Software Design Document

Online Document and Spreadsheet Processing Platform

Authors: Rishabh Raj & Arav Sawant

Table of Contents

1. Introduction
    - 1.1. Purpose
    - 1.2. Scope
    - 1.3. Definitions, Acronyms and Abbreviations
    - 1.4. References
2. System Architecture and Layers
    - 2.1. Architectural Style: Microservices
    - 2.2. Architectural Layers
    - 2.3. Architectural Diagram
    - 2.4. Major Components
3. System Components
    - 3.1. Decomposition Description (Functional Analysis & Refinement)
        - 3.1.1. Functional Decomposition
        - 3.1.2. Architecture
    - 3.2. Dependency Description
    - 3.3. Interface Description
    - 3.4. Adherence to Software Design Principles
4. Detailed Design
    - 4.1. Module Detailed Design
        - 4.1.1. AuthService Module
        - 4.1.2. DocumentService Module
        - 4.1.3. ConcurrencyService (CRDT) Module
        - 4.1.4. CollaborationGateway Module
        - 4.1.5. Other Service Modules
    - 4.2. Data Detailed Design
        - 4.2.1. Databased Schema (Relational)
        - 4.2.2. Content Datastructures (CRDTs)
    - 4.3. Requirements Traceability Matrix (RTM)

1. Introduction

1.1 Purpose

The purpose of this Software Design Document (SDD) is to provide a complete and implementable technical blueprint for the "*Online Document and Spreadsheet Processing Platform*". It translates the functional and non-functional requirements specified in the project's Software Requirements Specification (SRS) into a detailed architecture and component-level design.

1.2 Scope

The system is a web-based collaborative editing tool that allows teams to create, edit, view, and comment on documents and spreadsheets in real time. The scope of this design includes:

- User account management and authentication.
- Document and spreadsheet creation, storage, and permission management.
- A real-time, multi-user collaborative editing engine using **Conflict-free Replicated Data Types (CRDT)**.
- Support for rich text documents and grid-based spreadsheets with formula evaluation.
- Support for offline editing and synchronization.
- File import/export capabilities.

1.3 Definitions, Acronyms and Abbreviations

- **SDD:** Software Design Document
- **SRS:** Software Requirements Specification
- **CRDT:** Conflict-free Replicated Data Type. A data structure that can be replicated across multiple computers in a network, where the replicas can be updated independently and concurrently without coordination, and where it is always mathematically possible to resolve inconsistencies.
- **JWT:** JSON Web Token. A compact, URL-safe means of representing claims to be transferred between two parties.
- **API:** Application Programming Interface
- **REST:** Representational State Transfer. An architectural style for providing standards between computer systems on the web.
- **WebSocket:** A communication protocol providing full-duplex communication channels over a single TCP connection.
- **RTM:** Requirements Traceability Matrix

1.4 References

1. N/A.

2. System Architecture and Layers

This section formally defines the high-level architecture chosen for the system, outlining the guiding principles, logical layers, and the flow of data between major components.

2.1 Architectural Style: Microservices

A **Microservices Architecture** has been selected for the backend. The system is decomposed into a collection of small, independently deployable services, each organized around a specific business capability. This architectural style was chosen over a monolithic approach for several key reasons:

- **Scalability:** Each service (e.g., AuthService, DocumentService) can be scaled independently based on its specific load, optimizing resource utilization. For instance, if real-time collaboration generates heavy traffic, the CollaborationGateway can be scaled up without affecting other services.
- **Maintainability and High Cohesion:** By organizing services around single business responsibilities (e.g., authentication, document management), we achieve high functional cohesion. This makes the system easier to understand, develop, and maintain.
- **Fault Isolation:** An issue or failure in one service (e.g., ImportExportService) is less likely to cascade and bring down the entire application, enhancing overall system resilience.
- **Technology Flexibility:** Each microservice can be built using the most appropriate technology stack for its specific task, allowing for greater flexibility and optimization.

2.2 Architectural Layers

The system is logically divided into four distinct layers, which separate concerns and create a clear, manageable structure.

