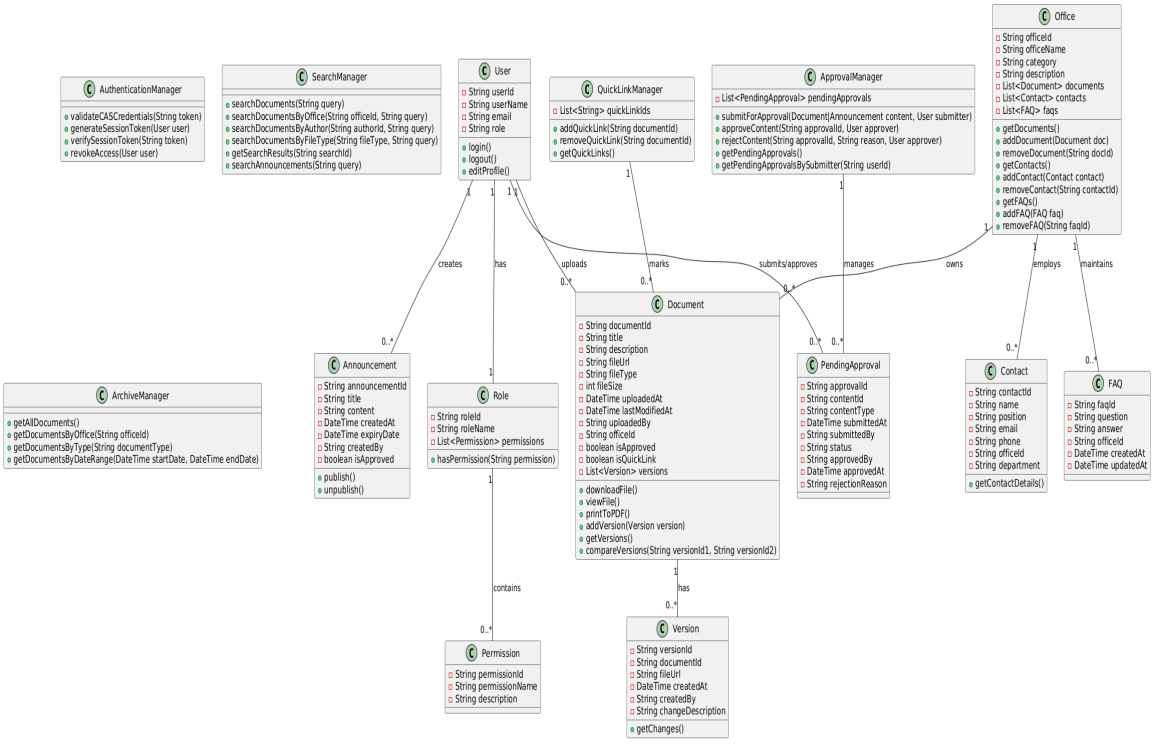
Product Design

Team 25
Redesigning Intranet.iiit.ac.in

Siddharth Mago (Primary Contributor)
Aviral Malhotra
Nilanjana De
Shravani K.
Vanshika Ahlawat

Design Model



Drive link –

https://drive.google.com/file/d/1811Q6gp4I-ZIFP8f2paAcMLAk7wmkwV2/view?usp=sharing

1. User	**Class state** • **userId**: Unique identifier for each user • **userName**: Full name of the user • **email**: Email address of the user • **role**: Role of the user (e.g., Student, IT Office Admin, HR Admin, Super Administrator, Office Admin, Faculty, Office Staff) **Class behavior** • **login()**: Authenticates a user via CAS and provides access to the system • **logout()**: Ends the user's current session • **editProfile()**: Allows users to update their profile information
2. Role	**Class state** • **roleId**: Unique identifier for each role • **roleName**: Name of the role (e.g., Student, Admin, Super Admin) • **permissions**: List of permissions associated with this role **Class behavior** • **hasPermission(String permission)**: Checks if the role has a specific permission
3. Permission	**Class state** • **permissionId**: Unique identifier for each permission • **permissionName**: Name of the permission • **description**: Description of what the permission allows **Class behavior** • No public methods as this is primarily a data container class
4. Authentication Manager	**Class state** • Internal authentication mechanisms and user session information **Class behavior** • **validateCASCredentials(String token)**: Verifies CAS authentication token • **generateSessionToken(User user)**: Creates a session token for a user • **verifySessionToken(String token)**: Checks if a session token is valid • **revokeAccess(User user)**: Terminates a user's access to the system

