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

Лабораторна робота №2
«Технології розроблення програмного забезпечення»
Тема: « E-mail клієнт»

Виконав:
Студент групи ІА-34
Марченко А.О.
Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Основи проектування.

Мета: Обрати зручну систему побудови UML-діаграм та навчитися будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.

2.1 Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Проаналізувати тему та спроєктувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.

• Спроєктувати діаграму класів предметної області.

• Вибрати 3 варіанти використання та написати за ними сценарії використання.

• На основі спроєктованої діаграми класів предметної області розробити основні класи та структуру бази даних системи. Класи даних повинні реалізувати шаблон Repository для взаємодії з базою даних.

• Нарисувати діаграму класів для реалізованої частини системи.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму варіантів використання відповідно, діаграму класів системи, вихідні коди класів системи, а також зображення структури бази даних.

2.2 Тереотичні відомості

2.2.1 Вступ до мови UML

Мова UML є загальноцільовою мовою візуального моделювання, яка розроблена для специфікації, візуалізації, проєктування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем [3]. Мова UML є досить строгим та потужним засобом моделювання, який може бути ефективно використаний для побудови концептуальних, логічних та графічних моделей складних систем різного цільового призначення. Ця мова увібрала в себе найкращі якості та досвід методів програмної інженерії, які з успіхом використовувалися протягом останніх років при моделюванні великих та складних систем.

З погляду методології ООАП (об'єктно-орієнтованого аналізу та проєктування) досить повна модель складної системи є певною кількістю взаємопов'язаних уявлень (views), кожне з яких відображає аспект поведінки або структури системи. У цьому найзагальнішими уявленнями складної системи прийнято вважати статичне і динамічне, які у своє чергу можуть поділятися інші більш приватні.

Принцип ієрархічної побудови моделей складних систем передбачає розгляд процес побудови моделей на різних рівнях абстрагування або деталізації в рамках фіксованих уявлень.

Рівень представлення (layer) – спосіб організації та розгляду моделі на одному рівні абстракції, що представляє горизонтальний зріз архітектури моделі, тоді як розбиття представляє її вертикальний зріз. При цьому вихідна або початкова модель складної системи має найбільш загальне уявлення та відноситься до концептуального рівня. Така модель, що отримала назву концептуальної, будується на початковому етапі проєктування і може не містити багатьох деталей та аспектів системи, що моделюється. Наступні моделі конкретизують концептуальну модель, доповнюючи її уявленнями логічного та фізичного рівня.

Загалом процес ООАП можна розглядати як послідовний перехід від розробки найбільш загальних моделей та уявлень концептуального рівня до більш приватних і детальних уявлень логічного та фізичного рівня. У цьому кожному етапі ООАП дані моделі послідовно доповнюються дедалі більше деталей, що дозволяє їм адекватно відбивати різні аспекти конкретної реалізації складної системи.

В рамках мови UML уявлення про модель складної системи фіксуються у вигляді спеціальних графічних конструкцій, що отримали назву діаграм.

Діаграма (diagram) – графічне уявлення сукупності елементів моделі у формі зв'язкового графа, вершинам і ребрам (дугам) якого приписується певна семантика. Нотація канонічних діаграм є основним засобом розробки моделей мовою UML.

У нотації мови UML визначено такі види 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).

Перелічені діаграми є невід'ємною частиною графічної нотації мови UML. Понад те, 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 потрібна для реалізації проєкту складної системи.

Кожна з цих діаграм деталізує та конкретизує різні уявлення про модель складної системи у термінах мови UML. При цьому діаграма варіантів використання являє собою найбільш загальну концептуальну модель складної системи, яка є вихідною для побудови інших діаграм. Діаграма класів, за своєю суттю, логічна модель, що відбиває статичні аспекти структурної побудови складної системи.

Діаграми кооперації та послідовностей є різновидами логічної моделі, які відображають динамічні аспекти функціонування складної системи. Діаграми станів та діяльності призначені для моделювання поведінки системи. І, нарешті, діаграми компонентів і розгортання служать уявлення фізичних компонентів складної системи і тому представляють її фізичну модель.

2.2.2. Діаграма варіантів використання (Use-Cases Diagram)

Діаграма варіантів використання (Use-Cases Diagram) – це UML діаграма за допомогою якої у графічному вигляді можна зобразити вимоги до системи, що розробляється [3]. Діаграма варіантів використання – це вихідна

концептуальна модель проєктованої системи, вона не описує внутрішню побудову системи.

