МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ОЛЕКСІЄНКО ОЛЕКСАНДР СЕРГІЙОВИЧ

Допускається до захисту:
В.о. завідувача кафедри
інформаційних технологій,
канд. техн. наук, доцент
_____ Оксана ЗЕЛІНСЬКА
«___» _____ 2024 р.

РОЗРОБКА МЕНЕДЖЕРА ПАРОЛІВ ДЛЯ ОСОБИСТОГО ТА КОРПОРАТИВНОГО ВИКОРИСТАННЯ

Спеціальність 122 «Комп’ютерні науки»
Кваліфікаційна (бакалаврська) робота

Керівник:
Павло РИМАР, старший викладач
кафедри інформаційних технологій

Оцінка:___/___/_____|
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2024

АНОТАЦІЯ

Олєксієнко О.С. Розробка менеджера паролів для особистого та корпоративного використання. Спеціальність 122 «Комп’ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У кваліфікаційній (бакалаврській) роботі було розроблено програмний додаток «Менеджер паролів для підвищення кібербезпеки в особистому та корпоративному використанні», що призначений для управління паролями та безпеки.

Додаток розроблений на мовах програмуванні JavaScript та TypeScript в середовищі розробки Node.js.

Ключові слова: дані, безпека, клієн-сервер, Node.js, TypeScript, JavaScript, ООП, MySQL, SSH.

68 с., 6 рис., 1 дод., 45 джерел.

ABSTRACT

Oleksiienko O.S. Password manager to increase cyber security in personal and corporate use. Specialty 122 "Computer Science". Vasyl’ Stus Donetsk National University, Vinnytsia, 2024.

In the qualifying (bachelor's) work, a software application "Password manager for increasing cyber security in personal and corporate use" was developed, which is designed for password management and security.

The application is developed in JavaScript and TypeScript programming languages in the Node.js development environment.

Keywords: data, security, client-server, Node.js, TypeScript, JavaScript, OOP, MySQL, SSH.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Шифрування – це метод перетворення відкритого тексту в закодований формат за допомогою спеціального алгоритму для забезпечення таємності інформації.

AWS – це платформа для хмарних сервісів, що пропонує широкий спектр інструментів для створення, запуску та управління програмами в хмарному середовищі.

NoSQL – це тип бази даних, що зберігає інформацію у форматах документів, пар ключ-значення, стовпців або графів і не використовує традиційні реляційні таблиці.

ООП – це парадигма програмування, де програми будуються як системи взаємодіючих об'єктів.

Back end – це сегмент веб додатку або сайту, що займається обробкою користувацьких запитів, взаємодією з базами даних та реалізацією бізнес-логіки.

Node.js – це відкрите середовище для виконання JavaScript, побудоване на двигуні V8, яке дозволяє запуск JavaScript на сервері.

RDS – це сервіс для управління реляційними базами даних в хмарі AWS, що спрощує конфігурацію та управління базами даних.

ВСТУП

Актуальність теми дослідження

В епоху цифровізації, коли кожен аспект життєдіяльності індивіда та організації перетворюється на потік даних у всесвітній мережі, питання кібербезпеки набуває стратегічної важливості. Відсутність ефективного менеджменту паролями є одним із найвразливіших ланок у безпеці даних, що робить менеджери паролів необхідним інструментом в особистому та корпоративному використанні для захисту конфіденційної інформації. Проблема втрати, забуття або неавторизованого отримання доступу до паролів постійно зростає, що підкреслює високу актуальність питання збереження паролів не тільки в сфері особистої безпеки, але й в масштабах всього бізнесу.

Також важливим є аспект вдосконалення існуючих рішень менеджерів паролів, багато з яких не враховують специфіки та потреби різних категорій користувачів та бізнес-структур, пропонуючи стандартні рішення, що не забезпечують достатнього рівня індивідуалізації та гнучкості в управлінні безпекою.

Мета

Головною метою даної роботи є підвищення рівня кібербезпеки через розробку та впровадження комплексного менеджера паролів, призначеного для захисту персональних та корпоративних даних. Це передбачає створення надійної системи управління автентифікацією, яка дозволить запобігати несанкціонованому доступу, забезпечить зручне зберігання та використання паролів, а також автоматизує процес відновлення доступу у випадку забуття або втрати паролів.

Завдання дослідження

Завданням цього дослідження є:

- розробка інтерфейсу користувача, який буде інтуїтивно зрозумілим та простим у використанні;
- вибір оптимальних технологій та мов програмування для розробки

менеджера паролів;

- створення зашифрованої бази даних для надійного зберігання паролів;
- інтеграція двофакторної автентифікації для додаткового рівня безпеки;
- реалізація автоматизації відновлення паролів;
- проведення комплексного тестування системи на безпеку та зручність використання.

Об’єкт дослідження

Методи та інструменти кібербезпеки, зокрема менеджмент парольними даними в різних середовищах.

Предмет дослідження

Предметом дослідження є принципи розробки безпечних систем автентифікації та менеджменту паролів, включаючи використання сучасних мов програмування та криптографічних бібліотек.

Теоретичне значення одержаних результатів

Теоретичне значення цієї бакалаврської дипломної роботи полягає у подальшому розвитку наукових знань в області кібербезпеки, зокрема в аспектах захисту і менеджментом парольними даними. Розглянуто сучасні методики та технології шифрування, які можуть бути використані для захисту конфіденційної інформації в особистому та корпоративному контексті. Вивчення різних підходів до менеджменту паролями допомагає виявити потенційні слабкі місця та можливості для підвищення безпеки.

Практичне значення одержаних результатів

Практичне значення роботи виявляється в розробці менеджера паролів, який дозволяє значно підвищити рівень захисту парольних даних у корпоративному та особистому використанні. Реалізоване рішення сприяє автоматизації процесів менеджменту паролями, зменшує ризик несанкціонованого доступу, та підвищує ефективність роботи користувачів завдяки інтуїтивно зрозумілому інтерфейсу та вдосконаленому доступу до парольних даних.

Структура роботи

Кваліфікаційна (бакалаврська) робота складається із вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено огляд та аналіз існуючих програмних рішень для менеджеру паролями, оцінено їхні переваги та недоліки, і визначено ключові вимоги до розроблюваної системи.

Другий розділ присвячений огляду та аналізу вибору інструментів для реалізації даного вебдодатку, а саме технології для написання серверної та клієнтської частини з також було оцінено їхні переваги та недоліки.

Третій розділ присвячений проектуванню архітектури менеджера паролів, включаючи вибір технологій, проектування бази даних, та методи шифрування, які забезпечать високий рівень безпеки зберігання даних.

Четвертий розділ описує реалізацію і тестування менеджера паролів, включаючи інтерфейс користувача та інтеграцію двофакторної автентифікації. Також розглядається впровадження системи в корпоративне середовище та оцінка її ефективності.

Для написання бакалаврської роботи використано – 45 літературних джерел.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ ПРОДУКТІВ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕМИ ПРОЕКТУ

1.1 Загальний аналіз проблематики, яке вирішує дане програмне забезпечення

Менеджмент паролями відіграє ключову роль у захисті кібербезпеки. З кожним днем з'являється все більше онлайн-платформ, що вимагають реєстрації, збільшуючи кількість вебсайтів, сервісів та додатків, які потребують створення нових облікових записів. Кожен новий додаток вимагає від користувача створення нового пароля, що може стати складним завданням з огляду на їх зростаючу кількість.

Зберігання паролів на папері або в текстових файлах є ризикованим, оскільки це піддає їх ризику бути викраденими хакерами, які можуть спробувати дістати доступ до особистих даних користувачів. Використання простих та легко здогадливих паролів також несе в собі загрози, як і практика використання одного пароля для кількох сайтів. Це може призвести до того, що при зломі одного сайту зловмисник зможе доступитися до інших облікових записів користувача. Користувачі часто вдаються до використання слабких, повторюваних паролів або зберігають їх у місцях, що не забезпечують належного захисту.

Ці проблеми можна вирішити за допомогою вебдодатку для менеджменту паролями, відомих як менеджери паролів. Ці додатки надають можливість зберігати всі паролі в одному захищеному місці, що полегшує їх використання та допомагає підтримувати організацію. Вони також захищають від використання слабких чи повторних паролів на різних платформах, пропонуючи генерацію випадкових надійних паролів.

Проте, існуючі програмні рішення для менеджменту паролями можуть мати недоліки та обмеження, які можуть знизити їх зручність та ефективність. Наприклад, деякі додатки можуть мати складний інтерфейс, обмежену підтримку платформ або високу вартість користування. Існує також ризик злому або втрати

даних через вразливості в системах безпеки цих додатків. Таким чином, розробка нового вебдодатку для менеджменту паролями, який враховує ці недоліки та забезпечує високий рівень безпеки, стає важливою задачею.

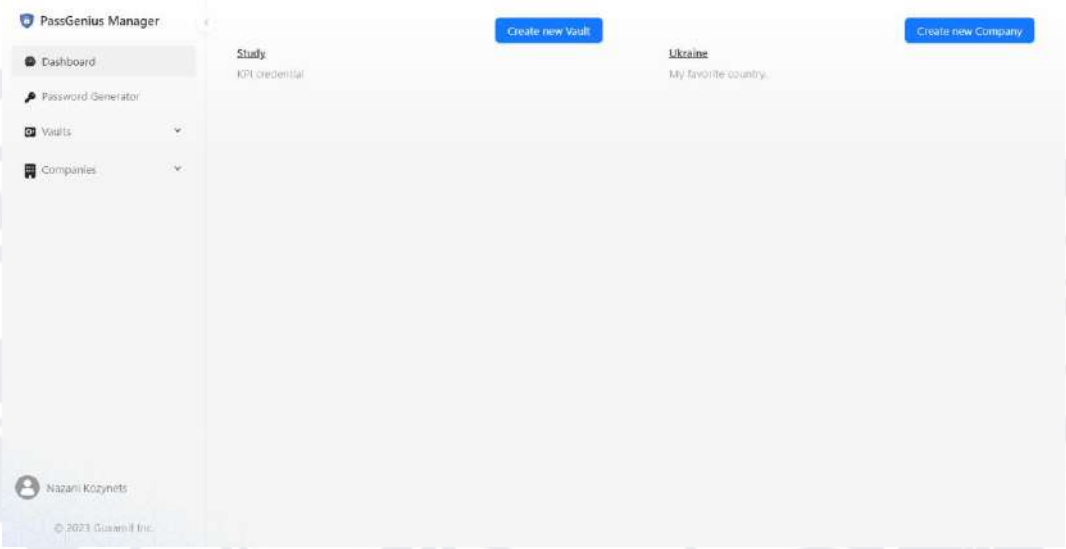

Рисунок 1.1 – Приклад головної сторінки менеджменту паролями

На рис. 1.1 зображено приклад головної сторінки менеджменту паролями. Його ключові функції включають безпечне зберігання паролів, створення випадкових надійних паролів та синхронізацію між різними пристроями. Вебдодаток забезпечує надійний захист проти несанкціонованого доступу та хакерських атак, а також пропонує користувачам зручний інтерфейс для менеджменту паролями.

Таким чином, створення нового вебдодатку для менеджменту паролями має на меті вирішення проблем зберігання, використання та організації паролів для різних сервісів та додатків. Цей додаток повинен забезпечувати користувачам зручний інтерфейс, високий рівень безпеки та доступність на різних платформах.

1.2 Огляд наявних програмних рішень

На сучасному ринку існує чимало додатків, які дозволяють керувати паролями, серед яких деякі користуються великою популярністю і широко використовуються. У цьому розділі розглянемо декілька таких додатків та оцінимо їхні переваги та недоліки.

1.1.1. *LastPass [2]*

LastPass вважається одним з найвідоміших вебдодатків для менеджменту паролями. Цей додаток пропонує зручний інтерфейс, можливість синхронізації між пристроями, автоматичне заповнення паролів на сайтах та в додатках, а також створення випадкових надійних паролів. Проте, у LastPass є певні недоліки:

- Неможливість локального зберігання даних: всі дані зберігаються у хмарі, що може бути незручним для деяких користувачів.
- Затримки у синхронізації: деякі користувачі відзначають проблеми зі швидкістю синхронізації між пристроями.
- Часті випадки витоку даних: LastPass зіткнувся з декількома випадками витоку даних, що викликає занепокоєння щодо безпеки сервісу.
- Обмеження у безкоштовній версії: LastPass обмежує доступ до деяких функцій у своїй безплатній версії, спонукаючи користувачів переходити на платні плани.

