Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Проєктування та розробка серверної Backend – частини медіасховища ТНТУ ім. І. Пулюя з використанням фреймворку Django та технології GraphQL

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 «Інженерія програмного Забезпечення»
(шифр і назва спеціальності)

(підпис) Єсипов Л. С. (прізвище та ініціали)

Керівник (підпис) Мудрик І. Я. (прізвище та ініціали)

Нормоконтроль (підпис) (прізвище та ініціали)

Завідувач кафедри (підпис) (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2024

РЕФЕРАТ

Кваліфікаційна робота на здобуття освітнього ступеню «бакалавр» за спеціальністю 121 – Інженерія програмного забезпечення написана Єсиповим Леонідом Сергійовичем, студентом Тернопільського національного технічного університету імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії. Група СП-41, 2024 рік.

Тема: Проєктування та розробка серверної Backend – частини медіасховища ТНТУ ім. І. Пулюя з використанням фреймворку Django та технології GraphQL.

Система медіасховища складається з клієнтської сторони з використанням фреймворку «Nuxt 3» та двох серверних сторін: перша також на Nuxt 3, а друга – на Django. Дана кваліфікаційна робота бакалавра демонструє проєктування та розробку обох серверних сторін системи. Сховище повинне мати надійний захист, бути зручним у використанні, а також мати можливість швидкого та зручного масштабування. Фреймворк Django було обрано завдяки його надійності, стабільності та зручності розробки. Завдяки великій спільноті існує безліч різноманітних модулів, які допомагають значно прискорити розробку. Nuxt 3 було обрано завдяки підтримці SSR, легкій інтеграції з бекендом, великій кількості модулів та плагінів, простоті використання та гнучкості. При розробці системи були взяті до уваги основні переваги та недоліки конкурентів. В ході роботи було розроблено структуру бази даних, схему GraphQL запитів, реалізовано авторизацію за допомогою технології JWT та всю серверну логіку медіасховища.

Ключові слова: медіасховище, Django, Python, GraphQL, MySQL, Nuxt3, SSR, бази даних, JWT, TypeScript, вебсайт.

ANNOTATION

Qualification work for the bachelor's degree in specialty 121 – Software Engineering was written by Yesypov Leonid Serhiyovych, a student of Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering. Group SP-41, 2024.

Topic: Design and Development of the server Backend – part of the media storage of Ternopil Ivan Puluj National Technical University using the Django Framework and GraphQL Technology.

The media repository system consists of a client-side using the "Nuxt 3" framework and two server-side components: the first also built on Nuxt 3, and the second on Django. This bachelor's qualification work demonstrates the design and development of both server-side components of the system. The repository must be secure, user-friendly, and capable of quick and convenient scaling. Django was chosen due to its reliability, stability, and ease of development. With its large community, there are numerous modules available that significantly speed up the development process. Nuxt 3 was selected for its support of SSR, easy integration with the backend, a large number of modules and plugins, simplicity of use, and flexibility. During the development of the system, the main advantages and disadvantages of competitors were taken into account. The work involved developing the database structure, designing the GraphQL query schema, implementing authorization using JWT technology, and the entire server-side logic of the media repository.

Keywords: media repository, Django, Python, GraphQL, MySQL, Nuxt 3, SSR, databases, JWT, TypeScript, website.

ЗМІСТ

ВСТУП

У сучасному світі організація та управління даними стали ключовими аспектами нашого життя. Засоби збереження еволюціонували протягом століть, відповідаючи на виклики та потреби кожної епохи.

Спочатку це були пергамент і чорнило, за допомогою яких писали середньовічні ченці, створюючи рукописні манускрипти, проте їх створення та збереження вимагало величезних зусиль і часу. Потім настав час паперових книг, які принесли революцію в доступі до знань, хоча все ще вимагали фізичного місця для зберігання та були вразливі до зношування. З появою перших обчислювальних систем ми перейшли на новий рівень збереження даних — перфокарти. Вони символізували початок цифрової ери, хоча й залишалися громіздкими та обмеженими в обсязі інформації, яку могли зберігати. Наступним кроком стали дискети, які подарували користувачам зручний спосіб зберігати й переносити дані. Проте вони швидко застаріли, поступаючись місцем більш ємним та компактним USB накопичувачам, які стали справжнім проривом, надаючи можливість переносити величезні обсяги даних у кишені.

В даний момент домінуючим засобом збереження інформації є хмарні технології. Хмари дозволяють зберігати величезні масиви даних, доступних з будь-якого місця та пристрою. Вони не тільки спрощують доступ до інформації, але й надають новий рівень безпеки та надійності.

Медіасховища стали критично важливими для різних установ, особливо для освітніх закладів. Вони забезпечують централізоване зберігання, управління та доступ до великих обсягів інформації, включаючи текстові документи, зображення, відео, аудіофайли та інші цифрові ресурси. З огляду на зростаючу кількість таких даних, ефективне медіасховище є необхідним інструментом для підтримки освітнього процесу та наукових досліджень.

З кожним роком зростає обсяг навчальних матеріалів та дослідницьких даних, які потребують належного зберігання і управління. Для університетів

медіасховища не лише полегшують організацію цих даних, але й сприяють ефективному їх використанню. Наприклад, студенти та викладачі можуть легко отримувати доступ до навчальних матеріалів, наукових статей, відеолекцій, курсів та інших ресурсів. Це підвищує якість навчання та забезпечує доступ до важливої інформації у будь-який час і з будь-якого місця.

Впровадження новітніх технологій у діяльність університетів є важливим аспектом їхнього розвитку. Розробка та використання власного медіасховища дозволить навчальному закладу краще контролювати зберігання даних, забезпечувати їх безпеку, змінювати функціонал під свої потреби. Для ТНТУ ім. І. Пулюя створення власного медіасховища є кроком до зміцнення своєї інформаційної інфраструктури та підвищення ефективності освітнього процесу.

Наявність розвинутого та зручного медіасховища може стати додатковим стимулом для абітурієнтів обрати саме ТНТУ ім. І. Пулюя. Студенти активно користуються цифровими технологіями та цінують безкоштовний доступ до різних сервісів, які може надати їм університет. Це може бути важливим критерієм при виборі місця навчання, особливо для спеціальностей, пов'язаних з інформаційними технологіями та програмною інженерією.

У світі швидко змінюваних технологій, вимоги до медіасховищ постійно зростають. Важливими аспектами є безпека, надійність, масштабованість та зручність використання. В медіасховищі мусить бути захист даних від несанкціонованого доступу, підтримка великого обсягу даних без втрати продуктивності, а також воно має легко адаптуватися до потреб користувачів.

Розробка медіасховища для ТНТУ ім. І. Пулюя з використанням фреймворку Django та технології GraphQL спрямована на створення системи, яка відповідає цим вимогам. Django, як один з найпопулярніших фреймворків для веброзробки, забезпечує надійність та стабільність роботи, а GraphQL надає гнучкість у запитах та передачі даних між клієнтом і сервером. Використання цих технологій дозволить створити масштабоване та зручне у використанні медіасховище, яке може підтримувати постійне зростання обсягу даних та забезпечувати високий рівень безпеки.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд існуючих рішень

На сучасному ринку представлено безліч сервісів, які надають послуги для зберігання та управління цифровими даними. Вони орієнтовані на різні категорії користувачів, від окремих осіб до великих організацій, пропонуючи різноманітні можливості для різних потреб. У цьому розділі розглянемо два рішення:

- Google Drive
- Dropbox

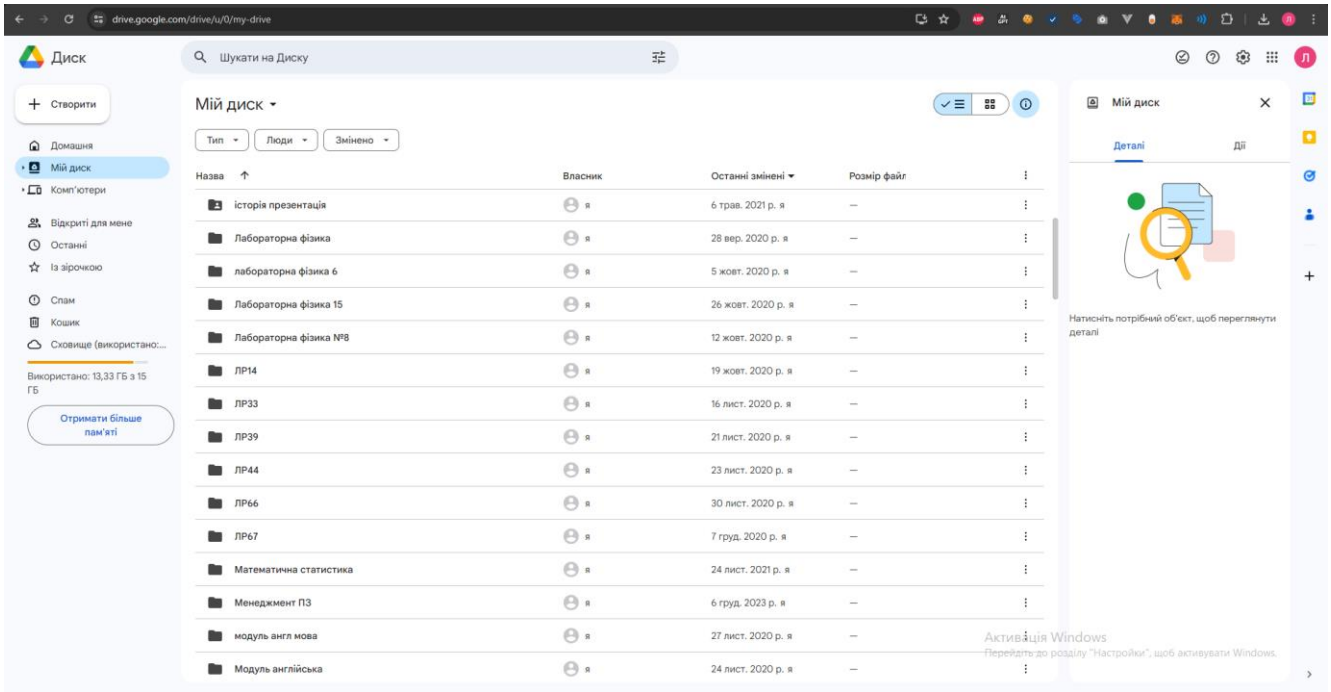


Рис. 1.1 – вигляд сховища Google Drive

Google Drive є одним із найпопулярніших хмарних сервісів для зберігання даних, особливо серед індивідуальних користувачів та малих бізнесів. Завдяки глибокій інтеграції з іншими продуктами Google, такими як Google Docs, Sheets та Slides, користувачі отримують безкоштовний доступ до створення документів, таблиць та презентацій. Це робить Google Drive ідеальним інструментом для

співпраці в реальному часі. Крім того, система налаштувань доступу дозволяє безпечно ділитися файлами, не турбуючись про несанкціонований доступ [1].

Окрім вебсайту, Google Drive пропонує програмне забезпечення для ПК, що дозволяє отримувати доступ до файлів навіть без підключення до інтернету. Для мобільних пристроїв є застосунок, який забезпечує оптимізований досвід використання сховища на смартфонах та планшетах. Алгоритми пошуку та фільтрації від Google відзначаються своєю ефективністю, дозволяючи знаходити файли не тільки за назвою, але й за вмістом. Користувачі можуть легко застосовувати різноманітні фільтри для швидкого доступу до потрібних даних.

Google Drive також пропонує історію змін файлів, що дозволяє користувачам протягом 30 днів відновлювати попередні версії документів. Це особливо корисно для роботи з важливими документами, де кожна зміна може мати значення. Завдяки цьому Google Drive стає потужним інструментом для керування даними та забезпечення безперебійного доступу до них.

Окрім сховища від Google, одним з домінуючих сховищ на ринку є сервіс Dropbox. Dropbox є одним з перших хмарних сервісів для зберігання даних, який приваблює користувачів завдяки простоті використання та надійності.

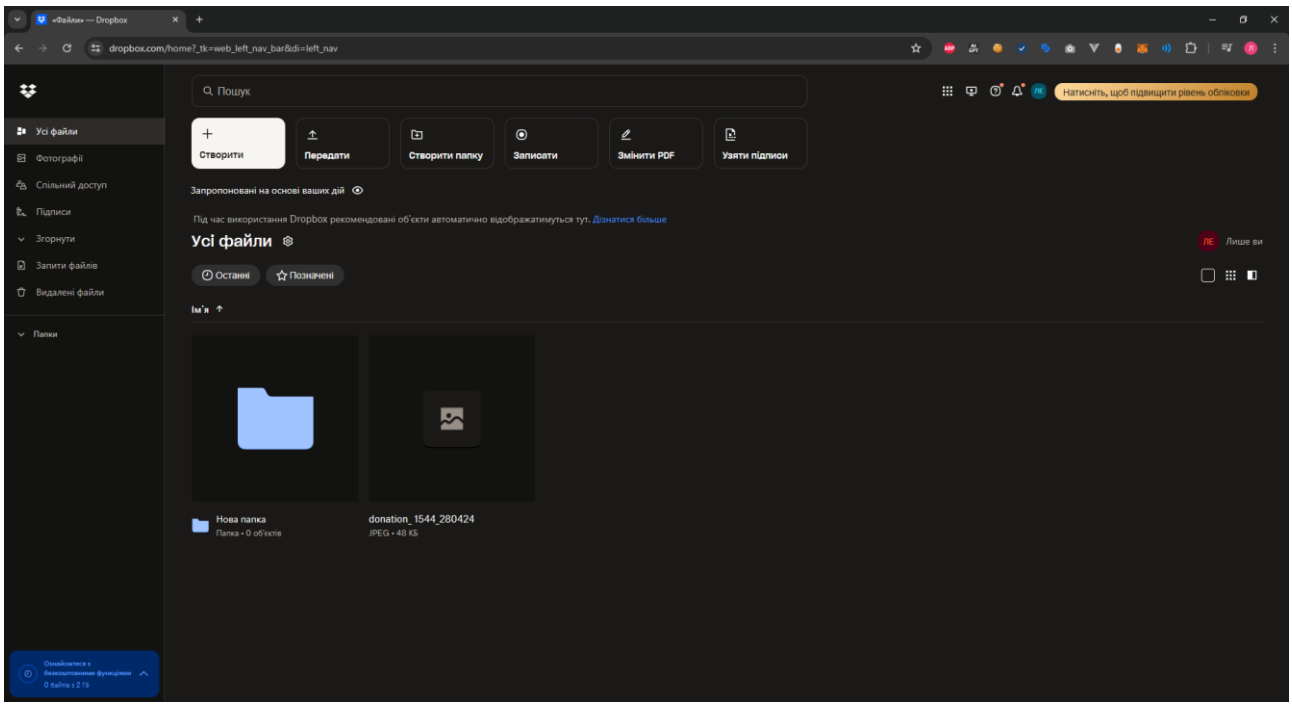


Рис 1.2 – Вигляд сховища Dropbox

Однією з ключових особливостей Dropbox є його орієнтація на простоту та ефективність. Користувачі можуть з легкістю завантажувати та зберігати файли будь-якого типу. Інтерфейс сервісу інтуїтивно зрозумілий, що робить його доступним для користувачів з різним рівнем технічної підготовки.

