PROJECT DOCUMENTATION

DROPBOX CLONE PROJECT

BY EMMANUEL GODWIN BASSEY
STUDENT NUMBER: 3092017
22/04/2025

CONTEXT

PROJECT OVERVIEW

This Dropbox Clone is a web application that mimics the core functionality of Dropbox, allowing users to store, manage, and share files in the cloud. The application provides user authentication, file upload/download capabilities, directory management, file sharing between users, and duplicate file detection.

The project implements a complete end-to-end solution for cloud storage, addressing both frontend user experience and backend security concerns. By leveraging modern web technologies and cloud services, the application offers a responsive, secure, and scalable platform for file management.

This cloud-based solution enables users to access their files from anywhere with internet connectivity. The system maintains a virtual directory structure that mirrors physical file organization while adding features like file sharing and duplicate detection that enhance the user experience beyond basic storage functionality.

The application follows best practices in web development, including responsive design principles, security measures to protect user data, and optimization techniques to ensure performance even with large files or slow network connections. Error handling and user feedback mechanisms are incorporated throughout to provide a smooth and reliable user experience.

TECHNOLOGIES

The technologies used in this project are modern and secure, ensuring a safe and reliable environment for users to process and store their data. The key technologies involved are:

Backend Technologies:

1. Python 3.8+ : programming language used to build the API
2. Flask 2.0.1: Web application framework
3. Firebase Admin SDK 5.03 : Server-side library for managing firebase services
4. Werkzeug 2.0.1 : Utility library used by flask for request and response operation handling.
5. Gunicorn 20.1.0 : Python HTTP server for production deployment

Front-End Technologies:

1. Html 5 : used for application structure and content
2. CSS : styling application structure and responsive design techniques
3. JavaScript : client-side programming for dynamic interactions
4. Firebase Client SDK : browser side libraries for firebase service integration
5. Font Awesome : icons for visual elements

Cloud Services:

1. Firebase Authentication: multi authentication security method for users
2. Firebase Firestore: document database for structured data storage with real time capabilities
3. Firebase Storage: scalable object storage optimized for user generated content
4. Firebase Security Rules: security model for data protection

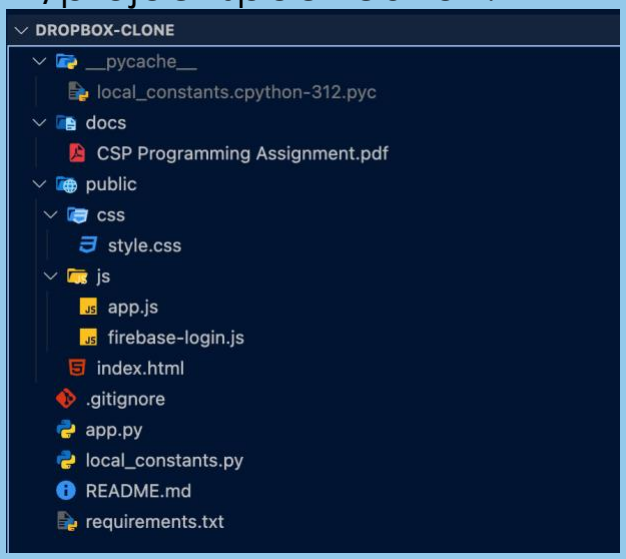
Development Tools:

1. Git & GitHub: Version control
2. PIP: python package manager
3. Virtual Environments : python environment for development

DIRECTORY STRUCTURE

Physical Project Structure

- **public/** – Contains all frontend assets accessible to clients:
  - **css/**
    - **style.css** – Comprehensive styling for all application components , includes responsive design patterns.
  - **js/**
    - **firebase-login.js** – Handles authentication and user session management.
    - **app.js** – Core frontend logic for file and directory management.
  - **index.html** – Main HTML file with a responsive layout structure.
- **app.py** – Main Flask backend application containing all API routes.
- **local_constants.py** – Stores Firebase credentials and environment variables.
- **requirements.txt** – Lists Python dependencies with exact version specifications.
- **docs/** – Contains documentation and project references:
  - **CSP Programming Assignment.pdf** – Original assignment/project specification.



