МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра Програмної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КУРСОВОЇ РОБОТИ

з дисципліни: «Бази даних»

Тема роботи: «Інформаційна система «Репозиторій пакунків». Колаборація над пакунками»

Виконав
ст. гр. ПЗПІ-23-3 Білоус А. А.

Керівник:
Ст. викл. каф. ПІ Широкопетлєва М. С.

Робота захищена на оцінку _____

Комісія:
Ст. викл. каф. ПІ Широкопетлєва М. С.
Доц. каф. ПІ Русакова Н. Є.
Доц. каф. ПІ Мазурова О.О.

Харків – 2024

Харківський національний університет радіоелектроніки

Кафедра Програмної інженерії

Дисципліна Бази даних

Спеціальність 121 Інженерія програмного забезпечення

Курс 2 **Група** ПЗПІ-23-3 **Семестр** 3

ЗАВДАННЯ

на курсову роботу студента

Білоуса Антона Андрійовича

1. Тема роботи: Інформаційна система «Репозиторій пакунків». Колаборація над пакунками.

2. Строк здачі закінченої роботи: 27.12.2024

3. Вихідні дані для роботи: методичні вказівки до виконання курсової роботи, вимоги до інформаційної системи, предметна область, що пов’язана з пакунковим репозиторієм та його менеджментом.

4. Зміст розрахунково-пояснювальної записки: вступ, аналіз предметної області; постановка задачі; проектування бази даних; опис програми; висновки; перелік джерел посилання.

5. Перелік графічного матеріалу: загальна діаграма класів, ER-діаграма, UML-діаграми, DFD-діаграма, схема БД в 1НФ, 2НФ, 3НФ, копії екранів (“скриншоти”) прикладної програми, приклади звітів прикладної програми.

6. Дата видачі завдання: 25.09.2024

КАЛЕНДАРНИЙ ПЛАН

Номер	Назва етапів курсової роботи	Строк виконання етапів роботи	Примітки
1	Аналіз предметної області	25.09.24 – 30.09.24	Виконано
2	Концептуальне моделювання	30.09.24 – 3.10.24	Виконано
2	Постановка задачі	3.10.24 – 10.10.24	Виконано
3	Побудова ER-діаграми та схеми БД	10.10.24 - 15.10.24	Виконано
4	Оформлення розділів 1, 2 та 3.1, 3.2 пояснювальної записки	15.10.24 – 18.10.24	Виконано
5	Перша контрольна точка з курсової роботи	20.10.24	Виконано
6	Нормалізація бази даних	20.10.24 – 15.11.24	Виконано
7	Створення програми	20.10.24 – 20.11.24	Виконано
8	Тестування програми, наповнення бази даних	20.11.24 – 5.12.24	Виконано
9	Друга контрольна точка з курсової роботи	7.12.24	Виконано
10	Реалізація остаточної версії програми	7.12.24 – 15.12.24	Виконано
11	Оформлення інших розділів пояснювальної записки	1.11.24 – 25.12.24	Виконано
12	Третя контрольна точка з курсової роботи	27.12.24	Виконано

Студент _____ Білоус А. А.,

Керівник _____ Ст. викл. каф. ПІ Широкопетлєва М. С..

«25» вересня 2024 р.

РЕФЕРАТ

Пояснювальна записка до курсової роботи: 73 с., 1 табл., 21 рис., 18 джерел.

АВТОМАТИЗАЦІЯ, БАЗА ДАНИХ, ПАКУНОК, РЕПОЗИТОРІЙ, AUR, MYSQL, RUST, SQL

Мета даної роботи – проектування та розробка інформаційної системи «Репозиторій пакунків. Колаборація над пакунками» спрямованої на забезпечення надійного зберігання інформації про програмні пакунки,та надання інструментів для колективної співпраці користувачів, що забезпечить ефективне середовище для спільної розробки та управління програмними пакунками.

Для реалізації інформаційної системи було обрано сучасний стек технологій, а саме: Rust – як основна мова програмування для всіх частин комп'ютерної програми, iced – бібліотека для побудови графічних інтерфейсів з Elm архітектурою, SQLx – бібліотека для низькорівневої роботи з базою даних, що забезпечує коректність SQL запитів і гнучкість, MySQL – як СУБД для зберігання інформації про пакунки, користувачів, та їх відносини, Neovim – як сучасний редактор коду для швидкої і зручної розробки.

Результат розробки – комп'ютерна програма, яка дозволяє зберігати та відображати інформацію про користувачів, пакунки, та відносини між ними, а також генерувати статистику про користувача та його пакунки. Комп'ютерна програма, створена з використанням мови програмування Rust є безпечною, правильною, надійною та швидкою.

ЗМІСТ

ВСТУП

У сучасному світі існують мільйони пакунків з програмним забезпеченням, програмними бібліотеками та іншою інформацією. Інформаційні системи управління цими пакунками стали критично важливим елементом сучасної розробки програмного забезпечення. Їх значення особливо зросло з розвитком відкритого програмного забезпечення та модульного підходу до розробки, де кожен проект може залежати від десятків, сотень, або навіть тисяч сторонніх компонентів.

Відсутність ефективних систем управління пакунками призводить до численних проблем у процесі розробки. Розробники стикаються з труднощами при пошуку потрібних бібліотек, виникають конфлікти версій, ускладнюється процес оновлення залежностей, а також з’являються проблеми з безпекою через використання застарілих версій компонентів. Це суттєво сповільнює процес розробки та може призвести до значних фінансових втрат.

Метою цієї курсової роботи є розробка інформаційної системи «Репозиторій пакунків. Колаборація над пакунками», яка спрощує процес менеджменту пакунків, створення їх відносин, керування залежностями, та надання користувачам різних ролей у розвитку репозиторію. У процесі роботи над системою було проведено детальний аналіз схожої системи AUR [1], котра ефективно використовується для надання користувачам дистрибутиву Arch Linux [2] можливості публікувати свої пакунки. Було спроектовано реляційну базу даних і розроблено комп’ютерну програму, яка дозволяє взаємодіяти з репозиторієм.

Комп’ютерну програму повністю написано мовою програмування Rust [3], для графічного інтерфейсу використовується бібліотека iced [4], для взаємодії з базою даних використовується бібліотека SQLx [5]. Інформація зберігається у базі даних MySQL [6]. Розробка виконувалася у текстовому редакторі Neovim [7].

1 АНАЛІЗ ТА КОНЦЕПТУАЛЬНЕ МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Дослідження предметної області є ключовим етапом у розробці інформаційної системи «Репозиторій пакунків. Колаборація над пакунками». Основною метою даного аналізу є визначення функціональних вимог та технічних особливостей системи, необхідних для ефективного управління програмними пакунками та забезпечення продуктивної співпраці розробників.

Репозиторій пакунків являє собою централізоване сховище інформації про програмне забезпечення, що забезпечує зберігання, версіонування та розповсюдження програмних компонентів. В сучасних умовах критично важливою є автоматизація таких процесів як управління залежностями, контроль версій, перевірка сумісності та забезпечення безпеки пакунків. Це дозволяє значно підвищити ефективність розробки програмного забезпечення та мінімізувати ризики, пов’язані з використанням сторонніх компонентів.

Для створення ефективної системи управління пакунками необхідно ретельно проаналізувати існуючі рішення та їх особливості. Такий аналіз допомагає виявити найбільш важливі функціональні можливості та уникнути потенційних проблем при проектуванні власної системи.

В якості системи аналогу розглянемо Arch User Repository (AUR)[1] – репозиторій користувацьких пакунків для дистрибутиву Arch Linux [2]. AUR є яскравим прикладом успішної реалізації концепції кооперативної розробки та управління пакунками. Система надає користувачам можливість самостійно створювати та поширювати пакунки, які не входять до офіційних репозиторіїв пакунків Arch Linux.

При відвідуванні головної сторінки AUR (див. рис. 1.1) можна побачити статистку свого облікового запису, всього репозиторію а також останні оновлення пакунків. За допомогою поля пошуку можна перейти до сторінки пошуку пакунків (див. рис. 1.2).

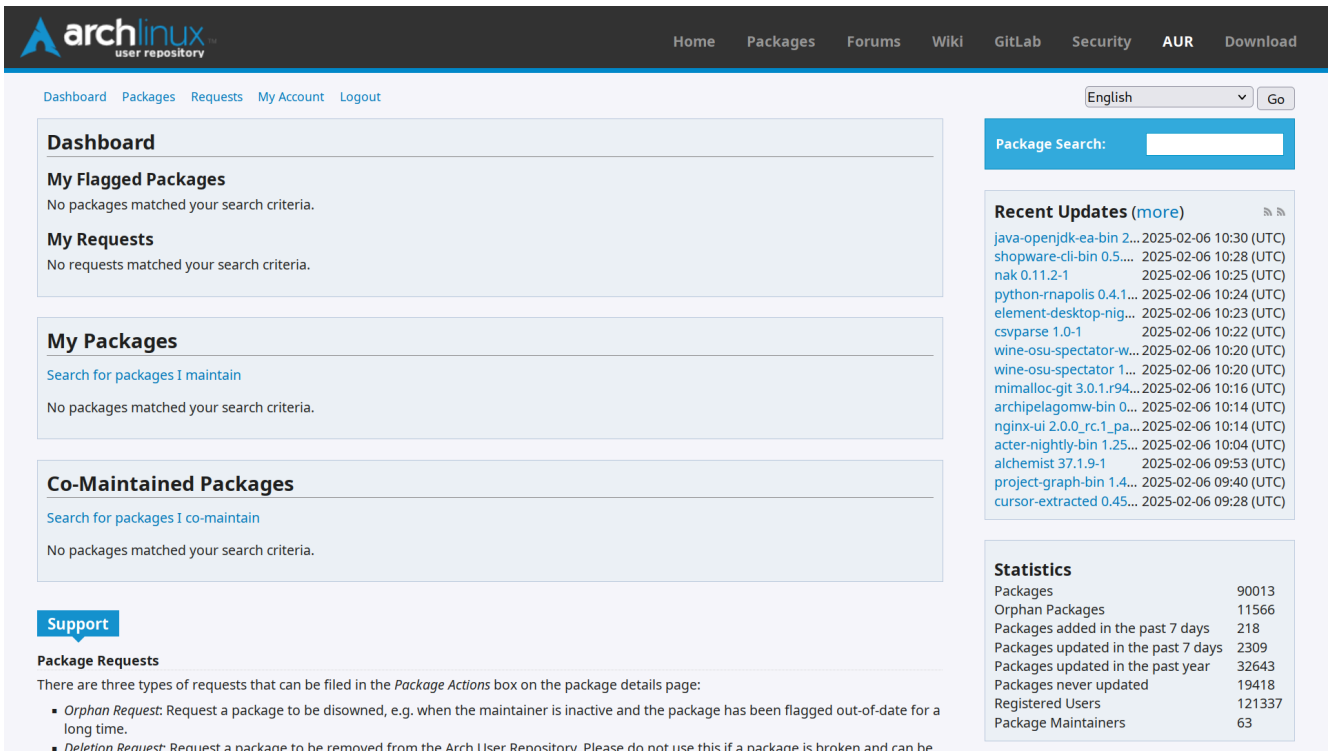

Рисунок 1.1 – Головна сторінка AUR (за даними [1])

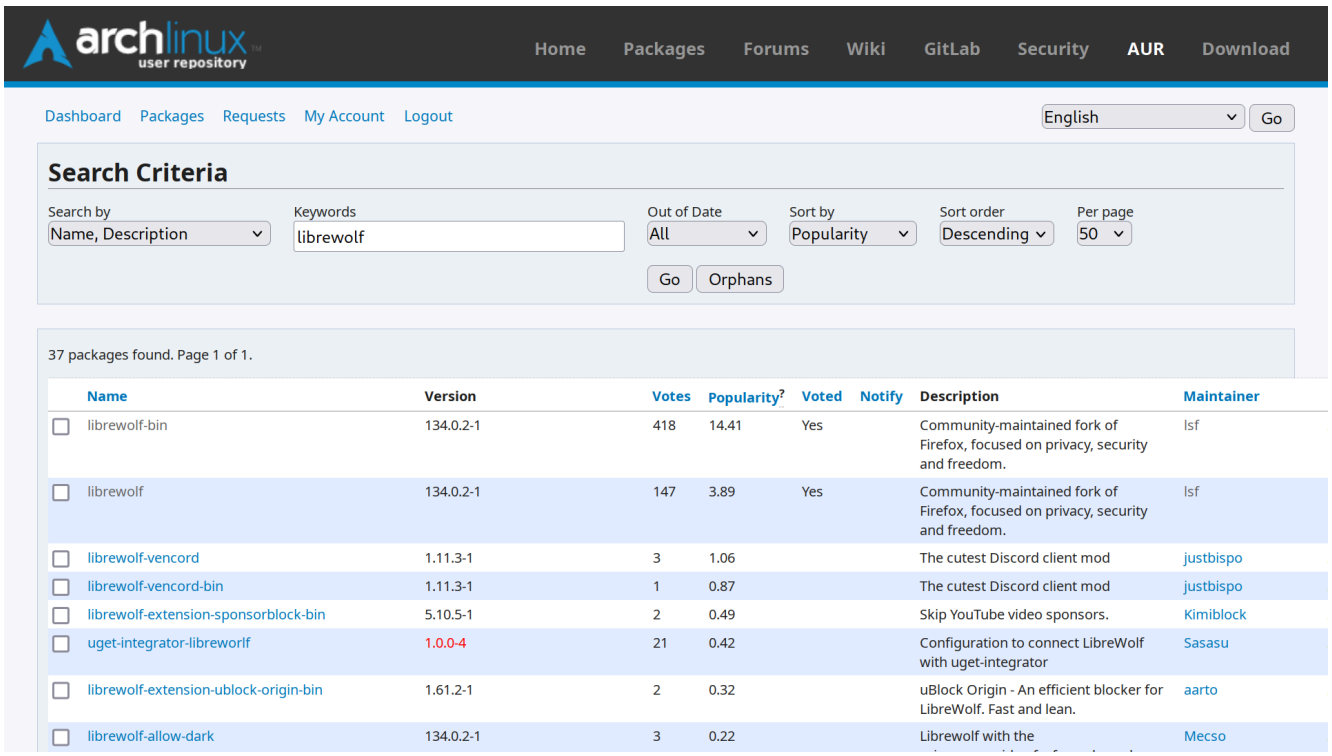

Рисунок 1.2 – Сторінка пошуку AUR (за даними [8])

Зі сторінки пошуку пакунків можна перейти до сторінки інформації пакунку, або ж до сторінки користувача який відповідає за супроводження пакунку. На сторінці інформації пакунку (див. рис. 1.3) можна побачити що пакунок має базу пакунку, пошукові слова, ліцензії, різні ролі користувачів, інформацію про

залежності, тощо. На сторінці інформації про користувача (див. рис. 1.4) можна побачити різні атрибути пов’язані з обліковим записом.

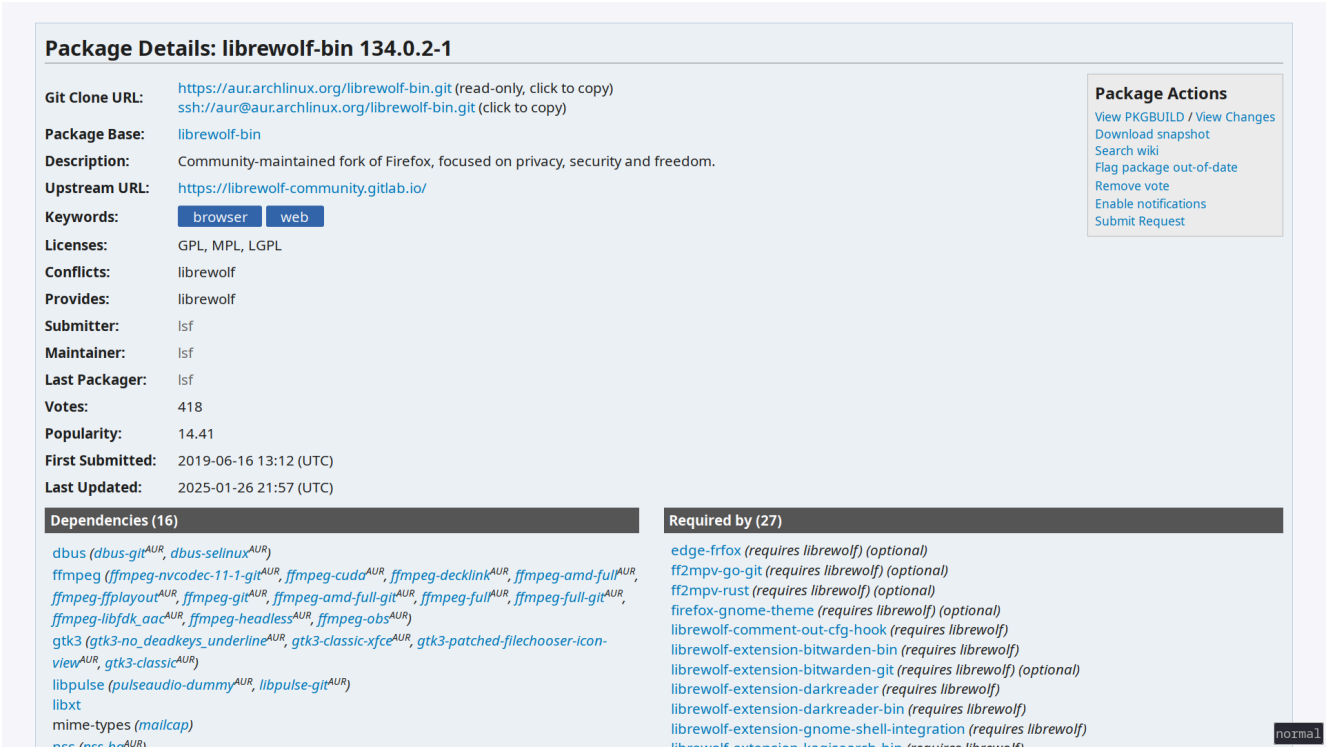

Рисунок 1.3 – Деталі пакунку в AUR (за даними [9])

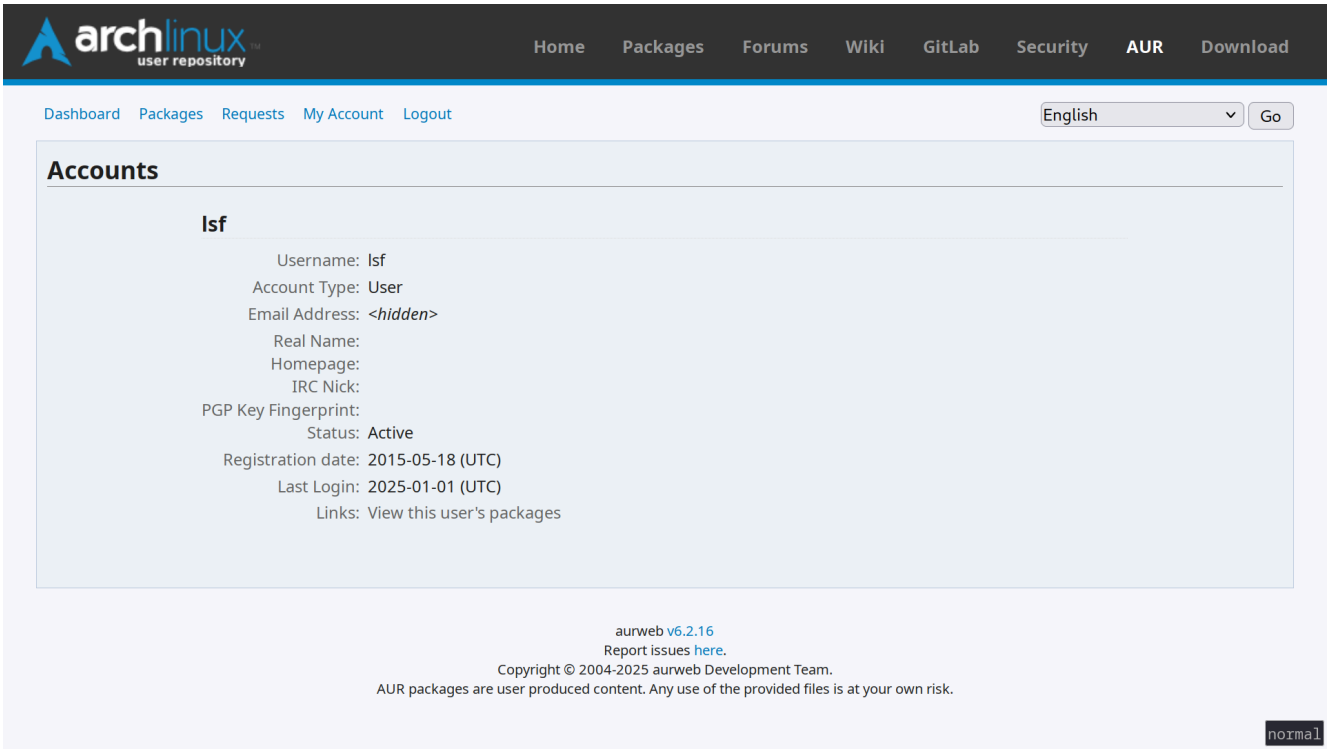

Рисунок 1.4 – Деталі користувача в AUR (за даними [10])

Навігація по репозиторію здебільшого здійснюється за допомогою гіперпосилань з параметрами для сторінки пошуку, де можна обрати багато критерій пошуку. Наприклад, при натисканні посилання «View this user's packages» в профілі користувача lsf, можна побачити всі пакунки які цей користувач підтримує (див. рис. 1.5).

