МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПРИРОДНИЧО-ГУМАНІТАРНИЙ ФАХОВИЙ КОЛЕДЖ

ДВНЗ «Ужгородський національний університет»

Реєстраційний номер \_\_\_\_\_\_\_\_\_\_\_

Дата реєстрації \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Логінов Євген Сергійович

студент 4 курсу

групи КН-43

денної форми навчання

ІНП № 253042ФМБД

**ЗВIT**

ПРОХОДЖЕННЯ ПЕРЕДДИПЛОМНОЇ ПРАКТИКИ

Галузь знань 12 «Інформаційні технології»,

спеціальність 121 «Інженерія програмного забезпечення»

**Розробка соціальної мережі «Thrume» з функціями дзвінків та транслювання екрану**

**Допущено до захисту** Науковий керівник:

Дата «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ Лінчук Н.Л. \_\_\_\_\_\_\_\_\_\_\_\_

Підпис \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Консультант з охорони праці:

Щербан І. І. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Консультант з економіки:

Безносько Ю.В.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Практика захищена «\_\_\_\_\_»\_\_\_\_\_\_\_\_ 2025 р. з оцінкою «\_\_\_\_\_\_\_\_\_\_\_\_»**

**Підписи членів комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Ужгород 2025

Автор проекту: Логінов Євген Сергійович

Тема дипломного проекту: Розробка соціальної мережі «Thrume» з функціями дзвінків та транслювання екрану

Науковий керівник: викл. Лінчук Наталія Леонідівна

Рецензент:

(науковий ступінь, посада, прізвище, ім’я, по батькові)

Дата захисту « » 2025 року

**АНОТАЦІЯ**

Дипломний проект складається із вступу, чотирьох розділів, висновків, переліку використаних джерел та додатків.

У першому розділі («Теоретична частина») розглянуто загальні відомості про технології, що були використані.

Другий розділ («Практична частина») містить інформацію про взаємодію елементів проекту та його функціональні частини. Також наводиться користувацький інтерфейс програмного засобу та інтерфейс адміністратора.

У розділі «Економічна частина» прораховано витрати на створення проекту, повну собівартість і гуртову ціну програмного продукту.

У розділі «Охорона праці» описано характеристику приміщення, в якому проходило написання проекту. Також проведено аналіз умов праці та розрахунок кількості комп’ютеризованих робочих місць та вогнегасників в заданому приміщенні.

Ключові слова: вебсайт, C#, Postman, ендпоїнт, ASP .NET, PostgreSQL, БД.

Обсяг пояснювальної записки: 130 сторінок.

Перелік додатків: схема проекту, лістинг програми.

Project author: Lohinov Yevhen Serhiiovych

Project subject: Development of a social network “Thrume” with the functions of calls and screen broadcasting

Supervisor: lect. Linchuk Nataliia Leonidivna

Reviewer:

Date: "\_\_\_\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2025

**ANNOTATION**

Graduate project consists of introduction, four chapters, conclusions, list of sources used and applications.

In the first section ("The Theoretical Part"), you will find general information about the technologies that were used.

The second section ("Practical part") contains information on the interaction of the project elements and its functional parts. Also provided is the user interface for the software and the admin interface.

In the section "Economic part" are calculated the costs for project creation, the full cost and the total price of the software product.

In the section "Occupational Health" applies the characteristics of the premises where the project was written. An analysis of the working conditions and the calculation of the number of computerized workplaces and fire extinguishers in the given premises was also conducted.

Keywords: website, C#, Postman, endpoint, ASP .NET, PostgreSQL, DB.

Explanatory note volume: 130

List of applications: project scheme, listing of the program.

**ЗМІСТ**

[ВСТУП 6](#_Toc198250978)

[1 ОГЛЯД ТЕОРЕТИЧНОЇ ТА ПРАКТИЧНОЇ ЧАСТИН ДИПЛОМНОГО ПРОЕКТУ 7](#_Toc198250979)

[1.1 Мова програмування C# 7](#_Toc198250980)

[1.2 Фреймворк ASP.NET 7](#_Toc198250981)

[1.3 Система керування базами даних PostgreSQl 9](#_Toc198250982)

[1.4 Огляд роботи проекту 12](#_Toc198250983)

[2 ЕКОНОМІЧНА ЧАСТИНА 24](#_Toc198250984)

[2.1 Характеристика та призначення роботи 24](#_Toc198250985)

[2.2 Розрахунок витрат на створення програмного продукту 24](#_Toc198250986)

[2.2.1 Розрахунок вартості витрат на матеріали 24](#_Toc198250987)

[2.2.2 Розрахунок витрат на оплату праці розробника 25](#_Toc198250988)

[2.3 Розрахунок витрат на утримання та експлуатацію ПЕОМ 26](#_Toc198250989)

[2.4 Розрахунок загальновиробничих витрат 27](#_Toc198250990)

[2.5 Обчислення собівартості програмного продукту «Thrume» 28](#_Toc198250991)

[2.6 Обчислення гуртової ціни програмного продукту «Thrume» 28](#_Toc198250992)

[2.7 Висновки щодо економічної доцільності розробки 29](#_Toc198250993)

[3 ОХОРОНА ПРАЦІ 30](#_Toc198250994)

[3.1 Аналіз потенційних небезпек при розробці 30](#_Toc198250995)

[3.2 Характеристика приміщення 32](#_Toc198250996)

[3.3 Заходи щодо поліпшення умов з охорони праці 36](#_Toc198250997)

[3.4 Інструкція з охорони праці при роботі з ПЕОМ та відеодисплейними терміналами 39](#_Toc198250998)

[3.4.1 Загальні положення 39](#_Toc198250999)

[3.4.2 Вимоги безпеки перед початком роботи 42](#_Toc198251000)

[3.4.3 Вимоги безпеки під час роботи 43](#_Toc198251001)

[3.4.4 Вимоги безпеки після закінчення роботи 44](#_Toc198251002)

[3.4.5 Вимоги безпеки в аварійних ситуаціях 45](#_Toc198251003)

[ВИСНОВКИ 46](#_Toc198251004)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 47](#_Toc198251005)

[Додаток 1 Фізична схема 49](#_Toc198251006)

[Додаток 2 Лістинг програми 50](#_Toc198251007)

# ВСТУП

Сучасні потреби користувачів вимагають інтегрованих комунікаційних рішень, що поєднують функції соціальних мереж із дзвінками та трансляцією екрану, роблячи розробку «Thrume» своєчасною.

Проблематика полягає у тому, що проєкт вирішує проблему фрагментації комунікаційних інструментів, пропонуючи єдину платформу замість використання кількох окремих додатків для спілкування та спільної роботи.

Мета роботи – це розробка соціальної мережі «Thrume», що надає користувачам можливості текстового спілкування, дзвінків та транслювання екрану. Для досягнення мети були поставлені завдання дослідження, серед яких: проаналізувати аналоги, спроєктувати архітектуру, реалізувати функціонал та провести тестування розробленої системи.

Об’єктом дослідження є процес розробки програмного забезпечення для соціальних комунікацій, а предметом дослідження – архітектура та функціонал соціальної мережі «Thrume». У роботі використано такі методи дослідження, як аналіз існуючих рішень, системне проєктування, програмна реалізація та тестування програмного продукту.

Наукова новизна полягає у розробці архітектурного рішення для соціальної мережі, що ефективно інтегрує різноманітні комунікаційні функції на сучасному технологічному стеку.

Практична значущість роботи полягає у створенні прототипу соціальної мережі «Thrume», готового до розгортання та використання для корпоративних, освітніх чи особистих цілей.

# ОГЛЯД ТЕОРЕТИЧНОЇ ТА ПРАКТИЧНОЇ ЧАСТИН ДИПЛОМНОГО ПРОЕКТУ

## Мова програмування C#

C# (вимовляється Сі-шарп) — об’єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (належить Microsoft).

Синтаксис C# близький до С++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато від своїх попередників — мов С++, Object Pascal, Модула і Smalltalk — С#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова С#, на відміну від C++, не передбачає множинне успадкування класів.[1]

Станом на травень 2025 року поточна стабільна версія мови C# 13.0, яка була випущена в 2024 році як частина платформи .NET 9.0.

## Фреймворк ASP.NET

ASP.NET — це безкоштовний веб-фреймворк для створення чудових веб-сайтів та веб-додатків з використанням HTML, CSS та JavaScript. Ви також можете створювати веб-API та використовувати технології реального часу, такі як Web Sockets.

ASP.NET пропонує три фреймворки для створення веб-додатків: Web Forms, ASP.NET MVC та ASP.NET Web Pages. Усі три фреймворки є стабільними та зрілими, і ви можете створювати чудові веб-додатки за допомогою будь-якого з них. Незалежно від того, який фреймворк ви оберете, ви отримаєте всі переваги та функціонал ASP.NET.[2]

ASP.NET надає такі переваги:

* + Єдине рішення для створення веб-інтерфейсу користувача та веб-API.
  + Розроблено для тестування (або: Спроектовано для тестування).
  + Razor Pages спрощує написання коду для сторінкових сценаріїв та підвищує його ефективність.
  + Blazor дозволяє використовувати в браузері мову C# разом із JavaScript; спільне використання серверної та клієнтської логіки додатків, написаних за допомогою .NET.
  + Можливість розробки та запуску в ОС Windows, macOS та Linux.
  + Відкритий вихідний код та орієнтація на спільноту.
  + Інтеграція сучасних клієнтських платформ та робочих процесів розробки.
  + Підтримка розміщення служб віддаленого виклику процедур (RPC) за допомогою gRPC.
  + Готова до роботи в хмарі система конфігурації, що базується на середовищі.
  + Вбудоване впровадження залежностей.
  + Спрощений, високопродуктивний, модульний конвеєр HTTP-запитів.
  + Можливість розміщення на таких платформах:
  + Kestrel
  + IIS
  + HTTP.sys
  + Nginx
  + Docker
  + Підтримка паралельного використання версій (або: Управління паралельними версіями / Розгортання пліч-о-пліч).
  + Інструментарій, що спрощує процес сучасної веб-розробки.

## Система керування базами даних PostgreSQl

PostgreSQL вирізняється багатим набором функцій, які роблять її гнучким та потужним інструментом:

1. Типи даних**:** PostgreSQL підтримує широкий спектр вбудованих типів даних, включаючи стандартні числові типи (цілі, з плаваючою комою), текстові (VARCHAR, TEXT, CHAR), дати та час (TIMESTAMP, DATE, TIME, INTERVAL), логічні (BOOLEAN), бінарні (BYTEA). Окрім цього, вона пропонує розширені типи, такі як масиви (arrays), геометричні типи (точки, лінії, полігони), мережеві адреси (CIDR, INET, MACADDR), UUID, XML та, що особливо важливо для сучасних додатків, JSON та JSONB (бінарний, оптимізований JSON з можливістю індексації). Користувачі також можуть створювати власні типи даних.
2. Індекси**:** Для прискорення виконання запитів PostgreSQL надає різноманітні типи індексів:
   * B-дерево (B-tree): Стандартний тип індексу, підходить для більшості операцій порівняння (=, <, >, BETWEEN, IN).
   * Хеш-індекси (Hash): Ефективні для точних співпадінь (=), але з певними обмеженнями (наприклад, не підтримують WAL до версії 10).
   * GiST (Generalized Search Tree): Дозволяє створювати індекси для складних типів даних, таких як геометричні дані, повнотекстовий пошук.
   * SP-GiST (Space-Partitioned Generalized Search Tree): Розширення GiST, ефективне для незбалансованих структур даних, таких як префіксні дерева.
   * GIN (Generalized Inverted Index): Оптимальний для індексації складених значень, таких як масиви або JSONB, де потрібно шукати окремі елементи всередині значення.
   * BRIN (Block Range INdexes): Ефективні для дуже великих таблиць, дані в яких фізично відсортовані або згруповані.  
     PostgreSQL також підтримує індекси за виразами (indexes on expressions), часткові індекси (partial indexes) та унікальні індекси.
3. Транзакції та паралелізм (ACID та MVCC): PostgreSQL суворо дотримується принципів ACID (Atomicity, Consistency, Isolation, Durability), що гарантує надійність транзакцій. Для керування паралельним доступом до даних використовується механізм MVCC (Multi-Version Concurrency Control). MVCC дозволяє читачам не блокувати записувачів, і навпаки, забезпечуючи високий рівень паралелізму та продуктивності в багатокористувацьких середовищах.
4. Процедури та функції: Користувачі можуть визначати власні функції та збережені процедури для інкапсуляції логіки на стороні сервера. Ці процедури можуть бути написані на стандартному SQL, а також на різних процедурних мовах, таких як:
   * PL/pgSQL: Вбудована мова, схожа на PL/SQL в Oracle.
   * PL/Tcl, PL/Perl, PL/Python: Дозволяють використовувати потужності відповідних скриптових мов.
   * PL/Java, PL/R, PL/Scheme та інші: Доступні як розширення.
5. Тригери: Тригери – це функції, які автоматично виконуються у відповідь на певні події в таблиці (INSERT, UPDATE, DELETE). Вони можуть виконуватися до або після операції, на рівні рядка або оператора.
6. Розширюваність: Однією з найсильніших сторін PostgreSQL є її розширюваність. Користувачі можуть додавати нові типи даних, функції, оператори, агрегатні функції, методи індексації та процедурні мови. Існує багата екосистема розширень, таких як PostGIS для геопросторових даних, Citus для розподілених баз даних, TimescaleDB для часових рядів та багато інших. Механізм Foreign Data Wrappers (FDW) дозволяє PostgreSQL отримувати доступ до даних, що зберігаються в інших базах даних або зовнішніх джерелах, ніби вони є локальними таблицями.
7. Реплікація та висока доступність: PostgreSQL пропонує різні механізми реплікації для забезпечення високої доступності та масштабування навантаження на читання:
   * Потокова реплікація (Streaming Replication): Асинхронна або синхронна передача змін (WAL-записів) з головного сервера на один або декілька резервних (standby) серверів.
   * Логічна реплікація (Logical Replication): Дозволяє реплікувати зміни на рівні окремих таблиць або баз даних, навіть між різними версіями PostgreSQL або на інші СКБД.
8. Інші важливі можливості:
   * Повнотекстовий пошук: Вбудовані засоби для індексації та пошуку текстової інформації, включаючи підтримку різних мов, ранжування результатів та векторів документів.
   * Обмеження цілісності: Підтримка первинних ключів, зовнішніх ключів (foreign keys), обмежень перевірки (check constraints) та унікальних обмежень (unique constraints).
   * Схеми (Schemas): Дозволяють організовувати об'єкти бази даних (таблиці, функції тощо) в логічні групи, подібно до просторів імен.
   * Віконні функції (Window Functions): Потужний інструмент для виконання обчислень над набором рядків, пов'язаних з поточним рядком.
   * Common Table Expressions (CTE): Дозволяють створювати тимчасові, іменовані набори результатів, які можна використовувати в межах одного запиту, покращуючи його читабельність та структуру.
   * Успадкування таблиць: Механізм, де таблиця може успадковувати стовпці від однієї або кількох батьківських таблиць (хоча для багатьох випадків використання зараз частіше рекомендують партиціонування).
   * Партиціонування таблиць (Table Partitioning): Дозволяє розбивати великі таблиці на менші, більш керовані частини (партиції), що може значно покращити продуктивність запитів та операцій обслуговування.

PostgreSQL розробляється активною та розподіленою глобальною спільнотою, що складається з окремих волонтерів та представників компаній. Це забезпечує незалежність проекту та його орієнтацію на технічну досконалість. Ліцензія PostgreSQL є дуже дозвільною, дозволяючи використовувати, модифікувати та розповсюджувати програмне забезпечення практично без обмежень, що сприяє його широкому впровадженню як у відкритих, так і в комерційних проектах.

Завдяки своїй стабільності, продуктивності та багатому набору функцій PostgreSQL використовується багатьма великими компаніями та організаціями по всьому світу, такими як Apple, Fujitsu, Red Hat, Cisco, Skype, Instagram (частково), а також урядовими установами. Вона підходить для різноманітних завдань: від веб-додатків та мобільних бекендів до аналітичних систем, геоінформаційних систем (ГІС) та систем керування фінансовими даними.[4]

## Огляд роботи проекту

У даному розділі розглянуто реалізовані веб-ендпоїнти (API) та їхню поведінку. Вони забезпечують взаємодію клієнтської частини з серверною. (рисунок 1.1 – 1.24).Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.1 Веб-ендпоїнт для надсилання antiforgery токену

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.2 Веб-ендпоїнт виходу з акаунту

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.3 Успішне повернення результату з ендпоїнту реєстрації

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.4 Повернення помилки 400 з ендпоїнту реєстрації через вже існуючі дані облікового запису

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.5 Повернення помилки 400 з ендпоїнту реєстрації через валідацію паролю

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.6 Повернення помилки 400 з ендпоїнту реєстрації через невалідний формат електронної пошти

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.7 Повернення помилки 401 з ендпоїнту , через те, що електронна пошта користувача ще не підтверджена.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.8 Повернення помилки 401 з ендпоїнту логіну через введення неправильного пароля.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.9 Повернення 200 з ендпоїнту логіну через успішну аутентифікацію.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.10 Ендпоїнт підтвердження емейлу

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.11 Ендпоїнт перенадсиланню підтвердження емейлу

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.12 Ендпоїнт надсилання коду для зміни паролю

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.13 Ендпоїнт зміни паролю акаунта

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.14 Ендпоїнт створення поста

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.15 Ендпоїнт отримання поста по юзернейму

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.16 Ендпоїнт для лайкання поста

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.17 Ендпоїнт для зміни фотографії профілю

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.18 Ендпоїнт для отримання інформації про акаунт

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.19 Ендпоїнт для створення коменту на пості

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.20 Ендпоїнт для видалення коментаря

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.21 Ендпоїнт для отримання діалогів між акаунтами

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.22 Ендпоїнт для надсилання повідомлення

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.23 Ендпоїнт для розпочатку діалога

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.24 Ендпоїнт для того щоб отримати конкретний діалог

# ЕКОНОМІЧНА ЧАСТИНА

## Характеристика та призначення роботи

Завданням даної дипломної роботи є розробка соціальної мережі «Thrume», що включатиме функціонал для обміну повідомленнями, створення публікацій, а також реалізацію можливостей здійснення аудіо/відео дзвінків та трансляції екрану між користувачами. Цей програмний продукт спрямований на створення зручної та багатофункціональної платформи для спілкування та взаємодії користувачів в онлайн-середовищі.

Метою економічної частини дипломного проекту є визначення сукупних витрат, пов’язаних із створенням програмного продукту «Thrume». Розрахунок вартості розробки охоплює аналіз витрат на допоміжні матеріали, оплату праці розробника, амортизацію та експлуатацію персональної електронно-обчислювальної машини (ПЕОМ), а також загальновиробничі витрати. Проведення цих розрахунків дозволить визначити повну собівартість програмного продукту та обчислити його орієнтовну гуртову ціну, що є важливим для оцінки економічної доцільності проекту.

## Розрахунок витрат на створення програмного продукту

### Розрахунок вартості витрат на матеріали

Для розробки програмного продукту «Thrume» необхідні певні допоміжні матеріали. Перелік цих матеріалів та їх орієнтовна вартість наведені в таблиці 1.

Таблиця 2.1 Допоміжні матеріали для розробки «Thrume»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Найменування матеріалу | Виробник і марка | Одиниця виміру | Норма затрат | Ціна, грн | Вартість, грн |
| 1 | Офісний папір формату А4 | Mondi Maestro Standard+ | Упак. | 0,25 | 220,00 | 55,00 |
| 2 | Картридж для принтера | Canon PG-545 | Шт. | 1 | 650,00 | 650,00 |
| 3 | Флеш-накопичувач USB 32GB | Kingston DataTraveler | Шт. | 1 | 250,00 | 250,00 |
|  | Всього | | | | | 955,00 |

Загальна вартість допоміжних матеріалів *Звм* для розробки соціальної мережі «Thrume» складає 955,00 грн.

### Розрахунок витрат на оплату праці розробника

Для розрахунку заробітної плати розробника програмного продукту «Thrume» використовується погодинна форма оплати праці. Загальна сума основної заробітної плати (ЗП) визначається за формулою (1):

|  |  |
| --- | --- |
|  | (1) |

де *t* – загальний час, витрачений на створення програмного продукту, год. – тарифна ставка працівника певного розряду за годину праці, грн./год.

