

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №2

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Діаграма варіантів використання. Сценарії варіантів
використання. Діаграми uml. Діаграми класів. Концептуальна модель
системи»

Варіант №26

Виконав:
студент групи ІА-23
Мозоль В.О

Перевірив:
Мягкий М. Ю.

Зміст

Тема.....	3
Мета.....	3
Завдання.....	3
Обрана тема.....	3
Короткі теоретичні відомості.....	4
Хід роботи.....	13
Завдання №2.....	13
Завдання №3.....	13
Завдання №4.....	18
Завдання №5.....	21
Структура існуючої частини проекту.....	23
Висновки.	23

Тема.

Діаграма варіантів використання. Сценарії варіантів використання.

Діаграми uml. Діаграми класів. Концептуальна модель системи

Мета.

Розробка та аналіз концептуальної моделі для обраного варіанту з використанням UML-діаграм. Зокрема створення діаграми варіантів використання. Розробка детальних сценаріїв використання. Проектування діаграми класів Створення концептуальної моделі бази даних.

Завдання.

1. Ознайомитися з короткими теоретичними відомостями.
2. Проаналізуйте тему та намалюйте схему прецеденту, що відповідає обраній темі лабораторії.
3. Намалюйте діаграму класів для реалізованої частини системи.
4. Виберіть 3 прецеденти і напишіть на їх основі прецеденти.
5. Розробити основні класи і структуру системи баз даних.
6. Класи даних повинні реалізувати шаблон Репозиторію для взаємодії з базою даних.

Обрана тема.

26 Download manager (iterator, command, observer, template method, composite, p2p)

Інструмент для скачування файлів з інтернету по протоколах http або https з можливістю продовження завантаження в зупиненому місці, розподілу швидкостей активним завантаженням, ведення статистики завантажень, інтеграції в основні браузери (firefox, opera, internet explorer, chrome).

Короткі теоретичні відомості.

Мова UML являє собою загальноцільову мову візуального моделювання, яка розроблена для специфікації, візуалізації, проектування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Мова UML є досить строгим і потужним засобом моделювання, який може бути ефективно використаний для побудови концептуальних, логічних і графічних моделей складних систем різного цільового призначення. Ця мова увібрала в себе найкращі якості та досвід методів програмної інженерії, які успішно використовувалися протягом останніх років при моделюванні великих і складних систем.

З точки зору методології ООАП (об'єктно-орієнтованого аналізу і проектування) досить повна модель складної системи представляє собою певну кількість взаємопов'язаних представлень (views), кожне з яких відображає аспект поведінки або структури системи. При цьому найбільш загальними представленнями складної системи прийнято вважати статичне і динамічне, які в свою чергу можуть підрозділятися на інші більш часткові.

Принцип ієрархічної побудови моделей складних систем приписує розглядати процес побудови моделей на різних рівнях абстрагування або деталізації в рамках фіксованих представлень.

Рівень представлення

Рівень представлення (layer) — спосіб організації і розгляду моделі на одному рівні абстракції, який представляє горизонтальний зріз архітектури моделі, в той час як розбиття представляє її вертикальний зріз.

При цьому вихідна або первинна модель складної системи має найбільш загальне представлення і відноситься до концептуального рівня. Така модель, що отримала назву концептуальної, будується на початковому етапі проектування і може не містити багатьох деталей і аспектів модельованої системи. Наступні моделі конкретизують концептуальну модель, доповнюючи її представленнями логічного і фізичного рівня.

В цілому ж процес ООАП можна розглядати як послідовний перехід від розробки найбільш загальних моделей і представлень концептуального рівня до більш часткових і детальних представлень логічного і фізичного рівня. При цьому на кожному етапі ООАП дані моделі послідовно доповнюються все більшою кількістю деталей, що дозволяє їм більш адекватно відображати різні аспекти конкретної реалізації складної системи.

Діаграма

У рамках мови UML всі представлення про модель складної системи фіксуються у вигляді спеціальних графічних конструкцій, що отримали назву діаграм.

Діаграма (diagram) — графічне представлення сукупності елементів моделі у формі зв'язного графа, вершинам і ребрам (дугам) якого приписується певна семантика. Нотація канонічних діаграм — основний засіб розробки моделей на мові UML.

В нотації мови UML визначені наступні види канонічних діаграм:

- варіантів використання (use case diagram)
- класів (class diagram)
- кооперації (collaboration diagram)
- послідовності (sequence diagram)
- станів (statechart diagram)
- діяльності (activity diagram)
- компонентів (component diagram)
- розгортання (deployment diagram)

Діаграма варіантів використання

Перелік цих діаграм та їх назви є канонічними в тому сенсі, що представляють собою невід'ємну частину графічної нотації мови UML. Більше того, процес ООАП нерозривно пов'язаний з процесом побудови цих діаграм. При цьому сукупність побудованих таким чином діаграм є самодостатньою в тому сенсі, що в них міститься вся інформація, яка необхідна для реалізації проекту складної системи.

Кожна з цих діаграм деталізує і конкретизує різні представлення про модель складної системи в термінах мови UML. При цьому діаграма варіантів використання представляє собою найбільш загальну концептуальну модель складної системи, яка є вихідною для побудови всіх інших діаграм. Діаграма класів, по своїй суті, логічна модель, що відображає статичні аспекти структурної побудови складної системи.

Діаграма варіантів використання (Use-Cases Diagram)

Діаграма варіантів використання (Use-Cases Diagram) - це UML діаграма, за допомогою якої в графічному вигляді можна зобразити вимоги до розроблюваної системи. Діаграма варіантів використання – це вихідна концептуальна модель проекрованої системи, вона не описує внутрішній устрій системи.

Діаграми варіантів використання призначені для:

Визначення загальної межі функціональності проекрованої системи

Сформулювати загальні вимоги до функціональної поведінки проектованої системи

Розробка вихідної концептуальної моделі системи

Створення основи для виконання аналізу, проектування, розробки і тестування

Діаграми варіантів використання є відправною точкою при зборі вимог до програмного продукту та його реалізації. Дана модель будується на аналітичному етапі побудови програмного продукту (збір і аналіз вимог) і дозволяє бізнес-аналітикам отримати більш повне уявлення про необхідне програмне забезпечення і документувати його.