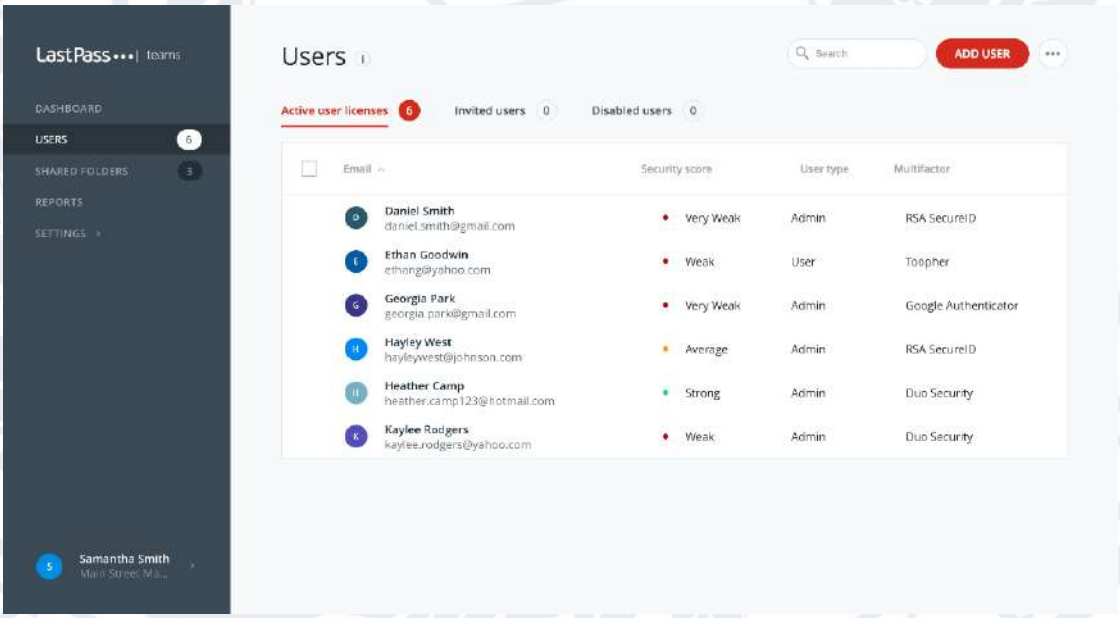

Рисунок 1.2 – Інтерфейс LastPass

1.1.2. *Keeper [4]*

Keeper – це додаток для менеджменту паролями, який акцентує увагу на безпеці. Він надає функції двофакторної автентифікації, автоматичного

заповнення паролів, синхронізації між пристроями та збереження файлів. Основні недоліки Keeper:

- Відсутність функції автоматичного оновлення паролів: Keeper не має можливості автоматично оновлювати паролі на сайтах, що може бути не зручно.
- Проблеми з синхронізацією: деякі користувачі зазначають, що між пристроями можуть виникати затримки у синхронізації.
- Обмеження безкоштовної версії: безкоштовна версія Keeper має обмежені можливості, такі як відсутність синхронізації та обмежений обсяг сховища.
- Брак вбудованого генератора паролів: хоча Keeper дозволяє зберігати існуючі паролі, він не має вбудованої функції для їх генерації.

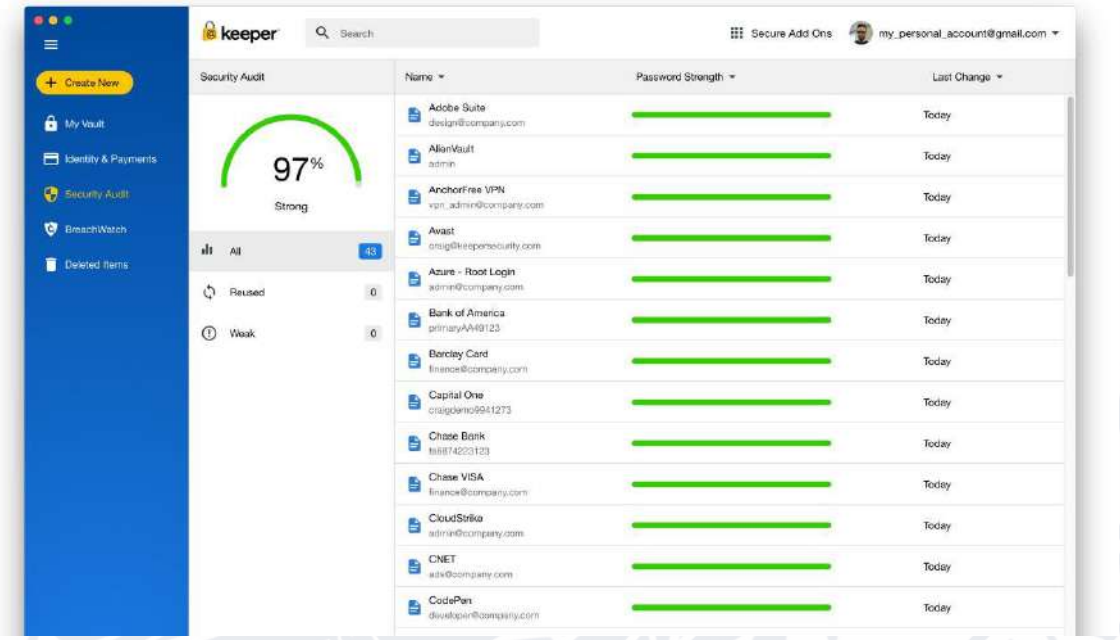

Рисунок 1.3 – Інтерфейс Keeper

1.1.3. *1Password [6]*

1Password – популярний додаток для менеджменту паролями, який пропонує багатий набір функцій, включаючи автоматичне заповнення паролів, генерацію паролів, синхронізацію між пристроями та збереження файлів. Недоліки 1Password:

- Висока вартість: платні плани 1Password мають відносно високі ціни, що може відштовхнути деяких користувачів.
- Обмежена підтримка платформ: 1Password підтримує менше платформ порівняно з деякими конкурентами.
- Фокус на індивідуальних користувачах: додаток більш орієнтований на індивідуальних користувачів і може бути не так ефективним для великих організацій.
- Відсутність безкоштовної версії: 1Password не пропонує безкоштовну версію, але надає пробний період.

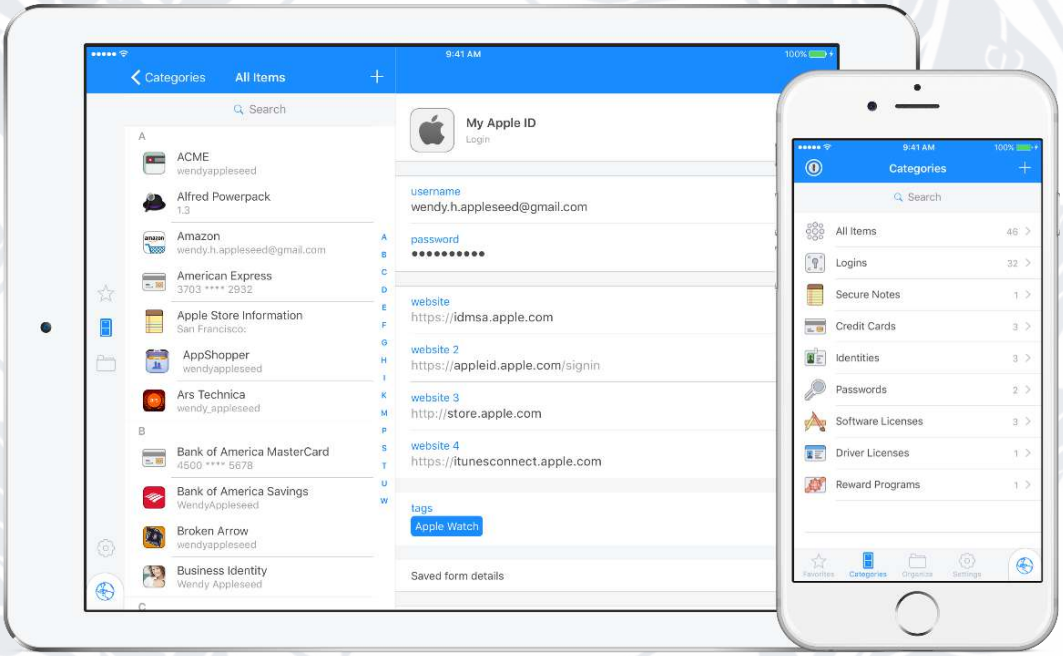

Рисунок. 1.4 – Інтерфейс 1Password

1.3 Постановка задачі

На основі детального аналізу існуючих програмних рішень для менеджменту паролями, а також виокремлення їх недоліків та можливостей для поліпшення, мета розробки вебдодатку для менеджменту паролями полягає у створенні продукту, який об'єднає високий рівень користувацької зручності, безпеки та доступності. Цей інструмент призначений для зберігання, генерації та менеджменту паролями користувачів.

Для досягнення цієї мети в рамках дипломного проекту були визначені наступні завдання:

1. Аналіз та визначення основних вимог до функціональності, безпеки та доступності вебдодатку для менеджменту паролями, враховуючи користувацькі потреби та стандарти безпеки.
2. Розробка архітектури системи, що враховує безпеку даних та інтеграцію з різними платформами, забезпечуючи гнучкість та сумісність додатку.
3. Впровадження функціоналу для зберігання паролів у безпечному, зашифрованому вигляді з можливістю синхронізації між різними пристроями.
4. Реалізація можливості створення випадкових, складних паролів із налаштуванням параметрів генерації для забезпечення додаткової безпеки.
5. Впровадження системи авторизації та аутентифікації, використовуючи сучасні механізми двофакторної авторизації для покращення захисту облікових записів.
6. Розробка користувацького інтерфейсу.
7. Тестування на відповідність стандартам безпеки, функціональності та зручності використання.
8. Створення стратегії впровадження та підтримки вебдодатку, включаючи маркетингові, технічні та розвиткові аспекти для забезпечення стабільності та продовження роботи додатку.

Виконання цих завдань дозволить реалізувати мету проєкту – створити вебдодаток для менеджменту паролями, який буде виділятися на ринку своєю зручністю, надійністю та доступністю для різних категорій користувачів.

РОЗДІЛ 2
ВИЗНАЧЕННЯ ВИБОРУ ІНСТРУМЕНТІВ ДЛЯ РЕАЛІЗАЦІЇ

2.1 Вибір та обґрунтування інструментарію для серверної частини системи

При створенні вебдодатку для менеджменту паролями критично важливо забезпечити ефективне взаємодію клієнт-сервер та обробку запитів. Серверна частина є відповідальною за обробку і зберігання даних, безпеку та втілення бізнес-логіки.

Основною перевагою використання фреймворку NestJS для серверної частини є його здатність ефективно управляти ресурсами, забезпечувати безпеку даних та зручно спілкуватися з клієнтом через API. NestJS використовує сучасний JavaScript, підтримує TypeScript з самого початку та пропонує модульну структуру, що сприяє спрощенню розробки та тестування.

Ці особливості виділяють NestJS серед інших альтернативних технологій, таких як Express.js та Koa.js, які, хоч і є популярними та гнучкими фреймворками, не пропонують аналогічного рівня структурованості та модульності. Крім того, NestJS має вбудовану підтримку інструментів для тестування, що є важливим для створення надійних вебдодатків.

З урахуванням вимог до продуктивності, безпеки, простоти використання та швидкості розробки, для реалізації обрано наступний стек технологій: мова програмування TypeScript, фреймворк NestJS, база даних MongoDB, сервіс відправки повідомлень AWS SES та зберігання файлів на AWS S3. Цей набір інструментів є сучасним, гнучким і надійним, що дозволяє створювати якісні веб сервіси з високим рівнем безпеки та продуктивності.

2.1.1. *Вибір мови програмування TypeScript*

Для створення серверної частини системи було обрано мову програмування TypeScript.

TypeScript — це розширення JavaScript, яке додає строгу типізацію та інші сучасні можливості програмування. Нижче наведено деякі з його переваг та недоліків.

Переваги TypeScript:

- TypeScript має розгалужений набір інструментів для автоматизації тестування, що дозволяє ефективно знаходити помилки та скорочувати час на відладку.
- Підтримка модульності та об'єктноорієнтованого програмування, що робить код більш організованим і легко зрозумілим.
- TypeScript дозволяє використовувати останні функції ECMAScript, які ще не підтримуються усіма браузерами, спрощуючи розробку для вебплатформи.
- Строга типізація допомагає виявляти помилки під час розробки, що поліпшує читабельність коду та знижує кількість помилок при виконанні.
- TypeScript підтримує різні системи збірки, такі як Webpack і Rollup, що дозволяє оптимізувати продуктивність та скоротити час завантаження сторінок.

Недоліки TypeScript:

- TypeScript може бути надмірним для малих проєктів, де додаткове навантаження на типізацію може бути невиправданим.
- TypeScript може бути складнішим для освоєння новачками порівняно з JavaScript, особливо для тих, хто не має досвіду з типізованими мовами програмування.
- Застосування TypeScript може призводити до збільшення часу розробки через необхідність визначення типів та інтеграцію сучасних функцій.

Для розробки серверної частини системи TypeScript є відмінним вибором, оскільки він дозволяє розробляти більш структурований та безпечний код за допомогою строго типізованих інтерфейсів, класів та функцій. Завдяки підтримці

Node.js, TypeScript легко інтегрується з серверними додатками. Строга типізація в TypeScript сприяє зменшенню помилок у розробці та забезпечує більшу стабільність та надійність серверного ПЗ. Крім того, можливість використання новітніх функцій ECMAScript може покращити продуктивність та ефективність серверної частини програмного забезпечення.

2.1.2. *Оптимізація за допомогою середовища виконання Node.js [19]*

Node.js було вибрано як середовище виконання для серверної частини системи, оскільки воно базується на JavaScript і дозволяє виконувати JavaScript код на сервері.

Node.js — це середовище виконання, що дозволяє розробникам використовувати JavaScript для створення серверних додатків та веб-додатків. Ось деякі переваги та недоліки Node.js:

Переваги Node.js:

- Дозволяє використовувати один і той же код на клієнтській і серверній сторонах, що скорочує час розробки й спрощує підтримку.
- Має велику кількість доступних зовнішніх бібліотек і модулів, що сприяє ефективній реалізації різноманітних задач.
- Проста архітектура і відкрите програмне забезпечення дають розробникам свободу у створенні додатків.
- Node.js забезпечує високу продуктивність і швидкість завдяки асинхронній обробці подій і мінімальному використанню ресурсів.