Трудомісткість операцій, що виконуються в процесі створення соціальної мережі «Thrume», наведена в таблиці 2.

Таблиця 2.2 Трудомісткість операцій розробки «Thrume»

|  |  |  |
| --- | --- | --- |
| № п/п | Найменування операції | Витрати, год |
| 1 | Постановка задачі, аналіз вимог та предметної області | 20 |
| 2 | Проектування архітектури системи, баз даних та API | 45 |
| 3 | Розробка UI/UX дизайну інтерфейсу користувача | 35 |
| 4 | Реалізація основних програмних модулів (реєстрація, авторизація, профілі, стрічка) | 80 |
| 5 | Реалізація функціоналу обміну повідомленнями (чати) | 40 |
| 6 | Реалізація функціоналу аудіо/відео дзвінків та трансляції екрану | 70 |
| 7 | Відлагоджування програмного продукту (Тв) | 50 |
| 8 | Тестування (функціональне, навантажувальне, UI/UX) | 40 |
| 9 | Підготовка технічної документації | 20 |
|  | Всього (t) | 400 |

Для визначення годинної тарифної ставки (ТЗПГОД) розробника приймаємо 16-й тарифний розряд згідно з Єдиною тарифною сіткою на 2025 рік. Посадовий оклад для 16-го розряду становить 8914,00 грн.  
Річний фонд робочого часу (ТЕОМ) при 40-годинному робочому тижні, враховуючи умови воєнного стану протягом усього 2025 року, становить 2088 годин. Середньомісячна норма тривалості робочого часу: 2088 год / 12 міс. = 174 год/міс. Тоді годинна тарифна ставка розробника:  
 = 8914,00 грн / 174 год = 51,23 грн/год.

Отже, витрати на основну оплату праці розробника згідно з формулою (5.1) становитимуть: = 400 год × 51,23 грн/год = 20492,00 грн.

Нарахування на заробітну плату (Єдиний соціальний внесок – ЄСВ) складають 22% від основної заробітної плати:

Загальні витрати на оплату праці розробника:  
ЗПзаг = + Знарах = 20492,00 грн + 4508,24 грн = 25000,24 грн.

## Розрахунок витрат на утримання та експлуатацію ПЕОМ

Припускаючи, що дія воєнного стану триватиме протягом усього 2025 року, річний фонд часу роботи ПЕОМ (ТПЕОМ) при 40-годинному робочому тижні становить 2088 годин. Цей показник буде використано для подальших розрахунків вартості експлуатації ПЕОМ.

Балансова вартість ПЕОМ (СБАЛ) обчислюється за формулою:

|  |  |
| --- | --- |
| СБАЛ = СРИН + ЗвСТ , | (2) |

де СРИН – ринкова вартість ПЕОМ, грн.;  
ЗвСТ – витрати на доставку й встановлення ПЕОМ, грн.

Приймаємо ринкову вартість ПЕОМ, необхідного для розробки програмного продукту (потужний ноутбук або настільний ПК), СРИН = 35000,00 грн. Витрати на доставку й встановлення ПЕОМ приймаємо у розмірі 10 % від ринкової вартості: ЗвСТ = 35000,00 грн × 0,10 = 3500,00 грн. Тоді балансова вартість ПЕОМ: СБАЛ = СРИН + ЗвСТ = 35000,00 грн + 3500,00 грн = 38500,00 грн.

Для нарахування амортизації використовуємо прямолінійний метод. Річні амортизаційні відрахування (ЗАМ) визначаються за формулою:

|  |  |
| --- | --- |
| ЗАМ = СБАЛ / Текспл. , | (3) |

де Текспл. – нормативний термін експлуатації ПЕОМ, років.  
Приймаємо термін експлуатації ПЕОМ Текспл. = 4 роки.  
Таким чином, річні амортизаційні відрахування складають:  
ЗАМ = 38500,00 грн / 4 роки = 9625,00 грн/рік.

Річні витрати на електроенергію (ЗЕЛ), споживану ПЕОМ, визначаються за формулою:

|  |  |
| --- | --- |
| ЗЕЛ = PПЕОМ × ТПЕОМ × СЕЛ × А , | (4) |

де PПЕОМ – сумарна потужність ПЕОМ, кВт;  
ТПЕОМ – річний фонд часу роботи ПЕОМ, год. (2088 год);  
СЕЛ – вартість 1 кВт·год електроенергії, грн. (приймаємо 4,32 грн/кВт·год згідно з теорією); А – коефіцієнт інтенсивного використання потужності ПЕОМ (приймаємо А = 0,85). Приймаємо сумарну потужність ПЕОМ PПЕОМ = 0,3 кВт (системний блок, монітор, периферія).  
ЗЕЛ = 0,3 кВт × 2088 год × 4,32 грн/кВт·год × 0,85 = 2297,90 грн/рік.

Інші витрати на експлуатацію ПЕОМ (ЗПР), що включають витрати на поточний ремонт, профілактичне обслуговування, оновлення програмного забезпечення тощо, приймаємо у розмірі 15% від балансової вартості ПЕОМ:  
ЗПР = СБАЛ × 0,15 = 38500,00 грн × 0,15 = 5775,00 грн/рік.

Повні річні витрати на експлуатацію ПЕОМ (З’ПЕОМ), без урахування зарплати програміста, яка розрахована окремо, визначаються як сума амортизаційних відрахувань, витрат на електроенергію та інших витрат на експлуатацію:

|  |  |
| --- | --- |
| ЗПЕОМ = ЗЗП + ЗАМ + ЗЕЛ + ЗПР , | (5) |

ЗПЕОМ = 9625,00 грн + 2297,90 грн + 5775,00 грн = 17697,90 грн/рік.

Вартість однієї години роботи ПЕОМ (СГОД) визначається за формулою:

|  |  |
| --- | --- |
| СГОД = ЗПЕОМ / ТПЕОМ , | (6) |

СГОД = 17697,90 грн / 2088 год = 8,48 грн/год.

Витрати на оплату машинного часу (), пов’язані з відлагодженням програмного продукту «Thrume», визначаються за формулою:

|  |  |
| --- | --- |
| = СГОД × | (7) |

де ТВ – фактичний час відлагодження програмного продукту на ПЕОМ, год.  
З таблиці 2, час на відлагоджування становить 50 годин.  
 = 8,48 грн/год × 50 год = 424,00 грн.

## Розрахунок загальновиробничих витрат

Загальновиробничі витрати включають витрати на освітлення, опалення, комунальні послуги, оренду приміщення (якщо це не власне), адміністративні витрати тощо. У даному розрахунку приймаємо їх у розмірі 30% від основної заробітної плати розробника:  
ЗЗАГ = × 0,30 = 20492,00 грн × 0,30 = 6147,60 грн.

## Обчислення собівартості програмного продукту «Thrume»

Повна собівартість (СПОВ) розробки програмного продукту «Thrume» визначається шляхом підсумовування всіх розрахованих статей витрат.

Таблиця 3 Собівартість програмного продукту за статтями витрат

|  |  |  |
| --- | --- | --- |
| № п/п | Статті витрат | Сума, грн. |
| 1 | Витрати на допоміжні матеріали | 955,00 |
| 2 | Основна заробітна плата розробника | 20492,00 |
| 3 | Нарахування на заробітну плату | 4508,24 |
| 4 | Витрати на оплату машинного часу | 424,00 |
| 5 | Загальновиробничі витрати | 6147,60 |
| 6 | Повна собівартість програмного продукту | 32526,84 |

Отже, повна собівартість розробки соціальної мережі «Thrume» складає 32526,84 грн.

## Обчислення гуртової ціни програмного продукту «Thrume»

Гуртова ціна програмного продукту «Thrume» визначається на основі повної собівартості та нормативного рівня рентабельності. Розрахунок проводиться за формулою:

|  |  |
| --- | --- |
| ЦРЕАЛ = СПОВ + Пр, | (8) |

де СПОВ – повна собівартість програмного продукту, грн.;  
Пр – прибуток, розрахований на основі нормативного рівня рентабельності, грн.

Прибуток визначається за формулою:

|  |  |
| --- | --- |
| Пр = СПОВ × (РН / 100) , | (9) |

де РН – нормативний рівень рентабельності, %.  
Приймаємо нормативний рівень рентабельності РН = 25%.  
Пр = 32526,84 грн × (25 / 100) = 32526,84 грн × 0,25 = 8131,71 грн.

Тоді гуртова ціна програмного продукту «Thrume»:  
ЦРЕАЛ = 32526,84 грн + 8131,71 грн = 40658,55 грн.

## Висновки щодо економічної доцільності розробки

Проведені розрахунки показують, що повна собівартість розробки соціальної мережі «Thrume» становить 32526,84 грн. Орієнтовна гуртова ціна програмного продукту, розрахована з урахуванням 25% рентабельності, складає 40658,55 грн.

Розробка соціальної мережі «Thrume» в рамках дипломного проекту є економічно обґрунтованою, оскільки дозволяє продемонструвати та закріпити практичні навички студента в галузі проектування, розробки та тестування складних програмних систем. Хоча в даному контексті не розглядається безпосередня комерційна реалізація, створений продукт має потенціал для подальшого розвитку та впровадження. Успішна розробка функціоналу аудіо/відео дзвінків та трансляції екрану може стати конкурентною перевагою та основою для створення затребуваного на ринку комунікаційного інструменту. Отримані економічні показники дають уявлення про порядок витрат на подібні проекти та можуть бути використані для подальшого планування.

# ОХОРОНА ПРАЦІ

## Аналіз потенційних небезпек при розробці

Розробка програмного забезпечення, зокрема соціальної мережі «Thrume» з функціями дзвінків та транслювання екрану, передбачає тривалу роботу за персональним комп’ютером в умовах офісного приміщення.

Потенційні небезпеки фізичного характеру:

* Підвищене значення напруги в електричному колі: Порушення ізоляції електропроводки ПК, монітора, блоків живлення або іншого офісного обладнання; помилкові дії персоналу при підключенні або ремонті; експлуатація несправного обладнання. Ураження електричним струмом різного ступеня важкості, від легкого поколювання до зупинки серця або дихання.
* Механічне травмування: Падіння важких предметів (наприклад, системного блоку з необережності), захаращеність проходів, нестійке розміщення обладнання, необережне пересування по офісу. Людина може отримати удари, порізи, переломи.
* Порушення зору: Тривала робота з монітором ПК, особливо з незадовільними ергономічними характеристиками (мерехтіння, неправильна яскравість/контрастність, відблиски), неправильна організація робочого місця (відстань до екрана, освітлення). Це може призвести до втоми очей, зниження гостроти зору, синдром «сухого ока», головний біль.

Потенційні небезпеки хімічного характеру:

* Виділення шкідливих речовин в повітря робочої зони: Нагрівання та робота компонентів ПК та оргтехніки (принтери, копіри) може призводити до виділення в малих кількостях озону, стиролу, формальдегіду та інших летких органічних сполук, особливо при недостатній вентиляції, накопичення пилу. Це може призвести до небажанних наслідків, таких як подразнення дихальних шляхів, алергічні реакції, головний біль, загальне погіршення самопочуття при тривалому впливі.

Потенційні небезпеки психофізіологічного характеру:

* Нервово-емоційне перенапруження: Висока інтенсивність розумової праці, необхідність обробки великих обсягів інформації, відповідальність за результат, стислі терміни виконання завдань (дедлайни), можливі конфліктні ситуації. Це викликає стрес, роздратованість, зниження концентрації уваги, професійне вигорання.
* Монотонність праці: Виконання однотипних операцій протягом тривалого часу (наприклад, написання коду, тестування). Це знижує інтерес до роботи, нудьга, зниження уваги, підвищення ймовірності помилок.
* Тривала статична робоча поза та нераціональна організація робочого місця: Тривале сидіння в одній позі, незручні меблі (стіл, стілець), неправильне розташування монітора та клавіатури. Людина може отримати болі в спині, шиї, плечах; розвиток остеохондрозу, захворювань опорно-рухового апарату; навантаження на кисті рук (тунельний синдром).

Потенційні небезпеки, пов’язані з порушеннями санітарно-гігієнічних умов:

* Незадовільні параметри мікроклімату: Невідповідність температури, відносної вологості, швидкості руху повітря встановленим нормам через несправність або відсутність систем опалення, вентиляції та кондиціонування. Небажані наслідки включать перегрівання або переохолодження організму, зниження працездатності, підвищення ризику простудних захворювань.
* Незадовільний рівень освітленості: Недостатнє природне або штучне освітлення, неправильний вибір або розміщення освітлювальних приладів, наявність відблисків на екрані. Це призводить до перенапруження зору, швидка втомлюваність, зниження продуктивності праці.
* Підвищений рівень шуму: Робота офісної техніки (ПК, принтери, кондиціонери), розмови колег, шум з вулиці. Як висновок, можливо буде підвищена стомлюваність, зниження концентрації уваги, головний біль, роздратованість.

Потенційні небезпеки, пов’язані з порушеннями правил пожежної безпеки:

* Виникнення пожежі: Коротке замикання в електропроводці або обладнанні, перегрів електронних компонентів ПК, використання несправних електроприладів, необережне поводження з вогнем (якщо таке допускається), захаращення шляхів евакуації, відсутність або несправність первинних засобів пожежогасіння. Це загрожує опіками, отруєнням продуктами горіння, матеріальними збитки, загрозою життю.
* Неправильний вибір або недостатня кількість первинних засобів пожежогасіння: Не враховано клас можливої пожежі (для офісів з ПК це класи А та Е), недостатня кількість вогнегасників відповідно до площі приміщення та кількості обладнання. Неможливість ефективно загасити пожежу на початковій стадії, швидке поширення вогню.
  1. **Характеристика приміщення**

Для забезпечення належних умов праці розробників соціальної мережі «Thrume» необхідно правильно організувати робочий простір в офісному приміщенні. Це включає визначення оптимальної кількості робочих місць, їх раціональне розміщення та забезпечення необхідними засобами пожежогасіння.

Дано наступні параметри приміщення: a = 13,3, b = 9,4 м, h = 4,6 м.

𝑆пр = a×b = 13,3×9,4=125,02м2

𝑉пр = a×b×h=13,3×9,4×4,6=575,092м3

Для розрахунку кількості робочих місць, необхідно врахувати площу та об’єм, що займає устаткування. Використаємо типові розміри офісних меблів для комп'ютеризованого робочого місця, наведені у методичних вказівках.

Таблиця 3.1 Параметри устаткування

|  |  |  |  |
| --- | --- | --- | --- |
| Параметри устаткування | Стіл | Крісло | Шафа |
| Довжина, (м) | 1,2 | 0,45 | 0,6 |
| Ширина, (м) | 0,9 | 0,45 | 0,4 |
| Висота, (м) | 0,725 | 0,4 | 1,8 |
| Займана площа, (м2) | 1,08 | 0,2025 | 0,24 |
| Займаний обсяг (м3) | 0,783 | 0,081 | 0,432 |

Площа устаткування, яке розташоване на одному робочому місці 𝑆уст: 𝑆уст=𝑆стола+𝑆крісла+0,5×𝑆шафи=1,08+0,2025+0,5×0,24=1,4025м2

Обсяг устаткування, яке розташоване на одному робочому місці 𝑉уст: 𝑉уст=𝑉стола+𝑉крісла+0,5×𝑉шафи=0,783+0,08+0,5×0,432=1,08м3

Згідно з ДСанПіН 3.3.2.007-98 та НПАОП 0.00-7.15-18, на одне робоче місце, обладнане відеодисплейним терміналом (ВДТ), повинно припадати не менше ніж 6 м2 вільної площі та 20 м2 вільного об’єму приміщення.

Максимально можлива кількість робочих місць за площею (Nр.м. по 𝑆): Nр.м. по 𝑆=𝑆пр/(𝑆уст+6)=125,02/(1,4025+ 6)=125,02/7,4025 = 16,88.

Максимально можлива кількість робочих місць за об’ємом (Nр.м. по 𝑉): Nр.м. по 𝑉=𝑉пр/(𝑉уст+20)=575,092/(1,08+20)=575,092/21,08=27,28

З отриманих результатів приймаємо мінімальне значення. Оскільки для комфортного розміщення з дотриманням усіх нормативних відстаней (як буде показано далі) 16 робочих місць може бути занадто щільно, і після аналізу планування оптимальною виявилася кількість 15 робочих місць.

Згідно з «Правилами експлуатації та типовими нормами належності вогнегасників» та НОРМОЮ № 6 «Належності газових вогнегасників для об’єктів України» (Примітка 6), приміщення, у яких розміщені електронно-обчислювальні машини (ЕОМ), слід оснащувати переносними газовими вогнегасниками з розрахунку один вогнегасник ВВК-1,4 (або аналогічний за вогнегасною здатністю, наприклад ВВК-2) на три ЕОМ, але не менше ніж один вогнегасник зазначеного типу на приміщення [21, НОРМА №6].

Кількість вогнегасників Nвогн для 15 робочих місць:

Nвогн=Nр.м./3=15/3=5

Отже, необхідно 5 вогнегасників типу ВВК-1,4 (або ВВК-2).

При розміщенні робочих місць необхідно дотримуватись таких вимог (ДСанПіН 3.3.2.007-98, НПАОП 0.00-7.15-18):

* Робочі місця з ВДТ розміщуються на відстані не менше 1м від стін зі світловими прорізами (вікнами).
* Відстань між бічними поверхнями ВДТ має бути не меншою за 1,2м.
* Відстань між тильною поверхнею одного ВДТ та екраном іншого не повинна бути меншою за 2м.
* Прохід між рядами робочих місць має бути не меншим ніж 1м (якщо це не прохід між тилом одного ВДТ та екраном іншого, де вимога 2м).

Для приміщення розмірами 13,3м×9,4м та 15 робочих місць пропонується наступне планування: 3 ряди по 5 робочих місць. Столи розміром 1,2м (ширина) × 0,9м (глибина). Вікна розташовані вздовж однієї з довших стін (13,3м).

Розрахунок розміщення по довжині приміщення (13,3м) наступний: П’ять столів у кожному ряду займатимуть загальну ширину 5 столів × 1,2 м/стіл = 6м. Між цими столами необхідно забезпечити чотири проходи, кожен шириною не менше 1,2м (відстань між бічними поверхнями ВДТ), що сумарно становить 4 проходи × 1,2 м/прохід = 4,8м. Таким чином, загальна довжина, яку займають столи та проходи між ними в одному ряду, складе 6,0м+4,8м = 10,8м. Решта простору по довжині приміщення розподіляється на проходи вздовж стін по краях рядів: (13,3м-10,8м)/2=1,25м з кожного боку, що відповідає нормативним вимогам.

Розрахунок розміщення по ширині приміщення (9,4 м) враховує три ряди столів. Перший ряд столів розміщується на відстані 1,0 м від стіни з вікнами. Глибина столу становить 0,9 м. Між тильною поверхнею столів першого ряду та екранами столів другого ряду забезпечується відстань 2,0 м. Глибина столу другого ряду – 0,9 м. Аналогічно, між тильною поверхнею столів другого ряду та екранами столів третього ряду витримується відстань 2,0 м, а глибина столу третього ряду становить 0,9 м. Сумарна глибина, яку займають три ряди столів та основні проходи між ними і від вікон, складає 1,0 м + 0,9 м + 2,0 м + 0,9 м + 2,0 м + 0,9 м = 7,7 м. Відповідно, прохід від тильної сторони третього ряду столів до протилежної стіни становитиме 9,4 м - 7,7 м = 1,7 м, що є достатнім для вільного пересування.

Шафи для документації (15 р.м. / 2 р.м./шафа ≈ 8 шаф) розміщуються вздовж стін у вільно доступних місцях.  
Вогнегасники (5 шт. ВВК-1,4 або ВВК-2) розміщуються рівномірно по приміщенню, в легкодоступних місцях.

Изображение выглядит как диаграмма, План, Прямоугольник, Технический чертеж

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.1 План виробничого приміщення з комп’ютеризованими робочими місцями: — комп’ютеризоване робоче місце з ВДТ, 8 — шафи для зберігання документації, 3 – сонцезахисні жалюзі, 5 — вогнегасник.

Таке планування забезпечує дотримання нормативних вимог, раціональне використання площі та створення комфортних умов для роботи.

* 1. **Заходи щодо поліпшення умов з охорони праці**

На основі проведеного аналізу потенційних небезпек та шкідливих факторів, розробляються заходи, спрямовані на створення безпечних, здорових та комфортних умов праці для розробників соціальної мережі «Thrume». Ці заходи базуються на вимогах чинного законодавства України з охорони праці та виробничої санітарії.