Virtual File System Structure

- **Root Directory (/)**

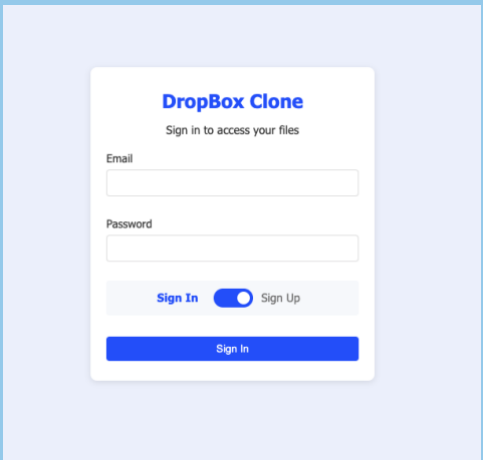
    - Automatically created for each user.
    - Serves as the base path for all personal files and folders.
- **Hierarchical Structure**
    - Supports unlimited nesting of folders and files.
- **Path-Based Navigation**
    - Uses Unix-style paths (e.g., /user/docs/notes.txt).
    - Allows navigation using .. for parent directories.
- **Special Directories**
    - A virtual "shared" folder gives users access to files shared by others.
- **Access Control**
    - Permissions are inherited from parent directories.
- **Directory Metadata**
    - Each directory stores:
        - Creation time
        - Owner information
        - Folder name
        - Reference to parent directory
- **Database Structure**
    - The file system is stored virtually in the database, not on the server's physical file system.
    - Enables easy access control and enhanced flexibility.

USER INTERFACE

Login and Authentication Interface

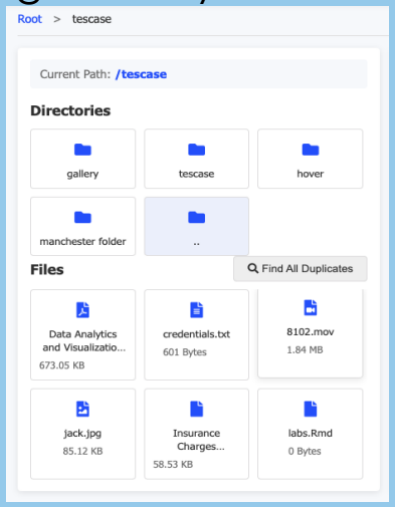
- Dual-Mode Form: Toggle between sign-in and sign-up.

- Validation Feedback: Real-time input checks with error messages.
- Persistent Sessions: Sessions remain active until logout.
- Security Features: Password strength meter and secure credential handling.
- Error Recovery: Failure feedback and recovery options.



File Explorer Interface

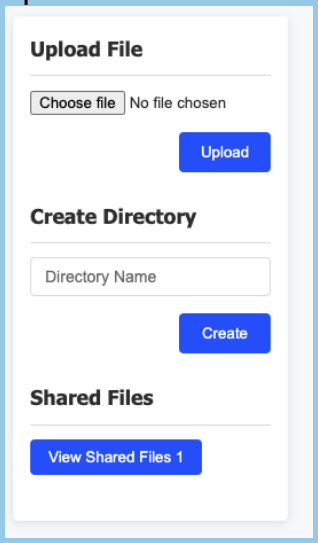
- Breadcrumb Navigation: Displays current folder path.
- Two-Panel Layout: Separate views for folders and files.
- Context-Sensitive Actions: Contextual options based on selection.
- Drag-and-Drop Support: Easy file uploads via drag interface.
- File Type Identification: Icons based on file types.
- Sorting Options: Organize by name, size, or date.