Dropbox активно підтримує командну роботу, надаючи потужні інструменти для співпраці. Використовуючи спільні папки, команди можуть обмінюватися файлами та працювати над ними спільно. Вбудовані функції коментування та приміток дозволяють обговорювати деталі безпосередньо в документі, що спрощує комунікацію та знижує потребу у використанні додаткових інструментів для обміну думками.

Dropbox також інтегрується з безліччю сторонніх сервісів та додатків, що значно розширює його функціональні можливості. Завдяки інтеграції з Microsoft Office, Google Workspace, Adobe та багатьма іншими платформами, користувачі можуть безпосередньо працювати з файлами, збереженими в Dropbox, без необхідності їх переміщення. Це сприяє зручності та ефективності роботи, дозволяючи зосередитися на завданнях, а не на технічних деталях.

Таким чином, можна зробити висновок, що Dropbox найбільше орієнтується на спільну роботу з файлами і є лідером в цьому напрямку.

1.2 Порівняльний аналіз конкурентів

Для створення якісного та конкурентоспроможного продукту потрібно враховувати всі переваги та недоліки існуючих рішень. У цьому розділі ми більш детально проаналізуємо, порівняємо між собою рішення конкурентів, висвітлимо їх переваги та недоліки. Це допоможе зрозуміти, які аспекти можуть бути враховані або покращені при розробці власного медіасховища.

Почнемо з найбільш відомого конкурента – Google Drive. Це платформа, створена компанією Google в 2012 році. В ньому можна виділити наступні переваги:

- Глибока інтеграція з Google Workspace – сховище має вбудовану інтеграцію з Google Docs, Sheets та Slides. Завдяки цьому користувачі, окрім завантаження файлів, можуть також створювати та редагувати документи безпосередньо в хмарі. Також безумовною перевагою є автоматичне збереження та синхронізація таких документів з іншими пристроями.
- Безкоштовний доступ і великий обсяг пам’яті. Користувачі отримують 15 ГБ безкоштовного сховища. Користувачі, яким потрібно мати більший обсяг пам’яті, мають доступ до різноманітних тарифів за доступними цінами.
- Потужні можливості пошуку та організації. Пошук можливий за вмістом файлів, а не тільки за назвою. Окрім цього, сховище використовує можливості штучного інтелекту для пошуку за вмістом фото. Також є можливість застосування фільтрів для швидкого доступу до потрібних даних.
- Історія змін і відновлення версій. Сховище надає можливість відновлення попередніх версій файлів протягом 30 днів. Це дуже корисно для користувачів, які працюють з важливими документами та часто вносять зміни.
- Мобільний застосунок та офлайн-доступ. Google Drive надає оптимізовані мобільні застосунки для смартфонів та планшетів. Для користувачів ПК існує програмне забезпечення, яке надає можливість роботи з файлами без підключення до інтернету.

Проте, як і в будь-якій системі, це сховище має і свої недоліки, а саме:

- Обмеження для великих файлів. Навіть з найдорожчим тарифом максимальний розмір завантаження файлів становить 750 ГБ на день. Хоч

це і може здатися великим числом, але для деяких користувачів цього все-таки може бути недостатньо.

- Складність управління правами доступу для великих команд. Хоч Google Drive і надає досить широкі можливості для керування доступом до файлів, сервіс не надає швидкого способу дати доступ до файлу певному статичному набору осіб. Наприклад, якщо викладач хоче дати доступ до файлу кожному студенту з групи, але не робити файл доступним по посиланню для всіх, йому доведеться окремо вказувати електронну пошту кожного студента.

Отже, можна зробити висновок, що Google Drive – зручний інструмент, який чудово підходить для особистого використання, або використання спільно з невеликою кількістю людей. Проте, для використання великою командою, або університетом, його можливостей недостатньо.

Далі варто розглянути Dropbox, його переваги:

- Простота та зручність використання. Інтерфейс цього сховища дозволяє легко завантажувати та організовувати файли, а його простота налаштувань і мінімалістичний дизайн дозволяють вільно користуватися сервісом користувачам з будь-яким рівнем технічної підготовки.
- Вбудована підтримка інтеграції з Microsoft Office, за допомогою якої можна працювати з файлами через веб інтерфейс. Також можна розширити можливості за рахунок інтеграцій зі сторонніми сервісами.
- Надійність та стабільність. Оскільки Dropbox є одним із перших хмарних сховищ, він зарекомендував себе як надійна платформа, завдяки чому має високий рівень довіри від користувачів.

Як і Google Drive, Dropbox також має свої недоліки:

- Безкоштовний план надає лише 2ГБ пам’яті, що значно менше, ніж у конкурентів.
- Пошук файлів в Dropbox не настільки розвинений, як у Google Drive, і не підтримує пошук за вмістом файлів.

Dropbox залишається одним із провідних хмарних сховищ завдяки своїй простоті, надійності та глибокій інтеграції з популярними сервісами. Однак, обмежений обсяг безкоштовного сховища та менш розвинені можливості пошуку можуть стати вирішальними факторами при виборі платформи для деяких користувачів.

1.3 Обґрунтування вибору технологій

Вибір технологій, які будуть використовуватись при розробці сайту, впливає на швидкість, якість розробки, функціональність, надійність та масштабованість майбутньої системи. В цьому розділі буде детальне пояснення причини вибору фреймворків та технологій, які були використані для створення проєкту.

Оскільки для серверної частини буде використовуватися мова програмування Python, розглянемо найбільш популярні фреймворки, які забезпечують надійність та гнучкість розробки: Flask, Tornado, Django, FastAPI та CherryPy.

Кожен із цих фреймворків має свої переваги та недоліки, які слід ретельно проаналізувати.

- Flask відомий своєю простотою та легкістю у налаштуванні, що робить його ідеальним для невеликих проєктів. Однак, при масштабуванні можуть виникати труднощі, оскільки він надає мало вбудованих функцій, а зростання складності проєкту сильно збільшує кількість коду, який потрібно написати.
- Tornado забезпечує високу продуктивність і асинхронну обробку запитів, що підходить для додатків з високими вимогами до обробки даних в реальному часі. Проте, його складність є також значним мінусом в контексті медіасховища для університету, оскільки з високою ймовірністю, підтримкою проєкту з року в рік будуть займатися різні студенти, і складність фреймворку може стати для них значним викликом,

що в кінцевому результаті може негативно вплинути на якість майбутніх оновлень.

- Django є потужним фреймворком, що пропонує "batteries included", тобто комплексний набір вбудованих функцій та інструментів. Це зменшує необхідність інтеграції сторонніх компонентів та прискорює процес розробки. Django особливо добре підходить для масштабних веб-додатків і включає потужну систему адміністрування, що полегшує управління контентом. Також величезна підтримка спільноти, форуми та якісна документація позитивно вплинуть на майбутню підтримку проєкту в контексті того, що нею можуть займатися різні люди з різною кількістю досвіду.
- FastAPI вирізняється сучасним підходом до розробки та підтримкою асинхронних запитів. Він ідеально підходить для створення API з високою продуктивністю. Однак, його порівняно молодий вік може означати обмежену екосистему та недостатню підтримку для великих проєктів. Також варто зазначити, що для повного розкриття FastAPI (використання асинхронних запитів до бази даних) в якості ORM потрібно використовувати SQLAlchemy, яка не є досконалою та має свої обмеження, які негативно вплинуть на швидкість та якість розробки.
- CherryPy пропонує простоту і стабільність. Він чудово підходить для проєктів, де потрібно мінімум складнощів, проте він менш популярний, може не забезпечити всіх сучасних вимог розробки і має не таку потужну підтримку спільноти, як в розглянутих аналогів.

Враховуючи вимоги до функціональності, масштабованості та швидкості розробки медіасховища, найкращим вибором буде Django.

Рис 1.3.1 – логотип фреймворку Django [2]

Django є чудовим інструментом для розробки вебсайтів та серверних додатків. Це високорівневий fullstack фреймворк, побудований на мові програмування Python. Він є одним із найбільш використовуваних і популярних фреймворків у світі, відомий своєю простотою, гнучкістю та потужністю. Обрання Django для розробки серверної частини нашого проєкту обґрунтовується кількома ключовими аспектами, які детально розглянуті нижче.

- Швидкість розробки та продуктивність.

  Однією з головних переваг Django є можливість швидко розгорнути проєкт завдяки його вбудованим можливостям. В ньому є вбудована система маршрутизації, завдяки якій можна визначати URL-маршрути, що будуть використовуватися. В випадку цього проєкту на стороні Django потрібен маршрут тільки під медіа-файли та GraphQL запитів, оскільки решта URL маршрутів будуть створені на стороні та оброблені клієнтською частиною. Також фреймворк «з коробки» надає автоматичну генерацію адміністративного інтерфейсу для управління даними на основі визначених моделей, що значно прискорює початковий етап розробки, і в більшості випадків цей інтерфейс достатньо хороший та зручний для використання протягом всіх його життєвих циклів.
- Архітектурна модель MVT.

Django базується на архітектурному шаблоні MVT (Model-View-Template), який забезпечує чітке розділення логіки, даних та відображення. Це допомагає підтримувати чистоту коду та спрощує його подальшу підтримку та розширення.

- Model в цій архітектурній моделі відповідає за структуру даних і взаємодію з базою даних. Django надає власний вбудований ORM, що дозволяє взаємодіяти з базою даних за допомогою об'єктно-орієнтованого підходу, не прописуючи SQL-запити вручну. Це значно спрощує роботу з даними та підвищує безпеку завдяки вбудованому захисті від SQL-ін'єкцій.
- View – це компонент, який виконує обробку HTTP-запитів, взаємодіє з моделями та повертає відповідь на запити користувачів.
- Template відповідає за відображення даних користувачу. Django надає вбудований механізм шаблонів, завдяки якому можна створювати динамічні сторінки. Однак, в цьому проекті, ця частина замінена на Nuxt3, що обговорюється окремо.

- Безпека та надійність.

  Django приділяє особливу увагу безпеці, надаючи вбудований захист від CSRF атак та SQL-ін'єкцій. Окрім цього, Django має вбудований механізм хешування паролів, завдяки якому зловмисник, навіть отримавши доступ до бази даних, не зможе дізнатися паролі користувачів.

- Підтримка спільноти.

  За час свого існування, Django набув великої та активної спільноти, яка постійно підтримує розвиток фреймворку, створюючи безліч сторонніх модулів, які розширюють функціональність проекту без необхідності винаходити велосипед форумів, блогів та курсів, що надають підтримку та навчальні матеріали для розробників будь-якого рівня.

- Практичний досвід та перевірена ефективність

Django використовується в багатьох великих та відомих комерційних платформах, що доводить його ефективність та надійність для розробки сучасних вебсайтів.

На основі наведених аргументів, можна зробити висновок, що Django є потужним, безпечним та гнучким інструментом для розробки серверної частини нашого проекту, що повністю відповідає вимогам до системи і забезпечує всі необхідні умови для успішної реалізації проекту.

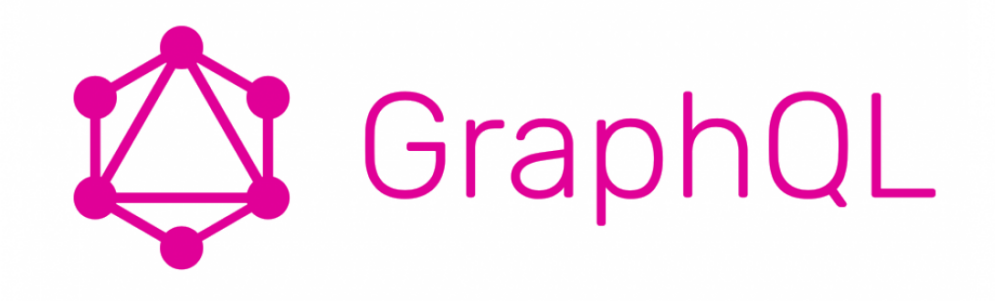


Рис 1.3.2 – логотип технології GraphQL

GraphQL — це мова запитів для API, яка забезпечує клієнтам можливість отримувати саме ті дані, які їм потрібні, без зайвого навантаження на базу даних. Вона була розроблена компанією Facebook у 2012 році, а вже в 2015 році стала відкритим стандартом. З тих пір GraphQL завоював велику популярність завдяки своєму унікальному підходу до взаємодії з базою даних, а також став чудовою альтернативою RESTful API.

Основні причини вибору GraphQL для даного проєкту:

- Зменшення обсягу даних, що передаються: У RESTful API кожен кінцевий пункт (endpoint) зазвичай повертає фіксований набір даних, що призводить до передачі зайвої інформації. Наприклад, якщо стороні клієнта в одному випадку потрібно отримувати тільки назву файлу і посилання на нього, а в іншому випадку потрібно отримувати всю наявну інформацію про файл, на RESTful API це можливо реалізувати тільки створюючи два різних кінцевих пункти на одну і ту ж модель, в той час як за допомогою GraphQL можна створити тільки один Query (в GraphQL

Query – це запит, який відповідає за читання даних), і на стороні клієнта вказувати в запиті тільки ті поля, які необхідно отримати.