Організаційні заходи:

* Навчання та інструктажі: Згідно з вимогами НПАОП 0.00-4.12-05 «Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці», для всіх працівників, залучених до розробки «Thrume», необхідно проводити:
* Вступний інструктаж при прийомі на роботу.
* Первинний інструктаж на робочому місці.
* Повторні інструктажі (не рідше одного разу на 6 місяців для робіт з ПК).
* Позапланові та цільові інструктажі за необхідності.
* Навчання та перевірку знань з питань електробезпеки для присвоєння І групи з електробезпеки користувачам ПК.
* Навчання правилам пожежної безпеки та діям у разі виникнення пожежі.
* Навчання наданню домедичної допомоги потерпілим від нещасних випадків [15].
* Режим праці та відпочинку: Встановити раціональний режим праці та відпочинку згідно з ДСанПіН 3.3.2.007-98 та НПАОП 0.00-7.15-18. Для розробників програмного забезпечення передбачити регламентовані перерви тривалістю 15 хвилин через кожні дві години роботи за комп'ютером. Під час перерв рекомендується виконувати вправи для очей та фізичні вправи. Робота над складними модулями, такими як функції дзвінків та транслювання екрану в «Thrume», може вимагати додаткових коротких перерв для збереження концентрації.
* Медичні огляди: Організувати проведення попередніх та періодичних медичних оглядів працівників, які працюють з ВДТ.
* Контроль за дотриманням вимог ОП: Здійснювати регулярний контроль за дотриманням працівниками вимог інструкцій з охорони праці.

 Технічні заходи:

* Все електрообладнання повинно бути справним, мати непошкоджену ізоляцію та відповідати вимогам ПУЕ.
* Передбачити захисне заземлення (занулення) металевих неструмоведучих частин електрообладнання. Опір контуру захисного заземлення для електроустановок до 1000 В у приміщеннях з ПК не повинен перевищувати 4 Ом.
* Використовувати розетки із заземлюючим контактом.
* Забезпечити працівників ергономічними меблями: регульованими стільцями, столами достатньої площі.
* Монітори повинні відповідати сучасним вимогам безпеки, мати можливість регулювання. Символи на екрані мають бути чіткими, зображення стабільним [НПАОП 0.00-7.15-18].
* Клавіатура та миша повинні розташовуватися ергономічно.
* Захист від механічних травм: Забезпечити стійке розміщення обладнання, вільні проходи.

Заходи щодо виробничої санітарії та гігієни праці:

* Підтримувати оптимальні параметри мікроклімату (температура, вологість, швидкість руху повітря) згідно з нормами для категорії робіт Іб: у холодний період: температура 21-23°С, вологість 40-60%, швидкість повітря ≤ 0,1 м/с; у теплий період: температура 22-24°С, вологість 40-60%, швидкість повітря ≤ 0,2 м/с.
* Забезпечити ефективну роботу систем опалення, вентиляції та кондиціонування (ДБН В.2.5-67:2013). Рекомендується 3-кратний обмін повітря за годину.
* Забезпечити комбіноване освітлення. КПО ≥ 1,5%. Вікна обладнати сонцезахисними пристроями.
* Штучне освітлення: освітленість на столі 300-500 лк. Використовувати світильники з низькою пульсацією. Обмежити блискість [ДБН В.2.5-28:2018].
* Рівень шуму на робочих місцях ≤ 50-55 дБА.
* Заходи: малошумна техніка, раціональне розміщення обладнання, звукопоглинаючі матеріали.

Заходи з пожежної безпеки:

* Можливі класи пожежі: А (тверді речовини), Е (електроустановки).
* Категорія приміщення за пожежною небезпекою: «Д».
* Ступінь вогнестійкості будівлі: Не нижче II.
* Системи виявлення пожежі та оповіщення: Обладнати АСПС та СОУЕ.
* Як розраховано в п. 4.2.4, необхідно 5 переносних газових (вуглекислотних) вогнегасників ВВК-1,4 (або ВВК-2).
* Вогнегасники розміщувати в легкодоступних місцях.
* Шляхи евакуації: Утримувати вільними. Розробити та вивісити плани евакуації.
* Інші заходи: Заборонити куріння (крім спецмісць), відключати електроприлади після роботи.

Впровадження цих заходів дозволить значно знизити ризики та забезпечити безпечні умови праці.

## Інструкція з охорони праці при роботі з ПЕОМ та відеодисплейними терміналами

### Загальні положення

* Дія цієї інструкції поширюється на всі підрозділи підприємства, де виконуються роботи з персональною електронно-обчислювальною машиною (далі – ПЕОМ або ПК). Інструкція розроблена відповідно до Положення про розробку інструкцій з охорони праці, затвердженого наказом Держпраці від 29.01.1998 № 9, Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці, затвердженого наказом Держпраці від 26.01.2005 № 15, Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями, затверджених наказом Мінсоцполітики від 14.02.2018 № 207, Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98, затверджених постановою Головного державного санітарного лікаря України від 10.12.1998 № 7, та Загальних вимог стосовно забезпечення роботодавцями охорони праці працівників, затверджених наказом ДСНС від 25.01.2012 № 67 (НПАОП 0.00-7.11-12).
* За цією інструкцією працівника, який використовує ПК (далі – користувач), інструктують перед початком роботи (первинний інструктаж), а потім через кожні 6 місяців (повторний інструктаж). Результати інструктажу заносять до Журналу реєстрації інструктажів з питань охорони праці на робочому місці, де має бути підпис особи, яка інструктує, та користувача. Користувач зобов’язаний дбати про особисту безпеку і здоров’я, а також про безпеку і здоров’я довколишніх при виконанні будь-яких робіт та під час перебування на території підприємства. До роботи на персональному комп’ютері допускаються особи, які пройшли відповідні інструктажі з питань охорони праці та пожежної безпеки.
* Користувач зобов’язаний виконувати правила внутрішнього трудового розпорядку; не допускати на своє робоче місце сторонніх осіб; не виконувати вказівок, які суперечать правилам охорони праці та пожежної безпеки; знати правила надання домедичної допомоги; знати розташування та вміти користуватись первинними засобами пожежогасіння; вміти працювати з ПК. Основними небезпечними та шкідливими виробничими факторами, що можуть впливати на користувача, є підвищений рівень статичної електрики, нерівномірність розподілу яскравості в полі зору, підвищена яскравість світлового зображення, ризик ураження електричним струмом, напруження зору та уваги, тривалі статичні навантаження.
* У приміщеннях із ПК має бути забезпечене природне і штучне освітлення. При розміщенні робочих місць необхідно унеможливити пряме засвічування екрана природним освітленням, для чого слід передбачити наявність сонцезахисних засобів (плівка, жалюзі, штори тощо). Світлові відблиски з клавіатури, екрана та інших частин ПК у напрямку очей користувача є неприпустимими. Основним обладнанням робочого місця є ПК або ноутбук, монітор, клавіатура, маніпулятор, робочий стіл та стілець (крісло). При розміщенні елементів робочого місця слід враховувати робочу позу користувача, простір для його розміщення, можливість огляду елементів робочого місця та простору поза ним, а також можливість робити записи та розміщувати на робочому столі необхідну документацію. Розміщення елементів робочого місця не повинно заважати рухам та переміщенню для експлуатації ПК.
* Монітор встановлюють так, щоб відстань від поверхні екрана до очей користувача становила 600–700 мм залежно від розміру екрана. Клавіатуру розміщують на робочому або окремому столі на відстані 100–300 мм від краю з боку користувача; її положення та кут нахилу (зазвичай 5–15°) залежать від побажання користувача, при цьому не допускається хитання клавіатури. Конструкція робочого столу має забезпечувати оптимальне розміщення на робочій поверхні обладнання з урахуванням його кількості, розмірів, конструктивних особливостей та характеру роботи. Крісло повинно підтримувати раціональну робочу позу під час виконання основних виробничих операцій та давати можливість змінювати позу; тип крісла обирають залежно від характеру та тривалості роботи.
* Раціональна поза користувача передбачає, що ступні розташовані на підлозі або на підставці для ніг; стегна зорієнтовані у горизонтальній площині; верхні ділянки рук вертикальні; кут ліктьового суглоба становить 70–90°; зап’ястя зігнуті під кутом не більше ніж 20°; нахил голови – у межах 15–20°, часті її повороти виключені. Для забезпечення оптимальної робочої пози необхідно засоби праці, з якими користувач має тривалий або найбільш частий зоровий контакт, розмістити у центрі зони зорового спостереження та моніторного поля, а також забезпечити відстань близько 500 мм між найважливішими засобами праці, з якими користувач працює найчастіше.
* ПК встановлюють на рівній твердій поверхні (столі), не допускаючи встановлення ПК та оргтехніки на хитких підставках чи похилих поверхнях. ПК не слід встановлювати впритул до стіни чи перегородки, а також не можна загороджувати вентиляційні отвори ПК сторонніми предметами. Розетка біля ПК має бути в доступному місці для своєчасного відімкнення в аварійних випадках; використання подовжувачів не рекомендовано. Під час переміщення ПК та периферійних пристроїв необхідно витягти вилку живлення з розетки. Не допускається ушкодження чи модифікування шнура живлення; заборонено ставити важкі речі на шнур живлення, тягнути чи надмірно перегинати його, скручувати та зав’язувати у вузол. ПК під’єднують до електромережі лише за допомогою справних штепсельних з’єднань та електророзеток заводського виробництва, які мають бути зі спеціальними контактами для під’єднання нульового захисного провідника. Їхня конструкція повинна забезпечувати з’єднання нульового захисного провідника раніше, ніж з’єднання фазового та нульового робочого провідників, а порядок роз’єднань при вимкненні має бути зворотнім. Заборонено під’єднувати електрообладнання до звичайної двошнурової електромережі. За невиконання цієї інструкції працівники несуть відповідальність згідно з чинним законодавством [16].

### Вимоги безпеки перед початком роботи

* Перед початком роботи необхідно увімкнути систему кондиціювання повітря в приміщенні, якщо вона передбачена. Слід ретельно оглянути робоче місце та привести його в порядок, переконавшись у відсутності сторонніх предметів та належному з’єднанні всіх блоків ПЕОМ з системним блоком. Важливо перевірити надійність встановлення апаратури на робочому столі; ВДТ має стояти не на краю стола. ВДТ слід повернути так, щоб було зручно дивитися на екран – під прямим кутом (а не збоку) і трохи зверху вниз, при цьому екран має бути трохи нахиленим (нижній його край ближче до користувача). Необхідно перевірити загальний стан апаратури, справність електропроводки, з’єднувальних шнурів, штепсельних вилок, розеток та заземлення захисного екрана.
* Забороняється експлуатація кабелів та проводів з пошкодженою або такою, що втратила захисні властивості ізоляцією; залишення під напругою кабелів та проводів з неізольованими провідниками; застосування саморобних подовжувачів, які не відповідають вимогам Правил будови електроустановок; застосування для опалення приміщення нестандартного (саморобного) електронагрівального обладнання або ламп розжарювання; користування пошкодженими розетками, з’єднувальними коробками, вимикачами, а також лампами зі слідами затемнення або випинання скла; підвішування світильників безпосередньо на струмопровідних проводах, обгортання електроламп і світильників папером, тканиною та іншими горючими матеріалами, експлуатація їх зі знятими ковпаками (розсіювачами); використання електроапаратури та приладів в умовах, що не відповідають вказівкам підприємств-виготовлювачів.
* Слід відрегулювати освітленість робочого місця та зафіксувати висоту крісла і зручний нахил його спинки. У разі необхідності, до процесора приєднується потрібна апаратура (принтер, сканер тощо), при цьому всі кабелі, що з’єднують системний блок з іншими пристроями, слід вмикати та вимикати тільки при вимкненому комп’ютері. Апаратуру комп’ютера вмикають вимикачами на корпусах у такій послідовності: стабілізатор напруги (якщо є), ВДТ, процесор, принтер (якщо передбачається друкування). Після ввімкнення необхідно відрегулювати яскравість свічення екрана ВДТ, мінімальний розмір світної точки, фокусування та контрастність, не роблячи зображення занадто яскравим. Рекомендовані параметри: яскравість свічення екрана – не менше 100 кд/м²; відношення яскравості екрана ВДТ до яскравості оточуючих поверхонь – не більше 3:1; мінімальний розмір точки свічення – не менше 0,4 мм для монохромного ВДТ і 0,6 мм для кольорового; контрастність зображення знаку – не менше 0,8. У разі виявлення будь-яких несправностей роботу розпочинати заборонено; про це слід повідомити керівника робіт.

### Вимоги безпеки під час роботи

* Під час роботи необхідно стійко розташувати клавіатуру на робочому столі, не допускаючи її хитання, при цьому має бути передбачена можливість її поворотів та переміщень. Положення клавіатури та кут її нахилу повинні відповідати побажанням користувача. Якщо в конструкції клавіатури не передбачено простору для опори долонь, їх слід розташовувати на відстані не менше 100 мм від краю столу в оптимальній зоні моторного поля. Під час роботи на клавіатурі слід сидіти прямо, не напружуючись. Для зменшення несприятливого впливу пристроїв типу «миша» (вимушена поза, постійний контроль дій) належить забезпечити вільну велику поверхню столу для переміщення «миші» та зручного упору ліктьового суглоба.
* Не дозволяються сторонні розмови та подразнюючі шуми. Періодично, при вимкненому комп’ютері, слід прибирати ледь змоченою мильним розчином бавовняною ганчіркою пил з поверхонь апаратури. Забороняється самостійно ремонтувати апаратуру, особливо ВДТ, де кінескоп знаходиться під високою напругою (близько 25 кВ); ремонт виконують тільки спеціалісти з технічного обслуговування, які також раз на півроку повинні відкривати процесор і видаляти пил та бруд пилососом. Не можна класти будь-які предмети на апаратуру комп’ютера, а також напої на клавіатуру або поруч з нею, оскільки це може вивести їх з ладу. Для зняття статичної електрики рекомендується час від часу доторкатися до металевих поверхонь (наприклад, батареї центрального опалення).
* Для зниження напруженості праці на ПЕОМ необхідно рівномірно розподіляти і чергувати характер робіт відповідно до їх складності. З метою зменшення негативного впливу монотонності доцільно застосовувати чергування операцій введення тексту та числових даних (зміна змісту робіт), чергування редагування текстів та введення даних (зміна змісту та темпу роботи) тощо.

### Вимоги безпеки після закінчення роботи

* Після закінчення роботи необхідно коректно завершити та записати у пам’ять комп’ютера файл, що знаходиться в роботі, вийти з програмної оболонки та повернутися в середовище операційної системи (якщо це передбачено). Далі слід вимкнути принтер та інші периферійні пристрої, потім вимкнути ВДТ і процесор. Якщо комп’ютер підключений до мережі через стабілізатор, його також потрібно вимкнути. Штепсельні вилки рекомендується витягнути з розеток. Клавіатуру варто накрити кришкою для запобігання потраплянню пилу.
* Необхідно прибрати робоче місце, покласти оригінали та інші документи у відповідне місце зберігання (наприклад, ящик стола). Після цього слід ретельно вимити руки теплою водою з милом. Якщо передбачено, можна вимкнути кондиціонер, а також освітлення і загальне електроживлення підрозділу, якщо працівник залишає приміщення останнім. Рекомендується, за наявності спеціально обладнаного приміщення, провести сеанс психофізіологічного розвантаження та зняття втоми з виконанням спеціальних вправ аутотренінгу.

### Вимоги безпеки в аварійних ситуаціях

Аварійна ситуація чи нещасний випадок можуть статися в разі ураження електричним струмом, загорання апаратури тощо. У разі раптового припинення подавання електроенергії необхідно вимкнути комп’ютер у такій послідовності: периферійні пристрої, ВДТ, процесор, стабілізатор напруги (якщо є), після чого витягнути штепсельні вилки з розеток. При виявленні ознак горіння (дим, запах гару) слід негайно вимкнути апаратуру, визначити джерело займання та вжити заходів щодо його ліквідації, повідомити керівника робіт та не допускати в небезпечну зону сторонніх осіб. Якщо є потерпілі, необхідно надавати їм першу домедичну допомогу та, за потреби, викликати швидку медичну допомогу.

# ВИСНОВКИ

В ході виконання дипломного проекту було успішно досягнуто поставлену мету щодо розробки соціальної мережі «Thrume». Оцінка технічних рішень показує, що обрані технології, зокрема C# та ASP.NET для бекенду і PostgreSQL для бази даних, разом із розробленою архітектурою, продемонстрували свою ефективність та відповідність вимогам проекту. Ефективність структури даних виявилася високою, оскільки спроектована база даних забезпечила оптимальну організацію та надійне зберігання інформації користувачів, їхніх взаємодій та контенту. Надійність системи була підтверджена результатами проведеного тестування, які продемонстрували стабільність роботи та коректне функціонування всіх модулів. Переваги перед аналогами полягають у комплексній інтеграції функцій соціальної взаємодії з інструментами для дзвінків та транслювання екрану в рамках єдиної платформи. Практична цінність розробленого продукту визначається широкими можливостями його застосування для корпоративного спілкування, освітніх цілей та об'єднання користувачів за інтересами. Перспективи розвитку проекту включають подальше розширення функціональності, розробку мобільних клієнтів та інтеграцію з іншими популярними сервісами. Набутий досвід під час роботи над проектом полягає у значному професійному розвитку, зокрема у вдосконаленні навичок проектування та розробки складних програмних систем та подоланні різноманітних технічних викликів.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wikipedia. C# – Вікіпедія. URL: https://uk.wikipedia.org/wiki/C\_Sharp (дата звернення: 16.05.2025).
2. Microsoft Learn. Обзор ASP.NET. URL: https://learn.microsoft.com/ru-ru/aspnet/overview (дата звернення: 16.05.2025).
3. Microsoft Learn. Загальні відомості про ASP.NET Core. URL: https://learn.microsoft.com/uk-ua/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-9.0 (дата звернення: 16.05.2025).
4. Вікіпедія. PostgreSQL — Вікіпедія. URL: https://uk.wikipedia.org/wiki/PostgreSQL (дата звернення: 16.05.2025).
5. Baeldung. Статті та підручники з програмування – Baeldung. URL: https://www.baeldung.com/ (дата звернення: 16.05.2025).
6. JetBrains. Інструменти для розробників – JetBrains. URL: https://www.jetbrains.com/ (дата звернення: 16.05.2025).
7. GeeksforGeeks. Статті та підручники з програмування – GeeksforGeeks. URL: https://www.geeksforgeeks.org/ (дата звернення: 16.05.2025).
8. Medium. Статті з різних тем – Medium. URL: https://medium.com/ (дата звернення: 16.05.2025).
9. GitHub. Хостинг для розробників – GitHub. URL: https://github.com/ (дата звернення: 16.05.2025).
10. Основи охорони праці: Підручник / К.Н.Ткачук, М.О.Халімовський, В.В.Зацарний та ін. – К.: Основа, 2006 – 448 с.
11. Запорожець О.І., Протоєрейський О.С., Франчук Г.М., Боровик І. М. Основи охорони праці. Підручник. – К.: Центр учбової літератури, 2009. – 264 с.
12. Закон України «Про охорону праці».
13. Закон України «Про пожежну безпеку» (або актуальний Кодекс цивільного захисту України).
14. Закон України «Про забезпечення санітарного та епідемічного благополуччя населення».
15. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу. Затверджено наказом МОЗ України від 08.04.2014 № 248.
16. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями». Затверджено наказом Міністерства соціальної політики України від 14.02.2018 № 207.
17. ДСанПіН 3.3.2.007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин». Затверджено постановою Головного державного санітарного лікаря України від 10.12.1998 № 7.
18. ДБН В.2.5-28:2018 «Природне і штучне освітлення».
19. ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку».
20. НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-обчислювальних машин». Наказ Держгірпромнагляду від 26.03.2010р. № 65
21. НПАОП 0.00-4.12-05 «Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці».
22. НПАОП 0.00-4.15-98 «Положення про розробку інструкцій з охорони праці».
23. Правила улаштування електроустановок (ПУЕ).
24. НАПБ А.01.001-2014 «Правила пожежної безпеки в Україні».
25. ДСТУ EN 2:2014 «Класифікація пожеж».
26. ДСТУ Б В.1.1-36:2016 «Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою».
27. ДБН В.1.1-7:2016 «Пожежна безпека об’єктів будівництва. Загальні вимоги».
28. ДБН В.2.5-56:2014 «Системи протипожежного захисту».
29. ДБН В.2.5-67:2013 «Опалення, вентиляція та кондиціонування».
30. Правила експлуатації та типові норми належності вогнегасників. Затверджено наказом МВС України від 15.01.2018 № 25. (Включаючи НОРМУ №6).