1. **Client Layer (Presentation Layer):**
    - **Description:** This is the user-facing layer of the application. It consists of a modern single-page application (SPA) that runs in the user's web browser.
    - **Responsibilities:** It is responsible for rendering the user interface, managing local user state, handling user input, and managing the client-side CRDT logic for real-time collaboration and offline capabilities.

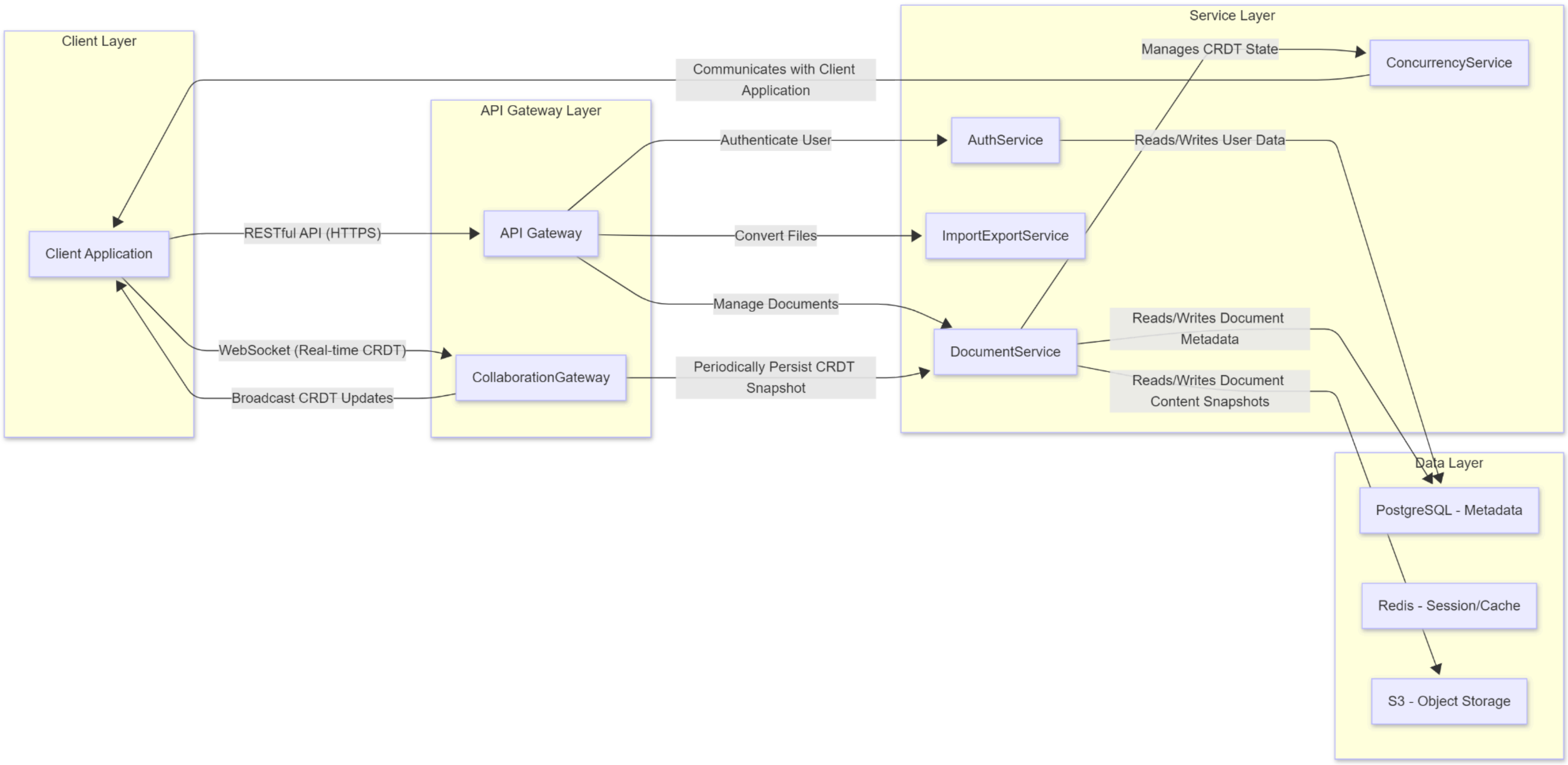
2. **API Gateway Layer (Communication Layer):**
   - **Description:** This layer acts as the single entry point for all client requests to the backend. It abstracts the complexity of the internal microservice landscape from the client.
   - **Responsibilities:** It routes incoming requests to the appropriate backend service. Specifically, it manages two types of communication:
     - **RESTful API:** For stateless HTTP requests (e.g., login, fetching document lists).
     - **WebSocket Gateway (CollaborationGateway):** For stateful, persistent connections required for real-time CRDT updates.
3. **Service Layer (Business Logic Layer):**
   - **Description:** This layer contains the core business logic of the application, implemented as a set of independent microservices. Each service encapsulates a specific domain.
   - **Responsibilities:** Each module (service) in this layer is responsible for a distinct set of functions, such as user authentication, document metadata management, or file conversion. They communicate with each other via internal APIs when necessary.