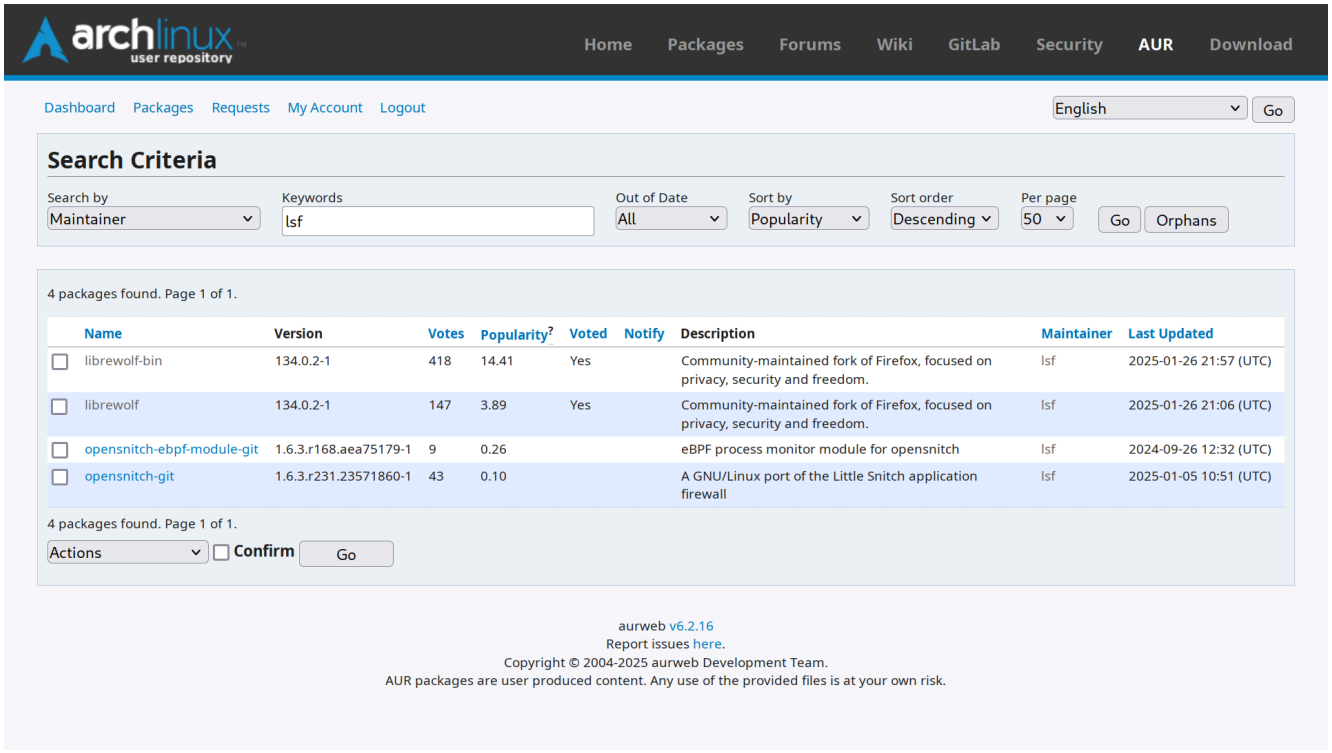

Рисунок 1.5 – Фунціонал пошуку AUR (за даними [11])

В контексті розробки пакункового репозиторію важливо визначити основні ролі користувачів та їхні потреби. Зареєстровані користувачі виступають основним рушієм системи, створюючи та підтримуючи програмні компоненти, в той час як інші учасники мають змогу отримувати інформацію про пакунки: залежності, відносини, ліцензії, ресурс походження, тощо.

Аналіз інформаційної системи виявив наступні ключові об'єкти предметної області: незареєстрований користувач, зареєстрований користувач, пакунок, база пакунку, залежності пакунків, відносини пакунків, роль користувача для бази пакунків. Інформаційна система передбачає взаємодію як для незареєстрованих, так і для зареєстрованих користувачів. Вона автоматизує процеси пов'язані з відстеженням залежностей пакунків, прав користувачів, пошуком пакунків і генеруванням статистики.

Кожний незареєстрований користувач має можливість ефективно:

- здійснювати пошук пакунків за різними параметрами;
- переглядати інформацію про пакунки, їх бази, залежності та відносини;
- переглядати статистику репозиторію;
- увійти у існуючий, або створити новий обліковий запис.

Разом з можливостями незареєстрованих користувачів, зареєстровані користувачі мають можливість ефективно:

- переглядати інформацію про користувачів;
- створювати пакунки та бази пакунків;
- видаляти власні пакунки та бази пакунків;
- додавати, видаляти та оновлювати залежності та відносини для власних пакунків;
- додавати, видаляти та оновлювати ролі інших користувачів для власних пакунків;
- оновлювати чужі пакунки та бази пакунків до яких були надані відповідні ролі;
- переглядати статистику свого облікового запису і своїх пакунків;
- змінювати інформацію свого облікового запису;
- вийти з облікового запису або видалити його;

На основі цього аналізу було визначено критичні вимоги до функціоналу репозиторію пакунків. Ця інформація становитиме основу для проектування системи, яка забезпечить ефективну кооперацію над програмними компонентами та автоматизує ключові процеси їх розробки та супроводу.

1.2 Концептуальне моделювання предметної області

Для виконання детального концептуального моделювання інформаційної системи «Репозиторій пакунків. Колаборація над пакунками» потрібно проаналізувати взаємозв’язки між основними об’єктами системи: користувачами, їх ролями, базами пакунків, пакунками, їх залежностями та відносинами.

В предметній області існують наступні поняття та зв’язки:

- користувач може мати багато типів ролей для багатьох баз пакунків;
- база пакунку може мати багато пакунків;

– пакунок обов’язково має одну базу пакунку;
– пакунок може мати багато типів відносин з назвами інших пакунків;
– пакунок може мати багато типів залежностей від назв інших пакунків.

Для розуміння структури інформаційної системи спроектовано загальну діаграму класів (див. рис. 1.6), яка стане основою для проектування бази даних та визначення функціональних вимог до системи.

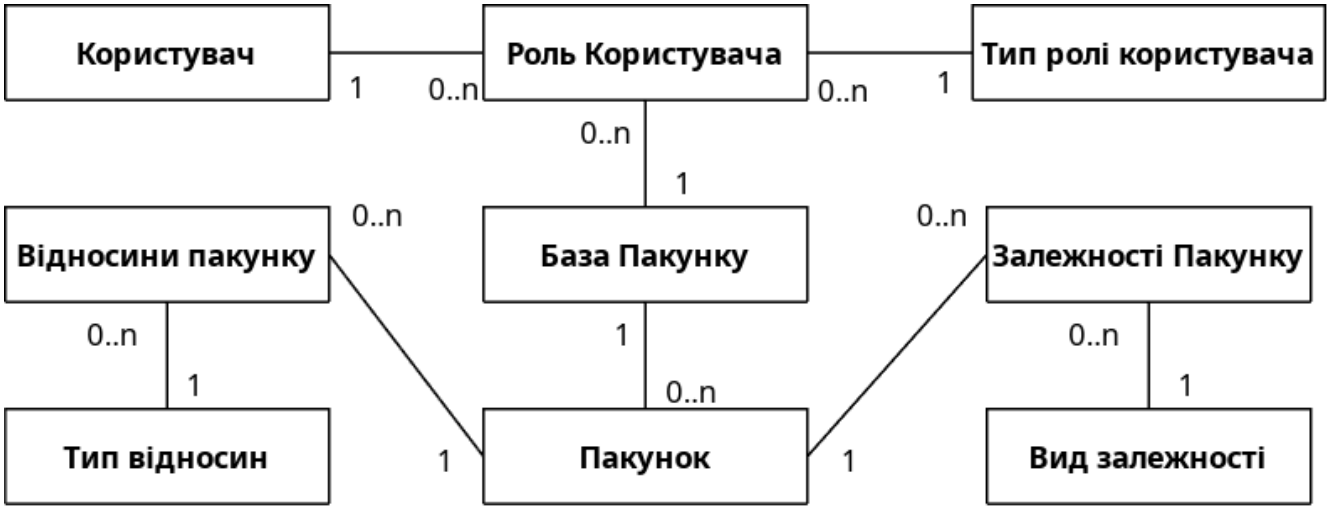

Рисунок 1.6 – Загальна діаграма класів (рисунок виконано самостійно)

На підставі проведеного аналізу для опису виділених понять предметної області можна визначити дані які буде зберігати система:

– об’єкт користувача буде зберігати інформацію про користувачів системи. Він буде містити ім’я користувача, електронну пошту, пароль, час останнього використання облікового запису, його створення, і оновлення. Ця інформація необхідна для автентифікації та авторизації користувачів, а також для аудиту дій користувачів в системі;
– об’єкт бази пакунку буде зберігати базову інформацію про групи пакунків. Він дозволить об’єднувати пакунки зі спільними компонентами та користувачами. Містить ім’я та опис бази пакунків, а також час створення та останнього оновлення. Ця інформація дозволяє групувати пакунки за їх базовою функціональністю;
– об’єкт типу ролі користувача буде описувати ролі користувачів у контексті баз пакунків. Містить назву ролі та опис. Приклади можливих ролей: submitter (відправник), packager (пакувальник), maintainer

(супроводжуючий), flagger (позначник). Цей об’єкт визначає дозволи користувачів на певні дії з групами пакунків;

- об’єкт ролі користувача буде прив’язувати користувачів до баз пакунків, та визначати їх ролі в цих базах. Містить групу пакунків, користувача, тип ролі, а також коментар від користувача. Ця інформація дозволяє визначити, хто, чому, і які права має в кожній групі пакунків;
- об’єкт пакунку буде зберігати інформацію про окремі пакунки. Містить групу пакунків, назву пакунку, версію, опис, веб-покликання на ресурс пакунку, час позначення, створення та останнього оновлення. Ця інформація є основною для ідентифікації та опису кожного пакунку;
- об’єкт виду залежності буде визначати типи залежностей пакунків (наприклад: проста залежність, залежність для збірки пакунку, опціональна залежність). Містить номер типу залежності та її назву. Цей об’єкт дозволить класифікувати залежності пакунків;
- об’єкт залежностей пакунку буде зберігати інформацію про залежності пакунків. Він буде містити архітектуру, умову, опис, пакунок, тип залежності та назву пакунку який є залежністю. Ця інформація дозволяє визначити, від яких інших назв пакунків залежить даний пакунок;
- об’єкт виду відносин буде визначати типи відносин між пакунками (наприклад: конфліктує, надає, замінює). Містить номер типу відношення та його назву. Цей об’єкт дозволяє класифікувати відносини між пакунками;
- об’єкт відносин пакунків буде зберігати інформацію про відносини пакунків. Містить архітектуру, вимоги, пакунок, тип відношення та назву пакунку до якого формується відношення. Ця інформація дозволяє визначити, які пакунки конфліктують з даним пакунком, які функції він надає, і які пакунки він замінює.

Для ефективного опису функціональної структури та процесів пакункового репозиторію, необхідно побудувати DFD-діаграму, яка описує потоки даних. Ця діаграма відобразить основні етапи обробки інформації та взаємодію між різними компонентами системи під час виконання бізнес-задачі отримання інформації про пакунки.

Зареєстровані користувачі створюють нові бази пакунків та пакунки та налаштовують залежності і відносини для них, а також надають ролі іншим зареєстрованим користувачам до своїх баз пакунків. Для виконання своїх цілей вони користуються інформацією про пакунки інших користувачів. Для отримання інформації про пакунки зареєстровані, а також незареєстровані користувачі використовують функції пошуку. Створимо DFD-діаграму (див. рис. 1.7), що наочно відобразить послідовність процесів та потоків даних під час виконання користувачем кроків для досягнення цільової задачі перегляду інформації про пакунок.

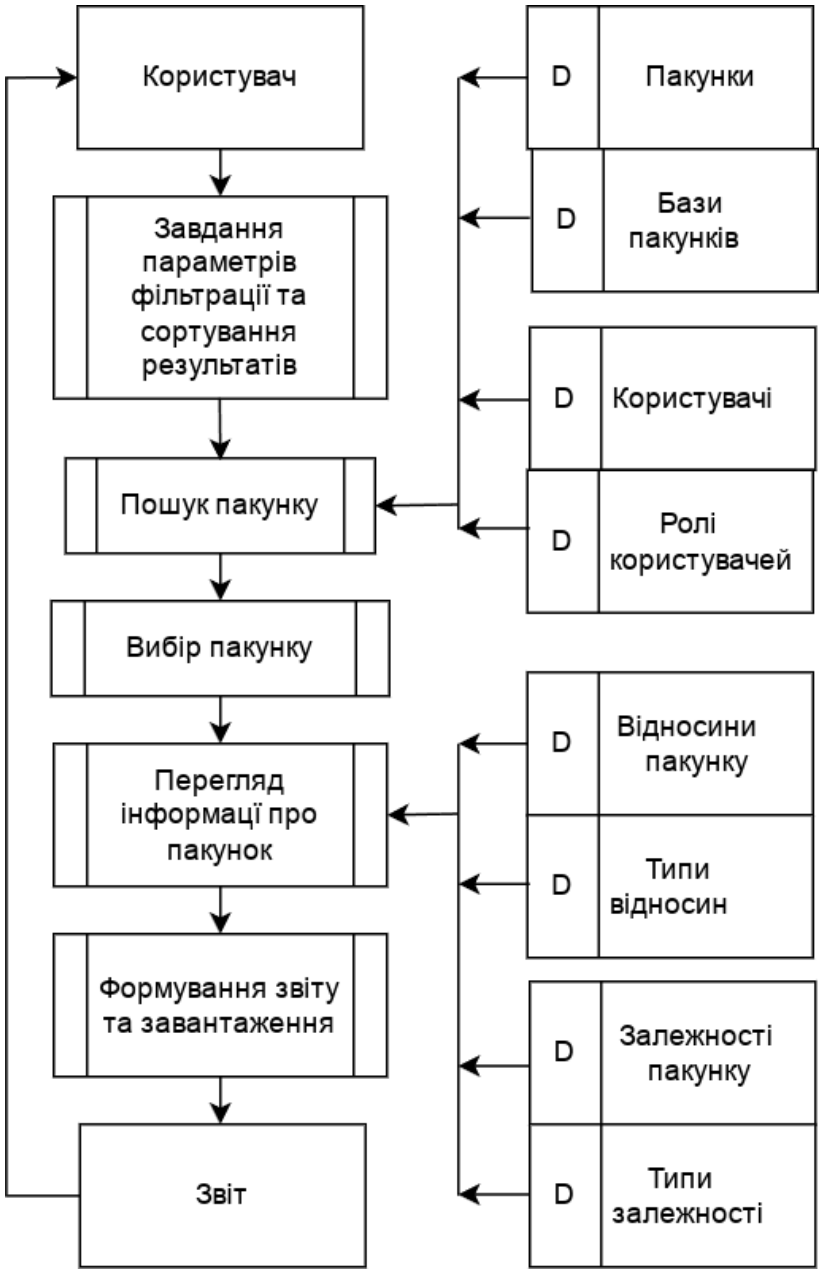

Рисунок 1.7 – Діаграма потоків даних (рисунок виконано самостійно)

Для забезпечення ефективної розробки та впровадження інформаційної системи критично важливим етапом є побудова діаграми варіантів використання (Use-Case Diagram). Дана діаграма (див. рис. 1.8) є фундаментальним інструментом моделювання, який дозволяє показати та систематизувати більшість можливих сценаріїв взаємодії між системою та її користувачами.

Рисунок 1.8 – Use-Case діаграма (рисунок виконано самостійно)

Діаграма варіантів використання репозиторію пакунків відображає взаємодію між системою та двома групами користувачів: будь-якими користувачами та тільки

автентифікованими. Кожна група користувачів має різний за розміром набір прав та можливостей.

Система надає функціональні можливості для управління пакунками, включаючи їх створення, модифікацію та адміністрування. Користувачі можуть переглядати інформацію про пакунки, генерувати звіти та керувати залежностями між пакунками відповідно до їхніх прав доступу.

Значущість діаграми варіантів використання полягає в тому, що вона служить потужним інструментом для:

- Визначення та узгодження вимог користувачів до системи
- Ідентифікації ключових бізнес-процесів та їх взаємозв’язків
- Оптимізації архітектурних рішень на ранніх етапах проектування
- Забезпечення відповідності розроблюваної системи реальним потребам користувачів

Важливою особливістю репозиторіїв пакунків є можливість керувати програмним забезпеченням та надавати користувачам доступ до актуальних версій програм. У репозиторіях пакунків основний документообіг пов’язаний з управлінням інформацією пакунків, їх відносинами та залежностями. Репозиторії ведуть облік завантажень пакунків та формують звітність про їх використання для аналізу популярності та стабільності.