File Operations Interface

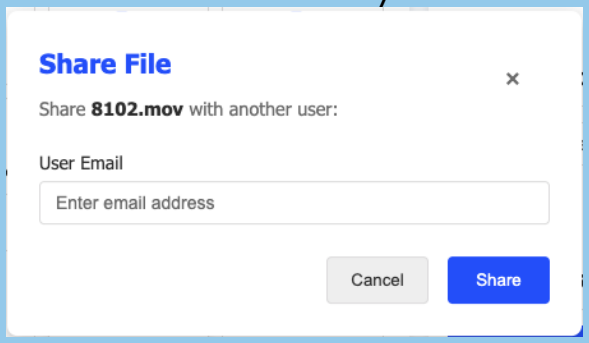
- Upload Progress Tracking: Visual progress bar and percentage.

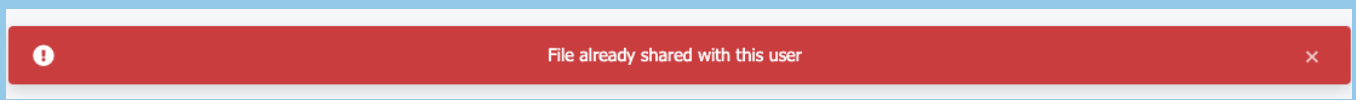
- Download Management: Uses browser's download manager.
- Multi-File Operations: Supports multi-file operations
- Confirmation Dialogues: Safeguards before destructive actions.
- Error Recovery: Retry options for failed operations.

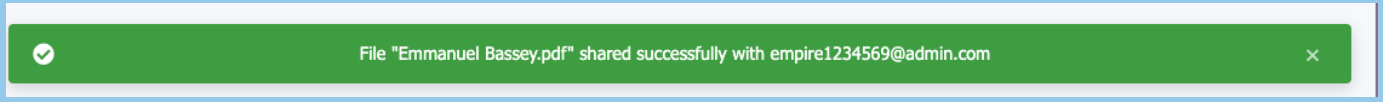


Sharing Interface

- Recipient Email Entry: Input and validate recipient emails.
- Access Control Options: Read-only or full access.
- Share Status Indicator: Visual tags for shared files.
- Revocation Controls: Instantly stop sharing.
- Notification System: Success or failure notifications.







Frontend Methods

Authentication (firebase-login.js)

- showNotification(message, type): Displays animated success/error/info banners with auto-dismiss.
- onAuthStateChanged(callback): Monitors Firebase login/logout and updates the UI accordingly.
- signIn(email, password): Logs in existing users with error handling.
- signUp(email, password): Registers users and initializes their on account creation storage.
- signOut(): Ends session and resets app state(back to login).
- initializeUser(user): Calls backend to set up root directory on first login.

File Explorer (app.js)

- loadCurrentDirectory(path): Loads directory contents and updates UI.
- renderDirectories(directories) / renderFiles(files): Renders directory/file cards and attaches events.
- navigateToDirectory(directoryId): Enables folder navigation, including back/parent navigation.
- updateBreadcrumbs(): Generates clickable path breadcrumbs.
- getFileIcon(fileName) / formatFileSize(bytes): Utility functions for icons and readable sizes.

File Operations

- uploadFile(file, overwrite): Supports chunk uploads, progress bar, duplicate check, and retry logic.
- downloadFile(fileId): Secure download with access checks and temporary URLs.
- deleteFile(fileId): Confirms and deletes file with success/failure feedbacks and metadata.
- createDirectory(name) / deleteDirectory(id): Adds/removes folders with validation.

File Sharing

- shareFile(field, recipientEmail): Shares files with other users after verifying emails.
- showShareForm(field, fileName) / hideShareForm(): Controls visibility of sharing UI.
- loadSharedFiles(): Displays files shared *with* the user.
- downloadSharedFile(shareId, fileName) / removeSharedFile(shareId): Handles downloads/removal of shared content securely.

Utilities and UI Enhancements

- handleNetworkChange(): Detects offline/online and adapts interface.
- showConfirmationDialog(message, callback): Modal for file manipulation actions.
- setupSessionTimeout(): Warns and times out inactive sessions.
- findAllDuplicates(): Scans for duplicate files using hashing.

Backend Methods

Core (app.py)

- **index():** Entry point that serves the single-page application HTML.
- **init_user(uid, email):** This endpoint initializes a new user by creating their profile and root directory when they first log in. It accepts a Firebase UID and the user's email address. If the user does not already exist in the users collection, a new user document and root directory (/) are created. A success

message is returned with status 200 OK, or 400 Bad Request if an error occurs.

