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

Лабораторна робота №2

Технології розроблення програмного забезпечення

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ. СЦЕНАРІЇ ВАРІАНТІВ ВИКОРИСТАННЯ. ДІАГРАМИ UML. ДІАГРАМИ КЛАСІВ. КОНЦЕПТУАЛЬНА МОДЕЛЬ СИСТЕМИ

E-mail клієнт

Виконав
студент групи ІА–22:
Артамонов Дмитро
Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

Тема: Діаграма варіантів використання. Сценарії варіантів використання. Діаграми UML. Діаграми класів. Концептуальна модель системи

Мета: Проаналізувати тему, намалювати схему прецеденту, діаграму класів, розробити основні класи і структуру бази

Хід роботи

..15 E-mail клієнт (singleton, builder, decorator, template method, interpreter, SOA)

Поштовий клієнт повинен нагадувати функціонал поштових програм Mozilla Thunderbird, The Bat і т.д. Він повинен сприймати і коректно обробляти

---

pop3/smtp/imap протоколи, мати функції автонастройки основних поштових провайдерів для України (gmail, ukr.net, i.ua), розділяти повідомлення на папки/категорії/важливість, зберігати чернетки незавершених повідомлень, прикріплювати і обробляти прикріплені файли.

Теоретичні відомості

У 2 лабораторній роботі розглядається використання діаграм UML для моделювання різних аспектів системи. Ключовими елементами є діаграми варіантів використання (Use Case Diagrams), які допомагають визначити основні сценарії взаємодії користувачів із системою.

1. UML (Unified Modeling Language) – це універсальна мова візуального моделювання, призначена для опису структури і поведінки програмних систем. Вона дозволяє створювати діаграми для різних аспектів системи, забезпечуючи чітке та наочне уявлення про її роботу. UML використовується на всіх етапах розробки: від аналізу вимог до тестування.
2. Діаграми варіантів використання (Use Case Diagrams) – це інструмент для моделювання функціональності системи з точки зору користувача. Вони відображають, як зовнішні актори (користувачі, інші системи) взаємодіють із системою для досягнення своїх цілей. Основні елементи діаграми: актори, варіанти використання (use cases) і зв'язки між ними.
3. Діаграма класів (Class Diagrams) – це інший важливий інструмент UML, що моделює структуру системи на рівні класів і їх взаємодій. Вона відображає класи, їх атрибути, методи, а також зв'язки між ними (асоціації, агрегації, композиції). Ця діаграма є основою для реалізації об'єктно-орієнтованого програмування, оскільки дозволяє чітко визначити структуру об'єктів у системі.
4. Концептуальна модель системи – це початковий етап побудови моделі, який допомагає визначити ключові компоненти системи, їх взаємодії та ролі. Вона може містити як діаграми варіантів використання, так і діаграми класів, забезпечуючи базу для подальшого аналізу та реалізації.
5. Репозиторій– це шаблон проєктування, який забезпечує абстракцію над операціями з базою даних. Він дозволяє взаємодіяти з базою даних через інтерфейси, що спрощує обслуговування та тестування коду. Репозиторій приховує деталі доступу до даних, що робить код більш організованим і підтримуваним.

Побудуємо схему прецендентів. Схема зображена на рисунку 1.