5. Document	**Class state** • **documentId**: Unique identifier for the document • **title**: Title of the document • **description**: Description of the document contents • **fileUrl**: Path or URL to access the document file • **fileType**: Type/format of the document (e.g., PDF, DOC) • **fileSize**: Size of the document in bytes • **uploadedAt**: Date and time when the document was uploaded • **lastModifiedAt**: Date and time when the document was last modified • **uploadedBy**: User ID of the uploader • **officeId**: Office that the document belongs to • **isApproved**: Flag indicating if the document has been approved • **isQuickLink**: Flag indicating if the document is featured as a quick link • **versions**: List of previous versions of this document **Class behavior** • **downloadFile()**: Allows users to download the document • **viewFile()**: Opens the document for viewing in the browser • **printToPDF()**: Converts and downloads the document as a PDF • **addVersion(Version version)**: Adds a new version of the document • **getVersions()**: Returns all versions of the document • **compareVersions(String versionId1, String versionId2)**: Shows differences between two versions
6. Version	**Class state** • **versionId**: Unique identifier for this version • **documentId**: ID of the parent document • **fileUrl**: Path or URL to access this version of the file • **createdAt**: Date and time when this version was created • **createdBy**: User ID of the creator of this version • **changeDescription**: Description of changes made in this version **Class behavior** • **getChanges()**: Returns the differences between this version and the previous one
7. Announcements	**Class state** • **announcementId**: Unique identifier for the announcement • **title**: Title of the announcement • **content**: Content of the announcement • **createdAt**: Date and time when the announcement was created • **expiryDate**: Date when the announcement expires • **createdBy**: User ID of the creator