Отже, документообіг репозиторію пакунків складається з наступних документів:

- інформація пакунку: назва, версія, опис, веб-покликання на веб-ресурс пакунку, дата позначення, дата створення, дата оновлення;
- файл потреб пакунку: унікальний ідентифікатор пакунку, назва, версія, архітектура процесора, список залежностей (обов’язкових та опціональних), конфлікти з іншими пакунками, контрольні суми файлів, розмір встановленого пакунку;
- журнал активності пакунку: статистика завантажень, оцінки користувачів, коментарі, повідомлення про помилки, запити на оновлення, історія модифікацій;
- інструкція зі збірки: скрипт для автоматизованої збірки пакунку, список додаткових залежностей для збірки, параметри конфігурації, інструкції

з тестування, відомості про середовище збірки та тестування, контактна інформація супроводжувача пакунку.

У процесі проектування бази даних репозиторію пакунків важливо визначити та врахувати обмеження цілісності даних, які забезпечують коректність та узгодженість інформації. Для кожного обмеження необхідно визначити стратегію обробки можливих порушень, щоб гарантувати надійне функціонування системи та збереження важливих даних:

а) при видаленні облікового запису:
   1) якщо обліковий запис володіє базами пакунків для яких немає інших володарів – видалення заборонено;
   2) якщо обліковий запис було видалено – його ролі мають бути видаленими, але бази пакунків та пакунки продовжують існувати.

б) при видаленні бази пакунку:
   1) всі пов’язані ролі користувачів мають бути видалені;
   2) всі пов’язані пакунки мають бути видалені.

в) при видалені пакунків:
   1) всі залежності та відносини пакунку мають бути видалені.

Додаткові обмеження:

- не можна створити залежність пакунку від самого себе;
- максимальний індивідуальний розмір об’єктів системи має бути обмежений для безпеки;
- назва пакунку повинна бути унікальною в межах репозиторію;
- назва бази пакунку повинна бути унікальною в межах репозиторію;
- юзернейм та пошта користувача повинні бути унікальними в межах репозиторію;
- обліковий запис не може мати дві однакові ролі для однієї бази пакунків;
- час збірки пакунка не може бути раніше ніж час його створення.

2 ПОСТАНОВКА ЗАДАЧІ

Потрібно розробити інформаційну систему «Репозиторій пакунків. Колаборація над пакунками», яка забезпечить ефективне зберігання, пошук, управління та спільну розробку програмних пакунків. Після аналізу предметної області та виявлених інформаційних потреб, було визначено наступні вимоги щодо функціоналу інформаційної системи.

Інформаційна система повинна зберігати інформацію про наступні об’єкти та дотримуватись вказаних правил:

а) облікові записи користувачів обов’язково мають:
   1) унікальний юзернейм не довше 31 символу;
   2) унікальну пошту не довше 255 символів;
   3) надійно збережений пароль за допомогою використання алгоритму argon2 [12];
   4) останній час використання облікового запису.

б) бази пакунків мають:
   1) обов’язкову унікальну назву не довше 127 символів;
   2) опціональний опис не довше 510 символів.

в) мусить існувати як мінімум 4 типи ролі для бази пакунків:
   1) submitter – власник бази пакунку котрий може редагувати базу пакунків;
   2) packager – пакувальник котрий збирає пакунки для баз пакунків;
   3) maintainer – супроводжуючий бази пакунку, той хто може редагувати пакунки цієї бази;
   4) flagger – мітчик, будь-який користувач який відмітив проблему у базі пакунку коментарем.

г) кожен обліковий запис може мати багато типів ролей як до однієї так і до багатьох баз пакунку. Для кожної прив’язки облікового запису до типу ролі і бази пакунку може бути коментар не довше 255 символів;

д) пакунок має:
   1) обов’язкову прив’язку до бази пакунку;
   2) обов’язкову унікальну назву не довше 127 символів;

3) обов’язкову версію не довше 127 символів;
4) опціональний опис не довше 255 символів;
5) опціональне веб-покликання на ресурс пакунку не довше 510 символів;
6) опціональний час помітки пакунку.

е) мусить існувати 3 типи відносин для пакунків:
1) provides – надає залежність;
2) conflicts – конфліктує з залежністю;
3) replaces – замінює залежність.

ж) кожен пакунок може мати багато відношень. Кожне відношення пакунку має:
1) опціональну архітектуру, не довше 63 символів;
2) опціональну вимогу, не довше 255 символів;
3) обов’язкову прив’язку до пакунку;
4) обов’язкову прив’язку до типу відношення;
5) обов’язкову назву пакунку-відношення, не довше 127 символів.

и) мусить існувати 4 типи залежності для пакунків:
1) depends – обов’язкова залежність;
2) makedepends – залежність під час встановлення пакунку;
3) checkdepends – залежність під час перевірки встановленого пакунку;
4) optdepends – опціональна залежність для розширення функціоналу пакунку.

к) кожен пакунок може мати багато залежностей. Кожна залежність пакунку має:
1) опціональну архітектуру, не довше 63 символів;
2) опціональну вимогу, не довше 255 символів;
3) опціональний опис, не довше 127 символів;
4) обов’язкову прив’язку до пакунку;
5) обов’язкову прив’язку до типу залежності;
6) обов’язкову назву пакунку-залежності, не довше 127 символів.

Інформаційна система повинна надавати наступні статистики:

а) статистика облікових записів:

1) загальна кількість облікових записів;
2) кількість активних облікових записів;
3) кількість неактивних облікових записів;
4) кількість користувачів котрі є творцями хоча б однієї бази пакунку;
5) кількість користувачів котрі супроводжують хоча б одну базу пакунку.

б) статистика репозиторію:
1) загальна кількість пакунків;
2) загальна кількість баз пакунків;
3) кількість пакунків у яких немає власника і які ніхто не супроводжує;
4) кількість пакунків доданих за останній тиждень;
5) кількість пакунків оновлених за останній тиждень;
6) кількість пакунків які ніколи не оновлювалися;

Інформаційна система повинна надавати наступні функції пошуку:

а) пошук пакунків, який фільтрується за допомогою текстової інформації з пошукового вводу, повинен мати наступні режими пошуку:
1) точний та загальний пошук за веб-покликанням;
2) точний та загальний пошук за назвою пакунку;
3) точний та загальний пошук за назвою бази пакунку;
4) загальний пошук за описом пакунку;
5) загальний пошук за описом бази пакунку;
6) загальний пошук за назвою та описом пакунку;
7) точний та загальний пошук за користувачем у якого є будь-яка роль до бази пакунку;
8) точний та загальний пошук за мітчиком бази пакунку;
9) точний та загальний пошук за пакувальником бази пакунку;
10) точний та загальний пошук за власником бази пакунку;
11) точний та загальний пошук за супроводжуючим бази пакунку.

б) можливість вибору ліміту кількості результатів пошуку для покращення швидкості процесів пошуку та сортування результатів;

в) можливість вибору вихідного або низхідного порядку сортування результатів;

г) сортування результатів за наступними параметрами:

1) назва пакунку;
2) версія пакунку;
3) назва бази пакунку;
4) час створення пакунку;
5) час останнього оновлення пакунку.

Інформаційна система повинна надавати можливість перегляду інформації, а також формування текстових звітів з наступних елементів:

а) інформація пакунку:

1) назва пакунку;
2) версія пакунку;
3) база пакунку;
4) опис пакунку;
5) веб-покликання на веб-ресурс пакунку;
6) час помітки пакунку;
7) час створення пакунку;
8) час оновлення пакунку;
9) перелік залежностей пакунку згрупованих за їх типами;
10) перелік відносин пакунку згрупованих за їх типами.

б) інформація бази пакунку:

1) назва бази пакунку;
2) опис бази пакунку;
3) час створення бази пакунку;
4) час оновлення бази пакунку;
5) перелік пакунків з цією базою;
6) перелік користувачів які володіють базою пакунку;
7) перелік користувачів які супроводжують базу пакунку;
8) перелік користувачів які є пакувальниками бази пакунку;
9) перелік користувачів які відмітили базу пакунку.

в) інформація облікових записів:

1) юзернейм користувача;
2) пошта користувача;

3) час останнього використання облікового запису;
4) час створення облікового запису;
5) час оновлення облікового запису;
6) перелік баз пакунків якими володіє обліковий запис;
7) перелік баз пакунків які супроводжує обліковий запис;
8) перелік баз пакунків для яких користувач пакує пакунки;
9) перелік баз пакунків які користувач відмітив.

г) інформація індивідуальної залежності пакунку:
1) архітектура для якої призначена залежність;
2) умова яка необхідна для залежності;
3) опис залежності;
4) пакунок;
5) тип залежності пакунку;
6) назва пакунку-залежності.

д) інформація індивідуального відношення пакунку:
1) архітектура для якої призначене відношення;
2) умова яка необхідна для відношення;
3) пакунок;
4) тип відношення пакунку;
5) назва пакунку-відношення.

Інформаційна система повинна забезпечити користувачів повним циклом керування своїми обліковими записами:

а) створення нового облікового запису за допомогою реєстрації з наступними даними:
1) унікальний юзернеймом;
2) унікальна пошта;
3) пароль довше 7 символів, який буде надійно збережений системою за допомогою алгоритму argon2 [12].

б) використання існуючого облікового запису за допомогою автентифікації з наступними даними:
1) юзернейм або електрона пошта, система повинна зрозуміти що користувач використовує;

2) пароль котрий при пропуску через argon2 буде однаковим з збереженим паролем облікового запису.

в) редагування власного облікового запису при якому оновиться час зміни облікового запису:
   1) редагування юзернейму облікового запису;
   2) редагування пошти облікового запису (для виконання цієї операції також потрібно ввести поточний пароль);
   3) редагування паролю облікового запису (для виконання цієї операції також потрібно ввести поточний пароль).

г) видалення облікового запису. При видаленні облікового запису всі ролі користувача мають бути видаленими.

Інформаційна система повинна забезпечити користувачів системою ролей для баз пакунків які забезпечують різні рівні доступу:

а) користувачі мають мати змогу видалити свої ролі для баз пакунків;

б) користувачі мають мати змогу отримати будь-який набір ролей для своєї бази пакунків;

в) роль автора бази пакунку «submitter» дозволяє користувачам виконувати будь-яку дію з їх базою пакунку та пакунками прив’язаними до цієї бази, а також давати будь-які ролі, окрім ролі мітчика «flagger», іншим користувачам;

г) роль супроводжуючого бази пакунку «maintainer» схожа на роль автора «submitter», але забороняє видалення бази пакунку та надання ролі автора іншим користувачам;

д) роль пакувальника бази пакунку «packager» не дає жодних прав, а лише позначає облікові записи відповідальні за збірку пакунків для бази даних пакунків;

е) роль мітчика бази пакунку «flagger» не дає жодних прав, а лише дає змогу користувачам сповістити авторів та супроводжуючих бази пакунку про проблеми у базі пакунку або пакунку з цією базою;

Інформаційна система повинна забезпечити автентифікованих користувачів повним циклом керування пакунками та їх базами:

а) створення нової бази пакунків може виконуватись як під час створення пакунку, так і як самостійний процес для якого треба вказати:
   1) унікальну назву бази пакунків;
   2) опціональний опис бази пакунків.

б) створення пакунку здійснюється за допомогою:
   1) унікальної назви пакунку;
   2) обрання існуючої, або створення нової бази пакунків за допомогою унікальної назви та опціонального опису;
   3) обов’язкової версії пакунку;
   4) опціонального опису пакунку;
   5) опціонального веб-покликання на веб-ресурс пакунку.

в) редагування інформації бази пакунку та пакунку;

г) отримання ролей до бази пакунку, або додання ролей іншим користувачам за правилами рівня доступу ролей вище;

д) створення, видалення та редагування залежностей для існуючого пакунку. Для створення залежності потрібно вказати:
   1) опціональну архітектуру залежності;
   2) опціональну умову залежності;
   3) опціональний опис залежності;
   4) вид залежності пакунків;
   5) назву пакунку-залежності.

е) створення, видалення та редагування відношень для існуючого пакунку. Для створення відношення потрібно вказати:
   1) опціональну архітектуру відношення;
   2) опціональну умову відношення;
   3) вид відношення пакунків;
   4) назву пакунку-відношення.

ж) видалення пакунку. При цьому кроці всі відносини та залежності пакунку мають бути видалені;

и) видалення бази пакунків. При цьому кроці всі пакунки з їх залежностями та відносинами, а також всі ролі користувачів для цієї бази пакунків мають бути видалені.

Інформаційна система повинна мати систему зворотного зв’язку з користувачем:

а) система отримання підтвердження користувача яка надасть опис критичності дії при видаленні інформації. Приклади дуже критичних випадків:
   1) при видаленні бази пакунків система має сповістити користувача що ця діє видалить всі пов’язані ролі та пакунки з їх залежностями і відносинами. Для підтвердження цієї дії користувач має ввести назву бази пакунків у форму підтвердження;
   2) при видаленні облікового запису користувач має бути сповіщений що його ролі будуть повністю видалені з системи. У випадку, якщо користувач має бази пакунків для яких він є єдиним автором, система має заборонити видалення облікового запису до моменту знаходження співавторів (має бути запропонована опція надання співавторства одному або кільком супроводжуючим) або видалення цих баз пакунків.

б) система сповіщення користувача про помилки в системі, наприклад:
   1) перебої з’єднання до сховища даних;
   2) нестачу ресурсів комп’ютеру;
   3) логічні помилки, котрі потрібно передати розробникам;
   4) знаходження аномалій в даних системи.

в) система сповіщення користувача про помилки в системі, наприклад:
   1) перебої з’єднання до сховища даних;
   2) знаходження аномалій в даних;
   3) конфлікти відносин залежностей (один пакунок конфліктує з тою залежністю, яку надає інший пакунок);

г) система візуального відображення статусу полів у формах даних та опису причини некоректності:
   1) виділення полів з правильними даними зеленим кольором;
   2) виділення полів з помилковими даними червоним кольором;
   3) виділення полів з потенційно помилковими даними жовтим кольором;

3 ПРОЕКТУВАННЯ БАЗИ ДАНИХ

3.1 Побудова ER-діаграми

Проведений аналіз предметної області дозволив визначити наступні сутності та їх атрибути:

- сутність «Користувач»: ім’я, електронна пошта, пароль, дата останнього використання, дата створення, дата оновлення;
- сутність «Пакунок»: назва, версія, опис, веб-покликання, дата позначення, дата створення, дата оновлення;
- сутність «База пакунку»: назва, опис, дата створення, дата оновлення;
- сутність «Тип ролі»: назва ролі, опис;
- сутність «Роль»: коментар;
- сутність «Тип залежності»: назва виду залежності;
- сутність «Залежність»: архітектура, умова, опис, назва залежного пакунку;
- сутність «Тип відношення»: назва виду відношення;
- сутність «Відношення»: архітектура, умова, назва пакунку з яким є відношення.

Між цими сутностями існують наступні зв’язки:

- «Користувач» - «Роль»: один до жодного або багатьох;
- «База пакунку» - «Роль»: один до жодного або багатьох;
- «Тип ролі» - «Роль»: один до жодного або багатьох;
- «База пакунку» - «Пакунок»: один до жодного або багатьох;
- «Пакунок» - «Залежність»: один до жодного або багатьох;
- «Тип залежності» - «Залежність»: один до жодного або багатьох;
- «Пакунок» - «Відношення»: один до жодного або багатьох;
- «Тип відношення» - «Відношення»: один до жодного або багатьох.

Зобразимо визначені сутності та відповідні зв’язки у вигляді ER-діаграми (див. рис. 3.1).

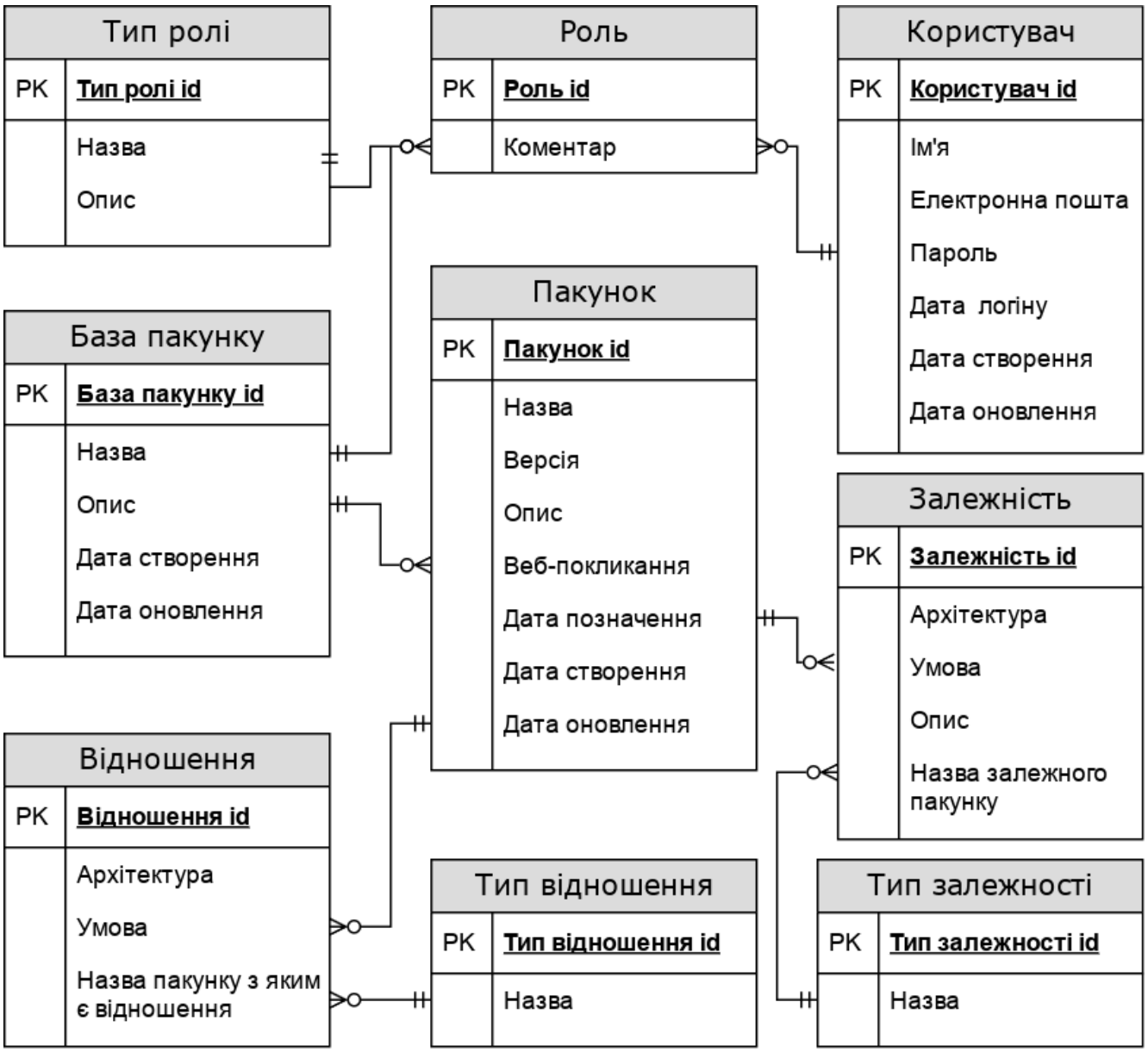

Рисунок 3.1 – ER-діаграма концептуальної моделі (рисунок виконано самостійно)

Отримана ER-діаграма буде використана для обрання та побудови логічної моделі бази даних.

