

# Project FileVault – Week 3


## Milestone 3: Database, File Visibility, Folders, Background Tasks & System Cleanup

---

### Objective

In this milestone, you're enhancing the FileVault system with features that mirror real-world file storage platforms:

- Add **folder structure and hierarchy**
- Implement **file visibility settings (public/private)**
- Introduce **background jobs**
- Refactor your growing codebase to use the **Repository pattern** for the data layer.

 You will still be using only the **CLI interface** this week.

### Concepts to Learn & Practice

- Background jobs / task queues
  - File visibility and access levels
  - Recursive data relationships (folders/subfolders)
  - Using the **Repository Pattern** to abstract data access
  - Worker vs. CLI separation of concerns
  - System refactoring and maintainability
- 

## Required Features

### 1. Folder Support

Extend file metadata to support folders.

- Add **type** field to every file: "**file**", "**folder**", or "**image**"
- Add **parent\_id** (for nesting inside folders)

Implement these new commands:

**mkdir <folder\_name> [parent\_id]**

Create a new folder.

```
$ vault mkdir "Documents"
Folder created: Documents
```

**ls [parent\_id]**

List contents of a folder (defaults to root)

```
$ vault ls
[folder] Documents/
[image] cat.jpg
[file] resume.pdf
```

Refactor upload command to do the following:

```
$ vault upload <file_name> <folder>
```

```
$ vault upload text.txt materials
- Creates the folder if it doesn't exist
- Associates the file with folder
```

## 2. File Visibility Controls

Each file (or folder) can be:

- **Private** (default): only owner can access
- **Public**: anyone (logged in or not) can see metadata

Add these CLI commands:

**publish <file\_id>**

Makes a file or folder public.

**unpublish** `<file_id>`

Makes a file or folder private again.

### 3. Background Jobs (Async Workers)

Introduce a worker that can run in the background to process tasks. Start with:

#### Thumbnail Generation (for images)

- When uploading an image, save the original AND schedule a background job to generate a thumbnail version (e.g., 100x100px).
- Save thumbnails in `./storage/thumbnails/<file_id>.jpg`
- You can use:
  - **Celery + Redis** (Python)
  - **BullMQ + Redis** (Node)

### 4. Refactor with the Repository Pattern

Move all database access logic into repository classes.

- Create:
  - `UserRepository`
  - `FileRepository`
  - `SessionRepository`
- All service logic must now **call repositories** instead of writing queries directly.

This improves:

- Testability
- Decoupling
- Readability

---

## Suggested Folder Structure (Updated)

```
project/
├── cli/
│   ├── commands/
│   │   ├── upload.py
│   │   ├── mkdir.py
│   │   ├── publish.py
│   │   └── ...
│   └── services/
│       ├── file_service.py
│       ├── user_service.py
│       └── thumbnail_service.py
├── repositories/
│   ├── user_repository.py
│   ├── file_repository.py
│   └── session_repository.py
├── workers/
│   └── thumbnail_worker.py
├── storage/
│   ├── uploads/
│   └── thumbnails/
├── database/
├── utils/
├── .env
└── README.md
```

## Architecture Diagram

- [Project Architecture](#)

## Bonus

- Add `mv <file_id> <parent_id>` to move files into folders
  - Add support for viewing public files from other users
  - Display folders first in `ls` listings
-

## Milestone Completion Checklist

- ☐ Files and folders support `parent_id` for nesting
  - ☐ Can create folders (`mkdir`)
  - ☐ Can list files/folders recursively
  - ☐ Can set/unset visibility on any file/folder
  - ☐ Image uploads trigger background thumbnail generation
  - ☐ Thumbnails are saved in a dedicated location
  - ☐ Database access is now abstracted via Repository pattern
  - ☐ Project folder structure is clearly organized
  - ☐ CLI still runs with session and ownership enforcement
- 



## Resources

- [Celery \(Python\) Quickstart](#)
- [BullMQ \(Node.js\) Queues](#)
- [Repository Pattern Intro](#)
- [Python Pillow \(Image Processing\)](#)