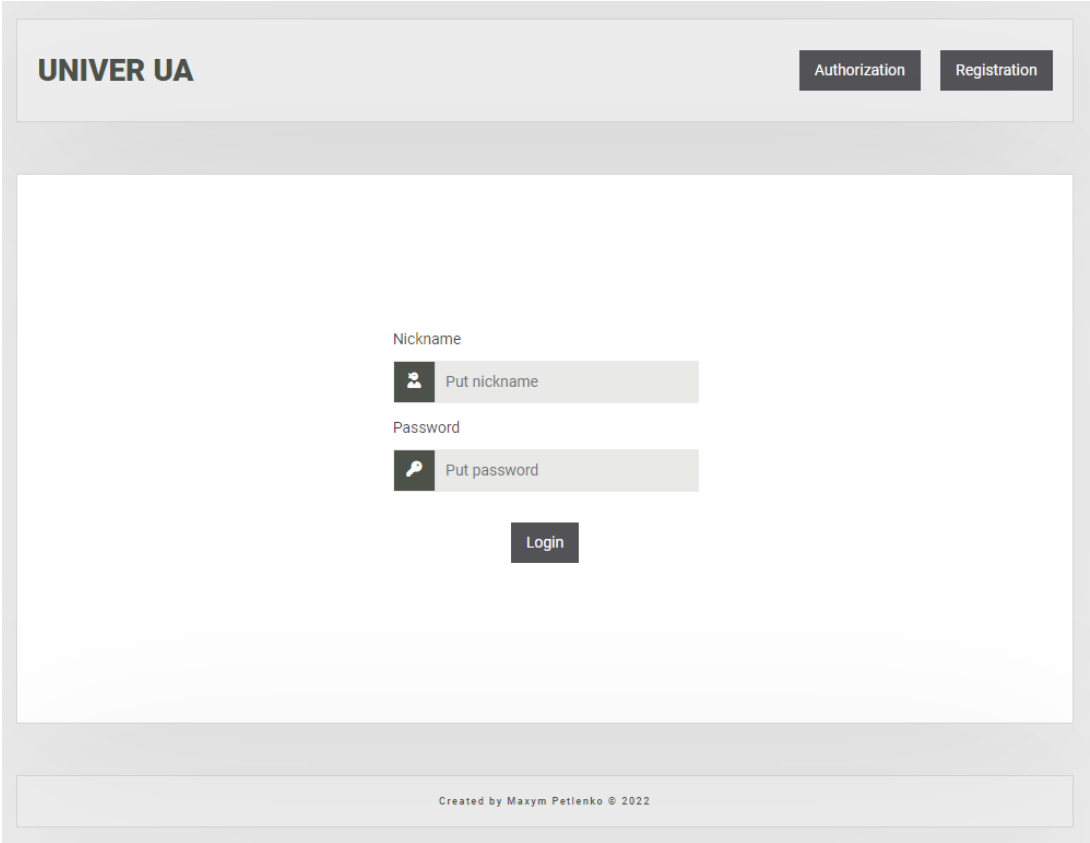


Рисунок 14 – Сторінка авторизації користувача

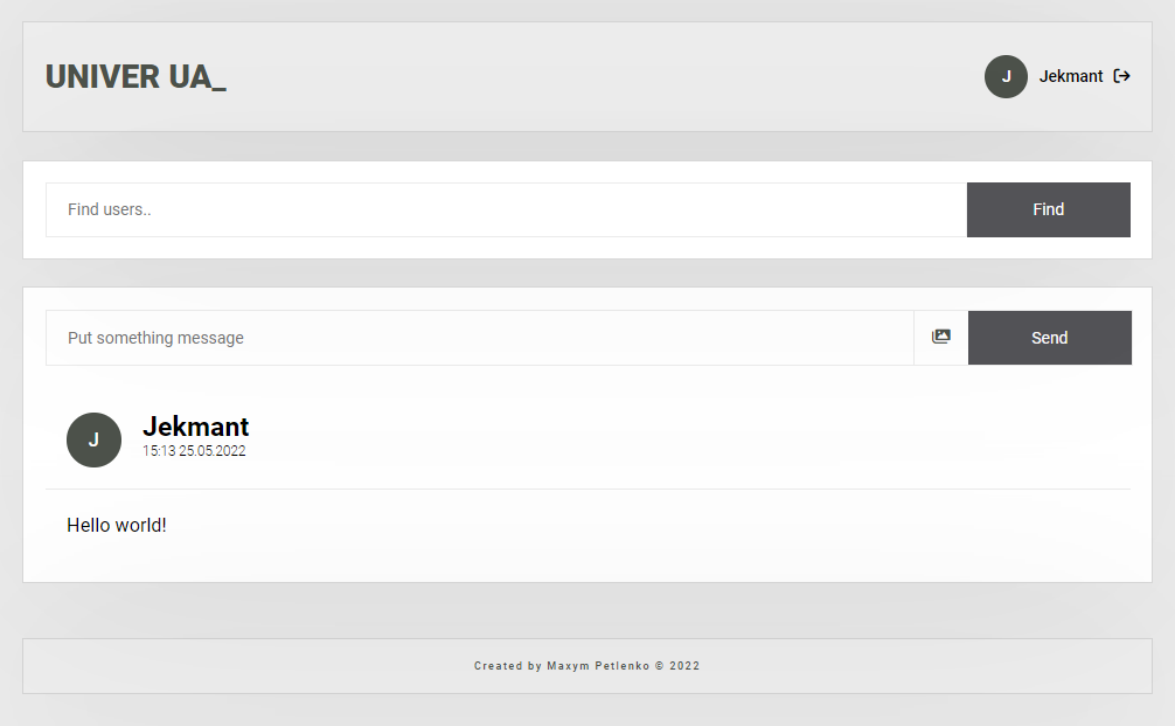


Рисунок 15 – Головна сторінка для авторизованої особи

Також вебдодаток дозволяє переглядати профілі інших користувачів. Подивитися профіль іншого користувача можна за допомогою пошуку або обравши користувача з глобальної стрічки. На рис. 16 зображено профіль тестового користувача.

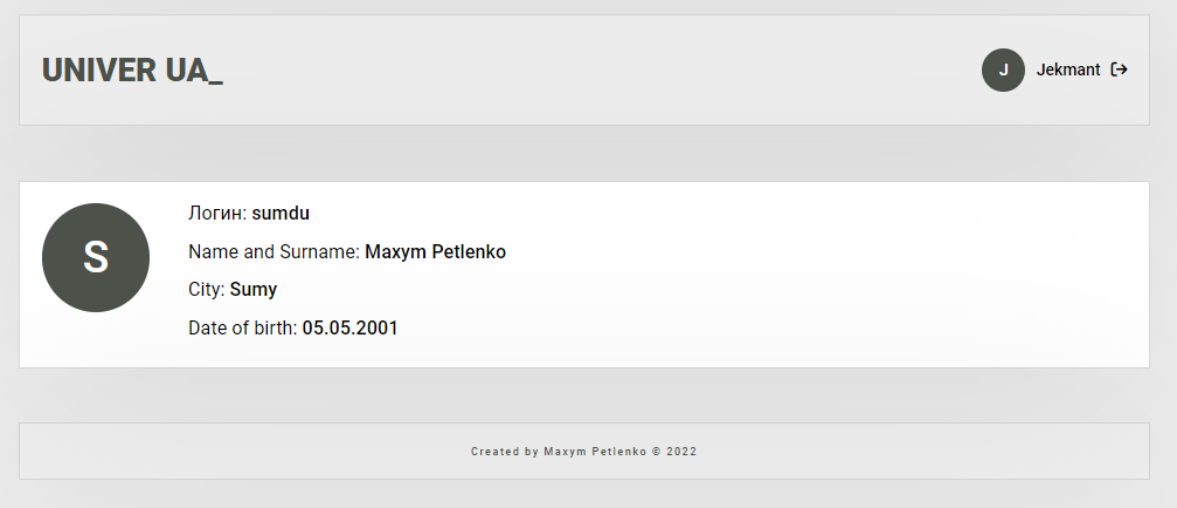


Рисунок 16 – Профіль користувача

Як ми можемо побачити, в профілі відображається багато потенційно конфіденційної інформації користувача. Але якщо її видно, то користувач сам цього бажає, бо при налаштуванні профілю можна вказати, що саме показувати іншим, а що не показувати. На рис. 17 продемонстровано саме такі налаштування профілю.