3.2 Вибір та побудова логічної моделі бази даних на базі ER-діаграми

Для створення логічної моделі бази даних репозиторію пакунків було обрано реляційну модель. Ця модель дозволяє ефективно організувати дані у вигляді взаємопов’язаних таблиць, що забезпечує чітку структуру інформації та гарантує узгодженість всіх зв’язків та інформації між сутностями.

Проведемо аналіз первинних та зовнішніх ключів на основі створеної ER-діаграми (див. рис. 3.1), відокремив кожну сутність у відповідну таблицю:

а) таблиця Users (Користувачі):
   1) Первинний ключ: id.

б) таблиця PackageBases (Бази пакунків):
   1) Первинний ключ: id.

в) таблиця PackageBaseRoles (Типи ролі):
   1) Первинний ключ: id.

г) таблиця PackageBaseUserRoles (Ролі):
   1) Складовий первинний ключ: (base, user, role);
   2) Зовнішні ключі: base, user, role.

д) таблиця Packages (Пакунки):
   1) Первинний ключ: id;
   2) Зовнішній ключ: base

е) таблиця DependencyTypes (Типи залежностей):
   1) Первинний ключ: id.

ж) таблиця PackageDependencies (Залежності):
   1) Первинний ключ: id;
   2) Зовнішні ключі: package, dependency_type.

и) таблиця RelationTypes (Типи відношення):
   1) Первинний ключ: id.

к) таблиця PackageRelations (Відношення):
   1) Первинний ключ: id;
   2) Зовнішні ключі: package, relation_type.

Використання зовнішніх ключів забезпечить цілісність даних та дозволить легко виконувати операції JOIN для отримання необхідної інформації з кількох таблиць одночасно, що надасть можливість ефективного пошуку пакунків та їх аналізу.

Створимо довідник атрибутів (див. таблицю 3.1). Він встановлює однозначну відповідність між термінами, використаними в ER-діаграмі (див. рис. 3.1) та їх представленням у логічній моделі бази даних (див. рис. 3.2). В логічній моделі особливу увагу приділено відображенню зв’язків між сутностями, які реалізовані через механізм зовнішніх ключів, що забезпечить цілісність даних та правильну роботу інформаційної системи.

Таблиця 3.1 – довідник атрибутів (таблиця виконана самостійно)

ER-діаграма	Логічна модель
Архітектура	arch
База пакунку id	id
База пакунку	PackageBases
База пакунку	base
Веб-покликання	URL
Версія	version
Відношення id	id
Відношення	PackageRelations
Дата логіну	last_used
Дата оновлення	updated_at
Дата позначення	flagged_at
Дата створення	created_at
Електронна пошта	email
Залежність id	id
Залежність	PackageDependencies
Коментар	comment
Користувач id	id
Користувач	Users
Користувач	user
Назва залежного пакунку	dependency_package_name
Назва пакунку з яким є відношення	relation_package_name
Назва	name
Опис	description
Пакунок id	id
Пакунок	Packages
Пароль	password
Роль id	id
Роль	PackageBaseUserRoles
Тип відношення id	id
Тип відношення	RelationTypes
Тип залежності id	id
Тип залежності	DependencyTypes
Тип ролі id	id
Тип ролі	PackageBaseRoles
Тип ролі	role
Умова	requirement

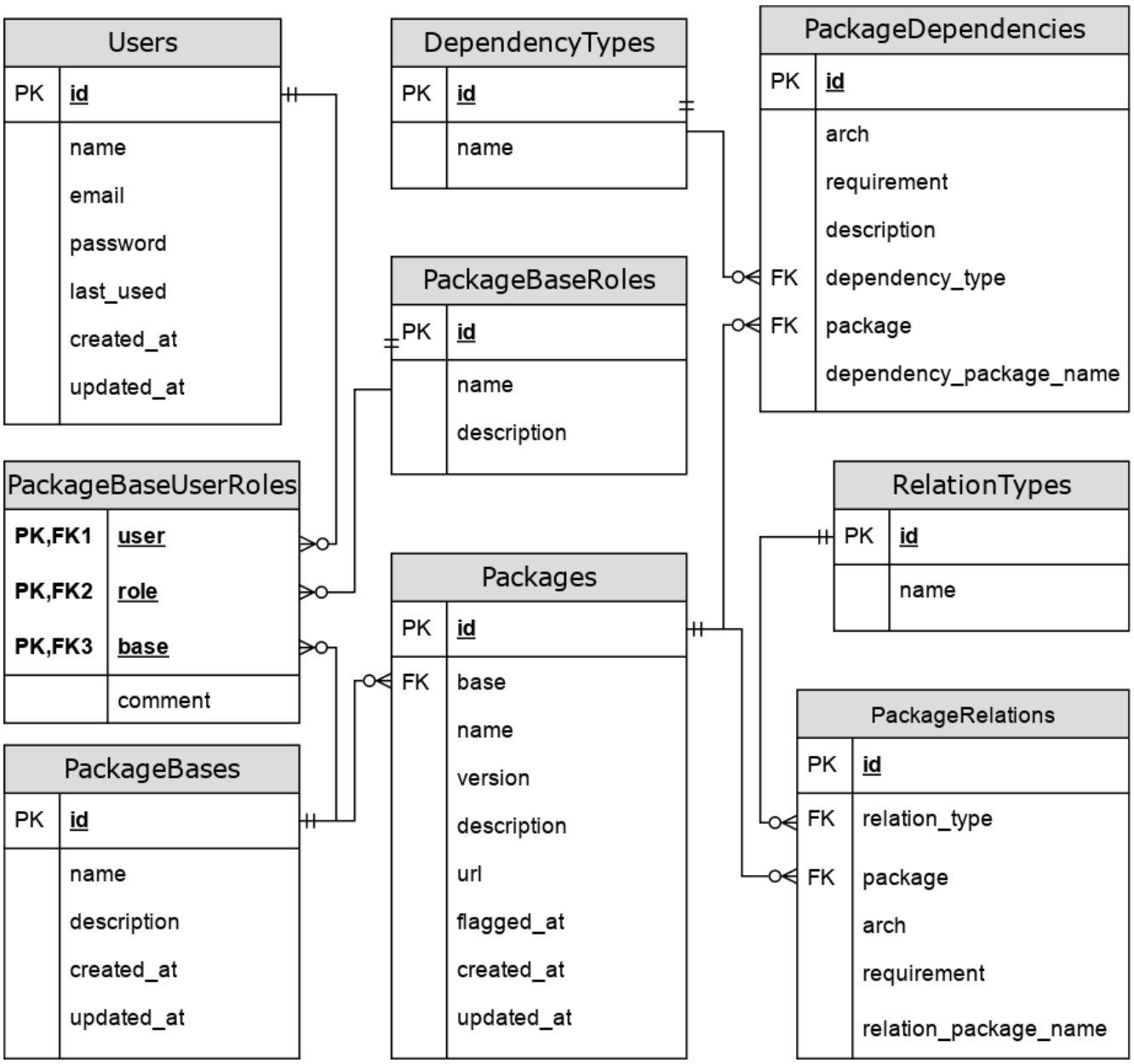

Рисунок 3.2 – Логічна модель бази даних (рисунок виконано самостійно)

Проведемо перевірку схеми логічної моделі бази даних на відповідність вимогам третьої нормальної форми. Для кожної таблиці треба визначити функціональні залежності та перевірити їх на відповідність першій, другій та третій нормальним формам [13].

а) Таблиця Users:

1) від первинного ключа id залежать всі неключові атрибути, а саме: name, email, password, last_used, created_at, updated_at;

2) таблиця відповідає всім вимогам 3НФ: всі атрибути атомарні (1НФ), всі неключові атрибути залежать безпосередньо від первинного ключа id (2НФ), а також відсутні транзитивні залежності.

б) Таблиця Packages:

1) від первинного ключа id залежать всі неключові атрибути, а саме: base, name, version, description, URL, flagged_at, created_at, updated_at;

2) таблиця відповідає всім вимогам 3НФ: всі атрибути атомарні (1НФ), всі неключові атрибути залежать безпосередньо від первинного ключа id (2НФ), а також відсутні транзитивні залежності.

в) Таблиця PackageBases:

1) від первинного ключа id залежать всі неключові атрибути, а саме: name, description, created_at, updated_at;

2) таблиця відповідає всім вимогам 3НФ: всі атрибути атомарні (1НФ), всі неключові атрибути залежать безпосередньо від первинного ключа id (2НФ), а також відсутні транзитивні залежності.

г) Таблиця PackageBaseRoles:

1) від первинного ключа id залежать всі неключові атрибути, а саме: name, description;

2) таблиця відповідає всім вимогам 3НФ: всі атрибути атомарні (1НФ), всі неключові атрибути залежать безпосередньо від первинного ключа id (2НФ), а також відсутні транзитивні залежності.

д) Таблиця PackageBaseUserRoles:

1) від складеного первинного ключа (base, user, role) залежать всі неключові атрибути, а саме: comment;

2) таблиця відповідає всім вимогам 3НФ: всі атрибути атомарні (1НФ), всі неключові атрибути залежать безпосередньо від складеного первинного ключа (base, user, role) (2НФ), а також відсутні транзитивні залежності.

е) Таблиця DependencyTypes:

1) від первинного ключа id залежать всі неключові атрибути, а саме: name;

2) таблиця відповідає всім вимогам 3НФ: всі атрибути атомарні (1НФ), всі неключові атрибути залежать безпосередньо від первинного ключа id (2НФ), а також відсутні транзитивні залежності.

ж) Таблиця PackageDependencies:

1) від первинного ключа id залежать всі неключові атрибути, а саме: arch, requirement, description, package, dependency_type, dependency_package_name;
2) таблиця відповідає всім вимогам 3НФ: всі атрибути атомарні (1НФ), всі неключові атрибути залежать безпосередньо від первинного ключа id (2НФ), а також відсутні транзитивні залежності.

и) Таблиця RelationTypes:

1) від первинного ключа id залежать всі неключові атрибути, а саме: name;
2) таблиця відповідає всім вимогам 3НФ: всі атрибути атомарні (1НФ), всі неключові атрибути залежать безпосередньо від первинного ключа id (2НФ), а також відсутні транзитивні залежності.

к) Таблиця PackageRelations:

1) від первинного ключа id залежать всі неключові атрибути, а саме: arch, requirement, package, relation_type, relation_package_name;
2) таблиця відповідає всім вимогам 3НФ: всі атрибути атомарні (1НФ), всі неключові атрибути залежать безпосередньо від первинного ключа id (2НФ), а також відсутні транзитивні залежності.

3.3 Побудова логічної моделі бази даних шляхом нормалізації

Нормалізація структури даних відіграє фундаментальну роль у процесі проектування сучасних інформаційних систем, оскільки забезпечує створення раціональної архітектури даних, що ефективно запобігає надлишковості інформації та можливим аномаліям. В контексті розробки інформаційної системи пакункового репозиторію, нормалізація набуває особливої значущості через комплексну природу зв’язків між пакунками, їх відносинами, залежностями та користувачами системи.

Розмір ER-діаграми інформаційної системи містить понад 25 атрибутів, тому згідно з методичними вказівками з курсового проектування було обрано

частину, що охоплює 5 взаємопов’язаних сутностей: «Користувач», «Пакунок», «База пакунку», «Тип ролі» та «Роль» (див. рис. 3.3). Така декомпозиція дозволяє детально опрацювати найбільш критичні аспекти системи, забезпечуючи при цьому можливість подальшого масштабування.

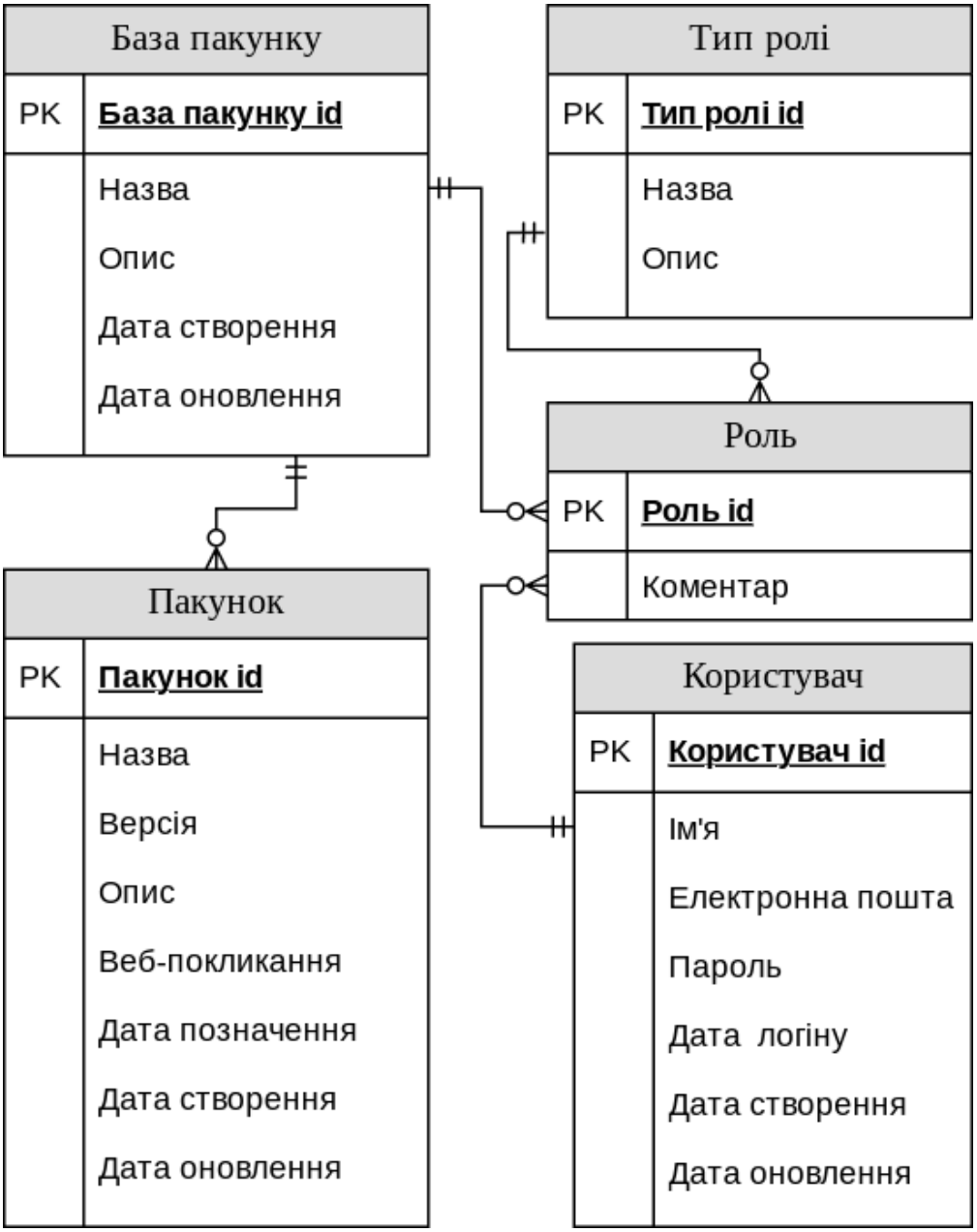

Рисунок 3.3 – Фрагмент ER-діаграми (рисунок виконано самостійно)

За визначенням, відношення знаходиться в першій нормальній формі (1НФ) [13], якщо всі його атрибути є атомарними, тобто неподільними. Для практичної перевірки відповідності 1НФ, створимо універсальне відношення T (див. рис. 3.4), що інтегрує атрибути всіх релевантних сутностей. Це відношення формує фундаментальну структуру для подальшої нормалізації даних.

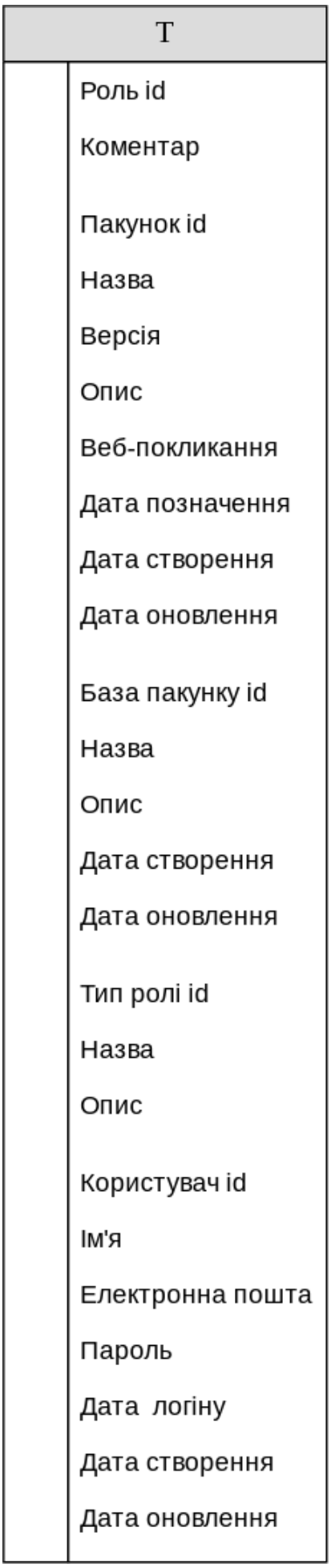

Рисунок 3.4 – Універсальне відношення T (рисунок виконано самостійно)

При визначенні первинного ключа особлива увага приділяється семантичному аналізу даних. Зокрема можна помітити потребу бачити яку роль має користувач для кожного пакунку. Тому найкращим ключовими атрибутами стануть «Пакунок id» та «Роль id». Вони надають можливість унікально ідентифікувати універсальне відношення та їх сполучення охоплює всі атрибути відношення.

Всі визначені атрибути є неподільними, а значення атомарні. Сполучений первинний ключ, який складається з «Пакунок id» та «Роль id», дозволяє унікально ідентифікувати кожний кортеж. Ключові поля не мають порожніх значень, кортежі не мають фіксованого порядку, тому, універсальне відношення T знаходиться в першій нормальній формі (1НФ).

Для перевірки універсального відношення T на відповідність другій нормальній формі (2НФ) [13], проаналізуємо його на існування часткових функціональних залежностей неключових атрибутів від частини первинного ключа (див. рис. 3.5).

Перелік атрибутів які залежать лише від частини ключа:

- «Роль id»: Пакунок id, Назва, Версія, Опис, Веб-покликання, Дата позначення, Дата створення, Дата оновлення, База пакунку id, Назва, Опис, Дата створення, Дата оновлення;
- «Пакунок id»: Роль id, Коментар, Тип ролі id, Назва, Опис, Користувач id, Ім’я, Електронна пошта, Пароль Дата логіну, Дата створення, Дата оновлення.

Можемо зробити висновок, що відношення не знаходиться в другій нормальній формі, адже деякі атрибути мають неповні функціональні залежності (залежать лише від частини ключа). Для приведення універсального відношення T до другої нормальної форми виділимо з нього два універсальних відношення T1 та T2 (див. рис. 3.6), де T1 буде містити атрибути які повністю залежать від частини ключа «Роль id», а T2 буде містити атрибути які повністю залежать від частини ключа «Пакунок id».