Актори (actor)

Актором називається будь-який об'єкт, суб'єкт або система, що взаємодіє з модельованою бізнес-системою ззовні для досягнення своїх цілей або вирішення певних завдань. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка служить джерелом впливу на модельовану систему.

Варіанти використання (use case)

Варіант використання служить для опису послуг, які система надає актору. Іншими словами, кожен варіант використання визначає набір дій, що виконуються системою при діалозі з актором. Кожен варіант використання представляє собою послідовність дій, яка повинна бути виконана проектованою системою при взаємодії її з відповідним актором, самі ці дії не відображаються на діаграмі.

Варіант використання відображається еліпсом, всередині якого міститься його коротке ім'я з великої літери у формі іменника або дієслова.

Приклади варіантів використання: реєстрація, авторизація, оформлення замовлення, перегляд замовлення, перевірка стану поточного рахунку і т.д.

Відношення на діаграмі варіантів використання

Відношення (relationship) — семантичний зв'язок між окремими елементами моделі.

Один актор може взаємодіяти з декількома варіантами використання. В цьому випадку цей актор звертається до кількох служб даної системи. У свою чергу, один варіант використання може взаємодіяти з декількома акторами, надаючи для всіх них свій функціонал.

Існують наступні відношення:

- асоціації
- узагальнення

- залежність (складається з включення і розширення)

Асоціація

Асоціація (association) – узагальнене, невідоме відношення між актором і варіантом використання. Позначається суцільною лінією між актором і варіантом використання.

Направлена асоціація (directed association) – те ж, що і проста асоціація, але показує, що варіант використання ініціалізується актором. Позначається стрілкою.

Направлена асоціація дозволяє ввести поняття основного актора (він є ініціатором асоціації) і другорядного актора (варіант використання є ініціатором, тобто передає актору довідкові відомості або звіт про виконану роботу).

Особливості використання відношення асоціації:

Один варіант використання може мати кілька асоціацій з різними акторами.

Два варіанти використання, що відносяться до одного і того ж актора, не можуть бути асоційовані, оскільки кожен з них описує закінчений фрагмент функціональності актора.

Узагальнення

Відношення узагальнення (generalization) – показує, що нащадок успадковує атрибути і поведінку свого прямого предка, тобто один елемент моделі є спеціальним або частковим випадком іншого елемента моделі. Може застосовуватися як для акторів, так для варіантів використання.

Графічно відношення узагальнення позначається суцільною лінією зі стрілкою у формі не зафарбованого трикутника, яка вказує на батьківський варіант використання.

Відношення включення та розширення

Відношення включення

Відношення включення (include) - окремий випадок загального відношення залежності між двома варіантами використання, при якому деякий варіант використання містить поведінку, визначену в іншому варіанті використання.

Залежний варіант використання називають базовим, а незалежний – включуваним. Включення означає, що кожне виконання варіанта використання А завжди буде включати в себе виконання варіанта використання Б. На практиці відношення включення використовується для моделювання ситуації, коли існує загальна частина поведінки двох або більше варіантів використання.

Загальна частина виноситься в окремий варіант використання, тобто типовий приклад повторного використання функціональності.

Особливості використання відношення включення:

Один базовий варіант використання може бути пов'язаний відношенням включення з декількома включуваними варіантами використання.

Один варіант використання може бути включений в інші варіанти використання.

На одній діаграмі варіантів використання не може бути замкнутого шляху по відношенню включення.

Відношення розширення

Відношення розширення (extend) – показує, що варіант використання розширює базову послідовність дій і вставляє власну послідовність. При цьому на відміну від типу відносин "включення" розширена послідовність може здійснюватися в залежності від певних умов.

Графічно зображення - пунктирна стрілка направлена від залежного варіанта (розширюючого) до незалежного варіанта (базового) з ключовим словом <<extend>>.

Відношення розширення дозволяє моделювати той факт, що базовий варіант використання може приєднувати до своєї поведінки деякі додаткові поведінки за рахунок розширення в варіанті іншому варіанті використання.

Наявність такого відношення завжди передбачає перевірку умови в точці розширення (extension point) в базовому варіанті використання. Точка розширення може мати деяке ім'я і зображена за допомогою примітки.

Особливості використання відношення розширення:

Один базовий варіант використання може мати кілька точок розширення, з кожною з яких повинен бути пов'язаний розширюючий варіант використання.

Один розширюючий варіант використання може бути пов'язаний відношенням розширення з декількома базовими варіантами використання.

Розширюючий варіант використання може, в свою чергу, мати власні розширюючі варіанти використання.

На одній діаграмі варіантів використання не може бути замкнутого шляху по відношенню розширення.

Сценарії використання

Діаграма варіантів використання надає знання про необхідну функціональність кінцевої системи в інтуїтивно-зрозумілому вигляді, однак не несе відомостей

про фактичний спосіб її реалізації. Конкретні варіанти використання можуть звучати занадто загально і розпливчасто і не є придатними для програмістів.

Для документації варіантів використання у вигляді деякої специфікації і для усунення неточностей та непорозумінь діаграм варіантів використання, як частина процесу збору та аналізу вимог складаються так звані сценарії використання.

Сценарії використання — це текстові представлення тих процесів, які відбуваються при взаємодії користувачів системи та самої системи. Вони є чітко формалізованими, покроковими інструкціями, що описують той чи інший процес у термінах кроків досягнення мети. Сценарії використання однозначно визначають кінцевий результат.

Сценарії використання описують варіанти використання природною мовою. Вони не мають загального, шаблонного виду написання, однак рекомендується наступний вигляд:

- Передумови — умови, які повинні бути виконані для виконання даного варіанту використання
- Постумови — що отримується в результаті виконання даного варіанту використання
- Взаємодіючі сторони
- Короткий опис
- Основний хід подій
- Винятки
- Примітки

Діаграми класів. Концептуальна модель системи

Діаграми класів використовуються при моделюванні ПС найбільш часто. Вони є однією з форм статичного опису системи з точки зору її проектування, показуючи її структуру. Діаграма класів не відображає динамічну поведінку об'єктів зображених на ній класів. На діаграмах класів показуються класи, інтерфейси і відношення між ними.

