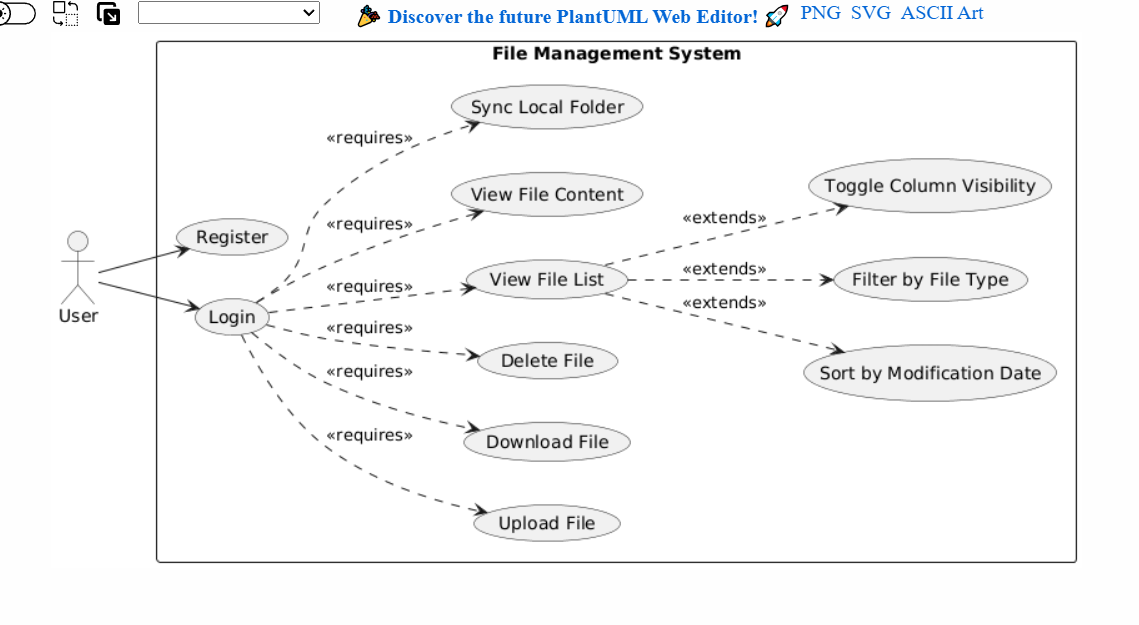
# ЗВІТ: Етап 1 - UML-специфікація

**Варіант:** 88  
**Система:** Тип 2 - Клієнт для взаємодії з віддаленою папкою з файлами  
**Типи файлів для перегляду:** .kt (Kotlin), .jpg  
**Операції:** Сортування за датою модифікації (зростання/спадання), фільтр для відображення файлів типу .kt, .jpg

## 1. Діаграма прецедентів (Use Case Diagram)

Діаграма відображає всі функціональні можливості системи:

