Міністерство освіти та науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Звіт з комп’ютерного практикуму
з предмету
«Проєктування і розроблення інформаційних систем та технологій»
на тему**: “Електронна бібліотека – Інтернет-сервіс для бібліотеки”**

Перевірив:
проф. Теленик С. Ф.

Виконавці:
команда 23
студенти групи ІА-32мп
Гунавардана Ш. С. Д.
Левадський Д. А.
Чечет М. О.

Зміст

Вступ

Електронна бібліотека (ЕБ) – це інформаційна система, яка забезпечує доступ користувачів до електронних документів, таких як книги, статті, журнали, звіти та інші. ЕБ можуть бути доступні в Інтернеті або локальній мережі.

ЕБ мають ряд переваг перед традиційними бібліотеками. Вони пропонують більш широкий спектр документів, більш швидкий доступ до інформації та більш зручне використання. ЕБ також є більш доступними, оскільки їх можна використовувати з будь-якого місця з доступом до Інтернету.

Розробка ЕБ є складним завданням, яке вимагає спеціальних знань і навичок у галузі інформаційних технологій. При розробці ЕБ необхідно враховувати такі фактори, як:

Перші електронні бібліотеки були розроблені в 1960-х роках. Вони були невеликими і містили обмежену кількість документів. З розвитком Інтернету в 1990-х роках електронні бібліотеки стали більш поширеними. Сьогодні існує велика кількість електронних бібліотек, які пропонують широкий спектр документів.

Електронні бібліотеки можна класифікувати за різними ознаками. За типом документів, які вони зберігають, електронні бібліотеки поділяються на:

- Загальні бібліотеки, які містять документи різного характеру
- Спеціалізовані бібліотеки, які містять документи з певної галузі знань

За методом доступу до документів електронні бібліотеки поділяються на:

- Відкриті бібліотеки, які доступні в Інтернеті
- Закриті бібліотеки, доступ до яких обмежений

За технологією, яка використовується для зберігання та обробки даних, електронні бібліотеки поділяються на:

- Бібліотеки, які використовують базу даних
- Бібліотеки, які використовують файлову систему
- Технології, що використовуються в електронних бібліотеках

Для зберігання та обробки даних в електронних бібліотеках використовуються різні технології. Найпоширенішими технологіями є:

- База даних
- Файлова система
- Сховища об'єктів

База даних – це структурована сукупність даних, яка зберігається та обробляється в єдиному місці. Бази даних використовуються для зберігання великої кількості документів.

Файлова система – це спосіб організації даних на комп'ютері. Файлова система використовується для зберігання невеликої кількості документів.

Сховища об'єктів – це технологія, яка дозволяє зберігати та обробляти дані у вигляді об'єктів. Сховища об'єктів використовуються для зберігання документів різного характеру.

Інтерфейс користувача (ІК) електронної бібліотеки – це спосіб взаємодії користувача з ЕБ. ІК повинен бути простим і зрозумілим для користувачів.

Існує два основних типи ІК електронних бібліотек:

- ІК, заснований на веб-браузері
- ІК, заснований на клієнтському додатку

ІК, заснований на веб-браузері, є найбільш поширеним типом ІК. Він дозволяє користувачам отримувати доступ до ЕБ з будь-якого комп'ютера з доступом до Інтернету.

ІК, заснований на клієнтському додатку, дозволяє користувачам отримувати доступ до ЕБ з локального комп'ютера. Він пропонує більш зручне використання, ніж ІК, заснований на веб-браузері.

Електронні бібліотеки є важливим інструментом для отримання інформації. Вони пропонують більш широкий спектр документів, більш швидкий доступ до інформації та більш зручне використання, ніж традиційні бібліотеки. Розробка ЕБ є складним завданням, яке вимагає спеціальних знань і навичок у галузі інформаційних технологій.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Мета та постановка задачі

Мета даної роботи полягає у створенні електронної бібліотеки, яка відповідає сучасним вимогам до інформаційних систем та забезпечує швидкий, зручний та ефективний доступ до великої кількості освітніх та наукових матеріалів. Основною метою є створення інтерактивного, безпечного та інтуїтивно зрозумілого ресурсу, який би спростив процес знаходження та використання інформації для користувачів різного віку та освітнього рівня.

У сучасному світі, де цифровізація стає невід'ємною частиною нашого життя, значення електронних бібліотек не може бути переоцінене. Вони відіграють ключову роль у забезпеченні легкого доступу до величезної кількості інформації, сприяючи освітньому розвитку та науковим дослідженням. Електронні бібліотеки відкривають нові горизонти для здобуття знань, роблячи їх доступними в будь-який час та з будь-якої точки світу. Це особливо важливо у контексті постійного розвитку технологій та зростаючих вимог до освітнього процесу.

Реалізація проекту створення електронної бібліотеки передбачає не тільки використання передових технологій та інноваційних підходів до обробки та зберігання інформації, але й вимагає ретельного аналізу потреб користувачів та ринкових тенденцій. Постійне оновлення та підтримка системи, забезпечення її безпеки та надійності, а також забезпечення інтуїтивно зрозумілого інтерфейсу – все це визначає якість сучасної електронної бібліотеки. Відповідно до цих вимог, проект спрямований на створення дієвого та ефективного інструменту доступу до знань для широкого кола користувачів.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

- провести аналіз предметної області;
- аналіз існуючих аналогів та цільової аудиторії;
- визначення та аналіз функціональних та нефункціональних вимог;
- вибір методів та засобів реалізації;
- розробка дизайну та інтерфейсу сайту;
- розробка технічного завдання та системи;
- створення архітектури веб-додатку;
- розробка моделі бази даних;
- програмування системи;
- тестування інформаційної системи;
- розробка документації;
- впровадження веб-додатку замовнику;
- супроводження протягом всього періоду реалізації.

1.2 Опис проблеми

Бібліотеки є важливим культурним і освітнім інститутом у будь-якому суспільстві. Вони надають доступ до знань і інформації, а також сприяють розвитку освіти, культури та творчості.

Традиційні бібліотеки використовують локальні інформаційні системи (ІС), які розміщені на власних серверах. Ці системи мають ряд переваг, зокрема:

- Повний контроль над даними та програмним забезпеченням
- Можливість адаптації ІС до конкретних потреб бібліотеки
- Високий рівень безпеки даних

Однак локальні ІС також мають ряд недоліків, зокрема:

- Висока вартість придбання та обслуговування
- Необхідність наявності кваліфікованих ІТ-фахівців
- Ограничена масштабованість

У останні роки все більшої популярності набувають SaaS-бібліотечні ІС. SaaS (Software as a Service) - це модель надання програмного забезпечення як послуги. У цьому випадку ІС розміщена на серверах провайдера SaaS і доступна користувачам через Інтернет.

SaaS-бібліотечні ІС мають ряд переваг, зокрема:

- Низька вартість придбання та обслуговування
- Необхідність наявності лише базових ІТ-знань
- Висока масштабованість

Однак SaaS-бібліотечні ІС також мають ряд недоліків, зокрема:

- Зменшений контроль над даними та програмним забезпеченням
- Ограничена можливість адаптації ІС до конкретних потреб бібліотеки
- Необхідність довіри до провайдера SaaS

На основі аналізу переваг і недоліків традиційних і SaaS-бібліотечних ІС можна виділити наступні проблеми, які необхідно вирішити при проектуванні і розробці SaaS-бібліотечної ІС:

- Контроль над даними та програмним забезпеченням. SaaS-бібліотечні ІС передбачають передачу даних і програмного забезпечення на сервери провайдера SaaS. Це може призвести до обмеженого контролю бібліотеки над своїми даними і програмним забезпеченням.
- Адаптивність до конкретних потреб бібліотеки. SaaS-бібліотечні ІС розробляються для широкого кола бібліотек. Це може призвести до обмеженої можливості адаптації ІС до конкретних потреб конкретної бібліотеки.
- Безпека даних. SaaS-бібліотечні ІС зберігають дані користувачів на серверах провайдера SaaS. Це вимагає від провайдера SaaS забезпечення високого рівня безпеки даних.

Для вирішення цих проблем необхідно розробити SaaS-бібліотечну ІС, яка буде забезпечувати:

- Надійний захист даних користувачів. ІС повинна використовувати сучасні технології безпеки для захисту даних користувачів від несанкціонованого доступу, модифікації або знищення.
- Гнучкість і адаптивність. ІС повинна забезпечувати можливість адаптації до конкретних потреб бібліотеки. Це можна досягти за допомогою модульної архітектури ІС, яка дозволяє легко додавати або змінювати функціональні можливості ІС.
- Прозорість та контроль. ІС повинна забезпечувати прозорість і контроль бібліотеки над своїми даними та програмним забезпеченням. Це можна досягти за допомогою використання відкритих стандартів і протоколів, а також надання бібліотеці можливості контролювати доступ до своїх даних.

Розв'язання цих проблем дозволить створити SaaS-бібліотечну ІС, яка буде відповідати потребам сучасних бібліотек.

1.3 Проектні припущення та обсяг продукту

В нашому веб-додатку після аналізу були зроблені наступні проектні припущення:

1. Операції веб-додатку повинні охоплювати великі фрагменти функціональності і зосереджуватися на операціях застосування, бо занадто дрібні операції погано впливають на продуктивність і масштабованість. Сервіс не повинен повертати необмежено великі об'єми даних.
2. Контракти даних мають бути спроектовані так, щоб їх можна було розширювати, не роблячи впливу на споживачів сервісу. Для підтримки змін, що не мають зворотної сумісності, можливо, доведеться створювати нові версії інтерфейсу сервісу, які функціонуватимуть паралельно з існуючими версіями
3. Шар сервісів повинен реалізовувати і забезпечувати тільки функціональність, обумовлену контрактом сервісу. Внутрішня реалізація і деталі сервісу ніколи не повинні розкриватися зовнішнім споживачам. При необхідності змінити контракт сервісу необхідно розглянути можливість створення версій контрактів
4. Відділення функціональністі шару сервісу від функціональності інфраструктури. У шарі сервісів реалізація наскрізної функціональності не повинна поєднуватися з кодом логіки сервісу, оскільки це може ускладнити розширення і обслуговування реалізацій.

5. Сутності повинні створюватись із стандартних елементів. По можливості будуть компонуватись складні типи і об'єкти передачі даних, використовувані сервісом, із стандартних елементів.
6. Потрібно враховувати можливість вступу некоректних запитів. Ніколи не можна робити припущення про те, що всі повідомлення, що поступають в сервіс, будуть коректними. Буде реалізовано логіку валідації усіх вхідних даних на підставі значень, діапазонів і типів даних, також буде відхилення неприпустимих даних.
7. Загальні терміни реалізації проекту повинні складати приблизно 12-18 місяців, що включає етапи планування, розробки, тестування та впровадження. Окрім цього, заплановано, що мінімально життєздатна версія (MVP) додатку буде реалізована протягом перших 4 місяців роботи над проектом.
8. Очікуваний бюджет проекту становить приблизно 20 000 - 30 000 доларів США. Ця сума передбачає витрати на розробку, базове обладнання та хмарні сервіси. З огляду на те, що команда складається з трьох студентів, передбачається, що витрати на заробітну плату будуть мінімальними або відсутніми.
9. Використання технологій та інструментів буде залежати від доступних ресурсів та потреб проекту. Основна увага буде приділена використанню безкоштовних або відкритих технологій, що дозволить мінімізувати витрати та спростити процес розробки.
10. Потрібно також враховувати можливість вступу некоректних запитів. Всі повідомлення, що поступають в сервіс, повинні

проходити через валідацію на підставі значень, діапазонів і типів даних, а також неприпустимі дані повинні бути відхилені.

1.4 Характеристика користувачів

Характеристика користувачів є ключовим аспектом при розробці електронних бібліотек, оскільки різноманітність аудиторії передбачає врахування їхніх особливостей, потреб та вимог. У цьому розділі ми детально розглянемо профілі користувачів нашого інтернет-сервісу, звертаючись до їхніх освітніх рівнів, наукових інтересів, інформаційних звичок та очікувань від електронних бібліотек.

При розробці електронної бібліотеки ключовим аспектом є розуміння характеристик користувачів, оскільки вони впливають на формування функціоналу та інтерфейсу сервісу. Різноманітність аудиторії передбачає врахування їхніх особливостей, потреб та вимог. Детально розглянемо профілі користувачів нашого інтернет-сервісу, звертаючи увагу на їхні освітні рівні, наукові інтереси, інформаційні звички та очікування від електронних бібліотек:

- студент. Ця група користувачів шукає матеріали для отримання базових знань та виконання навчальних завдань. Наш сервіс надає зручний пошук та фільтрацію, дозволяючи швидко знаходити необхідні ресурси. Також пропонуємо інструменти для вивчення нових тем через інтуїтивно зрозумілий інтерфейс;
- викладачі та науковці. Шукають спеціалізовані наукові ресурси для підтримки своїх досліджень та викладацької діяльності. Наш сервіс забезпечує доступ до наукових журналів, актуальних публікацій та розширені можливості пошуку. Також надаємо функції зберігання та організації персональних бібліотек;

- дослідники-аматори. Ця категорія включає індивідуальних дослідників, які займаються незалежними дослідженнями. Наш сервіс надає інструменти для детального пошуку, доступу до рідкісних або спеціалізованих ресурсів, а також можливість обміну коментарями та відгуками із спільнотою;
- професіонали та практикуючі фахівці. Шукають ресурси для підтримки своєї професійної діяльності. Наш сервіс забезпечує доступ до професійних публікацій та журналів, кейс-стаді та матеріалів для підвищення кваліфікації;
- школярі та їхні батьки. Шукають матеріали для підтримки шкільного навчання та підготовки до іспитів. Наш сервіс надає легкий доступ до навчальних посібників та інтерактивних ресурсів;
- публічні бібліотекарі та освітні працівники. Використовують ресурси для організації освітніх заходів та підтримки своєї професійної діяльності. Сервіс надає інструменти для керування освітніми ресурсами та інтеграції з іншими платформами.

1.5 Освітні рівні користувачів

Освітній рівень користувачів є важливим визначником їхніх індивідуальних потреб та очікувань від електронної бібліотеки. Розгляд цього аспекту допомагає адаптувати функціонал інтернет-сервісу до конкретних вимог різних груп користувачів.

Однією з ключових груп є студенти. Їх освітні потреби часто спрямовані на отримання базових знань та джерел для виконання навчальних завдань. Тому, наш сервіс надає зручний пошук та фільтрацію, що дозволяє швидко знаходити необхідні для навчання матеріали. Також враховується можливість вивчення нових тем і розширення знань через зручний інтерфейс.

Викладачі та науковці, які становлять іншу значущу групу, часто шукають спеціалізовані наукові ресурси та публікації для підтримки своїх досліджень та викладацької діяльності. Наш інтернет-сервіс спрямований на підтримку їх потреб, забезпечуючи розширені можливості пошуку, доступ до наукових журналів та актуальних публікацій. Також, ми надаємо можливість зберігання та організації власної бібліотеки для подальшого використання.

У той же час, інші фахівці та дослідники, які вже мають великий науковий досвід, можуть шукати високоспеціалізовані матеріали для вирішення складних наукових завдань. Відповідно, наш інтернет-сервіс надає інструменти для точного та глибокого пошуку, а також можливість взаємодії з іншими дослідниками через обмін коментарями та відгуками.

Освітні рівні користувачів формують унікальні потреби, які впливають на розробку функціоналу нашого інтернет-сервісу. Враховуючи ці потреби, ми прагнемо створити зручний та доступний інструмент для всіх категорій користувачів, сприяючи їхньому активному залученню у процеси наукових досліджень та освіти.

1.6 Наукові інтереси та галузі досліджень

Розуміння наукових інтересів та галузей досліджень користувачів є важливим елементом розробки ефективної електронної бібліотеки. Розглянемо різноманітні аспекти цього підгрупи аудиторії та як наш інтернет-сервіс спрямований на задоволення їхніх унікальних потреб.

Студенти, завдяки різноманітності своїх програм навчання, можуть мати інтереси в різних галузях науки. Інтернет-сервіс спрямований на надання доступу до актуальних джерел інформації у різних предметних областях. Це може включати матеріали для природничих наук, гуманітаріїв, соціальних та технічних наук. Ми розробляємо функціонал

для пошуку та фільтрації, щоб студенти могли ефективно знаходити не лише підручники та наукові статті, а й матеріали, що відповідають їхнім конкретним дисциплінам.

Викладачі та науковці, що є складовою іншої важливої групи, часто спеціалізуються в конкретних наукових галузях. Вони можуть шукати актуальні наукові публікації, результати досліджень та інші ресурси для підтримки своєї наукової роботи. Наш сервіс надає можливість вибору конкретних тематичних галузей для вивчення, підписки на журнали та доступу до баз даних, спеціалізованих на конкретних наукових напрямах.

Фахівці та дослідники із великим науковим стажем можуть бути зацікавлені в глибокому вивченні конкретних аспектів своєї галузі. Наш інтернет-сервіс надає інструменти для точного та детального пошуку, а також забезпечує можливість взаємодії з іншими дослідниками шляхом обміну коментарями та обговоренням новітніх тенденцій у їхній галузі.