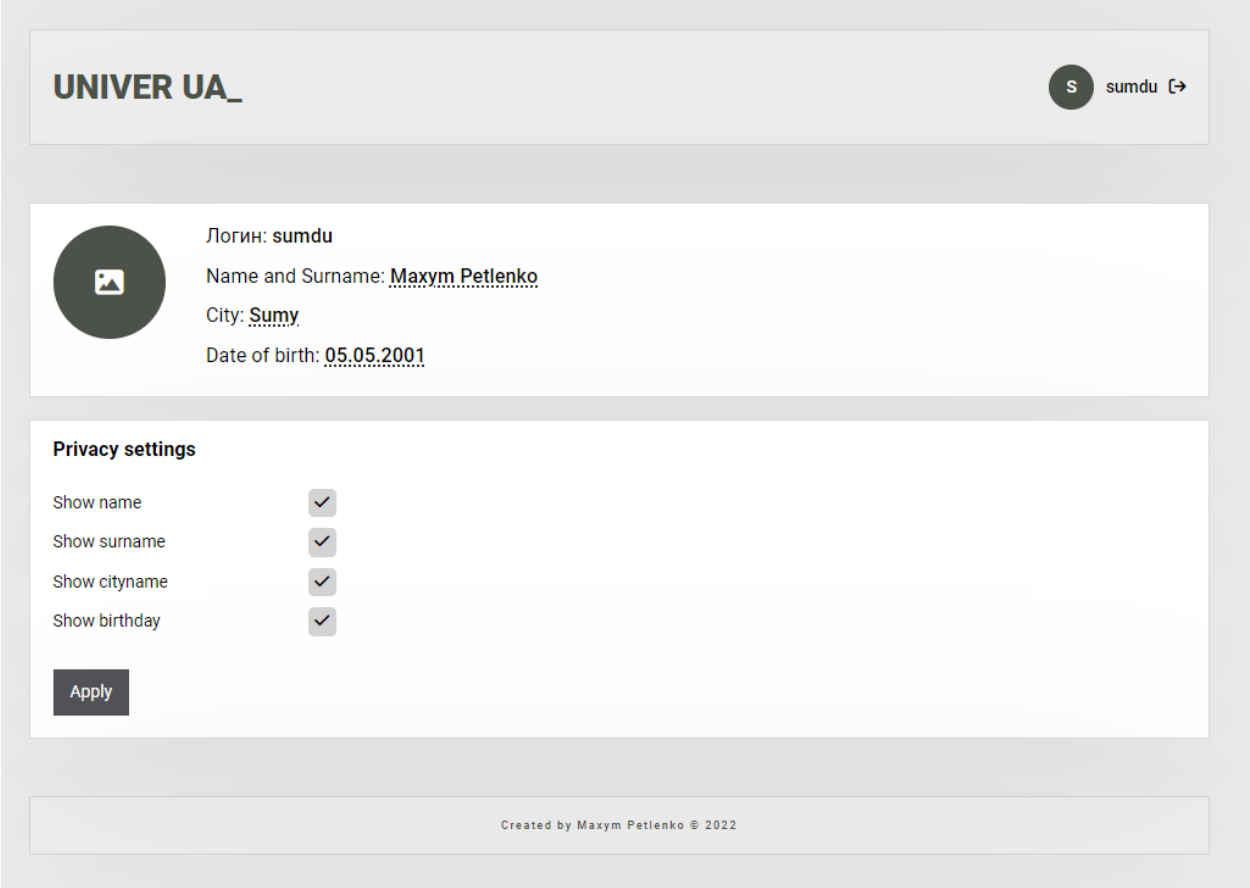


Рисунок 17 – Налаштування профілю користувача

Встановимо заборону вебдодатку показувати місто та дату народження. На рис. 18 показано профіль користувача після зміни налаштування конфіденційності.

Також зміни налаштувань впливають на пошук, відображаючи лише доступні дані. На рис. 19 показано результат пошуку, коли в налаштуваннях відкриті лише їм’я та прізвище.

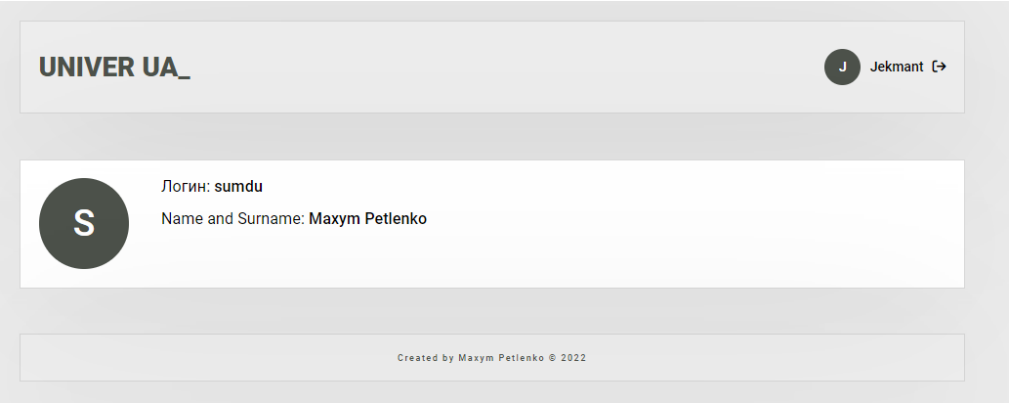


Рисунок 18 – Профіль зі зміненими налаштуваннями конфіденційності.

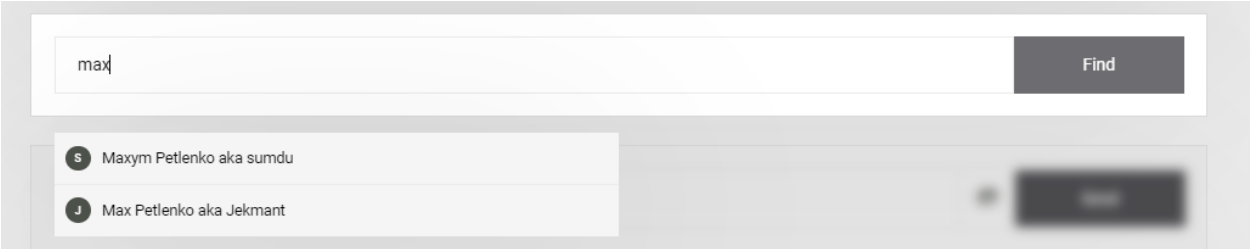


Рисунок 19 – Результат пошуку за іменем.

Але, якщо заборонити доступ до імені, то профіль не буде знайдено. На рис. 20 продемонстровано саме такий випадок.

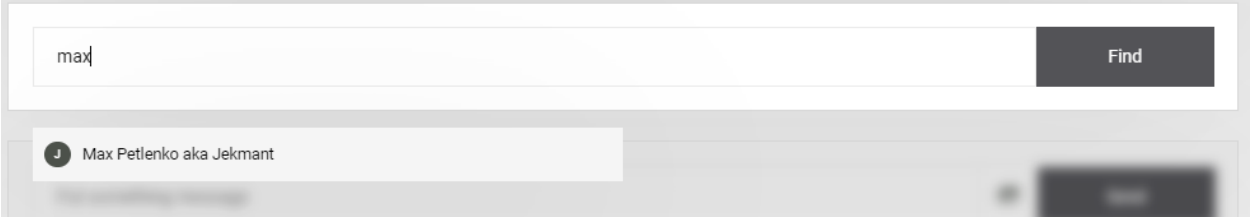


Рисунок 20 –Результат пошуку за іменем зі зміненими налаштуваннями.

Отже, ми розробили вебдодаток та базу даних для нього, використавши певні методології для забезпечення максимальної безпеки. Тому, на наступному кроці проведемо тестування безпеки бази даних.

3.2 Опис тестування безпеки баз даних

Тест на SQL-ін’єкції дозволяє визначити можливість виконання SQL запитів, якими керує користувач вебдодатку. Вразливість виявляється, якщо введені користувачем дані використовуються для SQL запитів без належної обробки. При успішному використанні вразливості вебдодаток дозволяє неавторизованому користувачеві отримати доступ до бази даних та до змін даних у ній.

Процес впровадження ін’єкції складається з додавання шкідливого коду до введених даних або іншу передачу цих даних до вебдодатку (наприклад через cookie). Успішна атака дає змогу отримати доступ до конфіденційної інформації та можливості змінювати її. Дана атака змінює логіку роботи попередньо визначеної у вебдодатку команди.

Усі SQL запити, написані програмістами, є попередньо визначеними та виконують свої конкретні задачі, але вони включають дані, що вводяться користувачем. Так, наприклад, команда

SELECT * FROM users WHERE user_login = '%s' LIMIT 1

має змінну %s, замість якої додаються дані, введені користувачем. Тому будь-який користувач може вставити дані такого типу «’ or 1=1 #». Якщо користувач спробує ввести такі символи, то запит буде мати вигляд:

SELECT * FROM users WHERE user_login = '' or 1=1 #' LIMIT 1.