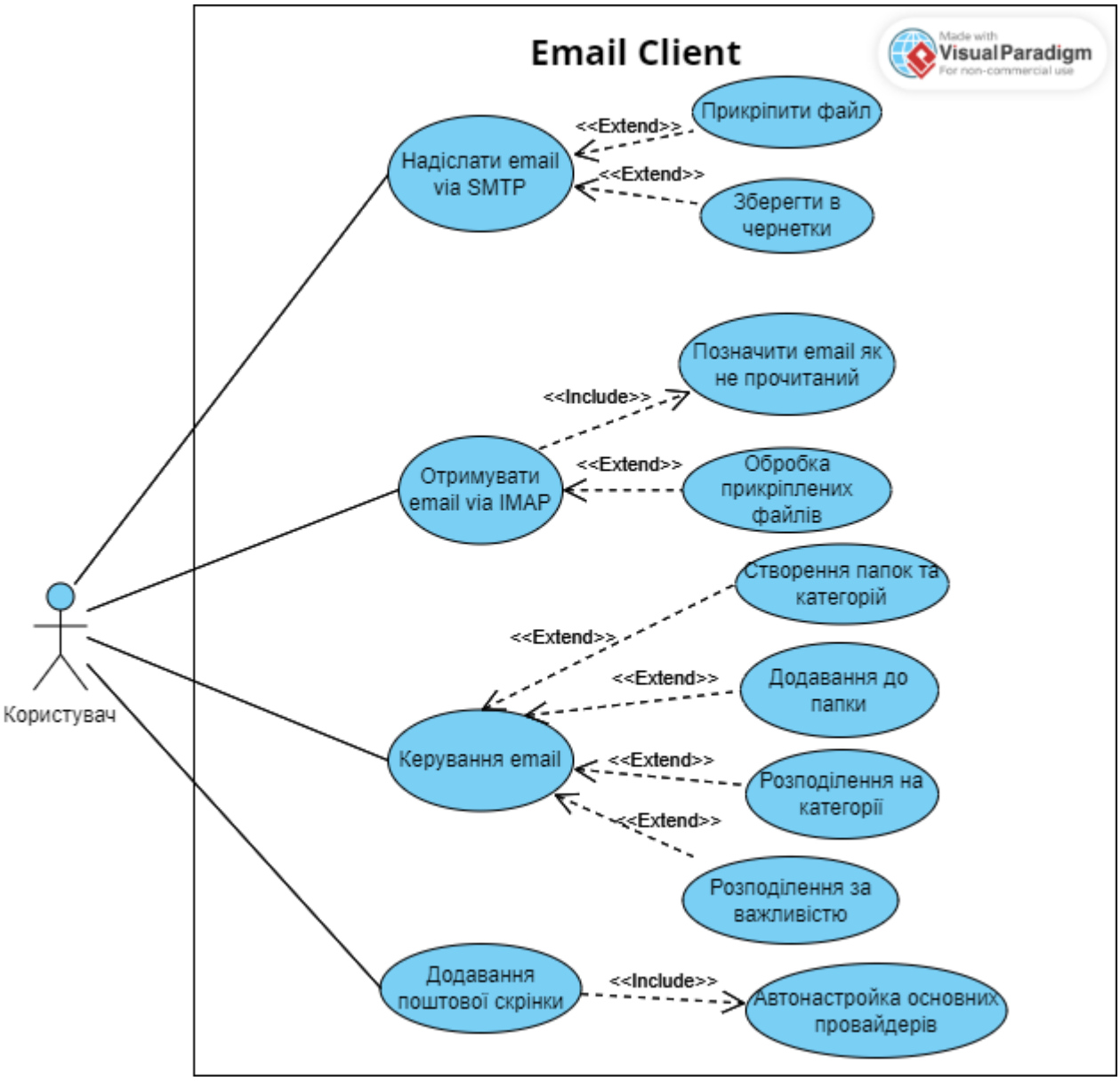

Рисунок 1. – Схема прецендентy

Оберемо 3 прецеденти і напишемо для них сценарії використання
Сценарій використання 1: Відправка email

Актори: Користувач.

Опис: Користувач з поштового клієнта надсилає електроного листа.

Основний хід подій:

1. Користувач за необхідності додає поштову скриньку.
2. Користувач натискає кнопку «Створити email».
3. Вказує адресу отримувача та поштову скриньку з якої буде відправлено email.
4. Користувач вказує Тему листа та інші корисні дані.
5. Система валідує введені дані та передає листа на поштовий сервер.
6. Надісланий лист поміщається в папку «Відправлені».

Виключення: Якщо введенні дані некоректні(наприклад, не вказано адресу отримувача чи відправника, не заповнено поле «Тема») система сповістить про помилку.

Сценарій використання 2: Додавання email до папки
Актори: Користувач.
Опис: Цей сценарій описує процес додавання отриманого email до папки
Основний хід подій:

1. Користувач обирає опцію "Всі листи".
2. Обирає листи якими він хоче керувати.
3. За потреби створює папку.
4. Обирає потрібну папку і додає листи.

Виключення: Якщо поштовий клієнт не може отримати доступ до списку папок (наприклад, через помилку доступу до сервера або відсутність підключення до інтернету), система повідомляє про це користувача.

Сценарій використання 3: Додавання поштової скриньки

Актори: Користувач.

Опис: Цей сценарій описує процес додавання нової поштової скриньки в поштовий клієнт.

