SWENG 837: Software Design Course Project

Object-Oriented Software System Design for Cloud-Based File Storage and Collaboration

by David Luo

Problem Statement

In the current workplace environment, teams and individuals commonly face difficulties in work caused by inefficient collaboration and disorganized file management. Tools that attempt to workaround these problems typically rely on a combination of local storage, email attachments, and third-party cloud services. These tools are widely used and effective to a limited extent. However, issues are still experienced by users such as version conflicts and connectivity issues which can result in decreased productivity and workflow disruptions.

The primary challenge in designing the system is creating a scalable, intuitive, and secure cloud-based storage system that enables seamless file organization, controlled access management, and real-time collaboration. Central elements will especially need to be addressed, including data structure optimization, role-based access control, file editing conflict resolution, and integration with external tools. By focusing on these design elements, the access of data can be enhanced, along with document consistency and the creation of a unified workspace for efficient team collaboration.

Design Solution

The proposed system for the Cloud-Based File Storage and Collaboration Tool is a scalable, secure, and intuitive platform with a high-level architecture that is modular and service-oriented.

Key System Components:

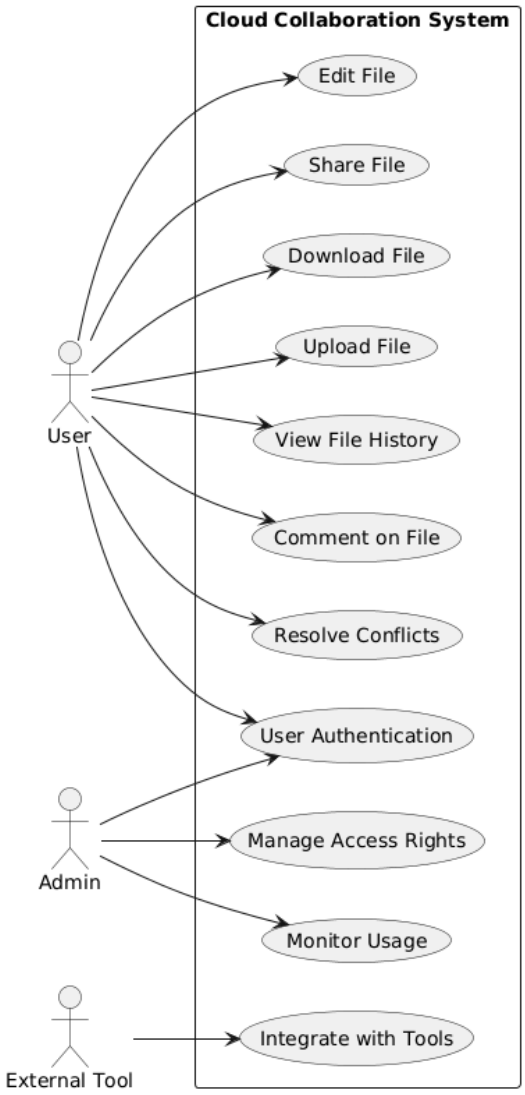
- Authentication Service: Handles user sessions
- Access Control Module: Handles role-based access management
- File Management Module: Handles file uploading/downloading
- Notification Service: Sends alerts to users
- Collaboration Engine: Handles session synchronization and real-time editing
- Database: Stores file metadata/history and access controls, among other objects
- Integration Layer: Allows for third-party API connectivity with the system
- Cloud Environment: Optimizes performance and availability

Design Patterns:

- GRASP, SOLID, GoF, Microservices to promote scalability, maintainability, and flexibility