Оскільки маємо символ «#» в кінці даних, ми автоматично відтинаємо праву частину запиту і маємо наступний SQL код:

SELECT * FROM users WHERE user_login = '' or 1=1.

Даний запит вже значно відрізняється від того, який визначений розроюником і може давати у відповідь конфіденційні дані, які користувач не повинен був отримати.

У подібних атак є декілька різних класів, кожен з яких відрізняється:

1) Внутрішньосмуговий клас, при якому дані шкідливого запиту відображаються на вебсторінці програми або зберігаються в JavaScript коді.
2) Позасмуговий клас, при якому дані шкідливого запиту не відображаються, але передаються будь-яким іншим шляхом. Наприклад, через електронну адресу.
3) Сліпий клас, коли дані не відображаються, але є можливість змінювати дані та просто спостерігати за поведінкою БД.

Успішна атака вимагає чіткої постановки SQL запиту, і це є основною проблемою для зловмисника. Тому дуже добре, якщо програма приховує

деталі усіх помилок від користувача, бо інакше зловмиснику буде легше побудувати логіку запиту.

Якщо говорити про цілі тестування, то ми маємо наступне: нам потрібно визначити точки коду, де виконуються SQL запити, оцінити вірогідність успішної ін’єкції та визначити потенційні наслідки ін’єкції, тобто дані, у яких може бути порушена конфіденційність, цілісність да доступність.

Першим кроком виявляємо точки взаємодії вебдодатку з базою даних. Зазвичай це форми авторизації, реєстрації, пошуку, що у випадку нашого вебдодатку наявне. Також не забуваємо, що вебдодаток зчитує інформацію з GET та POST параметрів, переданих безпосередньо із браузеру.

Тест на можливість підключення до MySQL серверу

Потенційною вразливістю є сама можливість підключення до MySQL серверу, оскільки це відркиває злочинцю можливість зламу паролю користувача. Для тестування можна використати будь-який MySQL клієнт та спробувати під’єднатись до серверу. Найчастішим використовуваним портом для MySQL серверу є 3306.

Тест на викрадення сесії або підміну її даних

Викрадення сесії або її підміна є вразливістю вебдодатку, але впливає на дані в базі даних, оскільки зловмисник може отримати доступ та право на редагування даних без аутентифікації.

В мові PHP є вбудована система сесій, яка має низький рівень захисту, що спричиняє нові ризики. Кожен користувач може при бажанні подивитися свій ідентифікатор сесії, а спеціалізовані програми можуть його спокійно підмінити, оскільки зберігається він у вигляді cookie. Недосвідчений користувач може спокійно показати зловмиснику даний ідентифікатор без підозр, що до його профілю можна отримати доступ за допомогою нього.

На рис. 21 показано як виглядає вікно з інформацією про сесію в браузері Google Chrome.

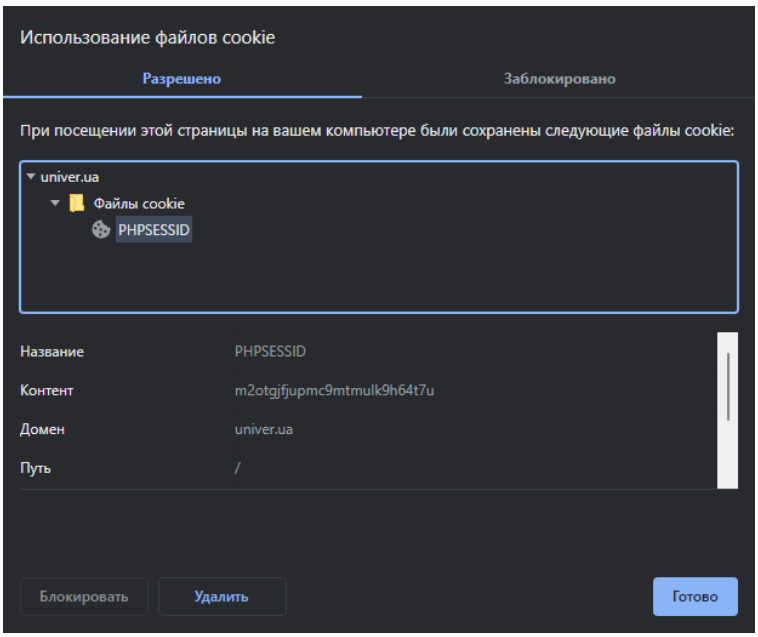


Рисунок 21 – Ідентифікатор сесії в браузері.

Для тестування вразливості нам потрібно лише 2 різних браузери. В одному з них проходимо авторизацію і отримуємо ідентифікатор сесії, а через інший браузер та додаток CookieManager ми можемо змінити ідентифікатор у другому браузері. Якщо в іншому браузері ми будемо авторизованими – вразливість виявлені і вона є критичною. Якщо ні – вебдодаток захищений від викрадення або підміни сесії.

Автоматизиваний тест за допомогою утиліти Owasp ZAP

Ще одним досить ефективним інструментом для тестування є утиліта Owasp ZAP (Zed Attack Proxy)[7]. Утиліта здатна провести автоматизовану атаку на вебдодаток з метою комплексного пошуку та виявлення вразливостей. Так, можливо вона не настільки ефективна, порівняно з ручним пошуком, але є достатньо швидкою та виявляє цілу низку можливих вразливостей. До них входить:

- SQL-ін’єкції;
- Порушення аутентифікації;
- Розкриття деяких критичних системних даних;
- Порушений контроль доступу;
- Міжсайтовий скриптинг (XSS);
- Використання бібліотек з відомими вразливостями;

- Інші неочевидні вразливості, які людина могла не помітити.

Інтерфейс програми досить простий і для проведення тестування не потребує значних зусиль. Усі вразливості відображаються на вкладинці «Сповіщення» після закінчення тестування.

3.3 Результати тестування та протидія виявленим вразливостям

Тест на SQL-ін'єкції є першочерговим у сфері вразливостей вебдодатків. Вебдодаток має декілька форм для введеня даних. Перше це реєстрація/авторизація, друге – пошук користувачів, третє – налаштування профілю. В теорії на форму авторизації можна застосувати внутрішньосмуговий клас атаки, але в нашому випадку вона не спрацювала, оскільки нам не вдалось пройти авторизацію без паролю або викликати помилку запиту, навіть знаючи формат запросу до БД. На рис. 22 показано спробу ін'єкції через поле імені користувача. При вдалій ін'єкції був би знайдений профіль. Захист поля з паролем також витримав перевірку.

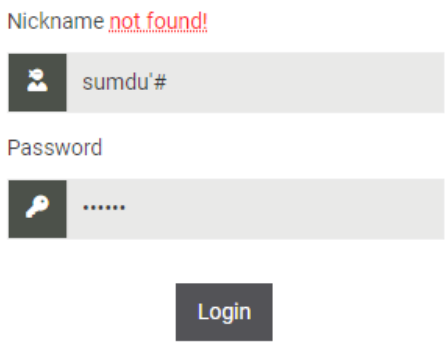


Рисунок 22 – Перевірка форми авторизації на ін'єкції.

Наступним об'єктом дослідження є форма реєстрації. На відміну від форми авторизації, форма реєстрації здатна вносити зміни в базу даних, оскільки додає дані до неї, а не лише зчитує. Але, як ми переконуємося на рис. 23, вебдодаток окрім вдалої реєстрації користувача з іменем sumdu'# додатково екранує спеціальні символи, які захищають вебдодаток від XSS атак.

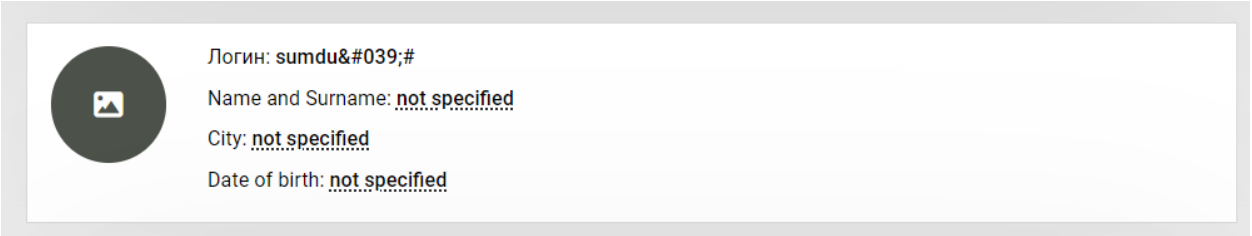


Рисунок 23 – Результат створення профілю зі спеціальними символами.