Представлення класів

Клас – це основний будівельний блок ПС. Це поняття присутнє і в ОО мовах програмування, тобто між класами UML і програмними класами є відповідність, яка є основою для автоматичної генерації програмних кодів або для виконання реінжинірингу. Кожен клас має назву, атрибути та операції. Клас на діаграмі показується у вигляді прямокутника, розділеного на 3 області. У верхній міститься назва класу, в середній – опис атрибутів (властивостей), в нижній – назви операцій – послуг, що надаються об'єктами цього класу.

Атрибути та операції класу

Атрибути класу визначають склад і структуру даних, що зберігаються в об'єктах цього класу. Кожен атрибут має ім'я і тип, що визначає, які дані він представляє. При реалізації об'єкта в програмному коді для атрибутів буде виділена пам'ять, необхідна для зберігання всіх атрибутів, і кожен атрибут матиме конкретне значення в будь-який момент часу роботи програми. Об'єктів одного класу в програмі може бути скільки завгодно багато, всі вони мають однаковий набір атрибутів, описаний у класі, але значення атрибутів у кожного об'єкта свої і можуть змінюватися в ході виконання програми.

Для кожного атрибута класу можна задати видимість (visibility). Ця характеристика показує, чи доступний атрибут для інших класів. В UML визначені наступні рівні видимості атрибутів:

- Відкритий (public) – атрибут видно для будь-якого іншого класу (об'єкта)
- Захищений (protected) – атрибут видно для нащадків даного класу
- Закритий (private) – атрибут не видно зовнішнім класам (об'єктам) і може використовуватися тільки об'єктом, що його містить

Останнє значення дозволяє реалізувати властивість інкапсуляції даних. Наприклад, оголосивши всі атрибути класу закритими, можна повністю приховати від зовнішнього світу його дані, гарантуючи відсутність несанкціонованого доступу до них. Це дозволяє скоротити число помилок у програмі. При цьому будь-які зміни в складі атрибутів класу ніяк не позначаються на решті частини ПС.

Відношення між класами

На діаграмах класів зазвичай показуються асоціації та узагальнення.

Кожна асоціація несе інформацію про зв'язки між об'єктами всередині ПС. Найбільш часто використовуються бінарні асоціації, що зв'язують два класи. Асоціація може мати назву, яка повинна виражати суть відображуваного зв'язку. Крім назви, асоціація може мати таку характеристику, як множинність. Вона показує, скільки об'єктів кожного класу може брати участь в асоціації. Множинність вказується у кожного кінця асоціації (полюса) і задається конкретним числом або діапазоном чисел. Множинність, вказана у вигляді зірочки, передбачає будь-яку кількість (в тому числі, і нуль).

Види відношень

Асоціація — найбільш загальний вид зв'язку між двома класами системи. Як правило, вона відображає використання одного класу іншим за допомогою деякої властивості або поля.

Узагальнення (наслідування) на діаграмах класів використовується, щоб показати зв'язок між класом-батьком і класом-нащадком. Воно вводиться на діаграму, коли виникає різновид якогось класу, а також у тих випадках, коли в

системі виявляються кілька класів, що володіють схожою поведінкою (в цьому випадку загальні елементи поведінки виносяться на більш високий рівень, утворюючи клас-батько).

Агрегацією позначається відношення "has-a", коли об'єкти одного класу входять в об'єкт іншого класу. Типовим прикладом такого відношення є списки об'єктів. У даному випадку список буде виступати агрегатом, а об'єкти, що входять у список, агрегованими елементами.

Композицією позначається відношення "owns-a". По своїй суті воно нагадує агрегацію, однак позначає більш тісний зв'язок між агрегатом і агрегованими елементами. Прикладом композиції може служити зв'язок між машиною і карбюратором: машина не буде функціонувати без карбюратора (тому відношення композиції). Список, в свою чергу, не втрачає своїх функцій без окремих елементів списку (тому відношення агрегації).

Логічна структура бази даних

Існують дві моделі бази даних — логічна та фізична. Фізична модель представляє набір двійкових даних у вигляді файлів, структурованих та згрупованих відповідно до призначення (сегменти, екстенти тощо), використовуючи їх для швидкого доступу до інформації та ефективного її зберігання. Логічна модель є структурою таблиць, уявлень, індексів та інших логічних елементів бази даних, що використовуються для програмування та роботи з базою.

Процес створення логічної моделі бази даних називається проектуванням бази даних. Проектування відбувається в тісному зв'язку з розробкою архітектури програмної системи, оскільки база створюється для зберігання даних, що надходять від програмних класів.

Є кілька підходів до зв'язування програмних класів із таблицями:

- Одна таблиця — один клас.
- Одна таблиця — кілька класів.
- Один клас — кілька таблиць.

Від вибору підходу залежить складність роботи з базою даних. Програмні класи представляють сутності проектованої системи, а таблиці — технічну реалізацію їх зберігання.

Нормальні форми

Нормальна форма — це властивість відношення в реляційній моделі даних, яка характеризує його з точки зору надлишковості, що може призвести до помилок у вибірках або змінах даних. Нормалізація — це процес приведення структури

бази даних до нормальних форм, що мінімізує логічну надлишковість та потенційні протиріччя.

Основні нормальні форми:

Перша нормальна форма (1НФ): кожен атрибут у відношенні містить тільки одне значення.

Друга нормальна форма (2НФ): кожен неключовий атрибут залежить від ключа функціонально повно.