Діаграми варіантів використання призначені для:

- визначення загальної межі функціональності проєктованої системи;
- формулювання загальних вимоги до функціональної поведінки проєктованої системи;
- подальшої розробка вихідної концептуальної моделі системи (діаграми класів);
- створення основи для виконання аналізу, проєктування, розробки та тестування.

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

Діаграма варіантів використання складається з низки елементів. Основними елементами є: варіанти використання або прецеденти (use case), актор або дійова особа (actor) та відносини між акторами та варіантами використання (relationship).

2.2.3. Діаграми класів

Діаграми класів використовуються при моделюванні програмних систем найчастіше. Вони є однією із форм статичного опису системи з погляду її проєктування, показуючи її структуру [3]. Діаграма класів не відображає динамічної поведінки об'єктів зображених на ній класів. На діаграмах класів показуються класи, інтерфейси та відносини між ними.

Клас – це основний будівельний блок програмної системи. Це поняття є і в мовах програмування, тобто між класами UML та програмними класами є

відповідність, що є основою для автоматичної генерації програмних кодів або для виконання реінжинірингу. Кожен клас має назву, атрибути та операції. Клас на діаграмі показується як прямокутник, розділений на 3 області. У верхній міститься назва класу, у середній – опис атрибутів (властивостей), у нижній – назви операцій – послуг, що надаються об'єктами цього класу.

Діаграми класів створюються при логічному моделюванні програмних систем та служать для наступних цілей:

• Для моделювання даних. Аналіз предметної області дозволяє виявити основні характерні нею сутності та зв'язку з-поміж них. Це зручно моделюється з допомогою діаграм класів. Ці діаграми є основою побудови концептуальної моделі.

• Для представлення архітектури програмної системи. Можна виділити архітектурно значимі класи і відобразити їх на діаграмах, що описують архітектуру програмної системи.

• Для моделювання навігації екранів. На таких діаграмах показуються прикордонні класи та їхній логічний взаємозв'язок. Інформаційні поля моделюються як атрибути класів, а кнопки, що управляють, – як операції та відносини.

• Для моделювання логіки програмних компонентів.

• Для моделювання логіки обробки даних.

2.2.4 Логічна структура бази даних

Розрізняють дві моделі бази даних – логічну та фізичну. Фізична модель бази даних представляє собор набір бінарних даних у вигляді файлів, структурованих та згрупованих згідно з призначенням (сегменти, екстенти та ін.), що використовується для швидкого та ефективного отримання інформації з жорсткого диска, а також для компактного зберігання та розміщення даних на жорсткому диску.

Логічна модель бази даних є структурою таблиць, уявлень, індексів та інших логічних елементів бази даних, що дозволяють власне програмування та використання бази даних.

Процес створення логічної моделі бази даних зветься проєктування бази даних (database design). Проєктування відбувається у зв'язку з опрацюванням архітектури програмної системи, оскільки база даних створюється зберігання даних, одержуваних з програмних класів.

Відповідно можна розрізнити кілька підходів до зв'язування програмних класів та таблиць: одна таблиця – один клас, одна таблиця – кілька класів, один клас – кілька таблиць. Залежно від обраного підходу, буде ускладнюватись (і уповільнюватися) робота з базою даних при додаванні даних, або отримання даних з БД та парсинг їх у відповідні класи.

З іншого боку, якщо програмні класи є сутності проєктованої системи (елементи предметної області), то таблиці відображають їх технічну реалізацію та спосіб зберігання та зв'язку.

Основним керівництвом під час проєктування таблиць є т. зв. нормальні форми баз даних.