Вебдодаток має серед функцій і пошук користувачів на головній сторінці. Внутрішньосмугова атака також не дала результатів при введені шкідливих даних в поле для пошуку, та знаходить профіль, зареєстрованого вище користувача зі спеціальними символами. Також результату не дало і введення шкідливих даних в налаштуваннях профілю, через що залишилось останнє місце для реалізації атаки.

Для перегляду профілей інших користувачів використовуються вхідні дані браузеру, а саме таким чином передається параметр id через GET запит. Для реалізації сліпої атаки спробуємо модифікувати запит і додати шкідливі спеціальні символи. В результаті вебдодаток відкрив налаштування власного профілю, що свідчить про невдалу атаку. На рис. 24 показано спробу атаки через GET запит.

Тест на можливість підключення до MySQL серверу є досить важливим, оскільки саме уявлення про розташування MySQL серверу в мережі є значною вразливістю, оскільки допомагає реалізувати такі атаки, як brute-force, DDoS. Для тестування ми візьмемо IP-адресу веб-серверу та проаналізуємо відкриті порти. Для сканування можемо використати будь-який сканер портів. Для прикладу було використано «Advanced Port Scanner». На рис. 25 показано результат аналізу сканування.

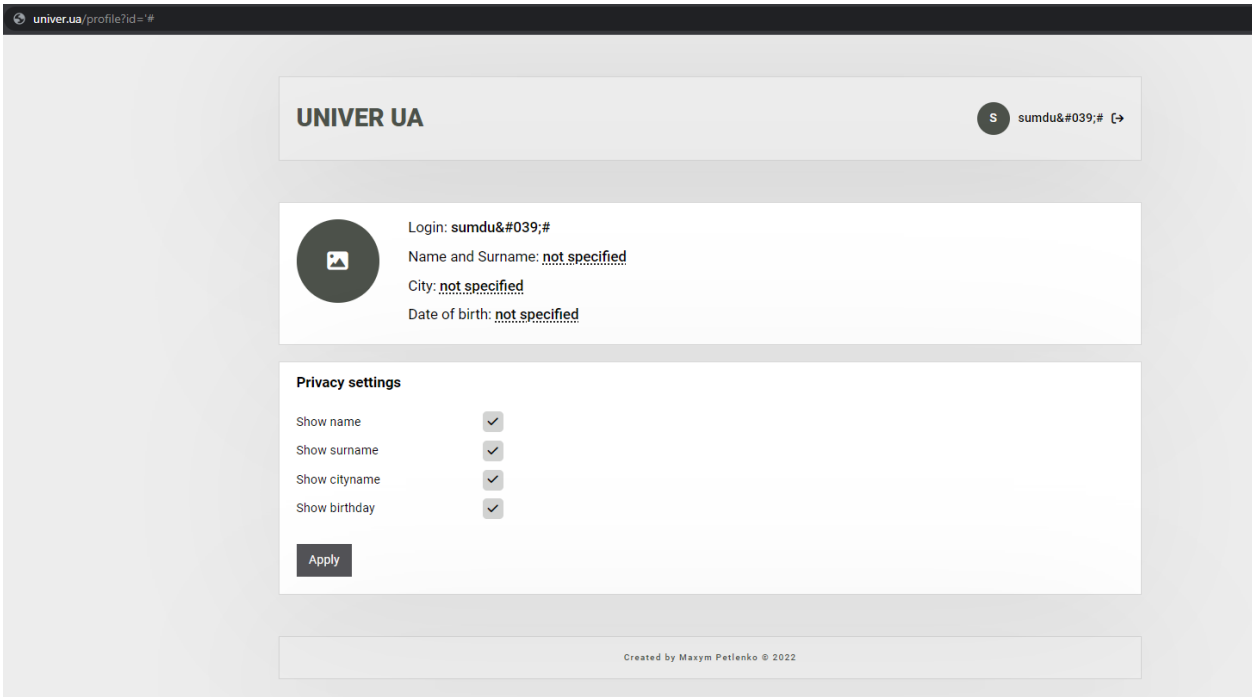


Рисунок 24. – Невдала спроба атаки через GET запит

Служба	Сведения
HTTP	Feed - Web app - Jekmant (nginx 1.18.0 Ubuntu)
Port **22** (TCP)	OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 Ubuntu Linux; protocol 2.0
Port **80** (TCP)	nginx 1.18.0 Ubuntu

Рисунок 25. – Результат сканування портів.

Як бачимо, порт для підключення до MySQL серверу закритий, але відритий порт 22. Даний порт використовується для SSH підключень і є вразливим не менше за порт для MySQL серверу. Тому його необхідно закрити. На рис. 26 показано команди для обмеження доступу до порту 22.

```
jekmant@jekmant:~$ sudo ufw deny 22
[sudo] password for jekmant:
Rule added
Rule added (v6)
jekmant@jekmant:~$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
```

Рисунок 26. – Закриття порту 22 від зовнішніх підключень

Після описаних вище дій маємо зміни у результаті сканування, що зображено на рис. 27. Як бачимо, на цей раз тест успішно завершено, вразливостей не виявлено і порт 22 вже не є відкритим.

Служба	Сведения
HTTP	Feed - Web app - Jekmant (nginx 1.18.0 Ubuntu)
Port **80** (TCP)	nginx 1.18.0 Ubuntu

Рисунок 27. – Результат сканування портів після зміни налаштувань.

Тест на викрадення сесії або підміну її даних є наступним за планом та може виявити вразливість, через яку зловмисник може змінити сессію та увійти до аккаунту іншого користувача без знання паролю. Для тестування потрібно: авторизуватись з одного пристрою; скопіювати дані cookie та підмінити їх в іншому. Якщо інший пристрій відобразить сесію вже авторизованого користувача – тестування виявило досить серйозну вразливість. На рис. 28 показано cookie після авторизації на одному пристрої.

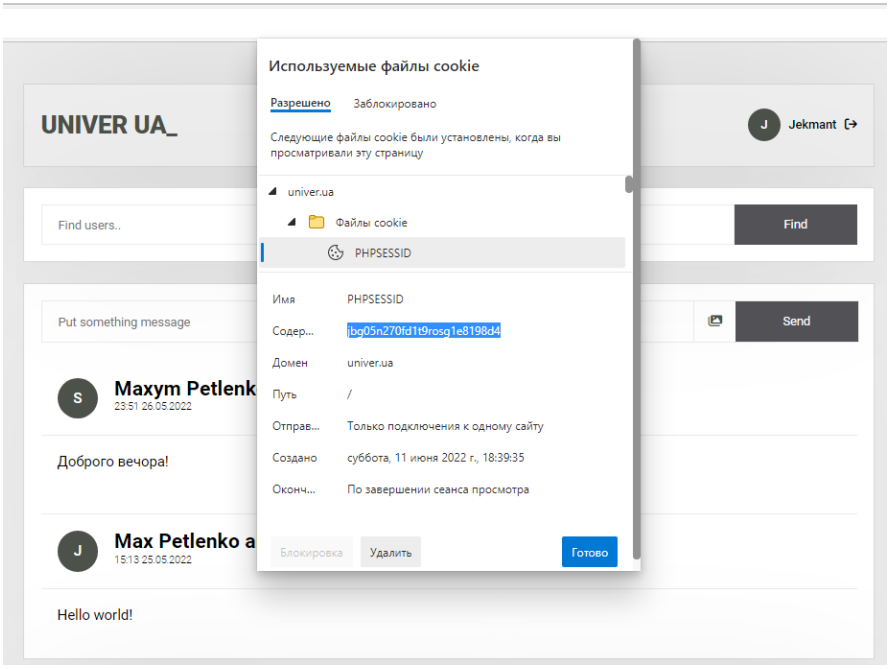


Рисунок 28. – Авторизація на легітимному пристрої та огляд його cookie.

Далі копіюємо cookie під назвою “PHPSESSID” та підміняємо його дані на іншому пристрої. На рис. 29 показано інтерфейс, за допомогою якого підмінили дані.