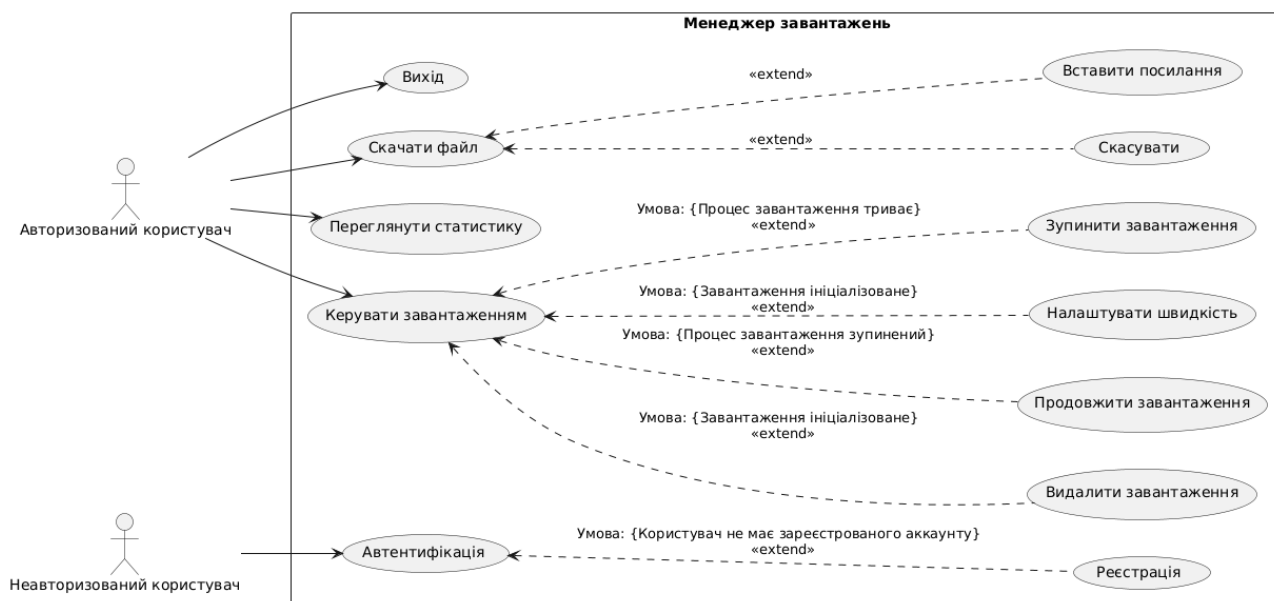
Третя нормальна форма (3НФ): немає транзитивних залежностей неключових атрибутів від ключа.

Нормальна форма Бойса-Кодда (BCNF): кожна функціональна залежність має в якості детермінанта потенційний ключ.

Нормалізація спрямована на виключення надлишковості та аномалій оновлення даних, забезпечуючи логічну чистоту моделі бази даних.

Хід роботи.

