VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI - 590018

Mini Project - BCS586

Report on

"LightViewer: A Cloud-Based Document Organizer for Efficient Storage, Categorization and Access"

Submitted in partial fulfillment of the requirement for the award of the degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted By

Karthik G. Sharma	**1DT22CS070**
Manu S	**1DT22CS089**
Chandrashekar Hiremath	**1DT23CS403**
Abhishek Wadekar	**1DT23CS400**

Under the Guidance of

Prof. Keerthana Shankar
Assistant Professor, Department of CSE

Department of Computer Science and Engineering

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT
(Affiliated to VTU, Belagavi and Approved by AICTE, New Delhi),
CE, CSE, ECE, EEE, ISE, ME Courses Accredited by NBA, New Delhi, NAAC A+

ACADEMIC YEAR 2024-25

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

Udayapura, Kanakapura Road, Bangalore-560082

Department of Computer Science and Engineering

CERTIFICATE

Certified that the Mini Project titled **"LightViewer: A Cloud-Based Document Organizer for Efficient Storage, Categorization, and Access"** carried out by **Karthik G. Sharma**, bearing **USN 1DT22CS070, Manu S**, bearing **USN 1DT22CS089, Chandrashekar Hiremath**, bearing **USN 1DT23CS403, Abhishek Wadekar**, bearing **USN 1DT23CS400,** bonafide students of Dayananda Sagar Academy 0f Technology and Management, is in partial fulfillment for the award of the **BACHELOR OF ENGINEERING** in **Computer Science and Engineering** from Visvesvaraya Technological University, Belagavi during the year 2024- 2025. It is certified that all the corrections/suggestions indicated for Internal Assessment have been incorporated in the report submitted to the department. The Project report has been approved as it satisfies the academic requirements in respect of the Mini Project Work prescribed for the said Degree.

Prof. Keerthana Shankar	**Dr. Nandini C**	**Dr. M Ravishankar**
Assistant Professor	Vice-Principal &	Principal
Department of CSE	HOD Department	DSATM,
DSATM, Bengaluru.	of CSE, DSATM.	Bengaluru.

DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT
Udayapura, Kanakapura Road, Bangalore-560082
Department of Computer Science and Engineering

DECLARATION

We, **Karthik G. Sharma**, bearing **USN 1DT22CS070, Manu S**, bearing **USN 1DT22CS089, Chandrashekar Hiremath**, bearing **USN 1DT23CS403, Abhishek Wadekar**, bearing **USN 1DT23CS400**, students of Fifth Semester B.E, Department of Computer Science and Engineering, Dayananda Sagar Academy of Technology and Management, Bengaluru, declare that the Mini Project Work titled "**LightViewer: A Cloud-Based Document Organizer for Efficient Storage, Categorization, and Access**" has been carried out by us and submitted in partial fulfilment of the course requirements for the award of degree in **Bachelor of Engineering** in **Computer Science and Engineering** from **Visvesvaraya Technological University, Belagavi** during the academic year **2024- 2025**.

Karthik G. Sharma	1DT22CS070
Manu S	1DT22CS089
Chandrashekar Hiremath	1DT23CS403
Abhishek Wadekar	1DT23CS400

Place: Bengaluru
Date:

ACKNOWLEDGEMENT

The satisfaction and the euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible. The constant guidance of these people and encouragement provided, crowned us with success and glory. We take this opportunity to express our gratitude to one and all.

It gives us immense pleasure to present before you our project titled "**LightViewer: A Cloud-Based Document Organizer for Efficient Storage, Categorization, and Access**". The joy and satisfaction that accompanies the successful completion of any task would be incomplete without the mention of those who made it possible. We are glad to express our gratitude towards our prestigious institution **DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT** for providing us with utmost knowledge, encouragement and the maximum facilities in undertaking this project.

We wish to express sincere thanks to our respected Principal **Dr. M Ravishankar**, Principal, DSATM for all his support.

We express our deepest gratitude and special thanks to **Dr. Nandini C , Vice-** Principal**&** H.O.D, Dept. of Computer Science Engineering, for all her guidance and encouragement.

We sincerely acknowledge the guidance and constant encouragement of our mini-project guide,

Prof. Keerthana Shankar, Assistant Professor, Dept. of Computer Science & Engineering.

ABSTRACT

LightViewer is a cloud-based document management system designed to address inefficiencies in managing and organizing files across platforms, enhancing productivity and collaboration for individuals and teams. Built on AWS services, it ensures scalable, secure storage and efficient file organization, utilizing AWS EC2 for processing, S3 for storage, and DynamoDB for metadata management. AWS Cognito integrates secure multi-factor authentication and role-based access control, while AWS Lambda enables real-time synchronization across devices. The intuitive Flutter-based interface ensures compatibility with iOS and Android, offering features like file categorization, metadata-driven advanced search, and real-time updates. Rigorous testing validated its usability, performance, and reliability, positioning LightViewer as a transformative tool for streamlining document management and fostering collaboration.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In today’s fast-paced digital era, managing and organizing documents has become increasingly challenging for both individuals and teams. Traditional methods of file storage and organization often lead to issues such as redundant copies, limited accessibility, and inefficient collaboration. With the ever-growing volume of digital content, the demand for innovative and efficient solutions to streamline document management has become more pressing than ever.

LightViewer, a cloud-based mobile application, addresses these challenges by offering a seamless platform for document organization and management. The application empowers users to store, categorize, and access their files efficiently, regardless of the device or operating system. Its real-time synchronization feature ensures that documents remain consistently updated across all devices, eliminating version conflicts and enabling smooth collaboration.

Moreover, its intuitive search functionality allows users to locate files quickly using customizable filters and tags, significantly enhancing productivity.

The LightViewer system is built on a robust cloud-based architecture designed for scalability, security, and accessibility. This architecture enables users to access their documents anytime and anywhere while maintaining a secure environment for their sensitive data. The application is designed with user experience at its core, providing a simple and intuitive interface that ensures ease of navigation. Key features, such as customizable tags and folders for document categorization, make it effortless to manage large volumes of data efficiently.

Real-time synchronization ensures that any changes made to documents are instantly reflected across all devices, fostering effective collaboration among team members and maintaining consistency across files. LightViewer further enhances its value proposition by incorporating advanced encryption and secure cloud storage protocols, offering users peace of mind regarding the safety of their information.

This report explores the design, functionality, and performance of LightViewer, highlighting how it simplifies document management for individuals and teams alike. By addressing common challenges associated with traditional document management systems, LightViewer stands out as a transformative tool that reduces administrative overhead and enhances overall productivity in the digital workspace.

CHAPTER 2

REQUIREMENT SPECIFICATION

2.1 Hardware Requirements

Android smartphone or tablet with a minimum configuration:

- Processor: Quad-core 1.5 GHz or higher
- RAM: 2GB or more
- Storage: Minimum 500MB free space for installation and caching
- Display: Touchscreen interface with a minimum resolution of 1280x720 pixels
- Internet Connectivity: Wi-Fi or mobile data for real-time synchronization and cloud operations

2.2 Software Requirements

2.2.1 Backend Requirements

Programming Language: Dart (for primary backend logic) & Python 3 (for AWS Lambda functions)

Web Framework: Dart-based backend framework (e.g., Aqueduct or Shelf)

Cloud Services:

- AWS EC2 (for hosting APIs and application logic written in Dart)
- AWS S3 (for secure and scalable storage of documents)
- AWS Cognito (for user authentication and access control)
- AWS DynamoDB (for metadata storage and search functionalities)
- AWS Lambda (for event-driven real-time synchronization and background tasks)

2.2.2 Frontend Requirements

Programming Framework: Flutter (for cross-platform compatibility across Android and iOS)

Development Environment: Android Studio (version 2022.1 or above)

Device Compatibility: Android 7.0 (Nougat) or later

2.3 Specific Requirements

2.3.1 Functional Requirements

1. User Authentication and Authorization
    - Registration

- Users can create an account by providing a username, email, and password
- Passwords are hashed and stored securely using industry-standard encryption

- Login: Registered users can log in using their credentials
- Logout: Users can log out securely from the application
- Password Management: Users can reset their passwords via a secure email verification process

2. Document Management

- Upload Documents: Users can upload documents and associate them with tags or categories
- Categorization: Documents can be organized based on user-defined tags or categories
- Search Functionality: Users can search for documents using keywords, categories, or metadata such as file type and date of creation

3. Real-Time Synchronization

All updates to files or metadata are synchronized across all connected devices in real time

4. File Handling

- Secure Upload and Download: Ensure secure upload and download of files with end-to-end encryption
- Integrity Validation: Verify the integrity of files during uploads and downloads to prevent data corruption

5. User Interface

- Responsive Design: The application should be optimized for Android devices with a consistent and responsive layout
- Ease of Use: Provide intuitive navigation and clear instructions for document management features

2.3.2 Non-functional Requirements

1. Performance
   - The system should provide real-time synchronization with minimal latency
   - Efficient handling of large document files and metadata for search and categorization
2. Security
   - Ensure secure handling of user data and documents using advanced encryption techniques
   - Protection against common vulnerabilities, including SQL injection, cross-site scripting (XSS), and unauthorized access
3. Usability
   - User-friendly interface with clear instructions
   - Easy navigation and interaction

4. Reliability
    - The system should be highly available and reliable
    - Proper error handling and user feedback
5. Scalability
    - The application should be able to handle an increasing number of users and media files
    - Efficient resource utilization