4. Data Layer (Persistence Layer):

   - **Description:** This layer is responsible for the persistent storage of data. A polyglot persistence approach is used, meaning different types of databases are chosen for different data needs.
   - **Responsibilities:** It includes relational databases for structured metadata (users, permissions), a key-value store for session data, and an object store for large file content snapshots.

2.3 Architectural Diagram

The following diagram provides a visual representation of the microservices architecture, illustrating the flow of communication between the distinct layers and their core components.



2.4 Major Components

The system is decomposed into the following major service components, which reside primarily in the Service and API Gateway layers:

- **Client Application:** The SPA that runs in the user's browser, managing the UI, CRDT state, and offline capabilities.
- **AuthService:** Handles all user identity and authentication tasks.
- **DocumentService:** Manages document and spreadsheet metadata, permissions, and content persistence.
- **ConcurrencyService:** A client-side library and a thin server-side component for managing CRDT data structures and ensuring convergence.
- **CollaborationGateway:** Manages stateful, real-time WebSocket connections and acts as a simple pub/sub broker for CRDT updates.
- **ImportExportService:** Handles file format conversions while importing/exporting docs and/or spreadsheets.

3. System Components

3.1 Decomposition Description (Functional Analysis & Refinement)

This section details the process of decomposing the system requirements into a final, robust modular architecture.

3.1.1 Functional Decomposition

1. **registerUser:** Register a new user account.
2. **authenticateUser:** Authenticate an existing user (Login).
3. **hashAndStorePassword:** Hash and securely store a user's password. [Ref: Mistake #9]
4. **updateUserProfile:** Allow a user to update their profile information.
5. **deleteUserAccount:** Allow a user to delete their own account. [Ref: Mistake #1]
6. **createDocument:** Create a new blank document or spreadsheet.
7. **deleteDocument:** Delete an existing document or spreadsheet.
8. **fetchFileContent:** Read/fetch the content and metadata of a file.
9. **grantUserPermissions:** Grant another user permission to a file.
10. **changeUserPermissions:** Change a user's permission level for a file.
11. **applyFormatting:** Apply formatting (e.g., bold, font size) to text in a document.
12. **InsertImage:** Insert an image into a document.
13. **addComment:** Add a comment to a section of a document or a cell in a spreadsheet.
14. **evaluateFormula:** Evaluate spreadsheet formulas.
15. **establishWebSocketSession:** Establish a persistent WebSocket connection for a user to a document session.
16. **mergeCrdtState:** Merge a local CRDT state change with changes from other replicas. [Ref: Mistake #10]
17. **broadcastCrdtUpdate:** Broadcast a CRDT state update to all users in a session.
18. **displayUserCursors:** Display the cursor positions or active cell selections of other users. [Ref: Mistake #8]
19. **handleChatMessages:** Send and receive real-time chat messages within a file session. [Ref: Mistake #3]
20. **performCollaborativeUndoRedo:** Implement a collaborative undo/redo of an operation based on the CRDT history. [Ref: Mistake #10]
21. **importFromFile:** Parse an imported .docx or .xlsx file into the system's internal CRDT format. [Ref: Mistake #4]
22. **exportToFile:** Serialize the system's internal CRDT format into a .pdf or .docx file for export. [Ref: Mistake #4]
23. **cacheOfflineData:** Cache application assets and file data for offline access. [Ref: Mistake #2]