# Додаток 1 Фізична схема

Изображение выглядит как текст, диаграмма, План, Параллельный

Контент, сгенерированный ИИ, может содержать ошибки.

# Додаток 2 Лістинг програми

{

"$schema": "https://json.schemastore.org/launchsettings.json",

"profiles": {

"http": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": false,

"applicationUrl": "http://localhost:5132",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"https": {

"commandName": "Project",

"dotnetRunMessages": true,

"launchBrowser": false,

"applicationUrl": "https://localhost:5133",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

}

}

using System.Security.Claims;

using Microsoft.AspNetCore.Antiforgery;

using Microsoft.AspNetCore.Authorization;

using Thrume.Database;

using Thrume.Domain.EntityIds;

using Thrume.Services;

namespace Thrume.Api.Endpoints;

public static class AccountEndpoints

{

public static IEndpointRouteBuilder MapAccountEndpoints(this IEndpointRouteBuilder app)

{

var accountGroup = app.MapGroup("/account").RequireAuthorization();

accountGroup.MapPut("/updateProfile", async (IFormFile file,IAntiforgery antiforgery, HttpContext context, AccountService service) =>

{

await antiforgery.ValidateRequestAsync(context);

var findFirst = context.User.FindFirst(ClaimTypes.NameIdentifier)?.Value;

if (findFirst is null)

{

return Results.Unauthorized();

}

await service.UpdateAvatarAsync(new AccountId(Guid.Parse(findFirst)), file);

return Results.Ok();

}).Accepts<IFormFile>("multipart/form-data");

accountGroup.MapGet("/me", async (HttpContext context, AppDbContext dbContext, ILoggerFactory factory) =>

{

var value = context.User.FindFirst(ClaimTypes.NameIdentifier)?.Value;

ILogger logger = factory.CreateLogger<Program>();

logger.LogError("context claims {}", value);

if (value is null) return Results.Unauthorized();

if (!Guid.TryParse(value, out var guid))

{

return Results.Unauthorized();

}

var findAsync = await dbContext.AccountDbSet.FindAsync(new AccountId(guid));

if (findAsync is null) return Results.Unauthorized();

return Results.Ok(findAsync);

});

return app;

}

}

UsingSystem.Collections;  
using System.Security.Claims;  
using Microsoft.AspNetCore.Antiforgery;  
using Microsoft.AspNetCore.Authorization;  
using Microsoft.AspNetCore.Mvc;  
using Thrume.Domain.EntityIds;  
using Thrume.Services;  
  
namespace Thrume.Api.Endpoints;  
  
internal record CreateCommentPayload(PostId PostId, string Content, CommentId? ParentCommentId);  
  
public static class CommentEndpoints  
{  
 public static IEndpointRouteBuilder MapCommentEndpoints(this IEndpointRouteBuilder app)  
 {  
 var commentsGroup = app.MapGroup("/comments").RequireAuthorization();  
   
 commentsGroup.MapPost("/", async (  
 HttpContext context,  
 [FromBody] CreateCommentPayload payload,  
 CommentService commentService, IAntiforgery antiforgery) =>  
 {  
 await antiforgery.ValidateRequestAsync(context);  
 var userIdClaim = context.User.FindFirst(ClaimTypes.**NameIdentifier**);  
 if (userIdClaim is null || !Guid.TryParse(userIdClaim.Value, out var userIdGuid))  
 {  
 return Results.Unauthorized();  
 }  
 var authorId = new AccountId(userIdGuid);  
   
 if (string.IsNullOrWhiteSpace(payload.Content) || payload.Content.Length > 1000)  
 {  
 return Results.BadRequest("Comment content is invalid or too long.");  
 }  
 await commentService.CreateCommentAsync(payload.PostId, authorId, payload.Content, payload.ParentCommentId);  
  
 return Results.Ok();   
 });  
   
 commentsGroup.MapDelete("/", async (  
 HttpContext context,  
 [FromBody] CommentId commentId,   
 CommentService commentService) =>  
 {  
 var userIdClaim = context.User.FindFirst(ClaimTypes.**NameIdentifier**);  
 if (userIdClaim is null || !Guid.TryParse(userIdClaim.Value, out var userIdGuid))  
 {  
 return Results.Unauthorized();   
 }  
 var userId = new AccountId(userIdGuid);  
  
  
 await commentService.DeleteCommentAsync(commentId, userId);  
  
  
 return Results.NoContent();  
 });  
  
 return app;  
 }  
}

using System.Security.Claims;

using Microsoft.AspNetCore.Mvc;

using Thrume.Domain.EntityIds;

using Thrume.Services;

using Microsoft.AspNetCore.Antiforgery;

namespace Thrume.Api.Endpoints;

internal record StartConversationPayload(AccountId OtherUserId);

internal record SendMessagePayload(ConversationId Id, string Content);

internal record ConversationListItemResponse(ConversationId Id, List<AccountId> ParticipantIds, DateTimeOffset CreatedAt);

internal record MessageResponse(MessageId Id, ConversationId ConversationId, AccountId SenderId, string Content, DateTimeOffset SentAt);

public static class MessageEndpoints

{

public static IEndpointRouteBuilder MapMessageEndpoints(this IEndpointRouteBuilder app)

{

var messagesGroup = app.MapGroup("/messages").RequireAuthorization();

messagesGroup.MapGet("/conversations", async (HttpContext context, MessageService messageService) =>

{

var userId = GetCurrentUserId(context);

if (userId == null) return Results.Unauthorized();

var conversations = await messageService.GetConversationsAsync(userId.Value);

var response = conversations.Select(c =>

new ConversationListItemResponse(

c.Id,

c.Participants.Select(p => p.Id).ToList(),

c.CreatedAt)

).ToList();

return Results.Ok(response);

});

messagesGroup.MapPost("/conversations/start", async (

HttpContext context,

[FromBody] StartConversationPayload payload,

MessageService messageService, IAntiforgery antiforgery) =>

{

await antiforgery.ValidateRequestAsync(context);

var userId = GetCurrentUserId(context);

if (userId == null) return Results.Unauthorized();

var conversation = await messageService.StartOrGetConversationAsync(userId.Value, payload.OtherUserId);

if (conversation == null)

{

return Results.BadRequest("Unable to start conversation.");

}

var response = new ConversationListItemResponse(

conversation.Id,

conversation.Participants.Select(p => p.Id).ToList(),

conversation.CreatedAt);

return Results.Ok(response);

});

messagesGroup.MapGet("/conversations/{conversationId}", async (

HttpContext context,

[FromBody] ConversationId conversationId,

[FromQuery] int page,

[FromQuery] int pageSize,

MessageService messageService) =>

{

var userId = GetCurrentUserId(context);

if (userId == null) return Results.Unauthorized();

var messages = await messageService.GetMessagesAsync(conversationId, userId.Value, page, pageSize);

var response = messages.Select(m =>

new MessageResponse(m.Id, m.ConversationId, m.SenderId, m.Content, m.SentAt)

).ToList();

return Results.Ok(response);

});

messagesGroup.MapPost("/conversations/", async (

HttpContext context,

[FromBody] SendMessagePayload payload,

MessageService messageService, IAntiforgery antiforgery) =>

{

await antiforgery.ValidateRequestAsync(context);

var userId = GetCurrentUserId(context);

if (userId == null) return Results.Unauthorized();

var message = await messageService.SendMessageAsync(userId.Value, payload.Id, payload.Content);

if (message == null)

{

return Results.BadRequest("Unable to send message.");

}

var response = new MessageResponse(message.Id, message.ConversationId, message.SenderId, message.Content, message.SentAt);

return Results.Ok();

});

return app;

}

private static AccountId? GetCurrentUserId(HttpContext context)

{

var userIdClaim = context.User.FindFirst(ClaimTypes.NameIdentifier);

if (userIdClaim != null && Guid.TryParse(userIdClaim.Value, out var userIdGuid))

{

return new AccountId(userIdGuid);

}

return null;

}

}

using System.Security.Claims;

using Microsoft.AspNetCore.Antiforgery;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Thrume.Database;

using Thrume.Domain.DTOs;

using Thrume.Domain.EntityIds;

using Thrume.Services;

namespace Thrume.Api.Endpoints;

public static class PostEndpoints

{

public static IEndpointRouteBuilder MapPostEndpoints(this IEndpointRouteBuilder app)

{

var postsGroup = app.MapGroup("/posts").RequireAuthorization();

postsGroup.MapPost("/", async (

HttpContext context,

AppDbContext dbContext,

[FromForm] CreatePostRequest createPostRequest, AccountService service, IAntiforgery antiforgery) =>

{

await antiforgery.ValidateRequestAsync(context);

var userIdClaim = context.User.FindFirst(ClaimTypes.NameIdentifier)!;

var guid = Guid.Parse(userIdClaim.Value);

await service.CreatePostsAsync(new AccountId(guid), createPostRequest);

return Results.Ok();

}).Accepts<IFormFile>("multipart/form-data").RequireAuthorization();

postsGroup.MapGet("/{userName}", async ([FromRoute]string userName, HttpContext context, AppDbContext dbContext) =>

{

var acc = await dbContext.PostDbSet

.AsNoTracking()

.Include(p => p.Images)

.Where(p => p.Author.UserName == userName)

.Select(p => new {p.Id, p.Content, p.Images, LikedBy = p.LikedBy.Select(acc => acc.Id), p.CreatedAt})

.ToListAsync();

return Results.Ok(acc);

}).AllowAnonymous();

postsGroup.MapPut("/like", async (HttpContext context, [FromBody] PostId id, PostService postService, IAntiforgery antiforgery) =>

{

await antiforgery.ValidateRequestAsync(context);

var accIdStr = context.User.FindFirst(ClaimTypes.NameIdentifier)!.Value;

var accId = new AccountId(Guid.Parse(accIdStr));

await postService.LikePostAsync(accId, id);

return Results.Ok();

});

return app;

}

}

using System.Text;

using Amazon.Runtime;

using Amazon.S3;

using Google.Apis.Auth.OAuth2;

using Google.Apis.Util.Store;

using Microsoft.AspNetCore.Authentication.Cookies;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Options;

using Microsoft.IdentityModel.Tokens;

using Thrume.Configuration;

using Thrume.Database;

using Thrume.Domain;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

using Thrume.Infrastructure;

using Thrume.Services;

namespace Thrume.Api.Extensions;

internal static class BuilderExtensions

{

//TODO: refactor that shit

public static WebApplicationBuilder AddEmailSender(this WebApplicationBuilder builder)

{

builder.Services.AddTransient<IEmailSender<Account>, IdentityEmailSender<Account>>();

return builder;

}

public static WebApplicationBuilder AddIdentity(this WebApplicationBuilder builder)

{

builder.Services.AddAuthorization();

builder.Services.AddOpenApi();

builder.Services.AddIdentity<Account, IdentityRole<AccountId>>()

.AddEntityFrameworkStores<AppDbContext>()

.AddDefaultTokenProviders();

builder.Services.Configure<IdentityOptions>(o =>

{

o.Password.RequireDigit = false;

o.Password.RequireLowercase = false;

o.Password.RequireUppercase = false;

o.Password.RequireNonAlphanumeric = false;

o.User.RequireUniqueEmail = true;

});

return builder;

}

public static WebApplicationBuilder AddDatabase(this WebApplicationBuilder builder)

{

builder.Services.AddDbContext<AppDbContext>(o =>

{

o.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnection"));

o.UseSnakeCaseNamingConvention();

});

return builder;

}

public static WebApplicationBuilder AddAuth(this WebApplicationBuilder builder)

{

builder.Services

.AddOptions<JwtConfiguration>()

.BindConfiguration(nameof(JwtConfiguration))

.ValidateDataAnnotations()

.ValidateOnStart();

builder.Services

.AddAuthentication(options =>

{

options.DefaultScheme = IdentityConstants.ApplicationScheme;

options.DefaultAuthenticateScheme = IdentityConstants.ApplicationScheme;

options.DefaultChallengeScheme = IdentityConstants.ApplicationScheme;

})

.AddCookie(IdentityConstants.ApplicationScheme)

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, y =>

{

y.SaveToken = false;

y.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]!)),

ValidateAudience = true,

ValidAudience = builder.Configuration["Jwt:Audience"],

ValidateIssuer = true,

ValidIssuer = builder.Configuration["Jwt:Issuer"],

ValidateLifetime = true,

};

});

return builder;

}

public static WebApplicationBuilder AddMinio(this WebApplicationBuilder builder)

{

builder.Services

.AddOptions<MinioConfiguration>()

.BindConfiguration(nameof(MinioConfiguration))

.ValidateDataAnnotations()

.ValidateOnStart();

builder.Services.AddScoped<IFileStorageRepository, ImageStorageRepository>();

builder.Services.AddSingleton<IAmazonS3>(ImplementationFactory);

return builder;

}

private static IAmazonS3 ImplementationFactory(IServiceProvider sp)

{

var minioSettings = sp.GetRequiredService<IOptions<MinioConfiguration>>().Value;

Console.WriteLine(minioSettings.SecretKey);

var config = new AmazonS3Config { ServiceURL = minioSettings.ServiceUrl, ForcePathStyle = true, UseHttp = !minioSettings.UseSsl };

var credentials = new BasicAWSCredentials(minioSettings.AccessKey, minioSettings.SecretKey);

return new AmazonS3Client(credentials, config);

}

}

{

"ConnectionStrings": {

"DefaultConnection": "Host=localhost;Port=5432;Database=thrume;Username=wildchild;Password=liberal132"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"Jwt": {

"Key": "hDzrTbuSPLNFw4CVQazt4U7s8O7svdaU",

"Issuer": "http://localhost:5132/",

"Audience": "http://localhost:5132/"

},

"MinioConfiguration": {

"ServiceURL": "http://localhost:9000",

"AccessKey": "minioadmin",

"SecretKey": "minioadmin",

"BucketName": "thrume-images",

"UseSSL": false

}

}

# Build stage: Use the .NET SDK image

FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build

WORKDIR /src

# Copy all csproj files and restore dependencies

COPY Thrume.Api/Thrume.Api.csproj Thrume.Api/

COPY Thrume.Database/Thrume.Database.csproj Thrume.Database/

COPY Thrume.Domain/Thrume.Domain.csproj Thrume.Domain/

RUN dotnet restore Thrume.Api/Thrume.Api.csproj

# Copy the entire solution and build

COPY . .

WORKDIR /src/Thrume.Api

RUN dotnet publish -c Release -o /app

# Runtime stage: Use the ASP.NET Core runtime image

FROM mcr.microsoft.com/dotnet/aspnet:9.0 AS runtime

WORKDIR /app

COPY --from=build /app ./

# Expose the port

EXPOSE 80

# Start the application

ENTRYPOINT ["dotnet", "Thrume.Api.dll"]

\*\*/.dockerignore

\*\*/.env

\*\*/.git

\*\*/.gitignore

\*\*/.project

\*\*/.settings

\*\*/.toolstarget

\*\*/.vs

\*\*/.vscode

\*\*/.idea

\*\*/\*.\*proj.user

\*\*/\*.dbmdl

\*\*/\*.jfm

\*\*/azds.yaml

\*\*/bin

\*\*/charts

\*\*/docker-compose\*

\*\*/Dockerfile\*

\*\*/node\_modules

\*\*/npm-debug.log

\*\*/obj

\*\*/secrets.dev.yaml

\*\*/values.dev.yaml

LICENSE

README.md

using Scalar.AspNetCore;

using Thrume.Api.Extensions;

using Thrume.Domain.Entity;

using Thrume.Infrastructure;

using Thrume.Services;

using Thrume.Api.Endpoints;

using Microsoft.AspNetCore.Http;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Antiforgery;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Authentication.Cookies;

var builder = WebApplication.CreateBuilder(args);

builder.Services.Configure<ServiceProviderOptions>(options =>

{

options.ValidateScopes = true;

options.ValidateOnBuild = true;

});

builder.Services.AddExceptionHandler<GlobalExceptionHandler>();

builder.Services.AddProblemDetails();

builder.AddEmailSender()

.AddIdentity()

.AddDatabase()

//.AddAuth()

.AddMinio();

builder.Services.AddScoped<IFileStorageRepository, ImageStorageRepository>();

builder.Services.AddTransient<AccountService>();

builder.Services.AddTransient<CommentService>();

builder.Services.AddTransient<PostService>();

builder.Services.AddTransient<MessageService>();

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowFrontend",

policy =>

{

policy.WithOrigins("http://localhost:5173")

.AllowAnyHeader()

.AllowAnyMethod()

.AllowCredentials();

});

});

// builder.Services.ConfigureApplicationCookie(opts =>

// {

// opts.Cookie.SameSite = SameSiteMode.None;

// opts.Cookie.SecurePolicy = CookieSecurePolicy.Always;

// opts.Events.OnRedirectToLogin = ctx =>

// {

// if (ctx.Request.Path.StartsWithSegments("/account"))

// ctx.Response.StatusCode = StatusCodes.Status401Unauthorized;

// else

// ctx.Response.Redirect(ctx.RedirectUri);

// return Task.CompletedTask;

// };

// opts.Events.OnRedirectToAccessDenied = ctx =>

// {

// ctx.Response.StatusCode = StatusCodes.Status403Forbidden;

// return Task.CompletedTask;

// };

// });

builder.Services.AddAntiforgery(options =>

{

options.HeaderName = "X-XSRF-TOKEN";

options.Cookie.Name = "MYAPP-XSRF-TOKEN";

options.Cookie.HttpOnly = false;

options.Cookie.SecurePolicy = CookieSecurePolicy.Always;

options.Cookie.SameSite = SameSiteMode.None;

});

builder.Services.ConfigureApplicationCookie(options =>

{

options.Cookie.HttpOnly = true;

options.Cookie.SecurePolicy = CookieSecurePolicy.Always;

options.Cookie.SameSite = SameSiteMode.None;

options.ExpireTimeSpan = TimeSpan.FromDays(14);

options.SlidingExpiration = true;

options.Events.OnRedirectToLogin = context =>

{

if (IsApiRequest(context.Request))

{

context.Response.StatusCode = StatusCodes.Status401Unauthorized;

}

else

{

context.Response.Redirect(context.RedirectUri);

}

return Task.CompletedTask;

};

options.Events.OnRedirectToAccessDenied = context =>

{

if (IsApiRequest(context.Request))

{

context.Response.StatusCode = StatusCodes.Status403Forbidden;

}

else

{

context.Response.Redirect(context.RedirectUri);

}

return Task.CompletedTask;

};

});

var app = builder.Build();

app.UseExceptionHandler();

app.MapScalarApiReference();

app.MapOpenApi();

app.UseCors("AllowFrontend");

app.UseAuthentication();

app.UseAuthorization();

var authGroup = app.MapGroup("/auth");

authGroup.MapPost("/logout", async (SignInManager<Account> signInManager, HttpContext httpContext) =>

{

if (signInManager.IsSignedIn(httpContext.User))

{

await signInManager.SignOutAsync();

}

return Results.Ok();

});

app.MapGroup("/auth").MapIdentityApi<Account>();

app.MapGet("/antiforgery/token", (IAntiforgery antiforgery, HttpContext context) =>

{

var tokens = antiforgery.GetAndStoreTokens(context);

return Results.Ok(new { requestToken = tokens.RequestToken });

}).AllowAnonymous();

app.MapMiscEndpoints();

app.MapPostEndpoints();

app.MapAccountEndpoints();

app.MapCommentEndpoints();

app.MapMessageEndpoints();

app.MapGet("/", () => Results.Redirect("/scalar"));

app.Run();

static bool IsApiRequest(HttpRequest request)

{

return request.Path.StartsWithSegments("/auth") ||

request.Path.StartsWithSegments("/api") ||

request.Path.StartsWithSegments("/account") ||

(request.Headers.TryGetValue("Accept", out var acceptHeader) &&

acceptHeader.Any(h => h != null && h.Contains("application/json", StringComparison.OrdinalIgnoreCase)));

}

using Microsoft.AspNetCore.Http;

using Thrume.Domain.EntityIds;

namespace Thrume.Common;

public static class DomainErrors

{

public static ProblemDetails ImageToLarge => new()

{

Detail = "File size is large than 10 MB.",

Status = StatusCodes.Status413RequestEntityTooLarge

};

public static ProblemDetails InvalidImageType => new()

{

Detail = "Image type must be .jpg/jpeg/png",

Status = StatusCodes.Status409Conflict

};

public static ProblemDetails AccountNotFound(AccountId id) => new()

{

Detail = $"Account with id {id} is not found.",

Status = StatusCodes.Status409Conflict

};

}

using System.Text.Json.Serialization;

namespace Thrume.Common;

public struct ProblemDetails()

{

/// <summary>

/// A URI reference [RFC3986] that identifies the problem type. This specification encourages that, when

/// dereferenced, it provide human-readable documentation for the problem type

/// (e.g., using HTML [W3C.REC-html5-20141028]). When this member is not present, its value is assumed to be

/// "about:blank".