Хід роботи

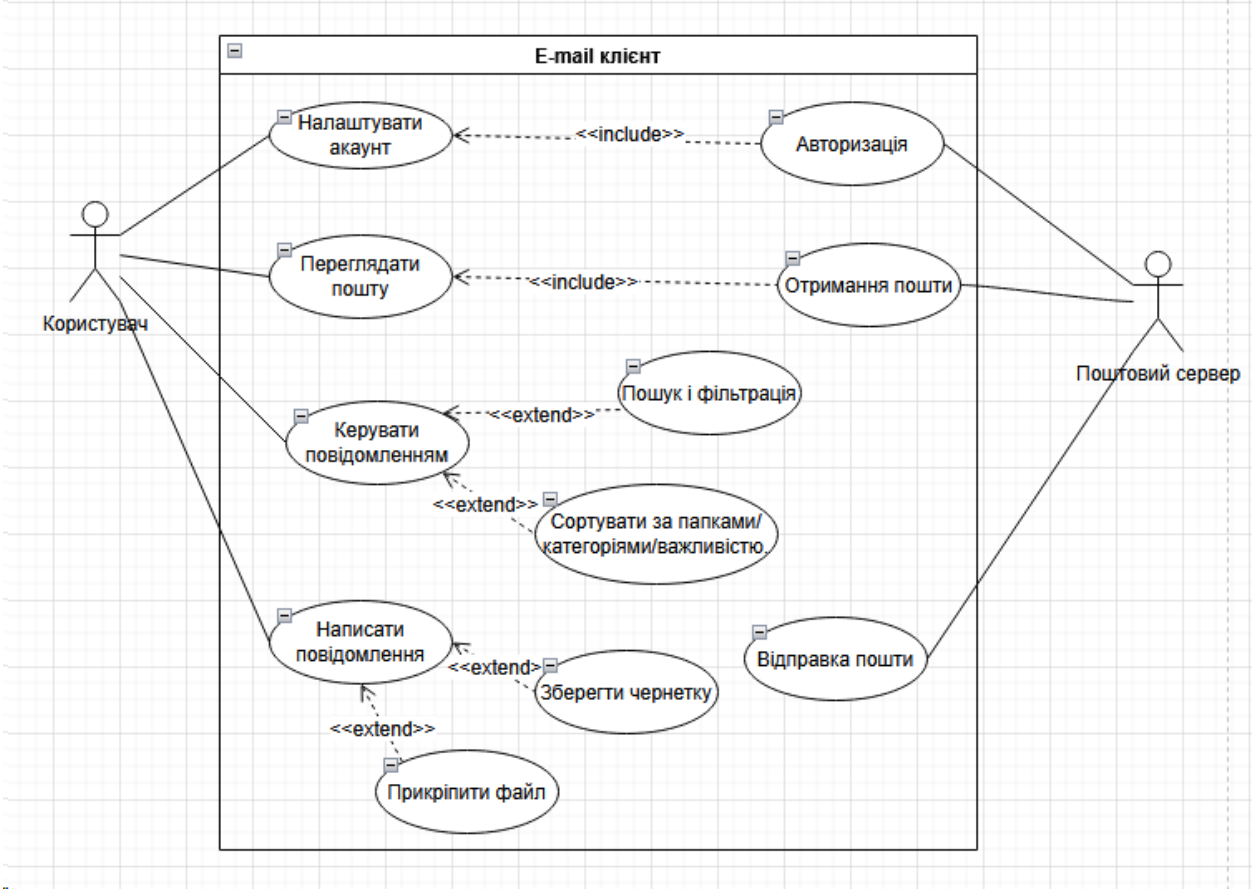

Рисунок2.17-Діаграма варіантів використання

1. Актори

У даній діаграмі є 2 актори:

- Користувач
- Поштовий сервер

2.Зв’язки між елементами

- Користувач → Налаштувати акаунт
- Користувач → Переглядати пошту
- Користувач → Керувати повідомленням
- Користувач → Написати повідомлення
- Налаштувати акаунт → (include) → Авторизація
- Переглядати пошту → (include) → Отримання пошти
- Керувати повідомленням → (extend) → Пошук і фільтрація

- Керувати повідомленням → (extend) → Сортувати за папками/категоріями/важливістю
- Написати повідомлення → (extend) → Зберегти чернетку
- Написати повідомлення → (extend) → Прикріпити файл
- Написати повідомлення → (include) → Відправка пошти
- Поштовий сервер → Авторизація
- Поштовий сервер → Отримання пошти
- Поштовий сервер → Відправка пошти

3.Варіанти використання

- Налаштування акаунта – користувач вводить параметри поштової скриньки (адреса, пароль, сервер), щоб мати змогу користуватися поштовим клієнтом.
- Авторизація – підтвердження облікових даних користувача на поштовому сервері для доступу до електронної пошти.
- Перегляд пошти – основна функція, яка дозволяє користувачу переглядати отримані повідомлення.
- Отримання пошти – поштовий клієнт звертається до поштового сервера через протокол и POP3/IMAP для завантаження нових повідомлень.
- Керування повідомленням – користувач може видаляти, переміщати або позначати листи як прочитані/непрочитані.
- Пошук і фільтрація – розширена функція, яка дозволяє знаходити потрібні повідомлення за ключовими словами, відправником чи датою.
- Сортування за папками/категоріями/важливістю – користувач може впорядковувати пошту для зручності.
- Написання повідомлення – користувач створює новий лист.

- Прикріплення файлу – до повідомлення додаються вкладення (зображення, документи тощо).
- Збереження чернетки – повідомлення може бути збережене для подальшого редагування.
- Відправка пошти – готовий лист передається на поштовий сервер через протокол SMTP для доставки адресату.

Сценарії використання

1. Сценарій: Налаштувати акаунт

Передумови:

- Користувач має поштовий акаунт.
- Клієнт встановлений і запущений.