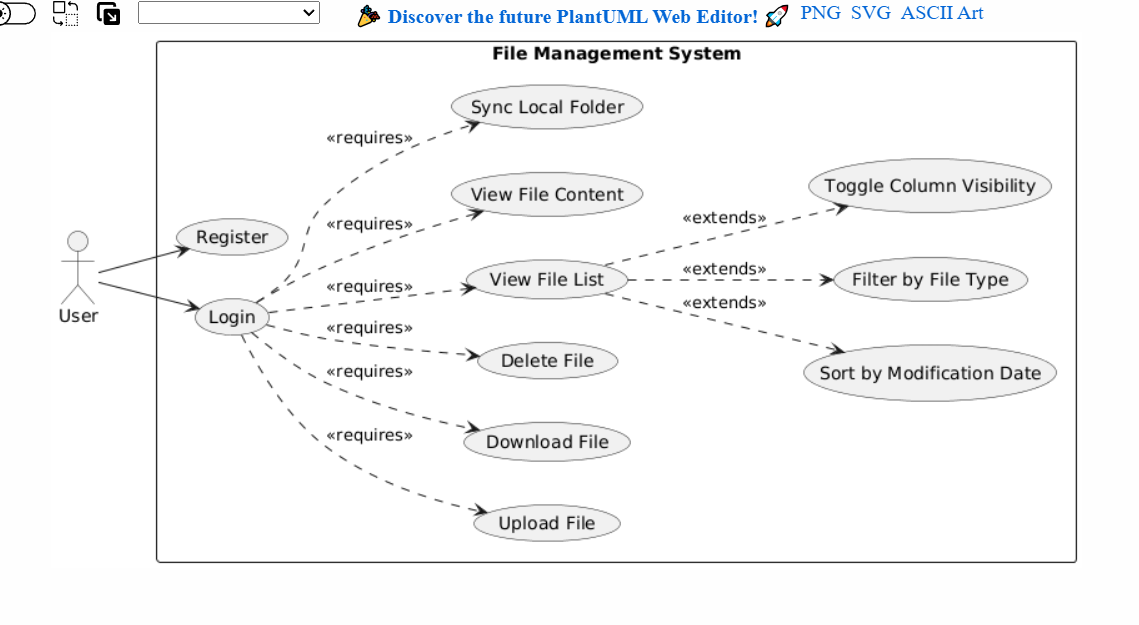
**Основні прецеденти:**

* **Register / Login** - автентифікація (requires для всіх операцій)
* **View File List** - перегляд списку файлів
* **Upload File** - завантаження файлу на сервер
* **Download File** - вивантаження файлу
* **Delete File** - видалення файлу (тільки автор)
* **View File Content** - перегляд вмісту .kt та .jpg файлів
* **Sort by Modification Date** (extends View File List) - індивідуальна операція варіанту 88
* **Filter by File Type** (extends View File List) - фільтрація .kt/.jpg
* **Toggle Column Visibility** (extends View File List) - приховування колонок
* **Sync Local Folder** (requires Register/Login) - синхронізація папки (Electron)

**Зв'язки:**

* requires - Register/Login обов'язкові для всіх операцій з файлами
* extends - Sort, Filter, Toggle розширюють View File List

## 2. Діаграма класів (Class Diagram) - Загальна

Основні класи системи:

**Domain Model:**

**User:** Поля: userId (Long), username (String), password (String), email (String) Методи: login() boolean, logout() void

**File:** Поля: fileId (Long), name (String), type (String), content (byte array), createdDate (DateTime), modifiedDate (DateTime), uploaderId (Long) Методи: upload() void, download() byte array, delete() void

**FileMetadata:** Поля: fileId (Long), name (String), type (String), size (long), createdDate (DateTime), modifiedDate (DateTime), uploaderName (String), editorName (String)

**Services:**

**AuthService:** Методи: authenticate(username, password) Token, validateToken(token) boolean, logout(token) void

**FileManager:** Методи: uploadFile(File, User) boolean, downloadFile(fileId) File, deleteFile(fileId) boolean, listFiles() List FileMetadata, sortByModifiedDate(order) List FileMetadata, filterByType(types) List FileMetadata

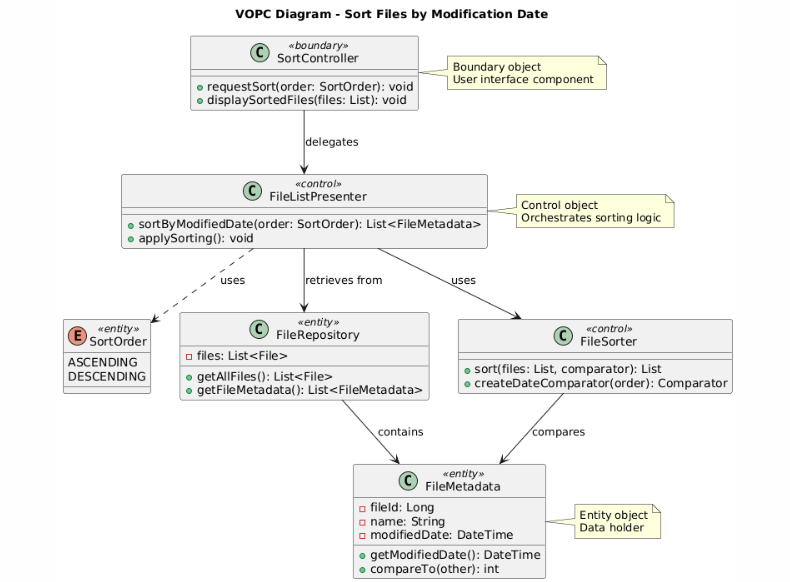
**SyncService:** Поля: localPath (String), remotePath (String) Методи: syncToRemote() void, syncToLocal() void, compareDirectories() List String