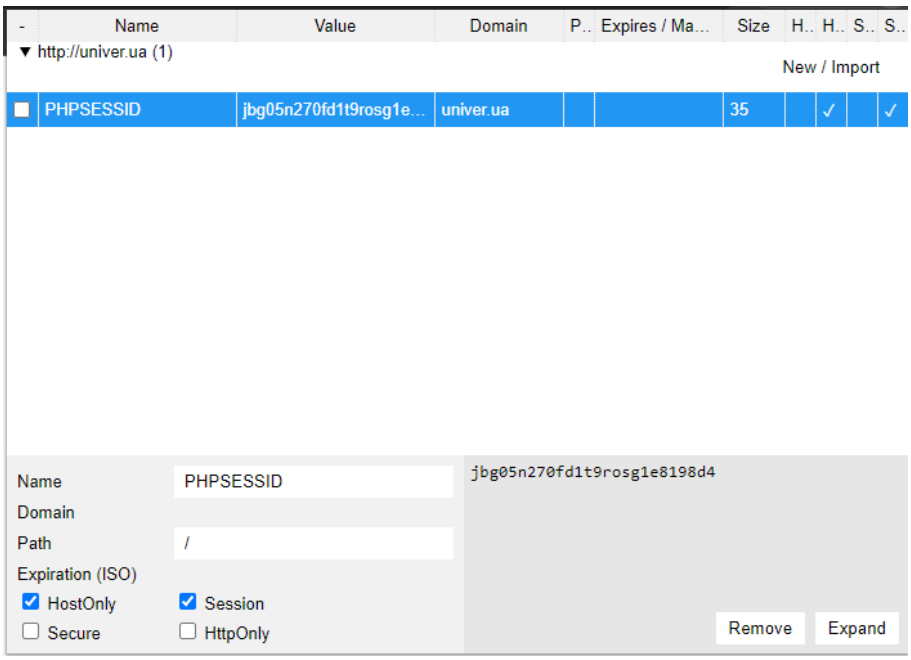


Рисунок 29. – Підміна даних в Cookie Manager.

За результатом тестування можна сказати, що підміна сессії не працює, оскільки одразу після цього на другому пристрої виникла помилка, яка продемонтрована на рис. 30.

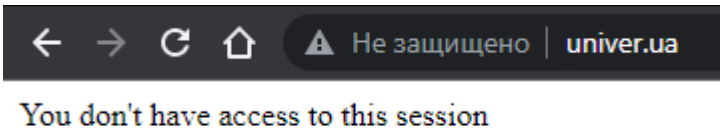


Рисунок 30. – Помилка доступу до сессії.

Автоматизиваний тест за допомогою утиліти Owasp ZAP допоможе знайти інші вразливості, які могли залишитись після ручного тестування. Оскільки інтерфейс програми інтуітивно зрозумілий, одразу перейдемо до результатів автоматичного тестування.

У результаті тестування отримано наступний результат: знайдено 2 вразливості середнього рівня критичності та одну вразливість низького рівня критичності. На рис. 31 показано результат автоматизованого тестування.

Перша вразливість під назвою CSP Header Not Set попереджує, що вебдодаток може бути вразливий до XSS-атак, але потрібні механізми для захисту уже реалізовано. Висновок утиліти базується лише на відсутності CSP заголовку в вебдодатку.

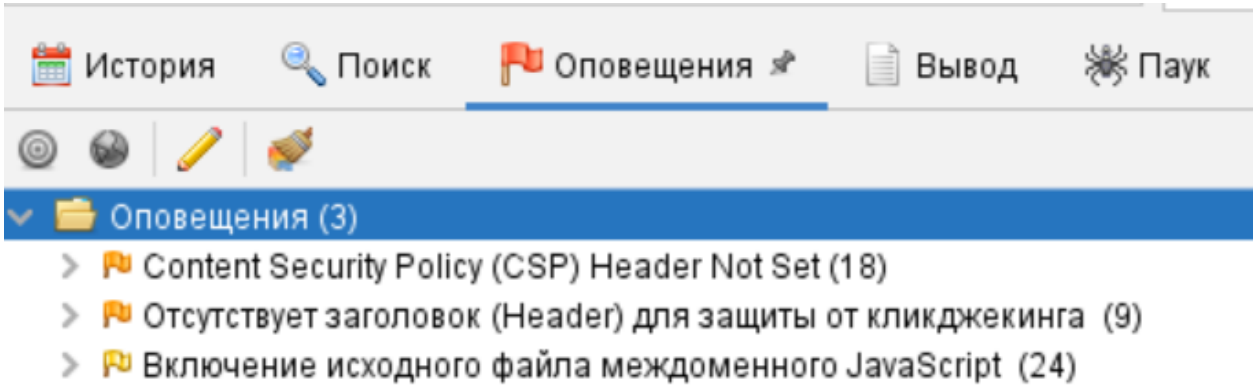


Рисунок 31. – Результат автоматизованого тестування утилітою Owasp ZAP.

Друга вразливість, яка наголошує на відсутності захисту від клікджекінгу, свідчить про можливість фішингу, суть якого полягає в можливості демонстрації користувачу сторінки вебдодатку з невидимими елементами, які він може натиснути та потрапити на «гачок» зловмисника. Для протидії цьому додамо конфігурацію для nginx у вигляді наступного рядку:

add_header X-Frame-Options "SAMEORIGIN";

Даний заголовок для nginx захищає користувачів від фішингових Clickjaking атак, якщо їх браузер має підтримку такої технології.

Третя вразливість свідчить про те, що вебдодаток використовує JavaScript код з інших джерел, але усі джерела є довіреними, тому дану вразливість не приймаємо до уваги.

ВИСНОВКИ

Поширене використання різноманітних інформаційних систем, що, як правило, містять різні типи баз даних, у різних галузях суспільного життя одночасно несе із собою великі ризики. Тому питання захисту інформаційних систем та їх баз даних є одним із нагальних на етапі їх створення.

У результаті проведеної роботи проаналізовано наявні системи керування базами даних та загрози, яким вони піддаються. Досліджено питання існуючих механізмів захисту баз даних.

В процессі виконання практичної частини сформовано модель системи безпеки для бази даних у вебдодатку, що розробляється. Спроєктовано модель вебдодатку з урахуванням типів даних та здійснена програмна реалізація вебдодатку, у якому реалізовано такі механізми захисту бази даних від сторонніх осіб:

- унеможливлення sql ін’єкцій;
- захист від зовнішніх підключень, що унеможливлює будь-яке віддалене підключення злочинця до бази даних;
- контроль трафіку;
- жорсткий контроль доступу та автоматичне збереження резервних копій.

Також було розроблено план тестування безпеки та здійснено його реалізацію для забезпечення максимальної стійкості вебдодатку та його бази даних до кібератак. Результати тестування виявили певні несуттєві недоліки і, таким чином, допомогли доопрацювати вебдодаток та макисмально захистити базу даних від несанкціонованого доступу.

СПИСОК ЛІТЕРАТУРИ

1. SQL или NoSQL [Electronic resource]. – Access mode: https://habr.com/ru/company/ruvds/blog/324936/

2. SQL vs NoSQL: 5 Critical Differences [Electronic resource]. – Access mode: https://www.xplenty.com/blog/the-sql-vs-nosql-difference/

3. Типи загроз для бази даних [Electronic resource]. – Access mode: https://habr.com/ru/company/otus/blog/557296/

4. The 10 biggest data hacks of the decade [Electronic resource]. – Access mode: https://www.cnbc.com/2019/12/23/the-10-biggest-data-hacks-of-the-decade.html

5. Захист баз даних — запорука безпеки корпоративної мережі [Electronic resource]. – Access mode: https://eset.ua/ua/blog/view/14/-

6. Захист даних у базах даних [Electronic resource]. – Access mode: https://pidru4niki.com/88683/informatika/zahist_danih_bazah_danih

7. OWASP Web Security Testing Guide [Electronic resource]. – Access mode: https://owasp.org/www-project-web-security-testing-guide/

8. SQL Injection Tutorial: Learn with Example [Electronic resource]. – Access mode: https://www.guru99.com/learn-sql-injection-with-practical-example.html

9. Чому технологія Zero Trust важлива для безпеки та продуктивності [Electronic resource]. – Access mode: https://www.itsec.ru/articles/pochemu-tekhnologiya-zero-trust-vazhna-dlya-bezopasnosti-i-proizvoditelnosti-udalennogo-personala

10. What is a DDoS Attack? DDoS Meaning, Definition & Types [Electronic resource]. Access mode: https://www.fortinet.com/resources/cyberglossary/ddos-attack

11. Різниця між базою даних NoSQL і SQL [Electronic resource]. Access mode: https://uk.natapa.org/difference-between-nosql-and-sql-database-872

12. Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others [Electronic resource]. Access mode: https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/