T1 та T2 зберігають першу нормальну форму (1НФ) та не містять неповних функціональних залежностей, оскільки кожне з них має лише один ключовий атрибут. Таким чином, можна зробити висновок, що ці відношення відповідають вимогам другої нормальної форми (2НФ).

Рисунок 3.5 – Універсальне відношення T із визначеними залежностями (рисунок виконано самостійно)

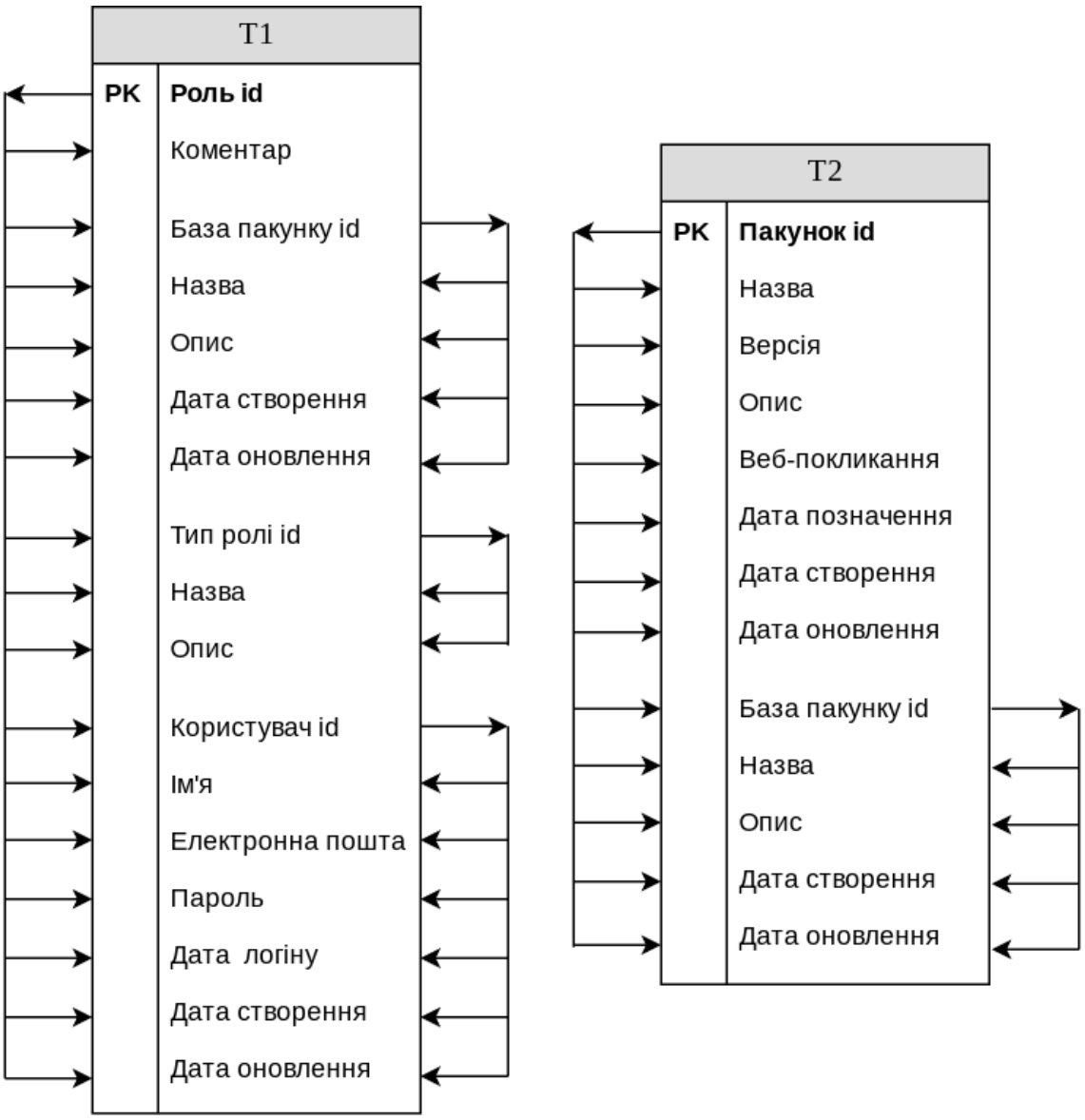

Рисунок 3.6 – Універсальні відношення T1 та T2 (рисунок виконано самостійно)

Після декомпозиції до 2НФ для досягнення третьої нормальної форми (3НФ) [13] в T1 та T2 кожен неключовий атрибут повинен залежати безпосередньо від первинного ключа. Наявні транзитивні залежності від неключових атрибутів:

- «Тип ролі id»: Назва, Опис;
- «База пакунку id»: Назва, Опис, Дата створення, Дата оновлення;
- «Користувач id»: Ім’я, Електронна пошта, Пароль, Дата логіну, Дата створення, Дата оновлення.

Винесемо з відношення T1 два відношення «Тип ролі» та «Користувач» за допомогою ключів «Тип ролі id» та «Користувач id» відповідно. Після чого сформуємо відношення T3 в котрому залишимо тільки зовнішні ключі «Тип ролі id» та «Користувач id» (див. рис. 3.7).

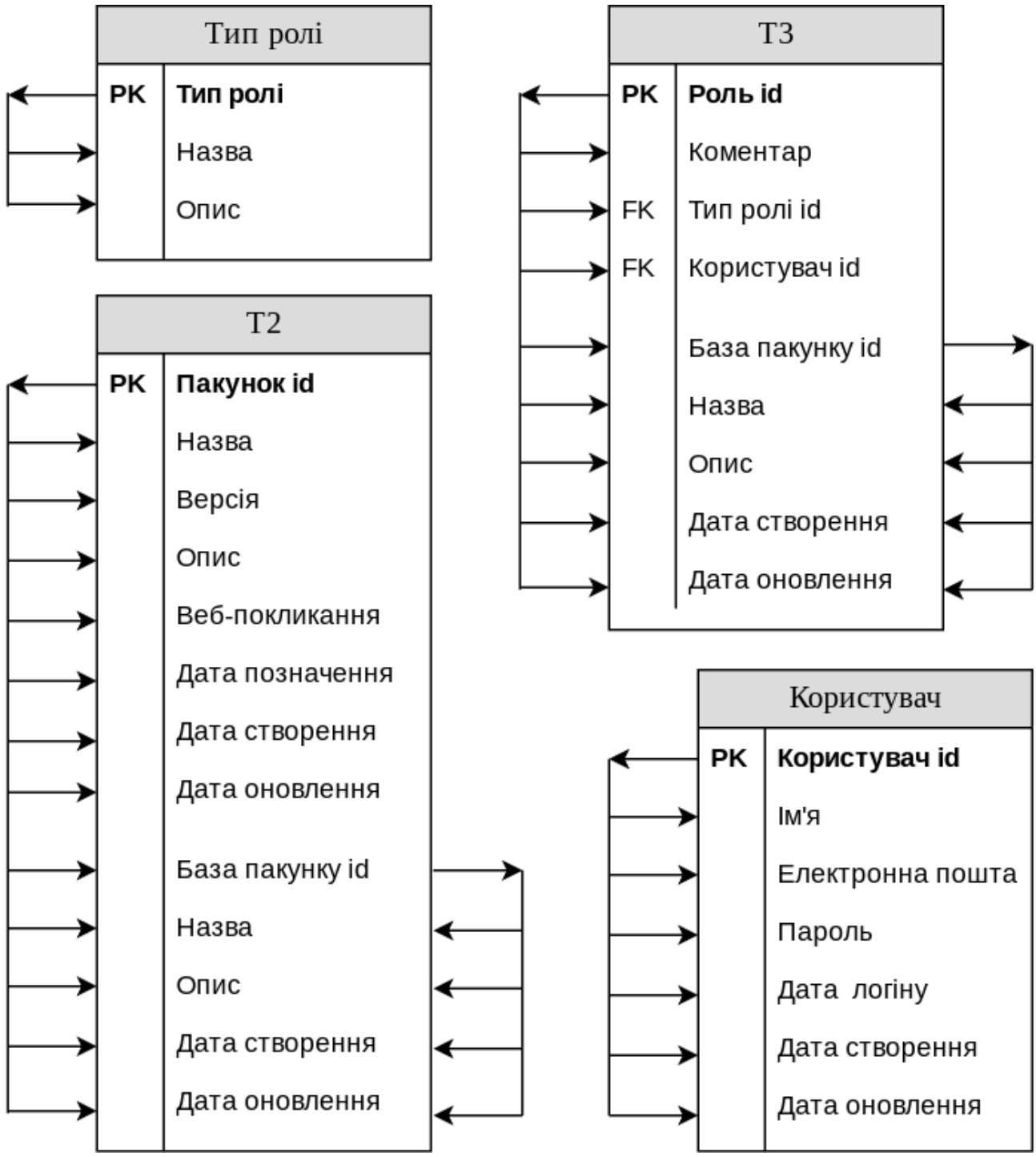

Рисунок 3.7 – Відношення «Тип ролі», «Користувач», Т2 та Т3 (рисунок виконано самостійно)

Можна помітити, що відношення Т2 та Т3 мають спільне відношення «База пакунку id». Утворимо відношення «Пакунок» та «Роль» виділив з відношень Т2 та Т3 відповідно відношення «База пакунку», залишимо на його місці ключ «База пакунку id».

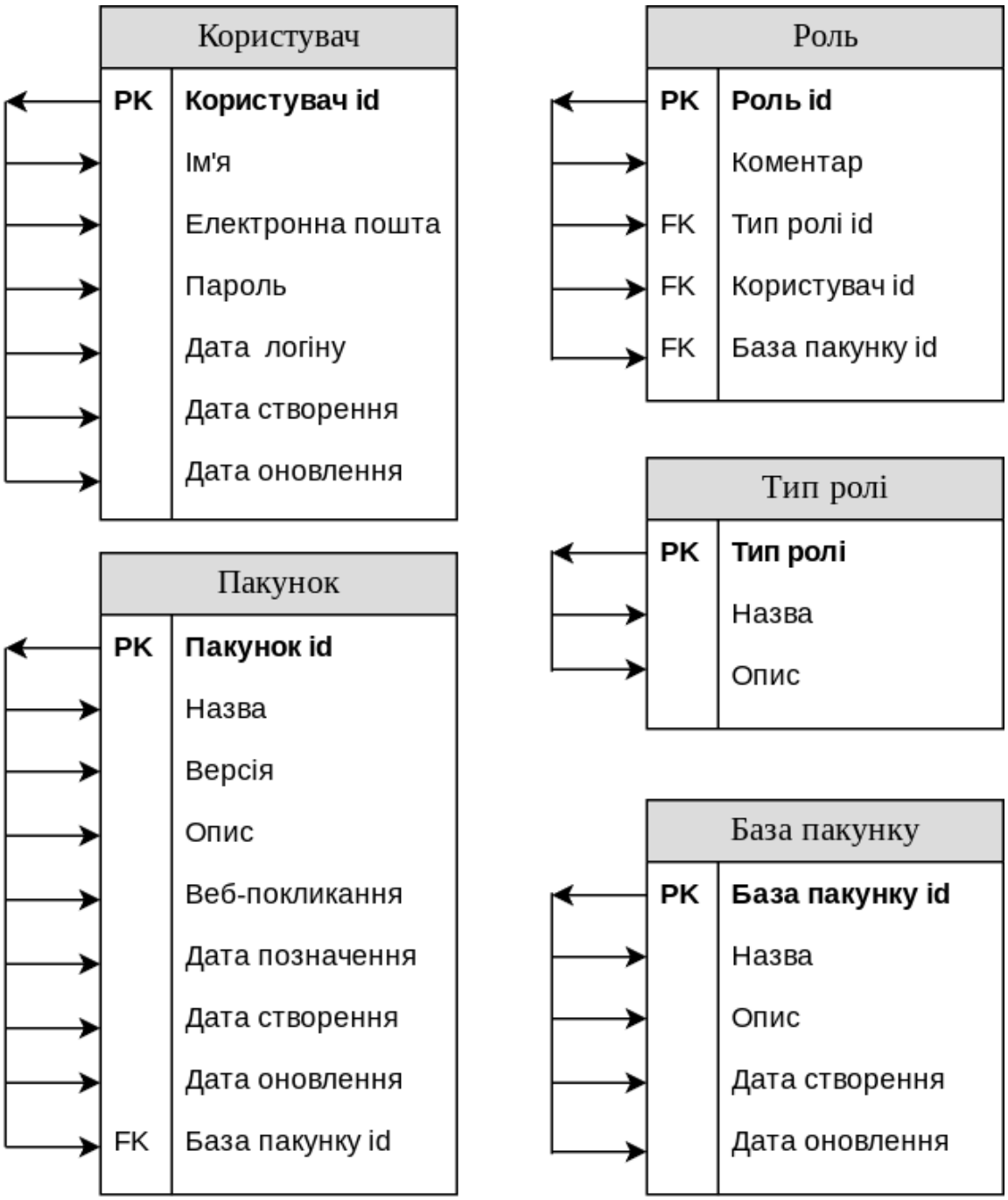

Рисунок 3.8 – Відношення «Користувач», «Пакунок», «База пакунку», «Тип ролі» та «Роль» (рисунок виконано самостійно)

Тепер всі відношення знаходяться в третій нормальній формі (3НФ): вони мають первинні ключі та всі атрибути є атомарними (1НФ), всі атрибути кожного відношення повністю функціонально залежать від первинного ключа (2НФ), жодне відношення не містить транзитивних залежностей (3НФ).

Побудуємо схему даних та позначимо зв’язки між сутностями (див. рис. 3.9). Після цього порівняємо отриману схему даних з початковою ER-діаграмою для подальшого аналізу.

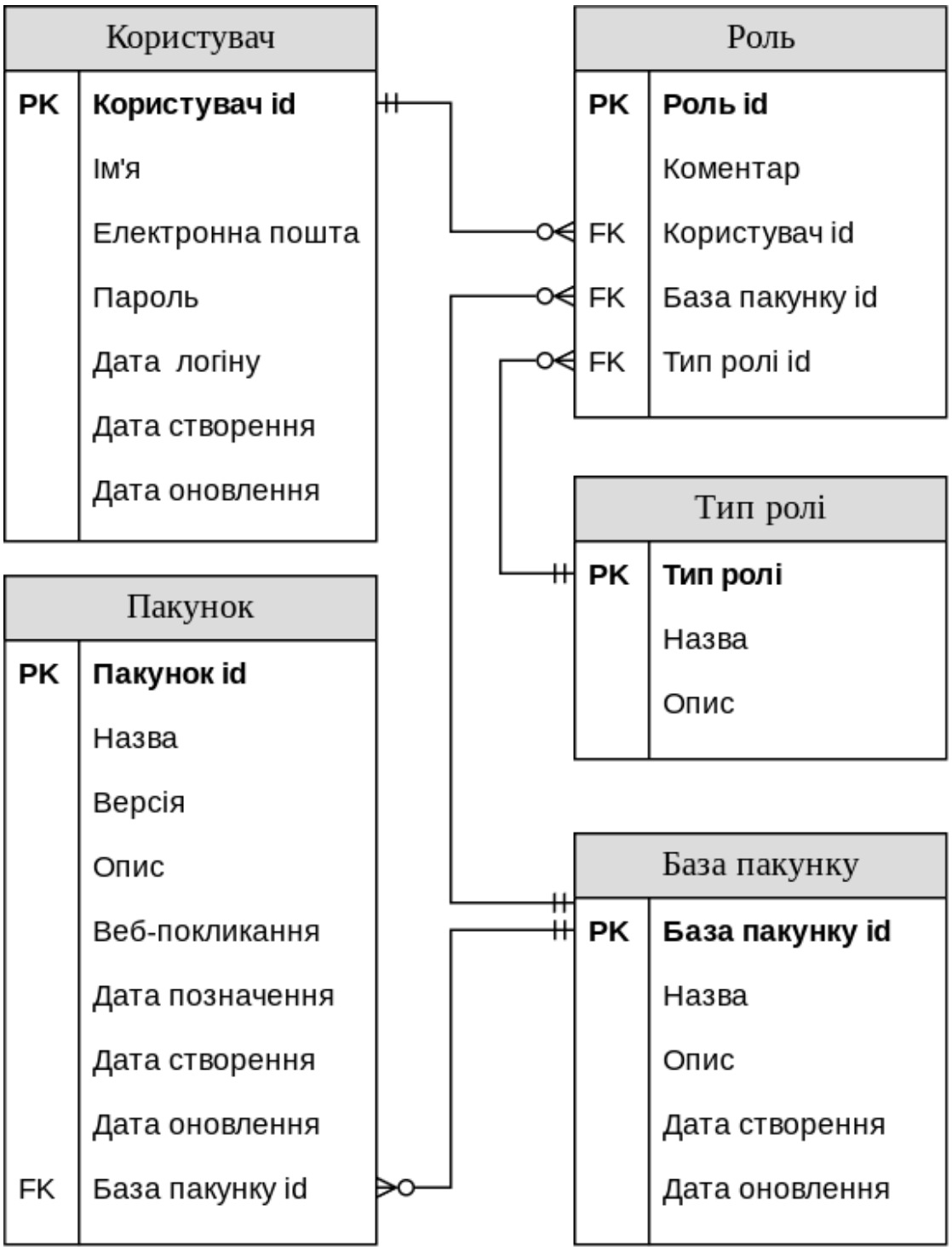

Рисунок 3.9 – Побудована схема даних (рисунок виконано самостійно)

Порівняння отриманої схеми даних з початковою ER-діаграмою (див. рис. 3.1) показує їхню повну відповідність, що свідчить про успішну нормалізацію схеми даних до третьої нормальної форми (3НФ). Це значить, що всі сутності, їх атрибути та зв’язки були правильно визначені, а всі непотрібні залежності та повторення даних були усунуті. Це забезпечить ефективність та масштабованість інформаційної системи.

4 ОПИС ПРОГРАМИ

4.1 Загальні відомості

Для забезпечення простоти, ефективності та елегантності розробки інформаційної системи «Репозиторій пакунків. Колаборація над пакунками» було використано операційну систему Arch Linux [2] та текстовий редактор Neovim [7].

Для реалізації всієї комп'ютерної програми було обрано сучасну мову програмування Rust [3], яка є надзвичайно швидкою, надійною та продуктивною. За зберігання даних відповідає база даних MySQL [6], вона відома своєю стабільністю, можливостями та швидкістю. Для взаємодії з базою даних було обрано бібліотеку SQLx [5], вона є дуже гарно спроектованим проектом, розрахована на асинхронні операції та підтримку багатьох баз даних на низькому рівні. Для написання інтерфейсу комп'ютерної програми було обрано бібліотеку iced [4], ця бібліотека фокусується на простоті та безпеці програм з графічним інтерфейсом за допомогою дотримання принципів проектування Elm [14].

4.2 Виклик і завантаження

Інформаційна система була розроблена з використанням мови Rust [3], тому результуюча комп'ютерна програма є невимогливою до потужностей комп'ютеру, не потребує сторонніх залежностей і може бути використана у вигляді самодостатнього файлу виконання.