- Зменшення кількості запитів: У RESTful API для отримання даних з різних ресурсів часто потрібно робити кілька запитів. Наприклад, щоб отримати інформацію про користувача і всі його файли та папки, можуть знадобитися окремі запити до різних кінцевих точок. У GraphQL можна отримати всю необхідну інформацію в одному запиті, що знижує навантаження на мережу і зменшує час очікування відповіді.
- Автоматична генерація документації та типізації: Однією з суттєвих переваг GraphQL є його здатність до автоматичної генерації документації, що значно пришвидшує розробку. Завдяки цьому навіть якщо розробкою Backend-частини і Frontend-частини вебсайту займаються різні розробники, не буде виникати зайвих запитань та непорозумінь. Також завдяки тому, що документація генерується автоматично (включно з тим, що в документації вказані типи всіх даних, які повертаються), існує багато інструментів, які на основі цього автоматично генерують типи та інтерфейси для TypeScript, що значно прискорює роботу Frontend-розробника.
- Потужні інструменти для розробників та інтеграція з популярними фреймворками: GraphQL надає розробникам потужні інструменти для тестування та дебагу, такі як GraphiQL, що значно полегшує процес розробки. Також практично для кожного популярного як Frontend, так і Backend фреймворку існують бібліотеки, які інтегрують в них інструменти для роботи з GraphQL. Для прикладу, для фреймворку Django існує модуль «Graphene-Django», а для Nuxt3 – «Nuxt GraphQL Middleware».

Вибір GraphQL для цього проекту забезпечує високу гнучкість та ефективність у роботі з даними. Його здатність надавати тільки потрібні дані за запитом, об'єднувати дані з різних моделей в один запит, автоматично генерувати

документацію та типи робить його ідеальним вибором для сучасних вебдодатків, що вимагають динамічної та ефективної обробки інформації.

Рис 1.3.3 – логотип бази даних MySQL

Серед великого вибору різних баз даних, для мене MySQL став очевидним вибором — це реляційна система керування базами даних (РСКБД), яка є однією з найпопулярніших і найширше використовуваних у світі. Вибір MySQL для проєкту обумовлений низкою вагомих причин:

- Продуктивність: MySQL здатний забезпечувати високу продуктивність при обробці запитів будь-якої складності. Система використовує оптимізовані алгоритми індексації та кешування, що пришвидшує виконання запитів і зменшує навантаження на ПК/сервер.
- Надійність: Завдяки своїй стабільності та підтримці транзакцій, MySQL забезпечує надійне зберігання та обробку даних. При використанні транзакцій існує можливість відмінити операцію у разі необхідності, і всі застосовані зміни буде скасовано. Також MySQL підтримує різні методи резервного копіювання, включаючи створення дампів, що полегшує відновлення даних у разі збою.
- Широка підтримка та сумісність: MySQL підтримується багатьма мовами програмування та інструментами, що робить його універсальним вибором для різних проєктів. Більшість сучасних ОС підтримують MySQL, а також

майже кожна мова програмування та фреймворк мають вбудовану підтримку роботи з цією системою.

- Простота використання та налаштування. MySQL дуже простий в установці та налаштуванні, що дозволяє швидко почати роботу з ним. Завдяки цьому розгортання проєкту на сервері потребує мінімум часу та зусиль. Користувачі Windows також можуть використовувати утиліту «MySQL Workbench», яка надає зручний графічний інтерфейс, завдяки якому можна створювати та редагувати схеми і таблиці без необхідності прописувати команди вручну.

MySQL — це надійна, продуктивна та безпечна РСКБД, яка є оптимальним вибором для розробки серверної частини нашого проекту. Всі вищезазначені переваги роблять MySQL ідеальним рішенням для зберігання та управління даними.

Для розробки якісної клієнтської частини необхідно було обрати потужний Frontend-фреймворк, в якого є хороша підтримка від спільноти, який зарекомендував себе як надійна та ефективна платформа, яка надає потужні інструменти для розробки сучасних вебдодатків.

Рис 1.3.4 – логотип фреймворку Nuxt3

Nuxt 3 — це сучасний фреймворк на основі Vue3, випущений в 2022 році і призначений для створення продуктивних вебдодатків. Фреймворк значно прискорює роботу завдяки своїй модульній архітектурі. Однією з ключових особливостей є підтримка різних стратегій рендерингу (візуалізації сторінки), а саме:

- SSR – рендеринг на стороні сервера.

- CSR – рендеринг на стороні клієнта.
- SSG – статична генерація сторінок.

Це дозволяє оптимально адаптувати додаток під вимоги будь-якого проєкту. Nuxt3 забезпечує реактивність через Vue3 Composition API, що підвищує гнучкість та ефективність роботи з компонентами. Загалом, Nuxt 3 спрощує розробку та оптимізацію продуктивності вебдодатків, забезпечуючи масштабованість і високу швидкість завантаження. Хоча Nuxt 3 початково задумувався як Frontend-фреймворк, з моменту його випуску розробники значно розширили його можливості. Вони активно працюють над тим, щоб перетворити Nuxt 3 на повноцінний Full Stack фреймворк, здатний повноцінно охоплювати як клієнтську, так і серверну частини додатків.

Основними перевагами цього фреймворку є:

- Продуктивність і оптимізація. В більшості тестів Nuxt3 показує себе одним із найшвидших сучасних Frontend-фреймворків. Це досягається за рахунок модульного дерева залежностей: фреймворк завантажує та компілює тільки ті модулі, які потрібні, що прискорює час завантаження та зменшує розмір фінального пакета. Окрім продуктивності, він є також одним із найшвидших в плані розробки. Це досягається завдяки використанню Vite як dev-сервера. Цей інструмент забезпечує надзвичайно швидку збірку, а також підтримку HMR (Hot Module Reload). Зміни в коді відображаються в браузері в реальному часі без необхідності перезавантажувати проєкт.
- Велика кількість готових модулів: Фреймворк пропонує безліч модулів та плагінів, що спрощують розробку та розширення функціональності без необхідності власноруч прописувати функціонал, який вже був створений кимось іншим.
- Підтримка TypeScript: Nuxt3 Має повну інтеграцію з TypeScript, що забезпечує зручну типізацію, безпеку коду та уникнення більшості «багів». Оскільки медіасховище на стороні бекенду використовує

GraphQL, як і зазначалося раніше, завдяки його автоматичній генерації документації, існують інструменти, які автоматично згенерують всі типи та інтерфейси на TypeScript, що значно прискорює розробку.

- Middleware (Проміжне програмне забезпечення): Однією з потужних функцій Nuxt 3 є підтримка middleware, що дозволяє керувати логікою обробки запитів і відповідей між клієнтом і сервером на різних етапах маршрутизації та рендерингу сторінок. Для медіасховища, де важливо контролювати доступ до ресурсів, захищати конфіденційні дані і оптимізувати процес завантаження, middleware надає значні переваги при реалізації автентифікації, авторизації та доступу до захищених сторінок.

Отже, Nuxt 3 – потужний фреймворк, який чудово підходить для розробки клієнтської частини медіасховища. Завдяки Nuxt 3 проєкт зможе забезпечити швидкий доступ до файлів, ефективне управління даними і високий рівень зручності для користувачів.

Для реалізації автентифікації я вирішив використати технологію JWT (JSON Web Token), яка ідеально підходить для забезпечення безпеки в веб-додатках. Для розробки проєкту, де на клієнтській стороні використовується Nuxt 3, а на серверній – Django та GraphQL, вибір JWT для аутентифікації має кілька суттєвих переваг, що пояснюють його використання замість стандартної аутентифікації Django.

- Безпека: Безпека є критично важливим аспектом для будь-якого медіасховища, і JWT надає кілька механізмів для забезпечення надійної аутентифікації
    - Цифровий підпис: JWT токени «підписуються» за допомогою алгоритмів шифрування, таких як HMAC або RSA, що забезпечує захист від підробки. Це гарантує, що токен не був змінений після його створення.
    - Вбудована інформація: Завдяки JWT, можна вбудовувати інформацію про користувача (наприклад, його id) безпосередньо в токен, що спрощує розпізнавання користувача на стороні сервера.

- Масштабованість: Використання автентифікації по JWT токену надає широкі можливості для реалізації додаткових платформ (наприклад, мобільного застосунку, закритого API, тощо), без необхідності змінювати серверну частину.
- Широка підтримка: JWT підтримується багатьма мовами та фреймворками, включно з Django та модулем «Graphene-Django», що робить використання JWT токенів ідеальним рішенням для цього проєкту.

1.4 Формування вимог до системи

Формування вимог до системи медіасховища є одним з ключових етапів в розробці проєкту, оскільки саме від цього залежить, наскільки ефективно та надійно система буде виконувати свої функції. Вимоги формуються на основі аналізу бізнес-процесів, потреб користувачів та технічних обмежень. Цей розділ охоплює функціональні та нефункціональні вимоги, що визначають основні характеристики системи та забезпечують її надійність, масштабованість, безпеку та зручність у використанні.

Функціональні вимоги:

- Реєстрація за ПІБ, електронною поштою та паролем.
- Авторизація за електронною поштою та паролем.
- Вихід з аккаунту.
- Можливість відкрити папку.
- Можливість попереднього перегляду файлу.
- Можливість завантажити файл з сервера, або на сервер.
- Можливість створення/перейменування папки.
- Можливість перейменування файлу.
- Можливість перемістити папку/файл в іншу папку.

- Можливість зміни прав доступу до файлу або папки (ввімкнути/ вимкнути доступ за посиланням).
- Пошук папок або файлів за назвою/датою завантаження.
- Можливість змінити ПІБ в профілі.
- Можливість відкрити адміністративну панель для користувача з правами адміністратора.
- Можливість перегляду списку користувачів та їх файлів для користувачів з відповідними правами доступу.
- Можливість редагування прав інших адміністраторів для користувача з правами адміністратора.
- Видалення файлів інших користувачів для користувача з правами адміністратора
- Блокування користувачів для користувача з правами адміністратора

Нефункціональні вимоги:

- Система повинна бути надійною, доступною та безпечною
- Медіасховище повинне бути зручним у використанні
- Завантаження та обробка медіафайлів має відбуватися швидко
- Система повинна масштабуватися для обслуговування великої кількості даних
- Система повинна бути розширюваною для додавання нових функцій та можливостей.

2 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ МЕДІАСХОВИЩА

Серверна частина будь-якого веб-додатку є фундаментом, на якому тримається вся система. У випадку з медіасховищем, яке надає користувачам можливість зберігати, обмінюватися та керувати великими файлами, особливо важливо якісно спроєктувати та реалізувати серверну архітектуру. Цей розділ присвячений детальному аналізу та опису процесу розробки серверної частини медіасховища, включаючи проектування бази даних, моделей, реалізацію GraphQL-запитів та мутацій, а також інтеграцію з клієнтською стороною.

Систематичний підхід до проєктування серверної частини дозволяє створити надійну та масштабовану систему, здатну ефективно обробляти великі обсяги даних та забезпечувати високу продуктивність. Неправильно спроєктована архітектура може призвести до проблем з масштабованістю, зниження продуктивності або навіть до краху системи під навантаженням. Тому важливо не лише забезпечити правильний вибір технологій, але й продумати зв'язки між компонентами системи, алгоритми обробки даних та методи взаємодії між клієнтом і сервером.

У цьому розділі ми розглянемо ключові аспекти проектування серверної частини, починаючи з структури бази даних та створення моделей, які будуть відображати ці дані. Ми опишемо, як були визначені GraphQL-запити та мутації, які забезпечують ефективний доступ до даних та маніпуляцію ними. Також розглянемо, як відбувається взаємодія між клієнтською та серверною сторонами, з акцентом на безпеку та ефективність передачі даних. Цей підхід дозволяє забезпечити стабільність і надійність роботи додатку, а також створити основу для його подальшого розвитку та масштабування.

Важливим елементом є також забезпечення захисту даних та конфіденційної інформації користувачів. Це включає в себе реалізацію механізмів аутентифікації та авторизації. У цьому контексті, вибір і впровадження JSON Web Token (JWT) для аутентифікації, як описано в попередньому розділі, відіграє ключову роль.

2.1 Визначення архітектури системи

Проєкт медіасховища використовує гібридну архітектуру, що поєднує елементи багаторівневої та мікросервісної архітектур. Архітектуру проєкту не можна чітко віднести до мікросервісної, оскільки всі її компоненти не є повністю незалежними: клієнтська частина не може працювати без серверної, а серверна частина без клієнтської не може повністю відтворити мету проєкта. До багаторівневої архітектури її теж не можна віднести, оскільки клієнтська і серверна частини розгортаються окремо, і якби окрім клієнтської частини існував також мобільний застосунок, вони б не були взаємопов’язаними.

Гібридна архітектура дозволяє ефективно розподілити обов'язки між компонентами системи, забезпечити їх незалежність та масштабованість. Основні елементи архітектури включають використання Django та GraphQL на серверній стороні, MySQL в якості бази даних, та Nuxt 3 для фронтенду з підтримкою SSR.

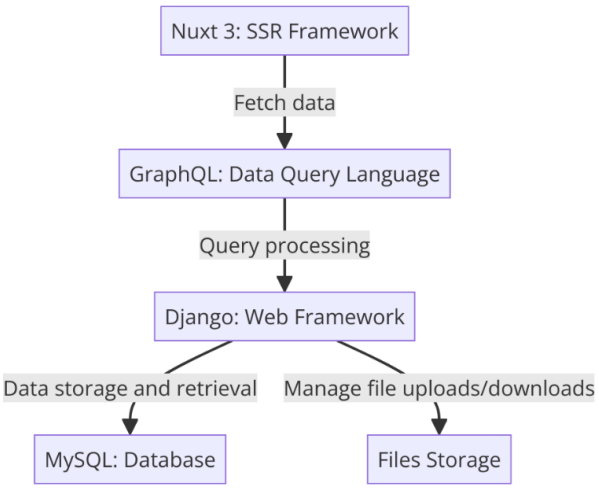


Рис. 2.1 – схема гібридної архітектури

Гібридна архітектура поєднує найкращі риси багаторівневої та мікросервісної архітектур:

- Багаторівнева архітектура дозволяє розділити систему на чітко визначені рівні, кожен з яких виконує певну функцію. Це спрощує

підтримку та розвиток системи, забезпечуючи ясність і структурованість.

- Мікросервісна архітектура дозволяє виділити окремі функціональні блоки, які можуть працювати незалежно один від одного. Це підвищує гнучкість і масштабованість системи, дозволяючи легко додавати нові функції або модулі.