24. **queueOfflineEdits:** Queue user edits made while offline as local CRDT updates. [Ref:

Mistake #2]

25. **syncOfflineUpdates:** Synchronize queued offline CRDT updates upon reconnection. [Ref: Mistake #6]
26. **logCriticalEvent:** Log a critical security or system event (e.g., failed login, file deletion). [Ref: Mistake #7]
27. **sanitizePII:** Strip Personally Identifiable Information (PII) from text before sending it to a third-party grammar check service.

3.1.2 Architecture

- **AuthService** (Cohesion: 7/7 - Functional)
- **DocumentService** (Cohesion: 7/7 - Functional)
- **ConcurrencyService (CRDT)** (Cohesion: 7/7 - Functional)
- **CollaborationGateway** (Cohesion: 6/7 - Sequential)
- **ImportExportService** (Cohesion: 7/7 - Functional)

3.2 Dependency Description

The dependencies between components are simplified in the CRDT model. The Client Application handles the core merge logic. It communicates with the AuthService and DocumentService via REST. For real-time updates, it sends CRDT state changes over a WebSocket to the CollaborationGateway. The CollaborationGateway's primary role is to broadcast these changes; it **does not** depend on a separate concurrency logic service. It is a simple and efficient message relayer.

3.3 Interface Description

- **Client to Backend (REST):** All standard requests (login, get document list, etc.) use a RESTful API over HTTPS. An Authorization header with a JWT is required for protected endpoints.
- **Client to Backend (WebSocket):** Used to send and receive CRDT state updates or deltas. The payload is the CRDT data itself, not an operation to be transformed.
- **Inter-Service Communication:** Backend services communicate with each other via internal, private RESTful APIs.

3.4 Adherence to Software Design Principles

DP 1: Single Responsibility Principle (SRP)

- **Principle:** A module or class should have one, and only one, reason to change.
- **Application:** The initial `UtilityModule` (Cohesion Score: 1/7) was a clear violation. It was decomposed into `AuditingService`, `ImportExportService`, and

`ThirdPartyProxyService`. Now, a change to the logging format only affects the `AuditingService`, adhering to SRP.

DP 2: Separation of Concerns (SoC)

- **Principle:** A system should be divided into distinct parts with minimal overlap in functionality.
- **Application:** The architecture separates the concern of real-time communication (`CollaborationGateway`) from the concern of data consistency (`ConcurrencyService`). The gateway doesn't need to know *how* CRDTs merge, only that it needs to transmit them. This separation is fundamental to the design.

DP 3: Dependency Inversion Principle (DIP)

- **Principle:** High-level modules should not depend on low-level modules. Both should depend on abstractions.
- **Application:** The `Client Application` (high-level) depends on an abstract `ConcurrencyService` interface. The specific CRDT implementation (e.g., Y.js, Automerge) is a low-level detail that can be changed without altering the application's core workflow.

DP 4: Keep It Simple, Stupid (KISS)

- **Principle:** Systems work best if they are kept simple rather than made complicated.
- **Application:** Adopting the CRDT model is a direct application of this principle to the server architecture. The `CollaborationGateway` becomes a simple, stupid message broker, which is far less complex and easier to scale than a stateful OT-based transformation server.

DP 5: Design for Testability

- **Principle:** A system should be designed to make testing easier.
- **Application:** Each service can be tested independently. The `AuthService` can be tested with mock database calls. The `ConcurrencyService` (CRDT logic) is a pure, client-side library that can be unit-tested extensively without needing any network or backend infrastructure.

DP 6: Design Defensively

- **Principle:** Handle all inappropriate usage cases and assume inputs are hostile.
- **Application:** All API endpoints in the `AuthService` and `DocumentService` will perform rigorous input validation (e.g., checking for valid email formats, non-empty titles). Permission checks are performed on every single document access request, even for internal service calls.