2.3.4 External Interface Requirements

1. User Interfaces
    - Registration and Login pages for user authentication
    - Feedback and result pages
2. Hardware Interfaces
    - The Cloud operations will interact with AWS infrastructure for backend processing, storage, and authentication
3. Software Interfaces
    - Dart Backend Framework: The application backend will utilize a Dart-based framework (e.g., Shelf or Aqueduct) for handling API requests
    - AWS Cloud Services: Integration with AWS S3, DynamoDB, Cognito, and Lambda for storage, metadata management, authentication, and serverless operations
    - Database Management: AWS DynamoDB will serve as the primary NoSQL database, ensuring efficient metadata handling
    - Python for Lambda Functions: Utilize Python scripts for executing event-driven tasks in AWS Lambda, such as real-time synchronization

2.3.5 System Features

1. User Authentication System
    - Secure user registration, login, and management
    - Password hashing and validation
2. File Handling System
    - Secure upload, processing, and download of files
    - Ensuring file integrity and security
3. User Interface Design
    - Responsive and intuitive design for user interaction
    - Clear instructions and feedback mechanisms

2.3.6 Security Requirements

1. Data Protection
   - Secure storage and handling of user data
   - Encryption of sensitive data
2. File Security
   - Secure handling of uploaded and downloaded files
   - Validation and sanitization of file inputs
3. User Authentication
   - Strong password policies
   - Protection against brute-force attacks

2.3.7 Performance Requirements

1. Efficiency
   - Fast uploading and downloading of files
   - Optimized processing of large media files
2. Response Time
   - Quick response times for user actions
   - Minimal delays in file processing

2.3.8 Scalability Requirements

1. User Load
   - Ability to handle a quite large number of simultaneous users
   - Efficient resource management for scalability
2. Data Volume
   - Handling large volumes of media files
   - Scalable storage solutions

By adhering to these software requirements, the LightViewer project will be equipped with the essential functionalities and characteristics to deliver a secure, reliable, and user-friendly application for efficient document management, and seamless collaboration across devices.

CHAPTER 3

SYSTEM ANALYSIS AND DESIGN

3.1 System Analysis

System analysis is a crucial phase in the software development lifecycle, involving the detailed study of the needs of the users and the constraints of the system. This phase includes gathering requirements, analyzing them, and modeling the system to meet those requirements.

Requirements Gathering

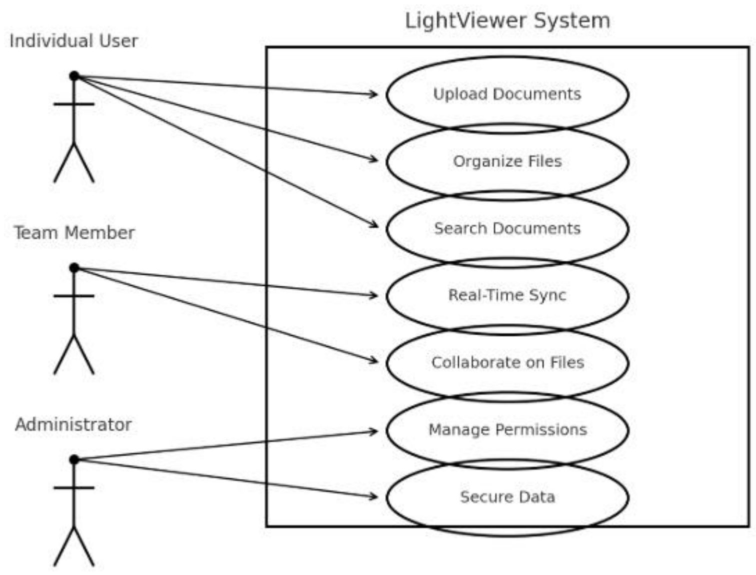
1. Functional Requirements:

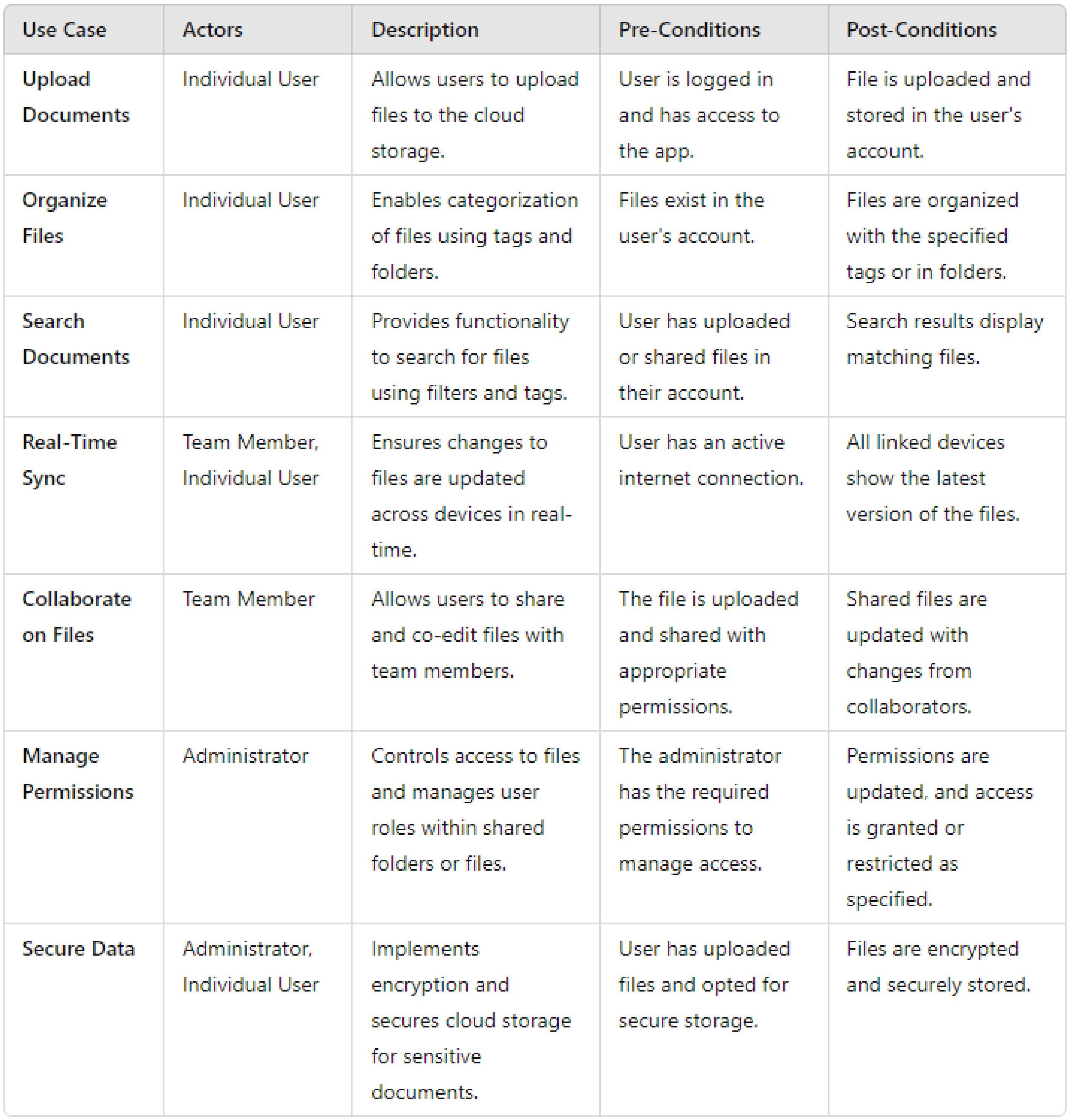
- Document Upload and Categorization: The system must allow users to upload documents and organize them using customizable tags or categories for efficient management
- Real-Time Synchronization: Ensure real-time updates to documents and metadata across all connected devices, maintaining consistency and eliminating version conflicts
- Advanced Search Functionality: Enable users to locate documents quickly using metadata filters, including tags, categories, file types, and creation dates
- User Interface: Provide an intuitive and responsive user interface for seamless navigation and efficient use of features such as file uploads, search, and categorization
- Error Handling: Implement robust error handling to gracefully manage issues, such as unsupported file formats or failed uploads, providing clear feedback to the user

2. Non-Functional Requirements:

- Performance: Uploading and downloading processes should be efficient, minimizing processing time
- Scalability: The system should scale to handle larger file sizes and its resolutions
- Security: System should ensure the security of the hidden files against unauthorized access & tampering
- Usability: The system should be easy to use, with clear instructions and an intuitive interface
- Maintainability: The system should be designed to facilitate easy updates and maintenance

Use Case Diagram:

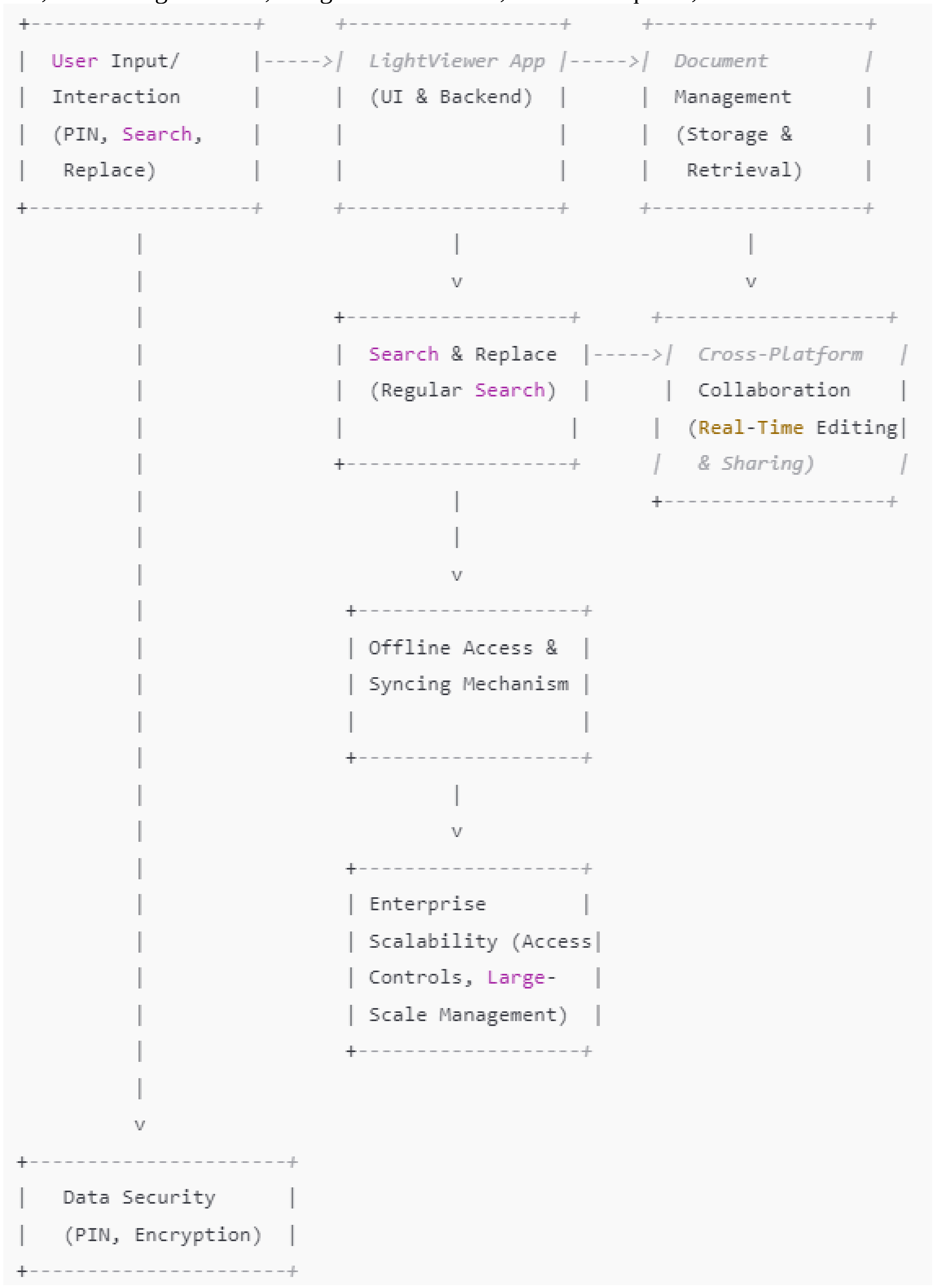




Use Case	Actors	Description	Pre-Conditions	Post-Conditions
Upload Documents	Individual User	Allows users to upload files to the cloud storage.	User is logged in and has access to the app.	File is uploaded and stored in the user's account.
Organize Files	Individual User	Enables categorization of files using tags and folders.	Files exist in the user's account.	Files are organized with the specified tags or in folders.
Search Documents	Individual User	Provides functionality to search for files using filters and tags.	User has uploaded or shared files in their account.	Search results display matching files.
Real-Time Sync	Team Member, Individual User	Ensures changes to files are updated across devices in real-time.	User has an active internet connection.	All linked devices show the latest version of the files.
Collaborate on Files	Team Member	Allows users to share and co-edit files with team members.	The file is uploaded and shared with appropriate permissions.	Shared files are updated with changes from collaborators.
Manage Permissions	Administrator	Controls access to files and manages user roles within shared folders or files.	The administrator has the required permissions to manage access.	Permissions are updated, and access is granted or restricted as specified.
Secure Data	Administrator, Individual User	Implements encryption and secures cloud storage for sensitive documents.	User has uploaded files and opted for secure storage.	Files are encrypted and securely stored.

Data Flow Diagrams

1. Context Diagram: This diagram illustrates the interaction between the user and the system, showing the main processes of file management.
2. Level 1 Data Flow Diagram: This diagram breaks down the main processes into sub-processes, detailing the flow of data within the system. It includes steps such as Document upload, metadata generation, categorization results, retrieval requests, and the final document.



```
+-------------------+     +------------------+     +------------------+
|  User Input/      |----->| LightViewer App |----->| Document         |
|  Interaction      |     | (UI & Backend)   |     | Management       |
|  (PIN, Search,    |     |                  |     | (Storage &       |
|   Replace)        |     |                  |     |  Retrieval)      |
+-------------------+     +------------------+     +------------------+
          |                        |                        |
          |                        v                        v
          |               +------------------+     +------------------+
          |               |  Search & Replace |----->|  Cross-Platform  |
          |               |  (Regular Search) |     |  Collaboration   |
          |               |                  |     |  (Real-Time Editing|
          |               +------------------+     |   & Sharing)      |
          |                        |               +------------------+
          |                        |
          |                        v
          |                +------------------+
          |                | Offline Access &  |
          |                | Syncing Mechanism |
          |                |                  |
          |                +------------------+
          |                        |
          |                        v
          |                +------------------+
          |                | Enterprise        |
          |                | Scalability (Access|
          |                | Controls, Large-   |
          |                | Scale Management)  |
          |                +------------------+
          |
          |
          v
+----------------------+
|   Data Security      |
|   (PIN, Encryption)  |
+----------------------+
```

3.2 System Design Introduction

System design involves defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. The design process includes both high-level design and detailed design.

3.2.1 Control Flow Diagram:

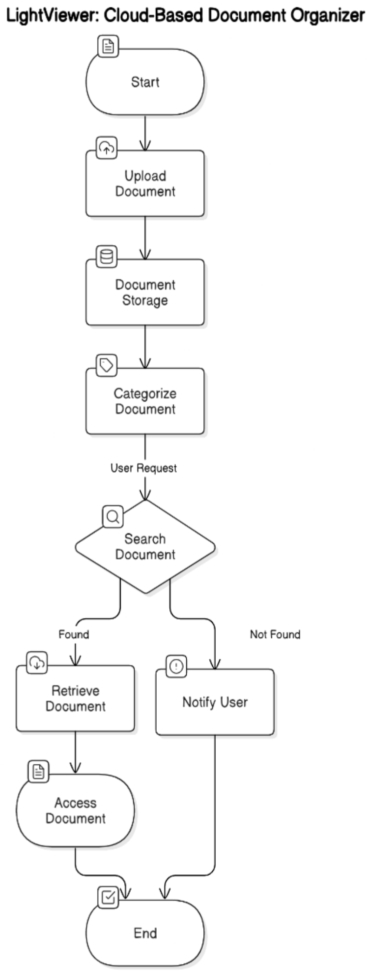


Fig 3.2.1 Control Flow diagram

CHAPTER 4

IMPLEMENTATION

4.1 **Modules**

1. Document Upload and Categorization Module:
   - Allows users to upload documents to the cloud
   - Enables tagging and categorization of documents using predefined or user-generated metadata
   - Validates file formats and sizes during the upload process

2. Real-Time Synchronization Module:
   - Monitors and synchronizes changes to documents and metadata across connected devices in real-time
   - Ensures consistency and eliminates version conflicts
   - Utilizes AWS Lambda and DynamoDB Streams for event-driven updates

3. Search and Retrieval Module:
   - Provides advanced search capabilities using metadata such as tags, categories, file types, and creation dates
   - Implements efficient indexing techniques for rapid document retrieval
   - Supports filtering and keyword-based searches

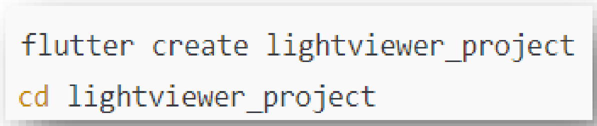
4. Security and Authentication Module:
   - Facilitates secure user authentication using AWS Cognito
   - Implements role-based access controls to ensure proper data protection
   - Encrypts user data and documents during storage and transmission

5. User Interface Module:
   - Offers a responsive and user-friendly design for seamless navigation
   - Includes features for document uploads, metadata tagging, and file searching
   - Displays synchronization status and error messages clearly

6. Error Handling and Notifications Module:
   - Detects and resolves issues such as invalid file formats, upload errors, or synchronization failures
   - Provides real-time feedback to users with actionable error messages and alerts

4.2 Setting Up the LightViewer Project

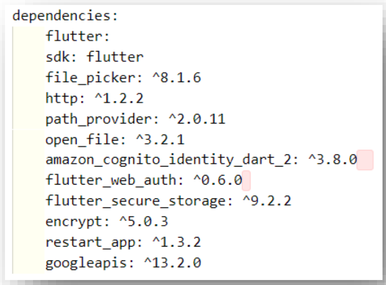
1. **Create a New Flutter Project**:
   Begin by setting up a new Flutter project. This involves initializing the project directory and configuring the required files for the application.



```
flutter create lightviewer_project
cd lightviewer_project
```

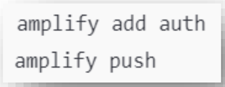
2. **Integrate Core Modules**:
   Create separate Dart files or directories within the lib/ folder for each core module of the application (e.g., authentication, file management, synchronization, and search). These will house the business logic and UI components for the respective features.