**FileRenderer:** Методи: renderKotlin(content) String, renderImage(content) Image, displayFile(file) void

**Зв'язки:**

* User (1) owns (0..\*) File
* FileManager manages File
* FileManager creates FileMetadata
* AuthService authenticates User
* FileRenderer renders File

## 3. VOPC-діаграма для індивідуальної операції (Sort by Modification Date)

**Діаграма для варіанту 88: Сортування за датою модифікації**

**View (Boundary) - SortController:** Методи: requestSortOrder(SortOrder) void, displaySortedFiles(files List) void

* Boundary object - User interface component
* Приймає запит від користувача (ASCENDING/DESCENDING)

**Operation (Control) - FileListPresenter:** Методи: sortByModifiedDate(order SortOrder) List FileMetadata, applySorting() void

* Control object - Orchestrates sorting logic
* Делегує запит до репозиторію

**Policy (Entity) - SortOrder:** Значення: ASCENDING, DESCENDING

* Entity object - Enum з правилами сортування

**Content (Entity) - FileRepository:** Поля: files (List File) Методи: getAllFiles() List File, getFileMetadata() List FileMetadata

* Entity object - Доступ до даних

**Content (Entity) - FileSorter:** Методи: sortFiles(List, comparator) List, createDateComparator(order) Comparator

* Control object - Алгоритм сортування

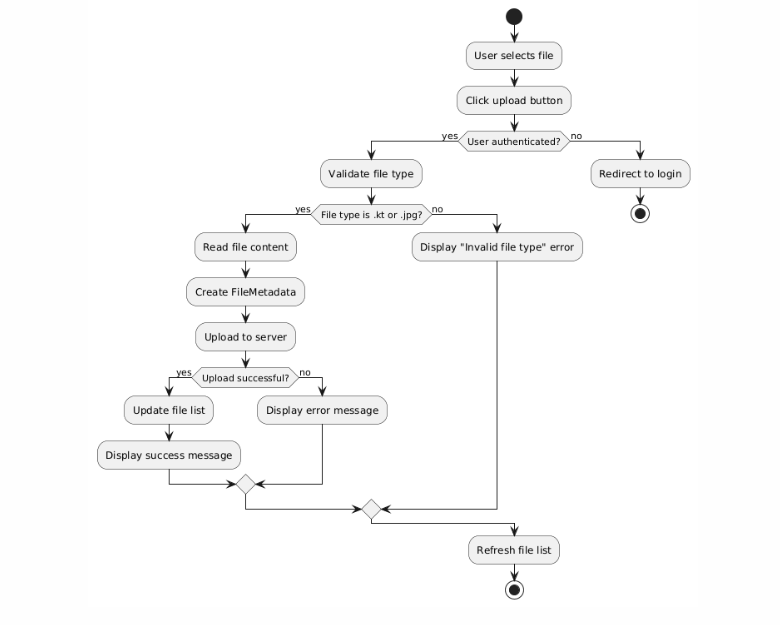
**Content (Entity) - FileMetadata:** Поля: fileId (Long), name (String), modifiedDate (DateTime) Методи: getModifiedDate() DateTime, compareTo(other) int

* Entity object - Data holder з полем для сортування

**Flow:**

1. User натискає "Sort by Modified Date (Desc)"
2. SortController делегує запит FileListPresenter
3. FileListPresenter використовує FileRepository.getAllFiles()
4. FileListPresenter використовує FileSorter.createDateComparator(DESCENDING)
5. FileSorter порівнює FileMetadata.modifiedDate
6. FileListPresenter повертає sorted files
7. SortController відображає sorted list