Недоліки Node.js:

- Може потребувати більше пам'яті та ресурсів сервера, що в деяких випадках знижує продуктивність.
- Однопоточність може вплинути на продуктивність при обробці великих об'ємів даних.
- Може бути складніше для новачків, особливо для тих, хто не знайомий з асинхронним програмуванням.

З урахуванням високої продуктивності та швидкості, Node.js ідеально підходить для розробки серверної частини системи, оскільки забезпечує ефективну обробку запитів і відповідей. Його здатність до асинхронної роботи й велика кількість модулів дозволяють швидко розгортати додатки, а використання JavaScript на сервері та клієнті знижує час розробки та підтримки коду. Node.js також має просту архітектуру та відкрите ПЗ, що дає розробникам мінімальні обмеження і високу продуктивність. Отже, попри деякі недоліки, переваги Node.js роблять його чудовим вибором для розробки серверних додатків та вебдодатків.

2.1.3. *Засхтосування фреймворку Nest.js*

Nest.js було вибрано як фреймворк для розробки серверної частини системи, заснований на Node.js. Цей фреймворк дозволяє розробникам створювати масштабовані та добре структуровані серверні додатки, використовуючи TypeScript та об'єктно-орієнтовані принципи програмування. Ось деякі з його переваг та недоліків:

Переваги Nest.js:

- Підтримує використання різноманітних патернів проєктування, як MVC, SOLID, Dependency Injection, що сприяє оптимізації продуктивності та скороченню часу розробки.
- Має вбудовану підтримку для широкого спектру технологій, включаючи WebSocket, gRPC, GraphQL, дозволяючи розробникам легко інтегрувати ці технології у свої додатки.
- Має активну спільноту розробників, що сприяє постійному оновленню фреймворку та підтримці.
- Дозволяє структурувати код з використанням модулів та ООП принципів, роблячи код більш організованим та зрозумілим.
- Nest.js базується на TypeScript, що вносить строгу типізацію та сучасні можливості програмування, знижуючи кількість помилок і полегшуючи розуміння коду.

Недоліки Nest.js:

- Може вимагати додаткових ресурсів сервера у порівнянні з іншими фреймворками, що може впливати на продуктивність сервера.
- Може представляти складнощі для освоєння новачками, особливо для тих, хто не знайомий з TypeScript чи принципами ООП.

Як фреймворк для серверної частини системи, Nest.js виявляється ідеальним вибором з наступних причин:

- Підтримка різних патернів проєктування та Dependency Injection сприяє чистоті коду та організації компонентів.
- Має підтримку різних технологій, які надають гнучкість у проєктуванні систем.
- Активна спільнота та численні модулі сприяють швидкій розробці та розв'язання проблем.
- Забезпечує масштабованість та структурованість за допомогою TypeScript та ООП, спрощуючи код та зменшуючи помилки.

Таким чином, незважаючи на деякі недоліки, переваги використання Nest.js роблять його відмінним варіантом для створення серверної частини системи, забезпечуючи стабільність, масштабованість і продуктивність розробленого ПЗ.

2.1.4. *Застосуваня JSON Web Token [24]*

У цій системі для процесів автентифікації та авторизації користувачів вирішили застосувати технологію JSON Web Token (JWT). JWT – це відкритий стандарт (RFC 7519), що дозволяє безпечно обмінюватися даними між сторонами через JSON об'єкт. Ці токени підписуються цифрово, що дозволяє перевіряти їхню справжність та надійність.

Основні переваги використання JWT у системі:

- JWT дозволяє безпечно передавати інформацію між сторонами, завдяки цифровому підпису, що гарантує її справжність.

- JWT забезпечує надійну та безпечну систему авторизації на основі токенів. Користувачі отримують токен після успішної аутентифікації, який потім використовують для авторизації запитів до сервера.
- JWT є гнучкими, оскільки вони можуть містити різні дані, необхідні для додатка, включаючи інформацію про користувача.

Потенційні недоліки використання JWT:

- Розробники повинні ретельно управляти життєвим циклом токенів, включаючи їх поновлення та анулювання.
- JWT можуть мати великий розмір, особливо якщо містять багато даних, що може сповільнити завантаження сторінок при повільному з'єднанні.
- Безпека JWT залежить від того, наскільки безпечно зберігаються та передаються токени. Втрата чи крадіжка токена може призвести до несанкціонованого доступу.

Незважаючи на деякі виклики, переваги JWT роблять їх відмінним інструментом для автентифікації та авторизації, забезпечуючи безпечний обмін інформацією між сторонами. Саме тому JWT було обрано як основну технологію для автентифікації та авторизації у цій системі.

2.1.5. *Застосування бази даних MongoDB*

MongoDB була обрана як база даних для розробки системи через її гнучкість та масштабованість як NoSQL база даних.

MongoDB — це документо-орієнтована база даних, що зберігає дані у форматі JSON-документів. Ось деякі з переваг і недоліків MongoDB та пояснення, чому вона ідеально підходить як база даних для розробки серверної частини системи.

Переваги MongoDB:

- Підтримка реплікації та шардингу вбудована в MongoDB, що забезпечує масштабованість та надійність системи.
- Активна спільнота розробників і відкрите програмне забезпечення гарантують постійну підтримку та оновлення.

- Швидке виконання запитів завдяки зберіганню даних у форматі документів JSON, що спрощує доступ і зменшує кількість запитів.
- MongoDB дозволяє гнучке зберігання даних без жорстких схем, надаючи розробникам більше можливостей при проектуванні системи.
- Широкий вибір зовнішніх бібліотек та модулів розширює можливості розробників і сприяє швидкій реалізації серверних додатків.

Недоліки MongoDB:

- Може бути важче для освоєння новачками, зокрема для тих, хто не знайомий з документо-орієнтованими базами даних.
- Може потребувати більше пам'яті та ресурсів сервера, особливо при обробці великих обсягів даних.

MongoDB виявляється оптимальним вибором для серверної частини системи з таких причин:

- Багатий вибір бібліотек і модулів дозволяє швидко розгортати серверні додатки.
- Активна спільнота забезпечує надійну підтримку та постійне оновлення бази даних.
- Гнучка схема дозволяє розробникам з легкістю адаптувати базу під потреби системи без значних змін у коді.
- Швидкість виконання запитів завдяки формату зберігання даних у вигляді документів JSON зменшує час доступу до даних.
- Підтримка реплікації та шардингу гарантує масштабованість і високу доступність даних.

Отже, хоча існують деякі потенційні недоліки, сильні сторони MongoDB, такі як гнучкість, швидкість обробки та вбудована масштабованість, роблять її відмінним рішенням для розробки серверної частини системи, дозволяючи ефективно управляти даними.

2.1.6. *Застосування бібліотеки Mongoose [26]*

Mongoose була вибрана як бібліотека для роботи з MongoDB, оскільки вона забезпечує зручний API та розширену підтримку для валідації, конфігурації схем, та взаємодії з документами. Ось декілька переваг та недоліків Mongoose і пояснення, чому вона ідеально підходить для MongoDB у контексті розробки серверної частини системи.

Переваги Mongoose:

- Вона надає можливість ефективно здійснювати запити до бази даних та керування даними, що додає гнучкості та спрощує взаємодію з базою даних.
- Відкрите програмне забезпечення та активна спільнота розробників, які забезпечують надійну підтримку та регулярні оновлення бібліотеки.
- Mongoose включає вбудовані засоби для валідації даних, що допомагає забезпечувати точність введеної інформації та знижує кількість помилок.
- Mongoose дозволяє розробникам легко структурувати та використовувати схеми та моделі для MongoDB, що сприяє зручному керуванню даними і зниженню кількості помилок.
- Розширені можливості через велику кількість доповнень та модулів, що дозволяють розробникам швидко втілювати різноманітні задачі та розгортати серверні застосунки.

Недоліки Mongoose:

- Mongoose може бути складним для освоєння новачками, особливо для тих, хто не має досвіду з MongoDB та схемами даних.

Mongoose виявляється ідеальним вибором для MongoDB при розробці серверної частини системи з таких причин:

- Багатий набір зовнішніх бібліотек та модулів розширює функціональні можливості розробників.
- Вбудована підтримка валідації даних дозволяє забезпечувати точність інформації, вносимої до системи.

- Активна спільнота та відкрите програмне забезпечення забезпечують динамічну підтримку і постійне оновлення.
- Mongoose забезпечує зручний та простий API для роботи з MongoDB, що дозволяє розробникам ефективно виконувати різноманітні завдання та швидко розгортати серверні додатки.
- Швидкі та гнучкі запити до бази даних завдяки простому API, що полегшує доступ до даних.
- Легкість створення та використання схем та моделей полегшує управління даними й зменшує кількість помилок.

Отже, попри деякі потенційні недоліки, переваги Mongoose, такі як зручне керування даними, гнучкість запитів, і сильна підтримка, роблять її відмінним вибором для використання з MongoDB у розробці серверної частини системи.

2.1.7. *Застосування хмарного сервісу Amazon Web Services [27]*

Amazon Web Services (AWS) було вибрано як хмарний сервіс для розгортання та зберігання даних у проєкті, завдяки його широкому спектру сервісів, таких як EC2 для розгортання серверних додатків, S3 для зберігання файлів, а також сильній підтримці розробки, моніторингу, та масштабування системи. Використання AWS забезпечує стабільність, надійність та безпеку проєкту.

Ось деякі переваги та недоліки AWS та обгрунтування його придатності для розробки серверної частини системи.

Переваги AWS:

- Забезпечує високу доступність та надійність з механізмами збереження даних, автоматичного масштабування та відновлення після збоїв.
- Багато зовнішніх бібліотек та модулів, що сприяє ефективній розробці та швидкому розгортанню серверних додатків.
- Активна спільнота розробників і відкрите програмне забезпечення забезпечують підтримку та оновлення платформи.

- Високий рівень безпеки з вбудованими механізмами автентифікації, авторизації, шифрування даних та іншими засобами захисту.
- AWS пропонує широкий спектр сервісів як EC2, S3, RDS, Elastic Beanstalk, які дозволяють розробникам легко розгортати, масштабувати та керувати додатками.

Недоліки AWS:

- Може бути дорогим для деяких проєктів, особливо якщо вони вимагають значних ресурсів.
- Може бути складним для освоєння новачками, особливо тим, хто не має досвіду з хмарними платформами.

Як база для розробки серверної частини системи, AWS є ідеальним вибором через:

- Підтримка від широкої та активної спільноти розробників, що сприяє надійній підтримці та оновленню платформи.
- Високий рівень безпеки з автентифікацією, авторизацією та захистом даних, що забезпечує безпечну роботу серверної частини.
- Високу доступність та надійність з вбудованими механізмами збереження даних та відновлення після збоїв, забезпечуючи стабільність системи.
- Широкий спектр сервісів та інструментів для розробки, моніторингу та масштабування, що дозволяє розробникам швидко втілювати та адаптувати різні аспекти системи.

AWS є ідеальним вибором для розробки серверної частини системи через його високу доступність, надійність, високий рівень безпеки, різноманітність сервісів для розробки, моніторингу, та масштабування, що дозволяє розробникам ефективно управляти інфраструктурою і швидко розгортати серверні додатки.

2.1.8. *Висновки*

У підсумку, для серверної частини системи було обрано TypeScript як мову програмування, Node.js як середовище виконання, Nest.js як фреймворк,

MongoDB як базу даних, Mongoose як бібліотеку для роботи з MongoDB, і AWS з його сервісами S3 і SES як хмарну платформу. Всі ці інструменти були вибрані на основі їхньої гнучкості, продуктивності та підтримки, що є критично важливими для успішної реалізації проєкту.

2.2 Вибір та обґрунтування інструментарію для клієнтської частини системи

Front end відіграє ключову роль у розробці вебсайтів і додатків, зосереджуючись на візуальному вигляді, інтерфейсі та взаємодії з користувачем. Головна мета front-end розробки полягає в створенні інтерфейсу, який є зручним, інтуїтивно зрозумілим та естетично привабливим для користувачів.

Для розробки використовуються технології такі як HTML, CSS і JavaScript.

- HTML (HyperText Markup Language) створює структуру сторінки.
- CSS (Cascading Style Sheets) відповідає за стилі та оформлення.
- JavaScript додає динаміку та інтерактивність.

Можливості Front end розширюються за допомогою:

- Розробка інтерактивних елементів, таких як кнопки та слайдери.
- Взаємодія з сервером через AJAX для оновлення сторінки без перезавантаження.
- Використання фреймворків і бібліотек, наприклад, React, Angular, та Vue.js, для спрощення розробки.
- Створення адаптивного дизайну для різних пристроїв і розмірів екранів.
- Реалізація анімацій та переходів для покращення користувацького досвіду.

Популярність Front end зростає з кількох причин:

- Швидкий розвиток технологій: динамічний прогрес у front-end технологіях веде до новацій, які роблять роботу розробників що захоплює.

- Кросплатформність: можливість створювати додатки, що працюють на різних платформах.
- Творчість та візуальні можливості: front-end дозволяє розробникам використовувати свої творчі здібності, працюючи з дизайном та візуалізацією.
- Важливість користувацького досвіду: покращення користувацького інтерфейсу прямо впливає на успіх продукту.
- Зростання вебсайтів і додатків: збільшення використання інтернет-технологій сприяє попиту на front-end розробників.

Таким чином, front end розробка є важливою частиною створення вебсайтів і додатків, що впливає на залученість користувачів, інноваційність та ефективність продуктів, забезпечуючи різнобічний розвиток і зростання цієї галузі.