Постумови: Акаунт успішно додано, користувач авторизований.

Взаємодіючі сторони:Користувач, E-mail клієнт, Поштовий сервер

Короткий опис: Користувач вводить дані свого акаунта (логін, пароль), клієнт перевіряє їх і зберігає.

Основний перебіг подій:

- Користувач обирає «Налаштувати акаунт».
- Вводить поштову адресу, пароль.
- Клієнт надсилає запит на поштовий сервер.
- Сервер підтверджує правильність даних.
- Клієнт зберігає акаунт і переходить до поштової скриньки.

Винятки:

- Некоректний логін/пароль - повідомлення про помилку.
- Сервер недоступний - повідомлення про відсутність з’єднання.

Примітки: Можлива підтримка кількох акаунтів.

2. Сценарій: Переглядати пошту

Передумови:

- Користувач налаштував акаунт і авторизувався.
- Є доступ до інтернету.

Постумови: Користувач бачить оновлений список повідомлень.

Взаємодіючі сторони: Користувач, E-mail клієнт, Поштовий сервер

Короткий опис: Користувач переглядає пошту. Клієнт отримує нові повідомлення з сервера та відображає їх у списку.

Основний перебіг подій:

- Користувач обирає «Переглядати пошту».
- Клієнт звертається до поштового сервера.
- Сервер повертає список повідомлень.
- Клієнт відображає повідомлення.
- Користувач відкриває вибраний лист для читання.

Винятки:

Сервер недоступний - клієнт повідомляє про помилку.

Немає нових повідомлень - повідомлення «Немає нових листів».

Примітки:

Доступні додаткові дії: пошук, фільтрація, сортування.

3. Сценарій: Написати повідомлення

Передумови:

- Користувач авторизований.
- Доступний поштовий сервер.

Постумови: Повідомлення відправлене або збережене як чернетка.

Взаємодіючі сторони: Користувач, E-mail клієнт, Поштовий сервер

Короткий опис: Користувач створює нове повідомлення, вводить одержувача, текст, може прикріпити файл. Після цього відправляє лист або зберігає його як чернетку.

Основний перебіг подій:

- Користувач обирає «Написати повідомлення».
- Вводить адресу одержувача, тему, текст.
- Додає вкладення .
- Натискає «Відправити».
- Клієнт надсилає повідомлення серверу.
- Сервер підтверджує відправку.
- Повідомлення відображається у папці «Відправлені».

Винятки:

Сервер недоступний - повідомлення зберігається у «Чернетках».

Некоректна адреса одержувача - повідомлення про помилку.

Примітки:

Можливе створення кількох чернеток і вкладення кількох файлів.