Архітектура складається з таких основних компонентів:

- Клієнтська частина (Frontend): Nuxt 3 використовується для створення реактивного інтерфейсу користувача з підтримкою серверного рендерингу (SSR). Це забезпечує швидкий відгук додатка та покращує продуктивність завдяки попередньому рендерингу сторінок на сервері перед їх відправленням клієнту. Такий підхід особливо корисний для медіасховища, де користувачі можуть працювати з великими файлами і очікують швидкої реакції системи.
- Серверна частина (Backend):Django служить основним фреймворком для обробки серверної логіки та управління даними. Використання GraphQL забезпечує ефективну взаємодію з клієнтською частиною, дозволяючи їй запитувати тільки необхідні дані. Це знижує кількість запитів до сервера і підвищує продуктивність. Серверна частина також обробляє аутентифікацію та авторизацію користувачів, забезпечуючи безпеку та контроль доступу.
- База даних (Database): MySQL використовується для зберігання даних, забезпечуючи надійне та швидке збереження великих обсягів інформації. Це особливо важливо для файлообмінника, де обробка і зберігання великих файлів є критичною функцією.

Гібридна архітектура цього проєкту має такі принципи:

- Поділ обов'язків (Separation of Concerns): Гібридна архітектура забезпечує чіткий поділ обов'язків між різними компонентами. Кожен рівень системи (клієнтський, серверний, база даних) відповідає за певні

аспекти роботи додатку, що спрощує його підтримку і розвиток. Мікросервісний підхід дозволяє виділити окремі функціональні частини, які можуть працювати автономно, що підвищує гнучкість системи.

- Масштабованість і гнучкість: Комбінація багаторівневої та мікросервісної архітектур дозволяє легко масштабувати серверну та клієнтську частини незалежно одна від одної. Наприклад, серверна може бути розширена для інтеграції з новими мікросервісами, такими як мобільний застосунок.
- Безпека: Архітектура включає в себе надійні механізми безпеки, такі як JWT для аутентифікації та авторизації, що забезпечують захист даних користувачів і їхню конфіденційність.
- Ефективність обробки даних: Використання GraphQL дозволяє знизити навантаження на сервер, забезпечуючи клієнтській частині можливість запитувати тільки ті дані, які їй потрібні. Це підвищує швидкість відповіді та знижує затримки, що особливо важливо для інтерактивних веб-додатків, таких як файлообмінник.

Клієнтська частина на Nuxt 3 отримує дані через запити до GraphQL, який обробляється серверною частиною на Django. GraphQL дозволяє гнучко визначати і отримувати саме ті дані, які потрібні клієнту, що знижує кількість запитів і прискорює обробку.

Серверна частина на Django обробляє запити на завантаження, зберігання та отримання файлів, забезпечуючи їхню безпеку і доступність. Обробка великих файлів і оптимізація їхнього зберігання є ключовими аспектами функціональності медіасховища.

Завдяки гібридній архітектурі, система готова до майбутніх оновлень. Наприклад, можна легко додати нові сервіси або інтеграції, такі як мобільний застосунок, без значних змін в існуючій інфраструктурі.

2.2 Визначення варіантів використання

У процесі розробки будь-якого програмного забезпечення невід’ємною частиною є діаграми варіантів використання. Вони необхідні для визначення функціональних вимог та забезпечення чіткого розуміння взаємодії між користувачами та системою. Давайте розглянемо, що таке діаграми варіантів використання, їх основні елементи, мету та переваги, а також застосування в даному проєкті.

Діаграми варіантів використання представляють собою графічне зображення системи, що описує взаємодію між користувачами (акторами) і функціональністю системи (варіантами використання). Основними компонентами діаграми є:

- Актори: Зовнішні об'єкти або ролі, які взаємодіють із системою. Це можуть бути користувачі, інші системи або пристрої.
- Варіанти використання: Описують функції або послуги, які надає система акторам. Це конкретні дії або процеси, що виконуються системою на запит актора.
- Зв'язки: Лінії, що з'єднують акторів з варіантами використання, показуючи, які дії вони можуть виконувати.

Діаграми варіантів використання широко використовуються при проєктуванні програмного забезпечення з низки причин:

- Визначення вимог: Вони допомагають визначити функціональні вимоги до системи, виявляючи всі необхідні можливості, які повинна надавати система кінцевому користувачу.
- Комунікація з користувачами: Діаграми є зрозумілим і доступним способом обговорення функціональності системи з користувачами та зацікавленими сторонами, що дозволяє уточнити вимоги та очікування.

- Документування системи: Вони служать документацією, яка показує, які можливості повинна надавати система, що корисно для розробників та тестувальників.
- Планування та оцінка: Діаграми допомагають у плануванні розробки, оцінюванні обсягу робіт і визначенні пріоритетів функціональності.

Рис. 2.2 – Діаграма варіантів використання для медіасховища

На діаграмі вище зображено, як кожен з акторів взаємодіє з системою медіасховища:

- Користувач: Має доступ до таких функцій, як реєстрація, авторизація, завантаження і вивантаження файлів, керування папками, пошук та зміна особистих даних.

- Адміністратор: Окрім того, що його розширює актор «Користувач», має додаткові можливості, включаючи моніторинг активності користувачів, керування правами доступу, редагування профілів користувачів та керування системними параметрами.

Використання діаграм варіантів використання в проєкті приносить кілька важливих переваг:

- Чітке розуміння функціональності: Діаграми надають ясний і вичерпний огляд усіх функцій, які повинна реалізовувати система, допомагаючи легко зрозуміти, як працює система.
- Планування розробки: Завдяки чіткому визначенню всіх варіантів використання, можна більш ефективно планувати розробку, визначати пріоритети та оцінювати обсяг робіт.
- Підвищення якості: Опис функціональних вимог через варіанти використання дозволяє уникнути непорозумінь та покращити якість кінцевого продукту шляхом чіткого визначення очікуваної поведінки системи.

Таким чином, діаграми варіантів використання є важливим інструментом для успішної реалізації проектів, забезпечуючи чітке та ефективне визначення та документування функціональних вимог, що сприяє створенню якісного програмного забезпечення, яке відповідає очікуванням користувачів.

2.3 Проєктування серверної частини

Після розгляду варіантів використання системи ми можемо приступити до проєктування серверної частини, зокрема структури бази даних та методів, які будуть використовуватися для взаємодії з користувачами та файлами. Серверна частина проекту реалізована з використанням Django та GraphQL, що забезпечує

високу продуктивність і гнучкість у роботі з даними. Варто уточнити, що моделі є прямим відображенням структури в базі даних, тобто Django завдяки потужностям вбудованої ORM автоматично генерує необхідні поля в базі даних на основі моделей.

Почнемо з опису моделі Profile, яка наслідується від стандартної моделі користувача в Django. Модель Profile визначає такі важливі атрибути, як фотографія профілю, режим відображення файлів, ліміт на використання дискового простору, і статус блокування. Це дозволяє ефективно управляти інформацією про користувачів та їхні права доступу.

Атрибути моделі:

- picture: Поле для зберігання шляху до зображення профілю.
- date: Дата створення профілю.
- file_display_mode: Режим відображення файлів, який може бути або у вигляді списку, або у вигляді сітки.
- blocked: Прапорець, що визначає, чи заблокований користувач.
- disk_limit: Максимально дозволений обсяг дискового простору для користувача.

Ця модель забезпечує гнучкість у налаштуванні користувацьких профілів та управлінні правами доступу, що є критичним для системи файлообмінника.

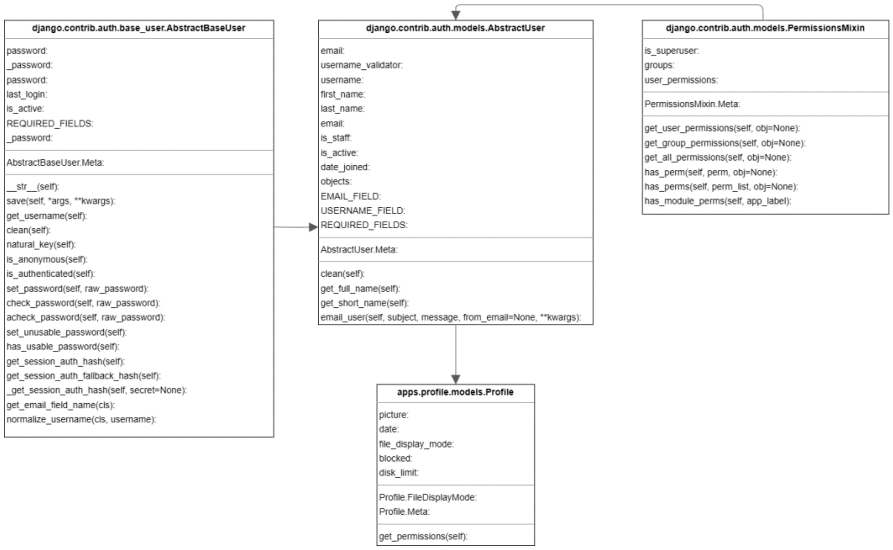


Рис. 2.3.1 – Діаграма класів для моделі Profile

Наступним кроком потрібно спроєктувати graphQL тип для цієї моделі, query та мутації.

GraphQL тип цієї моделі буде по суті ідентичним до самої моделі за виключенням того, що в ньому будуть додаткові поля «disk_usage» та «permissions», які не будуть зберігатися в базі даних, а обраховуватимуться безпосередньо при запиті.

В Query необхідно вказати всі можливі запити до моделі Profile. В даному проєкті до моделі Profile необхідно звертатися в трьох випадках:

- Отримати інформацію про конкретного користувача (user).
- Отримати інформацію про список користувачів (users).
- Отримати інформацію про себе (me).

Мутації для моделі Profile повинні надавати можливість змінювати інформацію про користувача в базі даних. В даному проєкті нам необхідно створити мутацію для реєстрації, входу в аккаунт, зміни інформації про користувача та інші, що стосуються JWT токена та систему безпеки, про яку детальніше буде описано пізніше.

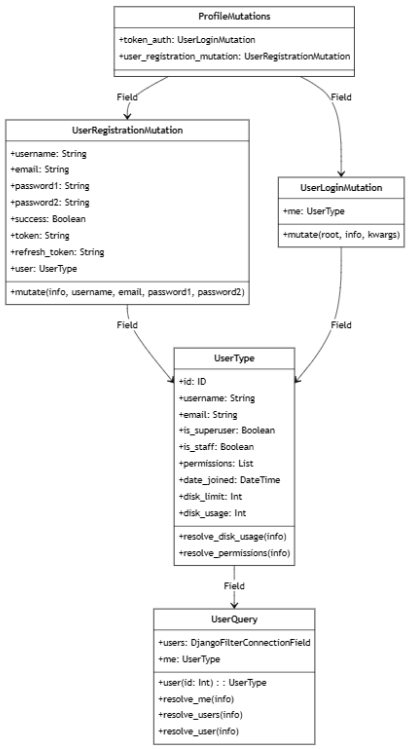


Рис. 2.3.2 Діаграма класів для GraphQL моделі «Profile»

Наступний крок у проєктуванні серверної частини медіасховища - розробка моделей файлової системи. Ця частина системи є критично важливою для забезпечення надійного зберігання, організації та доступу до файлів користувачів. Для цього буде створено дві основні моделі: Folder та File. Модель Folder буде відповідати за структуру папок, а модель File - за зберігання і управління файлами.

Модель Folder представляє структуру каталогів для організації файлів користувача. Вона підтримує ієрархічну структуру за допомогою MPTT (Modified Preorder Tree Traversal), що дозволяє створювати багаторівневі дерева папок. Основні атрибути і методи моделі Folder включають:

- name: Назва папки.
- user: Користувач, який володіє папкою.
- parent: Батьківська папка, що дозволяє створювати вкладені структури.
- created_at: Дата створення папки.
- updated_at: Дата останнього оновлення папки.
- link: Унікальне посилання для доступу до папки.
- accessed_users: Користувачі, які мають доступ до папки.
- is_shared: Прапорець, що визначає, чи буде доступною папка за посиланням для сторонніх користувачів.

Модель File відповідає за зберігання та управління файлами в системі. Кожен файл може бути пов'язаний з певною папкою і користувачем. Основні атрибути і методи моделі File включають:

- name: Назва файлу.
- file: Сам файл, що зберігається на сервері.
- folder: Папка, до якої відноситься файл.
- user: Користувач, який володіє файлом.
- link: Унікальне посилання для доступу до файлу.
- accessed_users: Користувачі, які мають доступ до файлу.
- is_shared: Прапорець, що визначає, чи доступний файл за посиланням для сторонніх користувачів.

- created_at: Дата створення файлу.
- updated_at: Дата останнього оновлення файлу.

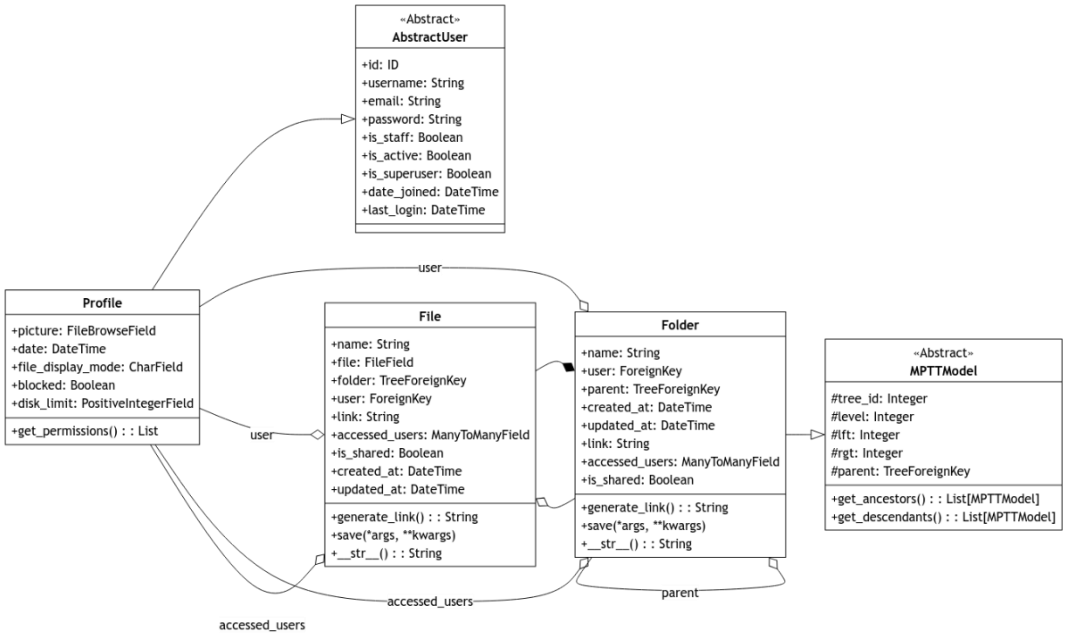


Рис. 2.3.3 – Діаграма класів для файлів та папок

На відміну від моделі Profile, створити query та мутації для файлової системи буде трішки складніше. Перш за все, варто врахувати, що сервер повинен забезпечити можливість пошуку файлів та папок за різними фільтрами. Також, оскільки Folder – MPTT модель, яка може мати великий рівень вкладеності, варто додати можливість отримувати так звані «хлібні крихти» (breadcrumbs), щоб на стороні клієнта можна було зручно відображати, на якому рівні вкладеності знаходиться користувач, а також дати йому додаткові можливості для навігації, щоб користувач міг в будь-який момент перейти на попередній рівень вкладеності.