/// </summary>

[JsonIgnore(Condition = JsonIgnoreCondition.WhenWritingNull)]

[JsonPropertyOrder(-5)]

[JsonPropertyName("type")]

public string? Type { get; set; } = null;

/// <summary>

/// A short, human-readable summary of the problem type. It SHOULD NOT change from occurrence to occurrence

/// of the problem, except for purposes of localization(e.g., using proactive content negotiation;

/// see[RFC7231], Section 3.4).

/// </summary>

[JsonIgnore(Condition = JsonIgnoreCondition.WhenWritingNull)]

[JsonPropertyOrder(-4)]

[JsonPropertyName("title")]

public string? Title { get; set; } = null;

/// <summary>

/// The HTTP status code([RFC7231], Section 6) generated by the origin server for this occurrence of the problem.

/// </summary>

[JsonIgnore(Condition = JsonIgnoreCondition.WhenWritingNull)]

[JsonPropertyOrder(-3)]

[JsonPropertyName("status")]

public int? Status { get; set; } = null;

/// <summary>

/// A human-readable explanation specific to this occurrence of the problem.

/// </summary>

[JsonIgnore(Condition = JsonIgnoreCondition.WhenWritingNull)]

[JsonPropertyOrder(-2)]

[JsonPropertyName("detail")]

public string? Detail { get; set; } = null;

/// <summary>

/// A URI reference that identifies the specific occurrence of the problem. It may or may not yield further information if dereferenced.

/// </summary>

[JsonIgnore(Condition = JsonIgnoreCondition.WhenWritingNull)]

[JsonPropertyOrder(-1)]

[JsonPropertyName("instance")]

public string? Instance { get; set; } = null;

/// <summary>

/// Gets the <see cref="T:System.Collections.Generic.IDictionary`2" /> for extension members.

/// <para>

/// Problem type definitions MAY extend the problem details object with additional members. Extension members appear in the same namespace as

/// other members of a problem type.

/// </para>

/// </summary>

/// <remarks>

/// The round-tripping behavior for <see cref="P:Microsoft.AspNetCore.Mvc.ProblemDetails.Extensions" /> is determined by the implementation of the Input \ Output formatters.

/// In particular, complex types or collection types may not round-trip to the original type when using the built-in JSON or XML formatters.

/// </remarks>

[JsonExtensionData]

public IDictionary<string, object> Extensions { get; set; } = new Dictionary<string, object>(StringComparer.Ordinal);

}

using System.Runtime.CompilerServices;

using System.Runtime.InteropServices;

namespace Thrume.Common;

public readonly struct Result<T>

{

public T? Value { get; }

public List<ProblemDetails> Errors { get; } = [];

public bool IsSuccess => Errors is [];

public bool IsFault => !IsSuccess;

private Result(T value) => Value = value;

private Result(params ReadOnlySpan<ProblemDetails> errors) => Errors.AddRange(errors);

public static Result<T> Success(T entity) => new(entity);

[OverloadResolutionPriority(1)]

public static Result<T> Failure(params ReadOnlySpan<ProblemDetails> errors) => new(errors);

public static Result<T> Failure(params List<ProblemDetails> errors) => new(CollectionsMarshal.AsSpan(errors));

public static Result<T> Failure(ProblemDetails error) => new(error);

public static implicit operator Result<T>(T entity) => Success(entity);

public static implicit operator Result<T>(ProblemDetails error) => Failure(error);

}

public static class ResultExtension

{

public static Result<T> ToResult<T>(this T? entity, ProblemDetails detailsOnError) =>

entity is null ? Result<T>.Failure(detailsOnError) : Result<T>.Success(entity);

public static void Match<T>(this Result<T> result, Action<T> onSuccess, Action<List<ProblemDetails>> onError)

{

if(result.IsSuccess) onSuccess(result.Value!);

else onError(result.Errors);

}

public static K Match<T, K>(this Result<T> result, Func<T, K> onSuccess, Func<List<ProblemDetails>, K> onError) =>

result.IsSuccess ? onSuccess(result.Value!) : onError(result.Errors);

}

namespace Thrume.Configuration;

public class JwtConfiguration

{

public string Key { get; init; } = null!;

public string Issuer { get; init; } = null!;

public string Audience { get; init; } = null!;

}

namespace Thrume.Configuration;

public class MinioConfiguration

{

public string ServiceUrl { get; init; } = null!;

public string AccessKey { get; init; } = null!;

public string SecretKey { get; init; } = null!;

public string BucketName { get; init; } = null!;

public bool UseSsl { get; init; }

}

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

namespace Thrume.Database.Configuration;

public sealed class AccountConfiguration : IEntityTypeConfiguration<Account>

{

public void Configure(EntityTypeBuilder<Account> builder)

{

builder.ToTable("accounts");

builder.Property(a => a.Id)

.HasConversion(

accId => accId.Value,

guid => new AccountId(guid)

)

.IsRequired();

builder.Property(a => a.UserName)

.IsRequired();

builder.Property(a => a.Email)

.IsRequired();

builder.Property(a => a.EmailConfirmed)

.HasDefaultValue(true)

.IsRequired();

builder.HasMany(a => a.Posts).WithOne(a => a.Author).HasForeignKey("author\_id");

builder.Property(a => a.PictureUrl).HasMaxLength(100);

}

}

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

namespace Thrume.Database.Configuration;

public class CommentConfiguration : IEntityTypeConfiguration<Comment>

{

public void Configure(EntityTypeBuilder<Comment> builder)

{

builder.HasKey(c => c.Id);

builder.Property(c => c.Id).ValueGeneratedNever();

builder.Property(c => c.Id)

.HasConversion(

c => c.Value,

v => new CommentId(v));

builder.Property(c => c.Content)

.IsRequired()

.HasMaxLength(1000);

builder.Property(c => c.CreatedAt)

.IsRequired();

builder.HasOne(c => c.Author)

.WithMany()

.HasForeignKey(c => c.AuthorId)

.IsRequired()

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(c => c.Post)

.WithMany(p => p.Comments)

.HasForeignKey(c => c.PostId)

.IsRequired()

.OnDelete(DeleteBehavior.Cascade);

builder.HasOne(c => c.ParentComment)

.WithMany(c => c.Replies)

.HasForeignKey(c => c.ParentCommentId)

.IsRequired(false)

.OnDelete(DeleteBehavior.ClientSetNull);

}

}

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

namespace Thrume.Database.Configuration;

public class ConversationConfiguration : IEntityTypeConfiguration<Conversation>

{

public void Configure(EntityTypeBuilder<Conversation> builder)

{

builder.HasKey(c => c.Id);

builder.Property(c => c.Id)

.HasConversion(

id => id.Value,

guid => new ConversationId(guid));

builder.Property(c => c.CreatedAt)

.IsRequired();

builder.HasMany(c => c.Participants)

.WithMany();

builder.HasMany(c => c.Messages)

.WithOne(m => m.Conversation)

.HasForeignKey(m => m.ConversationId)

.OnDelete(DeleteBehavior.Cascade);

}

}

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

namespace Thrume.Database.Configuration;

public sealed class ImageConfiguration : IEntityTypeConfiguration<Image>

{

public void Configure(EntityTypeBuilder<Image> builder)

{

builder.ToTable("images");

builder.Property(i => i.Id)

.HasConversion(

id => id.ImageUrl,

url => new ImageId(url))

.IsRequired();

builder

.HasOne(a => a.Post)

.WithMany(p => p.Images);

}

}

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

namespace Thrume.Database.Configuration;

public class MessageConfiguration : IEntityTypeConfiguration<Message>

{

public void Configure(EntityTypeBuilder<Message> builder)

{

builder.HasKey(m => m.Id);

builder.Property(m => m.Id)

.HasConversion(

id => id.Value,

guid => new MessageId(guid));

builder.Property(m => m.ConversationId).IsRequired();

builder.HasOne(m => m.Conversation)

.WithMany(c => c.Messages)

.HasForeignKey(m => m.ConversationId)

.OnDelete(DeleteBehavior.Cascade);

builder.Property(m => m.SenderId).IsRequired();

builder.HasOne(m => m.Sender)

.WithMany()

.HasForeignKey(m => m.SenderId)

.OnDelete(DeleteBehavior.Restrict);

builder.Property(m => m.Content)

.IsRequired()

.HasMaxLength(2000);

builder.Property(m => m.SentAt)

.IsRequired();

builder.HasIndex(m => m.ConversationId);

builder.HasIndex(m => m.SentAt);

}

}

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Metadata.Builders;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

namespace Thrume.Database.Configuration;

public sealed class PostConfiguration : IEntityTypeConfiguration<Post>

{

public void Configure(EntityTypeBuilder<Post> builder)

{

const int maxPostLen = 280;

builder.ToTable("posts");

builder.Property(a => a.Id)

.HasConversion(

accId => accId.Value,

guid => new PostId(guid)

).IsRequired();

builder.Property(p => p.Content).HasMaxLength(maxPostLen);

builder.HasMany(p => p.LikedBy)

.WithMany(a => a.LikedPosts)

.UsingEntity(j => j.ToTable("account\_post\_likes"));

}

}

using System;

using Microsoft.EntityFrameworkCore.Migrations;

using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

#nullable disable

namespace Thrume.Database.Migrations

{

/// <inheritdoc />

public partial class Init : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "accounts",

columns: table => new

{

id = table.Column<Guid>(type: "uuid", nullable: false),

concurrency\_stamp = table.Column<string>(type: "text", nullable: true),

user\_name = table.Column<string>(type: "character varying(256)", maxLength: 256, nullable: false),

normalized\_user\_name = table.Column<string>(type: "character varying(256)", maxLength: 256, nullable: true),

email = table.Column<string>(type: "character varying(256)", maxLength: 256, nullable: false),

normalized\_email = table.Column<string>(type: "character varying(256)", maxLength: 256, nullable: true),

email\_confirmed = table.Column<bool>(type: "boolean", nullable: false, defaultValue: true),

password\_hash = table.Column<string>(type: "text", nullable: true),

security\_stamp = table.Column<string>(type: "text", nullable: true),

phone\_number = table.Column<string>(type: "text", nullable: true),

phone\_number\_confirmed = table.Column<bool>(type: "boolean", nullable: false),

two\_factor\_enabled = table.Column<bool>(type: "boolean", nullable: false),

lockout\_end = table.Column<DateTimeOffset>(type: "timestamp with time zone", nullable: true),

lockout\_enabled = table.Column<bool>(type: "boolean", nullable: false),

access\_failed\_count = table.Column<int>(type: "integer", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("pk\_accounts", x => x.id);

});

migrationBuilder.CreateTable(

name: "AspNetRoles",

columns: table => new

{

id = table.Column<Guid>(type: "uuid", nullable: false),

name = table.Column<string>(type: "character varying(256)", maxLength: 256, nullable: true),

normalized\_name = table.Column<string>(type: "character varying(256)", maxLength: 256, nullable: true),

concurrency\_stamp = table.Column<string>(type: "text", nullable: true)

},

constraints: table =>

{

table.PrimaryKey("pk\_asp\_net\_roles", x => x.id);

});

migrationBuilder.CreateTable(

name: "AspNetUserClaims",

columns: table => new

{

id = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

user\_id = table.Column<Guid>(type: "uuid", nullable: false),

claim\_type = table.Column<string>(type: "text", nullable: true),

claim\_value = table.Column<string>(type: "text", nullable: true)

},

constraints: table =>

{

table.PrimaryKey("pk\_asp\_net\_user\_claims", x => x.id);

table.ForeignKey(

name: "fk\_asp\_net\_user\_claims\_asp\_net\_users\_user\_id",

column: x => x.user\_id,

principalTable: "accounts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "AspNetUserLogins",

columns: table => new

{

login\_provider = table.Column<string>(type: "text", nullable: false),

provider\_key = table.Column<string>(type: "text", nullable: false),

provider\_display\_name = table.Column<string>(type: "text", nullable: true),

user\_id = table.Column<Guid>(type: "uuid", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("pk\_asp\_net\_user\_logins", x => new { x.login\_provider, x.provider\_key });

table.ForeignKey(

name: "fk\_asp\_net\_user\_logins\_asp\_net\_users\_user\_id",

column: x => x.user\_id,

principalTable: "accounts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "AspNetUserTokens",

columns: table => new

{

user\_id = table.Column<Guid>(type: "uuid", nullable: false),

login\_provider = table.Column<string>(type: "text", nullable: false),

name = table.Column<string>(type: "text", nullable: false),

value = table.Column<string>(type: "text", nullable: true)

},

constraints: table =>

{

table.PrimaryKey("pk\_asp\_net\_user\_tokens", x => new { x.user\_id, x.login\_provider, x.name });

table.ForeignKey(

name: "fk\_asp\_net\_user\_tokens\_asp\_net\_users\_user\_id",

column: x => x.user\_id,

principalTable: "accounts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "posts",

columns: table => new

{

id = table.Column<Guid>(type: "uuid", nullable: false),

content = table.Column<string>(type: "text", nullable: false),

author\_id = table.Column<Guid>(type: "uuid", nullable: false),

created\_at = table.Column<DateTimeOffset>(type: "timestamp with time zone", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("pk\_posts", x => x.id);

table.ForeignKey(

name: "fk\_posts\_accounts\_author\_id",

column: x => x.author\_id,

principalTable: "accounts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "AspNetRoleClaims",

columns: table => new

{

id = table.Column<int>(type: "integer", nullable: false)

.Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),

role\_id = table.Column<Guid>(type: "uuid", nullable: false),

claim\_type = table.Column<string>(type: "text", nullable: true),

claim\_value = table.Column<string>(type: "text", nullable: true)

},

constraints: table =>

{

table.PrimaryKey("pk\_asp\_net\_role\_claims", x => x.id);

table.ForeignKey(

name: "fk\_asp\_net\_role\_claims\_asp\_net\_roles\_role\_id",

column: x => x.role\_id,

principalTable: "AspNetRoles",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "AspNetUserRoles",

columns: table => new

{

user\_id = table.Column<Guid>(type: "uuid", nullable: false),

role\_id = table.Column<Guid>(type: "uuid", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("pk\_asp\_net\_user\_roles", x => new { x.user\_id, x.role\_id });

table.ForeignKey(

name: "fk\_asp\_net\_user\_roles\_asp\_net\_roles\_role\_id",

column: x => x.role\_id,

principalTable: "AspNetRoles",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "fk\_asp\_net\_user\_roles\_asp\_net\_users\_user\_id",

column: x => x.user\_id,

principalTable: "accounts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateIndex(

name: "EmailIndex",

table: "accounts",

column: "normalized\_email");

migrationBuilder.CreateIndex(

name: "UserNameIndex",

table: "accounts",

column: "normalized\_user\_name",

unique: true);

migrationBuilder.CreateIndex(

name: "ix\_asp\_net\_role\_claims\_role\_id",

table: "AspNetRoleClaims",

column: "role\_id");

migrationBuilder.CreateIndex(

name: "RoleNameIndex",

table: "AspNetRoles",

column: "normalized\_name",

unique: true);

migrationBuilder.CreateIndex(

name: "ix\_asp\_net\_user\_claims\_user\_id",

table: "AspNetUserClaims",

column: "user\_id");

migrationBuilder.CreateIndex(

name: "ix\_asp\_net\_user\_logins\_user\_id",

table: "AspNetUserLogins",

column: "user\_id");

migrationBuilder.CreateIndex(

name: "ix\_asp\_net\_user\_roles\_role\_id",

table: "AspNetUserRoles",

column: "role\_id");

migrationBuilder.CreateIndex(

name: "ix\_posts\_author\_id",

table: "posts",

column: "author\_id");

}

/// <inheritdoc />

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "AspNetRoleClaims");

migrationBuilder.DropTable(

name: "AspNetUserClaims");

migrationBuilder.DropTable(

name: "AspNetUserLogins");

migrationBuilder.DropTable(

name: "AspNetUserRoles");

migrationBuilder.DropTable(

name: "AspNetUserTokens");

migrationBuilder.DropTable(

name: "posts");

migrationBuilder.DropTable(

name: "AspNetRoles");

migrationBuilder.DropTable(

name: "accounts");

}

}

}

// <auto-generated />

using System;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.EntityFrameworkCore.Migrations;

using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

using Thrume.Database;

#nullable disable

namespace Thrume.Database.Migrations

{

[DbContext(typeof(AppDbContext))]

[Migration("20250326154537\_Init")]

partial class Init

{

/// <inheritdoc />

protected override void BuildTargetModel(ModelBuilder modelBuilder)

{

#pragma warning disable 612, 618

modelBuilder

.HasAnnotation("ProductVersion", "9.0.2")

.HasAnnotation("Relational:MaxIdentifierLength", 63);

NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole<Thrume.Domain.AccountId>", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<string>("ConcurrencyStamp")

.IsConcurrencyToken()

.HasColumnType("text")

.HasColumnName("concurrency\_stamp");

b.Property<string>("Name")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("name");

b.Property<string>("NormalizedName")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("normalized\_name");

b.HasKey("Id")

.HasName("pk\_asp\_net\_roles");

b.HasIndex("NormalizedName")

.IsUnique()

.HasDatabaseName("RoleNameIndex");

b.ToTable("AspNetRoles", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<Thrume.Domain.AccountId>", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("integer")

.HasColumnName("id");

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

b.Property<string>("ClaimType")

.HasColumnType("text")

.HasColumnName("claim\_type");

b.Property<string>("ClaimValue")

.HasColumnType("text")

.HasColumnName("claim\_value");

b.Property<Guid>("RoleId")

.HasColumnType("uuid")

.HasColumnName("role\_id");

b.HasKey("Id")

.HasName("pk\_asp\_net\_role\_claims");

b.HasIndex("RoleId")

.HasDatabaseName("ix\_asp\_net\_role\_claims\_role\_id");

b.ToTable("AspNetRoleClaims", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<Thrume.Domain.AccountId>", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("integer")

.HasColumnName("id");

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

b.Property<string>("ClaimType")

.HasColumnType("text")

.HasColumnName("claim\_type");

b.Property<string>("ClaimValue")

.HasColumnType("text")

.HasColumnName("claim\_value");

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.HasKey("Id")

.HasName("pk\_asp\_net\_user\_claims");

b.HasIndex("UserId")

.HasDatabaseName("ix\_asp\_net\_user\_claims\_user\_id");

b.ToTable("AspNetUserClaims", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<Thrume.Domain.AccountId>", b =>

{

b.Property<string>("LoginProvider")

.HasColumnType("text")

.HasColumnName("login\_provider");

b.Property<string>("ProviderKey")

.HasColumnType("text")

.HasColumnName("provider\_key");

b.Property<string>("ProviderDisplayName")

.HasColumnType("text")

.HasColumnName("provider\_display\_name");

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.HasKey("LoginProvider", "ProviderKey")

.HasName("pk\_asp\_net\_user\_logins");

b.HasIndex("UserId")

.HasDatabaseName("ix\_asp\_net\_user\_logins\_user\_id");

b.ToTable("AspNetUserLogins", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<Thrume.Domain.AccountId>", b =>

{

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.Property<Guid>("RoleId")

.HasColumnType("uuid")

.HasColumnName("role\_id");

b.HasKey("UserId", "RoleId")

.HasName("pk\_asp\_net\_user\_roles");

b.HasIndex("RoleId")

.HasDatabaseName("ix\_asp\_net\_user\_roles\_role\_id");

b.ToTable("AspNetUserRoles", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<Thrume.Domain.AccountId>", b =>

{

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.Property<string>("LoginProvider")

.HasColumnType("text")

.HasColumnName("login\_provider");

b.Property<string>("Name")

.HasColumnType("text")

.HasColumnName("name");

b.Property<string>("Value")

.HasColumnType("text")