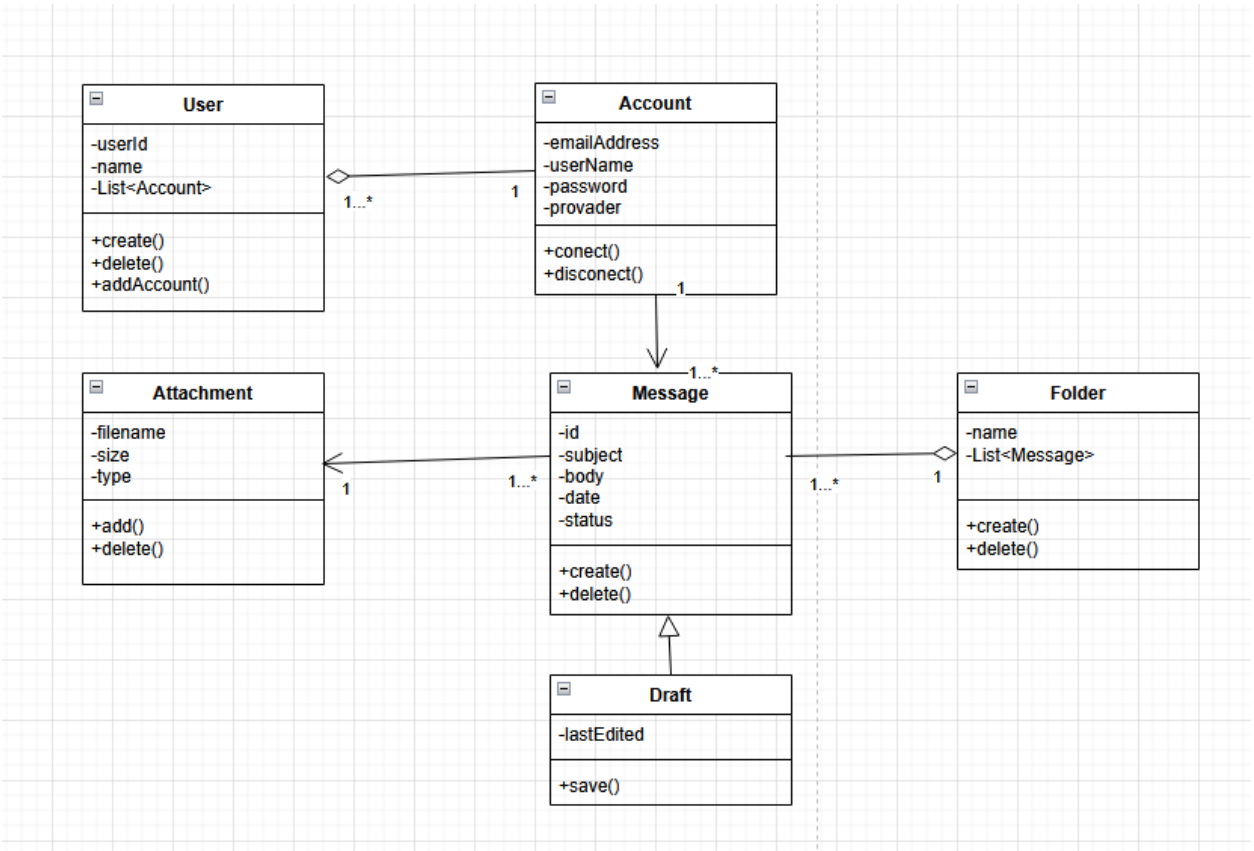

Рисунок 2.18- Діаграма класів

Діаграма моделює основні сутності поштового клієнта та їх зв’язки:

- User -користувач системи, який може мати один або кілька акаунтів.
- Account -поштовий акаунт із параметрами доступу (логін, пароль, налаштування сервера).
- MailServer – поштовий сервер (POP3/IMAP/SMTP), що обслуговує акаунти.
- Message - електронне повідомлення з темою, текстом, статусом.
- Folder -папка для впорядкування повідомлень (Вхідні, Відправлені, Чернетки тощо).
- Attachment – вкладений файл у повідомленні.
- Draft - спеціалізований клас, що успадковує Message і зберігає незавершені повідомлення.

Основні зв’язки:

- Користувач - має акаунти.
- Акаунт - належить до поштового сервера.
- Акаунт - містить повідомлення.
- Повідомлення - належить до папки.
- Повідомлення - може містити вкладення.
- Чернетка - є підкласом повідомлення.

Для реалізації шаблону repository були зроблені інтерфейси для взаємодії з базою даних