Основний хід подій:

1. Користувач відкриває налаштування поштового клієнта і обирає "Додати поштову скриньку".
2. Вводить свою електронну адресу.
3. Поштовий клієнт автоматично налаштовує поштові скриньки для одного з провайдерів (gmail, ukr.net, i.ua).
4. Користувач вводить пароль і підтверджує додавання.
5. Поштовий клієнт перевіряє правильність підключення.

Виключення: При неправильному введені облікових даних система повідомить про це користувача.

Побудуємо структуру бази даних (рисунок 2)

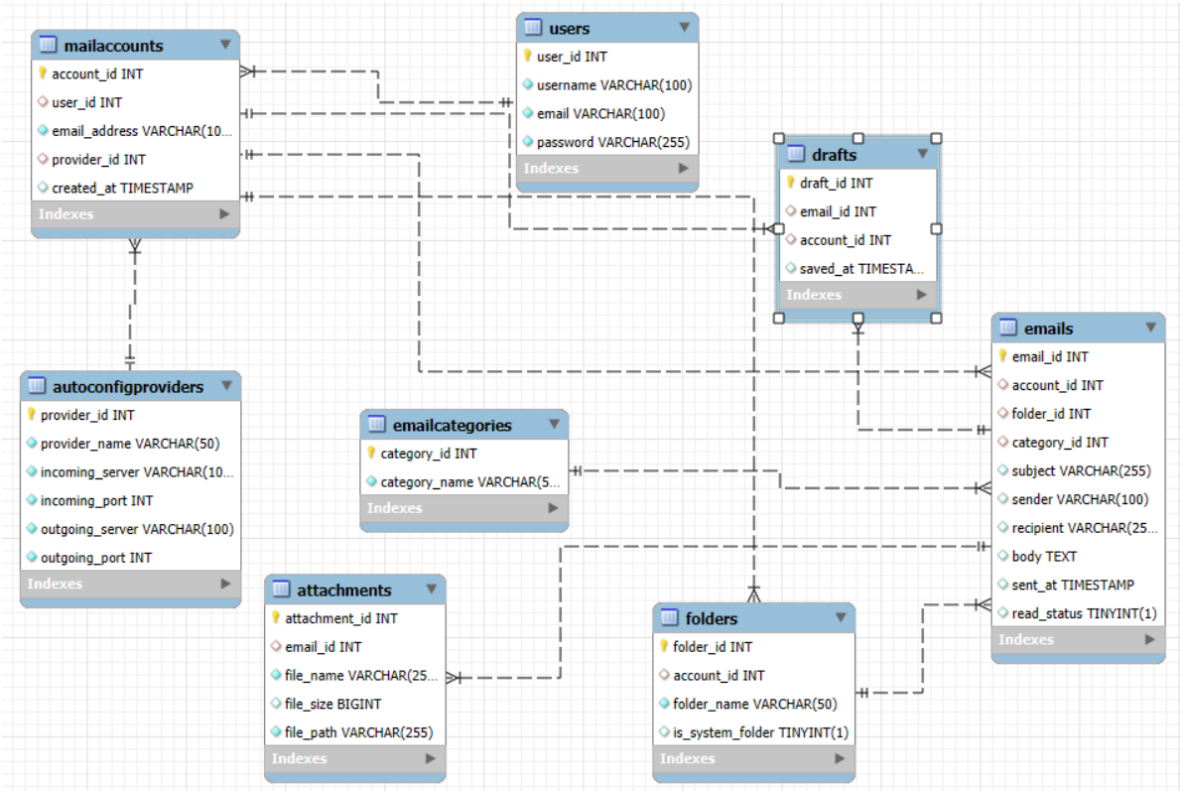

Рисунок 2. Структура бази даних

Ця діаграма представляє модель бази даних для системи управління поштовими акаунтами та електронними листами. Нижче наведено опис основних таблиць і зв'язків між ними:

1. users:

- Таблиця містить інформацію про користувачів системи.
- Поля: `user_id` (ідентифікатор користувача), `username`, `email`, `password`.

2. mailaccounts:

- Таблиця для зберігання інформації про поштові акаунти користувачів.

- Поля: `account_id`, `user_id` (посилання на користувача), `email_address`, `provider_id` (посилання на таблицю поштових провайдерів), `created_at` (дата створення акаунта).

3. autoconfigproviders:

- Зберігає дані про поштових провайдерів для автоматичної конфігурації.
- Поля: `provider_id`, `provider_name`, `incoming_server`, `incoming_port`, `outgoing_server`, `outgoing_port`.