## 4. Діаграма активності (Activity Diagram)



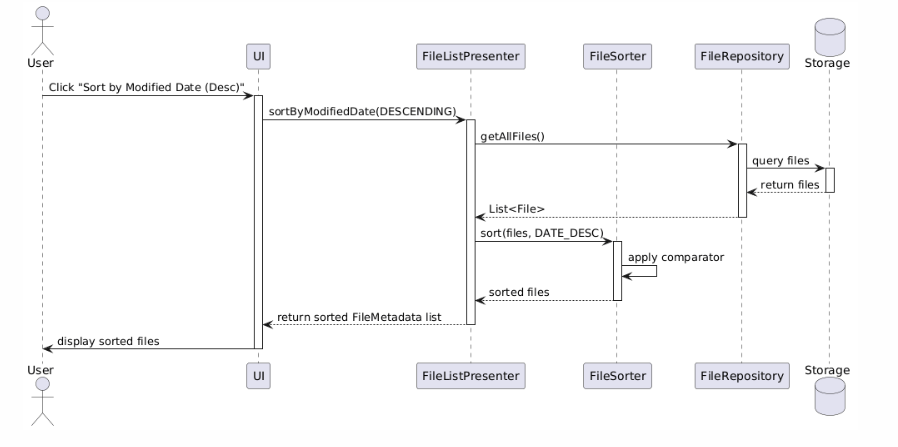
**Процес завантаження файлу:**

1. **Start:** User selects file
2. **Action:** Click upload button
3. **Decision:** User authenticated?
   * NO → Redirect to login → End
   * YES → Continue
4. **Action:** Validate file type
5. **Decision:** File type is .kt or .jpg?
   * NO → Display "Invalid file type" error → End
   * YES → Continue
6. **Action:** Read file content
7. **Action:** Create FileMetadata
8. **Action:** Upload to server
9. **Decision:** Upload successful?
   * YES → Update file list → Display success message → Refresh file list → End
   * NO → Display error message → End

**Альтернативні потоки:**

* Invalid type → помилка валідації
* Token invalid → редирект на login
* Upload failed → показ помилки, можливість retry

## 5. Діаграма взаємодії #1 (Sequence Diagram) - Сортування

**[**

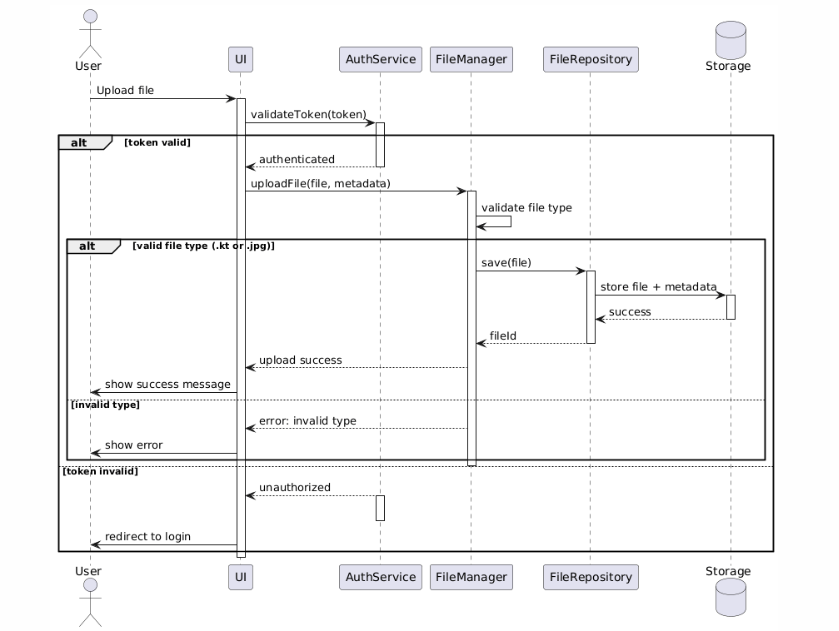
**Сценарій: User clicks "Sort by Modified Date (DESCENDING)"**