Враховуючи різноманітність наукових інтересів та галузей досліджень, наш інтернет-сервіс створений таким чином, щоб відповідати на потреби різних категорій користувачів. Ми прагнемо забезпечити широкий та різноманітний доступ до актуальної наукової інформації, сприяючи розвитку наукових досліджень у всіх їхніх різновидностях.

1.7 Інформаційні звички та технологічна грамотність

Різнорівневий розвиток інформаційних технологій та технологічна грамотність користувачів є критичними аспектами, що визначають їхню здатність ефективно користуватися електронною бібліотекою. Розглянемо, як інтернет-сервіс враховує ці аспекти, створюючи зручне та доступне середовище для різних рівнів інформаційної грамотності.

Студенти, які виявляють різні рівні інформаційної грамотності, можуть мати різні підходи до використання електронних ресурсів. Наш

сервіс надає простий та інтуїтивно зрозумілий інтерфейс для менш досвідчених користувачів, а також розширені функції для тих, хто здатен використовувати більше технічно орієнтовані інструменти.

Викладачі та науковці, які часто є більш технологічно грамотними, можуть користуватися розширеними можливостями пошуку та аналізу даних. Для них наш інтернет-сервіс пропонує інструменти для глибшого вивчення ресурсів, включаючи розширені фільтри та аналітичні можливості.

Особи з великим науковим стажем можуть виявляти високий рівень технологічної грамотності, шукаючи інноваційні та передові методи доступу до наукової інформації. Ми реагуємо на це, надаючи передові інструменти та можливості для ефективного використання інтернет-сервісу.

Враховуючи різноманітність рівнів технологічної грамотності, наш інтернет-сервіс працює над тим, щоб відповідати на потреби всіх категорій користувачів. Ми прагнемо створити дружнє та доступне середовище, яке забезпечить кожному користувачеві максимальний комфорт при використанні електронної бібліотеки, незалежно від його рівня інформаційної грамотності.

1.8 Очікування від електронних бібліотек

Очікування користувачів від електронних бібліотек визначаються їхнім досвідом, потребами та специфікою науково-дослідницької діяльності. Розглянемо, як наш інтернет-сервіс враховує ці очікування, створюючи функціонал, що відповідає різним потребам користувачів.

Студенти, які переважно шукають матеріали для навчання, очікують від електронної бібліотеки простоти та доступності. Наш сервіс пропонує інтуїтивно зрозумілий інтерфейс та ефективні інструменти пошуку,

спрощуючи процес знаходження інформації для студентів різних дисциплін.

Викладачі та науковці часто розраховують на електронні бібліотеки для доступу до останніх наукових досягнень та для підтримки власних досліджень. Наш сервіс ставить перед собою завдання забезпечити високий рівень актуальності і доступності наукової інформації, а також функціонал для організації та зберігання знань.

Фахівці з великим науковим досвідом можуть очікувати від електронної бібліотеки розширених можливостей пошуку, аналізу та взаємодії з іншими дослідниками. Наш сервіс надає інструменти для глибокого аналізу і структурування даних, а також можливість спілкування та обговорення результатів.

Очікування користувачів збільшуються із зростанням їхнього наукового досвіду та потреб. Наш інтернет-сервіс відповідає цим очікуванням, надаючи гнучкі налаштування та персоналізовані функції. Ми прагнемо створити інтелектуальний та ефективний інструмент, що задовольняє різноманітні потреби користувачів і сприятиме подальшому розвитку наукових досліджень у всіх сферах.

2. БІЗНЕС ПЛАН

У нашій системі бізнес моделі є два головних предмета аналізу та дослідження - організація як власник продукту, та IT компанія як розробник.

Роль організації заключается в створенні плану реалізації продукту, що має включати у себе комплекс аналітичних прорахунків на основі яких робиться висновок окупності фінального продутку та робиться відповідне рішення чи є сенс рухатись далі. Якщо первинний аналіз показав роботоспроможність даного IT рішення, то далі організація переходить до узгодження прорахованого етапу розробки та етапу реалізації готового продукту. Серед головних критеріїв плану реалізації можна виділити аналіз актуальності та затребуваності, прорахунок прибутків, кількість часу та людино-годин потрібних для повної та MVP реалізації, аналіз вартості розробки (див. рисунок 2.1).

Крім цього не мало важливою складовою також є ефективний аналіз потенційних ризиків та планування сценаріїв вирішення їх наслідків. Не можна нехтувати навіть мінімальними ризиками якщо мова йде про масштабованість великого enterprise продукту.

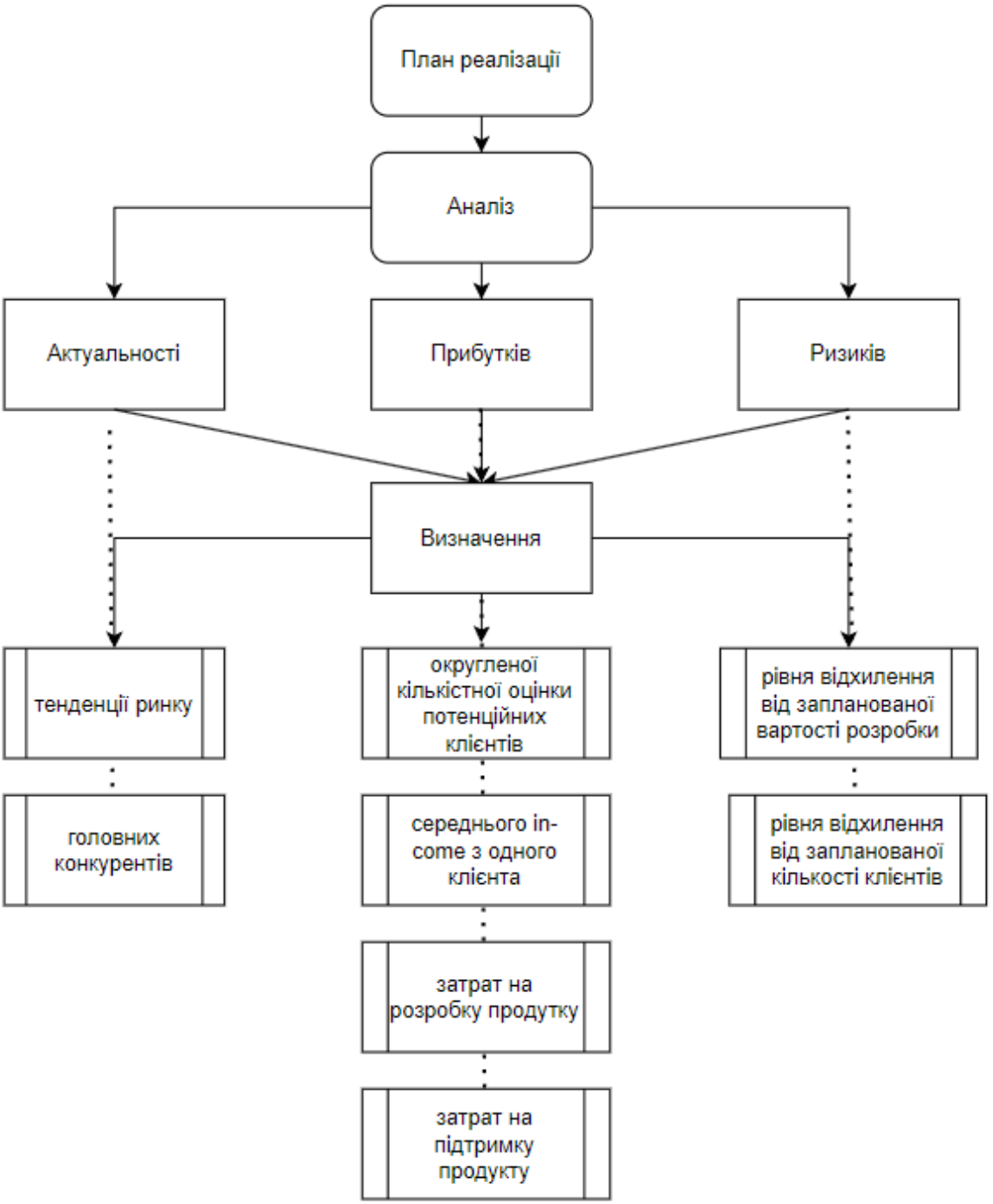

Рисунок 2.1 - План реалізації

Якщо після успішного планування з урахуванням середнього потенційного прибутку та всіх можливих ризиків організація отримує позитивний in-come - можна переходити до наступного етапу реалізації бізнес плану, а саме планування розробки ІС.

Цей етап заключається у створенні технічного плану розробки SaaS платформи, що в свою чергу включає у себе вибір IT компанії розробника та методології розробки (Agile або Waterfall), прорахунок пропорції вартості розробки до швидкості.

Усе має свою ціну, і час, або швидкість - не виключення. Продукт може розроблятися командою із 5 або 50 людей, відповідно сроки готовності продукту можуть варіюватися від декількох місяців до кількох років. На цьому етапі організації треба знайти оптимальний баланс між витраченими коштами та сроками готовності продукту.

Також не можна нехтувати потенційною проблемою втрати актуальності через занадто довгу розробку - якщо максимально зекономити на часі виготовлення ІС - може втратитись актуальність або стек використаних технологій може набути статусу deprecated ще до закінчення проекту, що зробить його maintenance and support збитковим.

Щоб максимізувати прибуток шляхом мінімізації кількості витраченого часу до моменту коли ІС набуде першого прибутку- треба включити у план розробки виготовлення MVP моделі. Наявність такої моделі надасть можливість якомога швидше вийти на ринок то почати отримувати перший in-come. Головною перевагою такої моделі є те, що для її розробки потрібно значно менше часу в порівнянні з повнофункціональною. Але тут треба врахувати що саме має входити до моделі MVP, бо має бути баланс між кількістю функціонала за який клієнти готові платити гроші.

На прикладі нашої ІС електронної бібліотеки - головним функціоналом виступає можливість зберігати, завантажувати та вивантажувати матеріали. Всі інші функціональні особливості можна вважати менш пріоритетними, а саме: функціонал розумного пошуку elasticsearch, редагування профілю, створення підкатегорій в рамках організації (кафедри, відділи, ін.).

Після узгодження ціни, плану та часових рамок розробки повної та MVP моделей ІС із компанією розробником - можна переходити до самої реалізації програмного продукту. Цю роль виконує компанія-розробник, яка вже має чіткий план та часові рамки.

Після успішного завершення розробки MVP моделі - наступним етапом йде запуск моделі у production. На цьому етапі головною метою є розуміння працездатності ІС, її зручності та пошук потенційних недоліків або помилок (bugs). Запуск такої моделі повинен виконуватись для специфічного набору користувачів, які попередньо були відібрані для такого виду тестування.

Також можливий сценарій підписання специфічного контракту з іншою компанією яка зацікавлена в послугах нашої організації та яка готова використовувати наш продукт вже на етапі первинної MVP моделі. Підписання контракту із такої компанією є дуже вигідним кроком, бо вона може предоставити велику базу користувачів, що в свою чергу допоможе максимально швидко та якісно покрити усі потенційні проблеми E2E тестування. Але щоб покрити ризики “сирої” моделі - відповідна ціна наших послуг буде не ринковою, що досить логічно. Таким чином ми, втрачаючи невеликий фінансовий обсяг, повністю покриваємо усі недоліки нашої моделі, після виправлення яких йде повноцінний запуск повної моделі ІС на ринок.

3. ТЕХНІЧНІ СПЕЦИФІКАЦІЇ

3.1 Бізнес-аналіз

Сучасний світ характеризується стрімким розвитком цифрових технологій, які проникають у всі сфери людської діяльності, від освіти до бізнесу. Ця тенденція зокрема впливає на сферу інформаційних систем, де постійно зростає попит на інноваційні та ефективні рішення. В контексті цієї динаміки, розробка електронної бібліотеки як інформаційної системи стає актуальною задачею. Така система відіграватиме важливу роль у забезпеченні доступу до обширного асортименту освітніх та наукових матеріалів, сприяючи навчальному процесу та самоосвіті користувачів.

У сучасному світі неперервного потоку інформації, забезпечення швидкого та зручного доступу до навчальних матеріалів є критично важливим. Електронна бібліотека, як інформаційна система, дозволить користувачам ефективно організовувати та управляти великим обсягом даних, забезпечуючи при цьому легкий доступ до необхідних ресурсів. Така система відіграє роль важливого інструменту у процесі навчання та дослідження, відкриваючи нові можливості для розширення знань та вмінь.

Електронна бібліотека як інформаційна система має численні переваги:

- забезпечення централізованого доступу до широкого спектру освітніх матеріалів;
- швидкий пошук та зручна категоризація матеріалів, що підвищує ефективність навчального процесу.

У даній роботі мова йде про платформу, що займається наданням послуг такого типу:

- реєстрація та авторизація користувачів;
- можливість завантажувати, зберігати та керувати файлами;

- створення та адміністрування організацій;
- публікація та управління приватними та загальнодоступними файлами.

Цілями розробки інформаційної системи є:

- забезпечення ефективного управління та доступу до освітніх ресурсів;
- реалізація зручного інтерфейсу для інтуїтивного пошуку та використання інформації;
- створення безпечного середовища для зберігання та обміну даними;
- підвищення якості навчального процесу через зручний доступ до актуальних матеріалів.

Проте не слід забувати про те, що електрона бібліотека - насамперед представляє собою інформаційну систему. Тому, важливо враховувати ще такі ключові функції та аспекти:

- забезпечення захисту персональних даних користувачів та забезпечення конфіденційності їхньої інформації;
- гарантування стабільної та безперебійної роботи системи, мінімізація часу простою та швидке вирішення будь-яких технічних проблем;
- забезпечення сумісності електронної бібліотеки з іншими інформаційними системами, такими як освітні платформи, наукові бази даних тощо;
- можливість розширення та адаптації системи до зростаючого числа користувачів та збільшення обсягу контенту;
- розробка інтуїтивно зрозумілого та зручного інтерфейсу, що сприяє покращенню взаємодії користувача з системою;
- надання якісної підтримки користувачам, включаючи допомогу у вирішенні проблем, відповіді на запитання та оновлення системи;

3.2 Аналіз обмежень

В рамках проекту з розробки електронної бібліотеки, важливо врахувати різні обмеження, які можуть вплинути на процес реалізації. Нижче наведено аналіз основних аспектів, що мають обмеження у трьох ключових областях: обмеження в часі, обмеженість ресурсів, та об’єм роботи.

Аспект	Обмеження в часі	Обмеженість ресурсів	Об’єм роботи	Заходи
Надійне ПЗ	Високий	Середній	Високий	Планування роботи в ітераціях, використання гнучких методологій, рання інтеграція тестування.
Привабливий дизайн веб-додатку	Високий	Середній	Середній	Працювати з досвідченими дизайнерами, використовувати готові шаблони та бібліотеки дизайну.
Простота веб-додатку	Високий	Високий	Середній	Використання Lean UX принципів, мінімалістичний дизайн, фокус на основні функції.
Забезпечення ясності і легкості навігації	Високий	Середній	Високий	Користувацьке тестування інтерфейсу, врахування зворотного зв’язку від користувачів, адаптивний дизайн.

Чіткий та зрозумілий контент	Середній	Середній	Середній	Робота з професійними копірайтерами, чітке структурування інформації, ефективне використання медіа.
SEO-оптиміз ація	Високий	Високий	Середній	Оптимізація структури сайту, регулярне оновлення контенту, робота з SEO-фахівцями.
Передбачува ність користувача системи	Високий	Середній	Високий	Використання аналітики поведінки користувачів, тестування A/B, створення персоналізованих рекомендацій.
Зворотній зв’язок з користуваче м	Високий	Низький	Високий	Впровадження простих механізмів зворотного зв'язку, використання соціальних медіа, регулярні оновлення на основі зворотного зв'язку. Швидка оплата:

Швидка оплата	Високий	Середній	Середній	Інтеграція з надійними платіжними шлюзами, оптимізація процесу оплати, тестування для забезпечення безпеки.
Терміни виконання	Високий	Високий	Високий	Ефективне планування проекту, пріоритезація завдань, гнучкість у управлінні проектом.
Використання технологій	Середній	Середній	Високий	Вибір перевірених технологій, навчання команди, використання готових рішень.
Серверна інфраструктура	Високий	Середній	Високий	Використання хмарних сервісів, масштабування ресурсів відповідно до потреб, моніторинг продуктивності.
Архітектура системи	Високий	Високий	Високий	Використання модульного підходу, врахування майбутнього розширення, ретельне планування архітектури.

Найбільші обмеження з часової точки зору пов'язані з розробкою надійного ПЗ, забезпеченням ясності навігації, а також розробкою передбачуваної системи для користувачів. Ці аспекти вимагають значних часових вкладень для гарантії якості та ефективності.

Найбільші виклики з точки зору ресурсів стосуються простоти веб-додатку, SEO-оптимізації та термінів виконання. Потрібно враховувати обмеження бюджету, людських та технічних ресурсів.

Високий об’єм роботи очікується в більшості аспектів, особливо в тих, що стосуються надійності ПЗ, ясності навігації та архітектури системи. Це вимагає значної кількості часу та ресурсів для ефективної реалізації.