ДОДАТОК А

SQL-Код для імпорту таблиць:

```
CREATE TABLE IF NOT EXISTS `posts` (
  `id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `post_title` varchar(256) CHARACTER SET utf8mb3 COLLATE utf8_general_ci
DEFAULT NULL,
  `post_text` text NOT NULL,
  `post_images` json DEFAULT NULL,
  `create_date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;

CREATE TABLE IF NOT EXISTS `rules` (
  `id` int NOT NULL AUTO_INCREMENT,
  `rule_name` text NOT NULL,
  `user_field` text NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;

INSERT INTO `rules` (`id`, `rule_name`, `user_field`) VALUES
(1, 'Показывать имя', 'user_name'),
(2, 'Показывать фамилию', 'user_surname'),
(3, 'Показывать город', 'user_city'),
(4, 'Показывать дату рождения', 'user_birthday');

CREATE TABLE IF NOT EXISTS `sessions` (
  `id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `session_id` varchar(255) NOT NULL,
  `session_hash` varchar(255) NOT NULL,
  `create_date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `last_date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `create_ip` varchar(255) NOT NULL,
  `last_ip` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `session_id` (`session_id`),
  KEY `session_hash` (`session_hash`),
  KEY `user_id` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;

CREATE TABLE IF NOT EXISTS `users` (
  `id` int NOT NULL AUTO_INCREMENT,
  `user_login` varchar(256) NOT NULL,
  `user_password` varchar(256) NOT NULL,
  `user_name` varchar(256) DEFAULT NULL,
  `user_surname` varchar(256) DEFAULT NULL,
  `user_birthday` varchar(256) DEFAULT NULL,
  `user_city` varchar(256) DEFAULT NULL,
  `user_avatar` text,
  `settings` json DEFAULT NULL,
  `register_ip` text NOT NULL,
  `last_ip` text NOT NULL,
```

```
  `register_date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `last_date` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `user_login` (`user_login`),
  KEY `user_name` (`user_name`),
  KEY `user_surname` (`user_surname`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
```

Файл /index.php вебдодатку:

```
<?php
session_start();

    $GLOBALS['sitemap'] = array (
        '404' => 'lenta.php',
        '403' => 'lenta.php',
        '500' => 'lenta.php',
        '/' => 'lenta.php',
        '/login[/]?' => 'login.php',
            '/handlers/login[/]?' => 'handlers/login.php',
        '/register[/]?' => 'register.php',
            '/handlers/register[/]?' => 'handlers/register.php',
        '/profile[/]?' => 'profile.php',
        '/logout[/]?' => 'handlers/logout.php',
        '/handlers/createPost[/]?' => 'handlers/posts/create_post.php',
        '/handlers/editProfile[/]?' => 'handlers/profile/edit_profile.php',
        '/handlers/editSettings[/]?' => 'handlers/profile/edit_settings.php',
        '/handlers/finder[/]?' => 'handlers/finder.php',
    );

    class uSitemap {
        public $title = '';
        public $params = null;
        public $classname = '';
        public $data = null;

        public $request_uri = '';
        public $url_info = array();

        public $found = false;

        function __construct() {
            $this->mapClassName();
        }

        function mapClassName() {

            $this->classname = '';
            $this->title = '';
            $this->params = null;

            $map = &$GLOBALS['sitemap'];
            $this->request_uri      =      parse_url($_SERVER['REQUEST_URI'],
PHP_URL_PATH);
            $this->url_info = parse_url($this->request_uri);
            $uri = urldecode($this->url_info['path']);
```

```
            $data = false;
            foreach ($map as $term => $dd) {
                $match = array();
                $i = preg_match('@^'.$term.'$@Uu', $uri, $match);
                if ($i > 0) {
                    $m = explode(',', $dd);
                    $data = array(
                        'classname' => isset($m[0])?trim($m[0]):'',
                        'title' => isset($m[1])?trim($m[1]):'',
                        'params' => $match,
                    );
                    break;
                }
            }
            if ($data === false) {
                if (isset($map['404'])) {
                    $dd = $map['404'];
                    $m = explode(',', $dd);
                    $this->classname = trim($m[0]);
                    //$this->title = trim($m[1]);
                    $this->params = array();
                    http_response_code(404);
                }
                $this->found = false;
            } else {
                $this->classname = $data['classname'];
                $this->title = $data['title'];
                $this->params = $data['params'];
                $this->found = true;
            }
            return $this->classname;
        }
    }

      $sm = new uSitemap();
      $routed_file = $sm->classname;
    if($routed_file != null || $routed_file != '') {
        require_once $_SERVER['DOCUMENT_ROOT'].'/controllers/session.php';
          require($routed_file);
    }

?>
```

Файл /controllers/session.php вебдодатку

```
<?php
session_start();

        require_once
$_SERVER['DOCUMENT_ROOT'].'/handlers/functions/DBConnector.php';
        $DBConnector = new DBConnector;
        $db = $DBConnector -> DBConnect();

        $session_id = session_id();
```

```
        function GetIP() {
            if(!empty($_SERVER['HTTP_CLIENT_IP'])) {
                $ip = $_SERVER['HTTP_CLIENT_IP'];
            } elseif(!empty($_SERVER['HTTP_X_FORWARDED_FOR'])) {
                $ip = $_SERVER['HTTP_X_FORWARDED_FOR'];
            } else {
                $ip = $_SERVER['REMOTE_ADDR'];
            }

            return $ip;
        }

        $sql = sprintf("SELECT * FROM sessions WHERE session_id = '%s' LIMIT
1",
            $db -> real_escape_string($session_id),
        );
        $result = $db -> query($sql);
        if($result -> num_rows) {
            $row = $result -> fetch_assoc();

            $id = $_SESSION['id'];
            $user_salt = "~Z#h@W";
            $user_ip = GetIP();
            $user_agent = $_SERVER['HTTP_USER_AGENT'];
            $session_hash                                                    =
md5($id.$session_id.$user_agent.$user_salt.$user_ip);

            $db_hash = $row['session_hash'];

            if(strcmp($session_hash, $db_hash) !== 0) {
                die('У вас нет доступа к этой сессии!');
            }
            else {
                $sql = sprintf("UPDATE sessions SET last_date = NOW(), last_ip
= '%s' WHERE session_id = '%s'",
                    $db -> real_escape_string($user_ip),
                    $db -> real_escape_string($session_id)
                );
                $db -> query($sql);
            }
        }

        mysqli_close($db);

?>
```