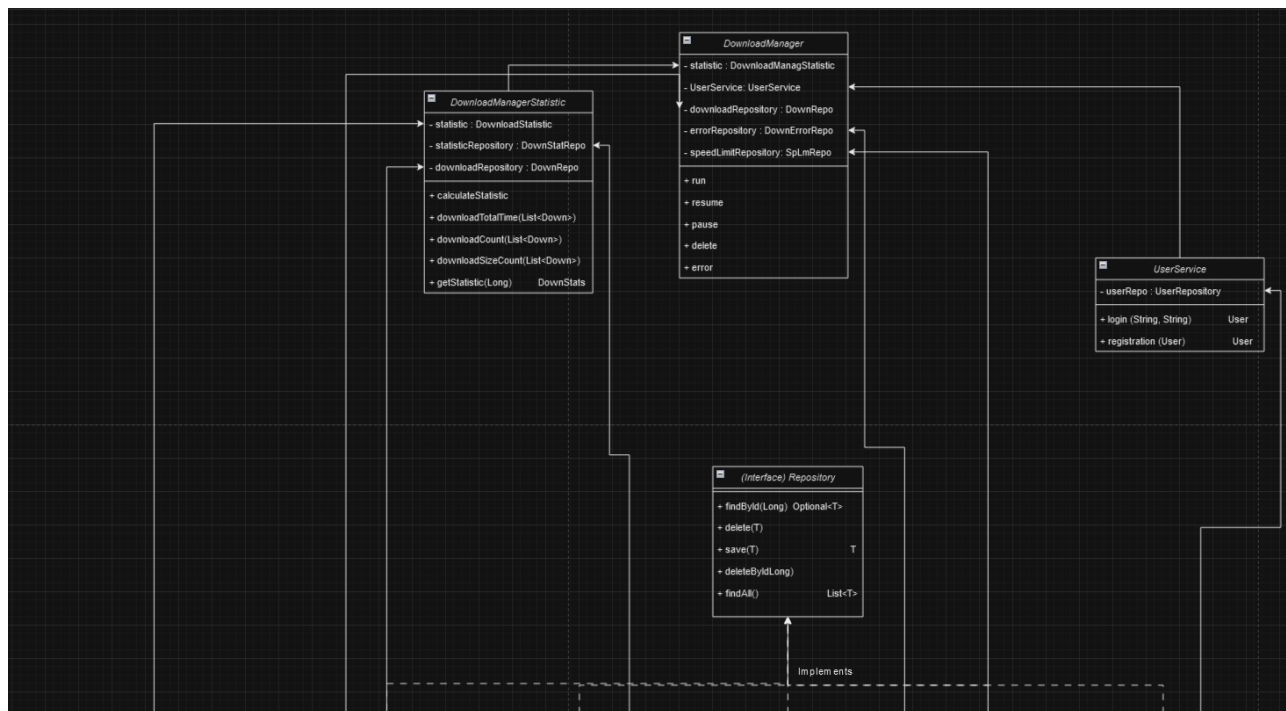
Завдання №2

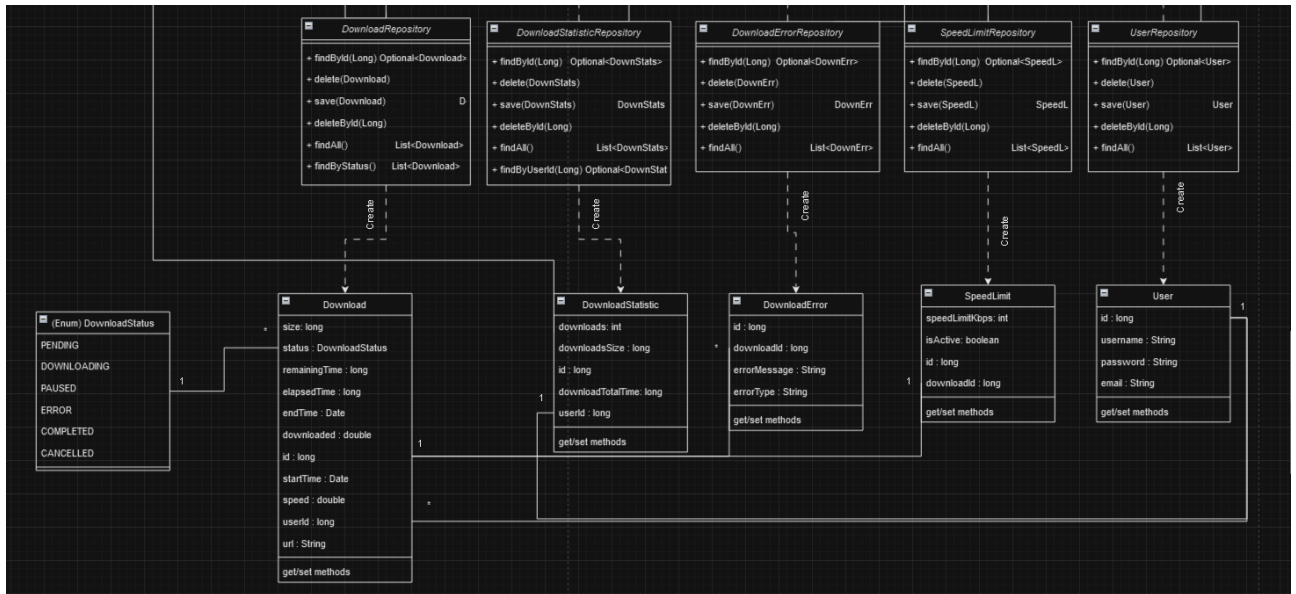


Завдання №3

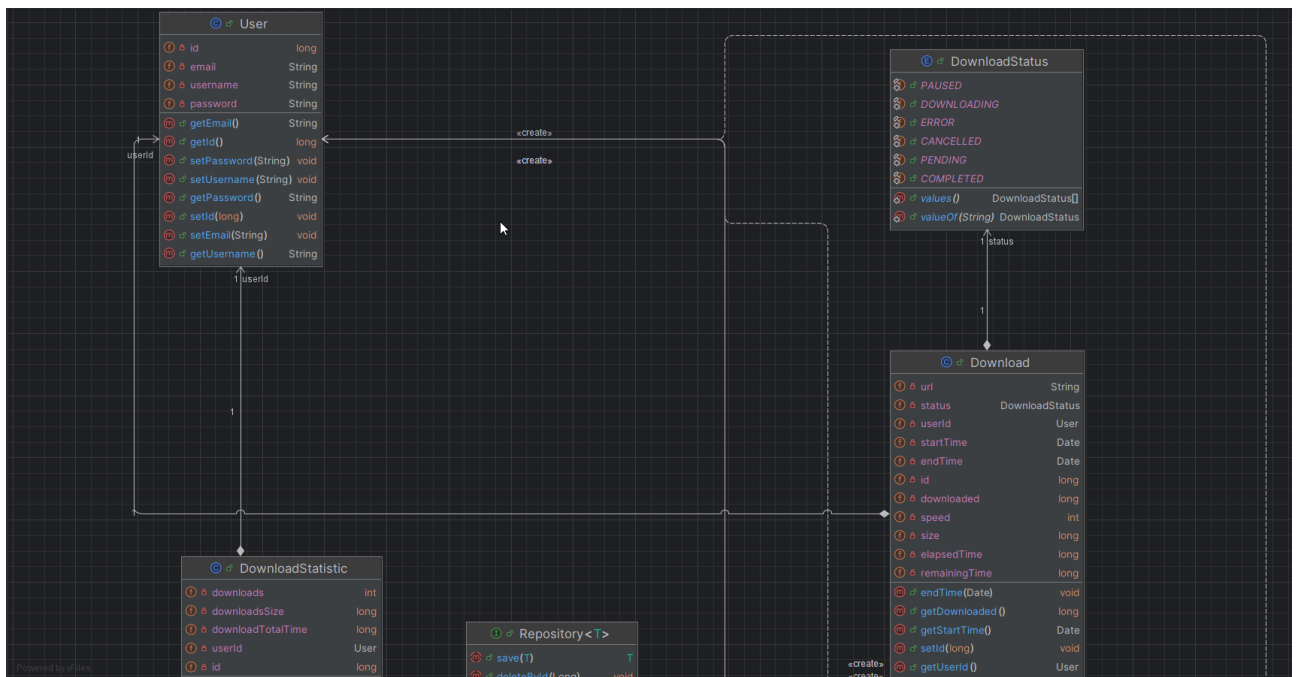
Діаграма класів для реалізованої частини системи (Фото схем з можливістю приближувати без втрати якості завантажено на гіт разом зі звітом під назвою downloadmanager.png та Діаграма класів.drawio.png)