3.3 Аналіз ризиків

Одним з найбільших ризиків є потенційні хакерські атаки та крадіжка конфіденційних даних. Важливо зосередитися на застосуванні комплексних заходів безпеки для захисту системи та даних користувачів.

Залежність від нових, не повністю перевірених технологій може призвести до затримок та додаткових витрат. Важливо використовувати надійні та перевірені технологічні рішення.

Потрібно звернути увагу на потенційні ризики, пов'язані з розробкою ПЗ, включаючи помилки, конфлікти між плагінами та їх застарілість. Регулярні оновлення та тестування допоможуть знизити ці ризики.

Відсутність ефективної SEO-стратегії може призвести до зниження видимості платформи в інтернеті, що негативно вплине на її популярність та ефективність. Залучення SEO-експертів є критично важливим для забезпечення успіху проекту.

Гнучкість і адаптивність проектного планування є ключовими для управління змінами вимог замовника або ринку.

Регулярне технічне обслуговування та резервне копіювання даних є важливими для забезпечення стабільності роботи системи.

Слід уважно перевіряти всі матеріали на предмет відповідності законодавству про авторські права.

Потрібно планувати архітектуру системи таким чином, щоб вона могла легко масштабуватися зі збільшенням числа користувачів.

Ризик	Оцінка	Наслідки	Методи боротьби
Вразливість інформаційної системи (хакерські атаки)	Висока	Захоплення інформаційних даних, втрата контролю над системою, виведення системи з ладу	Застосування належних заходів захисту: фаєрволи, шифрування, антивірусне ПЗ
Крадіжка конфіденційних даних	Висока	Втрата бази даних користувачів, репутаційні та фінансові втрати	Встановлення системи реєстрації зі складними логіном і паролем, двофакторна аутентифікація, шифрування даних
Ризик помилок при розробці	Середня	Фінансові втрати, затримка запуску	Призначення кваліфікованих розробників, регулярне тестування
Залежність від нових, неперевірених технологій	Висока	Фінансові та часові втрати, втрата продуктивності	Використання перевірених технологій, гнучкий підхід до впровадження нових рішень
Велика кількість плагінів	Середня	Уповільнення роботи системи, конфлікти між плагінами	Обмеження кількості плагінів, ретельний вибір та тестування

Застарілість плагінів	Середня	Некоректне відпрацювання функцій, вразливості у безпеці	Регулярне оновлення плагінів, моніторинг сумісності
Непередбачувані витрати на реалізацію проекту	Середня	Фінансові втрати	Ретельне планування бюджету, врахування резервного фонду
Відсутність SEO-оптимізації	Висока	Втрата конкурентоспроможності, часові та фінансові втрати	Розробка SEO-стратегії, співпраця з SEO-експертами
Зміна вимог до проекту	Середня	Затримки в розробці, збільшення бюджету	Гнучке планування, регулярне спілкування зі стейкхолдерами
Технічні неполадки	Висока	Збої у системі, втрата даних	Резервне копіювання даних, регулярне технічне обслуговування
Відмова від проекту замовником	Середня	Втрата інвестицій та ресурсів	Підписання договору, що визначає умови відмови
Низька якість контенту	Висока	Зниження інтересу користувачів, втрата репутації	Розробка стандартів контенту, контроль якості

Порушення авторських прав	Середня	Юридичні проблеми, штрафи	Перевірка контенту на відповідність авторським правам
Проблеми з масштабуванням системи	Висока	Неможливість впоратися зі збільшенням користувачів	Забезпечення масштабованої архітектури
Труднощі у підборі персоналу	Середня	Затримки в розробці, недостатня якість роботи	Ефективний підбір та навчання персоналу
Недостатнє фінансування	Висока	Припинення або затримка проекту	Детальне планування бюджету, пошук додаткових джерел фінансування

4. СЦЕНАРІЇ КОРИСТУВАЧА

Діаграма сценаріїв користувачів описує роботу інформаційної системи, тобто перелічує її функції або дії, які система повинна виконувати та взаємодію між акторами та компонентами системи. Акторами виступають: користувач, адміністратор.

Варіанти використання інформаційної системи:

1. реєстрація в системі:
    - користувач вводить свої персональні дані (ім'я, електронну адресу, пароль);
    - система перевіряє унікальність даних та надсилає підтвердження електронною поштою;
    - користувач підтверджує реєстрацію, натискаючи на посилання у листі;
2. авторизація в системі;
    - користувач вводить свої дані для входу (електронна адреса та пароль);
    - система перевіряє дані та надає доступ до користувацького інтерфейсу;
3. завантаження файлів на сервер:
    - користувач вибирає файл для завантаження та натискає кнопку "завантажити";
    - система перевіряє формат файлу та розмір, а потім завантажує його на сервер;
4. завантаження файлів із серверу;
    - користувач вибирає файл зі списку доступних та натискає "завантажити";
    - система надає файл для завантаження;
5. видалення файлів із серверу:
    - користувач вибирає файл та натискає "видалити";

- система запитує підтвердження та після отримання відповіді видаляє файл;

6. пошук існуючих файлів;
    - користувач вводить критерії пошуку (наприклад, назву файлу) у пошукове поле;
    - система відображає результати, що відповідають запиту;
7. створення організації:
    - користувач вводить дані організації (назва, адреса тощо) і натискає "створити";
    - система реєструє нову організацію і надає інтерфейс для її керування;
8. керування організацією:
    - адміністратор організації керує користувачами (додає/видаляє користувачів, надає права доступу);
    - система зберігає зміни та надає відповідні права користувачам;
9. зміна тарифного плану:
    - користувач вибирає новий тарифний план та натискає "змінити";
    - система оновлює інформацію про тариф та надає відповідні можливості;
10. модерація завантажуваних файлів:
    - адміністратор переглядає нові файли, перевіряє їх на відповідність стандартам;
    - система надає інструменти для видалення або схвалення файлів;
11. адміністрування платформи;
    - адміністратор керує налаштуваннями системи, перевіряє звіти, реагує на проблеми;

- система надає інструменти для моніторингу, налаштувань та управління системою.

Рисунок 4.1 - Сценарії використання платформи

5. СПЕЦИФІКАЦІЯ ВИМОГ

У цьому розділі подаються функціональні та нефункціональні вимоги до інтернет-сервісу електронної бібліотеки, що спрямований на задоволення потреб різних категорій користувачів.

Основні функції системи:

№	Назва	Опис
1	Реєстрація	Кожен користувач може зареєструватися в свій аккаунт, отримуючи доступ до користування платформою.
2	Авторизація	Зареєстрований користувач може авторизуватись, отримуючи доступ до своєї власної бібліотеки, а також до своїх організацій.
3	Створення та управління організаціями	Кожен користувач може створити свою власну організацію, обмежуючи доступ для всіх інших. Також власник організації може додавати до неї нових користувачів, керувати сховищем організації та завантаженими публікаціями.
4	Завантаження нових публікацій	Кожен користувач може завантажувати свої публікації в особисту бібліотеку або в організацію, членом якої він є.
5	Пошук публікацій	Кожен користувач може скористатися пошуком для знаходження потрібної йому публікації за назвою чи автором.

Функціональні вимоги:

№	Назва	Опис
1	Пошуковий двигун	Система повинна надавати потужний пошуковий механізм для швидкого та точного пошуку наукових матеріалів.
2	Персональні профілі	Користувачі мають можливість створювати власні профілі для зберігання обраних матеріалів.
3	Категорізація матеріалів	Можливість категоризації наукових матеріалів за темами, авторами та іншими параметрами.
4	Доступ до архівів	Можливість доступу до архівів наукових матеріалів.
5	Взаємодія з різними форматами	Підтримка різних форматів файлів для наукових робіт, включаючи PDF, ePub, та інші.
6	Мовна підтримка	Підтримка кількох мов для користування сервісом з урахуванням глобального характеру користувачів.
7	Взаємодія з завантаженими документами	Можливість завантаження документів з платформи на пристрій користувача.

Нефункціональні вимоги:

№	Назва	Опис
1	Продуктивність	Забезпечення швидкості та ефективності роботи системи при великій кількості запитів та користувачів. Максимальна кількість одночасних користувачів: 1000.

		Максимальний час відгуку системи: 2 секунди. Підтримка 95% запитів користувачів за менш ніж 1 секунду.
2	Безпека та Конфіденційність	Гарантування високого рівня захисту особистих даних та конфіденційної інформації користувачів. Шифрування даних: AES 256-біт. Час відновлення системи після безпекового інциденту: максимум 4 години. Проведення безпекових аудитів: щорічно.
3	Надійність	Забезпечення стійкості та надійності роботи Середній час між збоями (MTBF): 10000 годин. Час на відновлення системи після збою (MTTR): максимум 30 хвилин.
4	Мультиплатформенність	Забезпечення доступу до сервісу з різних платформ, включаючи веб-браузер, мобільні телефони та інші пристрої. Сумісність з операційними системами: Windows, macOS, Android, iOS. Підтримка версій веб-браузерів: останні 2 версії Chrome, Firefox, Safari та Edge. Адаптивний дизайн: підтримка екранів з розміром від 4 до 24 дюймів.

5.1 Опис організації

Наша місія полягає у забезпеченні доступу до широкого спектру знань та освітніх ресурсів через інноваційні технології для усіх користувачів, незалежно від їхнього місцезнаходження чи освітнього рівня.

5.1.1 Організаційна Структура

У рамках університетського проекту з розробки електронної бібліотеки, наша команда складається з трьох основних ролей:

1. Керівник Проекту:
    a. Відповідальності:
        - координація команди;
        - визначення основних цілей проекту;
    b. Студент: Гунавардана Широн Сісіра Джанакович
2. Технічний Спеціаліст:
    a. Відповідальності:
        - підтримка та оновлення програмного забезпечення;
        - забезпечення технічної інтеграції та безпеки.
    b. Студент: Левадський Данило Артемович
3. Менеджер з Контенту та Комунікацій:
    a. Відповідальності:
        - організація та каталогізація електронних ресурсів;
        - розвиток контент-стратегії;
        - комунікація з користувачами та рекламна діяльність.
    b. Студент: Чечет Микита Олексійович

Кожен член команди відіграє важливу роль у реалізації проекту, працюючи разом для досягнення спільної мети – створення функціональної та доступної електронної бібліотеки.

5.1.2 Цілі організації

У рамках проєкту, наша організація поставила за мету виконати такі цілі:

1. **Створення Гнучкої Системи Завантаження Контенту**. Надати користувачам можливість завантажувати різноманітні формати файлів, включаючи журнали, документи та наукові статті, для забезпечення широкого спектру освітніх та наукових ресурсів.
2. **Розробка Адаптивних Тарифних Планів**. Забезпечити можливість зміни тарифних планів, що дозволить користувачам керувати обсягом доступного їм простору для зберігання файлів відповідно до їх потреб.
3. **Інтеграція та Управління Організаційними Рахунками**. Розробити функціонал для створення та управління організаціями, дозволяючи їм мати приватний простір для зберігання та обміну файлами.
4. **Поліпшення Системи Авторизації та Безпеки**. Забезпечити надійну систему авторизації для захисту особистих даних користувачів та збереження конфіденційності їхнього контенту.
5. **Розвиток Інтерактивних Взаємодій з Користувачами та Організаціями**. Створити інструменти для ефективної взаємодії між користувачами та організаціями, включаючи функції додавання та видалення учасників організацій.

6. **Розробка Інтуїтивно Зрозумілого Користувацького Інтерфейсу**. Створити легкий у використанні та візуально привабливий інтерфейс, який забезпечить комфортне та продуктивне користування ресурсами бібліотеки.
7. **Оновлення та Розширення Колекції Ресурсів**. Постійно оновлювати та розширювати колекцію доступних матеріалів, забезпечуючи користувачам актуальні та різноманітні освітні ресурси.

5.1.3 Основні служби

Наша електронна бібліотека пропонує наступні ключові послуги:

1. **Завантаження Файлів**. Можливість завантажувати різноманітні типи файлів, включаючи журнали, документи, статті та інші освітні матеріали.
2. **Створення Організацій**. Функціонал створення організацій, який дозволяє зберігати файли в рамках організації, роблячи їх приватними і доступними лише для членів цієї організації.
3. **Авторизація Користувачів**. Система безпечної авторизації для користувачів, забезпечуючи надійний доступ до їхніх облікових записів та ресурсів.
4. **Взаємодія з Організаціями**. Можливість додавати або видаляти користувачів у межах організації, сприяючи ефективному управлінню корпоративними ресурсами та співпраці.

6. ПОБУДОВА АРХІТЕКТУРИ СИСТЕМИ

У рамках проекту розробки платформи електронної бібліотеки розглядаються різні архітектурні рішення, що відповідають сучасним стандартам та потребам користувачів. Кожна архітектура має свої особливості, переваги та недоліки, що впливають на вибір оптимального рішення для реалізації проекту.

6.1 Огляд існуючих архітектур

Монолітна архітектура базується на принципі єдиності усіх компонентів системи. У такій моделі інтерфейс користувача, бізнес-логіка та база даних інтегровані в один застосунок. Це спрощує процес розробки та розгортання, але з часом може ускладнити внесення змін через високу залежність компонентів один від одного. Монолітна архітектура підходить для невеликих та середніх проектів, але має обмеження у масштабованості та гнучкості.

Мікросервісна архітектура включає розподіл системи на окремі незалежні сервіси, кожен з яких виконує певну функцію та може бути розгорнутий незалежно. Це сприяє високій масштабованості та гнучкості, дозволяючи легко вносити зміни та оновлення. Кожен сервіс може бути розроблений з використанням різних технологій, що додає гнучкості у виборі інструментів. Проте, управління мікросервісами може бути складнішим, а також може зрости загальна вартість розробки та підтримки системи.

Клієнт-серверна архітектура базується на взаємодії між клієнтами (наприклад, веб-браузерами) та сервером, який обробляє запити та надає необхідні ресурси. Цей підхід дозволяє розділити відповідальність між клієнтом та сервером, забезпечуючи гнучкість у виборі технологій для

різних частин системи. Однак, це також означає залежність від з'єднання між клієнтом та сервером та можливі обмеження у пропускній здатності сервера.

Хмарна архітектура передбачає використання хмарних сервісів для розгортання та управління ресурсами платформи. Цей підхід дозволяє легко масштабувати систему, забезпечує високу доступність та еластичність у використанні ресурсів. Хмарна архітектура знімає необхідність утримання власної інфраструктури, але при цьому збільшує залежність від провайдера хмарних послуг та може призвести до зростання витрат на обслуговування.

Розглянемо кожну архітектуру більш детально.

6.2.1 Монолітна архітектура

В монолітній архітектурі надійність є високою через її простоту та єдиний кодовий базіс, який мінімізує зовнішні залежності. Однак, ця архітектура часто стикається з проблемами продуктивності при масштабуванні, оскільки всі компоненти тісно взаємопов'язані, що ускладнює швидкість відповіді при збільшенні навантаження. Початкові витрати на розробку та впровадження монолітної архітектури зазвичай нижчі, але довгострокові витрати на оновлення та масштабування можуть зрости. Також, монолітна архітектура має низьку гнучкість, адже внесення змін вимагає роботи з великим об'ємом коду, а масштабування такої системи часто обмежене та складне у реалізації.

6.2.2 Мікросервісна архітектура

Мікросервісна архітектура вважається надійною завдяки ізоляції та незалежності сервісів. Це означає, що збої в одному сервісі рідко

впливають на решту системи. З точки зору продуктивності, мікросервіси вирізняються можливістю масштабування окремих сервісів незалежно один від одного, що сприяє високій загальній продуктивності системи. Економічна ефективність мікросервісної архітектури полягає у легкості масштабування та оновлення, що може зменшити загальні витрати в довгостроковій перспективі. Також, мікросервіси володіють високою гнучкістю, що дозволяє легко вносити зміни та оновлювати окремі частини системи.

6.2.3 Клієнт-серверна архітектура

Клієнт-серверна архітектура забезпечує розділення відповідальності між клієнтом та сервером, що сприяє підвищенню загальної гнучкості та ефективності системи. Цей підхід дозволяє використовувати ресурси сервера більш ефективно, але залежить від якості з'єднання між клієнтом та сервером. Продуктивність клієнт-серверних систем може варіюватися в залежності від навантаження на сервер.

6.2.4 Хмарна архітектура

Хмарна архітектура відрізняється високою масштабованістю та гнучкістю завдяки використанню хмарних ресурсів. Це дозволяє легко адаптуватися до змінних потреб та обсягів навантаження, пропонуючи водночас високу продуктивність. Однак, хмарна архітектура часто вимагає вищих витрат на обслуговування та залежить від стабільності та безпеки хмарного провайдера.

6.3 Порівняльний аналіз існуючих архітектур

У процесі порівняння архітектурних рішень для платформи електронної бібліотеки було визначено кілька ключових критеріїв оцінки. Ці критерії допомагають визначити, як кожна архітектура відповідає конкретним потребам проекту. Основні критерії включають надійність, продуктивність, економічну ефективність, часові показники, гнучкість та масштабованість.

Оцінка кожного критерію для різних архітектурних рішень здійснюється на основі наявності (+) або відсутності (-) певних характеристик.