Для папок Query будуть такі:

- folder – отримати конкретну папку за посиланням на неї.
- folders – отримати список всіх папок користувача з можливістю застосувати фільтри.

Для файлів буде тільки один Query:

- files – отримати список всіх файлів користувача з можливістю застосувати фільтри.

В наступному пункті буде пояснення, чому немає окремого Query, щоб отримати конкретний файл за посиланням.

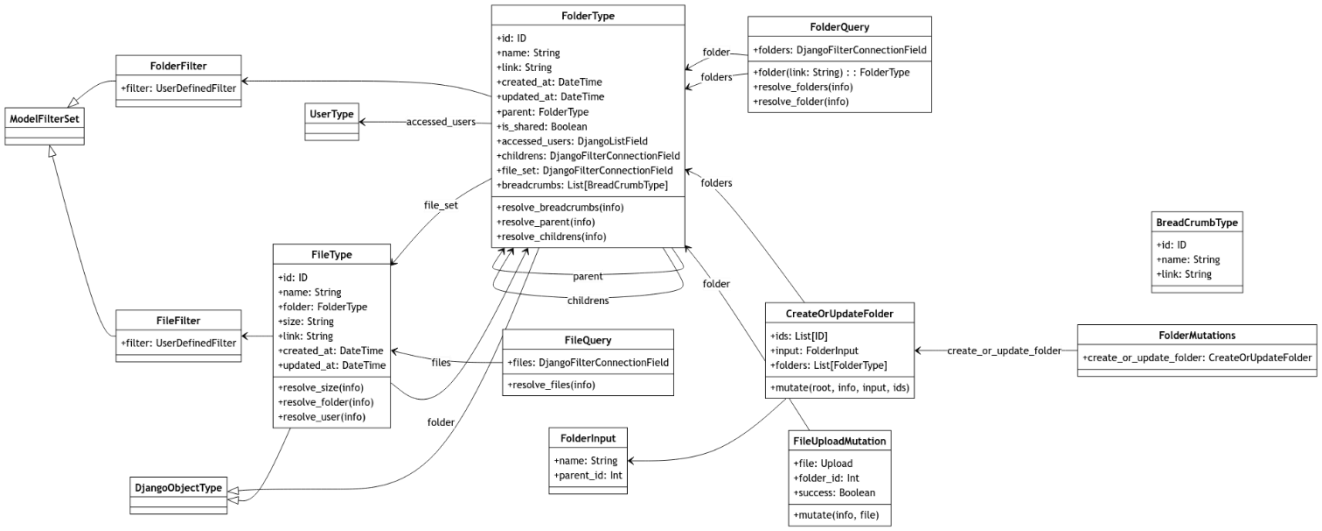


Рис. 2.3.4 – Діаграма класів для GraphQL файлової системи

Проєктування серверної частини медіасховища забезпечило створення масштабованої, надійної та зручної в користуванні системи, яка може обробляти великий обсяг даних та надавати доступ до них на різних рівнях. Ця система відповідає сучасним вимогам до безпеки та управління даними, що дозволяє користувачам ефективно організовувати та захищати свої файли.

2.4 Розробка серверної частини

На даний момент у процесі розробки медіасховища досягнуто значного прогресу. Було детально визначено функціональні та нефункціональні вимоги, які забезпечують базу для подальшої розробки. Проаналізовано та спроєктовано архітектуру системи, яка враховує її використання різними акторами та їх

взаємодію з системою. Розроблено та описано структуру бази даних, включаючи моделі для управління користувачами, папками та файлами. Всі ці компоненти закладають основу для створення надійної та масштабованої серверної частини.

Наступним кроком є реалізація спроєктованого функціоналу. Це включає написання коду, налаштування серверного середовища та інтеграцію всіх компонентів у цілісну систему. Розробка буде спрямована на забезпечення стабільної роботи сервера, коректного оброблення запитів користувачів та управління даними, а також на дотримання визначених вимог щодо продуктивності та безпеки.

2.4.1 Ініціалізація проєкту

Оскільки серверна частина написана на мові програмування Python, необхідно обрати середовище розробки, яке надавало би найкращий досвід написання коду на цій мові програмування. Мій вибір впав на PyCharm, який є безкоштовним для студентів і окрім того, що надає найбільший вбудований функціонал, також має плагіни – інструменти від сторонніх розробників, які розширюють функціонал IDE. Окрім цього, PyCharm має додатковий функціонал спеціально для розробки на фреймворку Django, вбудовані інструменти для роботи з git, вбудований термінал, можливість зручного налаштування віртуальних середовищ і взаємодії з базами даних.

Також, PyCharm надає можливість створення проєкту з заздалегідь заготовленою структурою під Django, тому варто скористатися цією можливістю і створити проєкт Django через PyCharm.

При розробці будь-якого проєкту, хорошою практикою є створення файлу requirements.txt, в якому будуть прописані всі бібліотеки, які використовуються в проєкті. Завдяки цьому для того, щоб налаштувати проєкт на сервері, можна не прописувати pip install для кожного модуля, а виконувати команду «pip install -r requirements.txt», яка автоматично встановить всі необхідні залежності.

Після того, як файл requirements.txt було створено, а всередині нього прописано всі необхідні залежності, за допомогою PyCharm можна створити

віртуальне середовище, відкрити термінал і прописати команду встановлення модулів, про яку згадувалось вище.

Наступним кроком при створенні проєкту є налаштування бази даних. Для цього в файлі «settings.py» необхідно в змінній «DATABASES» вказати конфігурацію бази даних. Проте, є кращий підхід. Для того, щоб не довелося вказувати ім’я користувача та пароль до бази даних безпосередньо в коді проєкту, варто використати .env файл. В корені проєкту необхідно створити файл «.env», в ньому створити змінні під хост бази даних, назву бази даних, ім’я користувача та пароль до неї. Після цього необхідно додати цей файл в «.gitignore», щоб ці конфіденційні дані не зберігалися в проєкті на github.

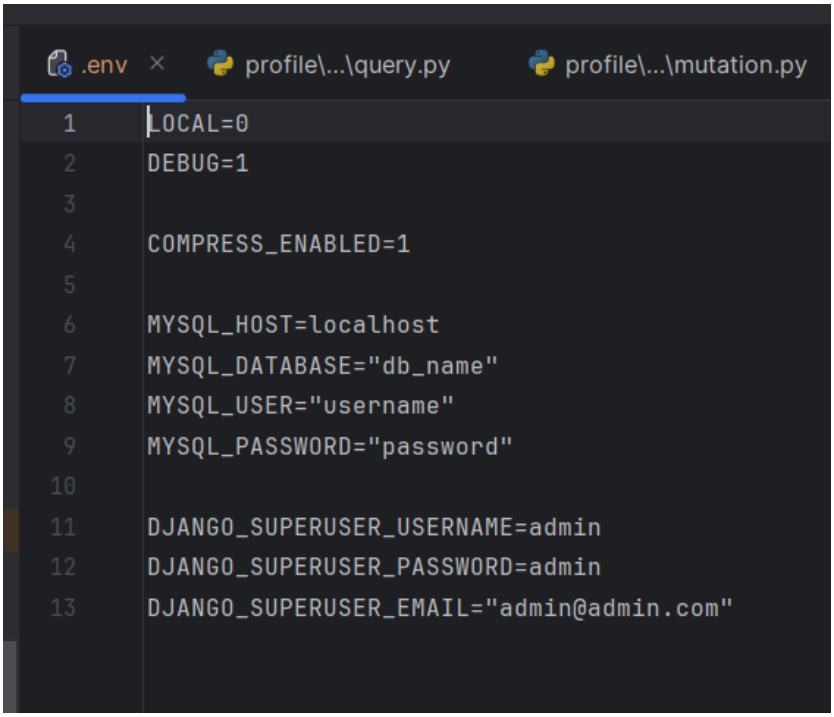


Рис. 2.4.1.1 – Приклад файлу .env

Після цього в файлі «settings.py» в змінній DATABASES вказувати дані зі змінних середовища.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': os.environ.get('MYSQL_DATABASE', 'database'),
        'USER': os.environ.get('MYSQL_USER', 'root'),
        'PASSWORD': os.environ.get('MYSQL_PASSWORD', 'password'),
        'HOST': os.environ.get('MYSQL_HOST', 'localhost'),
        'OPTIONS': {
            'charset': 'utf8',
        },
    }
}
```

Рис. 2.4.1.2 – Конфігурація бази даних з використанням змінних середовища

Для подальшої конфігурації проєкту необхідно провести «міграції» - це можливість, яку надає Django ORM для того, щоб автоматично генерувати таблиці в базі даних на основі створених моделей.

2.4.2 Реалізація спроєктованих моделей

В Django є можливість логічно згрупувати та розділити моделі. Наприклад, я хочу, щоб модель користувача та всі методи, які пов’язані з користувачем, були в одному файлі, а все, що пов’язано з файловою системою – в іншій. Для цього можна створювати окремі Django додатки. Для зручності всі додатки будуть створені в папці «apps», яку мною було створено в корені проєкту.

Створюємо додаток під профіль. Для цього необхідно через термінал відкрити папку apps і виконати команду «python ../manage.py startapp profile»

```
(venv) D:\TNTU\diploma\project\TNTUStorageBackend>cd apps/

(venv) D:\TNTU\diploma\project\TNTUStorageBackend\apps>python ../manage.py startapp profile
```

Рис. 2.4.2.1 – приклад створення Django додатка

Після цього Django автоматично створить папку додатка і згенерує всі необхідні файли для роботи з ним.

Після цього ми мусимо зайти в файл «models.py», що знаходиться в папці зі створеним додатком. Саме в ньому ми мусимо прописати спроєктовану модель користувача.

Варто зазначити, що в Django вже існує вбудована абстрактна модель користувача, в якій вже створені основні необхідні поля, такі як username, first_name, last_name, email і т.д, а також багато вбудованих корисних методів, частина яких відповідає за безпеку, тому при створенні моделі Profile ми будемо наслідуватись від «AbstractUser».

Рис. 2.4.2.2 – Реалізація моделі Profile

Далі необхідно аналогічно з попереднім прикладом створити додаток під файлову систему, і в папці models.py реалізувати спроєктовані моделі.

Для реалізації ієрархічної структури моделі «Folder» було використано модуль «Django MPTT» - модуль, який дозволяє працювати з ієрархічними даними в Django і надає інструменти для роботи з ними.

Варто також відмітити, що в моделі є зовнішній ключ (ForeignKey) на модель Profile, щоб можна було зрозуміти, якому користувачу належить ця папка. Окрім цього, є також поле accessed_users – відношення багато-до-багатьох, що показує, в яких користувачів є доступ до цієї папки.

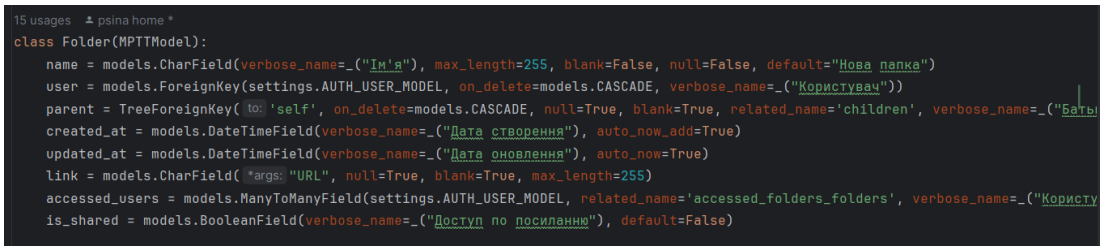


Рис. 2.4.2.3 – Реалізація моделі Folder

Медіасховище повинне надавати можливість відкрити доступ до папки за посиланням. Посилання на папку генерується в момент її створення. Для того, щоб уникнути дублікатів посилань, при генерації використовується повний шлях до цієї папки, після чого, з міркувань безпеки, хешується та зберігається в базі даних.

```
2 usages  psina home
def generate_link(self):
    """
    Генерує унікальне посилання на папку.
    """
    random_string = secrets.token_urlsafe(10)

    path = "/".join(self.get_ancestors().values_list('name', flat=True))
    data = f"{self.user.id}{path}"

    hashed_data = hashlib.sha256(data.encode('utf-8')).hexdigest()
    slugified_data = slugify(f"{hashed_data[:20]}{random_string}")
    if Folder.objects.filter(link=slugified_data).exists():
        return self.generate_link()
    return slugified_data

3 usages (2 dynamic)  psina home
def save(self, *args, **kwargs):
    if not self.pk:
        super(Folder, self).save( *args: *args, **kwargs)
    if not self.link:
        self.link = self.generate_link()
    super(Folder, self).save( *args: *args, **kwargs)
```

Рис. 2.4.2.4 – Методи збереження та генерації посилання моделі Folder

Для того, щоб можна було відносити певні файли до певних папок, в моделі File є зовнішній ключ на модель Folder. Як і в папках, в моделі файлу є зовнішній ключ на модель профілю, і відношення багато-до-багатьох, яке показує, які користувачі мають доступ до файлу. Поле file використовує FileField, яке Django надає для збереження файлів.

Як і в моделі Folder, модель File також має метод для генерації посилання на файл, який працює по схожій логіці.