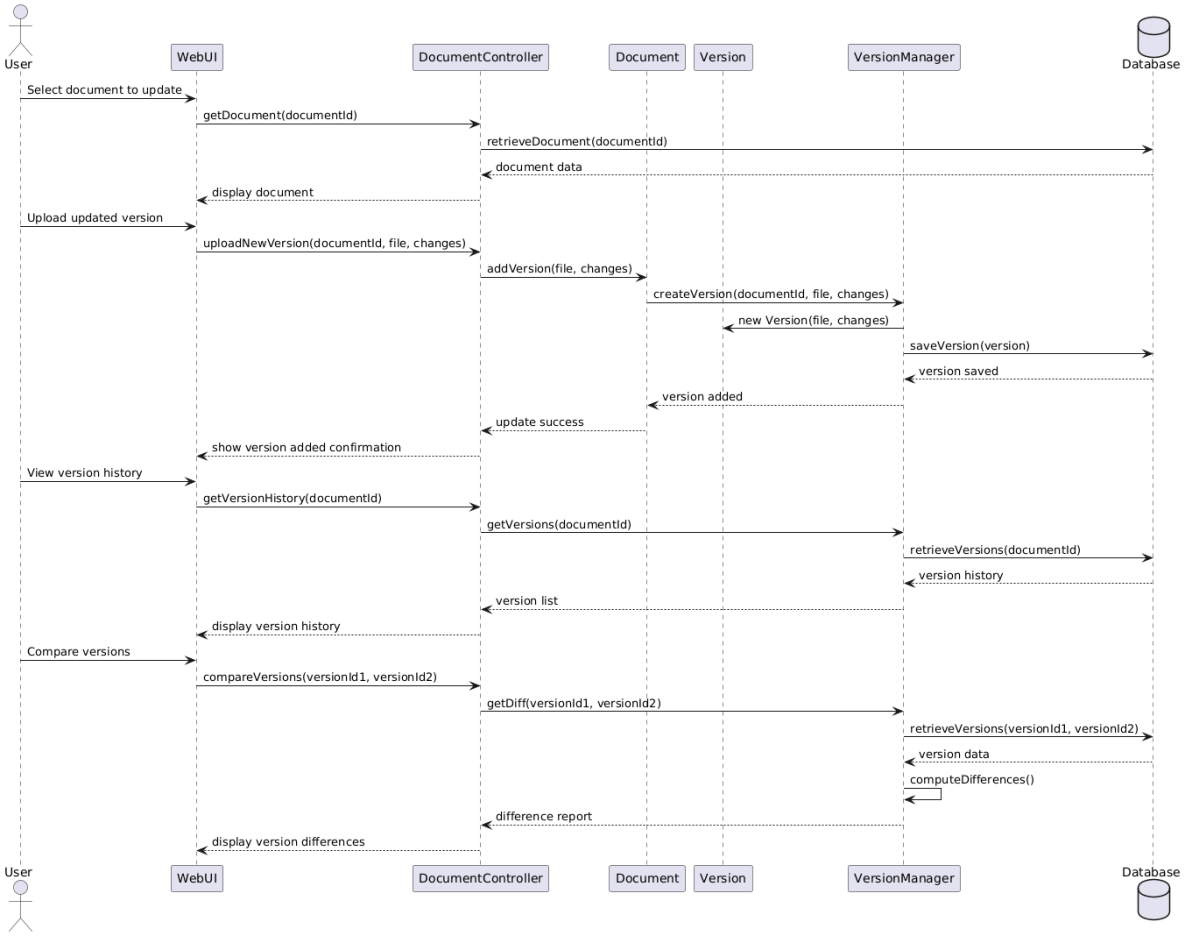
	• **isApproved**: Flag indicating if the announcement has been approved **Class behavior** • **publish()**: Makes the announcement visible to users • **unpublish()**: Removes the announcement from display
8. Offices	**Class state** • **officeId**: Unique identifier for the office • **officeName**: Name of the office (e.g., Academic Office, Admissions Office) • **category**: Category that the office belongs to (e.g., Academics, Students, Research, Administration) • **description**: Brief description of the office's function • **documents**: List of documents associated with this office • **contacts**: List of staff and faculty contacts in this office • **faqs**: List of FAQs associated with this office **Class behavior** • **getDocuments()**: Returns all documents associated with the office • **addDocument(Document doc)**: Adds a new document to the office • **removeDocument(String docId)**: Removes a document from the office • **getContacts()**: Returns all contacts associated with the office • **addContact(Contact contact)**: Adds a new contact to the office • **removeContact(String contactId)**: Removes a contact from the office • **getFAQs()**: Returns all FAQs associated with the office • **addFAQ(FAQ faq)**: Adds a new FAQ to the office • **removeFAQ(String faqId)**: Removes an FAQ from the office
9. Contact	**Class state** • **contactId**: Unique identifier for the contact • **name**: Name of the contact person • **position**: Job title or position • **email**: Email address • **phone**: Phone number • **officeId**: ID of the office the contact is associated with • **department**: Department within the office **Class behavior** • **getContactDetails()**: Returns formatted contact information