4. Detailed Design

4.1 Module Detailed Design

4.1.1 AuthService Module

- **Purpose:** Handles all user identity, registration, login, and profile management.
- **Pseudocode (User Registration):**

```
FUNCTION registerUser(email, password):
  IF email is not valid OR password is too weak THEN
    RETURN Error("Invalid input")
  ENDIF

  existingUser = Database.findUserByEmail(email)
  IF existingUser IS NOT NULL THEN
    RETURN Error("User already exists")
  ENDIF

  // Use a strong, one-way hashing algorithm as per Mistake #9
  passwordHash = BCrypt.hash(password)

  newUser = new User(email, passwordHash)
  Database.save(newUser)

  RETURN Success(newUser.id)
END FUNCTION
```

4.1.2 DocumentService Module

- **Purpose:** Manages document and spreadsheet metadata, permissions, and persistence of content snapshots.
- **Pseudocode (Create Document/Spreadsheet):**

```
FUNCTION createFile(userId, title, fileType): // fileType is 'document' or 'spreadsheet'
  // Create initial empty state for the given file type
  IF fileType IS 'spreadsheet' THEN
    // A grid-based CRDT structure
    initialContent = CRDT.Grid.create()
  ELSE
    // A sequence-based CRDT for text
    initialContent = CRDT.Sequence.create()
  ENDIF

  contentKey = ObjectStore.save(initialContent)
  newFile = new Document(title, userId, contentKey)
  Database.save(newFile)

  RETURN Success(newFile.id)
END FUNCTION
```

4.1.3 ConcurrencyService (CRDT) Module

- **Purpose:** A client-side library that provides CRDT data structures and the logic to merge them. The server-side component is minimal, mainly for persistence.
- **Pseudocode (Client-side Merge):**

```
CLASS ClientConcurrencyService:
  localCRDT = null

  // Called when an update is received from the Collaboration Gateway
  FUNCTION onStateReceived(remoteCRDT):
    // The core of CRDT: the merge function is associative, commutative, and idempotent.
    // The order of merging does not matter, and conflicts are resolved automatically.
    self.localCRDT.merge(remoteCRDT)

    // The UI is subscribed to changes in localCRDT and will update automatically
    UI.render(self.localCRDT.getValue())
  END FUNCTION

  // Called when the local user makes an edit
  FUNCTION onUserEdit(editAction):
     // Apply the edit to the local CRDT replica
     self.localCRDT.applyEdit(editAction)

     // Send the updated local state to the server for broadcast
     CollaborationGateway.sendUpdate(self.localCRDT)
  END FUNCTION
END CLASS
```

4.1.4 CollaborationGateway Module

- **Purpose:** Manages real-time WebSocket connections and acts as a simple, efficient pub/sub broker for CRDT updates.
- **Pseudocode (Handle Incoming CRDT Update):**

```
// This function is executed when a CRDT update arrives from a client
FUNCTION handleCRDTUpdate(docId, crdtUpdate, sendingClient):
  session = SessionManager.getSession(docId)
  IF session IS NULL THEN
    RETURN Error("No active session for this document")
  ENDIF

  // CRDT Model: The server does NOT need to understand or transform the data.
  // Its only job is to relay the update to other clients in the session.
  // This is significantly simpler and more scalable than an OT-based approach.
  FOR each client IN session.clients:
    IF client IS NOT sendingClient THEN
      client.send(crdtUpdate)
    ENDIF
  ENDFOR

  // Periodically, the latest merged state can be saved to the DocumentService as a snapshot
  // This is for performance, so new users don't have to replay the entire history.
  scheduleSnapshot(docId, crdtUpdate)
END FUNCTION
```

4.1.5 Other Service Modules

- **FormulaEvaluationService (Client-side):** A new sub-module within the Client Application responsible for parsing and computing spreadsheet formulas. It operates on the local CRDT grid state. When a cell's value changes, it triggers a re-evaluation of all dependent cells.

4.2 Data Detailed Design

This section details the structure of the data used throughout the system, both for persistence (backend) and in-memory representation (client-side).