```python
class File(models.Model):
    name = models.CharField(verbose_name=_("Ім'я"), max_length=255)
    file = models.FileField(upload_to='uploads/', verbose_name=_("Файл"))
    folder = TreeForeignKey(Folder, on_delete=models.CASCADE, verbose_name=_("Папка"), blank=True, null=True)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, verbose_name=_("Користувач"))
    link = models.CharField(verbose_name=_("Посилання для доступу"), max_length=255, blank=True, null=True)
    accessed_users = models.ManyToManyField(settings.AUTH_USER_MODEL, related_name='accessed_folders_files', verbose_name=_("Користу
    is_shared = models.BooleanField(verbose_name=_("Доступ по посиланню"), default=False)
    created_at = models.DateTimeField(verbose_name=_("Дата створення"), auto_now_add=True)
    updated_at = models.DateTimeField(verbose_name=_("Дата оновлення"), auto_now=True)

    psina home
    class Meta:
        ordering = ['-updated_at']

    2 usages  psina home
    def generate_link(self):
        """
        Генерує унікальне посилання на файл.
        """
        random_string = secrets.token_urlsafe(16)
        data = f"{self.user.id}{self.folder.get_ancestors(include_self=True).values_list('name', flat=True)}{self.name}{self.file.n
        hashed_data = hashlib.sha256(data.encode('utf-8')).hexdigest()
        slugified_data = slugify(f"{hashed_data}{random_string}")
        if File.objects.filter(link=slugified_data).exists():
            return self.generate_link()
        return slugified_data
```

Рис. 2.4.2.5 – Реалізація моделі File

2.4.3 Реалізація структури GraphQL

Для інтеграції GraphQL у проєкт на базі Django існує потужний модуль «Graphene Django». Цей інструмент забезпечує широкий спектр можливостей для легкої та ефективної імплементації GraphQL і дозволяє швидко створювати схеми, запити та мутації, забезпечуючи зручну та інтуїтивну роботу з даними. Окрім вбудованих можливостей, існує також багато модулів, які розширюють його функціональність, наприклад, модуль «Django GraphQL JWT», який надає готові схеми, запити та мутації для імплементації аутентифікації на основі JWT токенів.

Для встановлення модуля «Graphene Django» спочатку потрібно в файлі «settings.py» в списку «INSTALLED_APPS» додати "graphene_django". Після цього в головному файлі «urls.py» необхідно створити кінцевий пункт «graphQL»:

```python
urlpatterns = [
    path('admin/', admin.site.urls),
    path('i18n/', include('django.conf.urls.i18n')),
    path('lang/', context_processors.lang, name="lang"),
    re_path(r'^static/(?P<path>.*)$', serve, {'document_root': settings.STATIC_ROOT}),
    path("graphql/", csrf_exempt(jwt_cookie(GraphQLExtensionsView.as_view(graphiql=True)))),
    path('', include('apps.file_system.urls')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Рис. 2.4.3.1 – Глобальні кінцеві пункти проєкту

Після цього в файлі «settings.py» потрібно створити змінну «GRAPNENE», в якій потрібно вказати конфігурацію модуля, а саме посилання на файл зі схемою і проміжне програмне забезпечення, якщо таке є:

```
GRAPHENE = {
    "SCHEMA": "engine.schema.schema",
    'MIDDLEWARE': (
        'graphql_jwt.middleware.JSONWebTokenMiddleware',
    )
}
```

Рис. 2.4.3.2 – Конфігурація модуля «Graphene Django»

Далі потрібно налаштувати файл схеми. В схемі мають бути всі GraphQL запити та мутації. Для більшої зручності роботи, а також чистоти коду я прийняв рішення прописувати запити та мутації в Django додатках, в яких вони використовуються, а потім імпортувати їх в цей файл.

```
import graphene

from apps.file_system.graphql.mutation import FolderMutations
from apps.profile.graphql.mutation import ProfileMutations
from apps.profile.graphql.query import UserQuery
from apps.file_system.graphql.query import FolderQuery, FileQuery


1 usage  psina home
class Query(
    UserQuery,
    FolderQuery,
    FileQuery,
    graphene.ObjectType
):
    pass


1 usage  psina home
class Mutation(
    ProfileMutations,
    FolderMutations,
    graphene.ObjectType,
):
    pass


schema = graphene.Schema(query=Query, mutation=Mutation)
```

Рис. 2.4.3.3 – приклад файлу schema.py

І останнім кроком для завершення налаштування цього модуля є створення запитів та мутацій для певної моделі. Розглянемо на прикладі моделі Profile.

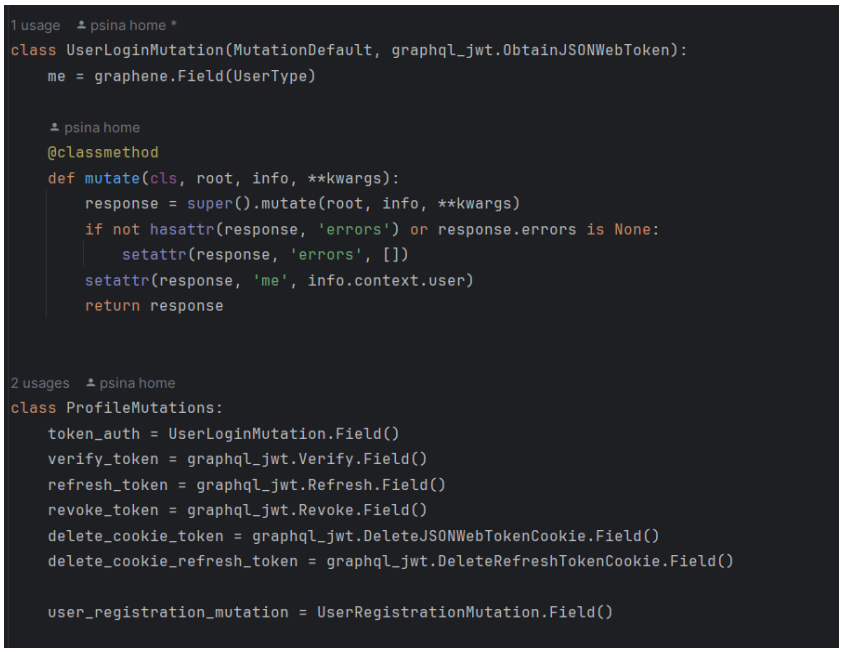
Спочатку потрібно створити тип. Оскільки ми прописуємо тип для моделі, потрібно наслідуватись від класу «DjangoObjectType», всередині цього класу потрібно створити мета-клас, в якому потрібно вказати, до якої моделі відноситься цей тип, список полів, які будуть використовуватись і інтерфейс. Інтерфейс відповідатиме за майбутнє представлення даних, які будуть повертатися у відповідь на запит до GraphQL.

Рис. 2.4.3.4 – Приклад створення GraphQL типу

Коли тип готовий, потрібно створити запит. В ньому потрібно вказати, які дані можна отримати. Для цього визначимо клас Query, в якому оголосимо всі необхідні запити.

Рис. 2.4.3.5 – Створення GraphQL запитів для моделі Profile

Окрім запитів, також необхідно створити мутації для внесення змін у дані. Мутації використовуються для створення, оновлення або видалення записів у базі даних.



```
1 usage  psina home *
class UserLoginMutation(MutationDefault, graphql_jwt.ObtainJSONWebToken):
    me = graphene.Field(UserType)

    psina home
    @classmethod
    def mutate(cls, root, info, **kwargs):
        response = super().mutate(root, info, **kwargs)
        if not hasattr(response, 'errors') or response.errors is None:
            setattr(response, 'errors', [])
        setattr(response, 'me', info.context.user)
        return response


2 usages  psina home
class ProfileMutations:
    token_auth = UserLoginMutation.Field()
    verify_token = graphql_jwt.Verify.Field()
    refresh_token = graphql_jwt.Refresh.Field()
    revoke_token = graphql_jwt.Revoke.Field()
    delete_cookie_token = graphql_jwt.DeleteJSONWebTokenCookie.Field()
    delete_cookie_refresh_token = graphql_jwt.DeleteRefreshTokenCookie.Field()

    user_registration_mutation = UserRegistrationMutation.Field()
```

Рис. 2.4.3.6 – створення GraphQL мутацій для моделі Profile

При реалізації запитів на папки та файли потрібно було також реалізувати фільтри. Для цього я використав модуль «Django GraphQL Extensions», який додає можливість створювати комплексні фільтри. Для того, щоб створити фільтр, потрібно створити для нього окремий клас, який повинен наслідуватись від моделі «ModelFilterSet», після чого в мета-класі graphQL типу потрібної моделі потрібно вказати в параметрі «filterset_class» створений під фільтр клас.

```
class FolderFilter(ModelFilterSet):
    filter = UserDefinedFilter(
        model=Folder,
        fields=["name", "user__id", "link", "created_at", "updated_at", "parent"],
    )

    psina home
    class Meta:
        order_by = ["id", "name", "link", "created_at", "updated_at", "parent"]
        model = Folder
```

Рис. 2.4.3.7 – створення комплексного GraphQL фільтру для моделі Folder.

2.4.4 Реалізація автентифікації

Для реалізації автентифікації за допомогою JWT токена, мною було використано модуль «Django GraphQL JWT». Це потужний модуль, який значно спрощує процес впровадження автентифікації та авторизації у проєкті, що використовує GraphQL. Django GraphQL JWT забезпечує функціонал для створення, перевірки та оновлення JWT токенів, що робить його ідеальним рішенням для безпечного управління доступом до ресурсів.

Щоб налаштувати модуль, необхідно додати його до списку «INSTALLED_APPS» у файлі «settings.py» та вказати Middleware для обробки JWT токенів. Конфігурація модуля вказується в тому ж файлі в змінній «GRAPHQL_JWT». Модуль пропонує велику кількість налаштувань в файлі конфігурацій, завдяки чому він гарно адаптується під вимоги будь-якого проєкту.

```
GRAPHQL_JWT = {
    "JWT_VERIFY": True,
    "JWT_VERIFY_EXPIRATION": True,
    "JWT_ALLOW_REFRESH": True,
    "JWT_LONG_RUNNING_REFRESH_TOKEN": True,
    'JWT_EXPIRATION_DELTA': timedelta(minutes=15),
    'JWT_REFRESH_EXPIRATION_DELTA': timedelta(days=7),
    "JWT_ALLOW_ARGUMENT": True,
}
```

Рис. 2.4.4.1 – Конфігурація модуля Django GraphQL JWT

Для підвищення зручності користувачів та забезпечення більшої безпеки було реалізовано логіку збереження JWT токенів у cookie. Такий підхід дозволяє значно зменшити ризик викрадення токенів через атаки типу XSS (Cross-Site Scripting) та покращує загальний рівень безпеки системи.

При вході користувача, сервер генерує JWT токен та відправляє його у відповідь в заголовку «Set-Cookie», після чого він зберігається у cookie на клієнтській стороні. Це забезпечує автоматичну автентифікацію користувача під час наступних запитів. Для автоматизації цього процесу модуль Django GraphQL JWT надає можливість використання декоратора «jwt_cookie» у файлі «urls.py» кінцевої точки GraphQL.

Крім того, було прийнято рішення щодо термінів дії токенів. Незважаючи на відсутність єдиного стандарту тривалості дії JWT токенів, було вирішено встановити термін дії токену автентифікації на 15 хвилин, а токену оновлення – на 7 днів. Такий вибір дозволяє забезпечити баланс між безпекою та зручністю для користувачів, оскільки токен автентифікації оновлюється досить часто для мінімізації ризиків, але не настільки часто, щоб це створювало незручності для користувачів.

2.4.5 Реалізація передачі файлів

При реалізації передачі файлів виникла низка проблем, які вимагали ретельного вирішення. У базі даних не можна зберігати об’єкти файлів безпосередньо. Хоча можливо конвертувати зображення у формат base64 і зберігати його в такому вигляді, значно кращим підходом є зберігання файлів на сервері, а в базі даних лише шляхів до них. Такий метод суттєво зменшує навантаження на базу даних і дозволяє в майбутньому реалізувати збереження файлів на різних дисках або навіть серверах, якщо обсяг даних зросте до відповідних масштабів.

Однак, виникає додаткова проблема – GraphQL не підтримує прямої передачі файлів. Для завантаження файлів на сервер, можливо реалізувати їх конвертацію у формат base64 на клієнтській стороні, а потім на сервері здійснювати зворотну конвертацію та збереження файлу. Проте, питання отримання файлів з сервера залишається відкритим. Конвертація у формат base64 для цієї задачі не є найоптимальнішим рішенням, оскільки це ускладнює код клієнтської частини та обмежує можливості завантаження і інтеграції файлів з медіасховища на інші сайти.

З огляду на ці обмеження, було прийняте рішення створити окремий маршрут у Django, який обслуговуватиме запити на отримання файлів. Цей підхід дозволяє

забезпечити прямий доступ до файлів, полегшує інтеграцію з іншими системами, а також збільшує надійність системи, оскільки навіть у випадку вразливостей клієнтської частини зловмисник не зможе отримати доступ до файлів користувача. Такий підхід зберігає всі переваги використання GraphQL для інших аспектів проєкту і не ускладнює проєкт.

Для реалізації цієї ідеї на стороні коду спочатку довелося в файлі «urls.py» додатку файлової системи створити кінцевий пункт, який буде відповідати за передачу файлів.

```
from apps.file_system.views import file_download_view
urlpatterns = [
    re_path(r'^media/(?P<path>.*)/$', file_download_view, name='file_download_view'),
]
```

Рис. 2.4.5.1 – Кінцевий пункт для отримання файлів

Після цього в файлі views.py того ж додатку було створено обробник цього кінцевого пункту. При написанні були оброблені всі помилки, які можуть статися, а також враховано, що якщо користувач намагається отримати файл, до якого в нього немає доступу, або якщо такого файла не існує, сервер повертатиме у відповідь помилку 404, яка вказує на те, що файл не знайдено.

```
def file_download_view(request, path):
    file = File.objects.filter(link=path).first()
    if not file:
        raise Http404(f""{path}" does not exist")
    user = request.user
    if file.user != user and not file.is_shared:
        raise Http404(f""{path}" does not exist")
    try:
        fullpath = safe_join(settings.MEDIA_ROOT, *paths: file.file.name)
    except SuspiciousFileOperation:
        raise Http404(f""{path}" does not exist")
    statobj = Path(fullpath).stat()
    if not was_modified_since(
        request.META.get("HTTP_IF_MODIFIED_SINCE"), statobj.st_mtime
    ):
        return HttpResponseNotModified()
    content_type, encoding = mimetypes.guess_type(str(fullpath))
    content_type = content_type or "application/octet-stream"
    try:
        file = open(fullpath, "rb")
        response = FileResponse( *args: file, content_type=content_type)
        response.headers["Last-Modified"] = http_date(statobj.st_mtime)
        if encoding:
            response.headers["Content-Encoding"] = encoding
        return response
    except FileNotFoundError:
        raise Http404(f""{path}" does not exist")
```

Рис. 2.4.5.2 – обробник кінцевого пункту для отримання файлів

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОХОРОНА ПРАЦІ

3.1 Характеристика життєдіяльності людини у системі «людина - машина – середовище існування»

Система «людина - машина - середовище існування» є однією з основних концепцій у дослідженні безпеки життєдіяльності. Ця система розглядає взаємодію людини з технічними засобами і навколишнім середовищем, спрямовану на забезпечення безпечних і ефективних умов життя та діяльності. Важливість цього підходу обумовлена не лише технічними й економічними аспектами, а й соціальними та екологічними вимогами, що стають все більш актуальними в сучасному світі.