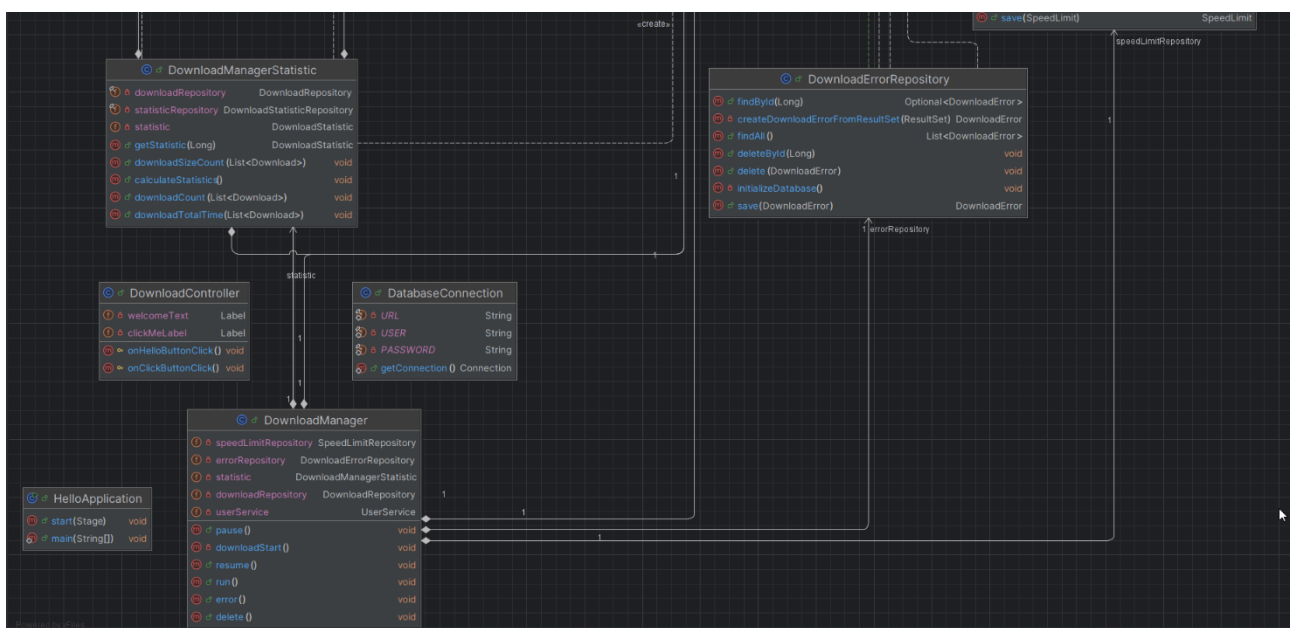
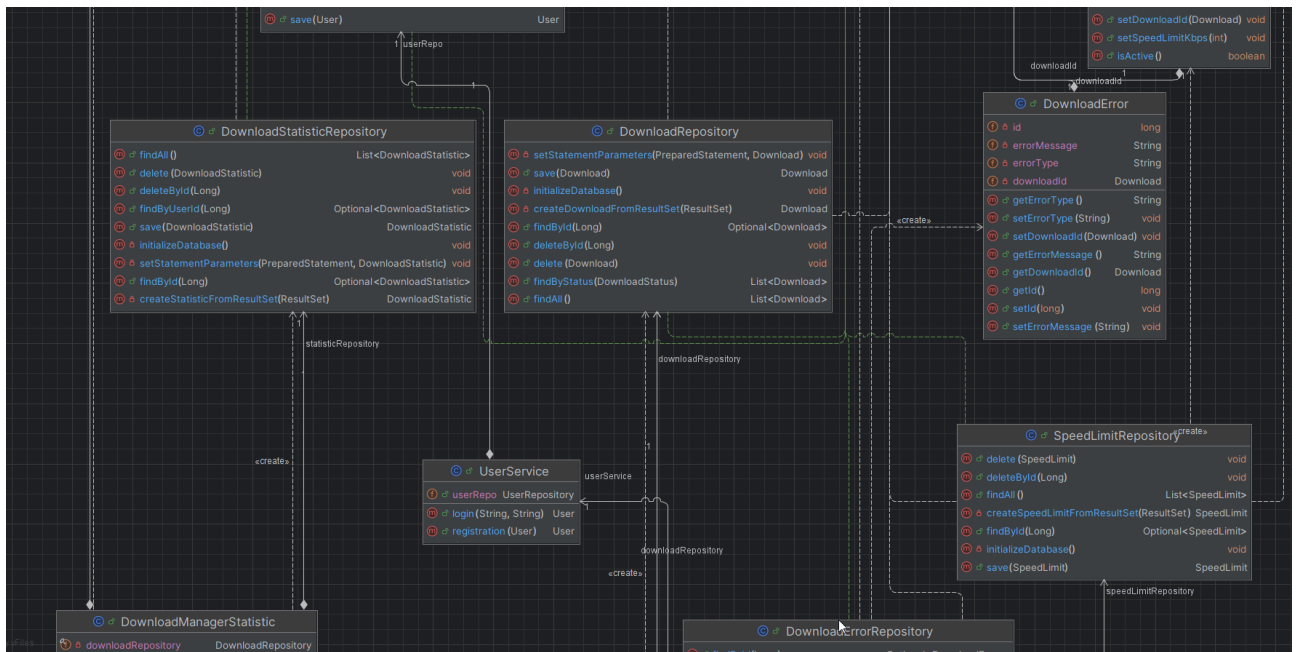
Draw.io (Вручну)





IntelliJ Idea:





Опис діаграми класів:

Основні Компоненти

1. Головні Класи

DownloadManager: Центральний клас, що керує процесом завантаження

DownloadController: Обробляє взаємодію з користувацьким інтерфейсом,

2. Моделі Даних

User: Представляє користувача з властивостями

Download: Представляє завантаження з властивостями

DownloadError: Представляє помилки які виникли із завантаженням

SpeedLimit: Інформація про обмеження

DownloadStatistic: Статистика менеджера

3. Обробка Статусів

DownloadStatus (Перелік):

- PAUSED (Призупинено)
- DOWNLOADING (Завантажується)
- ERROR (Помилка)
- CANCELLED (Скасовано)
- PENDING (Очікує)
- COMPLETED (Завершено)

4. Репозиторії

- UserRepository: Керує збереженням даних користувачів
- DownloadRepository: Керує записами завантажень
- DownloadStatisticRepository: Зберігає статистику завантажень
- DownloadErrorRepository: Обробляє логуювання помилок
- SpeedLimitRepository: Обмеженнями швидкості завантаження

Ключові Функції

- Реєстрація та вхід користувачів
- Керування Завантаженнями
- Запуск, пауза, відновлення та скасування завантажень
- Відстеження прогресу та швидкості завантаження
- Впровадження обмежень швидкості
- Відстеження Статистики
- Запис статистики завантажень через DownloadManagerStatistic
- Підрахунок загального часу та розміру завантажень

- Обробка Помилки
- Спеціальне відстеження помилок через клас DownloadError
- Зберігання типу та повідомлення помилки
- Інтеграція з Базою Даних
- Клас DatabaseConnection для обробки операцій з базою даних

Використовує URL, USER та PASSWORD для підключення

Зв'язки:

Клас DownloadManager діє як центральний координатор, підтримуючи зв'язки з:

- speedLimitRepository
- DownloadStatisticManager
- errorRepository
- downloadRepository
- userService

Наявні типи зв'язків:

- Композиція:

DownloadManager містить екземпляри всіх репозиторіїв

- Агрегація:

Download містить посилання на User і т.п.

- Асоціація:

UserService використовує UserRepository

- Залежність:

Усі репозиторії залежать від DatabaseConnection

Усі зв'язки відображені на діаграмі класів.