Критерії	Монолітна Архітектура	Мікросервісна Архітектура	Клієнт-Серверна Архітектура	Хмарна Архітектура
Надійність	+	+	+	+
Продуктивність	-	+	+	+
Економічна Ефективність	+	-	+	-
Часові Показники	-	+	+	+
Гнучкість	-	+	+	+
Масштабованість	-	+	+	+

У процесі порівняння архітектур розглядалися наступні аспекти:

- Надійність - здатність системи функціонувати стабільно та безперебійно протягом тривалого часу. Монолітна архітектура часто

вважається надійною через свою простоту, але комплексність може впливати на стабільність в довгостроковій перспективі.

- Продуктивність - спроможність системи швидко обробляти великі обсяги запитів. Мікросервісні та клієнт-серверні архітектури часто пропонують кращу продуктивність, особливо при масштабуванні.
- Економічна ефективність - загальні витрати на розробку, впровадження та підтримку системи. Монолітні системи можуть бути менш затратними на початкових етапах, але мікросервісна архітектура може пропонувати кращу довгострокову вартість.
- Часові Показники - швидкість реакції системи на запити користувачів. Сучасні архітектури, як мікросервісна та хмарна, забезпечують високу швидкість відгуку.
- Гнучкість - здатність системи адаптуватися до змін у вимогах та умовах. Мікросервісна архітектура вважається найбільш гнучкою, оскільки дозволяє незалежно модифікувати окремі компоненти.
- Масштабованість - можливість збільшення ресурсів та обсягу системи без втрати продуктивності. Мікросервісні та хмарні архітектури вирізняються високою масштабованістю.

6.4 Аналіз ризиків різних архітектурних рішень

При виборі архітектурного рішення для платформи електронної бібліотеки важливо розглянути потенційні ризики, які можуть вплинути на успішність проекту. Кожна архітектура має свої унікальні виклики та потенційні пастки.

У монолітній архітектурі основними ризиками є важкість масштабування та складність внесення змін. Через тісну взаємозалежність компонентів, оновлення однієї частини може призвести до непередбачених

проблем в інших частинах системи. Також, з часом система може стати надто громіздкою та важкою для управління.

Мікросервісна архітектура, хоча й вважається гнучкою та масштабованою, також несе ризики. Основним викликом є складність управління численними окремими сервісами, особливо у випадках їх залежності один від одного. Це може призвести до проблем із сумісністю та потреби у координації між командами, що відповідають за різні сервіси.

Клієнт-серверна архітектура несе в собі ризик залежності від стабільності та пропускної здатності сервера. Якщо сервер стає перевантаженим або зазнає збою, це може призвести до зниження доступності та продуктивності всієї системи. Також, безпека даних може стати проблемою, оскільки всі дані централізовано зберігаються на сервері.

Хмарна архітектура, хоча й пропонує високу масштабованість та еластичність, має потенційний ризик залежності від хмарного провайдера. Це означає, що проблеми з доступністю або безпекою на стороні провайдера можуть мати безпосередній вплив на роботу електронної бібліотеки. Також варто враховувати потенційні фінансові ризики, пов'язані зі зміною тарифів або політики провайдера.

6.5 Обгрунтування остаточного вибору архітектури

Мікросервісна архітектура виокремлюється як найбільш привабливий варіант для платформи електронної бібліотеки, забезпечуючи високу гнучкість та масштабованість, які є критично важливими для сучасних програмних систем. Ця архітектура також забезпечує високу продуктивність, що є ключовим для обробки великих обсягів запитів та даних.

Надійність мікросервісної архітектури забезпечується через ізоляцію та незалежність окремих сервісів, що дозволяє мінімізувати вплив можливих збоїв окремих компонентів на роботу всієї системи. Економічна ефективність такої моделі полягає у можливості оптимізації ресурсів та витрат на кожен сервіс окремо, а також у зменшенні загальних витрат на масштабування та оновлення системи.

Враховуючи ці аспекти, мікросервісна архітектура виглядає як найбільш підходящий варіант для розробки ефективної, гнучкої та масштабованої платформи електронної бібліотеки.

7. АРХІТЕКТУРА СИСТЕМИ

7.1. Компоненти та опис архітектурної системи

Система буде основана на REST комунікації між клієнтом та сервером.

Система буде розділена на два репозиторії, а саме:

1. front-end (клієнт)
2. back-end (сервер)

7.2 Характеристика архітектури

SaaS-бібліотечна ІС, яка розглядається в цій роботі, буде розроблена з використанням наступних технологій:

- Angular - фреймворк для розробки веб-додатків на основі TypeScript.
- NodeJS - фреймворк для розробки веб-додатків на основі JavaScript.
- PostgresQL - реляційна база даних.
- Amazon S3 Bucket - файлове сховище.

Ці технології дозволяють створити сучасну і гнучку SaaS-бібліотечну ІС, яка відповідає вимогам, які були описані в розділі "Опис проблеми".

7.2.1 Архітектура клієнтської частини

Клієнтська частина ІС буде розроблена з використанням Angular. Angular - це фреймворк для розробки веб-додатків на основі TypeScript. Він забезпечує широкий спектр функцій, які необхідні для створення сучасних і динамічних веб-додатків.

Angular-додаток буде розділений на наступні компоненти:

Головний компонент - це компонент, який відповідає за відображення основного інтерфейсу користувача.

Компоненти користувача - це компоненти, які відповідають за відображення окремих сторінок або форм.

Компоненти бібліотек - це компоненти, які відповідають за доступ до даних бібліотеки.

7.2.2 Архітектура серверної частини

Серверна частина ІС буде розроблена з використанням NodeJS. NodeJS - це фреймворк для розробки веб-додатків на основі JavaScript. Він забезпечує високу продуктивність і масштабованість, що є важливими факторами для SaaS-додатків.

Серверна частина ІС являє собою мікросервісний NodeJS застосунок, який складається з кількох модулів. Найважливішими модулями є:

- User Management Service. Цей модуль відповідає за повну взаємодію з користувачами, включаючи створення нового аккаунту, аутентифікацію, створення та керування організаціями тощо.
- Article Management Service. Модуль, що відповідає за взаємодію із публікаціями. Серед його функціоналу можна виділити завантаження файлу на сервер, завантаження файлу на пристрій користувача тощо.
- Authentication Service(RBAC service). Відповідає за обмеження прав користувача для доступу к різному функціоналу організацій в залежності від їхнього рівня доступу.

Взаємодія між мікросервісами в застосунку буде виконана за рахунок послідовного виклику необходніх ендпоїнтів за допомогою HTTP протоколу та передачі необхідних метаданих у заголовках запитів.

7.2.3 Архітектура бази даних

Дані ІС будуть зберігатися в реляційній базі даних PostgresQL. PostgresSQL - це потужна і надійні база даних, яка забезпечує високий рівень безпеки та масштабованості.

База даних буде включати наступні таблиці:

- користувачі(Users) - таблиця, яка зберігає інформацію про користувачів ІС;
- бібліотеки(Library) - таблиця, яка зберігає інформацію про бібліотеки(організації) у системі;
- публікації(Publication) – таблиця, яка зберігає детальну інформацію щодо певної публікації.
- теги(Publication_tags) - таблиця, яка зберігає інформацію про теги певної публікації, необхідні для пошуку та фільтрації робіт.
- ролі(Roles) - таблиця, що містить інформацію щодо ролей користувачів, необхідних для обмеження різного функціоналу в залежності від рівня доступу
- RBAC(roled-based access control) - зв’язуюча таблиця між користувачем, організацією(бібліотекою) та ролями, яка відповідає за надання кожному користувачу окремого рівня доступу для різних організацій

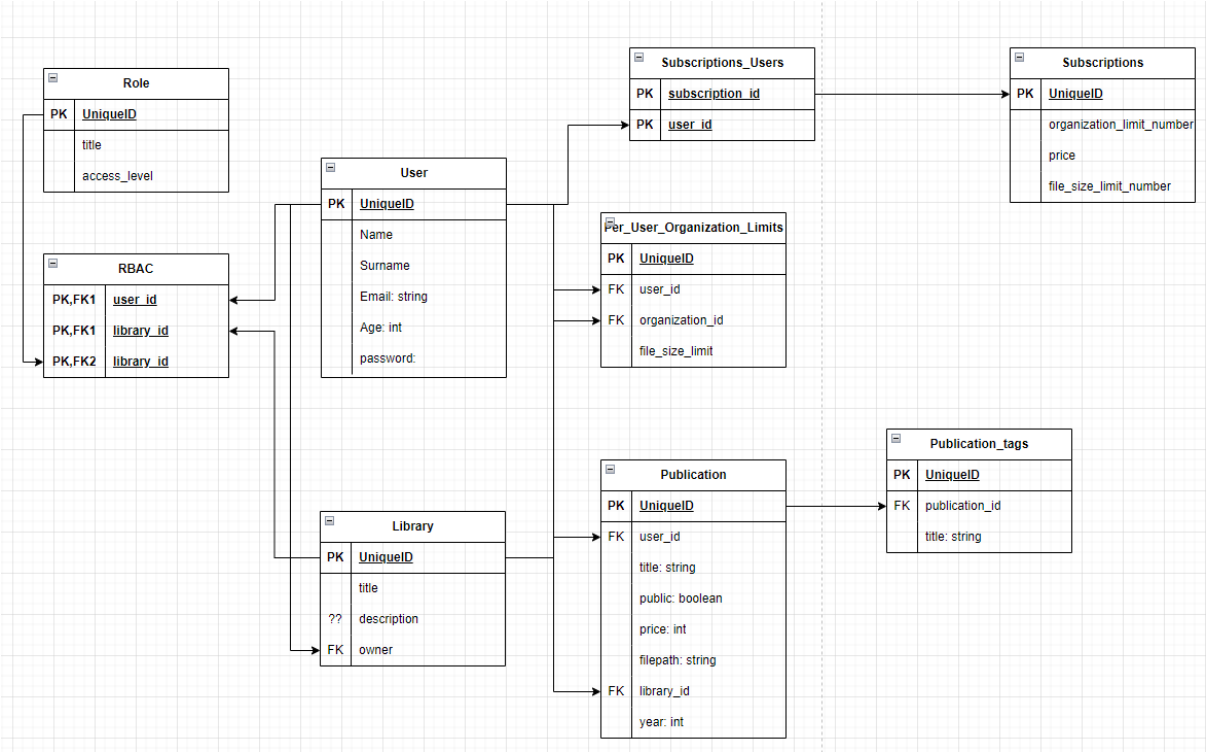

Рисунок 7.1 – ER діаграма бази даних ІС

7.2.4 Взаємодія між компонентами

Клієнтська частина ІС буде взаємодіяти з серверною частиною за допомогою REST API. REST API - це стандартний спосіб взаємодії між веб-додатками. Він забезпечує простий і зрозумілий спосіб обміну даними між клієнтською та серверною частинами.

Серверна частина ІС буде взаємодіяти з базою даних за допомогою драйвера PostgresQL. Драйвер PostgresQL - це програмне забезпечення, яке забезпечує доступ до бази даних PostgresQL з програмного коду.

Також серверна частина ІС взаємодіє з файловим сховищем S3 Bucket, який використовується для зберігання файлів із публікаціями.

Така трирівнева архітектура має назву **FANP**. Обрана архітектура є одним із модифікованою версією стеку **MEAN**, що дуже часто використовується для розробки веб-додатків. У нашому випадку, MongoDB та Express були замінені на PostgreSQL та Fastify відповідно.

Таким чином, **FANP** можна розшифрувати як Fastify, Angular, NodeJS та PostgreSQL.

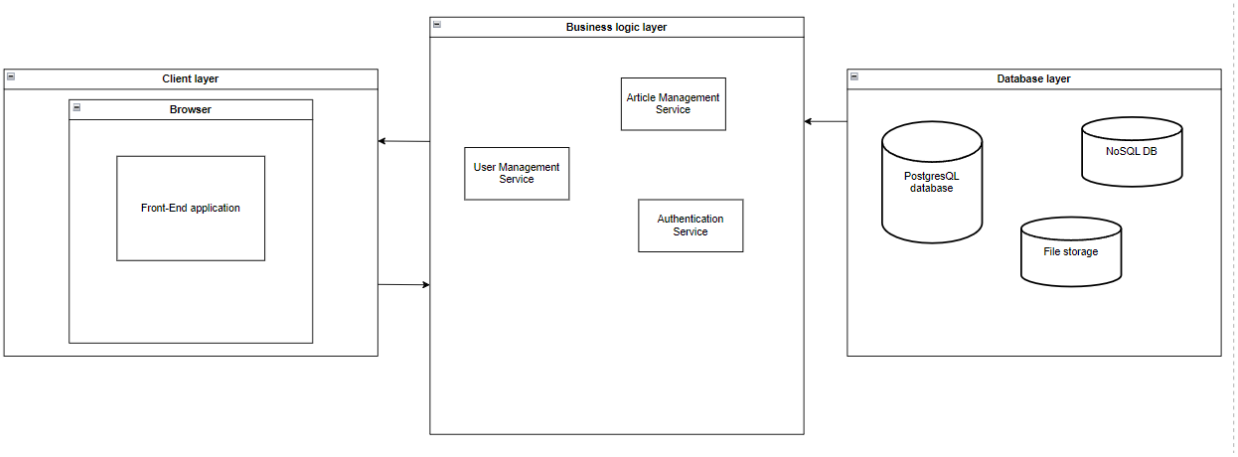

Рисунок 7.2 - Тришарова структура інформаційної системи

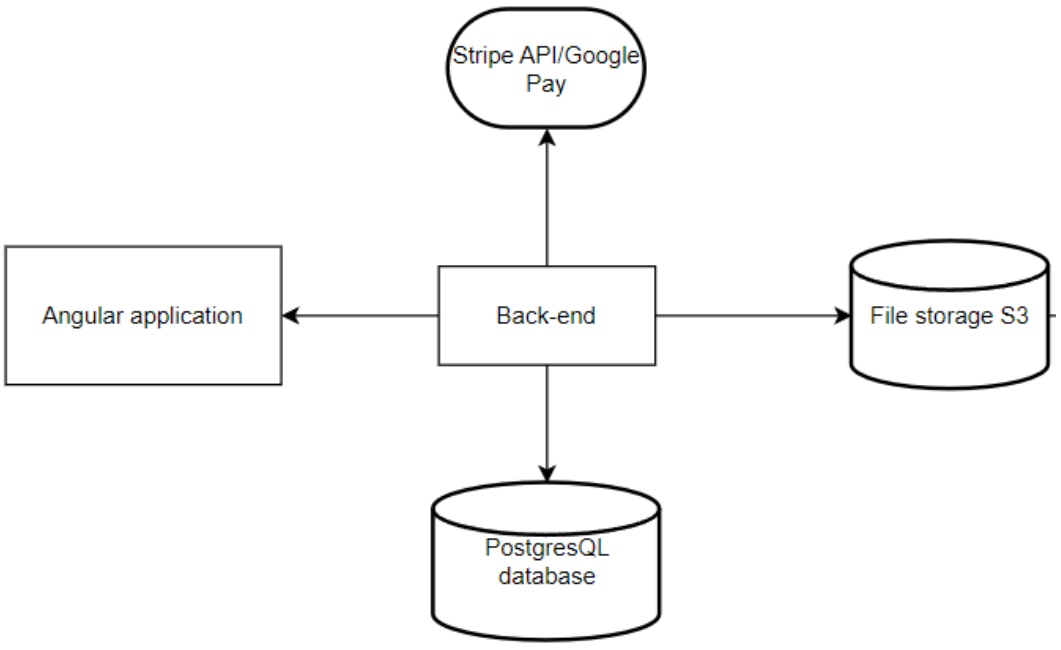

Рисунок 7.3 - Архітектура додатку

7.2.5 Спосіб зберігання файлів

Для спрощення логіки зберігання файлів, було обрано створити дворівневу файлову систему. Перший рівень - рівень звичайних авторів, тобто саме там зберігаються усі персональні публікації кожного користувача. Він може змінювати доступ до них, але вони будуть зберігатись саме там.

З метою ефективної роботи системи, папка створюється лише із першою публікацією користувача і видаляється, якщо у ній не залишилось файлів. Таким чином, система захищена від великої кількості пустих файлових дерев.

Другий рівень - рівень організацій. Під час створення організації, у файловій системи створюється папка, у якій будуть зберігатись загальні публікації самої організації, а також вкладена папка, яка буде містити публікації окремих користувачів.

Важлива різниця між загальними публікаціями та публікаціями окремих авторів у тому, що перші будуть бачити абсолютно всі учасники групи, а другі - будуть доступні лише їх авторам. Таким чином, лише адміністратор організації та сам автор вирішують, чи дозволяти іншим учасникам переглядати цей файл чи ні. На основі побудованого опису, можна схематично побудувати файлову структуру платформи.