4. emails:

- Основна таблиця для зберігання електронних листів.
- Поля: `email_id`, `account_id` (посилання на поштовий акаунт), `folder_id` (посилання на папку), `category_id` (посилання на категорію), `subject`, `sender`, `recipient`, `body`, `sent_at` (дата відправлення), `read_status` (статус прочитання).

5. folders:

- Зберігає інформацію про папки в поштових акаунтах.
- Поля: `folder_id`, `account_id` (посилання на акаунт), `folder_name` (назва папки), `is_system_folder` (вказує, чи це системна папка).

6. emailcategories:

- Зберігає категорії для електронних листів.
- Поля: `category_id`, `category_name`.

7. attachments:

- Таблиця для зберігання вкладень, прикріплених до електронних листів.
- Поля: `attachment_id`, `email_id` (посилання на електронний лист), `file_name`, `file_size`, `file_path`.

8. drafts:

- Зберігає інформацію про чернетки листів.
- Поля: `draft_id`, `email_id` (посилання на електронний лист), `account_id` (посилання на поштовий акаунт), `saved_at` (дата збереження чернетки).

Зв'язки між таблицями:

- Користувач може мати кілька поштових акаунтів.
- Поштовий акаунт може містити декілька електронних листів, папок, чернеток і вкладень.
- Кожен лист може бути прив'язаний до папки, категорії та мати кілька вкладень.

Побудуємо діаграму класів (рисунок 3, рисунок 4)

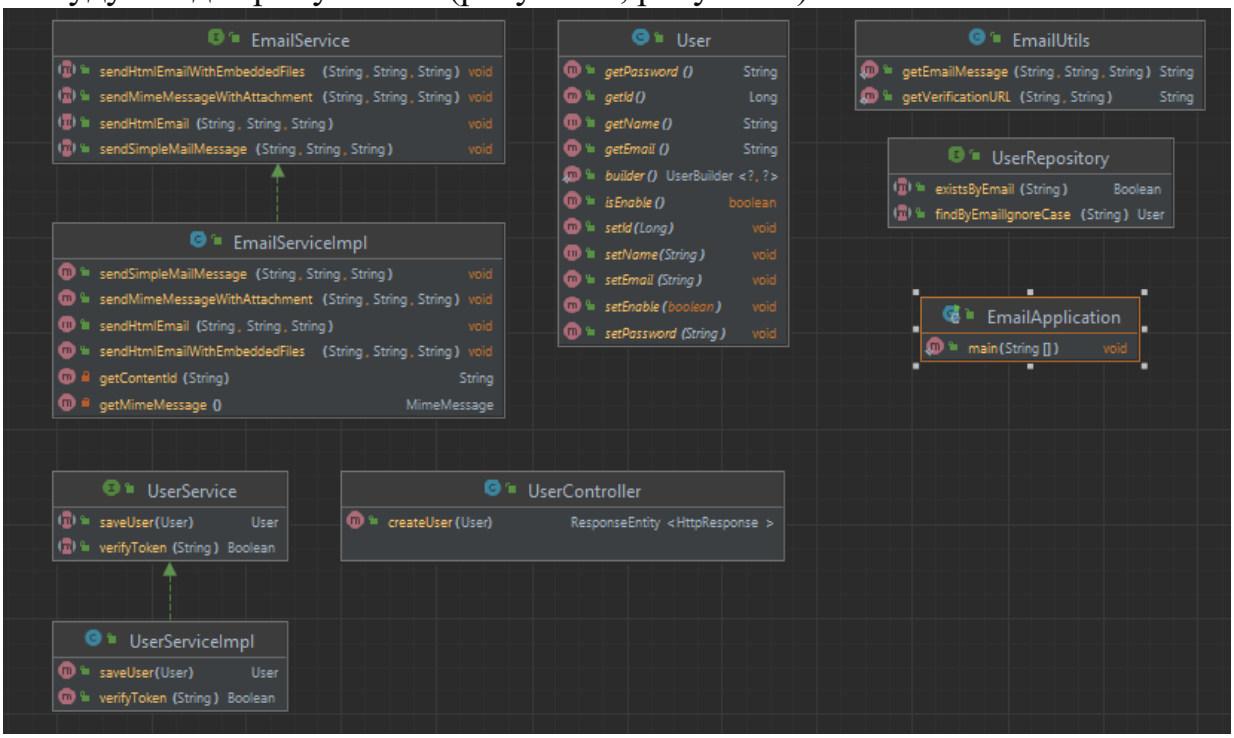

Рисунок 3. Діаграма реалізоаних класів

Ця діаграма класів представляє структуру Java-додатка для управління користувачами та відправкою електронних листів. Нижче наведено опис основних класів і їх методів:

1. EmailService (інтерфейс):

Визначає методи для відправки електронних листів.