Шаблони проєктування

- Шаблон Репозиторію: Використовується для абстракції доступу до даних
- Сервісний Шар: Реалізований для бізнес-логіки (UserService)

Це діаграма класів реалізованої частини системи. У майбутньому ця діаграма може відрізнитися від класів які існуватимуть в проєкті.

Завдання №4

1 - Скачати файл

Передумови: Наявність посилання.

Постумови. У випадку існування посилання на файл відбувається скачування. У випадку помилки: користувач отримує відповідне повідомлення (Винятки).

Сторони взаємодії. Авторизований користувач, менеджер завантажень.

Короткий опис. Цей варіант використання описує ініціалізацію завантаження користувачем

Основний потік подій

Цей варіант використання відбувається тоді, коли користувач прагне почати завантаження.

1. Користувач натискає кнопку "Завантажити файл"
2. Система відображає діалогове вікно
3. Користувач вводить URL файлу
4. Система перевіряє URL та починає завантаження:
 - Створює запис у черзі завантажень
 - Починає передачу даних
5. У разі помилки у вигляді неправильності URL, або недоступності файлу (404) Виняток №1.
6. У разі помилки з боку сервера (500) Виняток №2

Винятки

Виняток №1

Відсутність файлів або неправильність посилання. Якщо посилання не існує, або немає чого скачувати система повідомить користувача про це. Користувач зможе знову почати завантаження з новим посиланням або скасувати від завантаження, при цьому виконання варіанту буде завершено.

Виняток №2

У разі серверної помилки (HTTP 5xx) система пропонує повторити спробу пізніше.

Примітки

Відсутні

2 - Переглянути статистику завантаження

Передумови.

Постумови. Користувач отримує інформацію про завантаження

Сторони взаємодії. Авторизований користувач, менеджер завантажень.

Короткий опис. Цей варіант використання описує вивід системою статистику завантаження користувача.

Основний потік подій

Цей варіант використання відбувається тоді, коли користувач прагне переглянути свою статистику завантажень.

1. Користувач натискає кнопку "Статистика"

2. Система збирає та відображає:

- Загальну кількість завантажень
- Об'єм завантажених даних
- Час, витрачений на завантаження
- Співвідношення успішних/невдалих завантажень

3. У випадку відсутності завантажень у користувача, Виняток №1

4. У випадку помилки доступу до бази даних Виняток №2.

Винятки

Виняток №1

Відсутність завантажень у користувача. Якщо користувач немає ніяких завантажень, вікно зі статистикою буде мати початкові значення. Система повідомить користувача про відсутність завантажень.

Виняток №2

У разі виникнення помилки доступу до бази даних, система повідомляє про технічні проблеми та пропонує спробувати пізніше.

Примітки

Відсутні

3 - Зупинити завантаження

Передумови. Користувач розпочав завантаження файлу. Процес завантаження активний та не завершений.

Постумови. Якщо завантаження триває: процес завантаження призупиняється, прогрес зберігається

Якщо виникла помилка завантаження: система логує помилку та інформує користувача

Якщо завантаження вже зупинено або завершено: стан завантаження не змінюється

Сторони взаємодії. Авторизований користувач, менеджер завантажень.

Короткий опис. Цей варіант використання описує зупинку завантаження користувачем

Основний потік подій

Цей варіант використання відбувається тоді, коли користувач вирішує зупинити активне завантаження.

1. Користувач вибирає активне завантаження зі списку
2. Користувач натискає кнопку "Зупинити"
3. Система:
 - Зберігає поточний прогрес завантаження
 - Звільняє мережеві ресурси
 - Оновлює статус завантаження на "Призупинено"
 - Оновлює інтерфейс користувача
4. Якщо завантаження не вибрано, або вони відсутні, Виняток №1:
5. Якщо виникла помилка при спробі зупинки Виняток №2:

Винятки

Виняток №1

Якщо завантаження не вибрано, або вони взагалі відсутні, кнопка «Зупинити» не активна.

Виняток №2

Якщо виникла будь-яка помилка під час спроби зупинки завантаження.

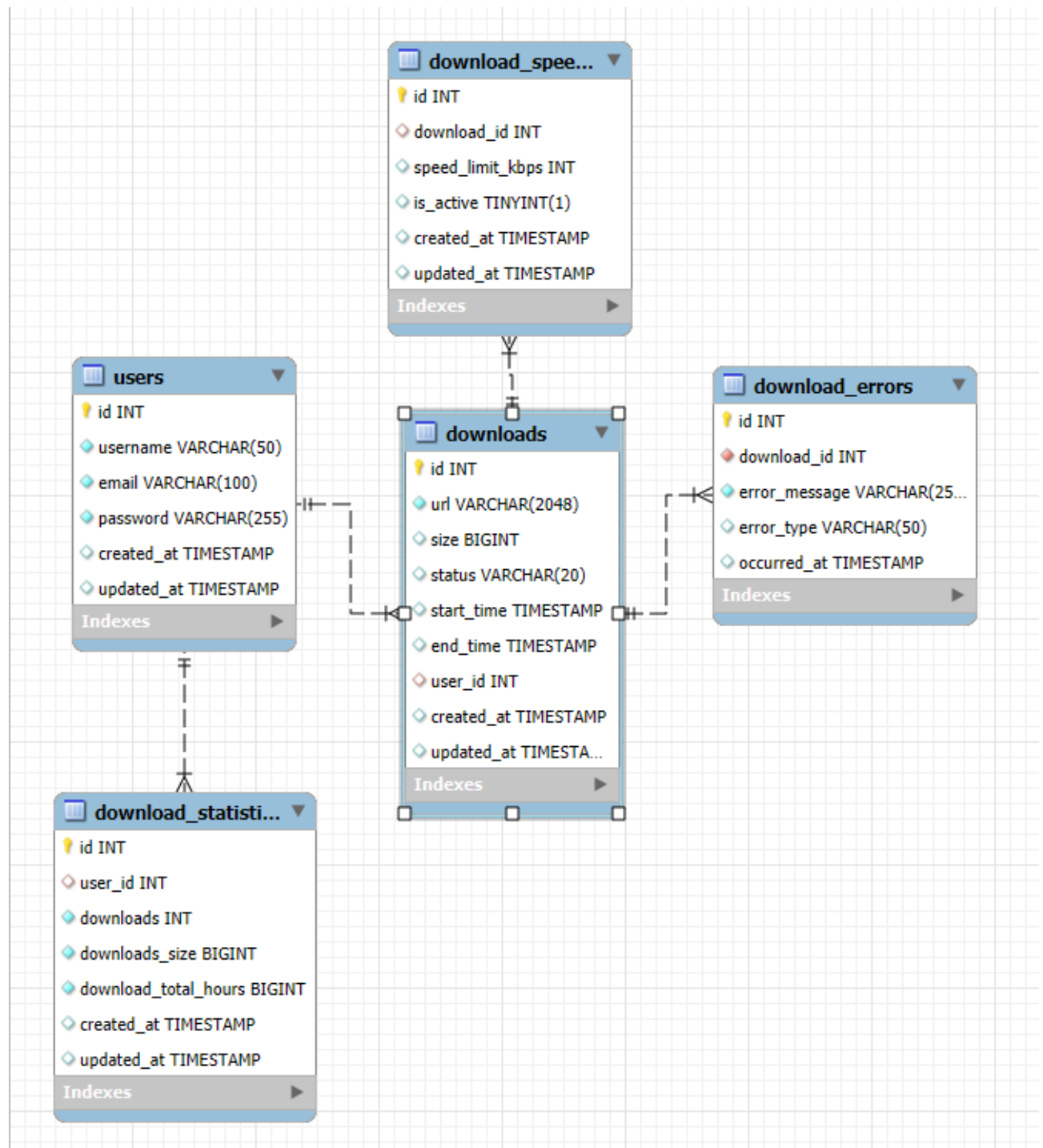
Наприклад: серверні помилки (HTTP 5xx), клієнтські помилки (HTTP 4xx), то система виводить лог про помилку, інформує користувача про проблему, намагається коректно завершити процес.