3. **Configure Dependencies**:
   Open the ".yaml" file and add the required dependencies for backend integration, UI design, and AWS service interaction.
   Example dependencies:

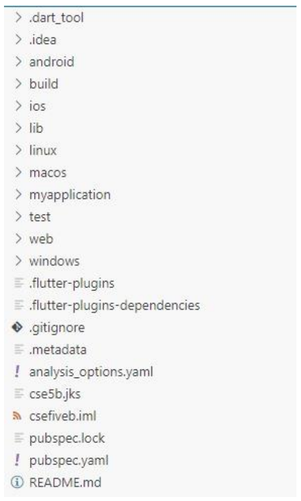


```
dependencies:
    flutter:
    sdk: flutter
    file_picker: ^8.1.6
    http: ^1.2.2
    path_provider: ^2.0.11
    open_file: ^3.2.1
    amazon_cognito_identity_dart_2: ^3.8.0
    flutter_web_auth: ^0.6.0
    flutter_secure_storage: ^9.2.2
    encrypt: ^5.0.3
    restart_app: ^1.3.2
    googleapis: ^13.2.0
```

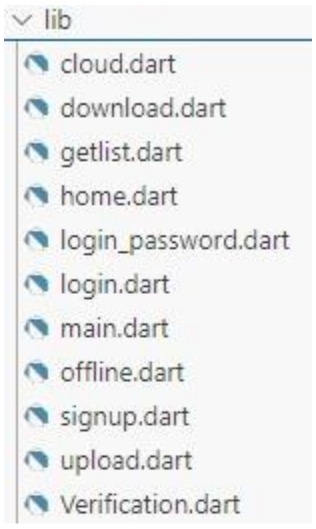
4. **Set Up Authentication**:
   Configure AWS Cognito for user registration, login, and secure access management. Use the AWS Amplify CLI to set up authentication services.



```
amplify add auth
amplify push
```



```
> .dart_tool
> .idea
> android
> build
> ios
> lib
> linux
> macos
> myapplication
> test
> web
> windows
  .flutter-plugins
  .flutter-plugins-dependencies
  .gitignore
  .metadata
! analysis_options.yaml
  cse5b.jks
  csefiveb.iml
  pubspec.lock
! pubspec.yaml
  README.md
```

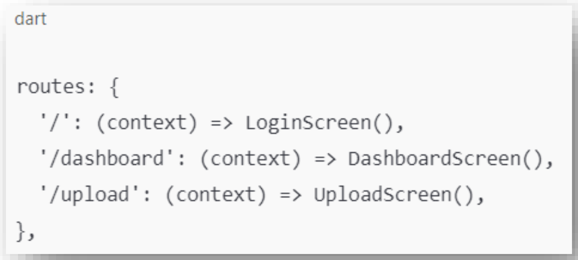


```
v lib
  cloud.dart
  download.dart
  getlist.dart
  home.dart
  login_password.dart
  login.dart
  main.dart
  offline.dart
  signup.dart
  upload.dart
  Verification.dart
```

Project directory *source files*

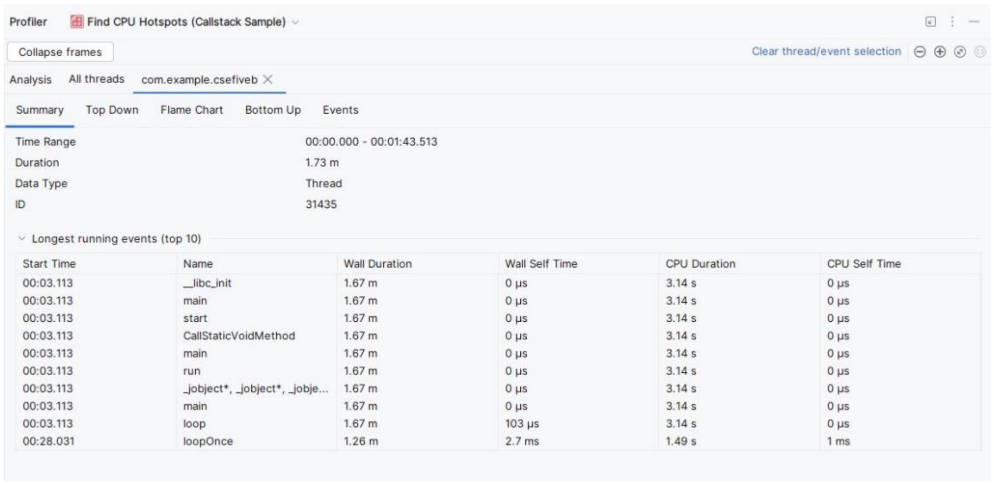
5. **Integrate Backend Services**:
   - Configure AWS Lambda functions using Python for serverless operations like metadata updates and real-time synchronization
   - Use AWS DynamoDB for metadata storage and search functionalities

6. **Define Navigation and Routing**:
   Set up navigation for the application using Flutter's navigation system. Define routes for key screens such as registration, login, dashboard, upload, and search functionalities.



```dart
routes: {
  '/': (context) => LoginScreen(),
  '/dashboard': (context) => DashboardScreen(),
  '/upload': (context) => UploadScreen(),
},
```

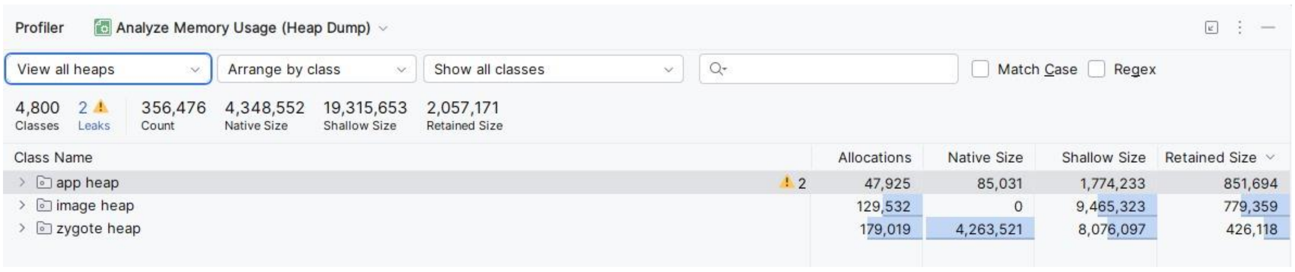
7. **UI Design and Testing**:
   Use Flutter's widgets to design a responsive and intuitive user interface. Test the application on both Android and iOS platforms to ensure compatibility and performance.
   Evaluating the app's performance is crucial to ensuring its reliability and responsiveness. This objective focuses on analyzing the usability, speed, and reliability of LightViewer in managing and organizing documents. Comparative analysis with traditional file systems highlights its advantages in terms of efficiency and user experience.



Time Range	00:00.000 - 00:01:43.513
Duration	1.73 m
Data Type	Thread
ID	31435

Start Time	Name	Wall Duration	Wall Self Time	CPU Duration	CPU Self Time
00:03.113	__libc_init	1.67 m	0 μs	3.14 s	0 μs
00:03.113	main	1.67 m	0 μs	3.14 s	0 μs
00:03.113	start	1.67 m	0 μs	3.14 s	0 μs
00:03.113	CallStaticVoidMethod	1.67 m	0 μs	3.14 s	0 μs
00:03.113	main	1.67 m	0 μs	3.14 s	0 μs
00:03.113	run	1.67 m	0 μs	3.14 s	0 μs
00:03.113	_jobject*, _jobject*, _jobje...	1.67 m	0 μs	3.14 s	0 μs
00:03.113	main	1.67 m	0 μs	3.14 s	0 μs
00:03.113	loop	1.67 m	103 μs	3.14 s	0 μs
00:28.031	loopOnce	1.26 m	2.7 ms	1.49 s	1 ms

CPU performance evaluation, using Profiler : Android Studio



4,800	2 ⚠	356,476	4,348,552	19,315,653	2,057,171
Classes	Leaks	Count	Native Size	Shallow Size	Retained Size

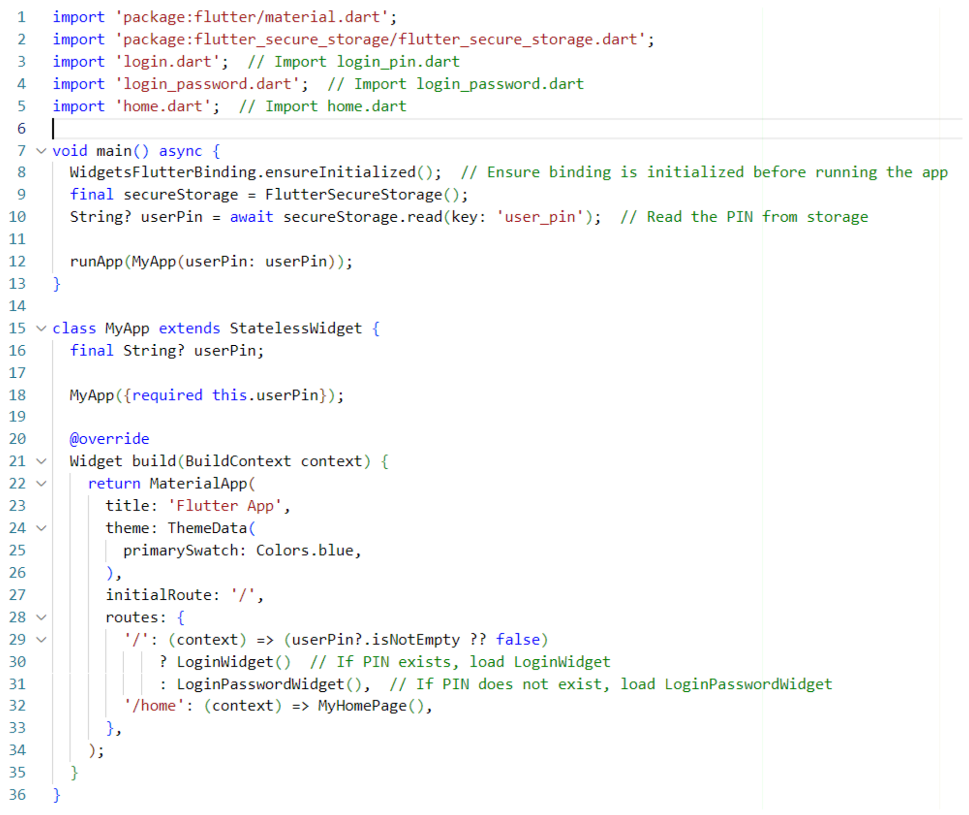
Class Name		Allocations	Native Size	Shallow Size	Retained Size
app heap	⚠ 2	47,925	85,031	1,774,233	851,694
image heap		129,532	0	9,465,323	779,359
zygote heap		179,019	4,263,521	8,076,097	426,118

Memory Usage Analysis , using Profiler: Android Studio

8. **Deploy and Test**:
   Deploy the backend services to AWS and connect them to the Flutter application. Test all functionalities, including real-time synchronization, file uploads, search, and authentication, to ensure the system operates as expected.