10. FAQs	**Class state** • **faqId**: Unique identifier for the FAQ • **question**: The frequently asked question • **answer**: The answer to the question • **officeId**: ID of the office the FAQ is associated with • **createdAt**: Date and time when the FAQ was created • **updatedAt**: Date and time when the FAQ was last updated **Class behavior** • No public methods as this is primarily a data container class.
11. Approval Manager	**Class state** • **pendingApprovals**: List of content items waiting for approval **Class behavior** • **submitForApproval(Document|Announcement content, User submitter)**: Submits content for review and approval • **approveContent(String approvalId, User approver)**: Marks content as approved • **rejectContent(String approvalId, String reason, User approver)**: Rejects content with a reason • **getPendingApprovals()**: Returns all pending approval requests • **getPendingApprovalsBySubmitter(String userId)**: Returns pending approvals submitted by a specific user
12. Pending Approval	**Class state** • **approvalId**: Unique identifier for the approval request • **contentId**: ID of the content that needs approval • **contentType**: Type of content (Document or Announcement) • **submittedAt**: Date and time when the content was submitted for approval • **submittedBy**: User ID of the person who submitted the content • **status**: Current status of the approval (e.g., Pending, Approved, Rejected) • **approvedBy**: User ID of the person who approved/rejected the content • **approvedAt**: Date and time when the content was approved/rejected • **rejectionReason**: Reason for rejection if the content was rejected **Class behavior** • No public methods as this is primarily a data container class.