Примітки

Відсутні

Завдання №5

Основні класи та структура системи бази даних



Опис таблиць

Users – Таблиця із даними користувача

Downloads – коротка інформація про завантаження (Здебільшого створена для виводу історії завантажень)

Download_speed – Інформація про обмеження швидкості

Download_errors – Інформація про помилки

Download_statistic – Статистика користувача

Відношення:

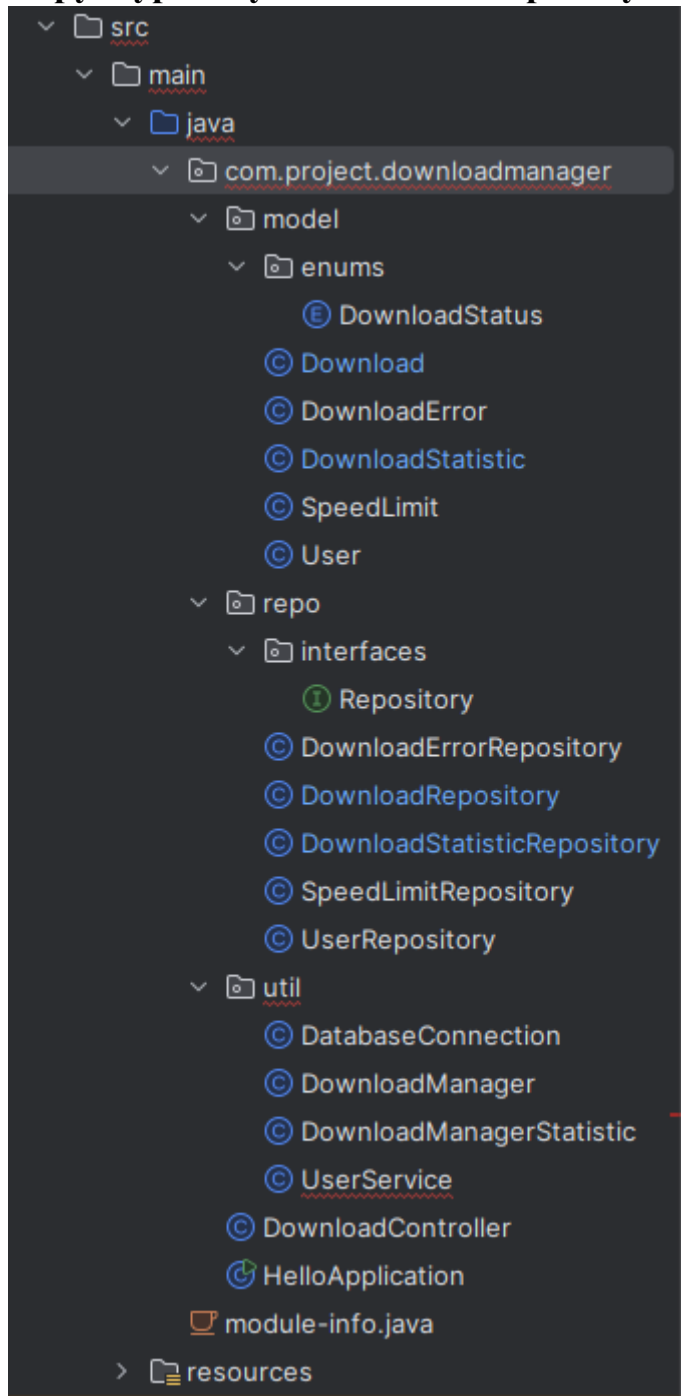
ONE-to-ONE – download_statistic to users (user_id у таблиці download_statistic унікальне поле і не може повторюватись)

MANY-to-ONE – downloads to users

ONE-to-ONE – download_speed to downloads (user_id у таблиці download_statistic унікальне поле і не може повторюватись)

MANY-to-ONE – download_errors to downloads

Структура існуючої частини проекту



Вихідні коди класів реалізованої частини системи наведені у гіт-репозиторії у папці DownloadManager.

Висновки.

Отже на цій лабораторній роботі, було розроблено та проаналізовано концептуальну модель системи менеджера завантажень з використанням UML-діаграм. Зокрема створення діаграми варіантів використання. Розробка детальних сценаріїв використання. Проектування діаграми класів Створення концептуальної моделі бази даних.