Послідовність: User натискає кнопку "Sort by Modified Date (Desc)" → UI викликає FileListPresenter.sortByModifiedDate(DESCENDING) → FileListPresenter викликає FileSorter.getAllFiles() → FileSorter запитує FileRepository.query files → FileRepository звертається до Storage → Storage повертає List File → FileRepository передає List File назад FileSorter → FileSorter викликає sort(files, DATE\_DESC) → FileSorter застосовує comparator → FileSorter повертає sorted files FileListPresenter → FileListPresenter повертає sorted FileMetadata list UI → UI відображає sorted files User

**Ключові моменти:**

* Сортування виконується через comparator
* FileRepository отримує всі файли з Storage
* FileSorter застосовує DATE\_DESC (modifiedDate порівняння)
* UI отримує вже відсортований список

## 6. Діаграма взаємодії #2 (Sequence Diagram) - Завантаження файлу

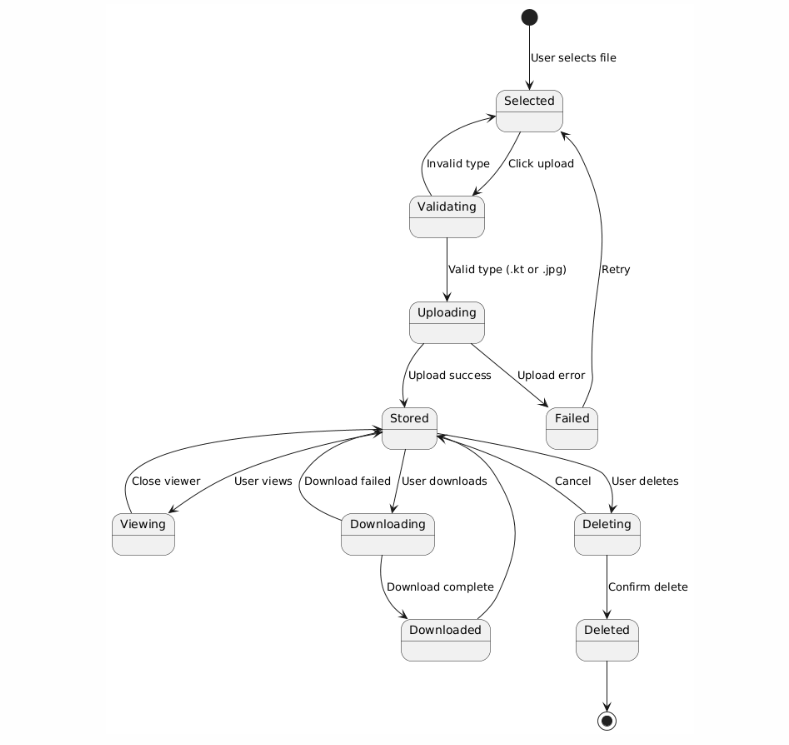
**Сценарій: User uploads file with authentication**

Послідовність: User → UI Upload file → UI → AuthService validateToken(token) → alt block [token valid]: AuthService → UI authenticated → UI → FileManager uploadFile(file, metadata) → FileManager → FileManager validate file type → alt block [valid file type (.kt or .jpg)]: FileManager → FileRepository save(file) → FileRepository → Storage store file + metadata → Storage → FileRepository success → FileRepository → FileManager filed → FileManager → UI upload success → UI → User show success message, else [invalid type]: FileManager → UI error invalid type → UI → User show error, else [token invalid]: AuthService → UI unauthorized → UI → User redirect to login

**Alt blocks:**

* Token validation - перший checkpoint
* File type validation - другий checkpoint (.kt або .jpg)
* Різні шляхи для успіху/помилки

## 7. Діаграма станів (State Diagram)



**Стани файлу в системі:**

**1. Initial State → Selected** User selects file

**2. Selected** Transitions:

* Invalid type / Click upload → Validating
* Retry → Selected

**3. Validating** Transitions:

* Valid type (.kt or .jpg) → Uploading
* Invalid type → Failed

**4. Uploading** Transitions:

* Upload success → Stored
* Upload error → Failed

**5. Stored** (центральний стан) Transitions:

* User views → Viewing
* Download failed → Stored
* User downloads → Downloading
* Cancel → Stored
* User deletes → Deleting