Directory API

- **get_directories(uid, path):** Retrieves all folders within the given path, including a '..' entry if not in root. Returns an array with 200 OK or 400 Bad Request on failure.
- **create_directory(uid, path, name):** Creates a new directory after validating the name and checking for duplicates. Returns 200 OK if successful, or 400/409 on error or name conflict.
- **delete_directory(uid, id):** Deletes the specified directory only if it's empty and owned by the user. Returns 200 OK, or 400/409 if it contains content or is unauthorized.
- **navigate_directory(uid, id, current_path):** Returns the path to navigate to, including handling parent directory logic. Returns 200 OK or 404 Not Found.

File API

- **get_files(uid, path):** Lists files in a folder.
- **upload_file(uid, file, path, overwrite):** Uploads files, hashes for deduplication.
- **delete_file(uid, file_id):** Removes file from storage and Firestore.
- **get_download_url(uid, file_id):** Generates a safe, time-limited download link.

Duplicate Detection

- **find_duplicates(uid, file_hash, size):** Identifies duplicate files in a directory by grouping them based on hash values. Returns duplicates or 400 Bad Request for invalid input.
- **find_all_duplicates(uid):** Scans all files across the user's storage for duplicates and returns grouped results. Returns 200 OK or 400 Bad Request..

Sharing API

- **share_file(uid, file_id, recipient_email):** Shares file with another user securely via email.
- **get_shared_files(uid):** Returns a Lists of all shared files with or by the user.
- **get_shared_file_url(uid, share_id):** Generates secure link to shared content.
- **remove_shared_file(uid, share_id):** Revokes file access with permission checks.

Data Models

User

- UID: Unique identifier (from Firebase Auth).
- Email: Used for login and sharing.
- Timestamps: For account creation and last login.
- Display Name & Settings: user-friendly identifier for sharing interfaces

Directory

- User ID: Owner reference.
- Name, Path, Parent: Full navigation metadata.
- Timestamps: Created and modified.
- Attributes: JSON for extensibility.

File

- User ID: Foreign key reference to the owner of the directory for access control.
- Name: User-defined directory name with input validation.
- Path: Route and file structure in which files/folders are nested.
- Content type: Detects the file format in which they are uploaded.
- Storage Path: Firebase Storage reference path for content retrieval

- Hash: SHA-256 content hash for duplicate detection.
- Timestamps: Used in tracking file changes and creation.
- Size: File size in bytes for storage accounting and display

Shared File

- Owner ID / Recipient ID: reference to the original file owner/ file receiver.
- File reference: Link to original file.
- Access type: Permissions (read-only).

Firebase Integration

Authentication

- Supports email/password.
- Handles JWTs, real-time authentication state, and timed-sessions with updates to the UI.
- Role-based access via custom claims.

Firestore Database

- Structured collections for users, directories, files, and shares.
- Indexes for efficient querying.
- Strong security rules and transactional operations.
- Real-time listener architecture (currently polling-based) supports realtime changes.

Firebase Storage

- Files organized in user-specific paths mirroring virtual directory structure
- Optional deduplication by content hash.
- Metadata and access control via secure URLs.
- Supports resumable uploads and content scanning.

Security

- Server-side validation for all actions.
- Strong CSP, rate limiting, input sanitization.
- Protection against path traversal.

- Full audit logging for sensitive events.

Conclusion

This Dropbox Clone project demonstrates the effective use of cloud service platforms to build a secure and responsive file storage system. By integrating a Flask backend with a modern JavaScript frontend and Firebase's cloud services, the application provides seamless file management, user authentication, and sharing functionality.

REFERENCES

1. **Firebase Documentation. Firebase Docs**
2. **Griffith College Dublin – Cloud Services & Platforms Lecture. Module Notes**
3. **Google Cloud Platform – Firebase Hosting Overview. Firebase Hosting**
4. **GITHUB Repo Link || https://github.com/Beez1/dropbox-clone**