Для розгортання та запуску системи необхідно використовувати Linux сумісну операційну систему, наприклад Arch Linux [2], та виконати наступні кроки:

а) збірка комп'ютерної програми:
   1) встановити Docker [15] та Docker Compose [16];
   2) встановити інструменти для компіляції вихідного коду Rust. Інструкції можна знайти на офіційному вебсайті Rust [3];
   3) завантажити вихідний код комп'ютерної програми з публічного репозиторію Gitea [17];
   4) перейти у теку завантаженого вихідного коду та скомпілювати комп'ютерну програму за допомогою команди «cargo c –release».

б) запуск бази даних:

1) в завантаженій теці вихідного коду потрібно перейти у теку «assets»;
2) у останній строчці файлу з назвою «compose.yaml» потрібно змінити «password» на надійний пароль;
3) в теці «assets» потрібно виконати команду «docker compose up -d».

в) запуск програми:
1) виконати команду «export DATABASE_URL="mysql://root:password@localhost:3306/repository"». Слід зауважити, що «password» слід замінити на виставлений пароль у файлі «compose.yaml»;
2) будь-де в теці вихідного коду комп'ютерної програми запустити команду «cargo r –release».

Слід зауважити, що змінна середовища «DATABASE_URL» є обов'язковою для виконання програми. В цій змінній лежить адреса до налаштованого екземпляру СУБД MySQL. Через складність самостійного налаштування СУБД MySQL, для максимальної зручності рекомендується використовувати контейнеризацію за допомогою Docker [15], особливо рекомендується його інструмент Compose [16], який спростить задачу до однієї команди «docker compose up -d».

4.3 Призначення і логічна структура

Інформаційна система полегшує кооперацію над пакунками у репозиторії для розробників. Система забезпечує інтуїтивний інтерфейс для навігації по пакункам, їх залежностям з відносинами, та користувачам. Перелік основних функцій системи:

- управління пакунками та їх інформацією;
- система користувацьких облікових записів;
- система різних рівнів доступу облікових записів до пакунків;
- пошук та катетеризація пакунків за багатьма факторами;
- надання аналітичної інформації про репозиторій, пакунки та користувачів.

Під час розробки було використано гексагональну архітектуру [18] проектування, також відому як «архітектура портів та адаптерів», завдяки чому частини проекту є чітко розділеними (див. рис. 4.1).

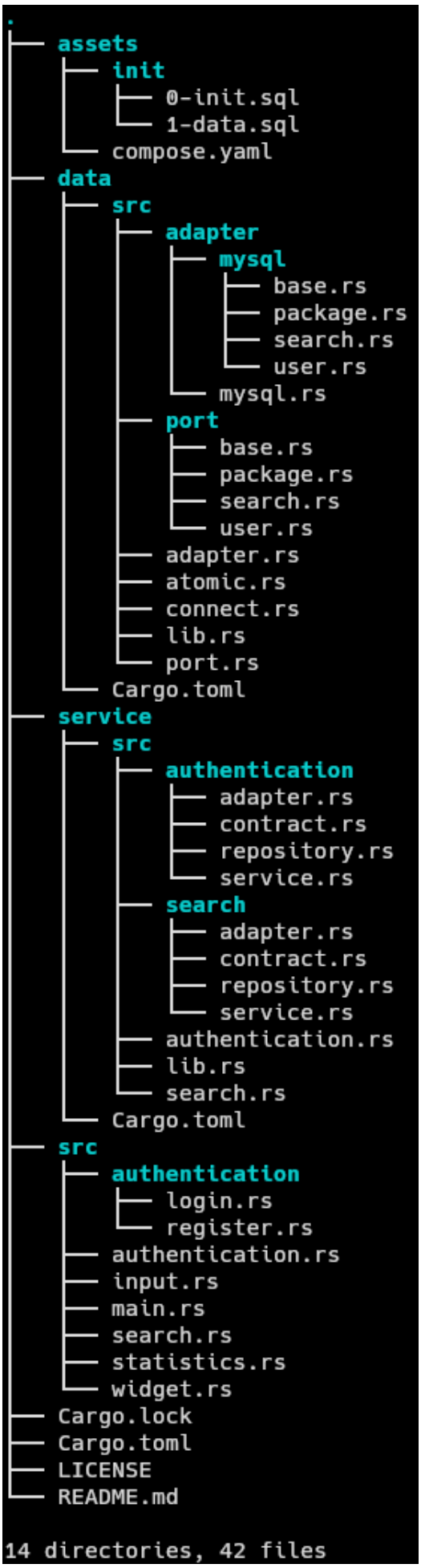

Рисунок 4.1 – Структура проекту (рисунок виконано самостійно)

Структура проекту складається з кількох рівнів:

а) Шар взаємодії з базою даних (тека data):
   1) декларація інтерфейсу взаємодії (тека ports);
   2) імплементації інтерфейсів взаємодії (тека adapters).

б) Шар сервісів для бізнес-логіки програми (тека service); сервіси мають:
   1) файли з назвою repository – декларація інтерфейсу репозиторію який використовує дані з частини декларації інтерфейсу шару бази даних;
   2) файли з назвою adapter – імплементація інтерфейсу репозиторію який оперує отриманням даних з частини імплементацій інтерфейсів взаємодії шару бази даних;
   3) файли з назвою contract – декларація контракту (інтерфейсу) який будується на інтерфейсі репозиторію і описує дані з котрими буде працювати сервіс;
   4) файли з назвою service – імплементація сервісу котрий оперує над даними з контракту, здійснює логічні операції та обчислення над цими даними.

в) Шар графічного інтерфейсу (тека src) використовує контракти з шару сервісів для перевірки даних від користувача та надсилання запитів до логічної частини комп’ютерної програми.

г) Головний файл проекту (тека src, файл main.rs) відповідає за компонування всіх шарів:
   1) встановлення підключення до бази даних;
   2) ініціалізацію адаптерів бази даних;
   3) ініціалізацію адаптерів репозиторіїв за допомогою створених підключень та адаптерів бази даних;
   4) запуск сервісів за допомогою створених репозиторіїв;
   5) ініціалізація графічних елементів передав їм створені сервіси;
   6) відображення та менеджмент графічних частин комп’ютерної програми.

Вихідний код проекту [17] має в собі всі використані під час розробки ресурси, такі як SQL скрипти та Docker Compose [16] файли, за допомогою яких можна створити тестову базу даних наповнену даними.

4.4 Опис фізичної моделі бази даних

Для інформаційної системи створено базу даних яка має дев’ять таблиць. Код для її створення описаний нижче.

Таблиця «Users» містить інформацію про користувачів системи та має таку структуру:

- id - службове додатне число, необхідне для ідентифікації таблиці та забезпечення надійної роботи бази даних;
- ім’я - унікальне текстове поле яке зберігає ім’я користувача (довжина до 31 символу), не може бути порожнім;
- пошта - унікальне текстове поле яке зберігає електронну пошту користувача (довжина до 255 символів), не може бути порожнім;
- пароль - текстове поле яке зберігає хеш пароля (довжина до 255 символів), не може бути порожнім;
- останній логін - зберігає дату останнього використання облікового запису, може бути порожнім;
- час створення - зберігає дату створення облікового запису не може бути порожнім;
- час оновлення - час оновлення даних в таблиці, автоматично оновлюється при зміні запису, не може бути порожнім.

```
-- Required info for an account
CREATE TABLE Users (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name        VARCHAR(31)  UNIQUE NOT NULL,
    email       VARCHAR(255) UNIQUE NOT NULL,
    password    VARCHAR(255) NOT NULL,

    last_used   TIMESTAMP NULL,

    created_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    updated_at  TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL
);
```

Таблиця «PackageBases» містить інформацію про бази пакунків:

- id - службове додатне число, необхідне для ідентифікації таблиці та забезпечення надійної роботи бази даних;

- назва - унікальне текстове поле, зберігає назву базового пакунку (до 127 символів), не може бути порожнім;
- опис - текстове поле (до 510 символів), може бути порожнім, зберігає опис пакунку;
- час створення - зберігає дату створення бази пакунку, не може бути порожнім;
- час оновлення - час оновлення даних в таблиці, автоматично оновлюється при зміні запису, не може бути порожнім.

```
-- Enables multiple packages to have the same base yet different
components
CREATE TABLE PackageBases (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name        VARCHAR(127) UNIQUE NOT NULL,
    description VARCHAR(510) NULL,

    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

Таблиця «PackageBaseRoles» визначає ролі користувачів для роботи з пакунками:

- id - службове додатне число, необхідне для ідентифікації таблиці та забезпечення надійної роботи бази даних;
- назва - унікальне текстове поле, зберігає назву ролі (наприклад: submitter, packager; довжина до 31 символу), не може бути порожнім;
- опис - текстове поле яке описує роль (до 255 символів), може бути порожнім.

```
-- User roles for working on packages: flagger, packager, submitter,
maintainer, etc.
CREATE TABLE PackageBaseRoles (
    id TINYINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name        VARCHAR(31) UNIQUE NOT NULL,
    description VARCHAR(255) NULL
);
```

Таблиця «PackageBaseUserRoles» описує ролі користувачів для баз пакунків:

- база пакунку - числове поле (ціле додатне число), зовнішній ключ на таблицю PackageBases;

- користувач - числове поле (ціле додатне число), зовнішній ключ на таблицю Users;
- роль - числове поле (ціле додатне число), зовнішній ключ на таблицю PackageBaseRoles;
- коментар - текстове поле для збереження приміток (до 255 символів), може бути порожнім;
- (base, user, role) - складний (композитний) первинний ключ, необхідний для ідентифікації таблиці та забезпечення надійної роботи бази даних;

```
-- Roles that a user has for a package
CREATE TABLE PackageBaseUserRoles (
    base INT UNSIGNED,
    user INT UNSIGNED,
    role TINYINT UNSIGNED,

    comment VARCHAR(255) NULL,

    PRIMARY KEY (base, user, role), -- composite key
    FOREIGN KEY (base) REFERENCES PackageBases(id) ON DELETE CASCADE,
    FOREIGN KEY (user) REFERENCES Users(id) ON DELETE CASCADE,
    FOREIGN KEY (role) REFERENCES PackageBaseRoles(id) ON DELETE
CASCADE
);
```

Таблиця «Packages» містить інформацію про окремі пакунки:

- id - службове додатне число, необхідне для ідентифікації таблиці та забезпечення надійної роботи бази даних;
- база - зовнішній ключ (ціле додатне число) на таблицю PackageBases;
- назва - унікальне текстове поле для збереження назви пакунку (до 127 символів), не може бути порожнім;
- версія - текстове поле для збереження версії пакунку (до 127 символів), не може бути порожнім;
- опис - текстове поле для збереження опису пакунку (до 255 символів), може бути порожнім;
- веб-покликання - текстове поле, зберігає посилання на ресурс пакунку (до 510 символів);
- час позначення - час, коли пакунок був позначений, може бути порожнім;
- час створення - зберігає дату створення пакунку, не може бути порожнім;

– час оновлення - час оновлення даних в таблиці, автоматично оновлюється при зміні запису, не може бути порожнім.

```
-- Information about the actual packages
CREATE TABLE Packages (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    base        INT UNSIGNED NOT NULL,
    name        VARCHAR(127) UNIQUE NOT NULL,
    version     VARCHAR(127) NOT NULL,
    description VARCHAR(255) NULL,
    url         VARCHAR(510) NULL,

    flagged_at TIMESTAMP NULL,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    FOREIGN KEY (base) REFERENCES PackageBases (id) ON DELETE CASCADE
);
```

Таблиця «DependencyTypes» визначає типи залежностей:

– id - службове додатне число, необхідне для ідентифікації таблиці та забезпечення надійної роботи бази даних;
– назва - унікальне текстове поле, зберігає назву типу залежності (наприклад: depends, makedepends; довжина до 31 символу), не може бути порожнім.

```
-- depends, makedepends, optdepends, etc.
CREATE TABLE DependencyTypes (
    id TINYINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(31) UNIQUE NOT NULL
);
```

Таблиця «PackageDependencies» відображає залежності пакунків:

– id - службове додатне число, необхідне для ідентифікації таблиці та забезпечення надійної роботи бази даних;
– архітектура - текстове поле, зберігає цільову архітектуру залежності (до 63 символів), може бути порожнім;
– умова - текстове поле, яке зберігає умову залежності (до 255 символів), може бути порожнім;
– опис - текстове поле, зберігає опис залежності (до 127 символів), може бути порожнім;

- пакунок - зовнішній ключ на таблицю Packages;
- тип залежності - зовнішній ключ на таблицю DependencyTypes;
- назва залежного пакунку - текстове поле яке зберігає назва залежного пакунку (до 127 символів), не може бути порожнім.

```
-- Track which dependencies a package has
CREATE TABLE PackageDependencies (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    arch        VARCHAR(63)  NULL,
    requirement VARCHAR(255) NULL,
    description VARCHAR(127) NULL,

    package INT UNSIGNED NOT NULL,
    dependency_type TINYINT UNSIGNED NOT NULL,
    dependency_package_name VARCHAR(127) NOT NULL, -- Not an actual
package, but an an alias. Allows for package substitution.

    FOREIGN KEY (package) REFERENCES Packages (id) ON DELETE CASCADE,
    FOREIGN KEY (dependency_type) REFERENCES DependencyTypes (id)
);
```

Таблиця «RelationTypes» визначає типи зв’язків між пакунками:

- id - службове додатне число, необхідне для ідентифікації таблиці та забезпечення надійної роботи бази даних;
- назва - унікальне текстове поле яке зберігає назву зв’язку (наприклад: conflicts, provides; довжина до 31 символу), не може бути порожнім.

```
-- conflicts, provides, replaces, etc.
CREATE TABLE RelationTypes (
    id TINYINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(31) UNIQUE NOT NULL
);
```

Таблиця «PackageRelations» описує зв’язки між пакунками:

- id - службове додатне число, необхідне для ідентифікації таблиці та забезпечення надійної роботи бази даних;
- архітектура - текстове поле, зберігає цільову архітектуру зв’язку (до 63 символів), може бути порожнім;
- умова - текстове поле, яке зберігає умову зв’язку (до 255 символів), може бути порожнім;
- пакунок - зовнішній ключ на таблицю Packages;
- тип зв’язку - зовнішній ключ на таблицю RelationTypes;

– тип зв’язку з пакунком - текстове поле, зберігає назву пакунку, з яким є зв’язок (до 127 символів), не може бути порожнім.

```
-- Track which conflicts, provides and replaces a package has
CREATE TABLE PackageRelations (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    arch        VARCHAR(63) NULL,
    requirement VARCHAR(255) NULL,

    package INT UNSIGNED NOT NULL,
    relation_type TINYINT UNSIGNED NOT NULL,
    relation_package_name VARCHAR(127) NOT NULL,