.HasColumnName("value");

b.HasKey("UserId", "LoginProvider", "Name")

.HasName("pk\_asp\_net\_user\_tokens");

b.ToTable("AspNetUserTokens", (string)null);

});

modelBuilder.Entity("Thrume.Domain.Account", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<int>("AccessFailedCount")

.HasColumnType("integer")

.HasColumnName("access\_failed\_count");

b.Property<string>("ConcurrencyStamp")

.IsConcurrencyToken()

.HasColumnType("text")

.HasColumnName("concurrency\_stamp");

b.Property<string>("Email")

.IsRequired()

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("email");

b.Property<bool>("EmailConfirmed")

.ValueGeneratedOnAdd()

.HasColumnType("boolean")

.HasDefaultValue(true)

.HasColumnName("email\_confirmed");

b.Property<bool>("LockoutEnabled")

.HasColumnType("boolean")

.HasColumnName("lockout\_enabled");

b.Property<DateTimeOffset?>("LockoutEnd")

.HasColumnType("timestamp with time zone")

.HasColumnName("lockout\_end");

b.Property<string>("NormalizedEmail")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("normalized\_email");

b.Property<string>("NormalizedUserName")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("normalized\_user\_name");

b.Property<string>("PasswordHash")

.HasColumnType("text")

.HasColumnName("password\_hash");

b.Property<string>("PhoneNumber")

.HasColumnType("text")

.HasColumnName("phone\_number");

b.Property<bool>("PhoneNumberConfirmed")

.HasColumnType("boolean")

.HasColumnName("phone\_number\_confirmed");

b.Property<string>("SecurityStamp")

.HasColumnType("text")

.HasColumnName("security\_stamp");

b.Property<bool>("TwoFactorEnabled")

.HasColumnType("boolean")

.HasColumnName("two\_factor\_enabled");

b.Property<string>("UserName")

.IsRequired()

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("user\_name");

b.HasKey("Id")

.HasName("pk\_accounts");

b.HasIndex("NormalizedEmail")

.HasDatabaseName("EmailIndex");

b.HasIndex("NormalizedUserName")

.IsUnique()

.HasDatabaseName("UserNameIndex");

b.ToTable("accounts", (string)null);

});

modelBuilder.Entity("Thrume.Domain.Post", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<string>("Content")

.IsRequired()

.HasColumnType("text")

.HasColumnName("content");

b.Property<DateTimeOffset>("CreatedAt")

.HasColumnType("timestamp with time zone")

.HasColumnName("created\_at");

b.Property<Guid>("author\_id")

.HasColumnType("uuid")

.HasColumnName("author\_id");

b.HasKey("Id")

.HasName("pk\_posts");

b.HasIndex("author\_id")

.HasDatabaseName("ix\_posts\_author\_id");

b.ToTable("posts", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<Thrume.Domain.AccountId>", b =>

{

b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole<Thrume.Domain.AccountId>", null)

.WithMany()

.HasForeignKey("RoleId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_role\_claims\_asp\_net\_roles\_role\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<Thrume.Domain.AccountId>", b =>

{

b.HasOne("Thrume.Domain.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_claims\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<Thrume.Domain.AccountId>", b =>

{

b.HasOne("Thrume.Domain.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_logins\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<Thrume.Domain.AccountId>", b =>

{

b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole<Thrume.Domain.AccountId>", null)

.WithMany()

.HasForeignKey("RoleId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_roles\_asp\_net\_roles\_role\_id");

b.HasOne("Thrume.Domain.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_roles\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<Thrume.Domain.AccountId>", b =>

{

b.HasOne("Thrume.Domain.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_tokens\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Thrume.Domain.Post", b =>

{

b.HasOne("Thrume.Domain.Account", "Author")

.WithMany("Posts")

.HasForeignKey("author\_id")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_posts\_accounts\_author\_id");

b.Navigation("Author");

});

modelBuilder.Entity("Thrume.Domain.Account", b =>

{

b.Navigation("Posts");

});

#pragma warning restore 612, 618

}

}

}

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace Thrume.Database.Migrations

{

/// <inheritdoc />

public partial class PfpAndMaxLen : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.AlterColumn<string>(

name: "content",

table: "posts",

type: "character varying(280)",

maxLength: 280,

nullable: false,

oldClrType: typeof(string),

oldType: "text");

migrationBuilder.AddColumn<string>(

name: "picture\_url",

table: "accounts",

type: "character varying(100)",

maxLength: 100,

nullable: true);

}

/// <inheritdoc />

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropColumn(

name: "picture\_url",

table: "accounts");

migrationBuilder.AlterColumn<string>(

name: "content",

table: "posts",

type: "text",

nullable: false,

oldClrType: typeof(string),

oldType: "character varying(280)",

oldMaxLength: 280);

}

}

}

// <auto-generated />

using System;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.EntityFrameworkCore.Migrations;

using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

using Thrume.Database;

#nullable disable

namespace Thrume.Database.Migrations

{

[DbContext(typeof(AppDbContext))]

[Migration("20250327141934\_PfpAndMaxLen")]

partial class PfpAndMaxLen

{

/// <inheritdoc />

protected override void BuildTargetModel(ModelBuilder modelBuilder)

{

#pragma warning disable 612, 618

modelBuilder

.HasAnnotation("ProductVersion", "9.0.2")

.HasAnnotation("Relational:MaxIdentifierLength", 63);

NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole<Thrume.Domain.Account.AccountId>", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<string>("ConcurrencyStamp")

.IsConcurrencyToken()

.HasColumnType("text")

.HasColumnName("concurrency\_stamp");

b.Property<string>("Name")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("name");

b.Property<string>("NormalizedName")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("normalized\_name");

b.HasKey("Id")

.HasName("pk\_asp\_net\_roles");

b.HasIndex("NormalizedName")

.IsUnique()

.HasDatabaseName("RoleNameIndex");

b.ToTable("AspNetRoles", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<Thrume.Domain.Account.AccountId>", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("integer")

.HasColumnName("id");

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

b.Property<string>("ClaimType")

.HasColumnType("text")

.HasColumnName("claim\_type");

b.Property<string>("ClaimValue")

.HasColumnType("text")

.HasColumnName("claim\_value");

b.Property<Guid>("RoleId")

.HasColumnType("uuid")

.HasColumnName("role\_id");

b.HasKey("Id")

.HasName("pk\_asp\_net\_role\_claims");

b.HasIndex("RoleId")

.HasDatabaseName("ix\_asp\_net\_role\_claims\_role\_id");

b.ToTable("AspNetRoleClaims", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<Thrume.Domain.Account.AccountId>", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("integer")

.HasColumnName("id");

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

b.Property<string>("ClaimType")

.HasColumnType("text")

.HasColumnName("claim\_type");

b.Property<string>("ClaimValue")

.HasColumnType("text")

.HasColumnName("claim\_value");

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.HasKey("Id")

.HasName("pk\_asp\_net\_user\_claims");

b.HasIndex("UserId")

.HasDatabaseName("ix\_asp\_net\_user\_claims\_user\_id");

b.ToTable("AspNetUserClaims", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<Thrume.Domain.Account.AccountId>", b =>

{

b.Property<string>("LoginProvider")

.HasColumnType("text")

.HasColumnName("login\_provider");

b.Property<string>("ProviderKey")

.HasColumnType("text")

.HasColumnName("provider\_key");

b.Property<string>("ProviderDisplayName")

.HasColumnType("text")

.HasColumnName("provider\_display\_name");

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.HasKey("LoginProvider", "ProviderKey")

.HasName("pk\_asp\_net\_user\_logins");

b.HasIndex("UserId")

.HasDatabaseName("ix\_asp\_net\_user\_logins\_user\_id");

b.ToTable("AspNetUserLogins", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<Thrume.Domain.Account.AccountId>", b =>

{

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.Property<Guid>("RoleId")

.HasColumnType("uuid")

.HasColumnName("role\_id");

b.HasKey("UserId", "RoleId")

.HasName("pk\_asp\_net\_user\_roles");

b.HasIndex("RoleId")

.HasDatabaseName("ix\_asp\_net\_user\_roles\_role\_id");

b.ToTable("AspNetUserRoles", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<Thrume.Domain.Account.AccountId>", b =>

{

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.Property<string>("LoginProvider")

.HasColumnType("text")

.HasColumnName("login\_provider");

b.Property<string>("Name")

.HasColumnType("text")

.HasColumnName("name");

b.Property<string>("Value")

.HasColumnType("text")

.HasColumnName("value");

b.HasKey("UserId", "LoginProvider", "Name")

.HasName("pk\_asp\_net\_user\_tokens");

b.ToTable("AspNetUserTokens", (string)null);

});

modelBuilder.Entity("Thrume.Domain.Entity.Account", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<int>("AccessFailedCount")

.HasColumnType("integer")

.HasColumnName("access\_failed\_count");

b.Property<string>("ConcurrencyStamp")

.IsConcurrencyToken()

.HasColumnType("text")

.HasColumnName("concurrency\_stamp");

b.Property<string>("Email")

.IsRequired()

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("email");

b.Property<bool>("EmailConfirmed")

.ValueGeneratedOnAdd()

.HasColumnType("boolean")

.HasDefaultValue(true)

.HasColumnName("email\_confirmed");

b.Property<bool>("LockoutEnabled")

.HasColumnType("boolean")

.HasColumnName("lockout\_enabled");

b.Property<DateTimeOffset?>("LockoutEnd")

.HasColumnType("timestamp with time zone")

.HasColumnName("lockout\_end");

b.Property<string>("NormalizedEmail")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("normalized\_email");

b.Property<string>("NormalizedUserName")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("normalized\_user\_name");

b.Property<string>("PasswordHash")

.HasColumnType("text")

.HasColumnName("password\_hash");

b.Property<string>("PhoneNumber")

.HasColumnType("text")

.HasColumnName("phone\_number");

b.Property<bool>("PhoneNumberConfirmed")

.HasColumnType("boolean")

.HasColumnName("phone\_number\_confirmed");

b.Property<string>("PictureUrl")

.HasMaxLength(100)

.HasColumnType("character varying(100)")

.HasColumnName("picture\_url");

b.Property<string>("SecurityStamp")

.HasColumnType("text")

.HasColumnName("security\_stamp");

b.Property<bool>("TwoFactorEnabled")

.HasColumnType("boolean")

.HasColumnName("two\_factor\_enabled");

b.Property<string>("UserName")

.IsRequired()

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("user\_name");

b.HasKey("Id")

.HasName("pk\_accounts");

b.HasIndex("NormalizedEmail")

.HasDatabaseName("EmailIndex");

b.HasIndex("NormalizedUserName")

.IsUnique()

.HasDatabaseName("UserNameIndex");

b.ToTable("accounts", (string)null);

});

modelBuilder.Entity("Thrume.Domain.Entity.Post", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<string>("Content")

.IsRequired()

.HasMaxLength(280)

.HasColumnType("character varying(280)")

.HasColumnName("content");

b.Property<DateTimeOffset>("CreatedAt")

.HasColumnType("timestamp with time zone")

.HasColumnName("created\_at");

b.Property<Guid>("author\_id")

.HasColumnType("uuid")

.HasColumnName("author\_id");

b.HasKey("Id")

.HasName("pk\_posts");

b.HasIndex("author\_id")

.HasDatabaseName("ix\_posts\_author\_id");

b.ToTable("posts", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<Thrume.Domain.Account.AccountId>", b =>

{

b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole<Thrume.Domain.Account.AccountId>", null)

.WithMany()

.HasForeignKey("RoleId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_role\_claims\_asp\_net\_roles\_role\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<Thrume.Domain.Account.AccountId>", b =>

{

b.HasOne("Thrume.Domain.Entity.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_claims\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<Thrume.Domain.Account.AccountId>", b =>

{

b.HasOne("Thrume.Domain.Entity.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_logins\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<Thrume.Domain.Account.AccountId>", b =>

{

b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole<Thrume.Domain.Account.AccountId>", null)

.WithMany()

.HasForeignKey("RoleId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_roles\_asp\_net\_roles\_role\_id");

b.HasOne("Thrume.Domain.Entity.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_roles\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<Thrume.Domain.Account.AccountId>", b =>

{

b.HasOne("Thrume.Domain.Entity.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_tokens\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Thrume.Domain.Entity.Post", b =>

{

b.HasOne("Thrume.Domain.Entity.Account", "Author")

.WithMany("Posts")

.HasForeignKey("author\_id")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_posts\_accounts\_author\_id");

b.Navigation("Author");

});

modelBuilder.Entity("Thrume.Domain.Entity.Account", b =>

{

b.Navigation("Posts");

});

#pragma warning restore 612, 618

}

}

}

using System;

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace Thrume.Database.Migrations

{

/// <inheritdoc />

public partial class AddImageSupportToPosts2 : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "images",

columns: table => new

{

id = table.Column<string>(type: "text", nullable: false),

post\_id = table.Column<Guid>(type: "uuid", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("pk\_images", x => x.id);

table.ForeignKey(

name: "fk\_images\_posts\_post\_id",

column: x => x.post\_id,

principalTable: "posts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateIndex(

name: "ix\_images\_post\_id",

table: "images",

column: "post\_id");

}

/// <inheritdoc />

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "images");

}

}

}

using System;

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace Thrume.Database.Migrations

{

/// <inheritdoc />

public partial class AddPostLikesFeature : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropForeignKey(

name: "fk\_posts\_accounts\_author\_id",

table: "posts");

migrationBuilder.AlterColumn<Guid>(

name: "author\_id",

table: "posts",

type: "uuid",

nullable: true,

oldClrType: typeof(Guid),

oldType: "uuid");

migrationBuilder.CreateTable(

name: "account\_post\_likes",

columns: table => new

{

liked\_by\_id = table.Column<Guid>(type: "uuid", nullable: false),

liked\_posts\_id = table.Column<Guid>(type: "uuid", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("pk\_account\_post\_likes", x => new { x.liked\_by\_id, x.liked\_posts\_id });

table.ForeignKey(

name: "fk\_account\_post\_likes\_accounts\_liked\_by\_id",

column: x => x.liked\_by\_id,

principalTable: "accounts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "fk\_account\_post\_likes\_posts\_liked\_posts\_id",

column: x => x.liked\_posts\_id,

principalTable: "posts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateIndex(

name: "ix\_account\_post\_likes\_liked\_posts\_id",

table: "account\_post\_likes",

column: "liked\_posts\_id");

migrationBuilder.AddForeignKey(

name: "fk\_posts\_accounts\_author\_id",

table: "posts",

column: "author\_id",

principalTable: "accounts",

principalColumn: "id");

}

/// <inheritdoc />

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropForeignKey(

name: "fk\_posts\_accounts\_author\_id",

table: "posts");

migrationBuilder.DropTable(

name: "account\_post\_likes");

migrationBuilder.AlterColumn<Guid>(

name: "author\_id",

table: "posts",

type: "uuid",

nullable: false,

defaultValue: new Guid("00000000-0000-0000-0000-000000000000"),

oldClrType: typeof(Guid),

oldType: "uuid",

oldNullable: true);

migrationBuilder.AddForeignKey(

name: "fk\_posts\_accounts\_author\_id",

table: "posts",

column: "author\_id",

principalTable: "accounts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

}

}

}

using System;

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace Thrume.Database.Migrations

{

/// <inheritdoc />

public partial class AddNestedCommentFeature : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "comment\_db\_set",

columns: table => new

{

id = table.Column<Guid>(type: "uuid", nullable: false),

content = table.Column<string>(type: "character varying(1000)", maxLength: 1000, nullable: false),

author\_id = table.Column<Guid>(type: "uuid", nullable: false),

post\_id = table.Column<Guid>(type: "uuid", nullable: false),

parent\_comment\_id = table.Column<Guid>(type: "uuid", nullable: true),

created\_at = table.Column<DateTimeOffset>(type: "timestamp with time zone", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("pk\_comment\_db\_set", x => x.id);

table.ForeignKey(

name: "fk\_comment\_db\_set\_accounts\_author\_id",

column: x => x.author\_id,

principalTable: "accounts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "fk\_comment\_db\_set\_comment\_db\_set\_parent\_comment\_id",

column: x => x.parent\_comment\_id,

principalTable: "comment\_db\_set",

principalColumn: "id");

table.ForeignKey(

name: "fk\_comment\_db\_set\_posts\_post\_id",

column: x => x.post\_id,

principalTable: "posts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateIndex(

name: "ix\_comment\_db\_set\_author\_id",

table: "comment\_db\_set",

column: "author\_id");

migrationBuilder.CreateIndex(

name: "ix\_comment\_db\_set\_parent\_comment\_id",

table: "comment\_db\_set",

column: "parent\_comment\_id");

migrationBuilder.CreateIndex(

name: "ix\_comment\_db\_set\_post\_id",

table: "comment\_db\_set",

column: "post\_id");

}

/// <inheritdoc />

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "comment\_db\_set");

}

}

}

using System;

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace Thrume.Database.Migrations

{

/// <inheritdoc />

public partial class MessagesFeature : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "conversation\_db\_set",

columns: table => new

{

id = table.Column<Guid>(type: "uuid", nullable: false),

created\_at = table.Column<DateTimeOffset>(type: "timestamp with time zone", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("pk\_conversation\_db\_set", x => x.id);

});

migrationBuilder.CreateTable(

name: "account\_conversation",

columns: table => new

{

conversation\_id = table.Column<Guid>(type: "uuid", nullable: false),

participants\_id = table.Column<Guid>(type: "uuid", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("pk\_account\_conversation", x => new { x.conversation\_id, x.participants\_id });

table.ForeignKey(

name: "fk\_account\_conversation\_accounts\_participants\_id",

column: x => x.participants\_id,

principalTable: "accounts",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

table.ForeignKey(

name: "fk\_account\_conversation\_conversation\_db\_set\_conversation\_id",

column: x => x.conversation\_id,

principalTable: "conversation\_db\_set",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateTable(

name: "message\_db\_set",

columns: table => new

{

id = table.Column<Guid>(type: "uuid", nullable: false),

conversation\_id = table.Column<Guid>(type: "uuid", nullable: false),

sender\_id = table.Column<Guid>(type: "uuid", nullable: false),

content = table.Column<string>(type: "character varying(2000)", maxLength: 2000, nullable: false),

sent\_at = table.Column<DateTimeOffset>(type: "timestamp with time zone", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("pk\_message\_db\_set", x => x.id);

table.ForeignKey(

name: "fk\_message\_db\_set\_accounts\_sender\_id",

column: x => x.sender\_id,

principalTable: "accounts",

principalColumn: "id",

onDelete: ReferentialAction.Restrict);

table.ForeignKey(

name: "fk\_message\_db\_set\_conversation\_db\_set\_conversation\_id",

column: x => x.conversation\_id,

principalTable: "conversation\_db\_set",

principalColumn: "id",

onDelete: ReferentialAction.Cascade);

});

migrationBuilder.CreateIndex(

name: "ix\_account\_conversation\_participants\_id",

table: "account\_conversation",

column: "participants\_id");

migrationBuilder.CreateIndex(

name: "ix\_message\_db\_set\_conversation\_id",

table: "message\_db\_set",

column: "conversation\_id");

migrationBuilder.CreateIndex(

name: "ix\_message\_db\_set\_sender\_id",

table: "message\_db\_set",

column: "sender\_id");

migrationBuilder.CreateIndex(

name: "ix\_message\_db\_set\_sent\_at",

table: "message\_db\_set",

column: "sent\_at");

}

/// <inheritdoc />

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "account\_conversation");

migrationBuilder.DropTable(

name: "message\_db\_set");

migrationBuilder.DropTable(

name: "conversation\_db\_set");

}

}

}

// <auto-generated />

using System;

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.EntityFrameworkCore.Storage.ValueConversion;

using Npgsql.EntityFrameworkCore.PostgreSQL.Metadata;

using Thrume.Database;

#nullable disable

namespace Thrume.Database.Migrations

{

[DbContext(typeof(AppDbContext))]

partial class AppDbContextModelSnapshot : ModelSnapshot

{

protected override void BuildModel(ModelBuilder modelBuilder)

{

#pragma warning disable 612, 618

modelBuilder

.HasAnnotation("ProductVersion", "9.0.2")

.HasAnnotation("Relational:MaxIdentifierLength", 63);

NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);

modelBuilder.Entity("AccountConversation", b =>

{

b.Property<Guid>("ConversationId")

.HasColumnType("uuid")

.HasColumnName("conversation\_id");

b.Property<Guid>("ParticipantsId")