System Design using Domain Modeling

1. UML Use Case Diagram

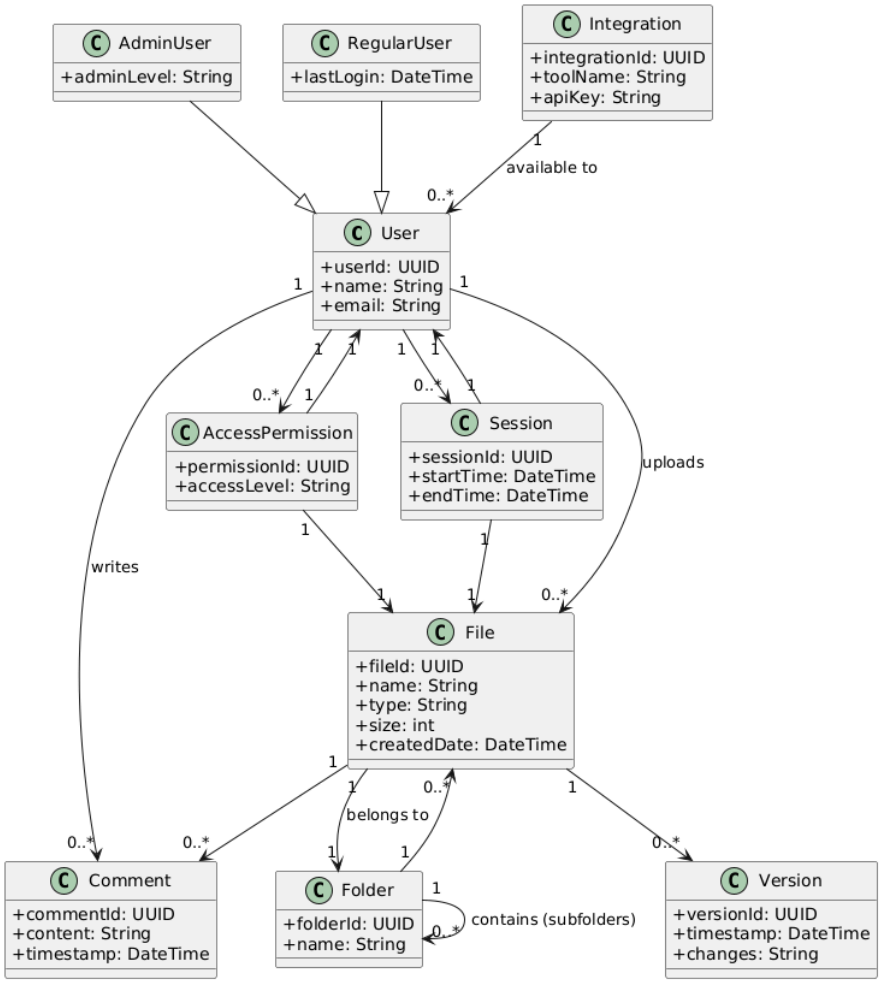


The above Use Case Diagram illustrates the interactions between actors (e.g. users and external systems) and the Cloud-Based File Storage and Collaboration Tool. It also identifies the functional requirements of the system by depicting what it should do from the point of view of its users/actors.

Actors:

- A User interacts with the system to edit or access a file
- An Admin is a user with extended privileges responsible for monitoring system activity and managing permissions
- External Tools are third-party applications that are integrated into the system, providing additional functionality