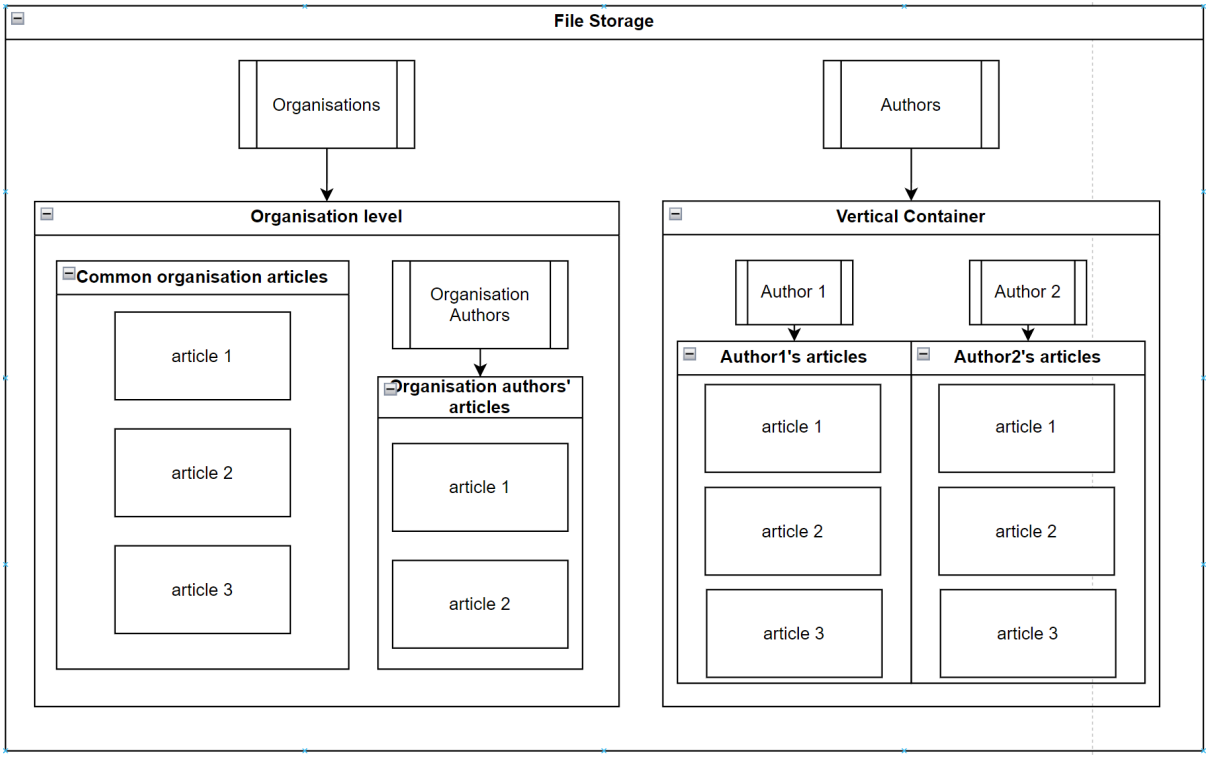

Рисунок 7.4 - Файлова структура платформи

7.3 Технології та інструменти

7.3.1 Архітектура Front-end

Для написання клієнтської частини застосунку було обрано популярний фреймворк Angular та мову програмування Typescript. Основні переваги Angular включають модульність, розширюваність, інструменти для роботи зі станом додатку.

Angular використовує компонентний підхід до розробки, де кожен елемент інтерфейсу (компонент) є незалежним та може бути легко перевикористаним. Це спрощує розробку складних інтерфейсів та полегшує їхнє тестування.

Фреймворк також надає великий набір інструментів для роботи з HTTP-запитами, формами, анімаціями та іншими аспектами фронтенд-розробки. Angular також підтримує двостороннє зв'язування даних, що робить автоматизацію оновлення інтерфейсу на основі змін у даних.

Typescript - це мова програмування, що є надмножиною JavaScript, і додає до неї строгу типізацію. Використання Typescript в проекті Angular дозволяє виявляти та виправляти помилки на етапі розробки, полегшуючи рефакторінг та забезпечуючи більшу стабільність коду.

Typescript також надає розширений функціонал для розробки, такий як робота з класами, інтерфейсами та модулями, що допомагає структурувати та організовувати код. Компіляція Typescript у JavaScript забезпечує сумісність з усіма сучасними браузерами та середовищами виконання.

Angular та Typescript в поєднанні створюють потужну платформу для розробки користувацьких інтерфейсів електронної бібліотеки, яка є якісною, швидкою та масштабованою.

Структура front-end буде наступною:

- assets - глобальні статичні активи, такі як зображення, svgs, логотип компанії тощо.
- core - головний модуль, який включає у себе функціонал авторизації, обробники статусу користувача, API сервіси, обробник помилок.
- shared - модуль що містить функціонал що використовується іншими окремими feature модулями
- feature modules - модулі що відповідають за окремий функціонал платформи
- utils - утиліти, що складаються з окремих допоміжних функцій
- main.ts - точка входу в застосунок

Рисунок 7.5 - Структура “Core” модулю

Рисунок 7.6 - Структура “Shared” модулю

7.3.2 Архітектура Back-end

Для написання серверної частини було обрано фреймворк Fastify. Його основні переваги включають високу продуктивність завдяки розумному підходу до обробки HTTP-запитів та оптимізованим обробникам маршрутів.

Fastify створений з урахуванням принципу "не перевантажуй", спрощуючи розробку та підтримку великих та складних проектів. Він підтримує асинхронність, дозволяючи ефективно обробляти багато запитів одночасно, що робить його ідеальним вибором для високонавантажених додатків.

Fastify пропонує велику кількість вбудованих плагінів, які полегшують роботу з логуванням, автентифікацією, валідацією та іншими завданнями. Його легкість використання та розширення дозволяє швидко створювати надійні та ефективні серверні застосунки.

Fastify ідеально підходить для проекту електронної бібліотеки, оскільки він забезпечує швидкість, низьку затрату ресурсів та зручний інтерфейс для розробки серверної логіки.

Для початку, ми заклали фундамент загальної технічної реалізації платформи - окремий темплейт, який у майбутньому можна перевикористати для розробки кожного мікросервісу. Для цього, наша команда у межах організації GitHub створили окремий репозиторій. Створили необхідну базову файлову та технічну структуру, встановили усі необхідні залежності та додали усі потрібні конфігурації й налаштунки засобів розробки, зокрема - лінтеру, хуків та TypeScript.

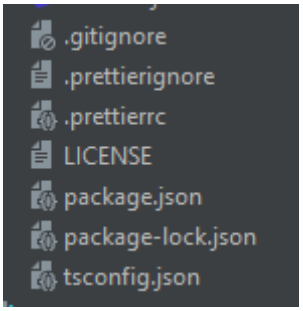

Рисунок 7.7 - Конфігураційні файли шаблону

Далі, ми побудували файлову структуру майбутніх мікросервісів. Вона розташована у кореневій папці src, яка поділяється на такі піддиректорії:

1. config - директорія, у якій зберігаються змінні середовища у папці env, типізація для зручного використання у межах проєкту самого конфігу, а також експортується сам конфіг в index.ts файлі. Так як майже усі дані, що знаходяться у директорії .env є секретами, тому вся директорія знаходиться у .gitignore файлі. Але дана директорія частіше використовується для локальної розробки, адже на сервері можно налаштувати змінні середовища та посилатись у конфігі на змінні середовища напряму, без використання секретних значень.
2. db - тут зберігається механізм, що відповідає за взаємодію із базою даних. Так само, наявна директорія config, у якій ми створюємо базове підключення до БД, потім

перевикористовуємо у файлі DBService.ts, для того, щоб обернути у необхідну технічну реалізацію.

3. modules - власне папка, у якій зберігається більша частина бізнес логіки. У темплейті наявний лише модуль utils, у якому зберігається RBAC частина необхідного сервісу.
4. server - остання важлива складова кожного мікросервісу. Власне тут буде створюватись веб-сервер, на який будуть іти усі необхідні запити. Наявні піддиректорії common, у якій зберігаються різні технічні файли, пов'язані із серверної реалізацією та logger - вже створений та налаштований логер системи на базі популярного пакету pino.

Після того, як було розроблено базову структуру кожного мікросервісу, необхідно мати можливість перевикористати увесь розроблений функціонал. GitHub надає можливість помічати репозиторії як "шаблонні" та використовувати їх при створенні нового репозиторію. Таким чином, не буде потреби щось переносити або копіювати. Достатньо лише обрати у налаштуваннях самого репозиторію пункт "Template repository".

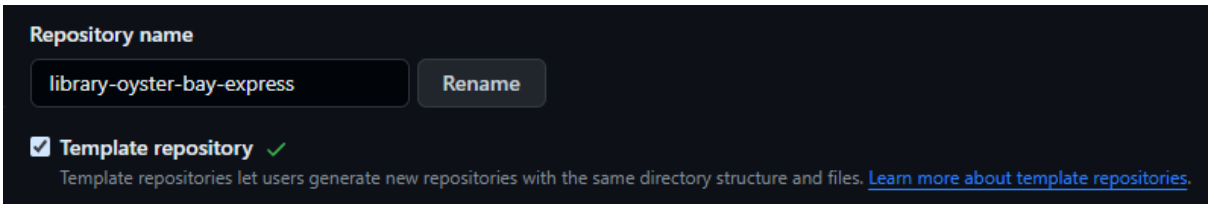

Рисунок 7.8 - Приклад налаштованого репозиторію-шаблону

Далі, при створенні нового репозиторію, достатньо буде лише обрати його у випадаючому списку існуючих шаблонів.

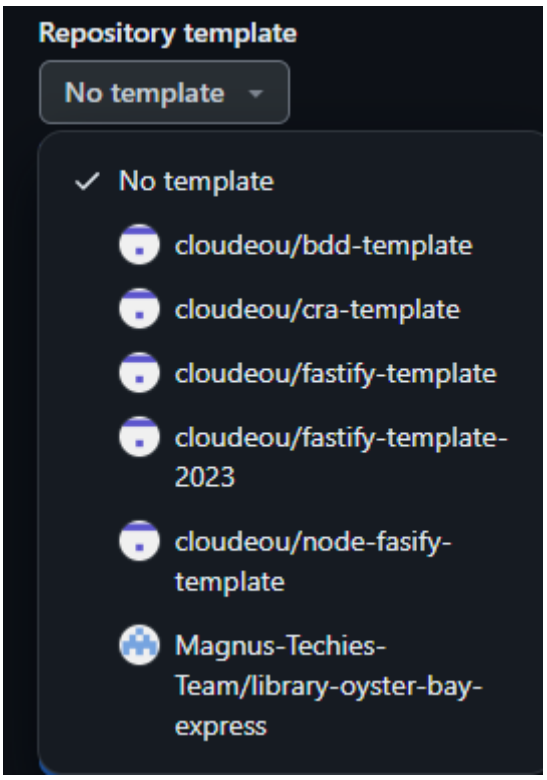

Рисунок 7.9 - Репозиторії-шаблони

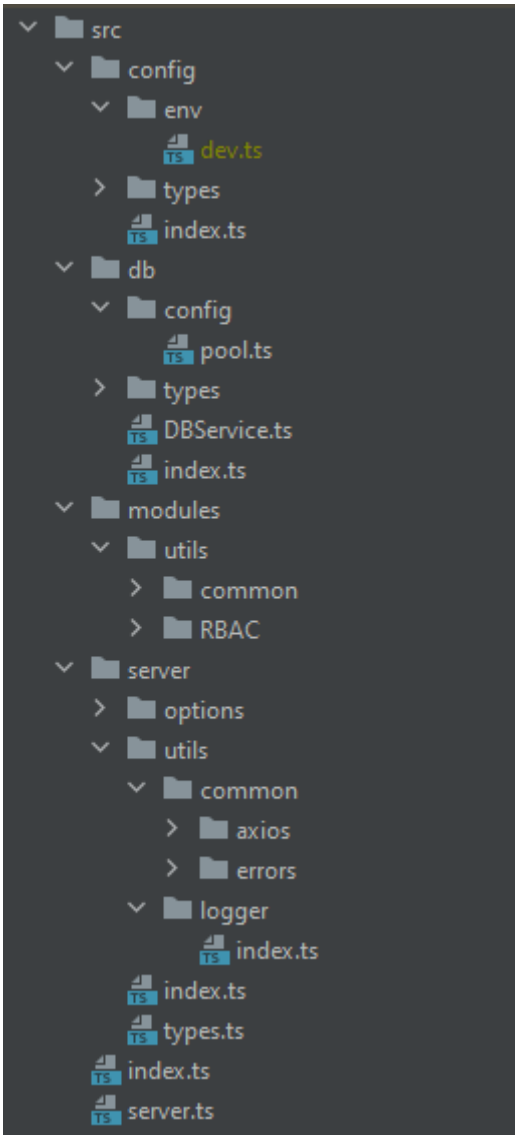

Рисунок 7.10 - Файлова структура папки src

7.3.3 База Даних

Як драйвер баз даних було вирішено використовувати PostgreSQL. PostgreSQL є високоякісною та потужною системою управління базами даних (СУБД), яка добре відома своєю надійністю, масштабованістю та здатністю працювати з великим обсягом даних. Вона входить до родини реляційних баз даних та забезпечує повний набір функцій для зберігання та опрацювання структурованої інформації.

Основні Особливості PostgreSQL:

- Реляційна Структура Даних: PostgreSQL використовує реляційну модель даних, що дозволяє зберігати інформацію у вигляді таблиць зі зв'язками між ними, що спрощує організацію та отримання даних.
- Надійність та Відновлення: Система має високий рівень надійності, забезпечуючи механізми резервного копіювання та відновлення для збереження даних в найкращому стані.
- Підтримка Транзакцій: PostgreSQL підтримує ACID (Atomicity, Consistency, Isolation, Durability) властивості транзакцій, що гарантує цілісність даних під час виконання операцій.
- Розширені Типи Даних: Він має багатий набір вбудованих та користувацьких типів даних, що дозволяє зберігати та опрацьовувати різноманітну інформацію.
- Масштабованість: PostgreSQL підтримує роботу з великою кількістю одночасних з'єднань та може ефективно працювати з обсягами даних різного розміру.
- Розширені Можливості Запитів: Запити можуть бути складні та включати в себе різноманітні операції, такі як об'єднання, групування, сортування тощо.
- Відкритий Код та Активна Спільнота: PostgreSQL є відкритим програмним забезпеченням, що означає, що ви можете змінювати його за своїми потребами. Також існує активна спільнота розробників та користувачів, яка надає підтримку та розвиває цю систему.

PostgreSQL є ідеальним вибором для електронної бібліотеки, оскільки вона дозволяє зберігати, організовувати та отримувати доступ до великих обсягів структурованої інформації у надійний та ефективний спосіб.

7.3.4 Зберігання Об'єктів

Amazon S3 (Simple Storage Service) є послугою зберігання об'єктів у хмарному сервісі Amazon Web Services (AWS). Вона надає високий рівень доступності, масштабованості та безпеки для зберігання та управління різнорідними об'єктами, такими як тексти, зображення, аудіо- та відеофайли, що використовуються в електронних бібліотеках.

Основні Особливості Amazon S3 Bucket:

- Масштабованість: S3 дозволяє зберігати петабайти даних, забезпечуючи високий ступінь масштабованості. Це робить його ідеальним вибором для проектів, які можуть зростати та змінюватися з часом.
- Доступність та Надійність: Amazon S3 гарантує високий рівень доступності, що дозволяє забезпечити неперервний доступ до об'єктів. Дані автоматично реплікуються на різних серверах та регіонах для надійності.
- Безпека Даних: S3 забезпечує різні рівні захисту даних, включаючи шифрування в покої та під час передачі, аутентифікацію користувачів та керування доступом.
- Зручний Інтерфейс: AWS надає інтуїтивно зрозумілий інтерфейс для управління та взаємодії з об'єктами в S3 Bucket. Це включає веб-консоль, інтерфейс командного рядка та програмний інтерфейс (API).
- Повідомлення та Події: S3 дозволяє налаштовувати події, такі як завантаження або видалення файлів, та сповіщати про них через інші служби AWS або HTTP-запити.
- Оплата за Використання: Система тарифів на основі споживаної місткості дозволяє економічно використовувати ресурси відповідно до реальних потреб проекту.

Amazon S3 є надійним та ефективним засобом зберігання об'єктів для електронних бібліотек, де важлива висока доступність, масштабованість та безпека даних.

8. ОНОВЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У світі швидко змінюючихся технологій та постійно зростаючих загроз кібербезпеки, регулярне оновлення програмного забезпечення та використання останніх патчів та технологій безпеки стає важливою складовою для забезпечення безпеки та надійності електронної бібліотеки.

Система електронної бібліотеки буде піддаватися регулярним оновленням, спрямованим на поліпшення функціональності, виправлення помилок та забезпечення сумісності з останніми технологічними стандартами. Оновлення можуть включати:

- Нові Функції та Можливості: Додавання нових функцій, які покращують користувацький досвід та розширюють функціональність системи.
- Оптимізація Продуктивності: Вдосконалення швидкодії та ефективності роботи системи для забезпечення максимальної продуктивності.
- Інтерфейс та Дизайн: Вдосконалення користувацького інтерфейсу та дизайну для забезпечення зручності використання.

8.1 Застосування Останніх Патчів та Технологій Безпеки:

Забезпечення безпеки системи є важливим пріоритетом, тому електронна бібліотека буде вживати ряд заходів щодо вдосконалення безпеки:

- Встановлення останніх патчів: Система буде регулярно оновлюватися з використанням останніх патчів для виправлення виявлених уразливостей та багів та забезпечення захисту від потенційних атак.