13. Search Files Manager	**Class state** • Internal search index and results cache **Class behavior** • **searchDocuments(String query)**: Performs a global search across all documents • **searchDocumentsByOffice(String officeId, String query)**: Searches for documents within a specific office • **searchDocumentsByAuthor(String authorId, String query)**: Searches for documents by a specific author • **searchDocumentsByFileType(String fileType, String query)**: Searches for documents of a specific file type • **getSearchResults(String searchId)**: Retrieves search results for a given search ID • **searchAnnouncements(String query)**: Searches for announcements
14. Quick Links Manager	**Class state** • **quickLinkIds**: List of document IDs marked as quick links **Class behavior** • **addQuickLink(String documentId)**: Marks a document as a quick link • **removeQuickLink(String documentId)**: Removes a document from quick links • **getQuickLinks()**: Returns all documents marked as quick links
15. Archive Manager	**Class state** • Internal archive index and organization **Class behavior** • **getAllDocuments()**: Returns all documents in the archive • **getDocumentsByOffice(String officeId)**: Returns documents belonging to a specific office • **getDocumentsByType(String documentType)**: Returns documents of a specific type • **getDocumentsByDateRange(DateTime startDate, DateTime endDate)**: Returns documents created within a date range

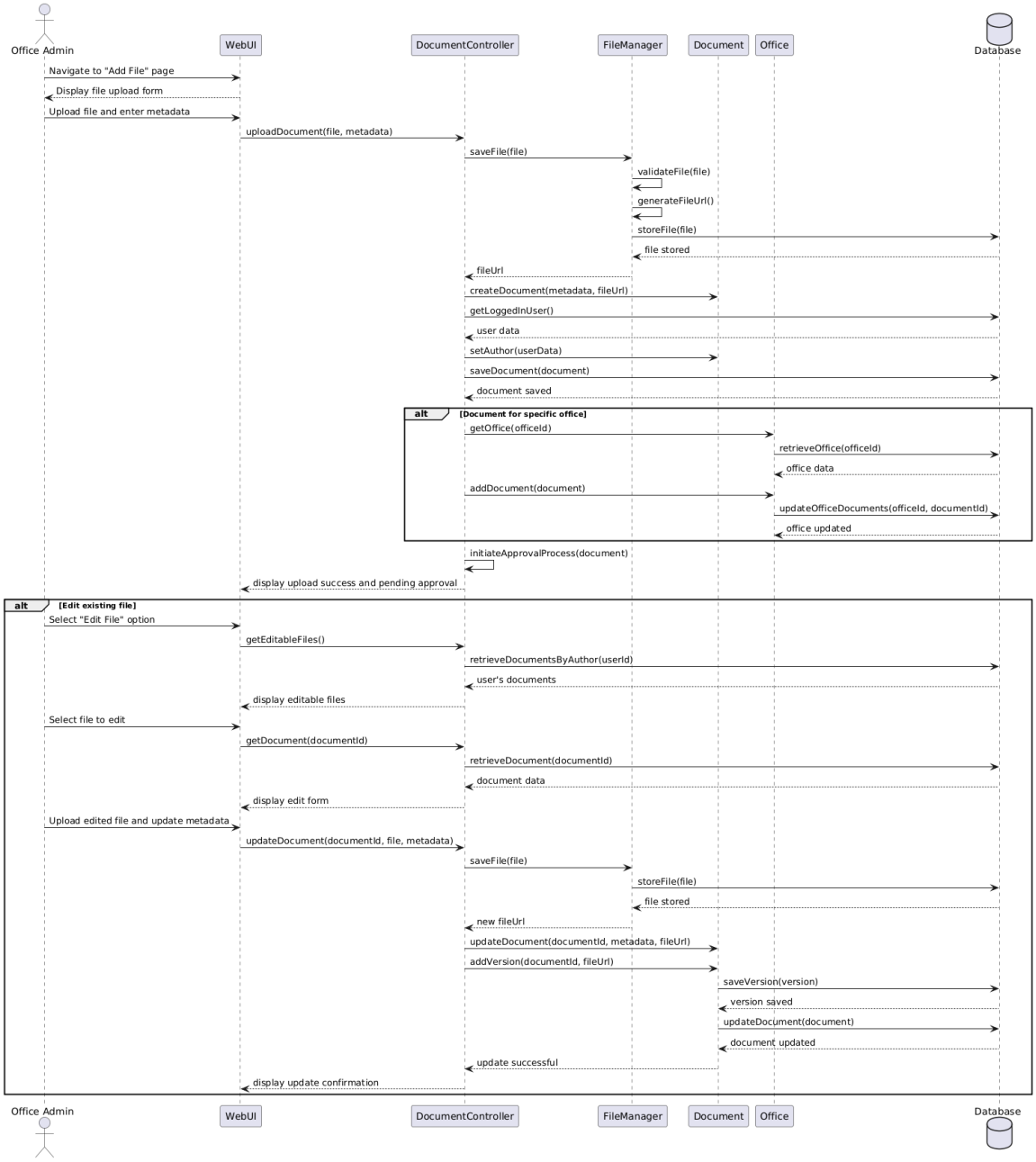
Sequence Diagram(s)