2. UML Domain Model

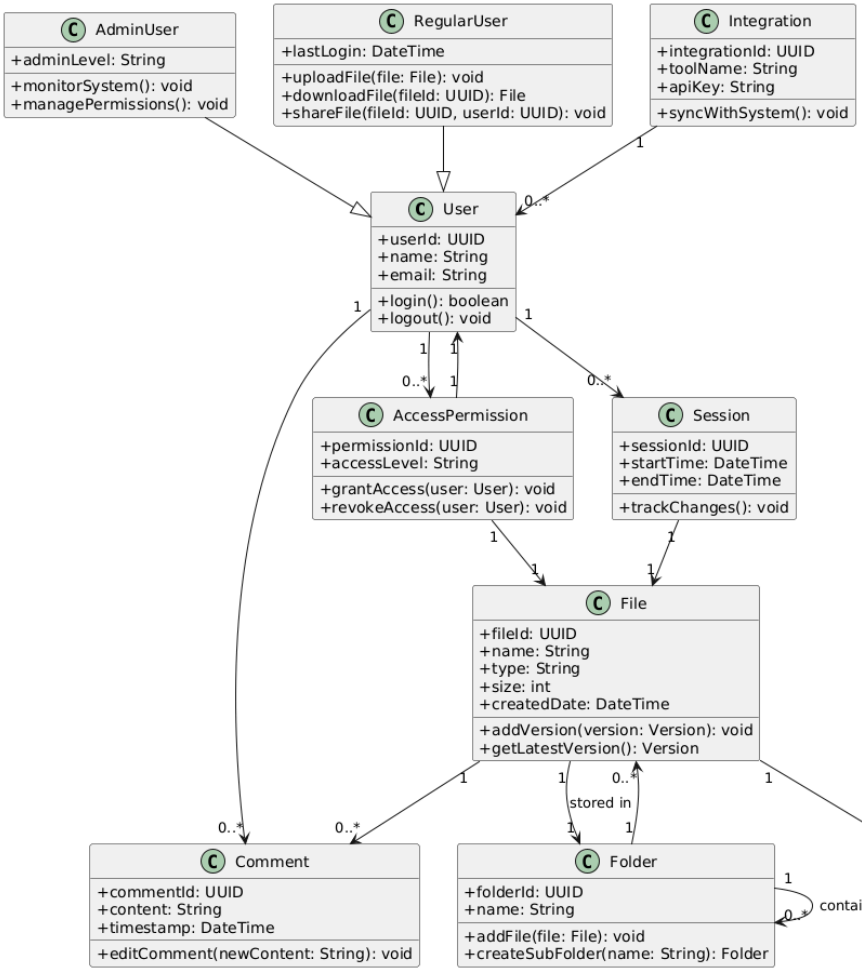


The above Domain Model illustration depicts the key interconnected concepts and relationships within the Cloud-Based File Storage and Collaboration Tool. Acting as the foundation of the system's object-oriented design, it ensures that the system maintains consistency and clarity.

Relationships:

- A User can own/access Files/Folders containing AccessPermissions
- A File belongs to a Folder and can contain Comments and Versions
- Integrations connect the system with external tools, providing additional functionality
- Sessions track user interactions/changes

3. UML Class Diagram

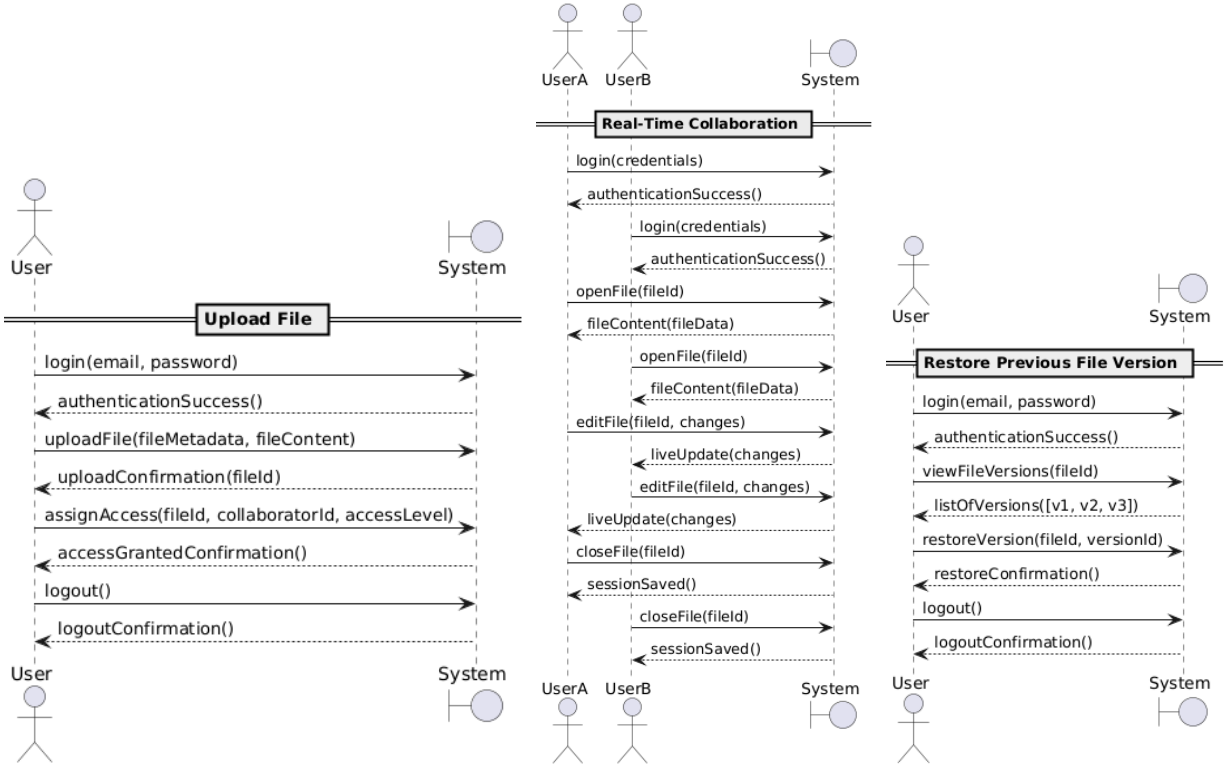


The above Class Diagram represents the overall structure of the Cloud-Based File Storage and Collaboration Tool in terms of its classes, attributes, operations, and relationships. It builds upon the Domain Model by incorporating additional depictions of the system's functionality.