**6. Viewing** Transition:

* Close viewer → Stored

**7. Downloading** Transitions:

* Download complete → Downloaded

**8. Deleting** Transition:

* Confirm delete → Deleted

**9. Downloaded** (final state)

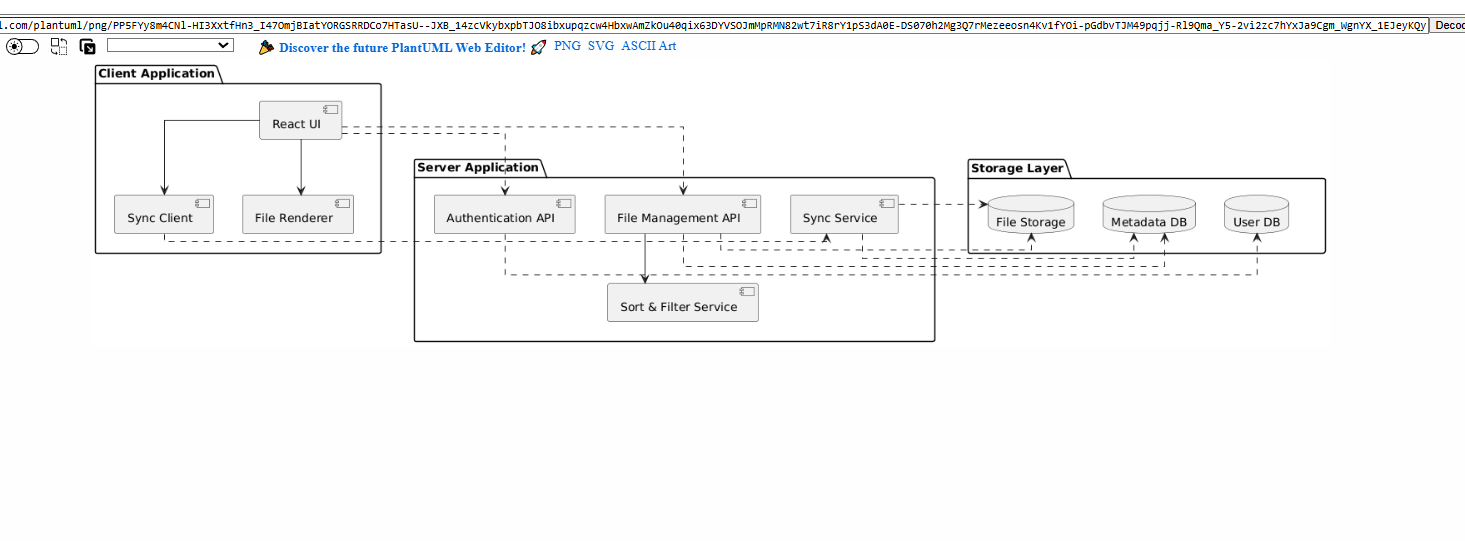
**10. Deleted** (final state)

**11. Failed** (final state)

**Особливості:**

* З Stored можна перейти до View/Download/Delete
* Failed та Deleted - фінальні стани
* Можливість Retry з Failed назад до Selected

## 8. Діаграма компонентів (Component Diagram)

**Архітектура системи:**

**Client Application:**

* **React UI** → взаємодіє з Server Application
* **Sync Client** → для Electron версії
* **File Renderer** → відображення .kt/.jpg

**Server Application:**

* **Authentication API** → JWT токени
* **File Management API** → CRUD операцій
* **Sort & Filter Service** → індивідуальна операція

**Storage Layer:**

* **File Storage** → фізичні файли на диску
* **Metadata DB** → PostgreSQL з метаданими
* **User DB** → дані користувачів

**Зв'язки:**

* React UI ↔ Authentication API (HTTP/REST)
* React UI ↔ File Management API (HTTP/REST)
* File Management API → File Storage (File I/O)
* File Management API → Metadata DB (JDBC)
* Authentication API → User DB (JDBC)
* Sort & Filter Service → Metadata DB (queries з ORDER BY)