.HasColumnType("uuid")

.HasColumnName("participants\_id");

b.HasKey("ConversationId", "ParticipantsId")

.HasName("pk\_account\_conversation");

b.HasIndex("ParticipantsId")

.HasDatabaseName("ix\_account\_conversation\_participants\_id");

b.ToTable("account\_conversation", (string)null);

});

modelBuilder.Entity("AccountPost", b =>

{

b.Property<Guid>("LikedById")

.HasColumnType("uuid")

.HasColumnName("liked\_by\_id");

b.Property<Guid>("LikedPostsId")

.HasColumnType("uuid")

.HasColumnName("liked\_posts\_id");

b.HasKey("LikedById", "LikedPostsId")

.HasName("pk\_account\_post\_likes");

b.HasIndex("LikedPostsId")

.HasDatabaseName("ix\_account\_post\_likes\_liked\_posts\_id");

b.ToTable("account\_post\_likes", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole<Thrume.Domain.EntityIds.AccountId>", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<string>("ConcurrencyStamp")

.IsConcurrencyToken()

.HasColumnType("text")

.HasColumnName("concurrency\_stamp");

b.Property<string>("Name")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("name");

b.Property<string>("NormalizedName")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("normalized\_name");

b.HasKey("Id")

.HasName("pk\_asp\_net\_roles");

b.HasIndex("NormalizedName")

.IsUnique()

.HasDatabaseName("RoleNameIndex");

b.ToTable("AspNetRoles", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<Thrume.Domain.EntityIds.AccountId>", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("integer")

.HasColumnName("id");

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

b.Property<string>("ClaimType")

.HasColumnType("text")

.HasColumnName("claim\_type");

b.Property<string>("ClaimValue")

.HasColumnType("text")

.HasColumnName("claim\_value");

b.Property<Guid>("RoleId")

.HasColumnType("uuid")

.HasColumnName("role\_id");

b.HasKey("Id")

.HasName("pk\_asp\_net\_role\_claims");

b.HasIndex("RoleId")

.HasDatabaseName("ix\_asp\_net\_role\_claims\_role\_id");

b.ToTable("AspNetRoleClaims", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<Thrume.Domain.EntityIds.AccountId>", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("integer")

.HasColumnName("id");

NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

b.Property<string>("ClaimType")

.HasColumnType("text")

.HasColumnName("claim\_type");

b.Property<string>("ClaimValue")

.HasColumnType("text")

.HasColumnName("claim\_value");

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.HasKey("Id")

.HasName("pk\_asp\_net\_user\_claims");

b.HasIndex("UserId")

.HasDatabaseName("ix\_asp\_net\_user\_claims\_user\_id");

b.ToTable("AspNetUserClaims", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<Thrume.Domain.EntityIds.AccountId>", b =>

{

b.Property<string>("LoginProvider")

.HasColumnType("text")

.HasColumnName("login\_provider");

b.Property<string>("ProviderKey")

.HasColumnType("text")

.HasColumnName("provider\_key");

b.Property<string>("ProviderDisplayName")

.HasColumnType("text")

.HasColumnName("provider\_display\_name");

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.HasKey("LoginProvider", "ProviderKey")

.HasName("pk\_asp\_net\_user\_logins");

b.HasIndex("UserId")

.HasDatabaseName("ix\_asp\_net\_user\_logins\_user\_id");

b.ToTable("AspNetUserLogins", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<Thrume.Domain.EntityIds.AccountId>", b =>

{

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.Property<Guid>("RoleId")

.HasColumnType("uuid")

.HasColumnName("role\_id");

b.HasKey("UserId", "RoleId")

.HasName("pk\_asp\_net\_user\_roles");

b.HasIndex("RoleId")

.HasDatabaseName("ix\_asp\_net\_user\_roles\_role\_id");

b.ToTable("AspNetUserRoles", (string)null);

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<Thrume.Domain.EntityIds.AccountId>", b =>

{

b.Property<Guid>("UserId")

.HasColumnType("uuid")

.HasColumnName("user\_id");

b.Property<string>("LoginProvider")

.HasColumnType("text")

.HasColumnName("login\_provider");

b.Property<string>("Name")

.HasColumnType("text")

.HasColumnName("name");

b.Property<string>("Value")

.HasColumnType("text")

.HasColumnName("value");

b.HasKey("UserId", "LoginProvider", "Name")

.HasName("pk\_asp\_net\_user\_tokens");

b.ToTable("AspNetUserTokens", (string)null);

});

modelBuilder.Entity("Thrume.Domain.Entity.Account", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<int>("AccessFailedCount")

.HasColumnType("integer")

.HasColumnName("access\_failed\_count");

b.Property<string>("ConcurrencyStamp")

.IsConcurrencyToken()

.HasColumnType("text")

.HasColumnName("concurrency\_stamp");

b.Property<string>("Email")

.IsRequired()

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("email");

b.Property<bool>("EmailConfirmed")

.ValueGeneratedOnAdd()

.HasColumnType("boolean")

.HasDefaultValue(true)

.HasColumnName("email\_confirmed");

b.Property<bool>("LockoutEnabled")

.HasColumnType("boolean")

.HasColumnName("lockout\_enabled");

b.Property<DateTimeOffset?>("LockoutEnd")

.HasColumnType("timestamp with time zone")

.HasColumnName("lockout\_end");

b.Property<string>("NormalizedEmail")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("normalized\_email");

b.Property<string>("NormalizedUserName")

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("normalized\_user\_name");

b.Property<string>("PasswordHash")

.HasColumnType("text")

.HasColumnName("password\_hash");

b.Property<string>("PhoneNumber")

.HasColumnType("text")

.HasColumnName("phone\_number");

b.Property<bool>("PhoneNumberConfirmed")

.HasColumnType("boolean")

.HasColumnName("phone\_number\_confirmed");

b.Property<string>("PictureUrl")

.HasMaxLength(100)

.HasColumnType("character varying(100)")

.HasColumnName("picture\_url");

b.Property<string>("SecurityStamp")

.HasColumnType("text")

.HasColumnName("security\_stamp");

b.Property<bool>("TwoFactorEnabled")

.HasColumnType("boolean")

.HasColumnName("two\_factor\_enabled");

b.Property<string>("UserName")

.IsRequired()

.HasMaxLength(256)

.HasColumnType("character varying(256)")

.HasColumnName("user\_name");

b.HasKey("Id")

.HasName("pk\_accounts");

b.HasIndex("NormalizedEmail")

.HasDatabaseName("EmailIndex");

b.HasIndex("NormalizedUserName")

.IsUnique()

.HasDatabaseName("UserNameIndex");

b.ToTable("accounts", (string)null);

});

modelBuilder.Entity("Thrume.Domain.Entity.Comment", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<Guid>("AuthorId")

.HasColumnType("uuid")

.HasColumnName("author\_id");

b.Property<string>("Content")

.IsRequired()

.HasMaxLength(1000)

.HasColumnType("character varying(1000)")

.HasColumnName("content");

b.Property<DateTimeOffset>("CreatedAt")

.HasColumnType("timestamp with time zone")

.HasColumnName("created\_at");

b.Property<Guid?>("ParentCommentId")

.HasColumnType("uuid")

.HasColumnName("parent\_comment\_id");

b.Property<Guid>("PostId")

.HasColumnType("uuid")

.HasColumnName("post\_id");

b.HasKey("Id")

.HasName("pk\_comment\_db\_set");

b.HasIndex("AuthorId")

.HasDatabaseName("ix\_comment\_db\_set\_author\_id");

b.HasIndex("ParentCommentId")

.HasDatabaseName("ix\_comment\_db\_set\_parent\_comment\_id");

b.HasIndex("PostId")

.HasDatabaseName("ix\_comment\_db\_set\_post\_id");

b.ToTable("comment\_db\_set", (string)null);

});

modelBuilder.Entity("Thrume.Domain.Entity.Conversation", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<DateTimeOffset>("CreatedAt")

.HasColumnType("timestamp with time zone")

.HasColumnName("created\_at");

b.HasKey("Id")

.HasName("pk\_conversation\_db\_set");

b.ToTable("conversation\_db\_set", (string)null);

});

modelBuilder.Entity("Thrume.Domain.Entity.Image", b =>

{

b.Property<string>("Id")

.HasColumnType("text")

.HasColumnName("id");

b.Property<Guid>("PostId")

.HasColumnType("uuid")

.HasColumnName("post\_id");

b.HasKey("Id")

.HasName("pk\_images");

b.HasIndex("PostId")

.HasDatabaseName("ix\_images\_post\_id");

b.ToTable("images", (string)null);

});

modelBuilder.Entity("Thrume.Domain.Entity.Message", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<string>("Content")

.IsRequired()

.HasMaxLength(2000)

.HasColumnType("character varying(2000)")

.HasColumnName("content");

b.Property<Guid>("ConversationId")

.HasColumnType("uuid")

.HasColumnName("conversation\_id");

b.Property<Guid>("SenderId")

.HasColumnType("uuid")

.HasColumnName("sender\_id");

b.Property<DateTimeOffset>("SentAt")

.HasColumnType("timestamp with time zone")

.HasColumnName("sent\_at");

b.HasKey("Id")

.HasName("pk\_message\_db\_set");

b.HasIndex("ConversationId")

.HasDatabaseName("ix\_message\_db\_set\_conversation\_id");

b.HasIndex("SenderId")

.HasDatabaseName("ix\_message\_db\_set\_sender\_id");

b.HasIndex("SentAt")

.HasDatabaseName("ix\_message\_db\_set\_sent\_at");

b.ToTable("message\_db\_set", (string)null);

});

modelBuilder.Entity("Thrume.Domain.Entity.Post", b =>

{

b.Property<Guid>("Id")

.HasColumnType("uuid")

.HasColumnName("id");

b.Property<string>("Content")

.IsRequired()

.HasMaxLength(280)

.HasColumnType("character varying(280)")

.HasColumnName("content");

b.Property<DateTimeOffset>("CreatedAt")

.HasColumnType("timestamp with time zone")

.HasColumnName("created\_at");

b.Property<Guid?>("author\_id")

.HasColumnType("uuid")

.HasColumnName("author\_id");

b.HasKey("Id")

.HasName("pk\_posts");

b.HasIndex("author\_id")

.HasDatabaseName("ix\_posts\_author\_id");

b.ToTable("posts", (string)null);

});

modelBuilder.Entity("AccountConversation", b =>

{

b.HasOne("Thrume.Domain.Entity.Conversation", null)

.WithMany()

.HasForeignKey("ConversationId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_account\_conversation\_conversation\_db\_set\_conversation\_id");

b.HasOne("Thrume.Domain.Entity.Account", null)

.WithMany()

.HasForeignKey("ParticipantsId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_account\_conversation\_accounts\_participants\_id");

});

modelBuilder.Entity("AccountPost", b =>

{

b.HasOne("Thrume.Domain.Entity.Account", null)

.WithMany()

.HasForeignKey("LikedById")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_account\_post\_likes\_accounts\_liked\_by\_id");

b.HasOne("Thrume.Domain.Entity.Post", null)

.WithMany()

.HasForeignKey("LikedPostsId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_account\_post\_likes\_posts\_liked\_posts\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<Thrume.Domain.EntityIds.AccountId>", b =>

{

b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole<Thrume.Domain.EntityIds.AccountId>", null)

.WithMany()

.HasForeignKey("RoleId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_role\_claims\_asp\_net\_roles\_role\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<Thrume.Domain.EntityIds.AccountId>", b =>

{

b.HasOne("Thrume.Domain.Entity.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_claims\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<Thrume.Domain.EntityIds.AccountId>", b =>

{

b.HasOne("Thrume.Domain.Entity.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_logins\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<Thrume.Domain.EntityIds.AccountId>", b =>

{

b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole<Thrume.Domain.EntityIds.AccountId>", null)

.WithMany()

.HasForeignKey("RoleId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_roles\_asp\_net\_roles\_role\_id");

b.HasOne("Thrume.Domain.Entity.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_roles\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<Thrume.Domain.EntityIds.AccountId>", b =>

{

b.HasOne("Thrume.Domain.Entity.Account", null)

.WithMany()

.HasForeignKey("UserId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_asp\_net\_user\_tokens\_asp\_net\_users\_user\_id");

});

modelBuilder.Entity("Thrume.Domain.Entity.Comment", b =>

{

b.HasOne("Thrume.Domain.Entity.Account", "Author")

.WithMany()

.HasForeignKey("AuthorId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_comment\_db\_set\_accounts\_author\_id");

b.HasOne("Thrume.Domain.Entity.Comment", "ParentComment")

.WithMany("Replies")

.HasForeignKey("ParentCommentId")

.HasConstraintName("fk\_comment\_db\_set\_comment\_db\_set\_parent\_comment\_id");

b.HasOne("Thrume.Domain.Entity.Post", "Post")

.WithMany("Comments")

.HasForeignKey("PostId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_comment\_db\_set\_posts\_post\_id");

b.Navigation("Author");

b.Navigation("ParentComment");

b.Navigation("Post");

});

modelBuilder.Entity("Thrume.Domain.Entity.Image", b =>

{

b.HasOne("Thrume.Domain.Entity.Post", "Post")

.WithMany("Images")

.HasForeignKey("PostId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_images\_posts\_post\_id");

b.Navigation("Post");

});

modelBuilder.Entity("Thrume.Domain.Entity.Message", b =>

{

b.HasOne("Thrume.Domain.Entity.Conversation", "Conversation")

.WithMany("Messages")

.HasForeignKey("ConversationId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired()

.HasConstraintName("fk\_message\_db\_set\_conversation\_db\_set\_conversation\_id");

b.HasOne("Thrume.Domain.Entity.Account", "Sender")

.WithMany()

.HasForeignKey("SenderId")

.OnDelete(DeleteBehavior.Restrict)

.IsRequired()

.HasConstraintName("fk\_message\_db\_set\_accounts\_sender\_id");

b.Navigation("Conversation");

b.Navigation("Sender");

});

modelBuilder.Entity("Thrume.Domain.Entity.Post", b =>

{

b.HasOne("Thrume.Domain.Entity.Account", "Author")

.WithMany("Posts")

.HasForeignKey("author\_id")

.HasConstraintName("fk\_posts\_accounts\_author\_id");

b.Navigation("Author");

});

modelBuilder.Entity("Thrume.Domain.Entity.Account", b =>

{

b.Navigation("Posts");

});

modelBuilder.Entity("Thrume.Domain.Entity.Comment", b =>

{

b.Navigation("Replies");

});

modelBuilder.Entity("Thrume.Domain.Entity.Conversation", b =>

{

b.Navigation("Messages");

});

modelBuilder.Entity("Thrume.Domain.Entity.Post", b =>

{

b.Navigation("Comments");

b.Navigation("Images");

});

#pragma warning restore 612, 618

}

}

}

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using Thrume.Database.Configuration;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

namespace Thrume.Database;

public sealed class AppDbContext(DbContextOptions<AppDbContext> dbOptions) :

IdentityDbContext<Account, IdentityRole<AccountId>, AccountId>(dbOptions)

{

public DbSet<Account> AccountDbSet => Set<Account>();

public DbSet<Post> PostDbSet => Set<Post>();

public DbSet<Image> ImageDbSet => Set<Image>();

public DbSet<Comment> CommentDbSet => Set<Comment>(); // Added for comments

public DbSet<Conversation> ConversationDbSet => Set<Conversation>();

public DbSet<Message> MessageDbSet => Set<Message>();

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

builder.Entity<IdentityRole<AccountId>>()

.Property(a => a.Id)

.HasConversion(

id => id.Value,

guid => new AccountId(guid)

);

new AccountConfiguration().Configure(builder.Entity<Account>());

new PostConfiguration().Configure(builder.Entity<Post>());

new ImageConfiguration().Configure(builder.Entity<Image>());

new CommentConfiguration().Configure(builder.Entity<Comment>());

builder.ApplyConfiguration(new ConversationConfiguration());

builder.ApplyConfiguration(new MessageConfiguration());

}

}

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Design;

using Microsoft.Extensions.Configuration;

namespace Thrume.Database;

public sealed class AppDesignTimeDbContextFactory : IDesignTimeDbContextFactory<AppDbContext>

{

public AppDbContext CreateDbContext(string[] args)

{

var configuration = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory() + @"\..\Thrume.Api")

.AddJsonFile("appsettings.json")

.Build();

var connectionString = configuration.GetConnectionString("DefaultConnection");

var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();

optionsBuilder.UseNpgsql(connectionString);

optionsBuilder.UseSnakeCaseNamingConvention();

return new AppDbContext(optionsBuilder.Options);

}

}

using Microsoft.AspNetCore.Http;

using Thrume.Domain.EntityIds;

namespace Thrume.Domain.DTOs;

public sealed record CreatePostRequest(string Content, IFormFileCollection Images);

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Http;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

namespace Thrume.Domain.DTOs;

public record struct UpdateAccountRequest(

IFormFile? Picture = null,

CreatePostRequest? PostReq = null);

using System.Collections.Immutable;

using Microsoft.AspNetCore.Identity;

using Thrume.Domain.EntityIds;

namespace Thrume.Domain.Entity;

public sealed class Account : IdentityUser<AccountId>

{

public override AccountId Id { get; set; } = Guid.CreateVersion7();

public override string? ConcurrencyStamp { get; set; } = Guid.CreateVersion7().ToString();

public List<Post> Posts { get; init; } = [];

public List<Post> LikedPosts { get; init; } = [];

public string? PictureUrl { get; init; }

}

using Thrume.Domain.EntityIds;

namespace Thrume.Domain.Entity;

public sealed class Comment

{

public CommentId Id { get; init; } = Guid.CreateVersion7();

public string Content { get; set; } = string.Empty;

public AccountId AuthorId { get; init; }

public Account Author { get; init; } = null!;

public PostId PostId { get; init; }

public Post Post { get; init; } = null!;

public CommentId? ParentCommentId { get; init; }

public Comment? ParentComment { get; init; }

public List<Comment> Replies { get; init; } = [];

public DateTimeOffset CreatedAt { get; init; } = DateTimeOffset.UtcNow;

}

using System;

using System.Collections.Generic;

using Thrume.Domain.EntityIds;

namespace Thrume.Domain.Entity;

public class Conversation

{

public ConversationId Id { get; set; }

public DateTimeOffset CreatedAt { get; set; }

public virtual ICollection<Account> Participants { get; set; } = new List<Account>();

public virtual ICollection<Message> Messages { get; set; } = new List<Message>();

}

using Thrume.Domain.EntityIds;

namespace Thrume.Domain.Entity;

public sealed class Image

{

public ImageId Id { get; init; }

public Post Post { get; init; }

}

using System;

using Thrume.Domain.EntityIds;

namespace Thrume.Domain.Entity;

public class Message

{

public MessageId Id { get; set; }

public ConversationId ConversationId { get; set; }

public virtual Conversation Conversation { get; set; } = null!;

public AccountId SenderId { get; set; }

public virtual Account Sender { get; set; } = null!;

public string Content { get; set; } = string.Empty;

public DateTimeOffset SentAt { get; set; }

}

using Thrume.Domain.EntityIds;

namespace Thrume.Domain.Entity;

public sealed class Post

{

public PostId Id { get; init; } = Guid.CreateVersion7();

public string Content { get; init; }

public List<Image> Images = [];

public Account Author { get; init; }

public List<Account> LikedBy { get; init; } = [];

public List<Comment> Comments { get; init; } = [];

public DateTimeOffset CreatedAt { get; init; } = DateTime.UtcNow;

}

using System;

using System.ComponentModel;

using System.Globalization;

using Thrume.Domain.EntityIds;

namespace Thrume.Domain.EntityIds.Converters;

public class ConversationIdTypeConverter : TypeConverter

{

public override bool CanConvertFrom(ITypeDescriptorContext? context, Type sourceType)

{

return sourceType == typeof(string) || base.CanConvertFrom(context, sourceType);

}

public override object? ConvertFrom(ITypeDescriptorContext? context, CultureInfo? culture, object value)

{

if (value is string stringValue)

{

if (Guid.TryParse(stringValue, out var guid))

{

return new ConversationId(guid);

}

}

return base.ConvertFrom(context, culture, value);

}

public override bool CanConvertTo(ITypeDescriptorContext? context, Type? destinationType)

{

return destinationType == typeof(string) || base.CanConvertTo(context, destinationType);

}

public override object? ConvertTo(ITypeDescriptorContext? context, CultureInfo? culture, object? value, Type destinationType)