- Моніторинг Загроз: Впровадження систем моніторингу, які вчасно виявлятимуть підозрілу активність та спрямовані на попередження можливих загроз.
- Шифрування та Захист Даних: Застосування сучасних методів шифрування та захисту даних для забезпечення конфіденційності та цілісності інформації.
- Аудит Безпеки: Регулярне проведення аудитів безпеки для виявлення та усунення можливих слабких місць системи.

Оновлення та забезпечення безпеки програмного забезпечення є невід'ємною частиною стратегії експлуатації електронної бібліотеки, щоб гарантувати її стійкість та відповідність вимогам сучасного інформаційного середовища.

9. ВИХІДНИЙ КОД ДОДАТКУ

9.1 Реалізація клієнтської частини

9.1.1 Дизайн інтерфейсу

Важливо сказати, що дизайн програмного застосунку був не основною метою розробки. Більшість часу було витрачено на розробку грамотної та оптимізованої системно-архітектурної частини застосунку, більша частина якої прихована від end-користувача. Вона полягає у правильній організації застосунку таким чином, щоб подальша підтримка та масштабованість проекту розроблялась максимально швидко та просто.

9.1.2 Прикладна частина

Клієнтська частина буде реалізована на основі фреймворку Angular 17 версії за допомогою Redux & Reactive бібліотек NgRx & RxJS.

Дана система буде включати у себе два окремих модулі, де перший – це front-end застосунок, який представляє сам інтерфейс онлайн магазину із можливістю придбання товарів, та другий – це модуль самого алгоритму рекомендації, який буде включений до застосунку

Front-end застосунок буде розділений на окремі модулі, такі як shared, core та app.

Головна мета shared модулю це групування усіх компонентів та директив що перевикористовуються у всьому застосунку. Такий підхід чітко відповідає означенню патерна DRY (don`t repeat yourself), бо весь функціонал що перевикористовується буде написаний лише один раз та не буде повторюватися. Структура shared модулю зображена на рисунку 9.1.

Рисунок 9.1 – Структура shared модулю

Core модуль був створений для інкапсуляції усієї базової логіки що потрібна для коректної роботи застосунку, наприклад API (Application Programming Interface) сервіси, які відповідають за взаємодію із сервером, інтерсептори (англ. Interceptors), які є посередниками між відправкою та отриманням запитів з та до сервера. Структура модулю зображена на рисунку 9.2.

Рисунок 9.2 – Структура core модулю

App модуль в свою чергу включає у себе усі окремі компоненти застосунку, такі як сторінки, приклад яких можна побачити на рисунку 9.3. Вони ж в свою чергу також входять до окремих модулів для того щоб реалізувати патерн лінивого підвантаження (lazy-loading). Суть даного підходу полягає у підвантажуванні та обробці тільки тих складових застосунку, до яких безпосередньо перейшов користувач. Така реалізація дозволяє суттєво зменшити час першого завантаження застосунку, бо замість обробки усього проекту буде підвантажена лише менша частина, потрібна лише для початкової сторінки.

Рис. 9.3 – Структура базового сторінкового модулю

Структура даних у проєкті буде реалізована на основі сучасного патерну реактивного програмування (reactive programming), який включає у себе використання окремої бібліотеки. У такому підході усі дані які прив'язані до конретних сторінок або модулів будуть згруповані у відповідні об'єкти (states). Діаграма роботи даного підходу зображена на рисунку 2.4.

Для двонапрямленого доступу до даних будуть використуватися спеціальні об’єкти actions, які інкапсулюють у собі спеціальні команди для взаємодії із об'єктом даних. Такий підхід є реалізацією патерну команда (command).

За модифікування об'єктів даних відповідають спеціальні функції (reducers), які містять логіку модифікації даних на основі вхідних команд (actions). Така реалізація є прикладом використання патерну State.

В більшості випадків об’єкти даних можуть буди доволі місткими, включаючи у себе багато інших внутрішніх об’єктів, що може стати проблемою у роботі з ними. Саме для цього використовуються спеціальні

вибіркові функції (selectors), які повертають конкретні частини складеного об’єкту, що значно спрощує роботу із даними.

За реалізацію побічних ефектів застосунку, таких як відправка запитів на сервер, будуть використані спеціальні сервіси які називаються ефектами (effects). Ці сервіси підписуються на усі вхідні команди (actions) та виконують відповідну логіку залежачи від команди. За такий функціонал відповідає патерн споглядач (observer).

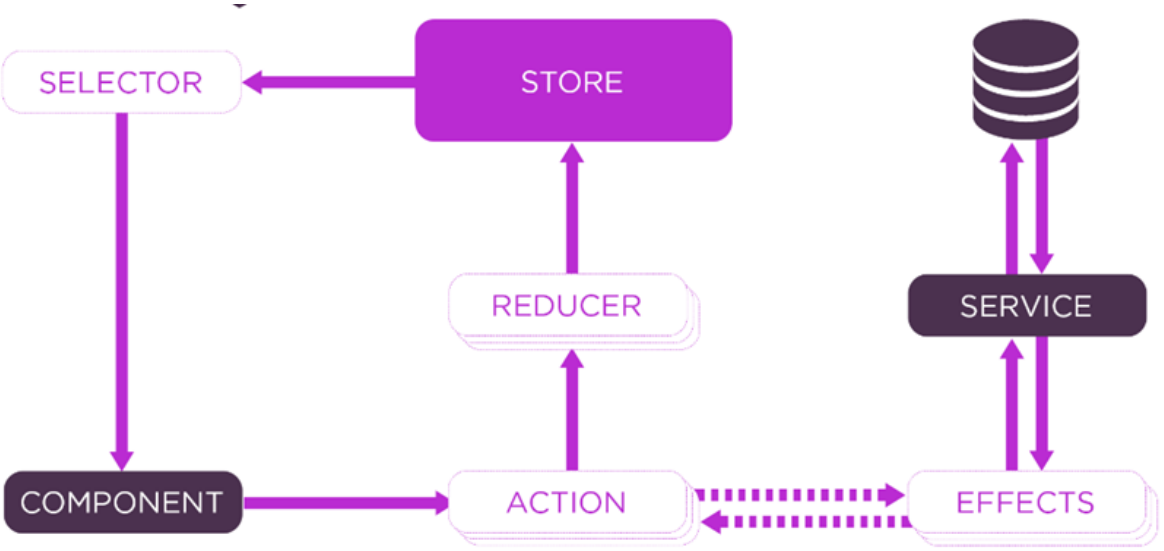

Рисунок 9.4 – Структура взаємодії усіх компонентів NgRx

Також структура окремих сторінкових модулів буде включати у себе структурний патерн контейнерних та презентаційних компонентів (container and presentational components), який полягає у сепарації складових на контейнерні компоненти, які містять у собі всю логіку сторінки, та презентаційних, які відповідають лише за представлення даних. Такий розподіл значно підвищує читабельність коду та допомагає краще масштабувати застосунок.

За презентативну стилістичну основу застосунку буде використаний препроцесор стилів SASS (Syntactically Awesome Style Sheets), який є надбудовою над стандартною мовою стилів CSS (Cascading Style Sheets). Даний препроцесор дозволяє не тільки значно швидше розробляти продукт

за допомогою вбудованого функціоналу, а й значно збільшити розуміння та читабельність коду.

Як один із пунктів реалізації патерну прогресивного веб застосунку (PWA) буде реалізований механізм service worker, який дозволяє кешувати дані та запити на сервер та реалізує функціонал режиму без доступу до мережі інтернет (offlane mode). Така функціональність є досить ефективною, бо допомагає покращити досвід користувача (user experience) шляхом підвищення швидкості обробки даних. Головний файлом зберігання Actions застосунку буде виглядати наступним чином.

```
3 usages  OtecValerian
export const getUser : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Get User'))
2 usages  OtecValerian
export const getUserSuccess : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Get User Success'), props<{user: User}>())
2 usages  OtecValerian
export const getUserFailure : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Get User Failure'), props<{error: HttpErrorResponse}>())

3 usages  OtecValerian
export const signIn : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Sign In'), props<{request: SignInRequest}>())
3 usages  OtecValerian
export const signInSuccess : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Sign In Success'), props<{user: User}>())
2 usages  OtecValerian
export const signInFailure : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Sign In Failure'), props<{error: HttpErrorResponse}>())

3 usages  OtecValerian
export const signUp : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Sign Up'), props<{request: SignUpRequest}>())
3 usages  OtecValerian
export const signUpSuccess : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Sign Up Success'), props<{user: User}>())
2 usages  OtecValerian
export const signUpFailure : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Sign Up Failure'), props<{error: HttpErrorResponse}>())

2 usages  OtecValerian
export const signOut : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Sign Out'))
3 usages  OtecValerian
export const signOutSuccess : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Sign Out Success'))
1 usage  OtecValerian
export const signOutFailure : ActionCreator<string, function... = createAction(storeUtil.getActionType(FEATURE_KEY, desc: 'Sign Out Failure'), props<{error: HttpErrorResponse}>())
```

Рисунок 9.5 – Структура взаємодії Actions

Effects функції будуть інкапсульовані у головному модулю та будуть спільними для всіх окремих feature - модулів.

Рисунок 9.6 – Структура взаємодії Effects

Для відокремлення публічного інтерфейсу усієї глобальної NgRx структури - буде використаний патерн Facade, за допомогою якого будуть реалізовані увесь функціонал що є у публічному доступі для окремих feature-модулів.

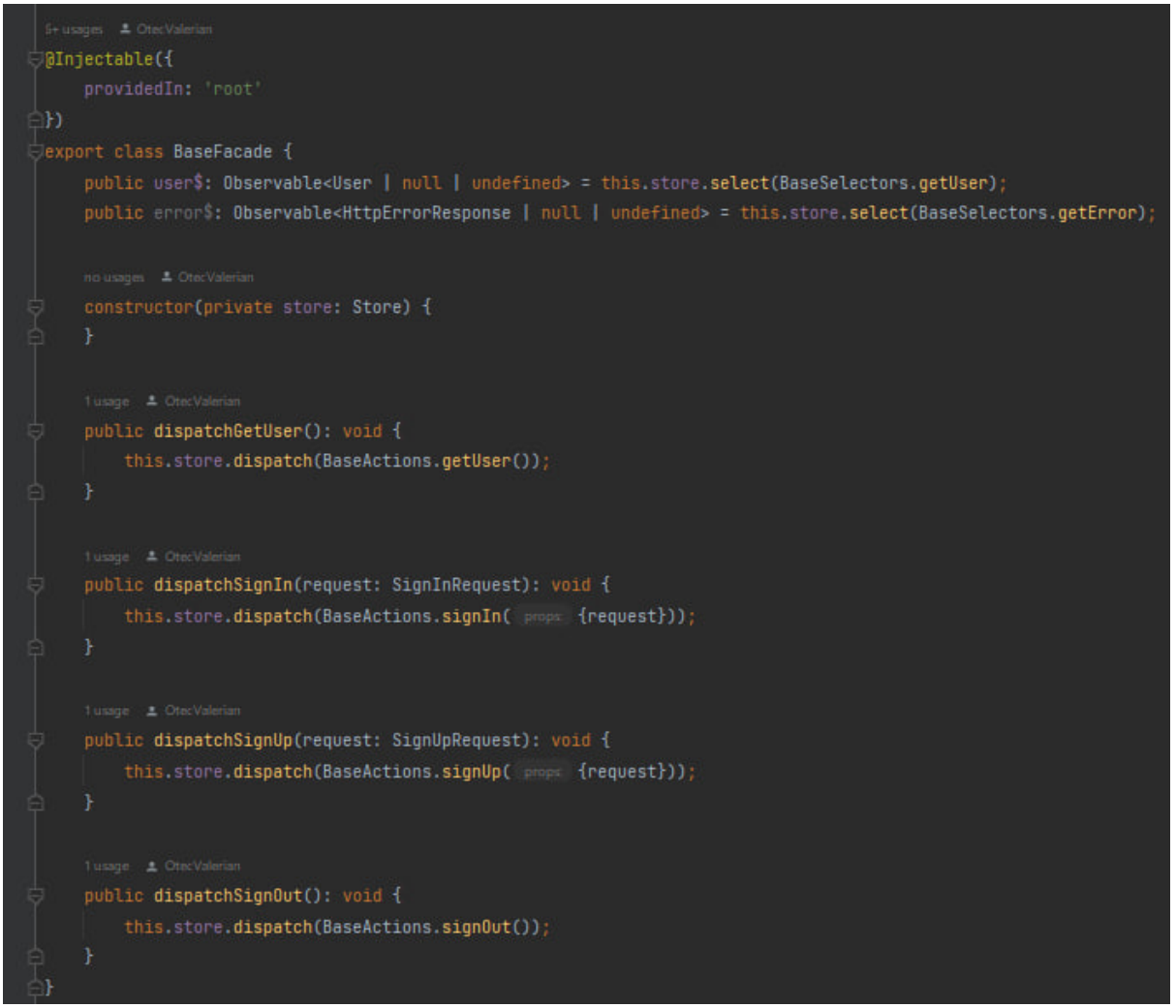

Рисунок 9.7 – Структура взаємодії Facades

Рисунок 9.8 – Структура взаємодії Reducers

9.2 Реалізація серверної частини

9.2.1 Авторизація та аутентифікація

Правильно написані авторизація та аутентифікація є невід'ємними складовими розроблення ІС. Саме від них залежить функціонування обмежень несанкціонованого доступу користувачем до недоступних йому бібліотек, а також можливіть створювати та керувати власними організаціями.

Як тип аутентифікації для розробленої системи було обрано JWT-аутентифікацію із двома токенами. При логіні система генерує для користувача два JWT токена, один короткостроковий, інший з тривалим часом існування. Коректний вхід в систему можливий лише за умови, що обидва токени є валідними. Задля безпеки даних, вони зберігаються в HTTP-only cookies, через що унеможливується загроза від міжсайтового скриптингу(XSS атака).

Під час реєстрації та аутентифікації, використовуючи особливості життєвого циклу HTTP запиту, на ранніх етапах його відправки на сервер відбувається хешування чутливих даних, як наприклад пароля користувача. За допомогою цього неможливо перехопити пароль в чистому вигляді, оскільки пароль надходить до серверу вже у вигляді, який неможливо перетворити назад. Єдиний спосіб підібрати пароль та отримати доступ до аккаунту, це методом перебору підібрати комбінацію символів, які при перетворенні на хеш співпадуть з паролем користувача.

Також, під час відправки запитів, інакших від аутентифікації та реєстрації, відбувається перевірка JWT токенів на їх валідність, і лише в цьому випадку користувач отримує доступ до користування системою.