2.1.9. *Застосування бібліотеки React*

React було вибрано для розробки клієнтської частини системи через його легкість, швидкість та гнучкість у використанні. Ця бібліотека JavaScript дозволяє створювати високоякісні користувацькі інтерфейси, що робить її ідеальною для сучасних вебдодатків

Ось деякі з переваг та недоліків використання React:

Переваги:

- Підтримка від Facebook: Розробка та підтримка React здійснюються Facebook, забезпечуючи бібліотеці стабільність та постійне оновлення.
- Добре документованість: Як одна з найпопулярніших бібліотек JavaScript, React підтримується широкою документацією та активною спільнотою.
- Компонентна архітектура: React спирається на компоненти, що дозволяє розбивати комплексний інтерфейс на менші, повторно використовувані блоки, спрощуючи управління кодом та його розуміння.

- Висока продуктивність: Завдяки використанню Virtual DOM, React мінімізує взаємодію з реальним DOM, значно підвищуючи швидкість роботи додатків.

Недоліки:

- Обмежена функціональність: Як бібліотека, React вимагає додаткових інструментів, таких як Redux, для повного керування станом додатків.
- Специфічний синтаксис: JSX, який використовується у React, може бути незвичним для розробників, звиклих до традиційного JavaScript.
- Висока крива навчання: Новачкам може бути складно освоїти React через необхідність знання JSX, Virtual DOM та інших специфічних концепцій.

React оптимально підходить для розробки клієнтської частини системи завдяки його спроможності забезпечувати високу продуктивність і масштабованість. Це особливо важливо для клієнтських додатків, де користувачі очікують швидкої реакції та плавної взаємодії. Крім того, React сприяє ефективному управлінню станом додатка, зменшенню серверних запитів і створенню більш реактивних користувацьких інтерфейсів. Його гнучкість дозволяє легко інтегрувати з іншими бібліотеками або фреймворками, забезпечуючи широкі можливості для розширення функціональності додатків.

2.1.10. *Застосування бібліотеки Redux*

Redux було обрано для управління станом додатка завдяки його здатності до однонапрямного потоку даних, що значно спрощує відстеження змін стану та допомагає уникати помилок у управлінні станом. Ось деякі ключові переваги та недоліки використання Redux:

Переваги:

- Простота тестування: Через централізоване зберігання стану, тестування стає більш прямолінійним, оскільки можливо легко маніпулювати та перевіряти різні стани.
- Розширюваність: Redux дозволяє легко розширювати функціональність за допомогою плагінів та middleware, адаптуючи додаток до різних вимог і сценаріїв використання.

- Інтеграція з React: Redux ефективно інтегрується з React, надаючи зручний інтерфейс для зв'язування React компонентів зі станом додатка.
- Централізоване управління станом: Redux зберігає всі стани додатка в одному місці, що полегшує управління та розуміння структури додатка, а також сприяє зменшенню помилок.

Недоліки:

- Складність освоєння: Для новачків Redux може здатися складним через потребу в освоєнні концепцій, таких як reducers, actions, store.
- Збільшення обсягу коду: Використання Redux часто призводить до необхідності писати додатковий код для дій, редукторів і мідлварів.
- Необхідність додаткового коду: Інтеграція Redux в проєкт вимагає написання додаткових конфігурацій та обгорток, що може зайняти додатковий час.

Redux є відмінним рішенням для управління станом в клієнтській частині системи, особливо великих додатків, де потрібна чітка організація стану та висока надійність. Централізоване управління станом полегшує підтримку та масштабування додатка, а також сприяє ефективному взаємодію між компонентами. Крім того, зменшення звернень до сервера та можливість локального зберігання даних покращують продуктивність та відгук додатка. Використання Redux разом з React сприяє створенню стабільних, підтримуваних і гнучких рішень.

2.1.11. *Застосування фреймворку Bootstrap [39]*

Bootstrap було вибрано як фреймворк для розробки інтерфейсу користувача завдяки його широкому набору готових компонентів та стилів, які дозволяють швидко створювати естетично привабливі та адаптивні інтерфейси. Завдяки великій спільноті розробників, Bootstrap також гарантує постійне оновлення та підтримку.

Ось ключові переваги та недоліки використання Bootstrap:

Переваги:

- Готові компоненти: Наявність готових до використання компонентів, таких як кнопки, форми і меню, спрощує та прискорює процес розробки.
- Велика спільнота: Велика спільнота розробників надає обширні ресурси, документацію та підтримку.
- Мобільна адаптивність: Bootstrap включає вбудовану підтримку мобільних дизайнів, що дозволяє додаткам автоматично адаптуватися до різних пристроїв.
- Швидкість та легкість використання: Bootstrap надає засоби для швидкого створення користувацьких інтерфейсів, уникаючи необхідності писати багато власного CSS.

Недоліки:

- Обмежений контроль над дизайном: Фреймворк має вбудовані стилі та компоненти, які можуть бути важко модифікувати для досягнення унікального дизайну.
- Стандартизований дизайн: Використання Bootstrap може призвести до того, що додатки будуть виглядати дуже схоже один на одного без значних кастомізацій.
- Потреба у знаннях HTML та CSS: Для ефективного використання Bootstrap необхідно мати базові знання HTML та CSS.

Bootstrap відмінно підходить для розробки клієнтської частини системи, дозволяючи швидко створювати стильні та мобільно-респонсивні користувацькі інтерфейси. Це забезпечує можливість зосередитися на функціональності, а не на створенні CSS з нуля. Завдяки широкому набору готових компонентів та шаблонів, Bootstrap зменшує час розробки та забезпечує сумісність з різними пристроями, а також сприяє підвищенню продуктивності за рахунок оптимізованого коду. Враховуючи ці переваги, Bootstrap стає популярним вибором для розробки клієнтської частини систем.

2.1.12. *Застосування бібліотеки Ant Design (ANTD) [40]*

Ant Design (ANTD) було вибрано для підтримки інтерфейсу користувача завдяки його якісним компонентам, зручному API, та зрозумілій документації. ANTD забезпечує можливості для створення взаємодійних інтерфейсів, що спрощує розробку та підтримку коду через компонентний підхід.

Переваги використання ANTD включають:

- Гнучкість: ANTD можна адаптувати до специфічних потреб проекту, що забезпечує велику варіативність у дизайні.
- Підтримка спільноти: ANTD підтримується широкою спільнотою розробників, що сприяє її розвитку і забезпечує регулярні оновлення та підтримку.
- Легкість інтеграції: Завдяки сумісності з React, ANTD може бути швидко інтегрована в проекти, що зменшує час розробки.
- Широкі можливості дизайну: ANTD надає різноманітні готові компоненти та шаблони, які допомагають створювати комплексні користувацькі інтерфейси.

Недоліки використання ANTD включають:

- Потреба в знаннях HTML та CSS: Для оптимального використання ANTD потрібні базові знання HTML та CSS.
- Складність освоєння: Високий поріг входу може ускладнити використання ANTD для новачків.
- Збільшення розміру проєкту: Велика кількість компонентів та залежностей може збільшити загальний обсяг проєкту.

ANTD підходить для розробки клієнтської частини системи завдяки його ефективності у створенні стильних та респонсивних інтерфейсів. Це дозволяє розробникам зосередитись на функціональності додатка, знижуючи необхідність витрачати час на створення стилів. Також, ANTD сприяє оптимізації продуктивності додатка, забезпечує доступність для різних пристроїв і дозволяє швидко інтегрувати компоненти в проєкти на базі React. Враховуючи ці переваги,

ANTD є вибором багатьох розробників для створення клієнтських частин вебсистем.

Висновки

Для розробки серверної частини системи оптимальним вибором стали технології TypeScript та Nest.js для програмування, MongoDB та Mongoose для менеджменту даними, а також AWS для деплойменту та забезпечення стабільності, надійності та безпеки проєкту. Сервіси, як AWS S3 для зберігання даних у хмарі та AWS SES для відправлення електронних листів, також використовуються для підвищення ефективності. Це дозволяє знизити кількість помилок, підвищити безпеку та надійність загалом. Завдяки цим технологіям розробники можуть ефективно керувати інфраструктурою, забезпечуючи стабільність проєкту та швидке розгортання серверних додатків.

Для клієнтської частини системи використовуються TypeScript та React для програмування, Redux для керування станом додатка, Bootstrap та Ant Design для стилізації та дизайну інтерфейсу. Ці технології забезпечують високу ефективність, швидкість та гнучкість у розробці, поліпшують зручність користування, та гарантують стабільність і безпеку проєкту. Вони дозволяють розробникам швидко створювати гнучкі, адаптивні та інтерактивні інтерфейси, оптимізувати роботу додатка та забезпечити високу продуктивність та зручність навігації.

Використання вказаних технологій для серверної та клієнтської частин забезпечує не лише технічну відмінність проєкту, але й згуртованість двох частин системи, дозволяючи розробникам ефективно інтегрувати функціональність та створювати інтуїтивно зрозумілий користувацький інтерфейс.

РОЗДІЛ 3
ОРГАНІЗАЦІЯ СТРУКТУРИ ТА АЛГОРИТМІВ РОЗРОБЛЕНОЇ СИСТЕМИ

3.1 Загальний огляд вебдодатку

Вебдодаток для менеджменту паролями було створено на основі технологій Nest та React (з використанням TypeScript), використовуючи MongoDB для зберігання даних. Ця система надає безпечне середовище для зберігання та менеджменту паролями, як для окремих осіб, так і для організацій.

Додаток організовано навколо концепції "сховищ", що є віртуальними контейнерами для зберігання різноманітних даних, таких як логіни, паролі, банківські дані, та SSH ключі. Система працює за моделлю "клієнт-сервер" і включає:

1. Клієнтську частину: Розроблено з використанням React.js, який опирається на TypeScript, забезпечує користувацький інтерфейс, представлення даних та взаємодію з користувачем.
2. Серверну частину: Будується на Nest.js з використанням TypeScript, відповідає за обробку запитів, менеджменту даними та безпеці.

Ці компоненти взаємодіють через API, що дозволяє обмінюватися даними у форматі JSON.

Основні модулі серверної частини включають:

1. Модуль управління компаніями та проєктами: Дозволяє керувати проєктами й користувацькими ролями в них.
2. Модуль управління даними: Забезпечує обробку та зберігання користувацьких даних із використанням шифрування для безпеки.
3. Модуль авторизації та аутентифікації: Відповідає за реєстрацію користувачів та їхню авторизацію.
4. Модуль управління сховищами: Відповідає за управління віртуальними сховищами та шифруванням збережених даних.

MongoDB, як високопродуктивна NoSQL база даних, використовується для швидкої обробки запитів і легкого масштабування. Система розгортається у хмарному середовищі, забезпечуючи високу доступність і масштабованість.

Безпека даних є пріоритетною, з шифруванням передачі та зберігання інформації та використанням двофакторної аутентифікації для захисту облікових записів. Система також розроблена згідно з найкращими практиками програмування та тестування, що забезпечує надійність і стабільність рішення.

3.2 База даних вебдодатку

Для відповідності поставленим вимогам, вебдодаток має забезпечувати ефективну взаємодію з базою даних. У структурі вебдодатку присутні наступні колекції документів: Users, Storages, Files, Verifications, Documents, Companies, UsersCompanies, Projects, UsersProjects, та Invites. Вебдодаток використовує NoSQL базу даних MongoDB, а для роботи з нею застосовується бібліотека Mongoose.

1. ***Users***:
    - firstName: Ім’я користувача.
    - lastName: Прізвище користувача.
    - fullName: Ім'я та прізвище користувача.
    - email: Унікальна пошта користувача.
    - isVerified: Чи верефіковааний користувач.
    - password: Пароля користувача.
    - salt: Сіль для хешування пароля.
2. ***Storages***:
    - title: Назва сховища.
    - description: Деталі сховища.
    - userId: Ідентифікатор користувача, якому належить сховище.
    - projectId: Ідентифікатор проєкту, якщо це належить компанії.

3. ***Files***:
    - id: Унікальний ідентифікатор файлу.
    - name: Назва файлу.
    - type: Тип файлу.
    - size: Розмір файлу.
4. ***Documents***:
    - title: Назва ресурсу.
    - userId: Ідентифікатор користувача, якому належить документу.
    - storageId: Ідентифікатор сховища, в якому знаходиться документу.
    - attachments: Файли, що були додані до документу.
5. ***Companies***:
    - name: Назва компанії.
    - description: Деталі компанії.
    - email: Унікальна пошта компанії.
    - userId: Ідентифікатор користувача, який створив компанію.
6. ***UsersCompanies***:
    - userId: Ідентифікатор користувач.
    - companyId: Ідентифікатор компанії.
    - role: Роль користувача в компанії.
7. ***Projects***:
    - id: Унікальний ідентифікатор проєкту.
    - name: Назва проєкту.
    - description: Опис проєкту.
    - type: Тип проєкту.
    - companyId: Ідентифікатор компанії.
    - userId: Ідентифікатор користувача.
8. ***UsersProjects***:
    - userId: Ідентифікатор користувача.
    - projectId: Ідентифікатор проєкту.

9. ***Verifications***:
    - key: Унікальний ключ для верифікації.
    - type: Тип верифікації.
10. ***Invites***:
    - key: Унікальний ключ для запрошення.

Mongoose дозволяє структурувати дані через схеми, де можна вказати атрибути даних, такі як тип, унікальність, обов’язковість, а також визначити користувацькі валідатори. Ця бібліотека, як ODM (Object Document Mapper) для MongoDB і Node.js, спрощує роботу з даними, представляючи їх як об'єкти, та дозволяє виконувати операції CRUD (створення, читання, оновлення, видалення) через методи, пов’язані з моделями.