Файл /handlers/finder.php вебдодатку

```
<?php
session_start();

    if($_SERVER["REQUEST_METHOD"] == "POST") {

        class Finder {

            private $answer = array();
```

```
            private $errors = array();

            private $string = null;
            private $rules = null;

            private $users = array();

            private $db = null;

            private function DBConnect() {
                require_once
$_SERVER['DOCUMENT_ROOT'].'/handlers/functions/DBConnector.php';
                $DBConnector = new DBConnector;
                $this -> db = $DBConnector -> DBConnect();
            }

            private function DBClose() {
                if($this -> db != null)
                    $this -> db -> close();
            }

            private function CheckAccount($id) {
                $sql = sprintf("SELECT * FROM users WHERE id = '%s' LIMIT 1",
                    $this -> db -> real_escape_string($id));
                $result = $this -> db -> query($sql);
                if($result -> num_rows) {
                    $row = $result -> fetch_assoc();
                    $return = $row;
                }
                else
                    $return = false;

                return $return;
            }

            public function GetRules() {
                $this -> DBConnect();

                $sql = sprintf("SELECT * FROM rules ORDER BY id ASC");
                $result = $this -> db -> query($sql);
                if($result -> num_rows) {
                    while($row = $result -> fetch_assoc()) {
                        $this -> rules[$row['user_field']] = $row;
                    }
                }
                else {
                    $this -> rules = null;
                }

                $this -> DBClose();
            }

            public function Find() {
                if($this -> string == null) return false;

                $this -> DBConnect();
```

```
                $string = $this -> string;

                $sql = sprintf("SELECT * FROM users WHERE user_login LIKE '%s'
OR (user_name LIKE '%s' AND settings -> '$[0].key' = '1' AND settings ->
'$[0].value' = 'true') OR (user_surname LIKE '%s' AND settings -> '$[1].key' =
'2' AND settings -> '$[1].value' = 'true') ORDER BY id DESC",
                    $this -> db -> real_escape_string('%'.$string.'%'),
                    $this -> db -> real_escape_string('%'.$string.'%'),
                    $this -> db -> real_escape_string('%'.$string.'%'));
                $result = $this -> db -> query($sql);

                if($result -> num_rows) {
                    while($row = $result -> fetch_assoc()) {
                        $user_settings = $row['settings'];
                        $user_settings = json_decode($user_settings, true);

                        $user_array =  array(
                            'id' => $row['id'],
                            'user_login' => $row['user_login'],
                            'user_avatar' => $row['user_avatar'],
                            'user_name' => null,
                            'user_surname' => null,
                        );

                        if(filter_var($user_settings[array_search($this    ->
rules['user_name']['id'],   array_column($user_settings,   'key'))]['value'],
FILTER_VALIDATE_BOOLEAN) == true)
                            $user_array['user_name'] = $row['user_name'];
                        if(filter_var($user_settings[array_search($this    ->
rules['user_surname']['id'],  array_column($user_settings,  'key'))]['value'],
FILTER_VALIDATE_BOOLEAN) == true)
                            $user_array['user_surname']                    =
$row['user_surname'];

                        $this -> users[] = $user_array;
                    }
                }
                else
                    $this -> users = array();

                $this -> DBClose();

                $this -> PrintFindResult();
            }

            private function PrintFindResult() {
                if($this -> users == null) return false;

                $html = '';

                foreach($this -> users as $user) {
                    if((isset($user['user_name'])  &&  $user['user_name']  !=
null) || (isset($user['user_surname']) && $user['user_surname'] != null))
                    $name             =             trim($user['user_name'].'
'.$user['user_surname'].' aka '.$user['user_login']);
```

```
                    else $name = $user['user_login'];

                    if($user['user_avatar'] != null) {
                        $avatar_url                                          =
'"../images/avatars/'.$user['user_avatar'].'"';
                        $avatar      =      "<div      class='item__avatar'
style='background-image:url(".$avatar_url.")'></div>";
                    }
                    else             $avatar             =             "<div
class='item__avatar'>".strtoupper($user['user_login'][0])."</div>";

                    $html .= "
                        <a    href='/profile?id=".$user['id']."'    class='wow
animate__animated animate__fadeInDown' data-wow-duration='250ms'>
                            <div class='result__item'>
                                ".$avatar."
                                <div class='item__info'>
                                    <p class='info__name'>
                                        ".$name."
                                    </p>
                                </div>
                            </div>
                        </a>
                    ";
                }

                $this -> ReturnAnswer('success', $html);
            }

            public function Validator($param, $name) {
                if($param == null || $param == '') {
                    switch($name) {
                        case 'string' : {
                            $this -> ReturnError($name, 'cannot be empty.');
                            break;
                        }
                    }
                }
                else {
                    $param = trim($param);
                    $param = htmlspecialchars($param, ENT_QUOTES);
                    $param = htmlentities($param, ENT_QUOTES);
                    $param = strip_tags($param);
                    $param = addslashes($param);

                    switch($name) {
                        case 'string' : {
                            $param = strval($param);

                            $this -> string = (string) $param;
                            break;
                        }
                        case 'ajax' : {
                            $param = boolval($param);

                            if($param == true)
```

```
                                $this -> ajax = true;
                            else
                                $this -> ReturnError($name, 'Request is made
only by ajax!');
                            break;
                        }
                    }
                }
            }

            private   function   ReturnAnswer($answer_type,   $answer_data   =
array()) {
                $this -> answer['type'] = $answer_type;
                $this -> answer['content'] = $answer_data;
                die(json_encode($this -> answer));
            }

            private function ReturnError($error_field, $error_text) {
                $this -> errors[] = [
                    'error_field' => $error_field,
                    'error_text' => $error_text
                ];
                $this -> answer['errors'] = $this -> errors;
                die(json_encode($this -> answer));
            }
        }

        if(isset($_POST['string']))
            $string = (string) $_POST['string'];
        else
            $string = null;

        if(isset($_POST['ajax']))
            $ajax = (bool) $_POST['ajax'];
        else $ajax = false;

        $Finder = new Finder;

        $Finder -> Validator($string, 'string');
        $Finder -> Validator($ajax, 'ajax');

        $Finder -> GetRules();
        $Finder -> Find();

    }

?>
```

Файл /handlers/login.php вебдодатку

```
<?php
session_start();

    if($_SERVER["REQUEST_METHOD"] == "POST" && !isset($_SESSION['logged']) ||
$_SESSION['logged'] !== true) {
```

```
        class Login {

            private $answer = array();
            private $errors = array();

            private $username = null;
            private $userpassword = null;
            private $ajax = false;

            private $db = null;

            private function DBConnect() {
                require_once
$_SERVER['DOCUMENT_ROOT'].'/handlers/functions/DBConnector.php';
                $DBConnector = new DBConnector;
                $this -> db = $DBConnector -> DBConnect();
            }

            private function DBClose() {
                if($this -> db != null)
                    $this -> db -> close();
            }

            private function GetIP() {
                if(!empty($_SERVER['HTTP_CLIENT_IP'])) {
                    $ip = $_SERVER['HTTP_CLIENT_IP'];
                } elseif(!empty($_SERVER['HTTP_X_FORWARDED_FOR'])) {
                    $ip = $_SERVER['HTTP_X_FORWARDED_FOR'];
                } else {
                    $ip = $_SERVER['REMOTE_ADDR'];
                }

                return $ip;
            }

            private function CheckAccount($username) {
                $this -> DBConnect();

                $sql = sprintf("SELECT COUNT(*) as amount FROM users WHERE
user_login = '%s' LIMIT 1",
                    $this -> db -> real_escape_string($username));
                $result = $this -> db -> query($sql);
                if($result -> num_rows) {
                    $row = $result -> fetch_assoc();
                    if($row['amount'] != 0)
                        $return = true;
                    else
                        $return = false;
                }
                else
                    $return = false;

                $this -> DBClose();

                return $return;
            }
```

```
            public function ToLogin() {
                $this -> DBConnect();

                $user_login = $this -> username;
                $user_password = md5($this -> userpassword . 'TYHcoe7IrW2e');

                $ip = $this -> GetIP();

                $sql = sprintf("SELECT * FROM users WHERE user_login = '%s'
LIMIT 1",
                    $this -> db -> real_escape_string($user_login));
                $result = $this -> db -> query($sql);
                if($result -> num_rows) {
                    $row = $result -> fetch_assoc();

                    $id = $row['id'];
                    $user_db_password = $row['user_password'];
                    $avatar = $row['user_avatar'];

                    if($user_password != $user_db_password)
                        $this -> ReturnError('userpassword', 'incorrect!');

                    $sql = sprintf("UPDATE users SET last_ip = '%s', last_date
= NOW() WHERE user_login = '%s' LIMIT 1",
                    $this -> db -> real_escape_string($ip),
                    $this -> db -> real_escape_string($user_login));
                    $this -> db -> query($sql);

                    $_SESSION['logged'] = true;
                    $_SESSION['username'] = $user_login;
                    $_SESSION['avatar'] = $avatar;
                    $_SESSION['id'] = $id;

                    $session_id = session_id();
                    $user_salt = "~Z#h@W";
                    $user_agent = $_SERVER['HTTP_USER_AGENT'];
                    $session_hash                                          =
md5($id.$session_id.$user_agent.$user_salt.$ip);

                    $sql = sprintf("DELETE FROM sessions WHERE user_id = '%s'",
                        $this -> db -> real_escape_string($id));
                    $result = $this -> db -> query($sql);

                    $sql = sprintf("INSERT INTO sessions (user_id, session_id,
session_hash, create_ip, last_ip) VALUES ('%s', '%s', '%s', '%s', '%s')",
                        $this -> db -> real_escape_string($id),
                        $this -> db -> real_escape_string($session_id),
                        $this -> db -> real_escape_string($session_hash),
                        $this -> db -> real_escape_string($ip),
                        $this -> db -> real_escape_string($ip)
                    );
                    $result = $this -> db -> query($sql);

                    $this -> ReturnAnswer('success');
                }
```

```
            $this -> DBClose();
        }

        public function Validator($param, $name) {
            if($param == null || $param == '') {
                switch($name) {
                    case 'username' : {
                        $this -> ReturnError($name, 'cannot be empty.');
                        break;
                    }
                    case 'userpassword' : {
                        $this -> ReturnError($name, 'cannot be empty.');
                        break;
                    }
                }
            }
            else {
                $param = trim($param);
                $param = htmlspecialchars($param, ENT_QUOTES);
                $param = htmlentities($param, ENT_QUOTES);
                $param = strip_tags($param);
                $param = addslashes($param);

                switch($name) {
                    case 'username' : {
                        $param = strval($param);

                        $this -> username = (string) $param;

                        if(!$this -> CheckAccount($this -> username))
                            $this -> ReturnError($name, 'not found!');
                        break;
                    }
                    case 'userpassword' : {
                        $param = strval($param);

                        $this -> userpassword = (string) $param;
                        break;
                    }
                    case 'ajax' : {
                        $param = boolval($param);

                        if($param == true)
                            $this -> ajax = true;
                        else
                            $this -> ReturnError($name, 'Request is made
only by ajax!');
                        break;
                    }
                }
            }
        }

        private function ReturnAnswer($answer_type, $answer_data =
array()) {
```

```
                $this -> answer['type'] = $answer_type;
                $this -> answer['content'] = $answer_data;
                die(json_encode($this -> answer));
            }

            private function ReturnError($error_field, $error_text) {
                $this -> errors[] = [
                    'error_field' => $error_field,
                    'error_text' => $error_text
                ];
                $this -> answer['errors'] = $this -> errors;
                die(json_encode($this -> answer));
            }
        }

        if(isset($_POST['username']))
            $username = (string) $_POST['username'];
        else
            $username = null;

        if(isset($_POST['userpassword']))
            $userpassword = (string) $_POST['userpassword'];
        else
            $userpassword = null;

        if(isset($_POST['ajax']))
            $ajax = (bool) $_POST['ajax'];
        else $ajax = false;

        $Login = new Login;

        $Login -> Validator($username, 'username');
        $Login -> Validator($userpassword, 'userpassword');
        $Login -> Validator($ajax, 'ajax');

        $Login -> ToLogin();

    }

?>
```