Log in

```
@POST("/sign-in", { preHandler: hashPassword })
 public async login(
   req: FastifyRequest<RouteGenericInterfaceLogin>,
```

```
    rep: FastifyReply
  ): Promise<FastifyReply> {
    const user = await this._userManagerService.login(req.body);
    const jwt = sign(user, <string>_CONFIG.app.security.REFRESH_TOKEN_SECRET,
{
      expiresIn: 21600,
    });
    const acc = sign(user, <string>_CONFIG.app.security.ACCESS_TOKEN_SECRET,
{
      expiresIn: 300,
    });
    const expDate = new Date("Fri, 31 Dec 9999 23:59:59 GMT");
    return rep
      .setCookie("ref", jwt, {
        path: "/",
        domain: "localhost",
        httpOnly: true,
        sameSite: "strict",
        expires: expDate,
      })
      .setCookie("acc", acc, {
        path: "/",
        domain: "localhost",
        httpOnly: true,
        sameSite: "strict",
        expires: expDate,
      })
      .setCookie("uuid", user.id, {
        path: "/",
        domain: "localhost",
        sameSite: "strict",
        expires: expDate,
      })
      .status(200)
      .send({ jwt: acc, user });
  }
```

Hash password

```
export const hashPassword = (
  req: FastifyRequest<
    RouteGenericInterfaceCreateUser | RouteGenericInterfaceLogin
  >,
  rep: FastifyReply,
  done: () => void
): void => {
  const hash = createHash("sha256");
```

```
 const encrypted = hash.update(req.body.password).digest("base64");
 req.body = { ...req.body, password: encrypted };
 done();
};
```

VerifyJWTHook

```
export const verifyJWTHook = async (
 req: FastifyRequest,
 rep: FastifyReply
): Promise<any> => {
 const refresh = req.cookies["ref"];
 const access = req.cookies["acc"];
 if (!refresh) {
   return rep.status(401).send(ErrorConstraints.NO_AUTH_TOKEN);
 }
 verify(
   refresh,
   <string>process.env.REFRESH_TOKEN_SECRET,
   (error, refreshDecoded: any) => {
     if (error) {
       console.log(error);
       return rep.status(401).send(ErrorConstraints.TOKEN_EXPIRED);
     }
     verify(
       access,
       <string>process.env.ACCESS_TOKEN_SECRET,
       (error, decoded) => {
         if (error || !decoded) {
           const newAccess = sign(
             refreshDecoded,
             <string>process.env.ACCESS_TOKEN_SECRET
           );
           rep.setCookie("acc", newAccess);
         }
         rep.setCookie("uuid", refreshDecoded.id);
       }
     );
   }
 );
};
```

9.2.2 Модуль роботи з публікаціями

При роботі з публікаціями важливим є те, що кожному різному формату файлів в HTTP запитах відповідає різних тип повертаємого змісту, який описується за допомогою заголовку Content-Type. Тому при завантаженні файлу на застосунок користувача з серверу, важливим є коректна відповідність між типом файлу публікації, та власне заголовком. На даний момент система підтримує 9 основних форматів документів:

- doc – старый формат Microsoft Word
- docx – новий формат Microsoft Word
- pdf – Adobe Portable Document Format, найбільш часто використовуємий формат
- rtf – Rich Text Format, відносно застарілий формат текстових документів із форматуванням від компаній Microsoft та Adobe
- odt – текстовий документ OpenDocument
- abw – документ текстового процесору AbiWord
- azv – текстовий формат книг Amazon Kindle
- txt – звичайний текст
- epub – формат електронних версій книг

_CONTENT_TYPES

```
export const _CONTENT_TYPE = {
 'pdf': 'application/pdf',
 'txt': 'text/plain',
 'rtf': 'application/rtf',
'docx':
'application/vnd.openxmlformats-officedocument.wordprocessingml.document',
 'doc': 'application/msword',
 'epub': 'application/epub+zip',
 'odt': 'application/vnd.oasis.opendocument.text',
 'azv': 'application/vnd.amazon.ebook',
```

```
        'abw': 'application/x-abiword'
      }
```

getFileContent

```
      public getFileContent(filepath: string) {
           return {
             format: filepath.split('.').pop(),
             content: readFile(filepath, (error: NodeJS.ErrnoException) => {
                  throw new BadRequestError(`Error occurred during loading file:
${error.message}`, 'ArticleModule');
             })
           };
         }
```

Download file endpoint

```
      @GET('/download/:id', { preValidation: verifyJWTHook })
       public async downloadArticle(
         req: FastifyRequest<RouteGenericInterfaceGetArticle>,
         rep: FastifyReply
       ) {
      const fileContent = await
this._articleManagerService.getArticleContent(req.query.id);
         const contentType = _CONTENT_TYPE[fileContent.format];
         rep.header('Content-Type', contentType).send(fileContent.content);
       }
```

9.2.3 Модуль RBAC

Модуль RBAC (Role-Based Access Control) в системі призначений для управління доступом користувачів до окремих організацій. Це забезпечує контроль над тим, які користувачі мають доступ до конкретних організацій і здатні взаємодіяти з ними.

Серед функціональних вимог можно виокремити такі пункти:

1. Ролі користувачів: Система повинна підтримувати створення різних ролей користувачів, таких як "Адміністратор", "Менеджер організації", "Звичайний користувач".
2. Адміністратор системи має можливість призначати ролі конкретним користувачам.
3. Кожен користувач повинен мати обмежений доступ до конкретних організацій. Це означає, що користувач може бути адміністратором однієї організації, але не мати доступу до інших.
4. Система повинна забезпечувати захист від несанкціонованого доступу до організацій, до яких користувач не має права доступу.

Для реалізації цих вимог, необхідно створити і налаштувати таблицю RBAC (Role-Based Access Control), де будуть вказані ролі користувачів і організації, до яких вони мають доступ. Кожен рядок в таблиці RBAC може містити таку інформацію:

user_id: Ідентифікатор користувача.

role_id: Ідентифікатор ролі користувача.

organization_id: Ідентифікатор організації.

За допомогою таблиці RBAC і відповідних запитів до бази даних система може перевіряти, чи має користувач права доступу до певної

організації. Користувачі можуть мати одну або більше ролей, і кожна роль може мати різні права доступу.

Додатково, можна розглянути використання системи сесійного керування та автентифікації для забезпечення безпеки авторизації користувачів.

Модуль RBAC в системі гарантує, що користувачі не зможуть взаємодіяти з тими організаціями, доступ до яких вони не мають. Це дозволяє забезпечити контроль і безпеку даних і дій користувачів у системі.

Важливо додати, що кожен рівень доступу має усі права попередніх рівнів доступу. Зокрема, у межах виконання даного проєкту, ми вирішили виокремити три рівні доступу:

1. User - має можливість переглядати файли організації, а також пропонувати додати новий файл в організацію.
2. Moderator - має можливість приймати або відхиляти файли звичайних користувачів. Таким чином, він модерує організацію, щоб вона не була заповнена непотрібними файлами.
3. Owner - має повний контроль над організацією. Даний рівень доступу може мати не одна людина, але людина, яка створила організацію - не зміниться.

Для спрощення бізнес-логіки, при взаємодії із платформою на правах звичайного користувача без організації - було встановлено такі ж самі правила. Тобто, у базі даних, загальна платформа вважається також організацією, де наявні свої користувачі, модератори та власники. Тим не менш, видалити саме цю - “публічну” організацію не можливо.

Технічно, модуль реалізовано у якості складової кожного мікросервісу через необхідність у коді зазначати необхідні рівні доступу

до виконання тієї чи іншої операції, а також необхідність зберігати інформацію користувача протягом усього контексту викликів.

Щоб не передавати цю інформацію у якості додаткових параметрів до кожного із методів, було вирішено скористатися модулем платформи Node.JS під назвою Async Hooks. Він надає можливість створювати тимчасове сховище типу “ключ” - “значення”, де ми зберігаємо усю інформацію користувача. У межах проєкту, сховище має назву “Async Storage”. Воно створюється окремо для кожного HTTP-виклику, тобто кожне сховище прив’язується до запиту напряму і знищується після того, як клієнт отримав відповідь від сервера. Розглянемо програмну реалізацію Async Storage.

Файл index.ts:

```
import { AsyncLocalStorage } from "async_hooks";
import { asyncMapStorage, KeyType } from "./types";

class AsyncStorageMap {
 asyncStorage: AsyncLocalStorage<asyncMapStorage> =
   new AsyncLocalStorage<asyncMapStorage>();

 public get(key: KeyType): number;

 public get(key: KeyType) {
   const store = this.asyncStorage.getStore();
   return store?.get(key);
 }

 public set = <T>(key: string, value: T): void => {
   const store = this.asyncStorage.getStore();
   store?.set(key, value);
 };

 public initStorage = (
   callback: () => void,
   defaults?: Record<string, any>
 ): void => {
   const store: asyncMapStorage = defaults
```

```
      ? new Map(Object.entries(defaults))
      : new Map();

    this.asyncStorage.run(store, () => {
      callback();
    });
  };
}
export default new AsyncStorageMap();
```

Файл resolvePromise.ts:

```
import { Callback, AuthorizationHookResponse } from "./types";

export const resolvePromise = (
 callback: Callback
): Promise<AuthorizationHookResponse> => {
 return new Promise((resolve, _rejects) => resolve(callback()));
};
```

Файл types.ts:

```
export type asyncMapStorage = Map<string, any>;

export type AuthorizationHookResponse = {
 userId: number;
 organizationId: number;
};

export type Callback = (...params: any) => any;

export type KeyType = keyof AuthorizationHookResponse;
```

У файлі index.ts наведена повна реалізація сховища, яке має три методи - get, set та initStorage. Перші два методи використовуються для отримання ідентифікаторів користувача та організації. Останній метод використовується перед обробкою HTTP-запиту користувача у prehandler хуках, згідно із життєвим циклом фреймворку Fastify. Prehandler хук

спрацьовує безпосередньо перед початком виконання основного контексту виклику, опрацьовує усю наявну логіку і передає керування контролеру.

У файлі resolvePromise.ts наведено обгортку для обробки promise-структур мови TypeScript. У файлі types.ts наведенод додаткові типи, які допомогають при роботі із RBAC модулем.

Як вже було зазначено, для створення сховища, необхідно викликати потрібний метод у prehandler хуку, перед виконанням роута. Для цього, ми розробили окрему функцію, якій надали назву authorizeUserHook.ts. Він визначає користувача та організацію на основі запиту. Інформація користувача зберігається у JWT-токені, а інформація про організацію, зокрема її ідентифікатор завжди наведено у квері параметрах запиту під ключем organizationId. Після цього, дані записуються у сховище та стають доступними до обробки у майбутньому. Розглянемо детально програмну реалізацію.

Файл authorizeUserHook.ts

```
import { FastifyReply, FastifyRequest } from "fastify";
import { AsyncResource } from "async_hooks";
import AsyncStorageMap from "../asyncStorage";
import { resolvePromise } from "../asyncStorage/resolvePromise";
import logger from "../../../../server/utils/logger";
import { AuthorizationHookResponse } from "../asyncStorage/types";

export const authorizeUserHook = (
 req: FastifyRequest,
 res: FastifyReply,
 done: any
) => {
 resolvePromise(async (): Promise<AuthorizationHookResponse | undefined> => {
   console.log(req.routerPath);
   try {
     const user = getUser(req);
     logger.info(
       {
```

```
        userId: user.id,
        organizationId: user.organizationId,
      },
      "Authorization User Hook Info"
    );
    return { userId: user.id, organizationId: user.organizationId };
  } catch (error) {
    done(error);
  }
}).then((value) => {
  if (value) {
    if (!value.organizationId) value.organizationId = 0;
      // instead of null there will be an auth function to get permission
based on user x-id
    AsyncStorageMap.initStorage(
      () => {
        const asyncResource = new AsyncResource("async-context");
        asyncResource.runInAsyncScope(done);
      },
      {
        userId: value.userId,
        userExternalId: value.organizationId,
      }
    );
  }
});
};
```

Ось приклад використання authorizeUserHook для роуту на створення окремої організації:

```
      @POST("/", { preValidation: authorizeUserHook })
public async createLibrary(
 req: FastifyRequest,
 rep: FastifyReply
): Promise<FastifyReply> {
 const library = await this._articleManagerService.createArticle(
   <any>req.body,
   await req.file()
 );
```

```
 return rep.status(200).send(library);
}
```

Після цього, необхідно встановити необхідний рівень доступу для взаємодії із окремими частинами програмного застосунку. Для цього було розроблено окремий метод, який має назву RBACEnforce та він додається до кожного методу, який взаємодіє із платформою у формі декоратора. У якості параметра функції, обирається необхідний рівень доступу, після чого при потраплянні контексту виклику у відповідну функцію, спочатку виконується логіка, закладена у декораторі, а потім загальна бізнес-логіка методу або функції.

Розглянемо більш детальну програмну реалізацію цього функціоналу.

Файл rbacEnforcementHook.ts:

```
      import AsyncStorageMap from "../asyncStorage";
      import { ForbiddenError } from
"../../../../server/utils/common/errors/error";
      import { AccessLevel } from "../../../../db/types/customTypes";
      import { getInstanceByToken } from "fastify-decorators";
      import AccessManager, { accessManagerToken } from
"../accessManagement";
      import { authorizeBasedOnIdentity } from
"../accessManagement/authorizeBasedOnIdentity";

      export function RBACEnforce(accessLevel: AccessLevel) {
       /*
        * Development note:
        * If AsyncStorageMap is not implemented in higher function sequence
        * it will always return undefined to any operation (get(), set()).
        *
        * The logic behind AsyncLocalStorage from async_hook is to provide
          * storage across asynchronous request context, thus it is
implemented
        * by default to have undefined value returned if it is not defined
```

```
        * because Event Loop is creating it by default.
        */

          return     function     (_target,     _propertyKey,     descriptor:
PropertyDescriptor) {
         const oldFun = descriptor.value;
         descriptor.value = async function () {
           const userId = AsyncStorageMap.get("userId");
           const organizationId = AsyncStorageMap.get("organizationId");
           console.log("[DEBUG] RBACEnforce: ", userId, organizationId);
           const accessManager =
             getInstanceByToken<AccessManager>(accessManagerToken);
           const userRbacInfo = await accessManager.getAccessLevel(
             userId,
             organizationId
           );
           if (userRbacInfo.rowCount === 0)
             throw new ForbiddenError(
                       `${userId}  does  not  have  access  to  organization
${organizationId}`,
               "RBACEnforce"
             );
           if (
                 authorizeBasedOnIdentity(userRbacInfo.rows[0].access_level,
accessLevel)
           ) {
             // eslint-disable-next-line prefer-rest-params
             return oldFun.apply(this, [...arguments]);
           } else {
             throw new ForbiddenError(
                   `Wrong  access  level  on  organization   ${organizationId}
for${userId}`,
               "RBACEnforce"
             );
           }
         };
       };
      }
```

Як ми бачимо, спочатку ми отримуємо дані із RBAC таблиці у методі getAccessLevel, а потім перевіряємо функцією authorizeBasedOnIdentity, чи дійсно користувач має необхідний доступ, чи ні.

Файл authorizeBasedOnIdentity.ts:

```
export function authorizeBasedOnIdentity(
 userAccessLevel: AccessLevel,
 resourceAccessLevel: AccessLevel
) {
 return userAccessLevel >= resourceAccessLevel;
}
```

Файл types.ts:

```
export enum AccessLevel {
 USER = 1,
 MODERATOR = 2,
 OWNER = 3,
}
```

Для подальшого покращення роботи платформи, дану систему можна змінити у двох напрямках.

По-перше, можна змінити розташування ідентифікатора організації із квері-параметрів у JWT-токен користувача. Це зменшить розміри посилань для користувачів.

Другий спосіб - розробити окремий мікросервіс, який буде займатися саме аутентифікацією прав користувача. Також, у мікросервіс треба буде додати кеш, у якому будуть зберігатись права доступу кожного користувача відносно кожної із організацій. Це забере в нас потребу постійно робити запити у таблицю RBAC, що повинно підвищити швидкодію усього додатку, адже розроблений кеш буде повністю відображати стан БД на поточний момент часу. Якщо користувач отримає нові права доступу, кеш зміниться відповідно.

До того ж, подібна реалізація покращить якість коду, адже не буде необхідності тримати частину однакової логіки в кожному мікорсервісі.

Також, це створює нам додаткову гнучкість у масштабуванні, адже подібну логіку набагато простіше масштабувати горизонтально.

Також,

9.2.4 Специфікація зберігання даних

Як вже було зазначено, для роботи із базою даних, ми обрали СКБД PostgreSQL. Для цього було розроблено окремий клас - DB, який створює пул для підключення до БД проєкту та виконує усі необхідні запити. Протягом усієї роботи додатку, усі запити обробляються у межах одного й того ж пулу, таким чином, нам не потрібно кожен раз створювати нове підключення. Наведемо програмну реалізацію цього класу.

Файл DB.ts:

```
import { Pool, QueryResult, QueryResultRow } from "pg";
import { Service } from "fastify-decorators";
import PgPool from "./config/pool";
import { PostgresError } from "../server/utils/common/errors/error";
import logger from "../server/utils/logger";

export const DBToken = Symbol("DBToken");

@Service(DBToken)
export class DB {
 private generalPool: Pool;

 constructor() {
   this.generalPool = new PgPool().getPool();
 }

 public async executeQuery<T extends QueryResultRow>(
   query: string,
```

```
    values?: any[]
  ): Promise<QueryResult<T>> {
    try {
      logger.info(`Query Executor Info:\n ${query}`);
      if (values) logger.info(`Query Executor Values:\n ${values}`);
      const result = await this.generalPool.query(query, values);
      return result;
    } catch (error: any) {
      logger.error(error);
      throw new PostgresError(error.message, "DB Module", error.code);
    }
  }
}
```

Клас має один generic-метод - executeQuery, який у якості параметрів отримує запит, а також можна передати масив значень, якщо використовується параметризований запит.

Також, Fastify пропонує розробку із використанням декораторів. Тобто, якщо обраний клас має патерн Singletone, тоді можна скористатись декоратором ‘@Service’ і під час запуску усього додатку, Fastify зберігає усі такі класи в окремий IoC-контейнер, доступ до якого можна отримати у будь-якому місці програми. Таким чином, це дуже зручна реалізація патерну Dependency Injection, вбудована у фреймворк Fastify.

Додатково, для підвищення якості коду, було розроблено спеціалізовані типи, які відображають поточний стан БД проєкту. Тобто, вони відображають дійсні назви усіх таблиць та їх колонок. Це необхідно було зробити, адже на проекті не використовується ORM, тому подібні схеми автоматично згенеруватись не можуть.

Подібна реалізація дозволяє контролювати будь-які зміни, пов’язані із базою даних, наприклад змінивши тип відповідей на запит, компілятор почне сигналізувати про невідповідність типізації. Ось приклад типізації відповіді із бази даних.

Файл libraries.ts:

```
export enum LibraryColumns {
 id = "id",
 name = "name",
 description = "description",
 owner_id = "owner_id",
 created_at = "created_at",
 updated_at = "updated_at",
}
```

Також, було розроблено механізм автоматичного створення таблиць, використовуючи існуючі переліки колонок самих таблиць.