Structure:

- AdminUser extends User with additional permissions to monitor and manage the system
- AccessPermission defines access control (e.g. read/write) while linking User to File

4. UML Sequence Diagrams



The above Sequence Diagrams illustrate the interaction flows between processes and objects in the events of three separate use cases--uploading a file, facilitating real-time collaboration, and restoring a version of a previous file.

Upload File

- User is authenticated and sends a request to upload a file, which is processed and confirmed with a unique fileId
- User assigns access permissions to a collaborator before logging out

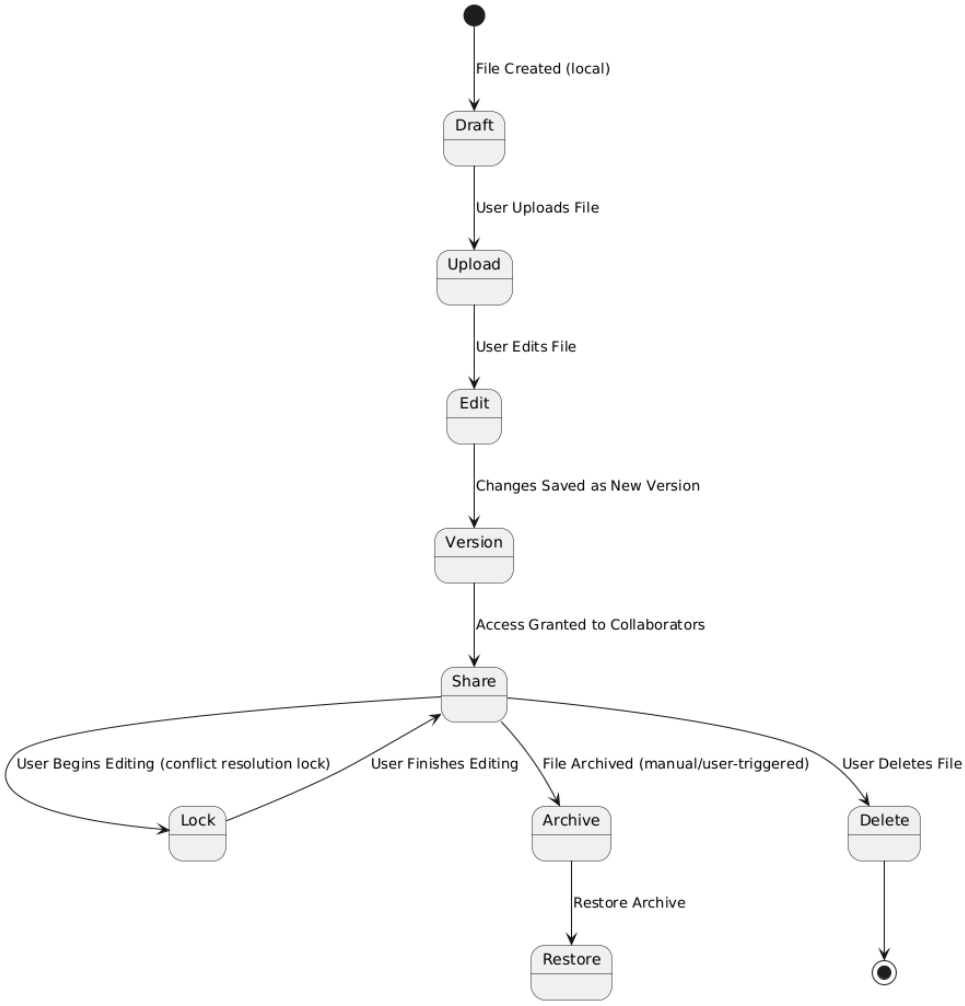
Real-Time Collaboration

- Two users are authenticated; they send requests to open & edit a file, which are processed and confirmed with unique fileIds--these changes are updated live
- When the file is closed, the session is saved for future use

Restore Previous File Version

- User is authenticated and sends a request to view a file version; a list of previous versions is provided for the user
- User then sends a request to revert a file to a selected file version, which is processed and confirmed with a unique fieldId