У MongoDB зв'язки між документами можуть бути реалізовані через посилання (Reference), де в одному документі зберігається ID іншого документа, або через вкладені документи (Embedded). Mongoose надає можливість використовувати метод populate, який дозволяє автоматично "заповнювати" ці посилання даними з відповідних документів, що робить обробку складних запитів значно зручнішою.

3.3 Детальний опис архітектури вебдодатку

Для створення клієнтської частини було вибрано підхід односторінкового додатка (Single Page Application, SPA), який дозволяє оновлювати вміст сторінки без її повторного завантаження. Використання AJAX-запитів, які ініціюються відповідно до дій користувача, дозволяє динамічно отримувати інформацію від сервера і оновлювати її на сторінці.

Цей метод вже встановився як ефективний і став звичним стандартом у сфері вебтехнологій, що зумовило його вибір для реалізації користувацької частини цього проєкту. Ураховуючи загальну архітектуру проєкту, було вирішено застосувати концепцію мікросервісів, яка зараз є широко використовуваною інфраструктурою в сучасних проєктах. Це дозволяє ефективно масштабувати та

управляти системою. Архітектурна схема вебдодатку представлена на рисунку 3.1.

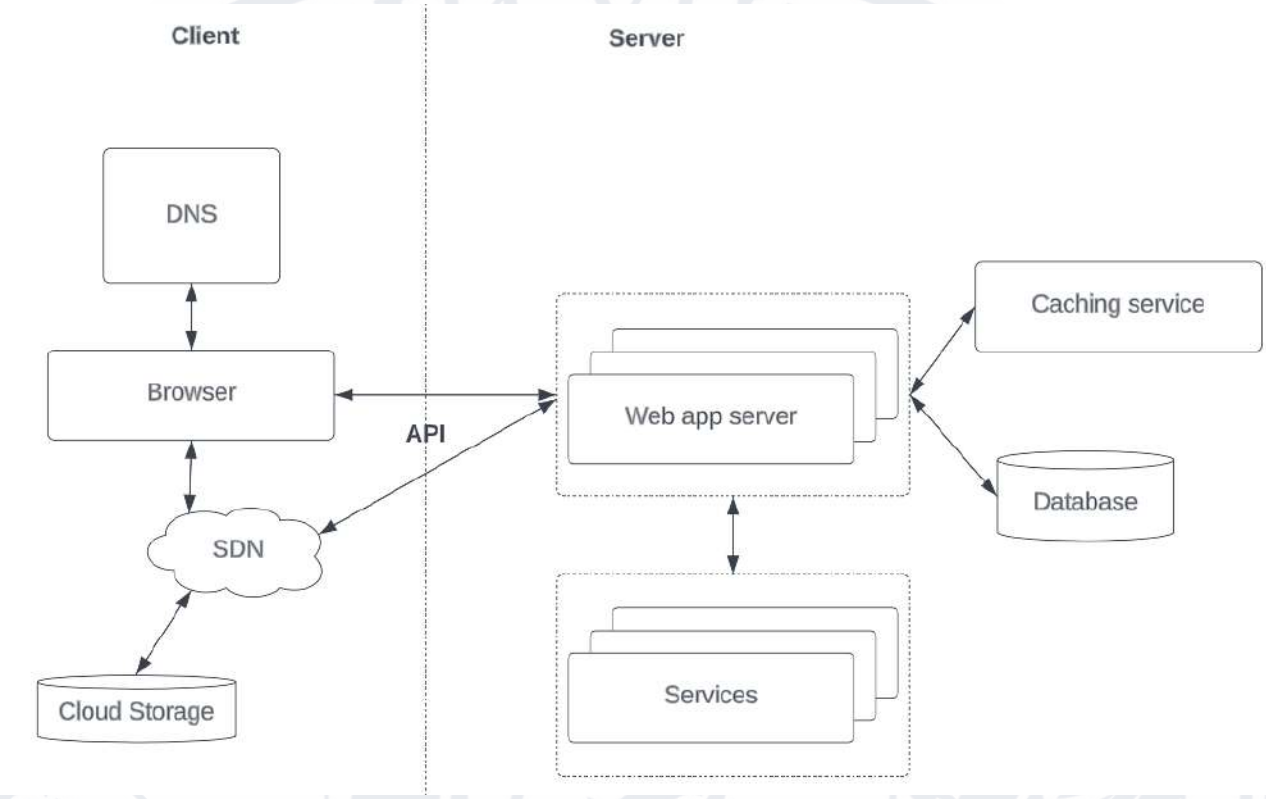

Рисунок 3.1 – Архітектура вебдодатку

В архітектурі даного проєкту значну увагу приділено процесу авторизації та захисту API-серверів. Клієнтська частина системи (UI) взаємодіє з сервером через HTTP-запити, які спочатку обробляються через мікросервіс API Gateway, для чого використовується NGINX. Цей компонент діє як прозорий посередник, що перенаправляє запити до відповідних серверів, дозволяючи їм спілкуватися з внутрішньою мережею без прямої взаємодії з публічною мережею.

NGINX у якості API Gateway слугує єдиною точкою входу, що уніфікує обробку запитів і забезпечує їх правильне спрямування до різних мікросервісів системи. Це дозволяє забезпечити співпрацю мікросервісів, написаних на різних мовах програмування, що спрощує інтеграцію та розширення системи.

В якості високопродуктивному вебдодатку та проксі, NGINX ідеально підходить для ролі API Gateway, забезпечуючи стабільність з'єднань та виконання

численних функцій, таких як балансування навантаження, управління сесіями, захист від DDoS-атак, та інше.

NGINX також використовується для моніторингу стану системи та виконання активного та пасивної перевірки стану компонентів системи, що допомагає оперативно виявляти та реагувати на потенційні проблеми.

Завдяки незалежності та високій ефективності як проксі-сервера, NGINX також ефективно розподіляє навантаження між компонентами системи, застосовуючи різноманітні стратегії балансування навантаження. Ця здатність NGINX забезпечує не тільки продуктивність, але й масштабованість системи.

Окрім того, NGINX виконує важливу роль у забезпеченні додаткового шару безпеки, використовуючи автентифікаційні та авторизаційні модулі для захисту доступу до системних ресурсів.

Використання NGINX в проєкті забезпечує не лише масштабованість і високу продуктивність, але й допомагає забезпечити високий рівень безпеки та стабільність, що підтверджує правильність вибору цієї технології для реалізації архітектури проєкту.

3.4 Алгоритм автентифікації користувачів у вебдодатку

Процес авторизації у вебдодатку складається з кількаетапної процедури, яка включає використання електронної пошти та пароля для первинної ідентифікації користувача, а також застосування верифікаційних кодів та токенів для подальшої взаємодії з системою.

Етап 1: Введення даних

Спочатку користувач вводить свою електронну адресу та пароль на сайті, після чого ці дані передаються на сервер через захищене з'єднання API. Важливо захищати ці дані, оскільки вони є чутливою інформацією.

Етап 2: Перевірка даних

На сервері відбувається перевірка, чи існує користувач з такою електронною адресою. Якщо так, система перевіряє, чи відповідає введений пароль захищеному хешу пароля в базі даних.

Етап 3: Верифікація кодом

У разі успішної верифікації пароля, на електронну пошту користувача надсилається верифікаційний код. Користувач має ввести цей код на сайті для продовження авторизації.

Етап 4: Помилки й повторні спроби

Якщо дані користувача не відповідають збереженим у системі даним, видається повідомлення про помилку. З міркувань безпеки система не уточнює, які саме дані введено некоректно.

Етап 5: Видача токенів

Після успішної верифікації користувач отримує два токени: access token для доступу до додатку і refresh token для оновлення access token, коли його термін дії закінчиться.

Етап 6: Взаємодія з API

З наявними токенами користувач може безпечно взаємодіяти з API системи. Кожен запит містить access token, який підтверджує авторизацію користувача.

Етап 7: Оновлення токенів

Коли термін дії access token закінчується, refresh token використовується для його оновлення. Якщо refresh token втрачений або компрометований, користувачу необхідно знову пройти авторизацію.

Цей детальний процес авторизації дозволяє забезпечити безпечний доступ до вебдодатку, гарантуючи захист даних користувачів і високий рівень безпеки системи.

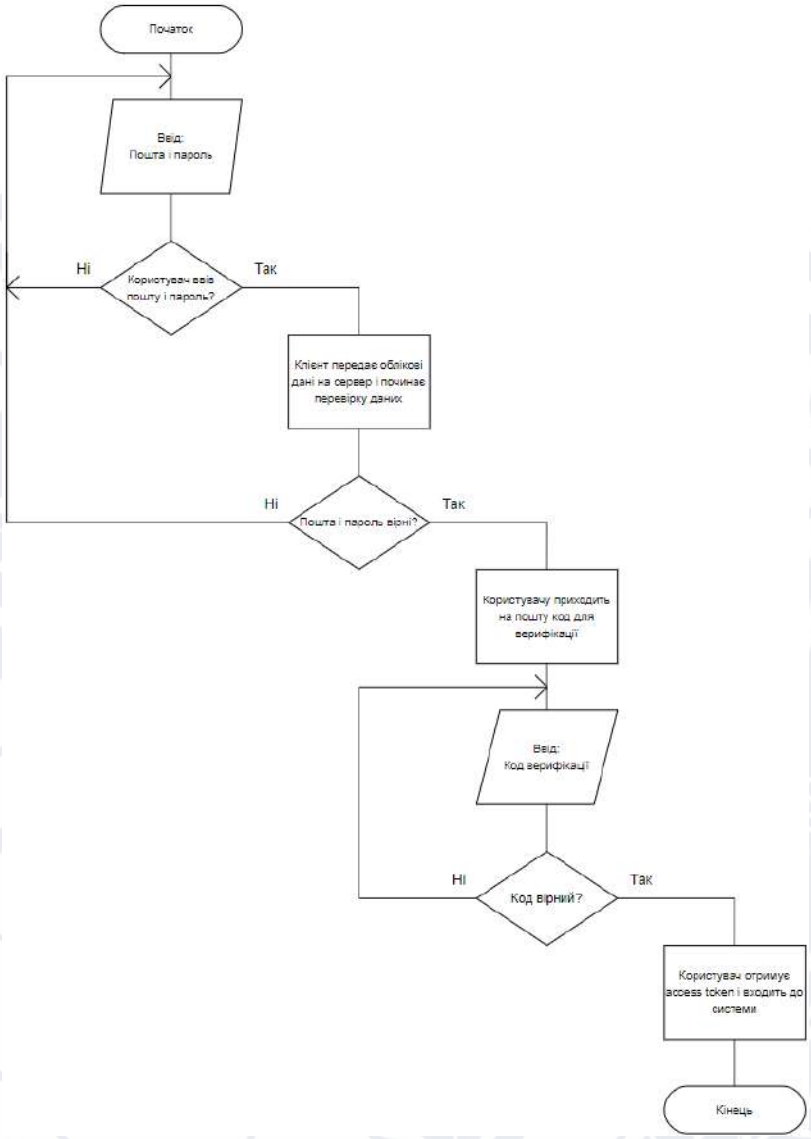

Рисунок 3.2 – Алгоритм автентифікації користувача у вебдодатку

3.5 Алгоритми створення та зберігання паролів

3.1.1. *Методика генерації надійних паролів*

Забезпечення сильного пароля є критично важливим для захисту системи від несанкціонованого доступу. Для створення ефективного пароля, який поєднує великі та малі літери, цифри, та спеціальні символи, та має довжину не менш як 24 символів, застосовується такий алгоритм:

1. Визначення довжини пароля: Налаштовуємо мінімальну кількість символів у паролі на 24, щоб забезпечити його складність.
2. Конфігурація символьних наборів: Створюємо чотири окремі масиви символів для малих літер, великих літер, цифр, та спеціальних символів.

3. Генерація початкових символів: Вибираємо по одному випадковому символу з кожного масиву за допомогою функції випадкового вибору, щоб забезпечити присутність у паролі різних типів символів.
4. Доповнення пароля: Продовжуємо додавати випадкові символи з усіх масивів до тих пір, поки не досягнемо потрібної довжини пароля.
5. Перемішування пароля: Рандомізуємо порядок символів у паролі для усунення будь-яких зрозумілих шаблонів.
6. Перевірка пароля: Перевіряємо, чи відповідає сформований пароль усім вимогам до сильного пароля. У разі необхідності, повторюємо процес генерації.

Цей метод (лістинг 3.1) гарантує створення надійних та безпечних паролів, що є важко зламати або вгадати завдяки їх високому рівню складності.

Лістинг 3.1. Приклад алгоритму генерації найдіного пароля

```
let createNewPassword = function ({ includeLowercase, includeUppercase, includeNumbers, includeSymbols, maxlong }) {
 let str = '';
 if (includeLowercase) srr += 'hftgeyfjl;dfgmolhiabbkd';
 if (includeUppercase) str += 'OIGSRFOSEHFOONSDOFN';
 if (includeNumbers) str += '26456345345';
 if (includeSymbols) str += '*^&*^&%&^%&^';

 let password = '';
 for (let i = 10; i < maxlong + 10; i++) {
  password += str(Math.floor(Math.random() * maxlong)];
 }

 return password;
};
```

3.1.2. *Методика збереження паролів*

Ефективне зберігання паролів вимагає строгих заходів безпеки, оскільки ці дані відкривають доступ до конфіденційної інформації користувачів. Система зберігання паролів, яку ви використовуєте, базується на шифруванні та детальній ізоляції даних.