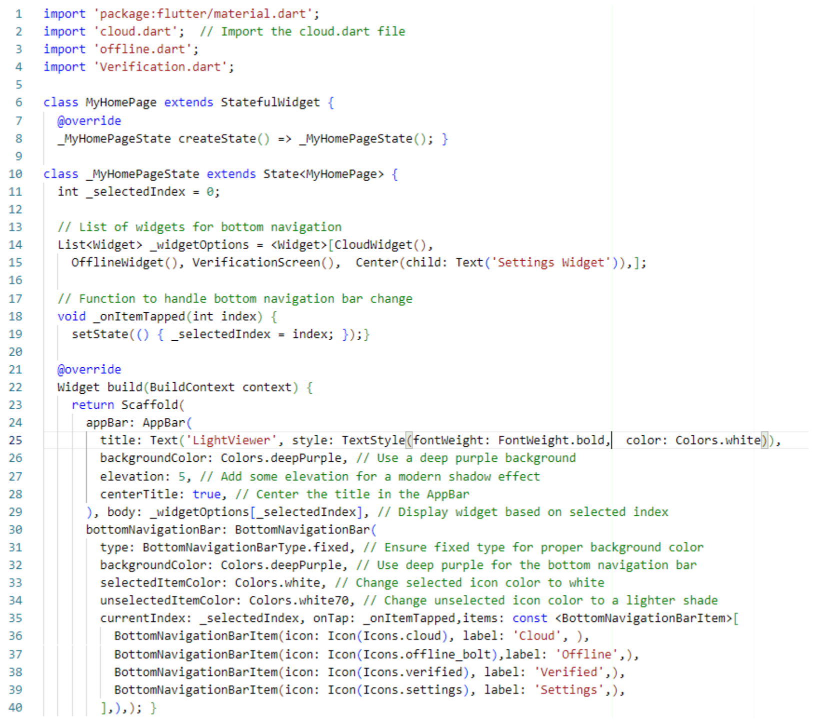
4.3 Source code:

1. Main.dart



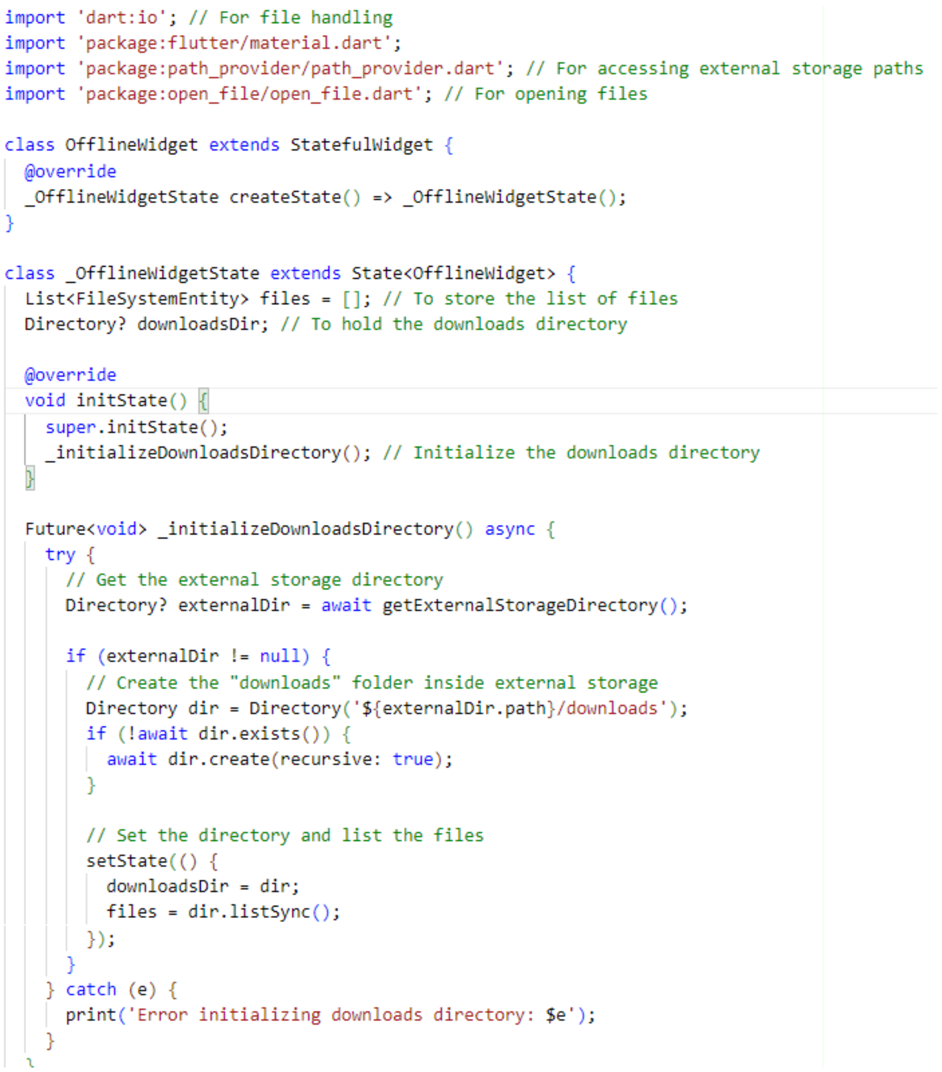
```dart
import 'package:flutter/material.dart';
import 'package:flutter_secure_storage/flutter_secure_storage.dart';
import 'login.dart';  // Import login_pin.dart
import 'login_password.dart';  // Import login_password.dart
import 'home.dart';  // Import home.dart

void main() async {
  WidgetsFlutterBinding.ensureInitialized();  // Ensure binding is initialized before running the app
  final secureStorage = FlutterSecureStorage();
  String? userPin = await secureStorage.read(key: 'user_pin');  // Read the PIN from storage

  runApp(MyApp(userPin: userPin));
}

class MyApp extends StatelessWidget {
  final String? userPin;

  MyApp({required this.userPin});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      initialRoute: '/',
      routes: {
        '/': (context) => (userPin?.isNotEmpty ?? false)
            ? LoginWidget()  // If PIN exists, load LoginWidget
            : LoginPasswordWidget(),  // If PIN does not exist, load LoginPasswordWidget
        '/home': (context) => MyHomePage(),
      },
    );
  }
}
```

2. Home.dart

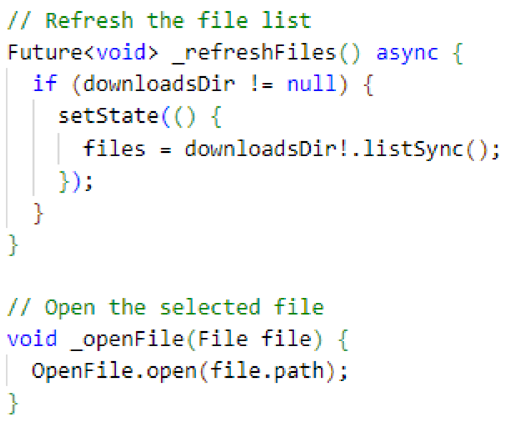


```dart
import 'package:flutter/material.dart';
import 'cloud.dart';  // Import the cloud.dart file
import 'offline.dart';
import 'Verification.dart';

class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => _MyHomePageState(); }

class _MyHomePageState extends State<MyHomePage> {
  int _selectedIndex = 0;

  // List of widgets for bottom navigation
  List<Widget> _widgetOptions = <Widget>[CloudWidget(),
    OfflineWidget(), VerificationScreen(),  Center(child: Text('Settings Widget')),];

  // Function to handle bottom navigation bar change
  void _onItemTapped(int index) {
    setState(() { _selectedIndex = index; });}

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('LightViewer', style: TextStyle(fontWeight: FontWeight.bold,  color: Colors.white)),
        backgroundColor: Colors.deepPurple, // Use a deep purple background
        elevation: 5, // Add some elevation for a modern shadow effect
        centerTitle: true, // Center the title in the AppBar
      ), body: _widgetOptions[_selectedIndex], // Display widget based on selected index
      bottomNavigationBar: BottomNavigationBar(
        type: BottomNavigationBarType.fixed, // Ensure fixed type for proper background color
        backgroundColor: Colors.deepPurple, // Use deep purple for the bottom navigation bar
        selectedItemColor: Colors.white, // Change selected icon color to white
        unselectedItemColor: Colors.white70, // Change unselected icon color to a lighter shade
        currentIndex: _selectedIndex, onTap: _onItemTapped,items: const <BottomNavigationBarItem>[
          BottomNavigationBarItem(icon: Icon(Icons.cloud), label: 'Cloud', ),
          BottomNavigationBarItem(icon: Icon(Icons.offline_bolt),label: 'Offline',),
          BottomNavigationBarItem(icon: Icon(Icons.verified), label: 'Verified',),
          BottomNavigationBarItem(icon: Icon(Icons.settings), label: 'Settings',),
        ],),); }
```