Методи:

- `sendHtmlEmailWithEmbeddedFiles(String, String, String)`: відправляє HTML-лист із вбудованими файлами.

- `sendMimeMessageWithAttachment(String, String, String)`: відправляє лист із вкладенням.

- `sendHtmlEmail(String, String, String)`: відправляє HTML-лист.

- `sendSimpleMailMessage(String, String, String)`: відправляє простий текстовий лист.

2. EmailServiceImpl (реалізація інтерфейсу EmailService):

Реалізує всі методи, визначені в EmailService:

- `sendSimpleMailMessage(String, String, String)`: реалізація для простого текстового листа.

- `sendMimeMessageWithAttachment(String, String, String)`: реалізація для листа з вкладенням.

- `sendHtmlEmail(String, String, String)`: реалізація для HTML-листа.

- `sendHtmlEmailWithEmbeddedFiles(String, String, String)`: реалізація для HTML-листа з вбудованими файлами.

Додаткові методи:

- `getContentId(String)`: повертає ідентифікатор контенту.

- `getMimeMessage()`: повертає об'єкт MimeMessage.

3. User:

Клас представляє користувача системи.

Методи:

- `getPassword()`: повертає пароль.

- `getId()`: повертає ідентифікатор користувача.

- `getName()`: повертає ім'я користувача.

- `getEmail()`: повертає електронну пошту.

- `isEnable()`: повертає статус активності користувача.

- `setId(Long)`, `setName(String)`, `setEmail(String)`, `setEnable(boolean)`, `setPassword(String)`: методи для встановлення значень відповідних полів.

4. UserService (інтерфейс):

Описує функціональність для роботи з користувачами.

Методи:

- `saveUser(User)`: зберігає користувача.

- `verifyToken(String)`: перевіряє токен.

5. UserServiceImpl (реалізація UserService):

Реалізує методи з UserService:

- `saveUser(User)`: зберігає користувача.

- `verifyToken(String)`: перевіряє валідність токена.

6. UserController:

Клас для обробки запитів щодо користувачів.

Метод:

- `createUser(User)`: створює нового користувача та повертає HTTP-відповідь.

7. EmailUtils:

Клас із допоміжними методами для роботи з електронною поштою.

Методи:

- `getEmailMessage(String, String, String)`: формує повідомлення електронної пошти.

- `getVerificationURL(String, String)`: генерує URL для верифікації користувача.

8. UserRepository:

Клас для роботи з базою даних користувачів.

Методи:

- `existsByEmail(String)`: перевіряє наявність користувача за електронною адресою.

- `findByEmailIgnoreCase(String)`: знаходить користувача за електронною адресою, не враховуючи регістр.

9. EmailApplication:

Основний клас для запуску програми.

Методи:

- `main(String[])`: точка входу для запуску додатка.

Ця діаграма класів відображає структуру додатка для обробки користувачів та відправки електронних листів, з різними сервісами, утилітами та контролерами.