1. Шифрування даних: Коли користувачі вводять свої облікові дані, такі як імена користувачів, паролі, банківські картки, або SSH ключі, система негайно шифрує цю інформацію. Шифрування відбувається на стороні клієнта за допомогою передових алгоритмів, забезпечуючи надійний захист від несанкціонованого доступу.
2. Зберігання зашифрованих даних: Зашифровані паролі зберігаються на віддаленому сервері Amazon S3, який відзначається високою надійністю і безпекою. Використання Amazon S3 дозволяє ізолювати чутливі дані від інших користувацьких даних, знижуючи ризики витоку інформації.
3. Зберігання ключів шифрування: Ключі, необхідні для розшифровки даних, зберігаються окремо в базі даних. Ці ключі єдині засоби для доступу до зашифрованих даних, і без відповідних ключів доступу до сервісу Amazon S3, неможливо отримати файл з зашифрованими даними.
4. Доступ до даних: Коли користувач запитує перегляд своїх даних, система звертається до серверної частини для генерації тимчасового лінку з доступом, що триває лише 5 секунд, що дозволяє завантажити і розшифрувати потрібні дані. Ця процедура забезпечує, що навіть у випадку компрометації сервера або сховища, отримання доступу до реальних даних користувачів є малоймовірним.

Лістинг 3.2. Приклад алгоритму шифрування даних

```
import * aes from 'aes';
funcrion genRanB(length: number) {
  const newArray = new Uint8Array(length);
```

```
  for (let i = 0; i < length; i++) {
    newArray[i] = Math.floor(Math.random() * 256);
  }
  return newArray;
}
funcrion enc(field: string) {
  var by = aesjs.utils.utf8.toBytes(field);
  var ss = genRanB (45);
  var aes = aes.ModeOfOperation.ctr(aes.Counter(ss));
  var ecBy = aes.encrypt(by);
  return (aes.utils.hex.fromBytes(ecBy));
}
```

Таким чином, застосування цих методів формує високо захищену систему зберігання паролів, яка гарантує збереження конфіденційності користувацьких даних на всіх етапах їх обробки.

РОЗДІЛ 4
АНАЛІЗ ВИКОНАННЯ ВЕБДОДАТКУ

4.1 Аналіз реалізації вебдодатку для менеджменту паролями

Концепція вебдодатку полягає у створенні ефективного рішення для менеджменту паролями, що включає зберігання паролів, їх генерацію, двофакторну автентифікацію (2FA), а також можливість колективного доступу для організаційних користувачів.

Вебдодаток використовує сучасні техніки шифрування для безпечного зберігання паролів, що забезпечує захист користувацької інформації, навіть у випадку потенційних кібератак.

Також, додаток містить вбудований генератор паролів, що дозволяє користувачам створювати складні паролі, збільшуючи таким чином їх безпеку. Генератор пропонує кілька параметрів, таких як довжина пароля та використання різних символів.

Вдосконалення захисту досягається через впровадження двофакторної автентифікації, яка вимагає від користувачів додаткової перевірки їхньої особи через верифікаційні коди чи унікальні посилання.

Особливою функцією вебдодатку є можливість колективного доступу до облікових записів у корпоративному середовищі, що дозволяє зручно та безпечно ділитися ресурсами. Контроль за спільним доступом здійснюється адміністраторами, що сприяє ефективному менеджменту ресурсами.

Ці функції підтримуються сучасною JWT-архітектурою автентифікації, яка забезпечує безпечне взаємодію між користувачем та сервером.

Загалом, вебдодаток для менеджменту паролями розроблений з акцентом на безпеку, легкість доступу та зручність користувачів.

4.1.1. Імплементація шифрування парольних моделей

Шифрування моделей паролів є ключовою функцією вебдодатка, забезпечуючи безпеку даних користувача. Для шифрування використано алгоритм AES, який є загальнодоступним і схваленим Агентством національної

безпеки США (NSA) для захисту секретної інформації, коли він використовується у схвалених NSA криптографічних модулях. AES замінив стандарт шифрування даних (DES) і використовує симетричний ключ для шифрування та дешифрування даних.

Процес шифрування містить наступні кроки:

1. Вибір моделі пароля: Користувач обирає конкретний тип моделі пароля для додавання даних, наприклад, "Ключ Значення", "Логін", "Банківська карта", або "SSH ключі".
2. Введення облікових даних: В залежності від обраної моделі, користувач вводить відповідні дані, такі як номер картки, дата закінчення, CVV, та інші.
3. Локальне шифрування: Введені дані шифруються безпосередньо на пристрої користувача перед їх передачею на сервер. Це запобігає можливості перехоплення чутливих даних у незашифрованому вигляді.
4. Створення підпису для файлу на S3 Bucket: Система автоматично генерує унікальний ключ для файлу, який використовується для створення підпису через AWS-SDK. Це надає тимчасові права для завантаження файла у приватне сховище S3, гарантуючи, що доступ має лише власник.
5. Зберігання зашифрованих даних на S3 Bucket: Після локального шифрування, зашифровані дані зберігаються у S3 Bucket, що відомий своєю надійністю та безпекою для об'єктного зберігання.
6. Зберігання ключа доступу до файлу: Назва унікального ключа файлу, що використовується для доступу до зашифрованих даних, зберігається в базі даних системи. Це забезпечує контрольований доступ до зашифрованої інформації.

Завдяки цим крокам, процес шифрування моделей паролів у вебдодатку гарантує надійний захист облікових даних користувача на всіх етапах використання додатку.

4.1.2. Управління корпоративними аспектами

Управління компанією є критичною функцією вебдодатка, що охоплює такі етапи:

1. Створення компанії: Користувачі мають можливість створити компанію, вказавши інформацію, таку як назву, електронну адресу та опис компанії.
2. Верифікація компанії: Після створення, компанія мусить бути верифікована через електронну адресу. Система відправляє лист з посиланням для верифікації на вказану електронну адресу, і користувач має підтвердити свою електронну адресу, перейшовши за посиланням.
3. Призначення ролі Власника: Після верифікації, ініціатор створення компанії автоматично стає Власником компанії і отримує права для керування компанією, включно з можливістю запрошувати нових учасників і призначати їм ролі.
4. Запрошення учасників: Власник може відправляти запрошення новим учасникам за допомогою їхніх електронних адрес. Отримувачі мають підтвердити участь, перейшовши за посиланням у листі.
5. Управління ролями: Власник та адміністратори можуть керувати ролями учасників, змінюючи їхній статус між Користувачем та Адміністратором або виключаючи їх з компанії.
6. Керування проєктами: Власники та адміністратори можуть створювати та управляти проєктами всередині компанії, встановлюючи сховища паролів для зберігання і менеджменту моделями паролів.
7. Долучення учасників до проєктів: Власники та адміністратори можуть додавати учасників до проєктів, надаючи їм доступ до конкретних ресурсів і сховищ.
8. Створення моделей паролів: Учасники з відповідними правами доступу можуть створювати нові моделі паролів у сховищах проєктів, забезпечуючи зберігання необхідних даних у централізованому місці.
9. Перегляд існуючих моделей паролів: Учасники з відповідними правами можуть переглядати всі деталі існуючих моделей паролів, забезпечуючи прозорість і доступ до критичної інформації.

10. Менеджмент існуючими моделями паролів: Учасники можуть змінювати або видаляти моделі паролів, налаштовуючи або оновлюючи інформацію за потребою.

Розробка функціоналу управління компанією в рамках вебдодатку для менеджменту паролями дозволила впровадити ключові можливості, які сприяють більшій гнучкості, безпеці та ефективності в керуванні паролями на корпоративному рівні:

1. Власник компанії: Користувач, який ініціює створення компанії, автоматично приймає роль Власника, отримуючи розширені можливості для керування компанійними ресурсами.
2. Управління учасниками: Власник та Адміністратори компанії володіють правами на управління статусами учасників, від призначення ролей до видалення з компанії.
3. Керування проєктами: Власник і Адміністратори мають повноваження створювати та керувати проєктами, в рамках яких можна зберігати і управляти сховищами паролів.
4. Доступ до проєктів: Власник та Адміністратори забезпечують допуск до проєктів, дозволяючи учасникам доступ до важливих сховищ паролів.
5. Створення та менеджмент моделями паролів: Учасники з правом доступу можуть створювати та керувати моделями паролів, що забезпечує централізований менеджмент важливими даними.

Ці характеристики забезпечують потужний контроль та адаптивність в менеджменті паролями, підвищуючи ефективність та безпеку обробки даних на рівні корпорації.

4.2 Тестування функціональності вебдодатку

У цьому розділі описано методику тестування вебдодатку, зокрема димове тестування, яке є ключовим для забезпечення коректної роботи вебдодатку після внесення змін.

Димове тестування використовується як частина процесів неперервної інтеграції та постійного розгортання (CI/CD) і допомагає швидко оцінити, чи функціонує вебдодаток відповідно до очікувань після оновлень. Цей метод включає автоматизоване проведення базового набору тестів, спрямованих на виявлення критичних дефектів, що можуть виникнути внаслідок змін в вебдодатку.

Головна мета димового тестування полягає в активації автоматичних тестів на вирішальних етапах функціонування або інтеграції вебдодатку, що дозволяє виявити ймовірні проблеми з продуктивністю, некоректними відповідями, доступністю та іншими ключовими аспектами, важливими для нормального функціонування вебдодатку.

Виконання димового тестування включає розробку серії автоматизованих тестів, які перевіряють основні сценарії взаємодії з вебдодатком. Ці тести здійснюються під час кожного оновлення вебдодатку, що забезпечує оперативне виявлення та вирішення проблем.

Для вебдодатку менеджменту паролями димове тестування може включати такі аспекти:

1. Перевірка доступності вебдодатку: перш за все, необхідно забезпечити, що вебдодаток доступний для користувачів. Це передбачає тестування підключення до сервера і перевірку доступності головної та інших ключових сторінок вебдодатку.
2. Аутентифікація та реєстрація користувачів: важливо перевірити, чи ефективно працюють функції реєстрації, входу в систему, виходу з системи та відновлення забутих паролів.
3. Створення та управління компанійними профілями: димове тестування повинно підтвердити правильну роботу основних операцій, пов’язаних з компанійними профілями, включаючи їх створення, редагування та управління учасниками та їхніми ролями.
4. Створення та управління проєктами та сховищами паролів: має бути забезпечено можливість створення та адміністрування проєктів, а також

додавання до них сховищ паролів, що є критичним аспектом димового тестування.

5. Перевірка функціональності моделей паролів: важливо переконатись, що користувачі можуть безпроблемно створювати, редагувати, шифрувати та видаляти моделі паролів.
6. Верифікація захисту даних: димове тестування також включає перевірку, що шифрування та зберігання даних виконуються належним чином, забезпечуючи захист даних, зокрема на базі даних і на S3 Bucket.
7. Перевірка інтерфейсу: переконатися, що інтерфейс є зрозумілим та інтуїтивно зручним для користувачів є важливим елементом димового тестування.
8. Тестування системи повідомлень: необхідно забезпечити, що користувачі отримують відповідні повідомлення про помилки та інші сповіщення від системи.
9. Перевірка продуктивності: важливо впевнитися, що вебдодаток здатен ефективно обробляти заплановане навантаження без збоїв в роботі.
10. Тестування на різних платформах та в різних браузерах: забезпечення стабільної роботи вебдодатку у різних браузерах і на різних платформах є частиною димового тестування, що включає перевірку сумісності та функціональності.

4.3 Пропозиції щодо щодо подальшого розвитку вебдодатку

У цьому розділі описано стратегії, спрямовані на поліпшення продуктивності, функціональності та користувацького досвіду вебдодатку. Аналізуючи результати тестування та загальний аналіз вебдодатку, можуть бути ідентифіковані потенційні слабкі сторони та аспекти, що потребують оптимізації або удосконалення.

Основні напрямки для вдосконалення включають:

- Покращення користувацького інтерфейсу: Наявний інтерфейс вже зручний, проте його можна оптимізувати, щоб підвищити простоту та

інтуїтивність використання. Зокрема, можна вдосконалити елементи навігації, пошуку, сортування та включити додаткові візуальні підказки та інструкції для користувачів.

- Розширення функціональності: Додати нові можливості до вже наявного набору функцій, таких як інтеграція з додатковими платформами, автоматизація резервного копіювання, більш гнучкий менеджмент доступу на основі ролей, а також покращені засоби для аналітики та звітності.
- Оптимізація продуктивності: З метою обслуговування збільшення числа користувачів та даних, важливо впровадити додаткові оптимізації, такі як кешування, оптимізація бази даних, та перегляд загальної архітектури системи для підтримки масштабування.
- Оновлення тестування та контролю якості: Постійне оновлення процесів тестування та контролю якості є критично важливим для забезпечення високої якості продукту. Це може включати розширення автоматизації тестування та регулярні перевірки безпеки.
- Розвиток спільноти користувачів: Активна спільнота користувачів може сприяти росту та популяризації вебдодатку. Важливо забезпечити організацію вебінарів, форумів та активну присутність у соціальних мережах.
- Підтримка багатомовності: Для залучення міжнародних користувачів важливо забезпечити підтримку декількох мов у вебдодатку, починаючи з найбільш вживаних світових мов. Розширення мовної підтримки може значно підвищити доступність та зручність користування вебдодатком для більш широкого кола користувачів.
- Вдосконалення служби підтримки: якість служби підтримки може Вдосконалення служби підтримки: Покращення служби підтримки може істотно вплинути на загальне задоволення користувачів. Важливо забезпечити швидку реакцію на запити, розширити базу знань, а також

впровадити технології чат-ботів для швидкого розв'язання типових питань.