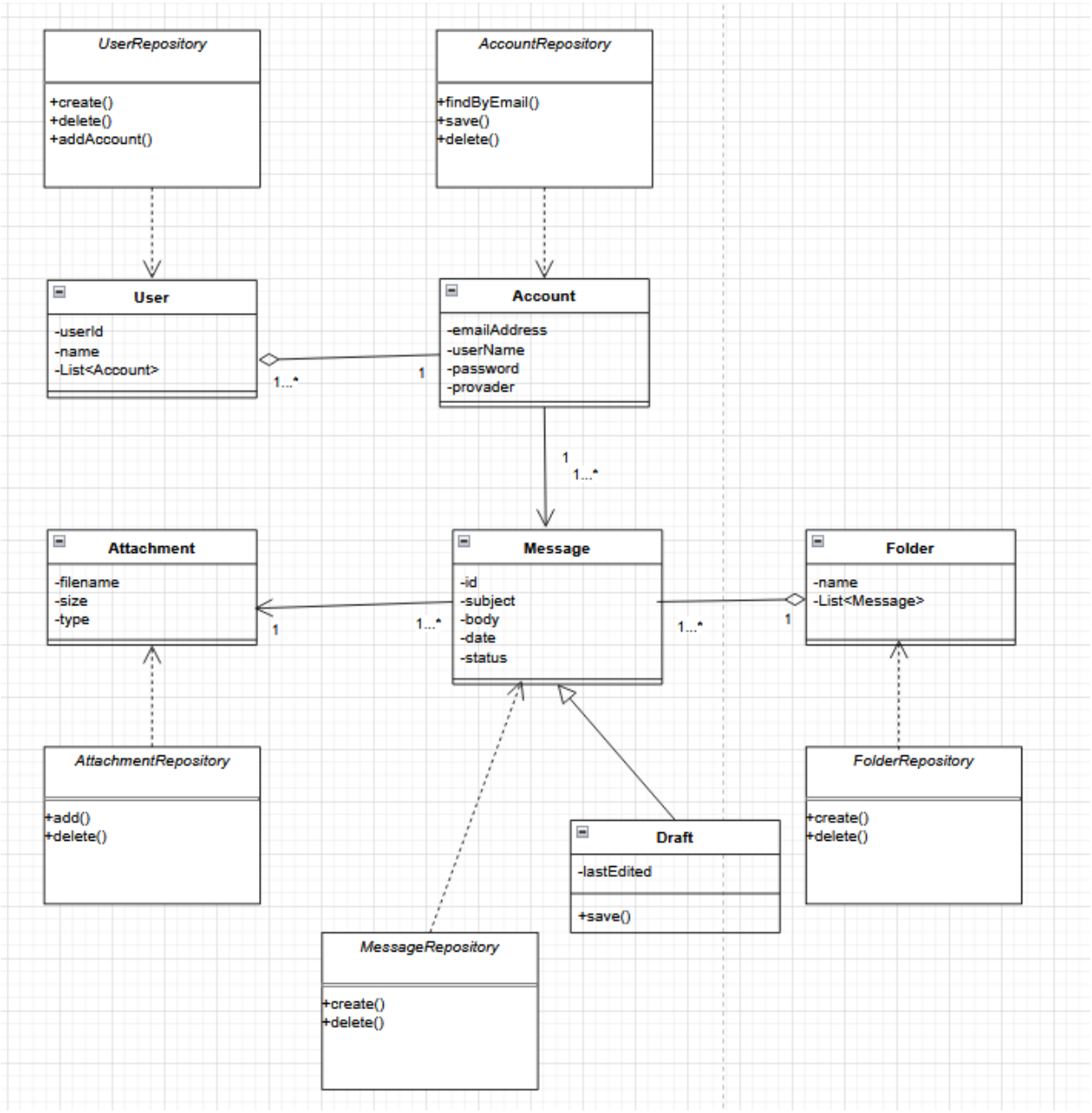

Рисунок 2.19-Діаграма класів з шаблоном Repository

Вихідний код класів на мові java:

```
public class Account {

  private int id;

    private String login;

    private String password;

    private String provider;

    private int userId;
```

```
    private int mailServerId;
    private List<Message> messages;
//гетери та сетери та інші
}
public class Attachment {
  private int id;
    private String filename;
    private long size;
    private String type;
    private int messageId;
//гетери та сетери та інші
}
public class Draft extends Message {
  private Date lastEdited;
//гетери та сетери та інші
}
public class Folder {
    private int id;
      private String name;
      private int accountId;
      List<Message> messages;
//гетери та сетери та інші
}
public class MailServer {
  private int id;
    private String name;
```

```
    private String smtpHost;
    private int smtpPort;
    private String pop3Host;
    private int pop3Port;
    private String imapHost;
    private int imapPort;
//гетери та сетери та інші
}
public class Message {
  private int id;
    private String subject;
    private String body;
    private Date date;
    private String status;
    private int accountId;
    private int folderId;
    private List<Attachment> attachments;
//гетери та сетери та інші
}
public class User {
  private int id;
    private String name;
    private List<Account> accounts;
//гетери та сетери та інші
}
public interface AccountRepository {
```

```
  Account findById(int id);
    List<Account> findByUser(int userId);
    void save(Account account);
    void update(Account account);
    void delete(int id);
}
public interface AttachmentRepository {
   Attachment findById(int id);
      List<Attachment> findByMessage(int messageId);
      void save(Attachment attachment);
      void delete(int id);
}
public interface DraftRepository {
   Draft findById(int id);
      List<Draft> findAll();
      void save(Draft draft);
      void update(Draft draft);
      void delete(int id);
}
ublic interface FolderRepository {
  Folder findById(int id);
    List<Folder> findByAccount(int accountId);
    void save(Folder folder);
    void update(Folder folder);
    void delete(int id);
}
```

```
public interface MailServerRepository {
  MailServer findById(int id);
    List<MailServer> findAll();
    void save(MailServer server);
    void update(MailServer server);
    void delete(int id);
}
public interface MessageRepository {
  Message findById(int id);
    List<Message> findByAccount(int accountId);
    List<Message> findByFolder(int folderId);
    void save(Message message);
    void update(Message message);
    void delete(int id);
}
public interface UserRepository {
  User findById(int id);
    List<User> findAll();
    void save(User user);
    void update(User user);
    void delete(int id);
}
```