4.2.1 Database Schema (Relational)

The relational database stores metadata and relationships, not the core file content. A PostgreSQL database is chosen for its robustness and support for structured data.

- **`USERS` Table**
  - `user_id` (UUID, Primary Key): Unique identifier for each user.
  - `email` (VARCHAR(255), Unique): User's email address, used for login.
  - `password_hash` (VARCHAR(255)): The *bcrypt* hash of the user's password.
  - `display_name` (VARCHAR(100)): The user's public name.
  - `created_at` (TIMESTAMP): Timestamp of account creation.
- **`FILES` Table**
  - `file_id` (UUID, Primary Key): Unique identifier for each document or spreadsheet.
  - `title` (VARCHAR(255)): The user-defined title of the file.
  - `owner_id` (UUID, Foreign Key to USERS): The user who owns the file.
  - `file_type` (ENUM('document', 'spreadsheet')): Specifies the type of the file.
  - `content_snapshot_key` (VARCHAR(1024)): A key or URL pointing to the latest snapshot of the file's content in the Object Store.
  - `created_at` (TIMESTAMP): Timestamp of file creation.
  - `last_modified` (TIMESTAMP): Timestamp of the last saved change.
- **`PERMISSIONS` Table**
  - `user_id` (UUID, Foreign Key to USERS, Part of Composite PK): The user who has permission.
  - `file_id` (UUID, Foreign Key to FILES, Part of Composite PK): The file the permission applies to.
  - `role` (ENUM('OWNER', 'EDITOR', 'COMMENTER', 'VIEWER')): The level of access granted.

4.2.2 Content Data Structures (CRDTs)

The core file content is managed on the client-side using CRDTs. These are complex data structures serialized as JSON or binary for storage and transmission.

- **For Documents (Sequence CRDT):**

    - A sequence CRDT (like *Y.js*'s Y.Text or an LSEQ implementation) will be used. This structure represents the document as a list of characters.
    - Each character has a unique, fractional position ID and attributes (e.g., `{bold: true}`).
    - **Insertions:** A new character is inserted between two existing character IDs by generating a new fractional ID.
    - **Deletions:** Characters are marked with a "tombstone" flag rather than being physically removed, which is crucial for the merge algorithm to work correctly.
- **For Spreadsheets (Map CRDT):**
    - A Map CRDT (like *Y.js*'s Y.Map) is used to represent the grid.
    - The top-level Map contains keys for sheet names (e.g., "Sheet1").
    - Each sheet is another Map where keys are cell coordinates (e.g., "A1", "C5") and the values are the cell's content.
    - **Cell Content:** The value for each cell is a small object that can contain its raw value (string or number), a formula string, and formatting attributes.
    - **Updates:** Changing a cell's value is an update operation on the map for that specific key. This is highly efficient as only the changed cell data needs to be broadcast.

4.3 Requirements Traceability Matrix (RTM)

This table maps each decomposed functional requirement (F#) to the specific design module(s) responsible for its implementation, ensuring complete coverage of the SRS.

Req. ID	Description	Design Module(s)
F1-F5	User account lifecycle (register, login, update, delete)	AuthService
F6-F8	File lifecycle (create, delete, read)	DocumentService
F9, F10	Permission management (grant, change role)	DocumentService
F11, F12	Rich text document editing	Client Application, ConcurrencyService (CRDT)

F13	Commenting on files	Client Application, DocumentService
F14	Spreadsheet formula evaluation	Client Application (FormulaEvaluationService)
F15	WebSocket connection management	CollaborationGateway
F16	Concurrency control (CRDT Merge)	Client Application (ConcurrencyService)
F17	Real-time broadcasting of CRDT states	CollaborationGateway
F18	Cursor/selection position display	Client Application, CollaborationGateway
F19	Real-time chat	Client Application, CollaborationGateway
F20	Collaborative undo/redo	Client Application (ConcurrencyService)
F21, F22	File import and export	ImportExportService, Client Application
F23-F25	Offline mode and synchronization	Client Application (OfflineService, ConcurrencyService)
F26	System and security logging	AuditingService, All Services
F27	Data sanitization for 3rd parties	ThirdPartyProxyService