- Підтримка мобільних платформ: Створення мобільних додатків для платформ iOS і Android може значно покращити зручність користування вебдодатком. Мобільні додатки забезпечують кращу інтеграцію з мобільними операційними системами та можуть використовувати такі технології, як біометрична ідентифікація та офлайн-доступ до даних.

ВИСНОВКИ

Мета даної роботи полягала у створенні вебдодатку для менеджменту паролями, забезпеченні безпеки збереження та ефективності менеджменту обліковими даними користувачів. Протягом проєкту було проведено аналіз сфери менеджменту паролями, розглянуто існуючі аналоги та визначено вимоги до розроблюваного програмного забезпечення. Вебдодаток розроблено за допомогою TypeScript, використовуючи React для клієнтської частини та Nest.js для серверної, з MongoDB як базою даних та AES для шифрування. Розроблений вебдодаток:

1. Забезпечує безпечне зберігання та шифрування користувацьких даних за допомогою сучасних криптографічних методів.
2. Дозволяє користувачам ефективно керувати паролями, оновлювати облікові дані та генерувати нові надійні паролі.
3. Включає менеджер компаній, що дозволяє керувати доступом до ресурсів, запрошувати та виключати учасників.
4. Реалізує системи авторизації та аутентифікації для розмежування доступу.
5. Дає можливість організувати облікові дані у проєкти та сховища паролів, полегшуючи їх менеджмент.
6. Гарантує високий рівень безпеки через двофакторну автентифікацію, знижуючи ризики несанкціонованого доступу.
7. Забезпечує шифрування даних на стороні клієнта перед їх зберіганням на сервері, що забезпечує конфіденційність.

Проект виконаний відповідно до технічного завдання, а його якість підтверджена ретельним тестуванням, включно з димовими тестами, що підтвердили функціональність вебдодатку. Запропоновано також шляхи для подальшого розвитку вебдодатку, включаючи розширення функціоналу, підвищення безпеки, інтеграцію з іншими сервісами, поліпшення користувацького інтерфейсу та оптимізацію загальної продуктивності.

Використання цього вебдодатку для менеджменту паролями здатне значно підвищити ефективність обробки облікових даних, зменшити ризики втрати або крадіжки інформації, а також спростити менеджмент паролями, особливо в корпоративному середовищі.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Password Statistics That Will Make You Change Your Bad Password Habits URL: https://techjury.net/blog/password-statistics/ (дата звернення: 17.05.2024)

2. Password Management (LastPass) URL: https://www.brandeis.edu/its/services/information-security/security-tools/password-management.html (дата звернення: 17.05.2024)

3. What is Dashlane? URL: https://support.dashlane.com/hc/en-us/articles/4679936285458-What-is-Dashlane- (дата звернення: 17.05.2024)

4. Keeper (password manager) URL: https://en.wikipedia.org/wiki/Keeper_(password_manager) (дата звернення: 17.05.2024)

5. Bitwarden Review 2023 — Open-Source, But Is It Good? URL: https://www.safetydetectives.com/best-password-managers/bitwarden/ (дата звернення: 17.05.2024)

6. 1Password Review URL: https://www.pcmag.com/reviews/agilebits-1password (дата звернення: 17.05.2024)

7. Nest.js documentation URL: https://docs.nestjs.com/ (дата звернення: 17.05.2024)

8. How JavaScript works: inside the V8 engine + 5 tips on how to write optimized code URL: https://blog.sessionstack.com/how-javascript-works-inside-the-v8-engine-5-tipson-how-to-write-optimized-code-ac089e62b12e (дата звернення: 17.05.2024)

9. TypeScript Documentation URL: https://www.typescriptlang.org/docs/ (дата звернення: 17.05.2024)

10. Express.js Tutorial URL: https://www.javatpoint.com/expressjs-tutorial (дата звернення: 17.05.2024)

11. Koa.js Documentation URL: https://koajs.com/ (дата звернення: 17.05.2024)

12. MongoDb in Action [Text] / K. Banker, P. Bakkum, S.Verch, D. Garreett, T. Hawkins. — 2nd edition. — Shelter Island: Manning, 2016. — 456 p. (дата звернення: 17.05.2024)

13. AWS SES Documentation URL: https://aws.amazon.com/ses/ (дата звернення: 17.05.2024)

14. AWS S3 Documentation URL: https://aws.amazon.com/s3/ (дата звернення: 17.05.2024)

15. ECMAScript 2015 Language Specification URL: https://262.ecma-international.org/6.0/ (дата звернення: 17.05.2024)

16. OOP documentation URL: https://developer.mozilla.org/en-US/docs/Glossary/OOP (дата звернення: 17.05.2024)

17. Webpack Documentation URL: https://devdocs.io/webpack/ (дата звернення: 17.05.2024)

18. Rollup Documentation URL: https://rollupjs.org/introduction/ (дата звернення: 17.05.2024)

19. Node.js Documentation URL: https://nodejs.org/en/docs (дата звернення: 17.05.2024)

20. gRPC, Restful API, GraphQL, Web Socket, TCP Sockets and UDP, WebTransport — Beyond modern client server communications URL: https://tech-lead.medium.com/grpc-vs-restful-api-vs-graphql-web-socket-tcp-sockets-and-udp-beyond-client-server-43338eb02e37 (дата звернення: 17.05.2024)

21. Model-View-Controller URL: https://docs.nestjs.com/techniques/mvc (дата звернення: 17.05.2024)

22. SOLID wikipedia URL: https://en.wikipedia.org/wiki/SOLID__(дата звернення: 17.05.2024)

23. Dependency Injection wikipedia URL: https://en.wikipedia.org/wiki/Dependency_injection (дата звернення: 17.05.2024)

24. JWT documentation URL: https://jwt.io/introduction (дата звернення: 17.05.2024)

25. NoSQL URL: https://uk.wikipedia.org/wiki/NoSQL (дата звернення: 17.05.2024)

26. Mongoose documentation URL: https://mongoosejs.com/docs/ (дата звернення: 17.05.2024)

27. AWS documentation URL: https://docs.aws.amazon.com/ (дата звернення: 17.05.2024)

28. Amazon EC2 URL: https://aws.amazon.com/ec2/ (дата звернення: 17.05.2024)

29. Amazon Relational Database Service URL: https://aws.amazon.com/rds/ (дата звернення: 17.05.2024)

30. AWS Elastic Beanstalk URL: https://aws.amazon.com/elasticbeanstalk/ (дата звернення: 17.05.2024)

31. About SMTP URL: https://uk.wikipedia.org/wiki/SMTP (дата звернення: 17.05.2024)

32. HTML, CSS, and JavaScript: Your Guide to Learning Fundamental Front End Languages URL: https://techbootcamps.utexas.edu/blog/html-css-javascript/ (дата звернення: 17.05.2024)

33. AJAX URL: https://en.wikipedia.org/wiki/Ajax_(programming) (дата звернення: 17.05.2024)

34. React: Documentation URL: https://react.dev/ (дата звернення: 17.05.2024)

35. Angular Tutorial URL: https://angular.io/tutorial (дата звернення: 17.05.2024)

36. Vue Documentation URL: https://vuejs.org/guide/introduction.html (дата звернення: 17.05.2024)

37. Virtual DOM and Internals URL: https://legacy.reactjs.org/docs/faq-internals.html (дата звернення: 17.05.2024)

38. Getting Started with Redux URL: https://redux.js.org/introduction/getting-started (дата звернення: 17.05.2024)

39. What is Bootstrap? [Електроний ресурс] — Режим доступу: https://www.w3schools.com/whatis/whatis_bootstrap.asp (дата звернення: 17.05.2024)

40. ANTD documentation URL: https://ant.design/components/overview/ (дата звернення: 17.05.2024)

41. NGINX documentation URL: https://nginx.org/en/docs/ (дата звернення: 17.05.2024)

42. What Is An SSL/TLS Certificate? URL: https://aws.amazon.com/what-is/ssl-certificate/ (дата звернення: 17.05.2024)

43. Advanced Encryption Standard URL: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard (дата звернення: 17.05.2024)

44. What is Smoke Testing? URL: https://www.techtarget.com/searchsoftwarequality/definition/smoke-testing (дата звернення: 17.05.2024)