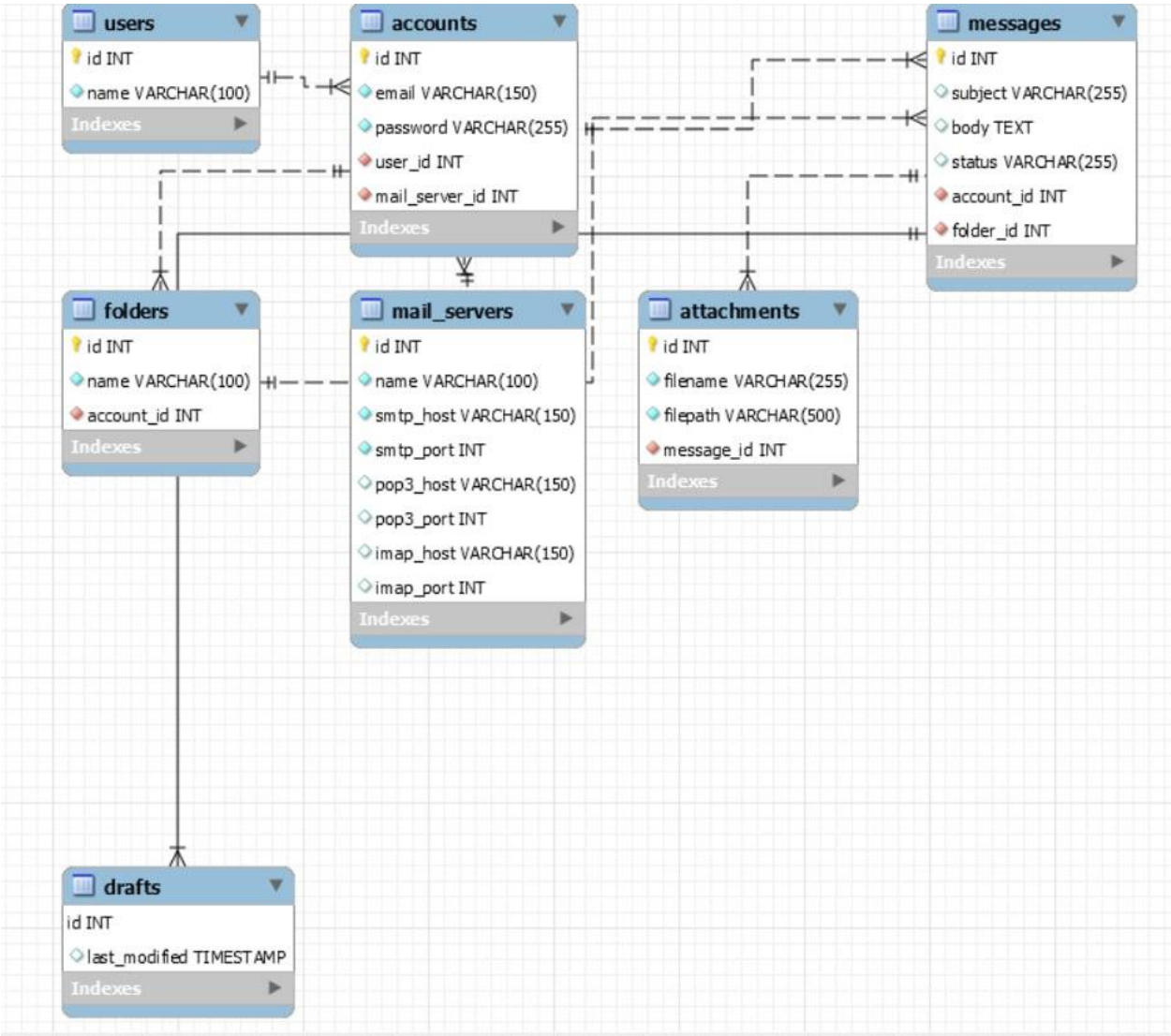

Рисунок 2.20-Структура бази даних

Питання до лабораторної роботи

1. Що таке UML?

UML (Unified Modeling Language) — це уніфікована стандартизована мова візуального моделювання, яка використовує графічні позначення для створення абстрактних моделей складних систем.

2. Що таке діаграма класів UML?

Діаграма класів UML — це статична структурна діаграма, яка візуально представляє структуру об'єктно-орієнтованої системи, показуючи класи, їх атрибути, методи та взаємозв'язки між ними.

3. Які діаграми UML називають канонічними?

Канонічні діаграми називають основні, найбільш фундаментальні типи діаграм UML, які найбільш широко використовуються для моделювання процесів та структур програмного забезпечення, такі як: 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).

4. Що таке діаграма варіантів використання?

Діаграми варіантів використання - це графічне представлення взаємозв'язків між акторами та одним або декількома варіантами використання, що підтримуються рішенням.

