CollaborationService

September 4, 2026

CollaborationService

Kind: Service

Source: `atloria-monorepo/apps/api/src/collaboration/collaboration.service.ts`

Service for managing collaboration sessions and document persistence

`CollaborationService` manages real-time collaboration sessions and persists shared document state for the API. It coordinates session lifecycle operations, document updates, and storage integration so connected clients can collaborate on a consistent document version.

Methods

Method	Signature	Returns	Description
`verifyDocumentAccess`	`verifyDocumentAccess(documentId: string, userId: string)`	`Promise<boolean>`	Verify if a user has access to a document
`getDocumentContent`	`getDocumentContent(documentId: string)`	`Promise<string	null>`	
`getCurrentVersion`	`getCurrentVersion(documentId: string)`	`Promise<number>`	Get current version number for a document
`getNextVersion`	`getNextVersion(documentId: string)`	`Promise<number>`	Get the next version number for a document
`saveDocumentState`	`saveDocumentState(documentId: string, content: string, userId: string)`	`Promise<SaveDocumentResult>`	Save document state to the database `userId` (optional, trailing — call sites stay source-compatible)

Method	Signature	Returns	Description
			is the human whose editing session triggered this save;…
`appendPendingUpdate`	`appendPendingUpdate(documentId: string, update: Uint8Array)`	`Promise<void>`	Append an encoded Yjs update to the per-document crash-recovery log.
`loadRecoveryState`	`loadRecoveryState(documentId: string)`	`Promise<{ snapshot: Uint8Array | null; updates: Uint8Array[] }>`	
`persistSnapshot`	`persistSnapshot(documentId: string, state: Uint8Array, coveredUpdateCount: number)`	`Promise<void>`	Persist the encoded Y.Doc state as the recovery snapshot and trim the update-log entries it covers.
`getPendingUpdateCount`	`getPendingUpdateCount(documentId: string)`	`Promise<number>`	Get the number of pending updates in the crash-recovery log
`clearRecoveryState`	`clearRecoveryState(documentId: string)`	`Promise<void>`	Drop the crash-recovery state for a document.
`createSession`	`createSession(documentId: string, userId: string)`	`Promise<void>`	Create a collaboration session record
`endSession`	`endSession(documentId: string, userId: string)`	`Promise<void>`	End a collaboration

Method	Signature	Returns	Description
			session
`updateSessionActivity`	`updateSessionActivity(documentId: string, userId: string)`	`Promise<void>`	Update session activity timestamp
`getActiveUsers`	`getActiveUsers(documentId: string)`	`Promise<CollaborationUser[]>`	Get active users for a document
`getSessionCount`	`getSessionCount(documentId: string)`	`Promise<number>`	Get session count for a document
`updateCursorPosition`	`updateCursorPosition(documentId: string, userId: string, cursorPosition: number, selectionStart: number, selectionEnd: number)`	`Promise<void>`	Update cursor position in session

Dependencies

- `PrismaService`
- `RedisService`
- `OutboxService` *(optional)*

Where it refuses work

- `CollaborationService` stops the work with an early return when `!client`, in 4 places.
- `CollaborationService` stops the work with an early return when `!document`.
- `CollaborationService` stops the work with an early return when `userIds.length === 0`.

When something fails

- `CollaborationService` handles failure in 22 places: it logs it and continues in 11, turns it into a return value in 9, lets it reach the caller in 1, and discards it silently in 1. A failure discarded silently leaves no trace for whoever debugs this later.

Diagram

```
sequenceDiagram
  participant Client
  participant Controller
  participant Service as CollaborationService
  participant Session as Collaboration Session
  participant Storage as Document Persistence
```

```
Client->>Controller: Join or update collaboration document
Controller->>Service: create/join session or apply update
Service->>Session: Resolve active session state
Session-->>Service: Current document state
Service->>Storage: Persist document changes
Storage-->>Service: Saved document/version
Service-->>Controller: Updated collaboration state
Controller-->>Client: Return state or broadcast update
```

Usage

```
import { Injectable } from '@nestjs/common';
import { CollaborationService } from './collaboration.service';

@Injectable()
export class DocumentWorkflowService {
  constructor(
    private readonly collaborationService: CollaborationService,
  ) {}

  async updateDocument(
    documentId: string,
    userId: string,
    update: Uint8Array,
  ) {
    // Use the collaboration service as the single entry point for
    // session-aware document updates and persistence.
    return this.collaborationService.applyUpdate(
      documentId,
      userId,
      update,
    );
  }
}
```

AI Coding Instructions

- Keep collaboration state changes inside `CollaborationService`; controllers and gateways should delegate session and persistence logic to this service.
- Persist document updates through the established storage integration rather than mutating in-memory session state only.
- Treat updates as concurrent: preserve the service's synchronization/versioning patterns when adding new write operations.
- Ensure session cleanup occurs when clients disconnect or sessions become inactive to prevent stale in-memory state.
- Validate document and user access before joining sessions or applying updates, using the existing authentication and authorization integration points.

Relationships

- DEPENDS_ON → `PrismaService`
- DEPENDS_ON → `RedisService`
- DEPENDS_ON → `OutboxService`

Referenced By

- `CollaborationModule` (MODULE_PROVIDES)
- `CollaborationModule` (MODULE_EXPORTS)
- `DocumentPersistenceService` (DEPENDS_ON)