Файл initLocalDatabase.ts

```
import {
 createLibraryQuery,
 createPublicationTagsQuery,
 createRBACQuery,
 createPublicationQuery,
 createRolesQuery,
 createUsersQuery,
 createSubscriptionsQuery,
 createPerUserOrganizationFileLimitsQuery,
 createSubscriptionsUsersQuery,
} from "./dbQueries";
import { getInstanceByToken } from "fastify-decorators";
import { DB, DBToken } from "./DBService";

export const initLocalDatabaseIfNotExists = async () => {
 const queryToExecute = `
   ${createLibraryQuery}
   ${createRBACQuery}
   ${createPublicationTagsQuery}
   ${createPublicationQuery}
   ${createRolesQuery}
   ${createUsersQuery}
   ${createSubscriptionsQuery}
   ${createPerUserOrganizationFileLimitsQuery}
```

```
        ${createSubscriptionsUsersQuery}
        `;
      await getInstanceByToken<DB>(DBToken).executeQuery(queryToExecute);
    };
```

Файл dbQueries.ts

```
    import { Tables } from "./types/tables";
    import { PublicationColumns } from "./types/tableColumns/publications";
    import { PublicationTagsColumns } from "./types/tableColumns/publicationTags";
    import { LibraryColumns } from "./types/tableColumns/libraries";
    import { RBACColumns } from "./types/tableColumns/rbac";
    import { RoleColumns } from "./types/tableColumns/roles";
    import { UserColumns } from "./types/tableColumns/users";
    import { PerUserOrganizationFileLimitsColumns } from "./types/tableColumns/perUserOrganizationFileLimits";
    import { SubscriptionsColumns } from "./types/tableColumns/subscriptions";
    import { SubscriptionsUsersColumns } from "./types/tableColumns/subscriptionsUsers";

    export const createPublicationTagsQuery = `create table if not exists ${Tables.publication_tags} (
        ${PublicationTagsColumns.id} serial primary key,
        ${PublicationTagsColumns.publication_id} integer not null,
        ${PublicationTagsColumns.tag_id} integer not null,
          ${PublicationTagsColumns.created_at} timestamp not null default current_timestamp,
          ${PublicationTagsColumns.updated_at} timestamp not null default current_timestamp
    );`;

    export const createPublicationQuery = `create table if not exists ${Tables.publications} (
        ${PublicationColumns.id} serial primary key,
        ${PublicationColumns.title} varchar(255) not null,
        ${PublicationColumns.filepath} varchar(255) not null,
        ${PublicationColumns.user_id} integer not null,
        ${PublicationColumns.library_id} integer not null,
```

```
        ${PublicationColumns.is_public} boolean not null default false,
        ${PublicationColumns.price} integer not null default 1,
        ${PublicationColumns.year} integer not null default 2020,
            ${PublicationColumns.created_at}  timestamp  not  null  default
current_timestamp,
            ${PublicationColumns.updated_at}  timestamp  not  null  default
current_timestamp
    );`;

    export  const  createLibraryQuery  =  `create  table  if  not  exists
${Tables.libraries} (
        ${LibraryColumns.id} serial primary key,
        ${LibraryColumns.name} varchar(255) not null,
        ${LibraryColumns.owner_id} integer not null,
        ${LibraryColumns.description} text,
             ${LibraryColumns.created_at}  timestamp  not  null  default
current_timestamp,
             ${LibraryColumns.updated_at}  timestamp  not  null  default
current_timestamp
    );`;

    export  const  createRBACQuery  =  `create  table  if  not  exists
${Tables.rbac} (
            ${RBACColumns.library_id} integer not null,
            ${RBACColumns.user_id} integer not null,
            ${RBACColumns.role_id} integer not null,
                   ${RBACColumns.created_at}  timestamp  not  null  default
current_timestamp,
                   ${RBACColumns.updated_at}  timestamp  not  null  default
current_timestamp,
             PRIMARY KEY (${RBACColumns.library_id}, ${RBACColumns.user_id},
${RBACColumns.role_id})
    );`;

    export  const  createRolesQuery  =  `create  table  if  not  exists
${Tables.roles} (
        ${RoleColumns.id} serial primary key,
        ${RoleColumns.title} varchar(255) not null,
        ${RoleColumns.access_level} integer not null,
               ${RoleColumns.created_at}  timestamp  not  null  default
current_timestamp,
```

```
            ${RoleColumns.updated_at}   timestamp   not   null   default
current_timestamp
      );`;

      export   const   createUsersQuery   =   `create   table   if   not   exists
${Tables.users} (
         ${UserColumns.id} serial primary key,
         ${UserColumns.first_name} varchar(255) not null,
         ${UserColumns.last_name} varchar(255) not null,
         ${UserColumns.email} varchar(255) not null,
         ${UserColumns.password} varchar(255) not null,
               ${UserColumns.created_at}   timestamp   not   null   default
current_timestamp,
               ${UserColumns.updated_at}   timestamp   not   null   default
current_timestamp
      );`;

      export const createSubscriptionsQuery = `
      CREATE TABLE IF NOT EXISTS ${Tables.subscriptions} (
         id SERIAL PRIMARY KEY,
         ${SubscriptionsColumns.user_id} INTEGER NOT NULL,
         ${SubscriptionsColumns.organization_limit_number} INTEGER NOT NULL,
         ${SubscriptionsColumns.price} INTEGER NOT NULL,
         ${SubscriptionsColumns.file_size_limit_number} INTEGER NOT NULL,
            ${SubscriptionsColumns.created_at}  TIMESTAMP  NOT  NULL  DEFAULT
CURRENT_TIMESTAMP,
            ${SubscriptionsColumns.updated_at}  TIMESTAMP  NOT  NULL  DEFAULT
CURRENT_TIMESTAMP,
         FOREIGN KEY (user_id) REFERENCES users(id)
      );`;

      export const createPerUserOrganizationFileLimitsQuery = `
      CREATE TABLE IF NOT EXISTS ${Tables.per_user_organization_file_limits}
(
         ${PerUserOrganizationFileLimitsColumns.id} SERIAL PRIMARY KEY,
         ${PerUserOrganizationFileLimitsColumns.user_id} INTEGER NOT NULL,
          ${PerUserOrganizationFileLimitsColumns.organization_id} INTEGER NOT
NULL,
          ${PerUserOrganizationFileLimitsColumns.file_size_limit} INTEGER NOT
NULL,
```

```
          ${PerUserOrganizationFileLimitsColumns.created_at} TIMESTAMP NOT
NULL DEFAULT CURRENT_TIMESTAMP,
          ${PerUserOrganizationFileLimitsColumns.updated_at} TIMESTAMP NOT
NULL DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY (${PerUserOrganizationFileLimitsColumns.user_id})
REFERENCES ${Tables.users}(${UserColumns.id}),
                                                FOREIGN KEY
(${PerUserOrganizationFileLimitsColumns.organization_id}) REFERENCES
${Tables.libraries}(${LibraryColumns.id})
      );`;

      export const createSubscriptionsUsersQuery = `
      CREATE TABLE IF NOT EXISTS ${Tables.subscriptions_users} (
         ${SubscriptionsUsersColumns.subscription_id} INTEGER NOT NULL,
         ${SubscriptionsUsersColumns.user_id} INTEGER NOT NULL,
          ${SubscriptionsUsersColumns.created_at} TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP,
          ${SubscriptionsUsersColumns.updated_at} TIMESTAMP NOT NULL DEFAULT
CURRENT_TIMESTAMP,
              PRIMARY KEY (${SubscriptionsUsersColumns.subscription_id},
${SubscriptionsUsersColumns.user_id}),
              FOREIGN KEY (${SubscriptionsUsersColumns.subscription_id})
REFERENCES ${Tables.subscriptions}(${SubscriptionsColumns.id}),
            FOREIGN KEY (${SubscriptionsUsersColumns.user_id}) REFERENCES
${Tables.users}(${UserColumns.id})
      );`;
```

9.2.5 Взаємодія між мікросервісами

У сучасних розподілених системах, таких як мікросервісні архітектури, взаємодія між окремими сервісами є ключовим аспектом для забезпечення ефективності та гнучкості системи. Одним із популярних підходів до такої взаємодії є API-орієнтований підхід. Саме його ми вирішили використовувати.

В основі API-орієнтованого підходу лежить використання Application Programming Interfaces (API) для комунікації між

мікросервісами. Кожен мікросервіс виставляє свій API, через який інші сервіси можуть надсилати запити та отримувати відповіді. Це дозволяє сервісам обмінюватися даними та функціоналом в стандартизованому, зрозумілому форматі, що сприяє легшій інтеграції та знижує залежність між різними частинами системи.

Для цього, на цілий мікросервіс створюється відповідний клас у форматі Singletone, кожен метод якого буде реалізувати необхідний API-запит на інший мікросервіс. URL-посилання на сам ресурс буде знаходитись у конфігураційному файлі мікросервісу.

Файл dev.ts:

```
import PinoPretty from "pino-pretty";
import { Config } from "../types/interface";
import * as rTracer from "cls-rtracer";

const devConfig: Config = {
 server: {
   host: "localhost",
   port: 3000,
 },
 db: {
   pg: {
     user: "postgres",
     password: "password",
     host: "localhost",
     database: "clean-db",
     port: 5432,
   },
 },
 services: {
   example: "http://localhost:3001",
 },
 app: {
   env: "dev",
   name: "Library-OBE",
   abbr: "LOBE",
```

```
    version: "1.0.0",
    documentation: {
      prefix: "/docs",
    },
    logger: {
      options: {
        level: "debug",
        mixin() {
          return { 'r-request-id': rTracer.id() };
        },
      },
      streams: [
        {
          level: "debug",
          stream: PinoPretty({
            translateTime: "HH:MM:ss Z",
            colorize: true,
            sync: true,
            destination: 2,
          }),
        },
      ],
    },
    security: {
      REFRESH_TOKEN_SECRET: "refresh-token",
      ACCESS_TOKEN_SECRET: "access-token"
    },
    services: {
        article: "http://localhost:3001",
    }
  },
};

export default devConfig;
```

Ось приклад конфігураційного файлу мікросервісу. Можна побачити окремі рядки коду, у яких прописан хост мікросервісу Article:

```
    services: {
        article: "http://localhost:3001",
    }
```

Даний файл, як зрозуміло із назви - dev.ts, використовується під час локальної розробки. За потреби, можна розробити конфігураційний файл, який буде значення із змінних оточення.

Для покращення комунікації та управління API в мікросервісних архітектурах часто використовуються API Gateway. API Gateway діє як єдина точка входу для всіх клієнтських запитів, яка направляє їх до відповідних мікросервісів. Це не тільки спрощує процес взаємодії для клієнтів, але й забезпечує додатковий рівень абстракції та безпеки.

Використання API Gateway значно спрощує масштабування системи та впровадження балансувальників навантаження. З API Gateway, розподіл навантаження може бути виконаний централізовано, що дозволяє легко масштабувати окремі сервіси залежно від їхньої поточної навантаженості та вимог. Також, це сприяє більш ефективному

10. КЕРІВНИЦТВО КОРИСТУВАЧА

На головній сторінці у користувача буде дві опції - увійти до існуючого аккаунту чи зробити новий.

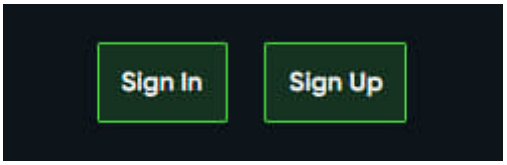

Рисунок 10.1 - Опції sign in/up

Саме модальне вікно авторизації виглядає наступним чином.

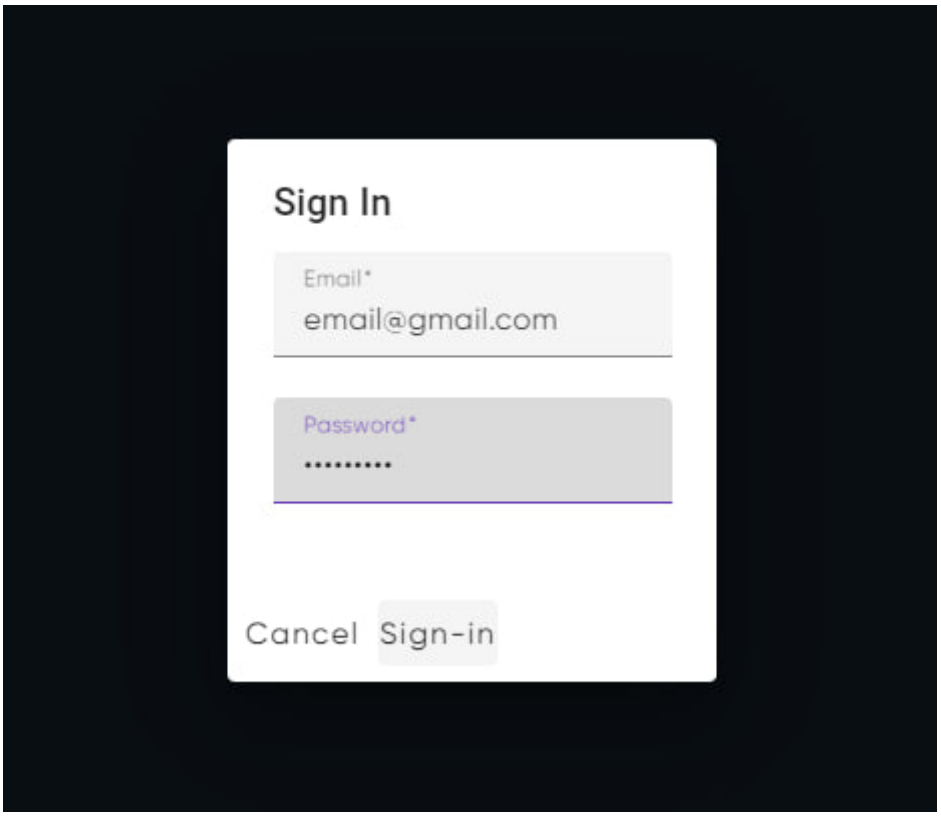

Рисунок 10.2 - Модальне вікно авторизації

На головній сторінці користувача буде текстове поле для пошуку існуючих записів, та кнопки-опції для створення нової організації або публікації.

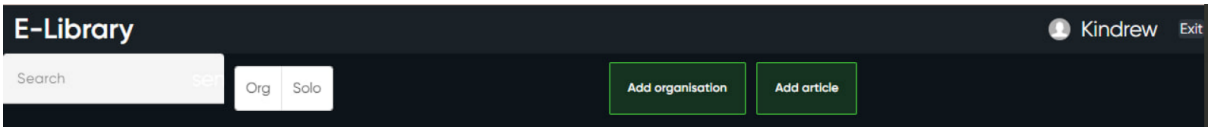

Рисунок 10.3 - Опції на головній сторінці

При кліку на додавання нової організації користувачу треба також вибрати комерційний план, модалка вибору якого буде виглядати наступним чином.

Рисунок 10.4 - Модальне вікно вибору комерційного плану

Висновки

У процесі виконання даної роботи ми реалізували проект електронної бібліотеки, яка відповідає сучасним вимогам до інформаційних систем. Ми успішно створили інтерактивний, безпечний та інтуїтивно зрозумілий ресурс, який значно спрощує процес знаходження та використання інформації для користувачів різного віку та освітнього рівня. Наша робота зосереджувалася на створенні рішення, що є не просто технологічно передовим, але й максимально корисним та зручним для кінцевих користувачів.

Проект електронної бібліотеки був спрямований на створення системи, яка не лише забезпечує швидкий доступ до широкого спектру освітніх та наукових матеріалів, але й відкриває нові горизонти для здобуття знань, роблячи їх доступними в будь-який час та з будь-якої точки світу. Ми провели глибокий аналіз потреб та вимог користувачів, що дозволило нам адаптувати систему до різних освітніх рівнів та наукових інтересів. Цей підхід забезпечив високу релевантність та користувацьку цінність нашого інтернет-сервісу.

Ми ретельно вивчили різні аспекти управління контентом, доступності та інтерфейсу користувача, щоб забезпечити, що наша система буде не тільки потужною, але й інтуїтивно зрозумілою та легкою у використанні. Особливу увагу було приділено адаптації системи до потреб різноманітної аудиторії, включаючи студентів, викладачів, дослідників та інших користувачів, які постійно шукають надійні та якісні освітні ресурси. Таким чином, наш проект не лише відповідає актуальним тенденціям у сфері освіти, але й сприяє інтеграції наукового співтовариства, надаючи зручні інструменти для спілкування та обміну знаннями.

У процесі розробки ми також врахували ключові аспекти безпеки, зручності та інтуїтивність інтерфейсу, що є критично важливим для ефективного та комфортного використання електронної бібліотеки. Тому, наші зусилля були спрямовані на створення системи, яка не тільки відповідає сучасним технічним стандартам, але й відповідає етичним нормам і стандартам конфіденційності. Це включає в себе захист персональних даних користувачів та забезпечення високого рівня безпеки при обробці та зберіганні інформації.

Завершення цього проекту не тільки демонструє нашу здатність ефективно розробляти сучасні інформаційні системи, але й підкреслює значення інноваційних технологій у сфері освіти та наукових досліджень. Ми впевнені, що ця робота зробить значний внесок у розвиток освітніх інституцій та допоможе у доступі до знань і інформації для широкого кола користувачів. Наш проект є яскравим прикладом того, як сучасні технології можуть трансформувати освітній процес, роблячи його більш доступним, гнучким та ефективним для всіх учасників освітнього процесу.