У центрі системи «людина - машина - середовище існування» знаходиться людина, яка виконує різні види діяльності за допомогою машин і обладнання. Людина є ключовим елементом цієї системи, оскільки від її фізичних, психічних та інтелектуальних можливостей залежить ефективність і безпека взаємодії з технічними засобами та навколишнім середовищем.

Життєдіяльність людини - це активний процес, який визначається сукупністю взаємопов'язаних процесів, що відбуваються в системі "людина - середовище існування". [3] Вона має властивості, що випливають з потреб, можливостей і цінностей людини.

Людина:

- Фізіологічні особливості організму: Важливо розуміти, як фізіологічні функції, такі як дихання, кровообіг та діяльність нервової системи, впливають на здатність людини виконувати робочі завдання. Наприклад, робота в умовах підвищеного фізичного навантаження або екстремальних температур може перевищувати можливості адаптації організму і призводити до втоми чи захворювань.
- Психологічні аспекти: Психологічний стан працівника, включаючи стійкість до стресу, рівень концентрації та емоційний баланс, значною

мірою визначає продуктивність та безпеку на робочому місці. Стресові ситуації або монотонність можуть знижувати увагу та збільшувати ризик помилок або нещасних випадків.

Машина:

- Характеристики та функції машин: Машини, які використовуються в роботі, повинні бути безпечними, надійними та ергономічно спроектованими. Це означає, що конструкція та системи управління мають враховувати зручність і безпеку оператора. Наприклад, сучасні виробничі машини часто обладнані автоматизованими системами, які мінімізують потребу у фізичній взаємодії з ними, зменшуючи ризик травм.
- Вплив машин на працівника: Робота з машинами може мати різні наслідки для фізичного і психічного стану працівників. Постійний вплив шуму, вібрацій або електромагнітних полів може спричинити довгострокові проблеми зі здоров’ям. Також необхідно враховувати, як фізичні навантаження, пов'язані з роботою з машинами, можуть впливати на м'язово-скелетну систему.

Організація робочого місця:

- Раціональна організація робочого простору є ключем до ефективної і безпечної праці. Це включає розташування обладнання та інструментів таким чином, щоб мінімізувати непотрібні рухи і зменшити ризик нещасних випадків. Оптимізація простору і забезпечення зручного доступу до всіх необхідних елементів робочого процесу сприяє підвищенню продуктивності.

Середовище існування:

- Фактори робочого середовища: Робоче середовище може суттєво впливати на здоров’я і безпеку працівників. Наприклад, екстремальні температури, висока вологість або наявність шкідливих речовин можуть створювати небезпечні умови. Регулярний моніторинг цих

факторів і впровадження захисних заходів, таких як вентиляція або засоби індивідуального захисту, є необхідними для зменшення ризиків.

- Ергономічні аспекти: Ергономіка робочого місця повинна враховувати як фізичні, так і психологічні можливості працівника. Правильне розташування обладнання, комфортні меблі та зручні інструменти допомагають зменшити фізичне навантаження та запобігти стомленню. Крім того, ергономічний дизайн може сприяти кращій концентрації і знижувати ризик помилок.

Розуміння взаємодії між людиною, машинами і середовищем їхнього існування є критично важливим для створення безпечного і здорового робочого середовища. Глибоке дослідження фізіологічних і психологічних аспектів людської діяльності, характеристик машин та факторів робочого середовища дозволяє розробляти ефективні стратегії для зменшення ризиків, підвищення продуктивності та покращення загальної якості життя працівників. Це, у свою чергу, сприяє зменшенню травматизму та професійних захворювань, підвищуючи ефективність і задоволення від роботи. [4]

3.2 Методи розрахунку економічної ефективності заходів щодо покращенню умов та охорони праці.

У сучасному світі економічна ефективність є ключовим критерієм оцінки будь-яких заходів, спрямованих на покращення умов праці та охорони праці. Інвестиції в ці заходи не лише сприяють підвищенню безпеки працівників, але й мають значний економічний ефект, який проявляється у зменшенні витрат на компенсації, зниженні рівня абсентеїзму та підвищенні продуктивності праці. Розрахунок економічної ефективності дозволяє обґрунтувати необхідність вкладення ресурсів у поліпшення умов праці та демонструє їхню довгострокову вигідність. [5]

Економічна ефективність заходів щодо покращення умов праці та охорони праці визначається як відношення результатів від впровадження цих заходів до витрат на їх реалізацію. Основними показниками ефективності є:

Скорочення витрат на компенсації та лікування працівників.

Підвищення продуктивності праці завдяки зниженню рівня травматизму та захворюваності.

Зниження втрат робочого часу через абсентеїзм.

Підвищення мотивації та задоволеності працівників умовами праці.

Для оцінки економічної ефективності заходів щодо покращення умов праці та охорони праці використовують різні методи, серед яких найбільш поширені:

- Метод оцінки витрат та вигод (Cost-Benefit Analysis, CBA)

  Цей метод передбачає зіставлення витрат на впровадження заходів із вигодами, отриманими в результаті їх реалізації. Кроки проведення CBA включають:

  - Визначення всіх витрат на заходи щодо поліпшення умов праці (витрати на обладнання, навчання персоналу, адміністративні витрати тощо).
  - Оцінка вигод, таких як зменшення витрат на компенсації, зниження втрат робочого часу, зменшення рівня абсентеїзму та підвищення продуктивності.
  - Розрахунок чистого економічного ефекту як різниці між загальними вигодами та витратами.

  Формула чистого економічного ефекту (ЧЕЕ):

  ЧЕЕ = Загальні вигоди − Загальні витрати

- Метод розрахунку внутрішньої норми прибутку (Internal Rate of Return, IRR)

  IRR - це процентна ставка, при якій чистий приведений дохід (NPV) від заходів дорівнює нулю. Цей метод дозволяє оцінити ефективність інвестицій в заходи щодо охорони праці порівняно з іншими можливими інвестиціями. Чим вища IRR, тим вигідніше вкладення

коштів у поліпшення умов праці. Формула для розрахунку IRR:

$$NVP = \sum_{t=1}^{n} \frac{CF_t}{(1+IRR)^t} - I_0 = 0$$

де CF_t – грошові потоки в t-му періоді, I_0 – початкові інвестиції.

- Метод розрахунку чистої приведеної вартості (Net Present Value, NPV)
  NPV визначає різницю між приведеною вартістю грошових потоків, отриманих від заходів, та початковими інвестиціями. Цей метод враховує тимчасову вартість грошей і дозволяє оцінити довгострокову вигідність заходів. Формула для розрахунку NPV:

$$NPV = \sum_{t=1}^{n} \frac{CF_t}{(1+r)^t} - I_0$$

  де r – дисконтна ставка.

- Метод розрахунку періоду окупності (Payback Period, PP)
  Період окупності показує, за який час інвестиції в заходи окупляться за рахунок отриманих вигод. Цей метод є простим і наочним інструментом для оцінки ризиків і ефективності вкладень. Формула для розрахунку PP:

$$\text{PP} = \frac{I_0}{CF}$$

  де CF – середньорічні грошові потоки.

До основних факторів, що впливають на економічну ефективність заходів щодо покращення умов праці та охорони праці, належать:

- Вартість впровадження заходів: витрати на обладнання, навчання персоналу та інші необхідні ресурси.
- Тривалість впливу заходів: чим довше діють заходи, тим більше вигод можна отримати.
- Рівень ризиків: зменшення професійних ризиків сприяє зниженню витрат на компенсації та лікування.

- Зміни в продуктивності праці: покращення умов праці сприяє підвищенню продуктивності та зниженню втрат робочого часу.

Для ілюстрації методів розрахунку економічної ефективності розглянемо приклад:

Встановлення вентиляційної системи.

Витрати: 100 000 грн на встановлення системи вентиляції

Вигоди:

- Зменшення витрат на лікування працівників: 20 000 грн на рік.
- Зниження абсентеїзму: 15 000 грн на рік.
- Підвищення продуктивності: 10 000 грн на рік.

Дисконтна ставка (r): 10%

Період дії системи: припустимо, що система діятиме 10 років.

Розрахунок:

- Чиста економічна ефективність (ЧЕЕ):

  ЧЕЕ=20000+15000+10000=45000 грн на рік

  Час окупності = $\frac{100000}{45000}$ = 2.22 роки

- NPV:

$$\text{NPV} = \sum_{t=1}^{10} \frac{45000_t}{(1+0.10)^t} - 100000 = 176{,}505.52 \text{ грн}$$

  Це означає, що за 10 років інвестиції в систему вентиляції принесуть чистий прибуток у розмірі 176,505.52 грн з урахуванням дисконтної ставки 10%.

На основі розрахунків можна зробити висновок, що інвестиції у встановлення вентиляційної системи є економічно доцільними. Високе значення NPV свідчить про значний чистий прибуток, а високий IRR вказує на те, що інвестиції приносять значну рентабельність порівняно з іншими можливими інвестиціями, які б мали дисконтну ставку 10% або нижче. Ці результати підтверджують, що заходи щодо покращення умов праці та охорони праці не лише забезпечують безпеку та комфорт працівників, але й сприяють економічній вигоді для підприємства.

ВИСНОВКИ

Кваліфікаційна робота показує процес проєктування та розробки серверної частини медіасховища для ТНТУ ім. І. Пулюя, використовуючи фреймворк Django та технологію GraphQL, що дозволило створити безпечну, масштабовану та гнучку систему для зберігання та роботи з медіа-контентом.

У ході дослідження спочатку було здійснено детальний аналіз існуючих рішень для зберігання файлів, а саме Google Drive та Dropbox. Порівняння різних підходів до реалізації медіасховищ дозволило виділити ключові переваги та недоліки кожного з них, що допомогло сформувати функціональні та нефункціональні вимоги до системи. Ці вимоги стали основою для вибору технологій.

Рішення щодо використання фреймворків Django та Nuxt 3, а також технології GraphQL, було прийнято з урахуванням їхніх переваг у контексті стабільності, зручності розробки, підтримки спільноти, масштабованості та гнучкості. Django забезпечує надійну і стабільну основу для розробки серверної частини, дозволяючи легко керувати базою даних і реалізовувати бізнес-логіку. GraphQL, завдяки своїй здатності точно визначати запити до даних, значно покращує продуктивність і ефективність роботи з даними. Використання Nuxt 3 в якості фреймворку для клієнтської частини забезпечує зручну інтеграцію з сервером та підтримку SSR, що є важливим для продуктивності і дозволяє краще реалізувати захист даних.

Архітектура медіасховища базується на гібридному підході, що поєднує елементи багаторівневої та мікросервісної архітектур. Це забезпечує ефективний розподіл обов'язків між компонентами системи, а також їхню незалежність і масштабованість. Такий підхід дозволяє системі бути гнучкою, розширюваною та легко масштабованою, забезпечуючи можливість для інтеграції нових сервісів або функцій, таких як мобільні додатки, без значних змін в існуючій інфраструктурі.

Особлива увага була приділена проєктуванню структури бази даних. Було спроєктовано тмоделі даних, включаючи визначення зв'язків між ними. На основі цього були створені UML діаграми, які допомогли візуалізувати структуру даних і забезпечити чітке розуміння взаємозв'язків між різними елементами системи. Крім того, були спроєктовані запити та мутації GraphQL, які визначають, як користувачі можуть взаємодіяти з даними. Це включало детальне визначення всіх необхідних запитів для отримання даних та мутацій для їх зміни або додавання.

На етапі реалізації всі спроєктовані компоненти були впроваджені у функціонуючу систему. Було розроблено серверну частину на Django, яка інтегрується з GraphQL для забезпечення ефективного доступу до даних. Були реалізовані всі необхідні моделі та зв'язки в базі даних, що забезпечує надійне зберігання та управління медіа-контентом. Крім того, було впроваджено механізм автентифікації з використанням технології JWT (JSON Web Tokens), що забезпечує надійний захист доступу до медіасховища та відповідає сучасним стандартам безпеки.

У процесі реалізації передачі файлів виникла низка проблем, що вимагали ретельного вирішення. Для зручності отримання файлів було створено окремий маршрут у Django, що забезпечує прямий доступ до файлів, полегшує інтеграцію з іншими системами та підвищує надійність, оскільки у разі вразливостей клієнтської частини зловмисники не зможуть отримати доступ до файлів користувачів.

Розроблена система медіасховища є потужним інструментом для організації, зберігання та обробки медіа-контенту. Вона не лише забезпечує високу продуктивність та безпеку, але й пропонує гнучкі можливості для майбутнього розширення. У перспективі, подальший розвиток проєкту може включати оптимізацію продуктивності, розширення функціональності системи, а також покращення користувацького інтерфейсу, що зробить медіасховище ще більш зручним і корисним для користувачів.