## 9. Діаграма розгортання (Deployment Diagram)

### 

### Stage 1: Desktop Deployment

**Client Machine:**

* **Electron App:**
  + Electron Shell
  + React UI
  + Local FS (для sync)

**Зв'язок:** HTTP → Local Server (localhost:8081)

**Local Server (localhost:8081):**

* **Spring Boot API:**
  + File Service
  + Auth Service

**Зв'язок:** JDBC → PostgreSQL + File Storage

### Stage 2: Web Deployment

**Browser:**

* **React Web App** (localhost:3000)

**Зв'язок:** HTTP → Local Server (localhost:8081)

**Local Server (localhost:8081):**

* **Spring Boot API:**
  + File Service (deployed)
  + Auth Service (deployed)

**Зв'язок:** JDBC → PostgreSQL + File Storage

**Відмінності:**

* Desktop: Electron обгортка + можливість sync
* Web: Браузер без Electron, без sync
* **Обидва використовують той самий локальний Spring Boot backend (localhost:8081)**

## 10. Database Schema (PostgreSQL)

### 10.1 Структура таблиць

**Таблиця users:**

Зберігає дані користувачів системи. Містить первинний ключ user\_id типу SERIAL (автоінкремент), унікальний username (до 50 символів), хешований пароль BCrypt (255 символів), email та мітку часу створення акаунту. Username має UNIQUE constraint для запобігання дублікатів.

**Таблиця file\_metadata:**

Основна таблиця для метаданих файлів. Первинний ключ file\_id типу SERIAL. Поля: назва файлу (255 символів), тип файлу без крапки (10 символів), розмір у байтах (BIGINT), шлях на диску сервера (500 символів).

Часові мітки: created\_date та modified\_date типу TIMESTAMP з автоматичним встановленням CURRENT\_TIMESTAMP при створенні.

Денормалізовані поля uploader\_name та editor\_name зберігають імена користувачів безпосередньо для швидкості запитів без JOIN.

**Foreign Keys:**

Два зовнішні ключі посилаються на таблицю users:

* uploader\_id references users(user\_id) з ON DELETE SET NULL
* editor\_id references users(user\_id) з ON DELETE SET NULL

Використання SET NULL означає що при видаленні користувача файли залишаються в системі (спільний робочий простір). Альтернатива CASCADE видалила б всі файли користувача.

### 10.2 Індекси для оптимізації

Створено три B-tree індекси:

**idx\_file\_uploader** на полі uploader\_id - прискорює запити типу "показати всі файли користувача". Використовується рідко в поточній реалізації через спільний робочий простір.

**idx\_file\_type** на полі type - оптимізує фільтрацію за типом файлу. Коли користувач вмикає фільтр .kt або .jpg, запит WHERE type IN використовує цей індекс замість full table scan.

**idx\_file\_modified** на полі modified\_date - **це ключовий індекс для індивідуальної операції варіанту 88**. Прискорює сортування ORDER BY modified\_date ASC/DESC. Без індексу PostgreSQL робить sequential scan + sort, з індексом - index scan (значно швидше для великої кількості файлів).

### 10.3 Зв'язки між таблицями

Users та file\_metadata мають зв'язок one-to-many через два foreign keys:

* Один user може бути uploader багатьох файлів
* Один user може бути editor багатьох файлів

При цьому один файл завжди має одного uploader та одного editor (який може співпадати з uploader).

Денормалізація через uploader\_name та editor\_name - trade-off між нормалізацією та продуктивністю. Запит списку файлів не потребує JOIN з таблицею users, що робить його швидшим, але займає більше місця та вимагає оновлення при зміні username.

## Висновки

Розроблено повний комплект UML-діаграм:

* 1 діаграма прецедентів
* 1 діаграма класів (загальна)
* 1 VOPC-діаграма (для індивідуальної операції - сортування)
* 1 діаграма активності
* 2 діаграми взаємодії (sequence)
* 1 діаграма станів
* 1 діаграма компонентів
* 1 діаграма розгортання (2 варіанти: Desktop + Web)

**Всього: 9 діаграм**