    FOREIGN KEY (package) REFERENCES Packages (id) ON DELETE CASCADE,
    FOREIGN KEY (relation_type) REFERENCES RelationTypes (id)
);
```

4.5 Опис програмної реалізації

При запуску комп’ютерної програми стартовим екраном є сторінка логіну (див. рис. 4.2). Користувачі з існуючими обліковими записами можуть увійти у свій обліковий запис за допомогою пошти або юзернейма та свого паролю який надійно та безпечно зберігається у зашифрованому вигляді в базі даних.

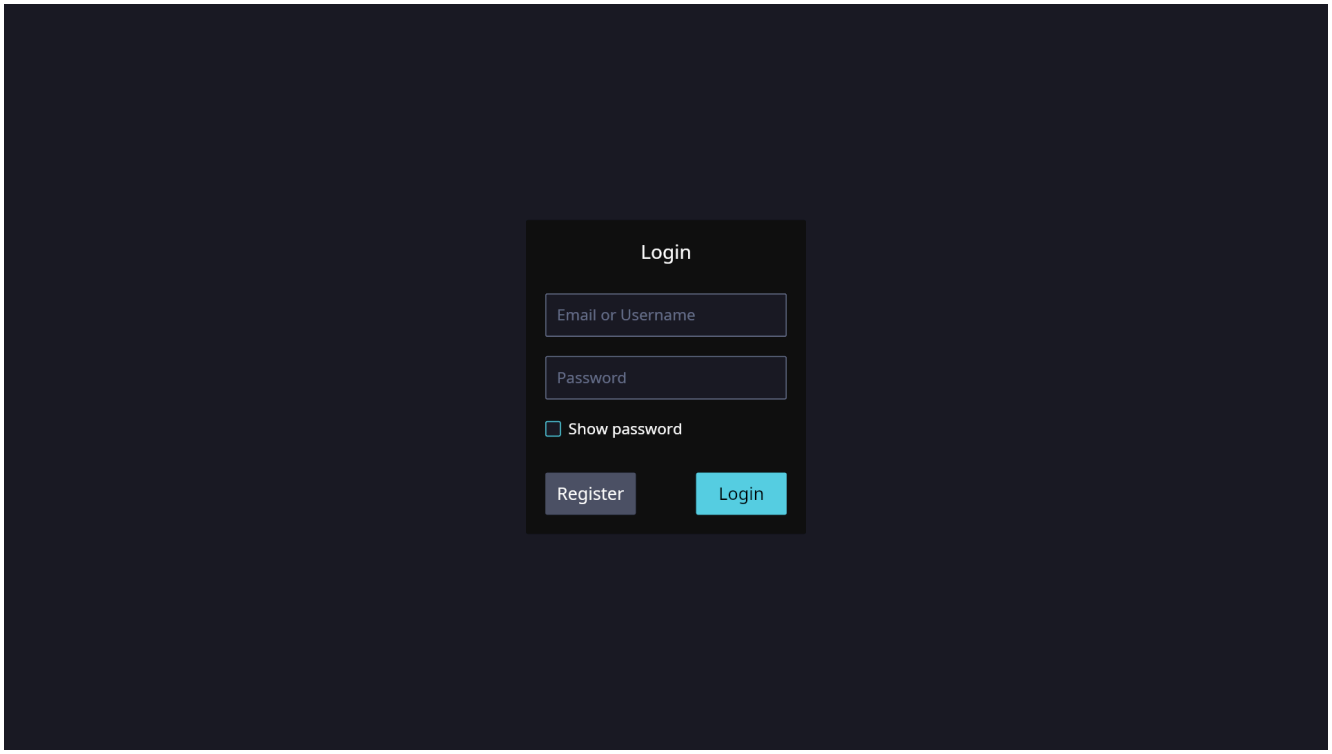

Рисунок 4.2 – Сторінка логіну (рисунок виконано самостійно)

Якщо у користувача немає облікового запису, то він може натиснути на кнопку реєстрації для переходу на сторінку реєстрації (див. рис. 4.3). Щоб створити новий обліковий запис. Користувач має надати ім’я користувача, електрону пошту та пароль.

Форми логіну та реєстрації перевіряють дані на валідність та не будуть робити зайвих запитів, якщо надана інформація не відповідає правилам інформаційної системи. У випадку перевірок. Які не можуть бути зроблені локально, система надішле запит до бази даних і відобразить результат у графічному інтерфейсі програми.

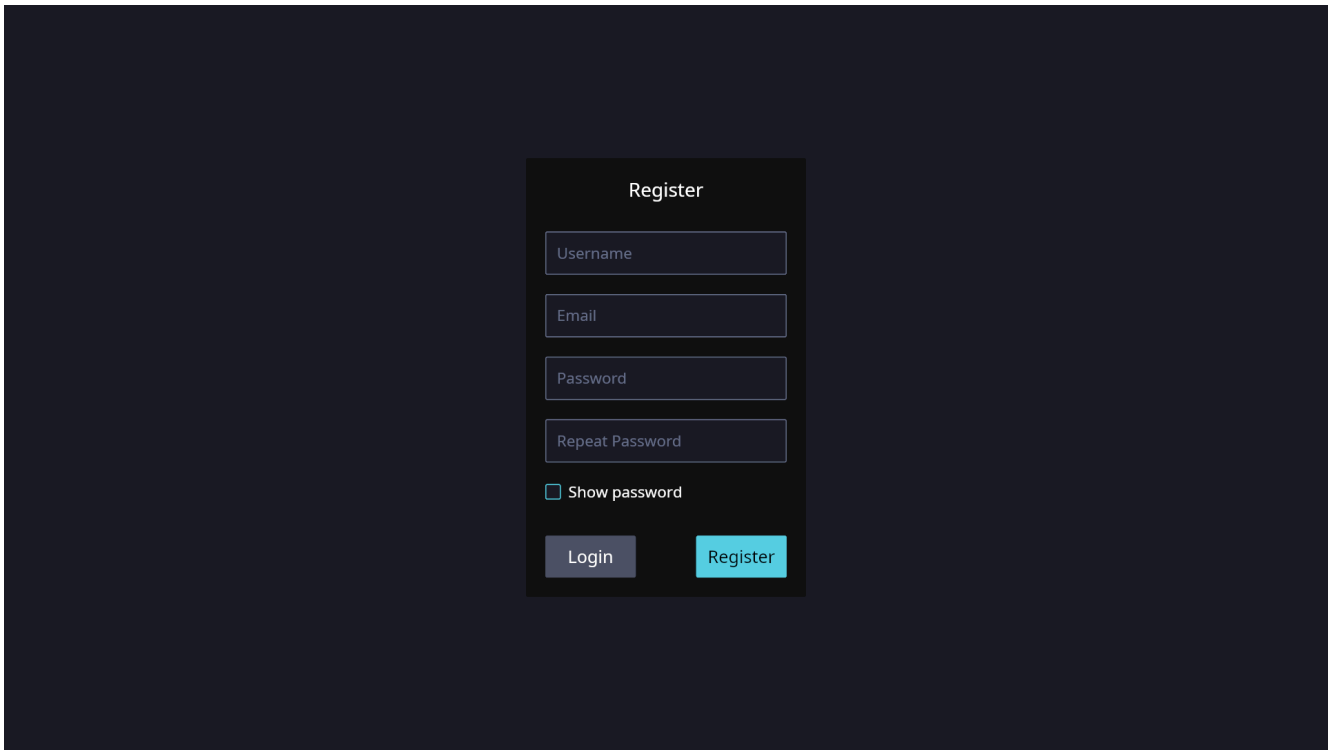

Рисунок 4.3 – Сторінка реєстрації (рисунок виконано самостійно)

Після успішного логіну або реєстрації програма перейде на сторінку пошуку (див. рис. 4.4), яка надає можливість шукати пакунки з багатьма способами фільтрування та сортування результатів. При наведені курсору миші на елементи пошуку можна побачити стисле пояснення їх функціоналу.

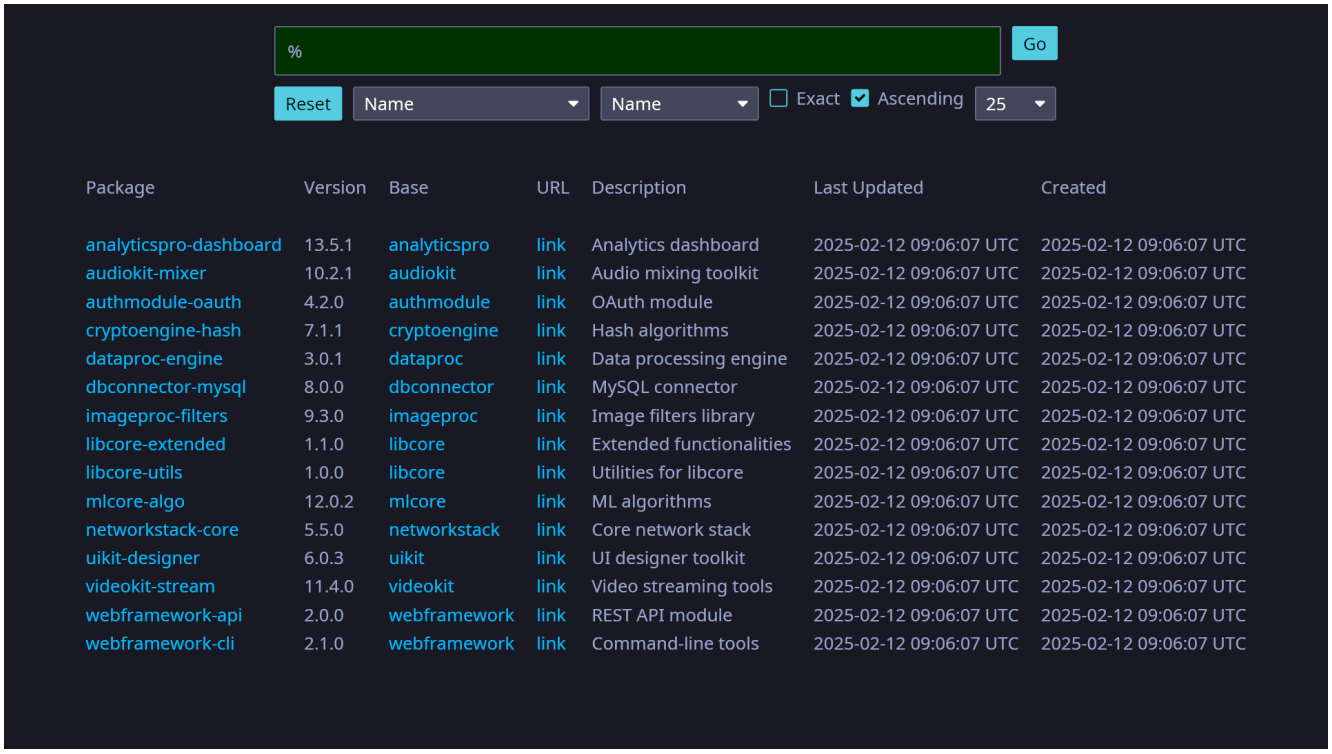

Рисунок 4.4 – Сторінка пошуку (рисунок виконано самостійно)

Назва пакунку, його бази та його веб-покликання на ресурс є інтерактивними. Якщо натиснути на назву пакунку, то відкриється вікно з переглядом інформації та статистики про пакунок (код формування відповідного запиту наведено в лістингу А.1). Якщо натиснути на назву бази пакунку, то відкриється вікно де буде інформація про базу пакунку (код формування відповідного запиту наведено в лістингу А.2). При натисканні на веб-покликання, воно відкриється в веб-браузері, котрий стоїть за замовчуванням в операційній системі користувача.

Розглянемо на прикладі сторінки пошуку обробку запиту на основі даних котрі ввів користувач у графічний інтерфейс.

Коли користувач натискає кнопку «Go», або натискає клавішу «Enter» у текстовому полі пошуку, генерується внутрішнє повідомлення «Search». Обробник повідомлень сторінки пошуку отримує це повідомлення, та починає перевірку даних текстового поля та створення структури даних з параметрами пошуку. Після чого, якщо на момент запиту не виконується інших пошукових запитів, дані з параметрами пошуку передаються до сервісу пошуку. Сервіс передає дані до адаптеру репозиторію пошуку. Адаптер репозиторію пошуку встановлює з’єднання до бази даних, і робить запит до адаптеру репозиторію пошуку бази даних з встановленим з’єднанням, після чого він закриє з’єднання до бази даних. Адаптер

репозиторію пошуку бази даних використовує передане йому підключення для виконання комплексного SQL запиту який будується на основі переданих йому параметрів пошуку. Після отримання результату з бази даних, він конвертує його у набір записів інформації про пакунок. Цей набір записів буде переданий назад до адаптеру репозиторію пошуку, потім до сервісу, й у кінці передається у повідомленні «RequestResult» до сторінки пошуку, яка зможе відобразити кожен запис як рядок у таблиці.

Якщо на будь-якому рівні абстракції виникне помилка, то вона буде передана до графічного інтерфейсу сторінки пошуку і користувач буде сповіщений про виникнення помилки.

У шарі графічного інтерфейсу екрану пошуку обробник повідомлення «Search» перевіряє дані пошукового запиту та надсилає їх до шару сервісу у пошуковий сервіс:

```
let search_data = Data {
    mode: self.mode.into(),
    order: self.order.into(),
    search: match self.input.submit() {
        Ok(x) => x,
        Err(t) => return Some(t.into()),
    },
    limit: self.limit.into(),
    exact: self.exact,
    ascending: self.ascending,
};

self.state = State::Searching;
let arc = self.service.clone();

return Some(Task::perform(async move {
    let Some(service) = arc.try_lock() else {
        return Err("other search request is being performed".into());
    };
    service.search(search_data).await
},
|r| Message::RequestResult(Arc::new(r)),
).into());
```

У шарі сервісу функція пошуку насилає дані до репозиторію пошуку:

```
async fn search(&self, data: Data) -> Result<Vec<search::Entry>> {
    self.repository.search(data.into()).await
}
```

У репозиторію пошуку функція пошуку встановлює підключення до сховища даних та надсилає дані пошукового запиту до шару взаємодії з базою даних:

```
async fn search(&self, data: Data) -> Result<Vec<Entry>> {
    let c = self.driver.open_connection().await?;
    let result = SR::search(&c, data).await?;
    D::close_connection(c).await?;

    Ok(result)
}
```

У шарі взаємодії з базою даних функція пошуку, на основі отриманих даних, будує комплексний динамічний запит та надсилає його використовуючи отримане з репозиторію пошуку підключення до сховища даних (код формування відповідного запиту наведено в лістингу Б.1).

Якщо виконати пошук пакунку за частковою назвою пакунку «core», відсортований в низхідному порядку за датою останнього оновлення, з лімітом у 75 результатів, то до бази даних буде побудований і відправлений наступний запит:

```
SELECT
  p.id,
  p.name,
  p.version,
  p.url,
  p.description,
  p.updated_at,
  p.created_at,
  pb.id AS base_id,
  pb.name AS base_name,
  (
    SELECT
      COUNT(DISTINCT pbur.user)
    FROM
      PackageBaseUserRoles pbur
    WHERE
      pbur.base = pb.id
      AND pbur.role = 3
  ) AS maintainers_num
FROM
  Packages p
  JOIN PackageBases pb ON p.base = pb.id
WHERE
  p.name LIKE ?
ORDER BY
  p.updated_at DESC
LIMIT
  75;
```

Якщо виконати пошук пакунку за повним юзернеймом користувача «alice», відсортований у висхідному порядку за часом створення, з лімітом у 25 результатів, то до бази даних буде побудований і відправлений наступний запит:

```
SELECT
  p.id,
  p.name,
  p.version,
  p.url,
  p.description,
  p.updated_at,
  p.created_at,
  pb.id AS base_id,
  pb.name AS base_name,
  (
    SELECT
      COUNT(DISTINCT pbur.user)
    FROM
      PackageBaseUserRoles pbur
    WHERE
      pbur.base = pb.id
      AND pbur.role = 3
  ) AS maintainers_num
FROM
  Packages p
  JOIN PackageBases pb ON p.base = pb.id
WHERE
  EXISTS (
    SELECT
      1
    FROM
      PackageBaseUserRoles pbur
      JOIN Users u ON pbur.user = u.id
    WHERE
      pbur.base = pb.id
      AND u.name = ?
  )
ORDER BY
  p.created_at ASC
LIMIT
  25;
```

Отриманий результат пошуку приходить у вигляді повідомлення «RequestResult» яке оброблюється в шарі графічного інтерфейсу:

```
Message::RequestResult(r) => match &*r {
    Ok(v) => self.state = State::Table(Table(v.clone())),
    Err(e) => self.state = State::Error(e.to_string()),
},
```

Після отриманого результату пошуку, він буде відображений у вигляді таблиці (код формування таблиці наведено в лістингу Б.2).

Подібну реалізацію мають всі частини програми. Представлений програмний код демонструє ефективну реалізацію принципів гексагональної архітектури [18]. Структура коду чітко відображає розділення на рівні, кожен з яких відповідає за визначену функціональну роль, що є ключовою характеристикою даної архітектурної парадигми. Цей підхід надає чітке розмежування відповідальності де кожен компонент системи, від інтерфейсу користувача до адаптерів баз даних, має чітко визначений набір обов'язків. Це полегшує процес розробки та спрощує підтримку програмного забезпечення у довгостроковій перспективі. Незалежність бізнес-логіки від інфраструктурних рішень дозволяє спростити процес міграції до простого доповнення арсеналу адаптерів. Сервісний шар, що містить основну бізнес-логіку пошуку, не залежить від конкретних технологій зберігання даних або реалізації графічного інтерфейсу, тому його можна буде використати в інших проектах. Крім того, модульна архітектура дозволяє проводити ізольоване тестування кожного компонента, за допомогою використання макетів (mock objects) для залежностей що забезпечить глибоке покриття коду тестами.

Комп'ютерна програма імплементує отримання наступних статистик (SQL запити цих статистик наведено в додатку В):

а) статистика облікових записів:
   1) загальна кількість користувачів;
   2) кількість активних користувачів;
   3) кількість користувачів які ніколи не заходили в систему;
   4) кількість користувачів, які ніколи не використовувалися більше року;
   5) кількість нових користувачів за останній тиждень;
   6) кількість нових користувачів за останній рік;
   7) кількість користувачів, які є супровідниками принаймні одного пакунка;
   8) кількість користувачів, які є авторами принаймні одного пакунка;
   9) кількість користувачів, які є пакувальниками принаймні одного пакунка;
   10) кількість користувачів, які позначили принаймні один пакунок.

б) статистика баз пакунків та пакунків репозиторію:

1) загальна кількість баз пакунків;
2) кількість осиротілих баз пакунків (вони не мають ні авторів, ні супроводжуючих);
3) кількість напівпокинутих баз пакунків (відправники та супровідники неактивні більше року);
4) кількість ніколи не оновлюваних баз пакунків;
5) кількість позначених баз пакунків;
6) загальна кількість пакунків;
7) кількість осиротілих пакунків (пакунків, які мають осиротілу базу);
8) кількість наполовину покинутих пакунків (пакунків, які мають напівпокинуту базу);
9) кількість ніколи не оновлюваних пакунків;
10) кількість позначених пакунків;
11) кількість пакунків які не мають залежностей;
12) кількість унікальних зламаних залежностей (пакунок залежить від назви пакунку, але жоден пакунок не постачає цю назву).

ВИСНОВКИ

В результаті виконання курсової роботи було розроблено інформаційну систему «Репозиторій пакунків» для організації ефективної кооперації над програмними пакунками. В процесі розробки було проведено ґрунтовний аналіз предметної області, визначено ключові вимоги до системи та спроектовано оптимальну структуру бази даних для зберігання інформації про пакунки, їх версії, залежності, користувачів та їхні ролі.

Створена система забезпечує зручний інтерфейс для пошуку пакунків, управління пакунками та кооперації між розробниками. Реалізовано функціонал створення та оновлення пакунків, відстеження залежностей та взаємозв’язків між пакунками, а також систему ролей для контролю доступу.

База даних спроектована на основі реляційної моделі та нормалізована до третьої нормальної форми, що забезпечує оптимальну структуру даних та відсутність їх надлишковості. Під час розробки було використано мову програмування Rust [3], систему управління базами даних MySQL [6] та графічну бібліотеку iced [4]. Крім того використовується контейнеризація за допомогою Docker [15] та Docker Compose [16] що забезпечує надійність роботи системи, її масштабованість та простоту розгортання в різних середовищах.

Розроблена інформаційна система значно спрощує процес управління та отримання інформації про програмні пакунки. Вона відповідає поставленим завданням та має потенціал подальшої реалізації завдяки використанню дуже модульної архітектури моделювання програмного забезпечення, відомої як Гексагональна архітектура [18].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. AUR (en) - Home. AUR (en) - Home. URL: https://aur.archlinux.org (дата звернення: 06.02.2025).

2. Arch Linux. Arch Linux. URL: https://archlinux.org (дата звернення: 09.02.2025).

3. Rust Programming Language. Rust Programming Language. URL: https://www.rust-lang.org (дата звернення: 09.02.2025).

4. iced - A cross-platform GUI library for Rust. iced - A cross-platform GUI library for Rust. URL: https://iced.rs (дата звернення: 09.02.2025).

5. GitHub - launchbadge/sqlx: The Rust SQL Toolkit. An async, pure Rust SQL crate featuring compile-time checked queries without a DSL. Supports PostgreSQL, MySQL, and SQLite. GitHub. URL: https://github.com/launchbadge/sqlx (дата звернення: 09.02.2025).

6. MySQL. MySQL. URL: https://www.mysql.com (дата звернення: 09.02.2025).

7. Neovim. Neovim. URL: https://neovim.io (дата звернення: 09.02.2025).

8. AUR (en) - Packages. AUR (en) - Home. URL: https://aur.archlinux.org/packages?O=0&K=librewolf (дата звернення: 06.02.2025).

9. AUR (en) - librewolf-bin. AUR (en) - Home. URL: https://aur.archlinux.org/packages/librewolf-bin (дата звернення: 06.02.2025).

10. AUR (en) - Account lsf. AUR (en) - Home. URL: https://aur.archlinux.org/account/lsf (дата звернення: 06.02.2025).

11. AUR (en) - Packages. AUR (en) - Home. URL: https://aur.archlinux.org/account/lsf (дата звернення: 06.02.2025).

12. GitHub - P-H-C/phc-winner-argon2: The password hash Argon2, winner of PHC. GitHub. URL: https://github.com/P-H-C/phc-winner-argon2 (дата звернення: 09.02.2025).

13. Database Normalization – Normal Forms 1nf 2nf 3nf Table Examples. Kolade Chris. URL: https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples (дата звернення: 09.02.2025).

14. Elm - delightful language for reliable web applications.. Elm - delightful language for reliable web applications.. URL: https://elm-lang.org (дата звернення: 11.02.2025).

15. Docker: accelerated container application development. Docker. URL: https://www.docker.com (дата звернення: 09.02.2025).

16. Docker compose. Docker Documentation. URL: https://docs.docker.com/compose (дата звернення: 09.02.2025).

17. repo. Білоус А. А.. URL: https://gitea.linerds.us/0x1D8/repo (дата звернення: 10.02.2025).

18. How to apply hexagonal architecture to Rust. Josip Benko-Đaković. URL: https://www.barrage.net/blog/technology/how-to-apply-hexagonal-architecture-to-rust (дата звернення: 09.02.2025).

ДОДАТОК А

SQL запити отримання інформації

А.1 Інформація пакунку

Інформація пакунку в репозиторії складається з наступної інформації:

- «package_name» – назва пакунку;
- «package_version» – версія пакунку;
- «package_url» – веб-покликання на ресурс пакунку;
- «package_description» – опис пакунку;
- «base_id» – ідентифікатор бази пакунку;
- «base_name» – назва бази пакунку;
- «base_description» – опис бази пакунку;
- «flagged_at» – час помітки пакунку;
- «updated_at» – час оновлення пакунку;
- «created_at» – час створення пакунку;
- «depends_aliases» – назви пакунків від котрих залежить пакунок;
- «makedepends_aliases» – назви пакунків від котрих пакунок залежить під час збірки;
- «optdepends_aliases» – назви пакунків від котрих пакунок опціонально залежить;
- «replaces_aliases» – назви пакунків котрі поточний пакунок замінює;
- «provides_aliases» – назви пакунків котрі поточний пакунок постачає;
- «conflicts_aliases» – назви пакунків з котрими пакунок конфліктує.

Наступний SQL запит забезпечує отримання цієї інформації:

```
  -- fetch package once
  WITH selected_package AS (
  SELECT
    *
  FROM
    Packages
  WHERE
    id = ?
    ),
dependency_aliases AS (
  SELECT
    pd.package,
    pd.dependency_type,
```

```
    pd.dependency_package_name AS alias
  FROM
    PackageDependencies pd
  WHERE
    pd.package = (
      SELECT
        id -- from CTE
      FROM
        selected_package
    )
    ),
relation_aliases AS (
  SELECT
    pr.package,
    pr.relation_type,
    pr.relation_package_name AS alias
  FROM
    PackageRelations pr
  WHERE
    pr.package = (
      SELECT
        id -- from CTE
      FROM
        selected_package
    )
    )
SELECT
  -- package
  sp.name AS package_name,
  sp.version AS package_version,
  sp.url AS package_url,
  sp.description AS package_description,
  sp.base AS base_id,
  -- timestamps
  sp.flagged_at,
  sp.updated_at,
  sp.created_at,
  -- package base
  pb.name AS base_name,
  pb.description AS base_description,

  -- deps
  GROUP_CONCAT(
    CASE WHEN da.dependency_type = 1 THEN da.alias END
  ) AS depends_aliases,
  GROUP_CONCAT(
    CASE WHEN da.dependency_type = 2 THEN da.alias END
  ) AS makedepends_aliases,
  GROUP_CONCAT(
    CASE WHEN da.dependency_type = 4 THEN da.alias END
  ) AS optdepends_aliases,

  -- relations
  GROUP_CONCAT(
    CASE WHEN ra.relation_type = 3 THEN ra.alias END
  ) AS replaces_aliases,
  -- replaces
  GROUP_CONCAT(
```

```
    CASE WHEN ra.relation_type = 2 THEN ra.alias END
  ) AS provides_aliases,
  -- provides
  GROUP_CONCAT(
    CASE WHEN ra.relation_type = 1 THEN ra.alias END
  ) AS conflicts_aliases -- conflicts
FROM
  selected_package sp
  JOIN PackageBases pb ON sp.base = pb.id
  LEFT JOIN dependency_aliases da ON sp.id = da.package
  LEFT JOIN relation_aliases ra ON sp.id = ra.package -- all
nonaggregated fields
GROUP BY
  sp.name, sp.version, sp.url, sp.description, sp.base, pb.name,
pb.description, sp.flagged_at, sp.updated_at, sp.created_at;
```

А.2 Інформація бази пакунку

Інформація бази пакунку в репозиторії складається з наступної інформації:

- «package_base_name» – ім’я бази пакунка;
- «package_base_description» – базовий опис бази пакунку;
- «submitters» – ідентифікатори та юзернейми користувачів, які мають роль автора;
- «maintainers» – ідентифікатори та юзернейми користувачів, які мають роль супроводжуючого;
- «packagers» – ідентифікатори та юзернейми користувачів, які мають роль пакувальника;
- «packages» – ідентифікатори та назви пакунків, які мають цю базу пакунків;

Наступний SQL запит забезпечує отримання цієї інформації:

```
-- fetch package base once
WITH selected_base AS (
  SELECT
    *
  FROM
    PackageBases
  WHERE
    id = ?
    ),
-- preaggregate user roles
user_roles AS (
  SELECT
    pbur.base,
    pbur.role,
    GROUP_CONCAT(
      CONCAT(pbur.user, ':', u.name)
```

```
    ) AS users
  FROM
    PackageBaseUserRoles pbur
    JOIN Users u ON pbur.user = u.id
  WHERE
    pbur.base = (
      SELECT
        id
      FROM
        selected_base
    )
-- avoid duplication
  GROUP BY
    pbur.base,
    pbur.role
    ),
-- preaggregate package info
package_info AS (
  SELECT
    p.base,
    GROUP_CONCAT(
      CONCAT(p.id, ':', p.name)
    ) AS packages
  FROM
    Packages p
  WHERE
    p.base = (
      SELECT
        id
      FROM
        selected_base
    )
-- avoid duplication
  GROUP BY
    p.base
    )
SELECT
  sb.name AS package_base_name,
  sb.description AS package_base_description,
  -- extract user roles
  MAX(CASE WHEN ur.role = 1 THEN ur.users END) AS
submitters,
  MAX(CASE WHEN ur.role = 3 THEN ur.users END) AS
maintainers,
  MAX(CASE WHEN ur.role = 2 THEN ur.users END) AS packagers,

  -- extract packages
  MAX(pi.packages) AS packages
FROM
  selected_base sb
  LEFT JOIN user_roles ur ON sb.id = ur.base
  LEFT JOIN package_info pi ON sb.id = pi.base
-- needed for aggregation functions
GROUP BY
  sb.name, sb.description;
```

ДОДАТОК Б

Код пошукового процесу

Б.1 Код формування та надсилання запиту пошуку до бази даних

```
async fn search(connection: &E, data: Data) -> Result<Vec<Entry>> {
    let mut builder = QueryBuilder::new(
        "SELECT \
            p.id, p.name, p.version, p.url, p.description, \
            p.updated_at, p.created_at, \
            pb.id AS base_id, pb.name AS base_name, \
        ( \
            SELECT COUNT(DISTINCT pbur.user) \
            FROM PackageBaseUserRoles pbur \
            WHERE pbur.base = pb.id AND pbur.role = 3 \
        ) AS maintainers_num \
        FROM Packages p \
        JOIN PackageBases pb ON p.base = pb.id ",
    );

    let mut push_search = |cond, param| {
        builder.push(format_args!(
            " {cond} {param} {} ",
            if data.exact { "=" } else { "LIKE" }
        ));
        builder.push_bind(if data.exact {
            data.search.to_string()
        } else {
            format!("%{}%", data.search.as_str())
        });
    };

    let join_user = " JOIN PackageBaseUserRoles pbur ON pb.id =
pbur.base JOIN Users u ON pbur.user = u.id WHERE ";

    match data.mode {
        Mode::Url => push_search("WHERE", "p.url"),
        Mode::Name => push_search("WHERE", "p.name"),
        Mode::PackageBase => push_search("WHERE", "pb.name"),
        Mode::Description => push_search("WHERE", "p.description"),
        Mode::BaseDescription => push_search("WHERE",
"pb.description"),
        Mode::NameAndDescription => {
            // WHERE (p.name LIKE '%search_term%' OR p.description
LIKE '%search_term%')
            builder.push(" WHERE p.name LIKE ");
            builder.push_bind(format!("%{}%", data.search.as_str()));
            builder.push(" OR p.description LIKE ");
            builder.push_bind(format!("%{}%", data.search.as_str()));
        }
        Mode::User => {
            push_search(
                "WHERE EXISTS ( \
                SELECT 1 \
                FROM PackageBaseUserRoles pbur \
                JOIN Users u ON pbur.user = u.id \
                WHERE pbur.base = pb.id AND",
```

```
                "u.name",
            );
            builder.push(" ) ");
        }
        Mode::Flagger => {
            push_search(join_user, "u.name");
            builder.push(" AND pbur.role = 4 ");
        } // 4
        Mode::Packager => {
            push_search(join_user, "u.name");
            builder.push(" AND pbur.role = 2 ");
        } // 2
        Mode::Submitter => {
            push_search(join_user, "u.name");
            builder.push(" AND pbur.role = 1 ");
        } // 1
        Mode::Maintainer => {
            push_search(join_user, "u.name");
            builder.push(" AND pbur.role = 3 ");
        } // 3
    }

    builder.push(format_args!(
        " ORDER BY {} {} LIMIT {};",
        match data.order {
            Order::Name => "p.name",
            Order::Version => "p.version",
            Order::BaseName => "pb.name",
            Order::UpdatedAt => "p.updated_at",
            Order::CreatedAt => "p.created_at",
        },
        if data.ascending { "ASC" } else { "DESC" },
        data.limit
    ));

    let mut entries = Vec::new();

    let mut rows = builder.build().fetch(connection);
    while let Some(row) = rows.try_next().await? {
        entries.push(Entry {
            id: row.try_get("id")?,
            name: row.try_get("name")?,
            version: row.try_get("version")?,
            base_id: row.try_get("base_id")?,
            base_name: row.try_get("base_name")?,
            url: row.try_get("url")?,
            description: row.try_get("description")?,
            // submitter_id: row.try_get("submitter_id")?,
            // submitter_name: row.try_get("submitter_name")?,
            updated_at: row.try_get("updated_at")?,
            created_at: row.try_get("created_at")?,
        });
    }

    Ok(entries)
}
```

Б.2 Код відображення зчитаних результатів пошуку у вигляді таблиці

```
pub fn view(&self) -> Element<'static, Message> {
    let mut table: Vec<_> = [
        "Package",      // 0
        "Version",      // 1
        "Base",         // 2
        "URL",          // 3
        "Description",  // 4
        "Last Updated", // 5
        "Created",      // 6
    ]
    .into_iter()
    .map(|s| { let mut v = Vec::with_capacity(self.0.len());
        v.push(s.into());
        v.push("".into());
        v }).collect();

    for entry in &self.0 {
        table[0].push(url(&entry.name,
Message::PackagePressed(entry.id)));
        table[1].push(text(entry.version.to_string()).into());
        table[2].push(url(&entry.base_name,
Message::BasePressed(entry.base_id)));
        table[3].push(
            entry.url.as_ref().map_or("-".into(), |s|
                tip(
                    url(&"link", Message::URLPressed(s.clone())),
                    s.clone(),
                    tip::Position::Bottom,
                ),
              ),
        );
        table[4].push(text(entry.description.to_string()).into());
        table[5].push(text(entry.updated_at.to_string()).into());
        table[6].push(text(entry.created_at.to_string()).into());
        //
table[5].push(Element::from(column( entry .maintainers .iter() .map(|
(id, s)| url(s, Message::UserPressed(*id))),)));
    }

    scroll(
        row(table
            .into_iter()
            .map(|v| Column::from_vec(v).spacing(5).into()))
        .spacing(20)
        .padding(30),
    )
}
```

ДОДАТОК В

SQL запити статистик

В.1 Статистика користувачів

Статистика користувачів репозиторію складається з наступної інформації:

- «total_users» – загальна кількість користувачів;
- «active_users» – кількість активних користувачів;
- «never_used_accounts» – кількість користувачів які ніколи не заходили в систему;
- «inactive_users» – кількість користувачів, які ніколи не використовувалися більше року;
- «new_users_past_week» – кількість нових користувачів за останній тиждень;
- «new_users_past_year» – кількість нових користувачів за останній рік;
- «maintainers_count» – кількість користувачів, які є супровідниками принаймні одного пакунка;
- «submitters_count» – кількість користувачів, які є авторами принаймні одного пакунка;
- «packagers_count» – кількість користувачів, які є пакувальниками принаймні одного пакунка;
- «flaggers_count» – кількість користувачів, які позначили принаймні один пакунок.

Наступний SQL запит забезпечує отримання цієї інформації:

```
  WITH user_stats AS (
  SELECT
    COUNT(*) AS total_users,
    COUNT(
      CASE WHEN last_used IS NOT NULL
      AND last_used <= updated_at - INTERVAL '1' YEAR THEN 1 END
    ) AS inactive_users,
    COUNT(
      CASE WHEN last_used IS NULL THEN 1 END
    ) AS never_used_accounts,
    COUNT(
      CASE WHEN created_at >= CURRENT_TIMESTAMP - INTERVAL '7' DAY
THEN 1 END
    ) AS new_users_past_week,
    COUNT(
      CASE WHEN created_at >= CURRENT_TIMESTAMP - INTERVAL '1' YEAR
```

```
THEN 1 END
    ) AS new_users_past_year
  FROM
    Users
),
role_counts AS (
  SELECT
    user,
    COUNT(CASE WHEN role = 3 THEN 1 END) AS maintainer_count,
    COUNT(CASE WHEN role = 1 THEN 1 END) AS submitter_count,
    COUNT(CASE WHEN role = 2 THEN 1 END) AS packager_count,
    COUNT(CASE WHEN role = 4 THEN 1 END) AS flagger_count
  FROM
    PackageBaseUserRoles
  GROUP BY
    user
)
SELECT
  us.total_users,
  us.inactive_users,
  us.never_used_accounts,
  (
    us.total_users - us.inactive_users - us.never_used_accounts
  ) AS active_users,
  us.new_users_past_week,
  us.new_users_past_year,
  (
    SELECT
      COUNT(*)
    FROM
      role_counts
    WHERE maintainer_count > 0
  ) AS maintainers_count,
  (
    SELECT
      COUNT(*)
    FROM
      role_counts
    WHERE submitter_count > 0
  ) AS submitters_count,
  (
    SELECT
      COUNT(*)
    FROM
      role_counts
    WHERE packager_count > 0
  ) AS packagers_count,
  (
    SELECT
      COUNT(*)
    FROM
      role_counts
    WHERE flagger_count > 0
  ) AS flaggers_count
FROM user_stats us;
```

В.2 Статистика пакунків

Статистика баз пакунків та пакунків репозиторію складається з наступної інформації:

- «total_package_bases» – загальна кількість баз пакунків;
- «orphaned_package_bases» – кількість осиротілих баз пакунків (вони не мають ні авторів, ні супроводжуючих);
- «semi_orphaned_package_bases» – кількість напівпокинутих баз пакунків (відправники та супровідники неактивні більше року);
- «never_updated_package_bases» – кількість ніколи не оновлюваних баз пакунків;
- «flagged_package_bases» – кількість позначених баз пакунків;
- «total_packages» – загальна кількість пакунків;
- «orphaned_packages» – кількість осиротілих пакунків (пакунків, які мають осиротілу базу);
- «semi_orphaned_packages» – кількість наполовину покинутих пакунків (пакунків, які мають напівпокинуту базу);
- «never_updated_packages» – кількість ніколи не оновлюваних пакунків;
- «flagged_packages» – кількість позначених пакунків;
- «packages_without_dependencies» – кількість пакунків які не мають залежностей;
- «unique_broken_dependencies» – кількість унікальних зламаних залежностей (пакунок залежить від назви пакунку, але жоден пакунок не постачає цю назву).

Наступний SQL запит забезпечує отримання цієї інформації:

```
WITH base_stats AS (
SELECT
  pb.id AS base_id,
  pb.created_at = pb.updated_at AS never_updated,
  -- true if package base was never updated after creation
  EXISTS (
    SELECT
      1
    FROM
      PackageBaseUserRoles pbur
    WHERE
      pbur.base = pb.id
```

```
        AND pbur.role IN (1, 3) -- at least one submitter (1) or
maintainer (3)
        ) AS has_owners,
    EXISTS (
      SELECT
        1
      FROM
        PackageBaseUserRoles pbur
        JOIN Users u ON u.id = pbur.user
      WHERE
        pbur.base = pb.id
        AND pbur.role IN (1, 3) -- submitter or maintainer
        AND u.last_used > NOW() - INTERVAL '1' YEAR -- have been
active in the last year
        ) AS has_active_owners,
    EXISTS (
      SELECT
        1
      FROM
        PackageBaseUserRoles pbur
      WHERE
        pbur.base = pb.id
        AND pbur.role = 4 -- at least one flagger
        ) AS has_flagger
  FROM
    PackageBases pb
),
package_stats AS (
  SELECT
    p.id AS package_id,
    p.base AS base_id,
    p.created_at = p.updated_at AS never_updated,
    -- true if package was never updated after creation
    p.flagged_at IS NOT NULL AS is_flagged,
    -- package is flagged
    NOT EXISTS (
      SELECT
        1
      FROM
        PackageDependencies pd
      WHERE
        pd.package = p.id
        AND pd.dependency_type IN (1, 2) -- only count critical
missing dependencies (depends, makedepends)
        ) AS no_dependencies
  FROM
    Packages p
),
broken_dependencies AS (
  SELECT
    pd.dependency_package_name
  FROM
    PackageDependencies pd
  WHERE
    NOT EXISTS (
      SELECT
        1
      FROM
```

```
        PackageRelations pr
      WHERE
        pr.relation_type = 2 -- "provides" relation
        AND pr.relation_package_name = pd.dependency_package_name --
at least one package "provides" the dependency
        )
  GROUP BY
    pd.dependency_package_name -- count unique broken dependencies
    )
SELECT
  (
    SELECT
      COUNT(*)
    FROM
      PackageBases
  ) AS total_package_bases,
  -- orphaned package bases with no submitters or maintainers
  SUM(
    CASE WHEN NOT has_owners THEN 1 ELSE 0 END
  ) AS orphaned_package_bases,
  -- semi-orphaned bases have only inactive submitters and
maintainers
  SUM(
    CASE WHEN has_owners
    AND NOT has_active_owners THEN 1 ELSE 0 END
  ) AS semi_orphaned_package_bases,
  -- only count package bases that still have packages to avoid
counting inactive bases
  SUM(
    CASE WHEN never_updated
    AND base_id IN (
      SELECT
        DISTINCT base
      FROM
        Packages
    ) THEN 1 ELSE 0 END
  ) AS never_updated_package_bases,
  -- package bases with at least one flagger
  SUM(
    CASE WHEN has_flagger THEN 1 ELSE 0 END
  ) AS flagged_package_bases,
  (
    SELECT
      COUNT(*)
    FROM
      Packages
  ) AS total_packages,
  -- orphaned packages are those whose base is orphaned
  SUM(
    CASE WHEN base_id IN (
      SELECT
        base_id
      FROM
        base_stats
      WHERE
        NOT has_owners
    ) THEN 1 ELSE 0 END
  ) AS orphaned_packages,
```

```
  -- semi-orphaned packages belong to semi-orphaned bases
  SUM(
    CASE WHEN base_id IN (
      SELECT
        base_id
      FROM
        base_stats
      WHERE
        has_owners
        AND NOT has_active_owners
    ) THEN 1 ELSE 0 END
  ) AS semi_orphaned_packages,
  -- count unique packages that were never updated
  (
    SELECT
      COUNT(DISTINCT package_id)
    FROM
      package_stats
    WHERE
      never_updated
  ) AS never_updated_packages,
  -- count unique flagged packages
  (
    SELECT
      COUNT(DISTINCT package_id)
    FROM
      package_stats
    WHERE
      is_flagged
  ) AS flagged_packages,
  -- count packages that have no required dependencies
  (
    SELECT
      COUNT(*)
    FROM
      package_stats
    WHERE
      no_dependencies
  ) AS packages_without_dependencies,
  -- count unique broken dependencies (dependency exists but no
package provides it)
  (
    SELECT
      COUNT(*)
    FROM
      broken_dependencies
  ) AS unique_broken_dependencies
FROM
  base_stats;
```