UC-01: Version Control of Documents.



Drive Link -
https://drive.google.com/file/d/1LwbDscu9EwL514LHkFS6pPjxbYNlkoC2/view?usp=sharing

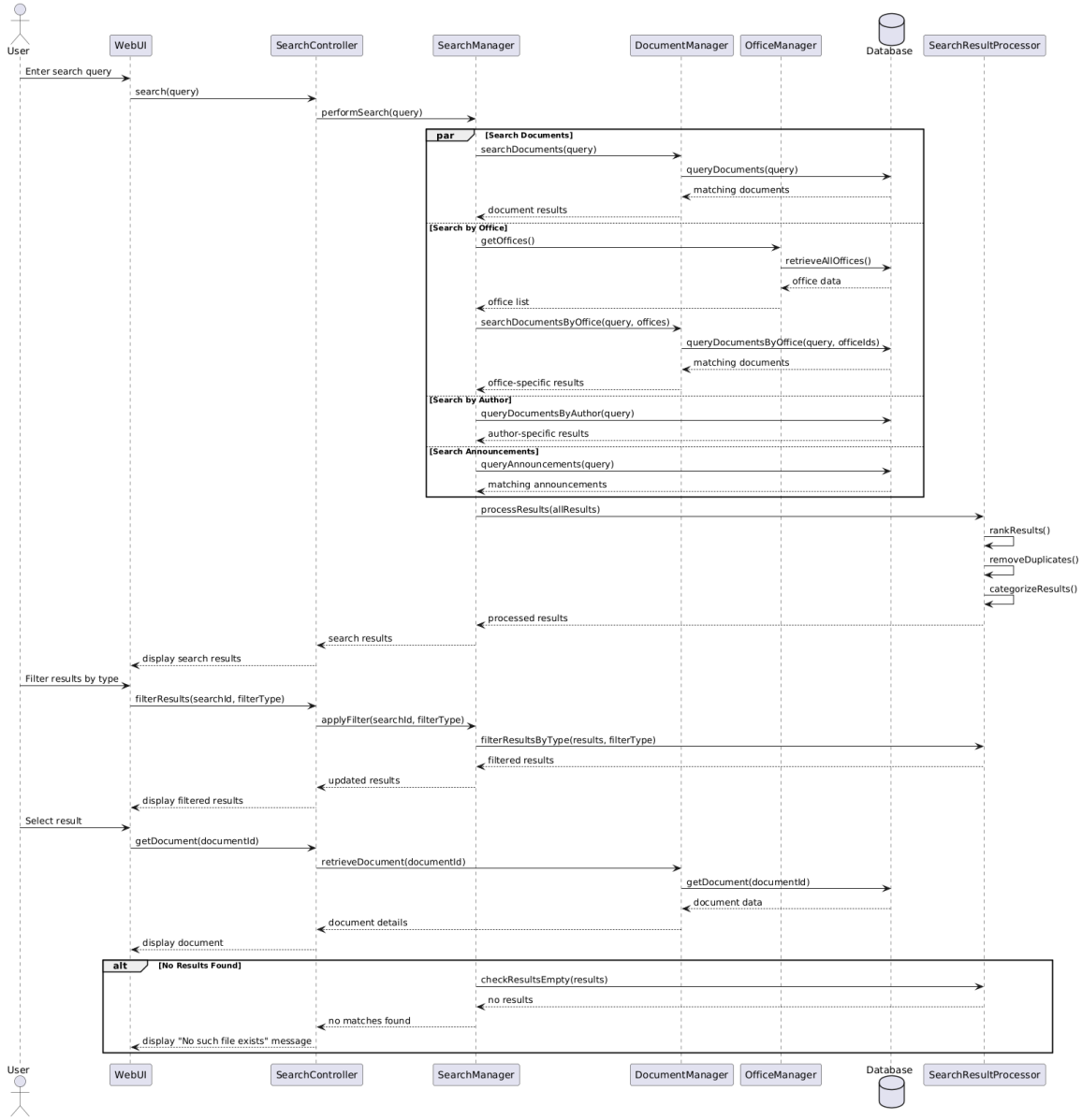
UC-02: Uploading Files and Details



Drive Link -

https://drive.google.com/file/d/19Xo_o2CEVtcbrNIvq7ommtC3qZxxEQnW/view?usp=sharing