5. UML State Diagram

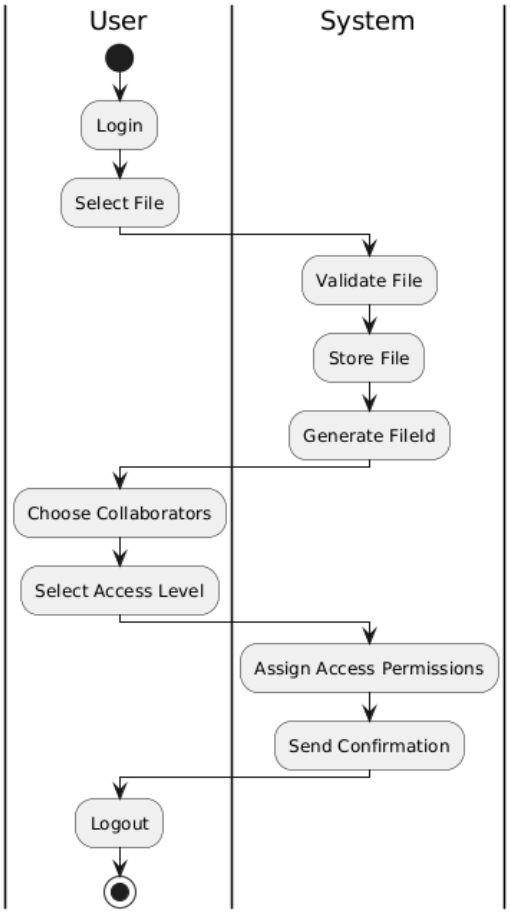


The above State Diagram follows the workflow of states that a file can encounter during its time in the Cloud-Based File Storage and Collaboration Tool.

Key States:

- Draft: File is created locally by the user before being uploaded
- Upload: File is saved by the user within the system
- Edit & Version: User creates an edit to the file, resulting in a new file version
- Share: File is made accessible to other collaborators
- Lock: File is locked for editing temporarily to avoid conflicts
- Archive & Restore: File is marked as inactive but can be restored at a later point
- Delete: Final state of the file after being removed from the system

6. UML Activity Diagram (Swimlane Diagram)



The above Activity Diagram depicts a workflow of activities that typically occurs within the Cloud-Based File Storage and Collaboration Tool. The diagram additionally represents swimlanes that determine the responsibilities of objects.

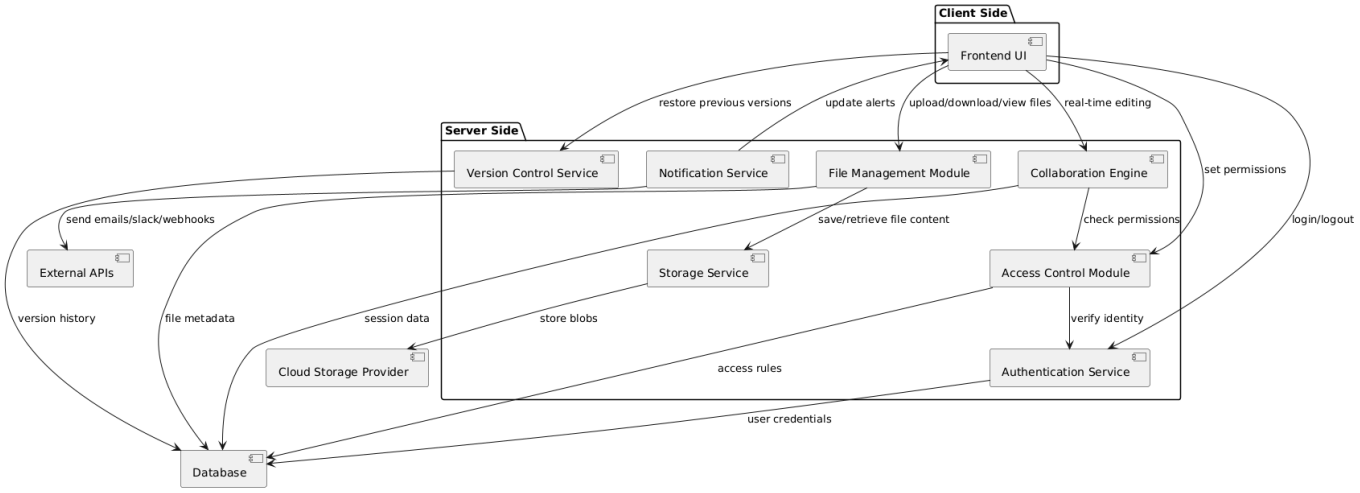
Swimlanes:

- User Lane: includes actions triggered by UI (e.g. login, file select, access level selection)
- System Lane: includes background tasks (e.g. file validation & storage, access control)

Workflow:

- User logs in and authenticates before selecting a file to manage
- System validates and stores file with a unique FileId
- User chooses file collaborators and selects their access levels
- System confirms the user's selection and assigns permissions for collaborators
- User logs out

7. UML Component Diagram

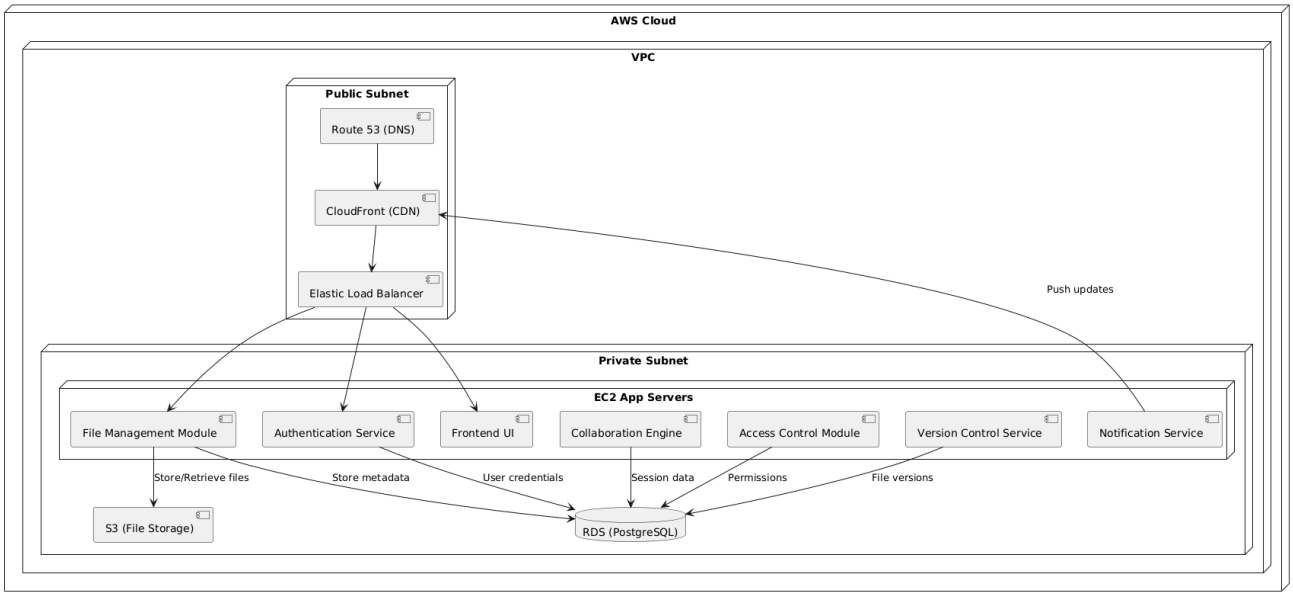


The above Component Diagram represents an overview of the major parts of the Cloud-Based File Storage and Collaboration Tool and their interactions with other objects. Dependency arrows between objects are shown within the diagram and reflect component communications and responsibilities.

Components:

- Frontend UI: User interface for system interaction
- Access Control Module: Handles role-based access management
- File Management Module: Handles file uploading/downloading
- Authentication Service: Handles user sessions
- Notification Service: Sends alerts to users
- Storage Service: Handles file storage logic
- Version Control Service: Handles file versioning and rollbacking
- Collaboration Engine: Handles session synchronization and real-time editing
- Database: Stores file metadata/history and access controls, among other objects
- External Integration APIs: Integrated with the system for application hooks, etc.

8. Cloud Deployment Diagram



The above Cloud Deployment Diagram illustrates how the Cloud-Based File Storage and Collaboration Tool is deployed in a cloud environment--in this case, Amazon Web Services (AWS). The diagram additionally depicts how software components are assigned to cloud infrastructure nodes.