{

if (destinationType == typeof(string) && value is ConversationId conversationId)

{

return conversationId.Value.ToString();

}

return base.ConvertTo(context, culture, value, destinationType);

}

}

using System.ComponentModel;

namespace Thrume.Domain.EntityIds;

[TypeConverter(typeof(AccountIdTypeConverter))]

//TODO: do i have another way to do this?

public readonly record struct AccountId(Guid Value)

{

public override string ToString() => Value.ToString();

public static implicit operator Guid(AccountId id) => id.Value;

public static implicit operator AccountId(Guid id) => new(id);

}

using System.ComponentModel;

using System.Globalization;

namespace Thrume.Domain.EntityIds;

public sealed class AccountIdTypeConverter : TypeConverter

{

public override bool CanConvertFrom(ITypeDescriptorContext? context, Type sourceType)

{

return sourceType == typeof(string) || base.CanConvertFrom(context, sourceType);

}

public override object? ConvertFrom(ITypeDescriptorContext? context, CultureInfo? culture, object value)

{

//if (value is not string stringValue) return base.ConvertFrom(context, culture, value);

// if (string.IsNullOrWhiteSpace(stringValue))

// {

// throw new NotSupportedException("Cannot convert an empty string to AccountId.");

// }

if (Guid.TryParse(value as string, out var guid)) return new AccountId(guid);

throw new NotSupportedException($"Cannot convert '{value}' to AccountId.");

}

}

using System.ComponentModel;

namespace Thrume.Domain.EntityIds;

[TypeConverter(typeof(CommentIdTypeConverter))]

public readonly record struct CommentId(Guid Value)

{

public override string ToString() => Value.ToString();

public static implicit operator Guid(CommentId id) => id.Value;

public static implicit operator CommentId(Guid id) => new(id);

}

using System.ComponentModel;

using System.Globalization;

namespace Thrume.Domain.EntityIds;

public sealed class CommentIdTypeConverter : TypeConverter

{

public override bool CanConvertFrom(ITypeDescriptorContext? context, Type sourceType)

{

return sourceType == typeof(string) || base.CanConvertFrom(context, sourceType);

}

public override object ConvertFrom(ITypeDescriptorContext? context, CultureInfo? culture, object value)

{

if (!Guid.TryParse(value as string, out var guid)) throw new NotSupportedException($"Cannot convert '{value}' to CommentId.");

return new CommentId(guid);

}

}

using System;

using System.ComponentModel;

using Thrume.Domain.EntityIds.Converters;

namespace Thrume.Domain.EntityIds;

[TypeConverter(typeof(ConversationIdTypeConverter))]

public readonly record struct ConversationId(Guid Value)

{

public override string ToString() => Value.ToString();

public static implicit operator Guid(ConversationId id) => id.Value;

public static implicit operator ConversationId(Guid id) => new(id);

}

using System.ComponentModel;

using System.Globalization;

namespace Thrume.Domain.EntityIds;

public readonly record struct ImageId(string ImageUrl);

public sealed class ImageIdTypeConverter : TypeConverter

{

public override bool CanConvertFrom(ITypeDescriptorContext? context, Type sourceType) =>

sourceType == typeof(string) || base.CanConvertFrom(context, sourceType);

public override object? ConvertFrom(ITypeDescriptorContext? context, CultureInfo? culture, object value) =>

new ImageId(value as string ?? throw new InvalidOperationException($"Cannot convert {value} as imageId"));

}

namespace Thrume.Domain.EntityIds;

public readonly record struct MessageId(Guid Value)

{

public override string ToString() => Value.ToString();

public static implicit operator Guid(MessageId id) => id.Value;

public static implicit operator MessageId(Guid id) => new(id);

}

using System.ComponentModel;

namespace Thrume.Domain.EntityIds;

[TypeConverter(typeof(PostIdTypeConverter))]

public readonly record struct PostId(Guid Value)

{

public override string ToString() => Value.ToString();

public static implicit operator Guid(PostId id) => id.Value;

public static implicit operator PostId(Guid id) => new(id);

}

using System.ComponentModel;

using System.Globalization;

namespace Thrume.Domain.EntityIds;

public sealed class PostIdTypeConverter : TypeConverter

{

public override bool CanConvertFrom(ITypeDescriptorContext? context, Type sourceType)

{

return sourceType == typeof(string) || base.CanConvertFrom(context, sourceType);

}

public override object ConvertFrom(ITypeDescriptorContext? context, CultureInfo? culture, object value)

{

if (!Guid.TryParse(value as string, out var guid)) throw new NotSupportedException($"Cannot convert '{value}' to PostId.");

return new PostId(guid);

}

}

using Amazon.S3;

using Amazon.S3.Model;

using Thrume.Common;

namespace Thrume.Infrastructure;

public static class AmazonS3Extensions

{

public static async Task<Result<PutObjectResponse>> PutResObjectAsync(this IAmazonS3 amazonS3, PutObjectRequest req)

{

try

{

return await amazonS3.PutObjectAsync(req);

}

catch (Exception e)

{

return Result<PutObjectResponse>.Failure(e.ToProblemDetails(-1));

}

}

}

using Microsoft.AspNetCore.Diagnostics;

using Microsoft.AspNetCore.Http;

using Microsoft.Extensions.Logging;

using Thrume.Common;

namespace Thrume.Infrastructure;

public class GlobalExceptionHandler : IExceptionHandler

{

private readonly ILogger<GlobalExceptionHandler> \_logger;

public GlobalExceptionHandler(ILogger<GlobalExceptionHandler> logger)

{

\_logger = logger;

}

public async ValueTask<bool> TryHandleAsync(HttpContext httpContext, Exception exception, CancellationToken cancellationToken)

{

\_logger.LogError(exception, "Exception occured: {Message}", exception.Message);

var problemDetails = new ProblemDetails

{

Detail = exception.Message,

Status = StatusCodes.Status500InternalServerError

};

await httpContext.Response.WriteAsJsonAsync(problemDetails, cancellationToken: cancellationToken);

return true;

}

}

using Google.Apis.Auth.OAuth2;

using Google.Apis.Util.Store;

using MailKit.Net.Smtp;

using MailKit.Security;

using Microsoft.AspNetCore.Identity;

using Microsoft.Extensions.Logging;

using MimeKit;

namespace Thrume.Infrastructure;

public sealed class IdentityEmailSender<TEntity> : IEmailSender<TEntity> where TEntity : class

{

private readonly ILogger<IdentityEmailSender<TEntity>> \_logger;

//TODO: NOT IDEAL

private readonly UserCredential \_userCredentials;

public IdentityEmailSender(ILogger<IdentityEmailSender<TEntity>> logger)

{

\_logger = logger;

\_userCredentials = Setup(CancellationToken.None).GetAwaiter().GetResult();

}

public async Task SendConfirmationLinkAsync(TEntity user, string email, string confirmationLink)

{

var smtpClient = new SmtpClient();

try

{

await smtpClient.ConnectAsync("smtp.gmail.com", 587, SecureSocketOptions.StartTls);

await smtpClient.AuthenticateAsync(new SaslMechanismOAuth2(\_userCredentials.UserId,

\_userCredentials.Token.AccessToken));

var message = new MimeMessage();

message.From.Add(new MailboxAddress("wildchild", "wildchild250336@gmail.com"));

message.To.Add(new MailboxAddress("asdasd", email));

message.Subject = "Test Email via OAuth";

var messageText = $"<p>Please confirm your account by <a href='{confirmationLink}'>clicking here</a>.</p>";

message.Body = new TextPart("html") { Text = messageText };

await smtpClient.SendAsync(message);

await smtpClient.DisconnectAsync(true);

}

catch (Exception e)

{

\_logger.LogError(e, "Cant send confirmation link to {to}", email);

throw;

}

finally

{

await smtpClient.DisconnectAsync(true);

smtpClient.Dispose();

}

}

public Task SendPasswordResetCodeAsync(TEntity user, string email, string resetCode)

{

throw new NotImplementedException();

}

public Task SendPasswordResetLinkAsync(TEntity user, string email, string resetLink)

{

throw new NotImplementedException();

}

private async Task<UserCredential> Setup(CancellationToken ct)

{

const string user = "wildchild250336@gmail.com";

const string api = "https://mail.google.com/";

await using var stream = new FileStream("credentials.json", FileMode.Open, FileAccess.Read);

var googleClientSecrets = await GoogleClientSecrets.FromStreamAsync(stream, ct);

var credentials = await GoogleWebAuthorizationBroker.AuthorizeAsync(

googleClientSecrets.Secrets,

[api],

user,

ct,

new FileDataStore("token.json", true));

if (!credentials.Token.IsStale) return credentials; //TODO

await credentials.RefreshTokenAsync(ct);

await Setup(ct);

return credentials;

}

}

using Thrume.Common;

namespace Thrume.Infrastructure;

public interface IFileStorageRepository

{

Task<Result<string>> UploadFileAsync(Stream fileStream, string fileName, string contentType);

Task<bool> DeleteFileAsync(string fileName);

}

using Amazon.S3;

using Amazon.S3.Model;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Options;

using Thrume.Common;

using Thrume.Configuration;

using static Thrume.Common.Result<string>;

using ImageUrl = string;

namespace Thrume.Infrastructure;

public sealed class ImageStorageRepository : IFileStorageRepository

{

private readonly IAmazonS3 \_s3Client;

private readonly MinioConfiguration \_configuration;

private readonly ILogger<ImageStorageRepository> \_logger;

public ImageStorageRepository(IAmazonS3 s3Client, IOptions<MinioConfiguration> settings,

ILogger<ImageStorageRepository> logger)

{

\_s3Client = s3Client;

\_configuration = settings.Value;

\_logger = logger;

}

public async Task<Result<ImageUrl>> UploadFileAsync(Stream fileStream, string fileName, string contentType)

{

(var isImageValid, ProblemDetails? problemDetails) = ImageValidation.IsImageValid(fileStream, fileName, contentType);

if (!isImageValid)

{

return Failure((ProblemDetails)problemDetails);

}

fileName = Guid.CreateVersion7().ToString();

var putRequest = new PutObjectRequest

{

BucketName = \_configuration.BucketName,

Key = fileName,

InputStream = fileStream,

ContentType = contentType

};

var putObjectResponse = await \_s3Client.PutResObjectAsync(putRequest);

\_logger.LogInformation("File '{FileName}' uploaded successfully to bucket '{BucketName}'.", fileName, \_configuration.BucketName);

return putObjectResponse.IsSuccess

? $"{\_configuration.ServiceUrl}/{\_configuration.BucketName}/{fileName}"

: Failure(putObjectResponse.Errors);

}

public async Task<bool> DeleteFileAsync(string fileName)

{

throw new NotImplementedException();

}

}

using Thrume.Common;

namespace Thrume.Infrastructure;

public static class ImageValidation

{

private static readonly Dictionary<string, List<byte[]>> \_fileSignature = new() //TODO

{

["jpeg"] = [

[0xFF, 0xD8, 0xFF, 0xE0],

[0xFF, 0xD8, 0xFF, 0xE2],

[0xFF, 0xD8, 0xFF, 0xE3]

],

["jpg"] = [

[0xFF, 0xD8, 0xFF, 0xE0],

[0xFF, 0xD8, 0xFF, 0xE2],

[0xFF, 0xD8, 0xFF, 0xE3]

],

["png"] = [

[0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A]

]

};

public static (bool, ProblemDetails?) IsImageValid(Stream fileStream, string fileName, string contentType)

{

if (string.IsNullOrEmpty(contentType) || !contentType.Contains("image"))

return (false, DomainErrors.InvalidImageType);

var key = fileName.Split('.')[1];

if(!\_fileSignature.TryGetValue(key, out var signatures))

return (false, DomainErrors.InvalidImageType);

if (fileStream.Length >= 10 \* 1024 \* 1024) //10 MB

return (false, DomainErrors.ImageToLarge);

var binReader = new BinaryReader(fileStream);

var headerBytes = binReader.ReadBytes(signatures.Max(m => m.Length));

return (signatures.Any(signature =>

headerBytes.Take(signature.Length).SequenceEqual(signature)), null);

}

}

using Microsoft.AspNetCore.Http;

using Microsoft.EntityFrameworkCore;

using Thrume.Common;

using Thrume.Database;

using Thrume.Domain.DTOs;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

using Thrume.Infrastructure;

namespace Thrume.Services;

public sealed class AccountService

{

private readonly AppDbContext \_dbContext;

private readonly IFileStorageRepository \_imageRepository;

public AccountService(AppDbContext dbContext, IFileStorageRepository imageRepository)

{

\_dbContext = dbContext;

\_imageRepository = imageRepository;

}

public async Task UpdateAvatarAsync(AccountId accountId, IFormFile file)

{

var result = await \_imageRepository.UploadFileAsync(file.OpenReadStream(), file.FileName, file.ContentType);

if (result.IsSuccess)

await \_dbContext.AccountDbSet

.Where(acc => acc.Id == accountId)

.ExecuteUpdateAsync(setters

=> setters.SetProperty(a => a.PictureUrl, result.Value));

}

public async Task CreatePostsAsync(AccountId id, CreatePostRequest request)

{

if (request.Images.Count >= 10) return;

List<string> imageLinks = new() { Capacity = request.Images.Count };

foreach (var formFile in request.Images)

{

var result = await \_imageRepository.UploadFileAsync(formFile.OpenReadStream(), formFile.FileName,

formFile.ContentType);

if (result.IsFault) return;

imageLinks.Add(result.Value!);

}

Account? account = await \_dbContext.AccountDbSet.FindAsync(id);

if (account is null) return;

Post post = new()

{

Author = account,

Content = request.Content,

Images = imageLinks.Select(link => new Image {Id = new ImageId(link)}).ToList()

};

account.Posts.Add(post);

await \_dbContext.SaveChangesAsync();

}

}

using Microsoft.EntityFrameworkCore;

using Thrume.Common;

using Thrume.Database;

using Thrume.Domain.DTOs; // DTOs need to be created

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

namespace Thrume.Services;

public class CommentService

{

private readonly AppDbContext \_dbContext;

public CommentService(AppDbContext dbContext)

{

\_dbContext = dbContext;

}

public async Task CreateCommentAsync(PostId postId, AccountId authorId, string content, CommentId? parentCommentId)

{

if (string.IsNullOrWhiteSpace(content) || content.Length > 1000)

{

return;

}

var postExists = await \_dbContext.PostDbSet.AnyAsync(p => p.Id == postId);

if (!postExists)

{

return;

}

if (parentCommentId.HasValue)

{

var parentExists = await \_dbContext.CommentDbSet

.AnyAsync(c => c.Id == parentCommentId.Value && c.PostId == postId);

if (!parentExists)

{

return;

}

}

var comment = new Comment

{

Content = content,

AuthorId = authorId,

PostId = postId,

ParentCommentId = parentCommentId,

};

\_dbContext.CommentDbSet.Add(comment);

await \_dbContext.SaveChangesAsync();

}

public async Task DeleteCommentAsync(CommentId commentId, AccountId userId)

{

var comment = await \_dbContext.CommentDbSet.FindAsync(commentId);

if (comment is null)

{

return;

}

if (comment.AuthorId != userId)

{

return;

}

\_dbContext.CommentDbSet.Remove(comment);

await \_dbContext.SaveChangesAsync();

}

}

// Thrume.Services/MessageService.cs

using Microsoft.EntityFrameworkCore;

using Thrume.Database;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

namespace Thrume.Services;

public class MessageService

{

private const int MaxMessageLength = 2000;

private readonly AppDbContext \_dbContext;

public MessageService(AppDbContext dbContext)

{

\_dbContext = dbContext;

}

/// <summary>

/// Finds an existing 1-on-1 conversation between two users or creates a new one.

/// </summary>

public async Task<Conversation?> StartOrGetConversationAsync(AccountId user1Id, AccountId user2Id)

{

if (user1Id == user2Id)

{

return null;

}

var existingConversation = await \_dbContext.ConversationDbSet

.Include(c => c.Participants)

.Where(c => c.Participants.Count == 2 &&

c.Participants.Any(p => p.Id == user1Id) &&

c.Participants.Any(p => p.Id == user2Id))

.FirstOrDefaultAsync();

if (existingConversation != null)

{

return existingConversation;

}

var user1 = await \_dbContext.AccountDbSet.FindAsync(user1Id);

var user2 = await \_dbContext.AccountDbSet.FindAsync(user2Id);

if (user1 == null || user2 == null)

{

return null;

}

var newConversation = new Conversation

{

CreatedAt = DateTimeOffset.UtcNow,

Participants = new List<Account> { user1, user2 }

};

\_dbContext.ConversationDbSet.Add(newConversation);

await \_dbContext.SaveChangesAsync();

return newConversation;

}

/// <summary>

/// Sends a message within a specific conversation.

/// </summary>

public async Task<Message?> SendMessageAsync(AccountId senderId, ConversationId conversationId, string content)

{

if (string.IsNullOrWhiteSpace(content) || content.Length > MaxMessageLength)

{

return null;

}

var conversation = await \_dbContext.ConversationDbSet

.Include(c => c.Participants)

.FirstOrDefaultAsync(c => c.Id == conversationId);

if (conversation == null || conversation.Participants.All(p => p.Id != senderId))

{

return null;

}

var message = new Message

{

ConversationId = conversationId,

SenderId = senderId,

Content = content,

SentAt = DateTimeOffset.UtcNow

};

\_dbContext.MessageDbSet.Add(message);

await \_dbContext.SaveChangesAsync();

return message;

}

/// <summary>

/// Retrieves messages for a given conversation, ordered by time (newest first), with pagination.

/// </summary>

public async Task<List<Message>> GetMessagesAsync(ConversationId conversationId, AccountId requesterId, int page = 1, int pageSize = 20)

{

var isParticipant = await \_dbContext.ConversationDbSet

.Where(c => c.Id == conversationId)

.AnyAsync(c => c.Participants.Any(p => p.Id == requesterId));

if (!isParticipant)

{

return new List<Message>();

}

if (page < 1) page = 1;

if (pageSize < 1) pageSize = 1;

if (pageSize > 100) pageSize = 100;

var messages = await \_dbContext.MessageDbSet

.Where(m => m.ConversationId == conversationId)

.Include(m => m.Sender)

.OrderByDescending(m => m.SentAt)

.Skip((page - 1) \* pageSize)

.Take(pageSize)

.AsNoTracking()

.ToListAsync();

return messages;

}

/// <summary>

/// Retrieves a list of conversations for a specific user.

/// </summary>

public async Task<List<Conversation>> GetConversationsAsync(AccountId userId)

{

var conversations = await \_dbContext.ConversationDbSet

.Where(c => c.Participants.Any(p => p.Id == userId))

.Include(c => c.Participants)

.OrderByDescending(c => c.CreatedAt)

.AsNoTracking()

.ToListAsync();

return conversations;

}

}

using Microsoft.EntityFrameworkCore;

using Thrume.Common;

using Thrume.Database;

using Thrume.Domain.Entity;

using Thrume.Domain.EntityIds;

namespace Thrume.Services;

public sealed class PostService

{

private readonly AppDbContext \_dbContext;

public PostService(AppDbContext dbContext)

{

\_dbContext = dbContext;

}

public async Task LikePostAsync(AccountId accountId, PostId postId)

{

var account = await \_dbContext.AccountDbSet.FindAsync(accountId);

var post = await \_dbContext.PostDbSet

.Include(p => p.LikedBy)

.FirstOrDefaultAsync();

if (account == null || post == null)

{

return;

}

if (post.LikedBy.Any(a => a.Id == accountId))

{

return;

}

post.LikedBy.Add(account);

await \_dbContext.SaveChangesAsync();

return;

}

public async Task UnlikePostAsync(AccountId accountId, PostId postId)

{

var account = await \_dbContext.AccountDbSet.FindAsync(accountId);

var post = await \_dbContext.PostDbSet

.Include(p => p.LikedBy)

.FirstOrDefaultAsync(p => p.Id == postId);

if (account == null || post == null)

{

return;

}

var likedAccount = post.LikedBy.FirstOrDefault(a => a.Id == accountId);

if (likedAccount == null)

{

return;

}

post.LikedBy.Remove(likedAccount);

await \_dbContext.SaveChangesAsync();

}

public async Task<Post?> GetPostByIdAsync(PostId postId, AccountId? currentAccountId = null) =>

await \_dbContext.PostDbSet

.Include(p => p.Author)

.Include(p => p.LikedBy)

.FirstOrDefaultAsync(p => p.Id == postId);

}