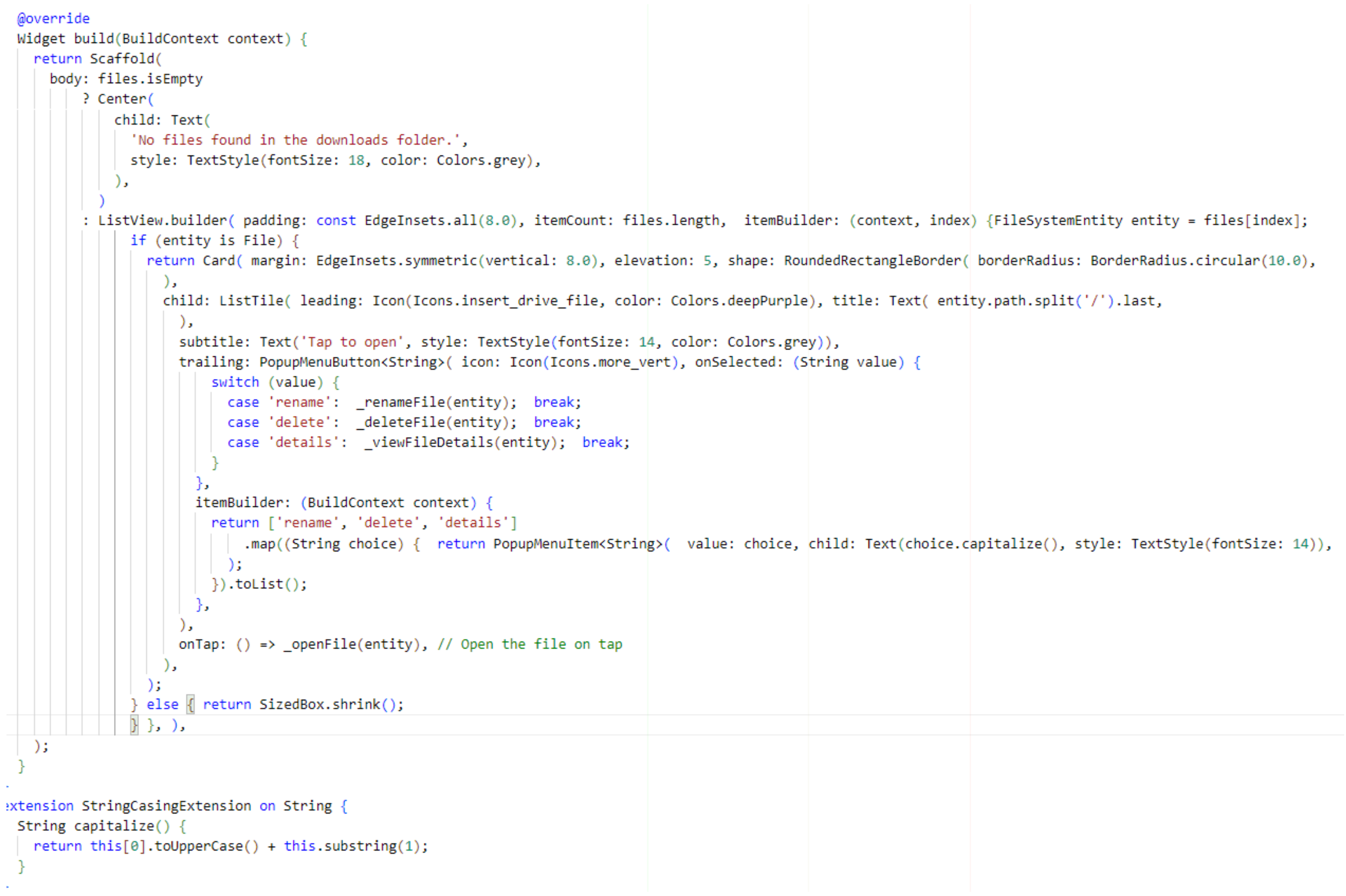
3. Offline.dart



```dart
import 'dart:io'; // For file handling
import 'package:flutter/material.dart';
import 'package:path_provider/path_provider.dart'; // For accessing external storage paths
import 'package:open_file/open_file.dart'; // For opening files

class OfflineWidget extends StatefulWidget {
  @override
  _OfflineWidgetState createState() => _OfflineWidgetState();
}

class _OfflineWidgetState extends State<OfflineWidget> {
  List<FileSystemEntity> files = []; // To store the list of files
  Directory? downloadsDir; // To hold the downloads directory

  @override
  void initState() {
    super.initState();
    _initializeDownloadsDirectory(); // Initialize the downloads directory
  }

  Future<void> _initializeDownloadsDirectory() async {
    try {
      // Get the external storage directory
      Directory? externalDir = await getExternalStorageDirectory();

      if (externalDir != null) {
        // Create the "downloads" folder inside external storage
        Directory dir = Directory('${externalDir.path}/downloads');
        if (!await dir.exists()) {
          await dir.create(recursive: true);
        }

        // Set the directory and list the files
        setState(() {
          downloadsDir = dir;
          files = dir.listSync();
        });
      }
    } catch (e) {
      print('Error initializing downloads directory: $e');
    }
```

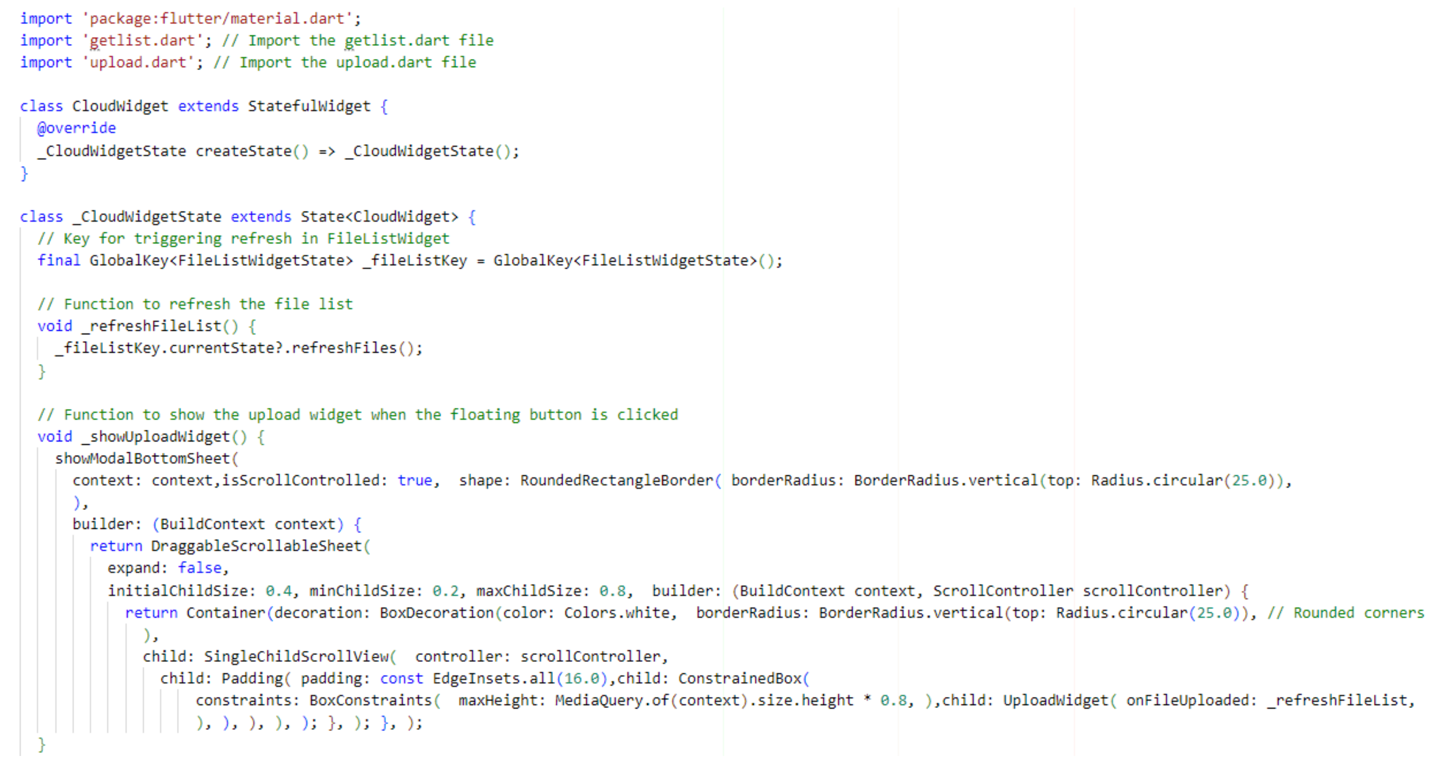


```dart
// Refresh the file list
Future<void> _refreshFiles() async {
  if (downloadsDir != null) {
    setState(() {
      files = downloadsDir!.listSync();
    });
  }
}

// Open the selected file
void _openFile(File file) {
  OpenFile.open(file.path);
}
```



```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: files.isEmpty
        ? Center(
          child: Text(
            'No files found in the downloads folder.',
            style: TextStyle(fontSize: 18, color: Colors.grey),
          ),
        )
        : ListView.builder( padding: const EdgeInsets.all(8.0), itemCount: files.length,  itemBuilder: (context, index) {FileSystemEntity entity = files[index];
            if (entity is File) {
              return Card( margin: EdgeInsets.symmetric(vertical: 8.0), elevation: 5, shape: RoundedRectangleBorder( borderRadius: BorderRadius.circular(10.0),
                ),
                child: ListTile( leading: Icon(Icons.insert_drive_file, color: Colors.deepPurple), title: Text( entity.path.split('/').last,
                  ),
                  subtitle: Text('Tap to open', style: TextStyle(fontSize: 14, color: Colors.grey)),
                  trailing: PopupMenuButton<String>( icon: Icon(Icons.more_vert), onSelected: (String value) {
                      switch (value) {
                        case 'rename':  _renameFile(entity);  break;
                        case 'delete':  _deleteFile(entity);  break;
                        case 'details':  _viewFileDetails(entity);  break;
                      }
                    },
                    itemBuilder: (BuildContext context) {
                      return ['rename', 'delete', 'details']
                          .map((String choice) {  return PopupMenuItem<String>(  value: choice, child: Text(choice.capitalize(), style: TextStyle(fontSize: 14)),
                        );
                      }).toList();
                    },
                  ),
                  onTap: () => _openFile(entity), // Open the file on tap
                ),
              );
            } else { return SizedBox.shrink();
            } }, ),
  );
}

xtension StringCasingExtension on String {
  String capitalize() {
    return this[0].toUpperCase() + this.substring(1);
  }
```