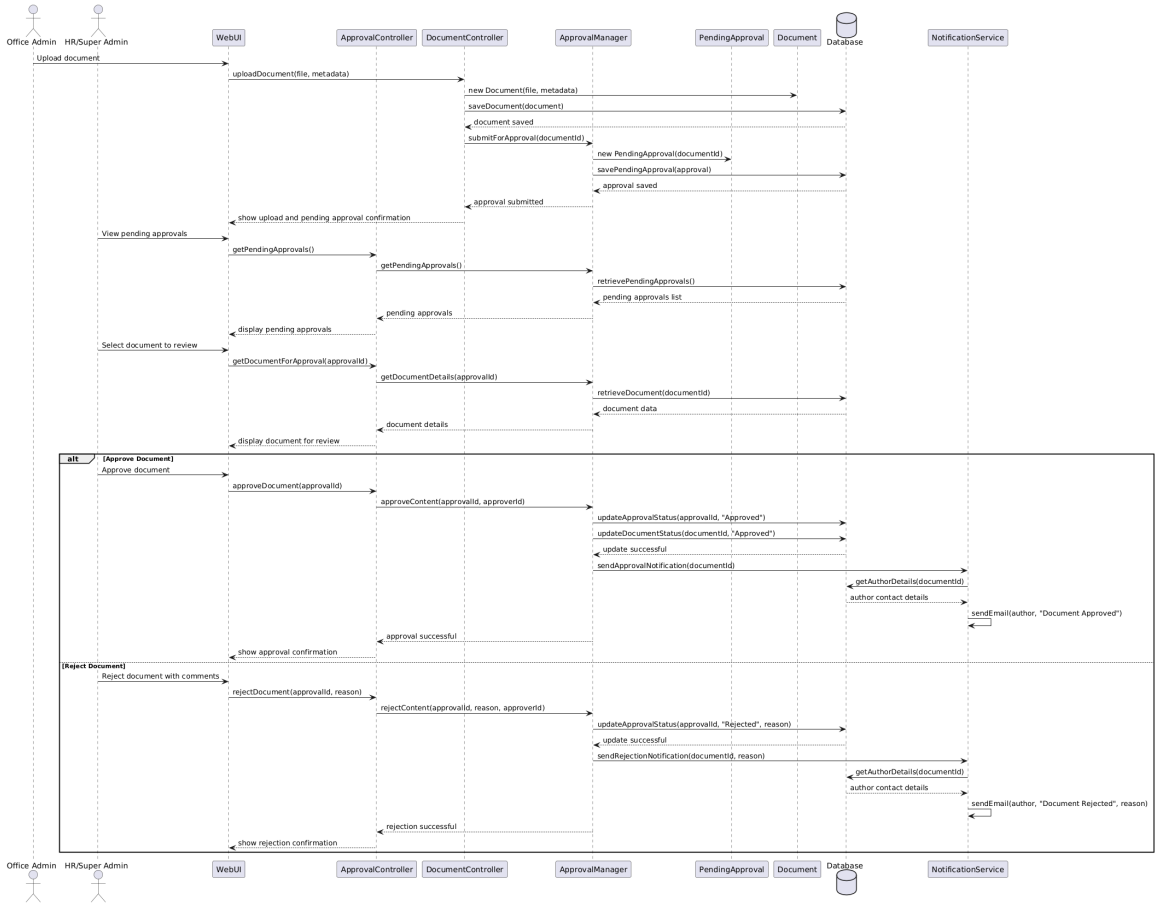
UC-05: Search Files



Drive Link -

https://drive.google.com/file/d/1h3aYs5sBXFQiiVGCNkHoolUnPM7qmqND/view?usp=sharing

UC-11/UC-12: Approve Changes/Pending Changes



Drive Link -

https://drive.google.com/file/d/1oEj6BS7szwiO9_WjWvnx21l2HMcXZwb5/view?usp=sharing

Design Rationale

1. Authentication Approach

Issue: How to handle user authentication securely?

Considered Alternatives:

1. **Custom Authentication System**
    - Pros: Complete control over authentication flow
    - Cons: Security vulnerabilities if not implemented correctly; high development overhead; maintenance burden
2. **CAS (Central Authentication Service) Integration**
    - Pros: Single sign-on capability; already used in university systems; secure
    - Cons: Integration complexity; limited customization options

Decision:

Selected CAS Authentication, as specified in the SRS requirements. This leverages the existing university authentication infrastructure and provides a seamless single sign-on experience for all users. The security is managed by the university's IT department, reducing our development and maintenance burden while ensuring compliance with institutional security policies.

2. Document Management and Version Control

Issue: How to implement efficient document versioning?

Considered Alternatives:

1. **Complete Document Storage for Each Version**
    - Pros: Simple implementation; direct access to full document in any version
    - Cons: Storage inefficiency; duplicate files for minor changes
2. **Delta-based Version Control (Git-like)**
    - Pros: Storage efficiency; sophisticated difference tracking
    - Cons: Complex implementation; potential performance issues for large documents
3. **Hybrid Approach with Metadata + Files**
    - Pros: Reasonable storage efficiency; simpler implementation than delta-based
    - Cons: More complex than full storage; requires additional metadata management

Decision:

Implemented the Hybrid Approach with a Version class that maintains metadata about changes along with file references. This strikes a balance between implementation complexity and storage efficiency. It allows for straightforward version comparison while avoiding duplicate storage of unchanged content. The compareVersions() method on the Document class leverages this structure to show differences between versions.

3. Content Approval Workflow

Issue: How to design the approval workflow for content?

Considered Alternatives:

1. **Single-level Approval**
    - Pros: Simple workflow; faster approval process
    - Cons: Limited validation; potential for inappropriate content
2. **Multi-level Hierarchical Approval**
    - Pros: Robust validation; multiple checkpoints
    - Cons: Process delays; complex workflow management
3. **Role-based Approval with Escalation**
    - Pros: Balanced approach; appropriate approval levels based on content type
    - Cons: Moderate complexity in implementation

Decision:

Selected Role-based Approval with the ApprovalManager and PendingApproval classes. This approach allows different approval paths based on content type and user roles, with Super Administrators having highest authority. This balances the need for content validation with operational efficiency. The solution is flexible enough to handle both simple approvals (e.g., routine document uploads) and more sensitive content that might require additional scrutiny.

4. Search Functionality Implementation

Issue: How to implement efficient search functionality?