45. What is CI/CD? URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd (дата звернення: 17.05.2024)

ДОДАТОК А

Лістинг програмного коду

```
var cloudinary = require('cloudinary');
const myError = require('../myError');
const config = require('../config');
const cloud = config["cloudinary"];
cloudinary.config({
  cloud_name: cloud["cloud_name"],
  api_key: cloud["api_key"],
  api_secret: cloud["api_secret"]
});
module.exports = {
  async addMedia(buffer, next) {
    return new Promise((resolve, reject) => {
      cloudinary.v2.uploader
        .upload_stream(
          { resource_type: 'raw' },
          (err, result) => {
            if (err) {
              return next(new myError(err.http_code, err.message));
            } else {
              resolve(result);
            }
          })
        .end(buffer);
    });
  }
  // async deleteMedia(public_id) {
  //    return await cloudinary.v2.uploader.destroy(public_id, { resource_type: 'raw' }, function (error,
result) { console.log(result, error); })
  // }}
var ObjectId = require('mongodb').ObjectID;
const Movie = require('../models/movies');
class MoviesRepository {
    constructor() { }
    async addMovies(movie_items) {
        const movieDoc = await new Movie(movie_items).save();
        return movieDoc;
    }
    async deleteMovies(movie_id) {
        const item = await Movie.deleteOne({ _id: ObjectId(movie_id) })
        return true;
    }
    async getMovies() {
        const fullItems = await Movie.find().populate("stars", ["fullname"]);
        return fullItems;
    }
    async getMoviesById(movie_id) {
        if (String(movie_id).length != 24) return null;
```

```
        const item = await Movie.findOne({ _id: ObjectId(movie_id) }).populate("stars", ["fullname"]);
        return item;
    }
    async getMoviesByAll(obj) {
        const item = await Movie.findOne({ title: obj.title, ReleaseYear: obj.ReleaseYear
}).populate("stars", ["fullname"]);
        return item;
    }};
module.exports = MoviesRepository;
const MediaRepository = require('./../repositories/mediaRepository');
const myError = require('../myError');
module.exports = {
    // async deleteMediaFile(url, next) {
    //     try {
    //         if (url !== 'ERROR Media.js 22') {
    //             const result = await MediaRepository.deleteMedia(url.split('/')[7]);
    //         }
    //     } catch (err) {
    //         if (err instanceof myError) return next(err);
    //         else return next(new myError(500, err.message));
    //     }
    // },
    async addMediaFile(fileUrl, req, next) {
        try {
            if (req.files.textFile) {
                const media_url = await MediaRepository.addMedia(req.files.textFile.data, next)
                return media_url.url
            } else {
                return 'ERROR Media.js 22'
            }
        } catch (err) {
            return next(new myError(500, err.message));
        }
    },
}
const fetch = require("node-fetch");
const MoviesRepository = require('../repositories/moviesRepository');
const StarsRepository = require('../repositories/starsRepository');
const MediaController = require('./media');
const myError = require('../myError');
const movieRepository = new MoviesRepository();
const starRepository = new StarsRepository();
var ObjectId = require('mongodb').ObjectID;
const page_size = 9;
module.exports = {
    async getMovies(req, res, next) {
        try {
            let page_str = req.query.page;
            let title_search = req.query.title;
            let search_selection = req.query.search_selection;
            let unsort = req.query.unsort;
            let sort = req.query.sort;
```

```
let reset = req.query.reset;
if (reset !== undefined) {
    page_str = undefined
    title_search = undefined
    search_selection = undefined
    unsort = undefined
    sort = undefined
}
let page;
if (page_str === undefined) page = 1;
else {
    page = Number(page_str);
    if (isNaN(page)) return next(new myError(400, 'page is not a number'));
    if (!Number.isInteger(page)) return next(new myError(400, 'page is not an integer'));
    if (page < 1) return next(new myError(400, 'invalid page value (page < 1)'));
}
let movies = await movieRepository.getMovies();
let selected_movie = "selected", selected_actor = ""
if (search_selection === "Actors") {
    selected_actor = "selected";
    selected_movie = ""
}
const movies_page = movies.slice(offset, offset + page_size);
const arr = [];
for (const movie of movies_page) {
    arr.push({
        id: movie.id,
        title: movie.title
    });
}
const paginator_pages = []
let title_query = '';
if (!(title_search === undefined)) title_query = '&title=' + title_search;
if (page === 1) paginator_pages.push({
    element_text: '<-'
})
else paginator_pages.push({
    element_page: page - 1,
    element_text: '<-',
    title_query: title_search
})
if (page > 5) {
    paginator_pages.push({
        element_page: 1,
        element_text: 1,
        title_query: title_search
    })
    if (page != 6) paginator_pages.push({
        element_text: '...'
    })
}
for (i = Math.max(page - 4, 1); i < page; i++) {
```

```
            paginator_pages.push({
                element_page: i,
                element_text: i,
                title_query: title_search
            })
        }
        paginator_pages.push({
            element_text: page
        })
        for (i = page + 1; i <= Math.min(page + 4, max_page); i++) {
            paginator_pages.push({
                element_page: i,
                element_text: i,
                title_query: title_search
            })
        }
        if (page < max_page - 4) {
            if (page != max_page - 5) paginator_pages.push({
                element_text: '...'
            })
            paginator_pages.push({
                element_page: max_page,
                element_text: max_page,
                title_query: title_search
            })
        }
        if (offset + page_size < size) paginator_pages.push({
            element_page: page + 1,
            element_text: '->',
            title_query: title_search
        })
        else paginator_pages.push({
            element_text: '->'
        })
        params = {
            head_title: 'MOVIE',
            movies_page: arr,
            movies_current: 'current',
            paginator_pages,
            title_query: title_search,
            display_unsort,
            display_sort,
            into_span_sort,
            into_span_select,
            sort_query,
            search_selection,
            display_select,
            selected_actor,
            selected_movie,
        }
        res.status(200).render('movies', params);
    } catch (err) {
```

```
            return next(new myError(500, err.message));
        }
    },
    async getMoviesFromImport(req, res, next) {
        try {
            const page_str = req.query.page;
            const length = req.params.length;
            const fullLength = req.params.full;
            let tags = ""
            if (fullLength === length) {
                tags = `<div id="snackbar">All ${fullLength} of the ${length} imported movies have been
added to the database.</div>
                <script>
                  var x = document.getElementById("snackbar");
                  x.className = "show";
                  setTimeout(function(){ x.className = x.className.replace("show", ""); }, 10000);

                </script>`
            } else {
                tags = `<div id="snackbar">Only ${length} of the ${fullLength} films were added to the
database. Due to errors in the file or this movie is already in the database.</div>
                <script>
                  var x = document.getElementById("snackbar");
                  x.className = "show";
                  setTimeout(function(){ x.className = x.className.replace("show", ""); }, 10000);

                </script>`
            }
            const sort = req.query.sort;
            let page;
            if (page_str === undefined) page = 1;
            else {
                page = Number(page_str);
                if (isNaN(page)) return next(new myError(400, 'page is not a number'));
                if (!Number.isInteger(page)) return next(new myError(400, 'page is not an integer'));
                if (page < 1) return next(new myError(400, 'invalid page value (page < 1)'));
            }
            let movies = await movieRepository.getMovies();
            const size = movies.length;
            const max_page = Math.ceil(size / page_size);
            const offset = page_size * (page - 1);
            if (offset === 0 && size === 0) {
                const paginator_pages = []
                paginator_pages.push({
                    element_text: '<-'
                })
                paginator_pages.push({
                    element_text: '->'
                })
                res.status(200).render('movies', {
                    head_title: 'MOVIES',
                    movies_page: null,
```

```
            movies_current: 'current',
            paginator_pages: paginator_pages
        });
        return;
    }
    if (offset >= size) return next(new myError(400, 'offset is bigger than movies number (page size is 8)'));
    let display_sort = `<input type="submit" name="sort" class="sortButton" id="hider" value="Sort!" />`,
        display_unsort = ""
    if (sort === "Sort!") {
        display_sort = ""
        display_unsort = `<input type="submit" id="hider1" name="unsort" class="unsortButton" value="Remove sorting" />`
        movies.sort((a, b) => a.title !== b.title ? a.title < b.title ? -1 : 1 : 0);
    }
    const movies_page = movies.slice(offset, offset + page_size);
    const arr = [];
    for (const movie of movies_page) {
        arr.push({
            id: movie.id,
            title: movie.title
        });
    }
    const paginator_pages = []
    if (page === 1) paginator_pages.push({
        element_text: '<-'
    })
    else paginator_pages.push({
        element_page: page - 1,
        element_text: '<-',
    })
    if (page > 5) {
        paginator_pages.push({
            element_page: 1,
            element_text: 1,
        })
        if (page != 6) paginator_pages.push({
            element_text: '...'
        })
    }
    for (i = Math.max(page - 4, 1); i < page; i++) {
        paginator_pages.push({
            element_page: i,
            element_text: i,
        })
    }
    paginator_pages.push({
        element_text: page
    })
    for (i = page + 1; i <= Math.min(page + 4, max_page); i++) {
        paginator_pages.push({
```

```
                element_page: i,
                element_text: i,
            })
        }
        if (page < max_page - 4) {
            if (page != max_page - 5) paginator_pages.push({
                element_text: '...'
            })
            paginator_pages.push({
                element_page: max_page,
                element_text: max_page,
            })
        }
        if (offset + page_size < size) paginator_pages.push({
            element_page: page + 1,
            element_text: '->',
        })
        else paginator_pages.push({
            element_text: '->'
        })
        params = {
            head_title: 'MOVIE',
            movies_page: arr,
            movies_current: 'current',
            paginator_pages,
            display_unsort,
            display_sort,
            tags
        }
        res.status(200).render('movies', params);
    } catch (err) {
        return next(new myError(500, err.message));
    }
},
async showMovieById(req, res, next) {
    try {
        const id_str = req.params.id
        const obj = await movieRepository.getMoviesById(id_str)
        if (obj === null)
            return next(new myError(404, 'movie not found'));
        else {
            const tags = `<div id="snackbar">The film "${obj.title}" has been successfully added to
the database.</div>
            <script>
              var x = document.getElementById("snackbar");
              x.className = "show";
              setTimeout(function(){ x.className = x.className.replace("show", ""); }, 10000);

            </script>`
            res.status(200).render('movie', {
                head_title: 'MOVIE',
                movies_current: '',
```

```
                movie: obj,
                tags
            });
        }
    } catch (err) {
        return next(new myError(500, err.message));
    }
},
async getMovieById(req, res, next) {
    try {
        const id_str = req.params.id
        const obj = await movieRepository.getMoviesById(id_str)
        if (obj === null)
            return next(new myError(404, 'movie not found'));
        else {
            res.status(200).render('movie', {
                head_title: 'MOVIE',
                movies_current: '',
                movie: obj
            });
        }
    } catch (err) {
        return next(new myError(500, err.message));
    }
},
async errorIntoForm(req, res, next) {
    try {
        const title = req.params.title
        const format = req.params.format
        const stars = req.params.stars
        const ReleaseYear = req.params.ReleaseYear
        console.log("title:" + title);
        console.log("format:" + format);
        console.log("stars:" + stars);
        console.log("ReleaseYear:" + ReleaseYear);
        // res.status(200).render('movie', {
        //     head_title: 'MOVIE',
        //     movies_current: '',
        //     movie: obj
        // });
    } catch (err) {
        return next(new myError(500, err.message));
    }
},
async createMovie(req, res, next) {
    try {
        let body = req.body
        const theSameMovie = await movieRepository.getMoviesByAll(body)
        body.title = body.title.trim()
        body.stars = body.stars.trim()
        if (body.title === "") {
            //res.status(303).redirect('/movies/' + ObjectId('60dd8c5ab1cdc2474c5c9660'))
```

```
            res.status(200).render('new_movie', {
                head_title: 'MOVIE',
                movies_current: '',
                title: body.title,
                stars: body.stars,
                ReleaseYear: body.ReleaseYear
            });
            //res.status(303).redirect('/err/' + body.title + '/' + body.format + '/' + body.stars + '/' + body.ReleaseYear)
            return;
            //return next(new myError(400, 'Bad Request. The "Title" field is empty!'));
        }
        if (body.stars === "") {
            return next(new myError(400, 'Bad Request. The "Stars" field is empty!'));
        }
        for (let char of body.stars) {
            console.log(char.length === 1 && !(char.match(/[a-za-я]/i) || char.match(/,/) || char.match(/-/) || char.match(/\s/)));
            if (char.length === 1 && !(char.match(/[a-za-я]/i) || char.match(/,/) || char.match(/-/) || char.match(/\s/))) {
                return next(new myError(400, 'Bad Request. Actor names cannot contain numbers or symbols other than " ", "-"!'));
            }
        }
        body.title = body.title[0].toUpperCase() + body.title.slice(1);
        let arrStars = body.stars.split(',');
        let checkTheSame = [];
        for (let star of arrStars) {
            star = star.trim()
            let check = true;
            for (let i = 0; i < checkTheSame.length; i++) {
                if (checkTheSame[i] === star) {
                    check = false;
                }
            }
            if (check && star !== "") {
                checkTheSame.push(star)
            }
        }
        arrStars = checkTheSame
        if (arrStars.length === 0) return next(new myError(400, 'Bad Request. You have entered incorrect star data!'));
        if (theSameMovie !== null) {
            let StarsLength = 0
            for (let star of theSameMovie.stars) {
                for (let i = 0; i < arrStars.length; i++) {
                    if (arrStars[i] === star.fullname) {
                        StarsLength++;
                    }
                }
            }
            if (StarsLength === theSameMovie.stars.length) {
```

```
                return next(new myError(400, 'Sorry! We were unable to add a "new" movie because it is already in the database.!'));
            }
        }
        checkTheSame = []
        body.stars = [];
        for (let word of arrStars) {
            if (word[0] === " ") {
                word = word.slice(1);
            }
            let item = await starRepository.getStarsByName(word);
            if (item == null) {
                const new_body = { fullname: word }
                item = await starRepository.addStars(new_body)
            }
            body.stars.push(item._id)
        }
        const movieObj = await movieRepository.addMovies(body)
        res.status(303).redirect('/movies/show/' + movieObj._id)
    } catch (err) {
        return next(new myError(500, err.message));
    }
},
async importMovie(req, res, next) {
    try {
        const body = req.body
        let fullLengthItems = 0;
        let Length = 0;
        if (!req.files.textFile.name.endsWith('.txt')) {
            return next(new myError(400, 'The file is specified incorrectly. File must have the extension ".txt"'));
        }
        body.fileUrl = await MediaController.addMediaFile(body.fileUrl, req, next)
        const response = await fetch(body.fileUrl);
        let movies = await response.text();
        movies = movies.split("\r\n")
        let obj_2_create = {
            title: undefined,
            ReleaseYear: undefined,
            format: undefined,
            stars: undefined,
        };
        for (let str_movie_attribute of movies) {
            let skipErr = false
            str_movie_attribute = str_movie_attribute.trim();
            let arrFullname = []
            if (str_movie_attribute.startsWith('Title: ')) {
                fullLengthItems++
                obj_2_create.title = str_movie_attribute.slice(7)
                obj_2_create.title = obj_2_create.title[0].toUpperCase() + obj_2_create.title.slice(1);
            }
            else if (str_movie_attribute.startsWith('Release Year: ')) {
```

```
            obj_2_create.ReleaseYear = str_movie_attribute.slice(14)
        }
        else if (str_movie_attribute.startsWith('Format: ')) {
            obj_2_create.format = str_movie_attribute.slice(8)
        }
        else if (str_movie_attribute.startsWith('Stars: ')) {
            str_movie_attribute = str_movie_attribute.slice(7)
            const stars_id = [];
            for (let char of str_movie_attribute) {
                if (char.length === 1 && !(char.match(/[a-za-я]/i) || char.match(/,/) || char.match(/-/) ||
char.match(/\s/))) {
                    skipErr = true
                }
            }
            if (skipErr) {
                obj_2_create = {};
                continue;
            }
            let arrStars = str_movie_attribute.split(',');
            let checkTheSameStar = [];
            for (let star of arrStars) {
                star = star.trim()
                let check = true;
                for (let i = 0; i < checkTheSameStar.length; i++) {
                    if (checkTheSameStar[i] === star) {
                        check = false;
                    }
                }
                if (check && star !== "") {
                    checkTheSameStar.push(star)
                }
            }
            arrStars = checkTheSameStar
            if (arrStars.length === 0) {
                obj_2_create = {};
                continue;
            }
            for (let word of arrStars) {
                if (word[0] == " ") {
                    word = word.slice(1);
                }
                arrFullname.push(word)
                let item = await starRepository.getStarsByName(word);
                if (item == null) {
                    const new_body = { fullname: word }
                    item = await starRepository.addStars(new_body)
                }
                stars_id.push(item._id)
            }
            obj_2_create.stars = stars_id
        } else {
            obj_2_create = {};
```

```
            }
            if (obj_2_create.title && obj_2_create.ReleaseYear && obj_2_create.format &&
obj_2_create.stars) {
                const theSameMovie = await movieRepository.getMoviesByAll(obj_2_create)
                let StarsLength = 0
                if (theSameMovie !== null) {
                  for (let star of theSameMovie.stars) {
                    for (let i = 0; i < arrFullname.length; i++) {
                      if (arrFullname[i] === star.fullname) {
                        StarsLength++;
                      }
                    }
                  }
                  if (StarsLength === theSameMovie.stars.length) continue;
                }
                const movieObj = await movieRepository.addMovies(obj_2_create)
                Length++
                obj_2_create = {};
              }
            }
            res.status(303).redirect('/movies/import/' + Length + '/' + fullLengthItems)
        } catch (err) {
          return next(new myError(500, err.message));
        }
    },
    async deleteMovieById(req, res, next) {
      try {
        const id_str = req.params.id
        const obj = await movieRepository.deleteMovies(id_str)
        if (obj === null) return next(new myError(404, 'movie not found'));
        else
          res.status(303).redirect('/movies')
      } catch (err) {
        return next(new myError(500, err.message));
      }
    },
};
const mstMovieRouter = require('express').Router();
const mstMovieController = require('../controllers/mstMovies')
const myError = require('./../myError');
const mongoose = require('mongoose');
const movieSchema = new mongoose.Schema({
    title: { type: String, required: true },
    ReleaseYear: { type: Number, required: true },
    format: { type: String, required: true },
    stars: [{ type: mongoose.Types.ObjectId, ref: 'Star', required: true }],
}, { collection: 'movies' });
module.exports = mongoose.model('Movie', movieSchema);
const mongoose = require('mongoose');
const starSchema = new mongoose.Schema({
    fullname: { type: String, required: false },
}, { collection: 'stars' });
```

```
module.exports = mongoose.model('Star', starSchema);
```