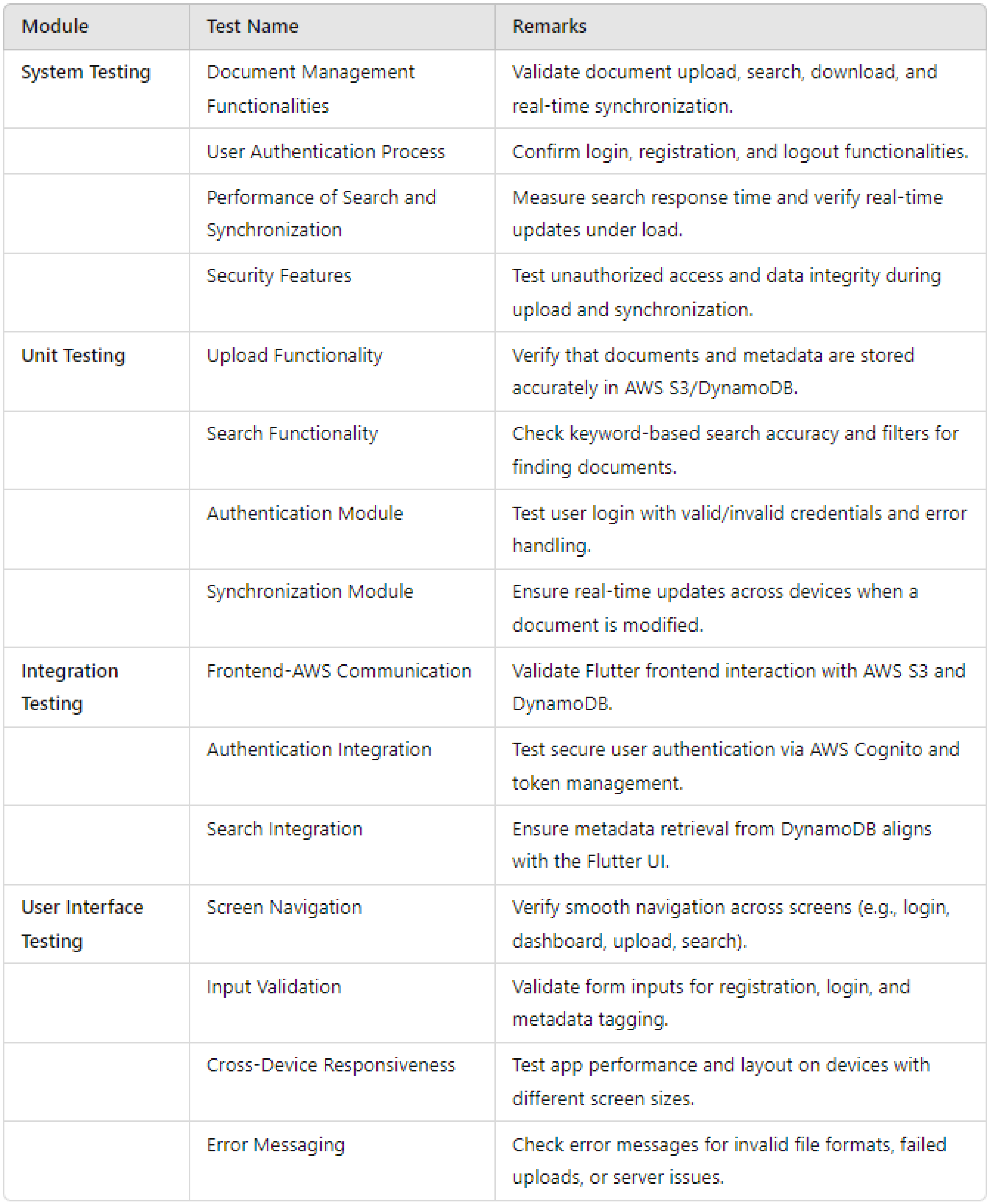
4. Cloud.dart



```
import 'package:flutter/material.dart';
import 'getlist.dart'; // Import the getlist.dart file
import 'upload.dart'; // Import the upload.dart file

class CloudWidget extends StatefulWidget {
  @override
  _CloudWidgetState createState() => _CloudWidgetState();
}

class _CloudWidgetState extends State<CloudWidget> {
  // Key for triggering refresh in FileListWidget
  final GlobalKey<FileListWidgetState> _fileListKey = GlobalKey<FileListWidgetState>();

  // Function to refresh the file list
  void _refreshFileList() {
    _fileListKey.currentState?.refreshFiles();
  }

  // Function to show the upload widget when the floating button is clicked
  void _showUploadWidget() {
    showModalBottomSheet(
      context: context,isScrollControlled: true,  shape: RoundedRectangleBorder( borderRadius: BorderRadius.vertical(top: Radius.circular(25.0)),
      ),
      builder: (BuildContext context) {
        return DraggableScrollableSheet(
          expand: false,
          initialChildSize: 0.4, minChildSize: 0.2, maxChildSize: 0.8,  builder: (BuildContext context, ScrollController scrollController) {
            return Container(decoration: BoxDecoration(color: Colors.white,  borderRadius: BorderRadius.vertical(top: Radius.circular(25.0)), // Rounded corners
              ),
              child: SingleChildScrollView(  controller: scrollController,
                child: Padding( padding: const EdgeInsets.all(16.0),child: ConstrainedBox(
                    constraints: BoxConstraints(  maxHeight: MediaQuery.of(context).size.height * 0.8, ),child: UploadWidget( onFileUploaded: _refreshFileList,
                    ), ), ), ), ); }, ); }, );
  }
```

CHAPTER 5

TESTING



Module	Test Name	Remarks
System Testing	Document Management Functionalities	Validate document upload, search, download, and real-time synchronization.
	User Authentication Process	Confirm login, registration, and logout functionalities.
	Performance of Search and Synchronization	Measure search response time and verify real-time updates under load.
	Security Features	Test unauthorized access and data integrity during upload and synchronization.
Unit Testing	Upload Functionality	Verify that documents and metadata are stored accurately in AWS S3/DynamoDB.
	Search Functionality	Check keyword-based search accuracy and filters for finding documents.
	Authentication Module	Test user login with valid/invalid credentials and error handling.
	Synchronization Module	Ensure real-time updates across devices when a document is modified.
Integration Testing	Frontend-AWS Communication	Validate Flutter frontend interaction with AWS S3 and DynamoDB.
	Authentication Integration	Test secure user authentication via AWS Cognito and token management.
	Search Integration	Ensure metadata retrieval from DynamoDB aligns with the Flutter UI.
User Interface Testing	Screen Navigation	Verify smooth navigation across screens (e.g., login, dashboard, upload, search).
	Input Validation	Validate form inputs for registration, login, and metadata tagging.
	Cross-Device Responsiveness	Test app performance and layout on devices with different screen sizes.
	Error Messaging	Check error messages for invalid file formats, failed uploads, or server issues.

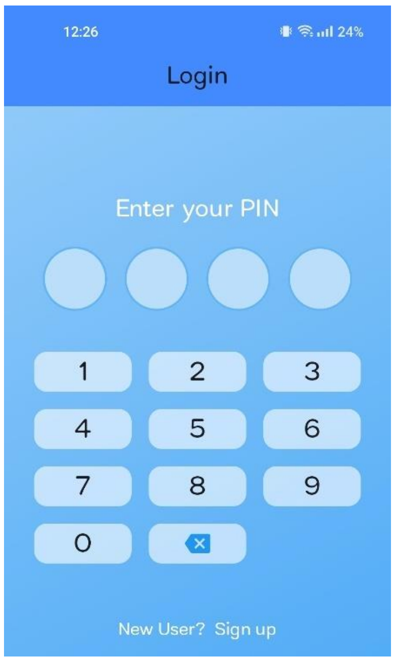
CHAPTER 6

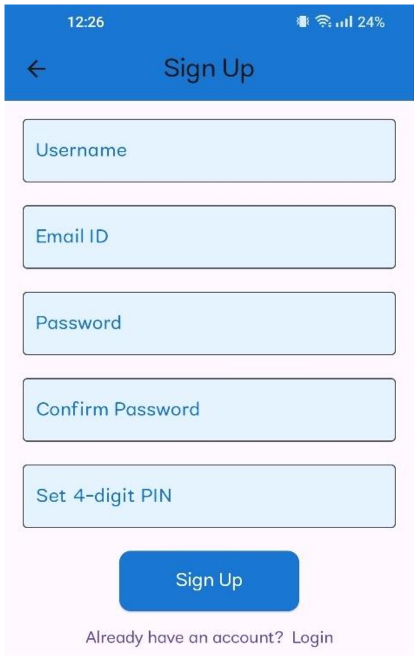
RESULT ANALYSIS AND SCREENSHOTS

6.1 Result Analysis:

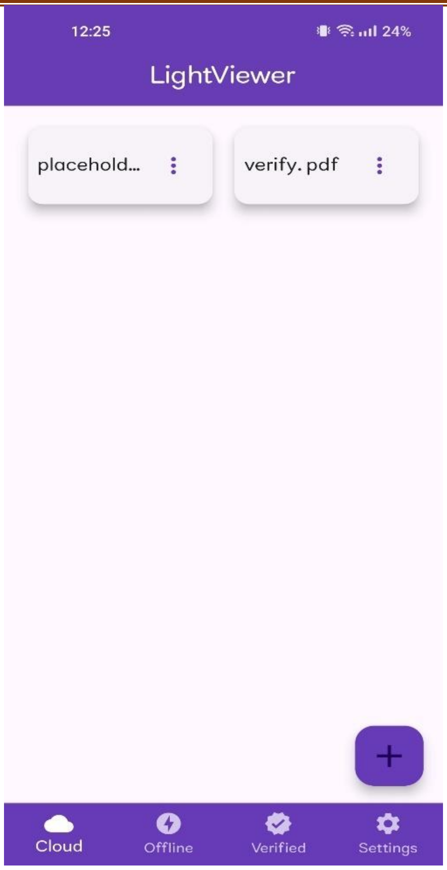
1. Performance Evaluation
   - Document upload and retrieval times were tested under various network conditions. Results indicate an average upload time of 3 seconds for files under 5MB and search results displayed within 2 seconds
   - Real-time synchronization across devices was achieved with negligible latency (<1 second)
2. Functional Accuracy
   - Upload Functionality: All tested files were uploaded successfully and correctly tagged with metadata
   - Search Functionality: Accurate results were retrieved using keywords and filters
   - Authentication: Successful user login, logout, and password recovery in all test cases
3. User Feedback
   - The application interface was rated highly for ease of use and responsiveness during user testing sessions
4. System Robustness
   - No data corruption or synchronization errors were observed during stress testing with simultaneous uploads/downloads