5. Що таке варіант використання?

Варіант використання - це опис того, як певний суб'єкт (актор) взаємодіє з системою для досягнення певної мети, а також можливих результатів цієї взаємодії

6. Які відношення можуть бути відображені на діаграмі використання?

Асоціація - відношення між актором і варіантом використання, що показує взаємодію.

Include (включення**)** - один варіант використання завжди містить інший як обов’язкову частину.

Extend (розширення) - додатковий варіант використання, який виконується за певних умов.

Узагальнення (generalization) - відношення наслідування між акторами або варіантами використання.

7. Що таке сценарій?

Сценарій - це послідовність кроків або дій, які описують конкретний випадок взаємодії актора із системою для досягнення певної мети. Сценарій деталізує варіант використання, пояснюючи, як саме система повинна реагувати на дії користувача чи зовнішнього процесу.

8. Що таке діаграма класів?

Діаграма класів – це структурна діаграма UML, яка відображає класи системи, їх атрибути, методи та відношення між ними. Вона описує статичну структуру програмного забезпечення і використовується для моделювання предметної області та проектування архітектури системи.

9. Які зв’язки між класами ви знаєте?

Асоціація - зв’язок між класами, який показує, що об’єкти одного класу взаємодіють з об’єктами іншого.

Агрегація - "ціле–частина", коли один клас містить інші, але вони можуть існувати окремо.

Композиція - сильна форма агрегації, коли частина не може існувати без цілого.

Узагальнення - відношення наслідування між класами (клас-нащадок успадковує властивості базового класу).

Залежність - один клас тимчасово використовує інший (наприклад, як параметр методу).

10. Чим відрізняється композиція від агрегації?

Агрегація – це слабкий зв’язок. Частини можуть існувати незалежно від цілого. Якщо ‘ціле’ знищується, його ‘частини’ можуть залишитися.

Композиція – це сильний зв’язок. Частини не можуть існувати без цілого. Якщо ‘ціле’ знищується, знищуються й усі його ‘частини’.

11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на

діаграмах класів?

Агрегація позначається порожнім ромбом біля класу-цілого. Композиція позначається затемненим ромбом біля класу-цілого.

12. Що являють собою нормальні форми баз даних?

Нормальні форми баз даних — це набір правил і принципів організації даних у реляційних базах даних, які покликані усунути надлишковість, уникнути аномалій при вставці, оновленні та видаленні даних, а також забезпечити логічну цілісність даних**.**

Змінна відношення знаходиться в першій нормальній формі (1НФ) тоді і тільки тоді, коли в будь-якому допустимому значенні відношення кожен його кортеж містить лише одне значення для кожного з атрибутів.

Змінна відношення знаходиться в другій нормальній формі тоді і тільки тоді, коли вона знаходиться в першій нормальній формі, і кожен неключовий атрибут функціонально повно залежить від її потенційного ключа.

Змінна відношення знаходиться у третій нормальній формі тоді і лише тоді, коли вона знаходиться у другій нормальній формі, і відсутні транзитивні функціональні залежності неключових атрибутів від ключових.

Змінна відношення знаходиться в нормальній формі Бойса-Кодда (інакше – в посиленій третій нормальній формі) тоді і тільки тоді, коли кожна її нетривіальна і неприведена зліва функціональна залежність має в якості свого детермінанта певний потенційний ключ.

13. Що таке фізична модель бази даних? Логічна?

Розрізняють дві моделі бази даних – логічну та фізичну. Фізична модель бази даних представляє собор набір бінарних даних у вигляді файлів, структурованих та згрупованих згідно з призначенням (сегменти, екстенти та ін.), що використовується для швидкого та ефективного

отримання інформації з жорсткого диска, а також для компактного зберігання та розміщення даних на жорсткому диску. Логічна модель бази даних є структурою таблиць, уявлень, індексів та інших логічних елементів бази даних, що дозволяють власне програмування та використання бази даних.

14. Який взаємозв'язок між таблицями БД та програмними класами?

Взаємозв'язок між таблицями бази даних і програмними класами полягає у відображенні структур даних та їхніх взаємозв’язків між двома середовищами: реляційною БД і об’єктно-орієнтованим програмуванням .

Висновок: В процесі роботи було створено діаграми варіантів використання, які відображають взаємодію користувачів із системою, а також розроблено сценарії цих варіантів, що допомагає чітко визначити послідовність дій та поведінку системи. Крім того, побудовано діаграми класів предметної області, що забезпечують наочне уявлення про структуру даних та взаємозв’язки між об’єктами системи. Такий підхід сприяє кращому розумінню системи та полегшує її подальшу реалізацію.