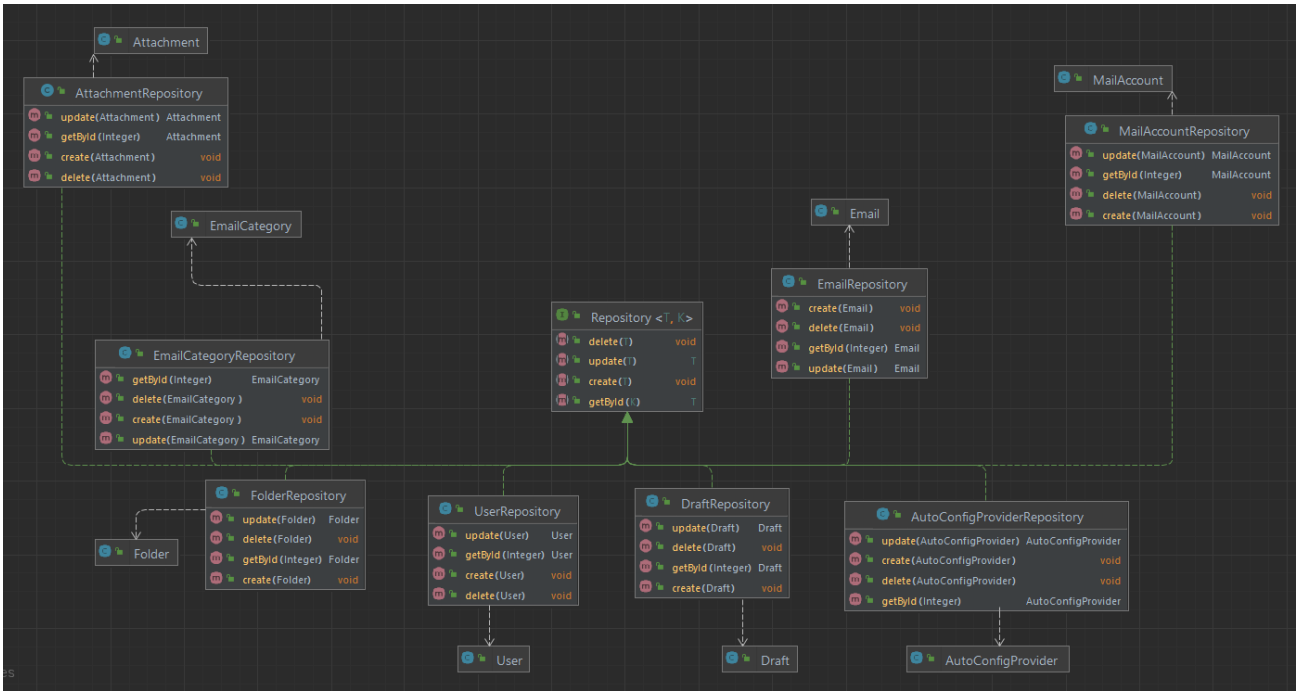

Рисунок 4. Класи даних, які реалізовують шаблон Репозиторію

На діаграмі класів зображені кілька репозиторіїв і відповідних їм сутностей, які використовуються для роботи з різними об'єктами в системі електронної пошти. Ось короткий опис основних елементів:

1. Attachment (Додаток):

Використовує `AttachmentRepository`, який дозволяє виконувати операції CRUD (Create, Read, Update, Delete) з додатками:

- `update(Attachment)`
- `getById(Integer)`
- `create(Attachment)`

- `delete(Attachment)`

2. EmailCategory (Категорія електронної пошти):

Використовує `EmailCategoryRepository` для роботи з категоріями електронних листів:

- `update(EmailCategory)`
- `getById(Integer)`
- `create(EmailCategory)`
- `delete(EmailCategory)`

3. Folder (Папка):

Використовує `FolderRepository` для роботи з папками:

- `update(Folder)`
- `getById(Integer)`
- `create(Folder)`
- `delete(Folder)`

4. Email (Електронна пошта):

Використовує `EmailRepository` для управління електронними листами:

- `create(Email)`
- `delete(Email)`

- `getById(Integer)`

- `update(Email)`

5. User (Користувач):

Використовує `UserRepository` керує користувачами системи:

- `update(User)`

- `getById(Integer)`

- `create(User)`

- `delete(User)`

6. Draft (Чернетка):

Використовує `DraftRepository` відповідає за чернетки:

- `update(Draft)`

- `getById(Integer)`

- `create(Draft)`

- `delete(Draft)`

7. MailAccount (Обліковий запис електронної пошти):

Використовує `MailAccountRepository` для облікових записів електронної пошти:

- `update(MailAccount)`

- `getById(Integer)`

- `create(MailAccount)`

- `delete(MailAccount)`

8. AutoConfigProvider (Автоконфігурація):

Використовує `AutoConfigProviderRepository` для роботи з провайдерами автоконфігурації:

- `update(AutoConfigProvider)`

- `getById(Integer)`

- `create(AutoConfigProvider)`

- `delete(AutoConfigProvider)`

9. Repository<T, K>:

Загальний інтерфейс репозиторію з основними методами для роботи з будь-якими сутностями:

- `delete(T)`

- `update(T)`

- `create(T)`

- `getById(K)`

Зв'язки між класами показують, що кожен репозиторій відповідає за певну сутність і забезпечує CRUD-операції.

Посилання на GitHub: https://github.com/qjsoq/trpz

Висновок: В даній лабораторній роботі я проаналізував тему, намалював схему прецедентів, діаграму класів, розробив основні класи і структуру бази даних.