Файл /handlers/register.php вебдодатку

```
<?php
session_start();

    if($_SERVER["REQUEST_METHOD"] == "POST" && !isset($_SESSION['logged']) ||
$_SESSION['logged'] !== true) {

        class Register {

            private $answer = array();
            private $errors = array();

            private $username = null;
            private $userpassword = null;
```

```
            private $userpasswordconfirm = null;
            private $ajax = false;

            private $db = null;

            private function DBConnect() {
                require_once
$_SERVER['DOCUMENT_ROOT'].'/handlers/functions/DBConnector.php';
                $DBConnector = new DBConnector;
                $this -> db = $DBConnector -> DBConnect();
            }

            private function DBClose() {
                if($this -> db != null)
                    $this -> db -> close();
            }

            private function GetIP() {
                if(!empty($_SERVER['HTTP_CLIENT_IP'])) {
                    $ip = $_SERVER['HTTP_CLIENT_IP'];
                } elseif(!empty($_SERVER['HTTP_X_FORWARDED_FOR'])) {
                    $ip = $_SERVER['HTTP_X_FORWARDED_FOR'];
                } else {
                    $ip = $_SERVER['REMOTE_ADDR'];
                }

                return $ip;
            }

            private function CheckAccount($username) {
                $this -> DBConnect();

                $sql = sprintf("SELECT COUNT(*) as amount FROM users WHERE
user_login = '%s' LIMIT 1",
                    $this -> db -> real_escape_string($username));
                $result = $this -> db -> query($sql);
                if($result -> num_rows) {
                    $row = $result -> fetch_assoc();
                    if($row['amount'] != 0)
                        $return = false;
                    else
                        $return = true;
                }
                else
                    $return = false;

                $this -> DBClose();

                return $return;
            }

            public function ToRegister() {
                $this -> DBConnect();

                $user_login = $this -> username;
                $user_password = md5($this -> userpassword . 'TYHcoe7IrW2e');
```

```
                $sql = sprintf("SELECT * FROM rules ORDER BY id ASC");
                $result = $this -> db -> query($sql);
                if($result -> num_rows) {
                    while($row = $result -> fetch_assoc()) {
                        $user_settings[] = ['key' => $row['id'], 'value' =>
true];
                    }
                    $user_settings = json_encode($user_settings, true);
                }
                else {
                    $user_settings = null;
                }

                $ip = $this -> GetIP();

                $sql = sprintf("INSERT INTO users (user_login, user_password,
settings, register_ip, last_ip) VALUES ('%s', '%s', '%s', '%s', '%s')",
                    $this -> db -> real_escape_string($user_login),
                    $this -> db -> real_escape_string($user_password),
                    $this -> db -> real_escape_string($user_settings),
                    $this -> db -> real_escape_string($ip),
                    $this -> db -> real_escape_string($ip),
                );
                $result = $this -> db -> query($sql);

                $id = $this -> db -> insert_id;

                $_SESSION['logged'] = true;
                $_SESSION['username'] = $user_login;
                $_SESSION['avatar'] = null;
                $_SESSION['id'] = $id;

                $session_id = session_id();
                $user_salt = "~Z#h@W";
                $user_agent = $_SERVER['HTTP_USER_AGENT'];
                $session_hash                                                =
md5($id.$session_id.$user_agent.$user_salt.$ip);

                $sql = sprintf("DELETE FROM sessions WHERE user_id = '%s'",
                    $this -> db -> real_escape_string($id));
                $result = $this -> db -> query($sql);

                $sql = sprintf("INSERT INTO sessions (user_id, session_id,
session_hash, create_ip, last_ip) VALUES ('%s', '%s', '%s', '%s', '%s')",
                    $this -> db -> real_escape_string($id),
                    $this -> db -> real_escape_string($session_id),
                    $this -> db -> real_escape_string($session_hash),
                    $this -> db -> real_escape_string($ip),
                    $this -> db -> real_escape_string($ip)
                );
                $result = $this -> db -> query($sql);

                $this -> DBClose();

                $this -> ReturnAnswer('success');
```

```
            }

            public function Validator($param, $name) {
                if($param == null || $param == '') {
                    switch($name) {
                        case 'username' : {
                            $this -> ReturnError($name, 'cannot be empty.');
                            break;
                        }
                        case 'userpassword' : {
                            $this -> ReturnError($name, 'cannot be empty.');
                            break;
                        }
                        case 'userpasswordconfirm' : {
                            $this -> ReturnError($name, 'cannot be empty.');
                            break;
                        }
                    }
                }
                else {
                    $param = trim($param);
                    $param = htmlspecialchars($param, ENT_QUOTES);
                    $param = htmlentities($param, ENT_QUOTES);
                    $param = strip_tags($param);
                    $param = addslashes($param);

                    switch($name) {
                        case 'username' : {
                            $param = strval($param);

                            $this -> username = (string) $param;

                            if(!$this -> CheckAccount($this -> username))
                                $this -> ReturnError($name, 'taken by another
user!');
                            break;
                        }
                        case 'userpassword' : {
                            $param = strval($param);

                            $this -> userpassword = (string) $param;
                            break;
                        }
                        case 'userpasswordconfirm' : {
                            $param = strval($param);

                            $this -> userpasswordconfirm = (string) $param;

                            if($this -> userpasswordconfirm != $this ->
userpassword)
                                $this -> ReturnError($name, 'do not match!');
                            break;
                        }
                        case 'ajax' : {
                            $param = boolval($param);
```

```
                            if($param == true)
                                $this -> ajax = true;
                            else
                                $this -> ReturnError($name, 'Request is made
only by ajax!');
                            break;
                        }
                    }
                }
            }

            private   function   ReturnAnswer($answer_type,   $answer_data   =
array()) {
                $this -> answer['type'] = $answer_type;
                $this -> answer['content'] = $answer_data;
                die(json_encode($this -> answer));
            }

            private function ReturnError($error_field, $error_text) {
                $this -> errors[] = [
                    'error_field' => $error_field,
                    'error_text' => $error_text
                ];
                $this -> answer['errors'] = $this -> errors;
                die(json_encode($this -> answer));
            }
        }

        if(isset($_POST['username']))
            $username = (string) $_POST['username'];
        else
            $username = null;

        if(isset($_POST['userpassword']))
            $userpassword = (string) $_POST['userpassword'];
        else
            $userpassword = null;

        if(isset($_POST['userpasswordconfirm']))
            $userpasswordconfirm = (string) $_POST['userpasswordconfirm'];
        else
            $userpasswordconfirm = null;

        if(isset($_POST['ajax']))
            $ajax = (bool) $_POST['ajax'];
        else $ajax = false;

        $Register = new Register;

        $Register -> Validator($username, 'username');
        $Register -> Validator($userpassword, 'userpassword');
        $Register -> Validator($userpasswordconfirm, 'userpasswordconfirm');
        $Register -> Validator($ajax, 'ajax');

        $Register -> ToRegister();
```

```
    }

?>
```

Файл /handlers/functions/DBConnector.php вебдодатку

```
<?php

    class DBConnector {

        private $db_server = null;
        private $db_username = null;
        private $db_password = null;
        private $db_name = null;

        private $db = null;

        public function DBConnect() {
            $this -> SetDBParams();

            $this -> db = new mysqli($this -> db_server, $this -> db_username,
$this -> db_password, $this -> db_name);
            mysqli_set_charset($this -> db, 'utf8');

            if($this -> db === false) {
                die("ERROR: Can't connect to database. " . $this -> db ->
connect_error);
            }
            else return $this -> db;
        }

        private function DBClose() {
            if($this -> db != null)
                $this -> db -> close();
        }

        private function SetDBParams() {
            $this -> db_server = 'localhost';
            $this -> db_username = 'website';
            $this -> db_password = '?vZOu%31~D}GcP12TyD68bupNoJE~6W1';
            $this -> db_name = 'univer';
        }

    }

?>
```