AWS Resources:

- Amazon Route 53: handles DNS resolution
- Amazon CloudFront: distributes static content
- Amazon RDS: handles database management
- Amazon S3: stores file content
- Elastic Load Balancer: distributes incoming traffic
- EC2 App Servers: hosts UI frontend and backend

9. Skeleton Class and Tables Definition

User.java

```
class User {
    String userId;
    String name;
    String email;
    String passwordHash;
    Role role;
    void login(String email,
String password);
    void logout();
    void updateProfile(String
name, String email);
}
```

File.java

```
class File {
    String fileId;
    String fileName;
    String ownerId;
    Date uploadDate;
    String currentVersionId;
    void upload(byte[] content);
    void delete();
    void share(String userId,
PermissionLevel level);
    void restoreVersion(String
versionId);
}
```

FileVersion.java

```
class FileVersion {
    String versionId;
    String fileId;
    Date modifiedDate;
    String modifiedBy;
    byte[] content;
    void createNewVersion(byte[]
newContent);
}
```

Permission.java

```
class Permission {
    String permissionId;
    String fileId;
    String userId;
    PermissionLevel level;
    boolean
hasAccess(PermissionLevel
requiredLevel);
}
```

EditSession.java

```
class EditSession {
    String sessionId;
    String fileId;
    List<String> participantIds;
    Date startedAt;
    Date endedAt;
    void start();
    void syncChanges(String
userId, String delta);
    void end();
}
```

Notification.java

```
class Notification {
    String notificationId;
    String userId;
    String message;
    Date createdAt;
    boolean isRead;
    void markAsRead();
}
```

Table for User

Field	**Type**	**Information**
userId	UUID-PK	User unique identifier
name	Varchar	User display name
email	Varchar	User unique email
passwordHash	Varchar	User encrypted password
role	Varchar	Viewer, editor, administrator

Table for File

Field	**Type**	**Information**
fileId	UUID-PK	File unique identifier
fileName	Varchar	File name
ownerId	UUID-FK	File owner
uploadDate	Varchar	File upload date
currentVersionId	UUID-FK	File latest version

Table for FileVersion

Field	**Type**	**Information**
versionId	UUID-PK	File version ID
fileId	UUID-FK	File unique identifier
modifiedDate	Timestamp	File version creation date
modifiedBy	UUID-FK	User who created the file version
content	Blob	File binary content

Table for Permission

Field	**Type**	**Information**
permissionId	UUID-PK	User unique identifier
fileId	UUID-FK	User display name
userId	UUID-FK	User unique email
level	Varchar	User encrypted password

Table for EditSession

Field	**Type**	**Information**
sessionId	UUID-PK	Session unique identifier
fileId	UUID-FK	File that session is created on
startedAt	Timestamp	Start time of session
endedAt	Timestamp	End time of session

Table for Notification

Field	**Type**	**Information**
notificationId	UUID-PK	Notification unique identifier
userId	UUID-FK	User who receives the notification
message	Text	Notification message content
createdAt	Timestamp	Notification creation time
isRead	Boolean	Flag that denotes whether the user read the notification

The skeleton classes above (each containing attributes and corresponding to a table) reflect central behaviors of the system. Object relationships are mirrored both in class definitions and schema, with the skeleton classes being created for easy extension whenever needed.

10. Design Patterns

A number of software design patterns and principles were utilized when planning the Cloud-Based File Storage and Collaboration Tool.

GRASP Patterns

- Controller: system-level operations (e.g. file uploading & sharing) are handled by controller components (e.g. FileManager) for more efficient organization of system behavior by assigning responsibilities to lower-level components

SOLID Principles

- Single Responsibility: Classes are focused on specific tasks (e.g. the User class handles authentication and profile management while the File class manages versioning and metadata)

GOF Patterns

- Observer: Edit sessions and notifications are updated in real-time, as the system is enabled to notify users of events without a high degree of coupling

Microservices Patterns

- Service Decomposition: The system is broken down into more loosely coupled services (e.g. for authentication and file management) to allow for easier scaling and independent deployment

Conclusion

The Cloud-Based File Storage and Collaboration project offered valuable insights into object-oriented software system design. The importance of domain modeling, modular architecture, and software design patterns were explored, along with the creation of UML diagrams. A deeper understanding was made in modelling more complex software systems and being able to translate real-world problems into creating design models.

The skills explored in this software design project as a whole can almost certainly be applied in the future, especially in the field of software architecture. Experience was notably gathered in designing scalable, maintainable, and secure software systems from the ground up with the help of diagrams. This experience will most likely prove to be critical when working on larger-scale projects.