Considered Alternatives:

1. **Database Full-Text Search**
    - Pros: Leverages database capabilities; simpler implementation
    - Cons: Limited to database capabilities; potential performance issues with complex queries
2. **Dedicated Search Engine (Elasticsearch)**
    - Pros: Powerful search capabilities; optimized for search operations
    - Cons: Additional infrastructure requirement; integration complexity
3. **Hybrid Approach with Indexed Metadata**
    - Pros: Balance of performance and complexity; targeted search capabilities
    - Cons: Requires careful index management; more complex than simple database search

Decision:

Implemented the Hybrid Approach with SearchManager that maintains indexed metadata for common search patterns. This provides good search performance without the infrastructure complexity of a dedicated search engine. The approach allows searching by multiple parameters (document name, office, author, etc.) while maintaining reasonable response times. The implementation is also extensible if more advanced search features are needed in the future.

5. Content Organization Strategy

Issue: How to organize content for intuitive navigation?

Considered Alternatives:

1. **Flat Structure with Tags**
    - Pros: Flexibility in categorization; powerful for search
    - Cons: Potential for disorganization; relies heavily on search
2. **Hierarchical Structure by Office/Department**
    - Pros: Clear organizational boundaries; intuitive for users familiar with university structure
    - Cons: Content may belong to multiple departments; potential silos
3. **Hybrid Approach with Primary Organization + Cross-referencing**
    - Pros: Clear primary organization with flexibility; supports multiple access patterns
    - Cons: More complex to maintain; requires consistent tagging

Decision:

Selected the Hybrid Approach with primary organization by Office (as shown in the Office class) along with cross-referencing capabilities through the search functionality. This matches users' mental models of the university structure while providing flexibility. The QuickLinks feature provides an additional navigation shortcut for frequently accessed content, addressing the need for easy access to important documents regardless of their organizational location.

6. File Storage Strategy

Issue: How to handle file storage and access control?

Considered Alternatives:

1. **Database BLOB Storage**
    - Pros: Unified storage with metadata; simplifies backup
    - Cons: Database size growth; potential performance issues
2. **File System with References**
    - Pros: Efficient for large files; standard file system operations
    - Cons: Separate backup requirements; potential synchronization issues
3. **Cloud Storage Integration (S3-compatible)**
    - Pros: Scalability; built-in redundancy; potential cost savings
    - Cons: External dependency; potential network latency

Decision:

Implemented File System with References, stored in the Document and Version classes. This approach provides good performance for file operations while keeping the database size manageable. The file paths are stored as references in the database, maintaining the relationship between metadata and the actual files. This approach also allows for future migration to cloud storage if needed, as only the file paths would need to be updated.

7. User Interface Framework

Issue: What UI framework to use for the intranet?

Considered Alternatives:

1. **Custom HTML/CSS/JavaScript**
    - Pros: Complete control; no external dependencies
    - Cons: Development time; maintenance burden
2. **WordPress with Custom Theme**
    - Pros: Built-in CMS capabilities; extensive plugin ecosystem
    - Cons: Potential security issues; overweight for specific requirements
3. **Flask with Modern Frontend Framework**
    - Pros: Lightweight backend; flexible; good integration with Python
    - Cons: Requires more custom development than WordPress

Decision:

Selected Flask with a modern frontend framework, as mentioned in the system requirements of the SRS. This provides a lightweight, flexible foundation that can be customized to the specific needs of the intranet. Flask's integration with Python makes it suitable for implementing the required functionality while allowing for a modern, responsive user interface. The approach also aligns with the technical capabilities specified in the system requirements.

8. Access Control Granularity

Issue: How to implement appropriate access control?

Considered Alternatives:

1. **Simple Role-based Access Control**
    - Pros: Straightforward implementation; easy to understand
    - Cons: Limited flexibility; potential for overly broad permissions
2. **Attribute-based Access Control**
    - Pros: Fine-grained control; context-aware permissions
    - Cons: Complex implementation; potential performance impact
3. **Hybrid Role-based with Permission Refinement**
    - Pros: Balanced approach; reasonable flexibility with understandable structure
    - Cons: More complex than simple RBAC; requires careful permission design

Decision:

Implemented Hybrid Role-based Access Control with the Role and Permission classes. This approach provides a clear structure based on user roles (Student, IT Office, HR, etc.) while allowing for fine-grained permission adjustments when needed. The model supports the different access requirements specified in the SRS, such as HR having more power than IT Office Admins, and Super Administrators having complete control. This balances security needs with system usability and administrative overhead.