6.2 Screenshots:

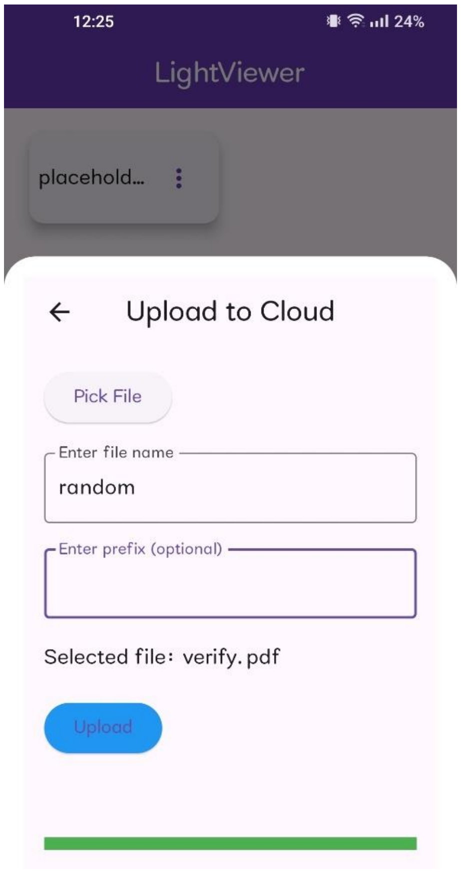




Login Screen: Description- Allows users to enter credentials for secure access.



Dashboard: Displays uploaded documents with options for search, upload, and categorization.

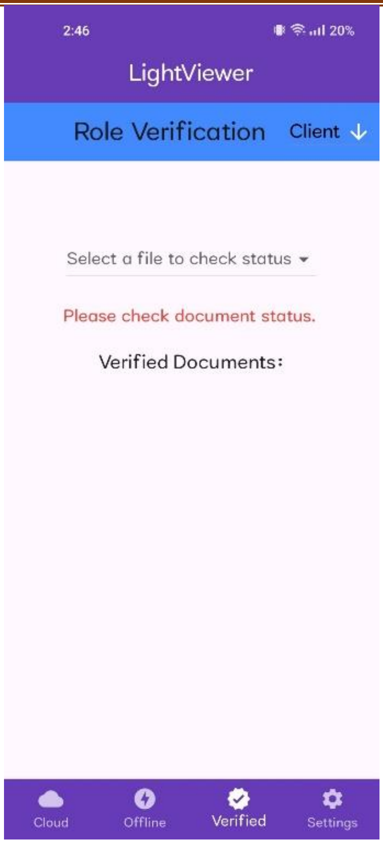


Enter file name: random

Enter prefix (optional):

Selected file: verify. pdf

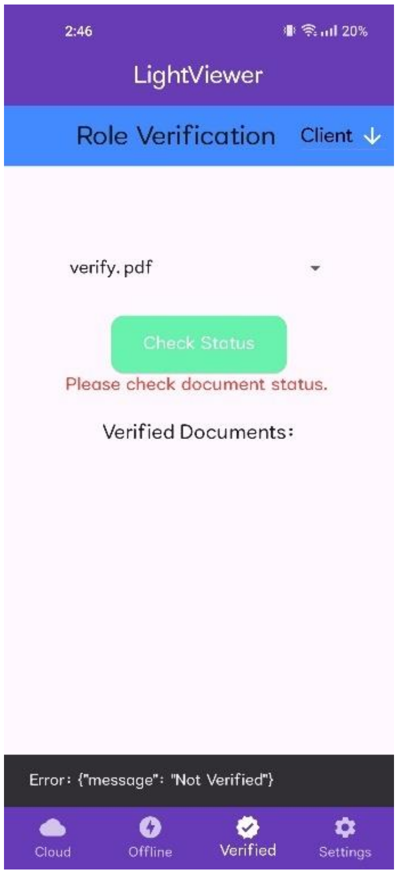
Upload Screen: Enables users to select and upload files while tagging them with metadata.



Please check document status.

Verified Documents:

Document Verification: Shows the list of the documents verified.



Please check document status.

Verified Documents:

```
Error: {"message": "Not Verified"}
```

Error Handling: Displays error messages for invalid inputs or unsupported actions.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion:

In today's fast-paced digital era, efficient document management poses a significant challenge for individuals and organizations. LightViewer addresses this issue through its robust cloud-based architecture, real-time synchronization, and intuitive design. Developed using Flutter, the application ensures cross-platform compatibility, providing seamless access on both Android and iOS devices. On the backend, LightViewer harnesses the power of AWS services, including EC2, S3, Cognito, and DynamoDB, to deliver a scalable, secure, and reliable platform capable of managing large volumes of data with ease. Its real-time synchronization feature ensures users always work with the most up-to-date versions of their documents, fostering collaboration and eliminating version conflicts. The advanced search functionality, powered by DynamoDB, enables swift retrieval of documents using metadata, significantly enhancing productivity by reducing time spent navigating disorganized files. Performance evaluations reveal that LightViewer outperforms traditional document management systems, which often lack essential features such as cloud integration, real-time updates, and efficient search capabilities. With its user-friendly design, speed, and reliability, LightViewer proves to be a transformative tool for streamlining workflows and boosting productivity for individuals and teams alike. As digital data continues to grow exponentially, innovative solutions like LightViewer will play a pivotal role in optimizing document management, paving the way for further advancements in the field and delivering substantial benefits to users worldwide.

Future Enhancements:

LightViewer is poised for significant growth with several promising future enhancements that will elevate its functionality and address evolving user demands. One key area of development is the incorporation of advanced security features, such as biometric authentication methods like fingerprint and facial recognition. These measures will not only enhance security but also improve user convenience by providing seamless access. Another exciting enhancement is the introduction of AI-powered search, which will leverage natural language processing to allow users to perform intuitive, context-aware searches. This will enable users to locate documents using conversational queries, streamlining the search process and making it more efficient. Additionally, LightViewer plans to introduce offline access capabilities, giving users the flexibility to manage and view their documents even without an internet connection. Once reconnected, the platform will automatically synchronize changes, ensuring smooth continuity of work. In line with the growing trend of collaboration, LightViewer will also expand its features to support cross-platform, real-time collaborative editing and document sharing, enabling multiple users to work together seamlessly on the same document from different devices. Furthermore, to cater to the needs of large organizations, the platform's architecture will be enhanced to support enterprise scalability. This will include advanced access controls, robust user management, and the ability to handle large volumes of documents effectively. With these advancements, LightViewer will continue to solidify its position as a critical tool for modern document management, offering users increased flexibility, security, and collaboration capabilities, while fostering innovation and addressing the challenges of the digital workspace.

BIBLIOGRAPHY

[1] Shivam Bundele, Abhishek Nilkhan, Swadhin Das, Rutuja Kadu, “A Blockchain-Based Verification System for Academic Certificates,” IJARSCT, May 2023

[2] Jaswanth Alahari, Dasaiah Pakanati, Harshita Cherukuri, Om Goel, Prof. (Dr.) Arpit Jain, “Best Practices for Integrating OAuth in Mobile Applications for Secure Authentication,” Shodh Sagar, Oct. 2023

[3] J. Han, C. Wang, J. Miao, M. Lu, Y. Wang, and J. Shi,” Research on Electronic Document Management System Based on Cloud Computing,” Computers, Materials & Continua, vol. 69, no. 1, pp. 289-306, 2021.

[4] S. Talluri and S. T. Makani,” Managing Identity and Access Manage ment (IAM) in Amazon Web Services (AWS),” Journal of Artificial Intelligence & Cloud Computing, vol. 2, no. 1, pp. 1-5, Feb. 2023, doi: 10.47363/JAICC/2023(2)147.

[5] A. Ikuomola, E. A. Oyekan, and O. M. Orogbemi, ”A Secured Cloud Based Electronic Document Management System,” International Journal of Innovative Research and Development, vol. 11, no. 6, pp. 123-130, Dec. 2022

[6] AWS, "Amazon EC2 Documentation," [Online]. Available: https://docs.aws.amazon.com/ec2/

[7] AWS, "Amazon S3 Documentation," [Online]. Available: https://docs.aws.amazon.com/s3/

[8] AWS, "Amazon Cognito Documentation," [Online]. Available:https://docs.aws.amazon.com/cognito/

[9] AWS, "Amazon DynamoDB Documentation,"[Online]. Available: docs.aws.amazon.com/dynamodb