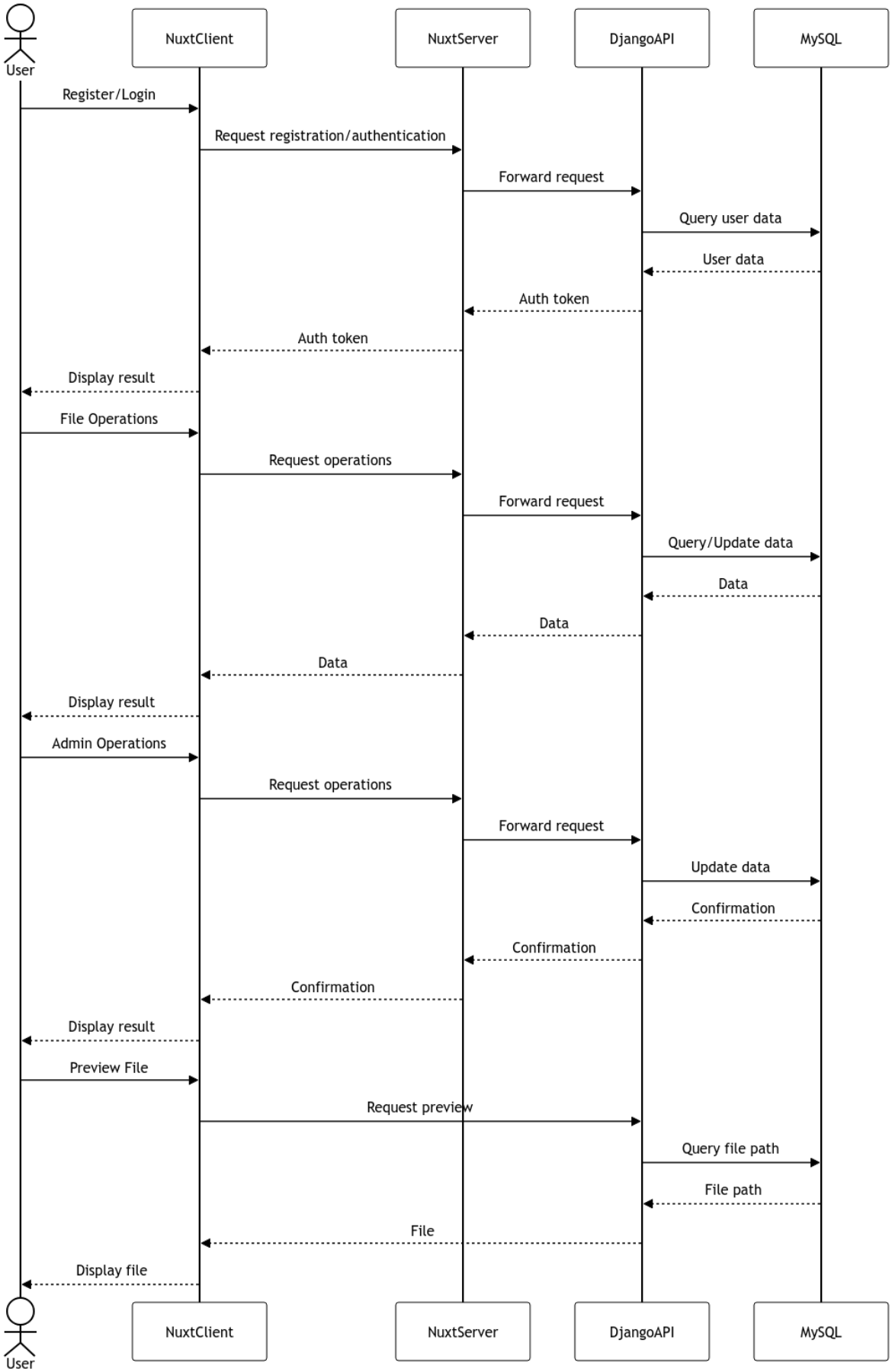
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. What Is Google Drive and How Does it Work? [Електронний ресурс] – Режим доступу до ресурсу: https://www.cloudwards.net/how-does-google-drive-work/
2. Taking Your Django App Live [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/django-unleashed/7-taking-your-django-app-live-deployment-and-beyond-121c9cbcf38d
3. Голд А. Python Projects / А. Голд, Л. Кассел., 2015. – 384 с. – (Wiley. John Wiley & Sons, LTD).
4. Mastering Django: A Beginner\'s Guide, 2022. – 302 с. – (Taylor & Francis). – (Mastering Computer Science).
5. Сесіл Р. М. Чиста архітектура / Роберт Мартін Сесіл., 2019. – 368 с. – (Фабула). – (#PROSystem).
6. Head First. Патерни проєктування / Е.Фрімен, Е. Робсон, Б. Бейтс, К. Сієрра., 2020. – 672 с. – (Фабула). – (#PROSystem).
7. Серіков Я. Основи охорони праці / Я. Серіков, Б. Халмурадов. – Харків: Центр учбової літератури, 2024. – 250 с.
8. Вахонєва Т. Основи охорони праці в Україні / Тетяна Вахонєва. – Київ: Дакор, 2019. – 508 с.
9. Запорожець О. Безпека життєдіяльності / О. Запорожець, В. Заплатинський. – Харків: Центр учбової літератури, 2024. – 448 с.
10. Мягченко О. Безпека життєдіяльності людини та суспільства / Олександр Мягченко. – Харків: Центр учбової літератури, 2023. – 384 с.
11. Запорожець О. Основи охорони праці. Підручник / О. Запорожець, О. Протоєрейський. – Харків: Центр учбової літератури, 2021. – 264 с.
12. Мудрик І. Проблеми аналізу унікальності контенту веб-сайтів у роботі SEO-оптимізатора / І. Мудрик, Р. Карагодін., 2023. – 161 с. – (ТНТУ).

13. М.Р. Петрик, І.Я. Мудрик Архітектура програмного забезпечення (на базі використання CASE-засобів IBM(sad)) навчальний посібник, Тернопіль: ТНТУ імені Івана Пулюя, 2017. 100с.
14. Glova B. Application of deep learning in neuromarketing studies of the effects of unconscious reactions on consumer behavior / B. Glova, I. Mudryk., 2020. – 337 с. – (Institute of Electrical and Electronics Engineers Inc.).
15. Analysis technology of neurological movements considering cognitive feedback influences of cerebral cortex signals / [M. Petryk, I. Mudryk, D. Mykhalyk та ін.]., 2022. – 45 с. – (CEUR-WS).

ДОДАТКИ

Додаток А. Діаграма послідовностей



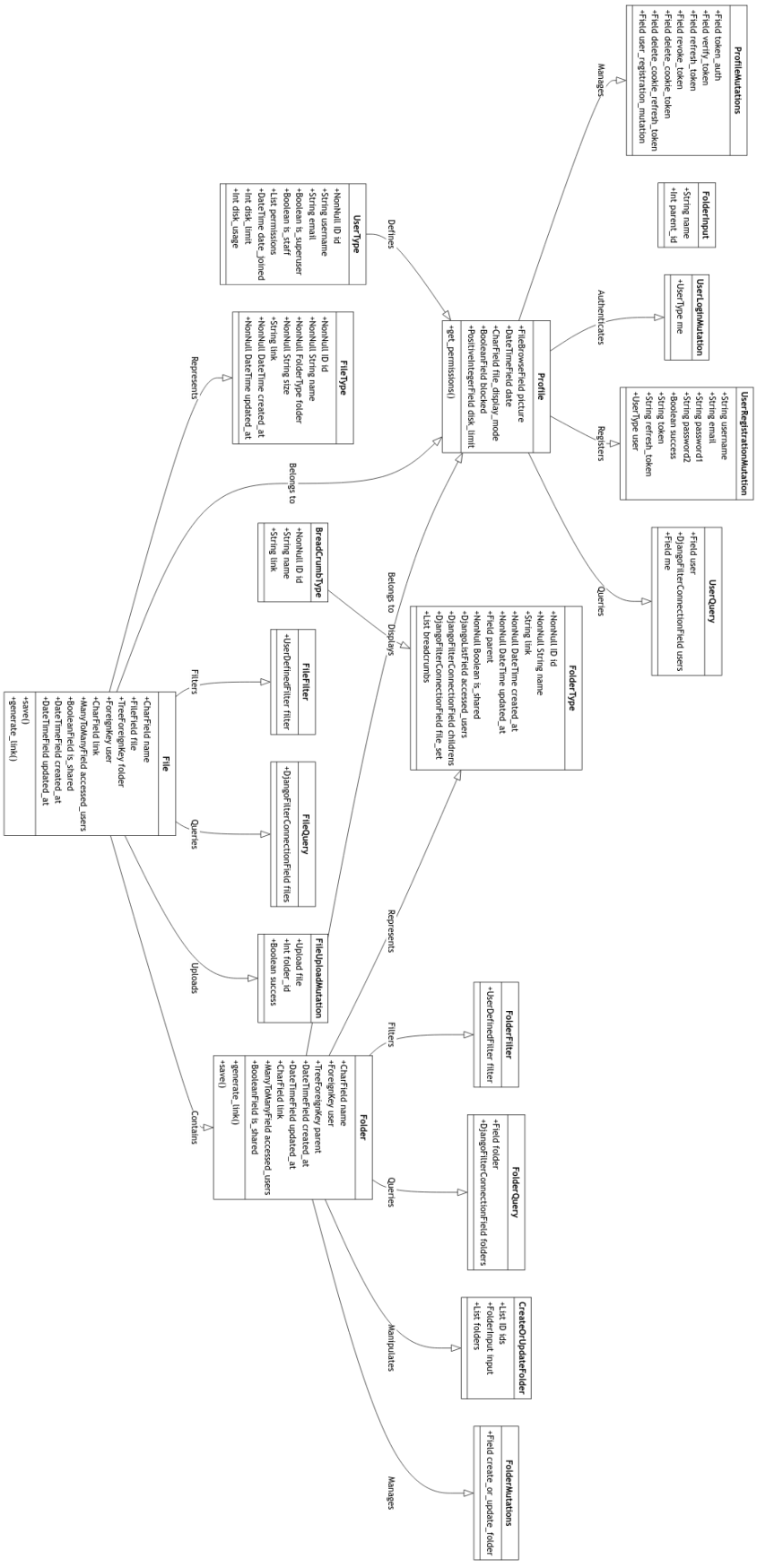
Додаток Б. Перша частина оновлення JWT токену в серверній частині Nuxt3

```
export default defineGraphqlServerOptions({
    async doGraphqlRequest({event, operationName, operationDocument, variables,}) {
        function doRequest(cookie: string) {
            return $fetch.raw('http://127.0.0.1:8000/graphql/', {
                method: 'POST',
                ignoreResponseError: true,
                credentials: 'include',
                body: {
                    query: operationDocument,
                    variables,
                    operationName,
                },
                headers: {
                    cookie: cookie,
                },
            })
        }
        function doRefreshToken(cookie: string) {
            return $fetch.raw('http://127.0.0.1:8000/graphql/', {
                method: 'POST',
                ignoreResponseError: true,
                credentials: 'include',
                body: {
                    query: 'mutation refreshToken{ refreshToken { token refreshToken payload refreshExpiresIn } }',
                },
                headers: {
                    cookie: cookie,
                },
            })
        }
        interface refreshTokenResponse {
            success: boolean;
            cookies: string;
        }
        async function refreshToken<T = refreshTokenResponse>(event: any, cookie: string): Promise<T> {
            const response_refresh = await doRefreshToken(cookie);
            if (response_refresh.status === 200) {
                if (!response_refresh._data.errors) {
                    const is_transferred = transferSetCookies(event, response_refresh);
                    if (is_transferred) {
                        const cookies_refresh_json = SetCookieParser.parse(response_refresh);
                        const cookie_format = cookies_refresh_json.map(item => `${item.name}=${item.value}`).join('; ');
                        return {
                            success: true,
                            cookies: cookie_format,
                        }
                    }
                } else {
                    const expires = new Date(new Date() - 1000).toUTCString();
                    ['JWT', 'JWT-refresh-token'].forEach((cookie_name) => {
                        if (getCookie(event, cookie_name)) {
                            appendHeader(event, 'set-cookie', `${cookie_name}=; expires=${expires}; Path=/; HttpOnly; Max-Age=0; SameSite=Lax`);
                        }
                    });
                    return {
                        success: false,
                        cookies: cookie,
                    }
                }
            }
            return {
                success: false,
                cookies: cookie,
            }
        }
        function transferSetCookies(event, response) {
            const setCookies = response.headers.get('set-cookie');
            if (setCookies) {
                const splitCookies = splitCookiesString(setCookies);
                if (splitCookies) {
```

Додаток В. Друга частина оновлення JWT токену в серверній частині Nuxt3

```
                    for (let cookie of splitCookies) {
                        if (cookie.name !== 'sessionid') {
                            appendHeader(event, 'set-cookie', cookie)
                        }
                    }
                    return true;
                }
            }
            return false;
        }
        interface JwtPayload {
            exp: number;
            origIat: number;
            username: string;
        }
        function parseJwt<T = JwtPayload>(token: string): T | null {
            try {
                return JSON.parse(Buffer.from(token.split('.')[1], 'base64').toString());
            } catch (e) {
                return null;
            }
        }
        const cookie = getHeader(event, 'cookie')
        const jwt_token = getCookie(event, 'JWT');
        const jwt_refresh_token = getCookie(event, 'JWT-refresh-token');
        let need_refresh = false;
        if (jwt_token) {
            const parsed_jwt = parseJwt(jwt_token);
            if (parsed_jwt) {
                const exp_date = new Date(parsed_jwt.exp * 1000);
                const created_date = new Date(parsed_jwt.origIat * 1000);
                // if 80% of token life is passed
                if (exp_date - created_date < 0.8 * (exp_date - new Date())) {
                    need_refresh = true;
                }
            }
        }
        if (jwt_refresh_token && !jwt_token) {
            need_refresh = true;
        }
        let refresh_response: refreshTokenResponse = {
            success: false,
            cookies: cookie,
        }
        if (need_refresh) {
            refresh_response = await refreshToken(event, cookie);
        }
        let result = await doRequest(refresh_response.cookies);
        if (result._data.errors) {
            let need_to_logout = false;
            for (let error of result._data.errors) {
                if (error.extensions?.code === 'AUTHENTICATION_REQUIRED') {
                    need_to_logout = true;
                }
                if (error.message === 'Invalid token') {
                    need_to_logout = true;
                }
            }
            if (need_to_logout) {
                const expires = new Date(new Date() - 1000).toUTCString();
                ['JWT', 'JWT-refresh-token'].forEach((cookie_name) => {
                    if (getCookie(event, cookie_name)) {
                        appendHeader(event, 'set-cookie', `${cookie_name}=; expires=${expires}; Path=/; HttpOnly; Max-Age=0; SameSite=Lax`);
                    }
                });
            }
        } else {
            transferSetCookies(event, result);
        }
        return result._data
    }
})
```

Додаток Г. Повна діаграма класів медіасховища



Додаток Ґ. Диск з роботою