# Product Design: Polish

## UNT's Best

Arnav Verma
Walid Esmael
Matthew Norman
Mohamed Babiker

**Sprint 3**
12/04/2025

| Revision Number | Revision Date | Summary of Changes | Author(s) |
|---|---|---|---|
| 1 | 10/05/2025 | Added System & Architecture Design, Created overall architecture overview, UML diagram, and design summary showing how frontend, backend, and Azure services interact. | Walid Esmael |
| 2 | 10/06/2025 | Designed ER diagram, explained data flow, and ensured database structure consistency. | Mohamed Babiker |
| 3 | 10/07/2025 | Designed wireframes for login, dashboard, editor, and template library with short captions and explanations. | Mathew Norman,<br>Walid Esmael |
| 4 | 10/07/2025 | Final Document & Formatting, Compiled and formatted the final design document, merged all diagrams, and wrote rationale and explanations for design choices. | Arnav Verma |
| 6 | 10/16/2025 | Updated the Front End Class Diagram | Mohamed Babiker |
| 7 | 10/17/2025 | Updated the Class Diagrams<br>Updated ER Diagrams<br>Updated Information Architecture Diagram | Arnav Verma |
| 8 | 10/31/2025 | • Added Sections 6–10<br>• Full OpenAPI 3.1.0 integration<br>• Cosmos DB schema and change feed<br>• Auth0 replacement<br>• SignalR and SSE streaming<br>• Sequence diagrams<br>• Deployment topology<br>• API versioning strategy | Arnav Verma |
| 9. | 11/21/2025 | Added Sequence Diagrams for:<br>Authentication,<br>Document Services, | Arnav Verma |
| 10. | 11/27/2025 | LLM Service Sequence Diagrams | Mathew Norman |
| 11. | 12/04/2025 | Updated Information Architecture Diagram | Arnav Verma |

| | | | |
|---|---|---|---|
| **12.** | **12/04/2025** | **Updated CI/CD, Deployment Architecture** | **Arnav Verma** |

## Architecture:

Polish is an SPA (React and TypeScript) that communicates with an Azure App Service hosted Node.js and Express API. While the API coordinates AI edits, conversion, versioning, storage, auth, and export, the SPA manages uploads, prompting, and real time editing.

Core Azure integrations:
• Azure OpenAI – generate/modify text and structures in editor.
• Azure Blob Storage – originals, previews, exports.
• Azure Cosmos DB / Azure SQL – users, docs, versions, change logs.
• Azure SignalR – live AI token streaming and real time edits.
• Azure AD B2C – authentication/identity.
• Azure Key Vault – secrets and keys.
• Azure Application Insights – telemetry and monitoring.

Summary of data and control flow: Browser → SPA (HTTPS/WebSocket) → API → (OpenAI, Blob, DB, SignalR, Key Vault, App Insights). Upload, preview, prompt, stream diffs, accept/reject, autosave, version, and export (DOCX, PDF, or LaTeX) are the steps involved.
Why this is effective: Meets nonfunctional requirements (≤3s replies, 99.9% uptime, security/encryption), cross platform access, instant export, live visual feedback, in editor AI, and universal file support.

### Interfaces & Component Responsibilities

**Frontend (TypeScript & React)**

- FileUpload uses POST to upload DOCX, PDF, or LaTeX files. provides a preview, upload progress, and file type and size validation.
- The document and AI recommended edits are rendered by EditorCanvas. It enables users to instantly accept or reject edits over a SignalR channel.
- Prompts and a chosen text scope are sent to apply by AIPromptPanel & shows the state of AI processing and the outcomes that are streamed by SignalR.
- VersionSidebar supports restore actions and lists document versions.
- Every 30 seconds, accepted diffs are automatically committed.
- ExportModal enables export to LaTeX, PDF, or DOCX, manages downloads and shows progress.

**Backend (Express Services and Node.js)**

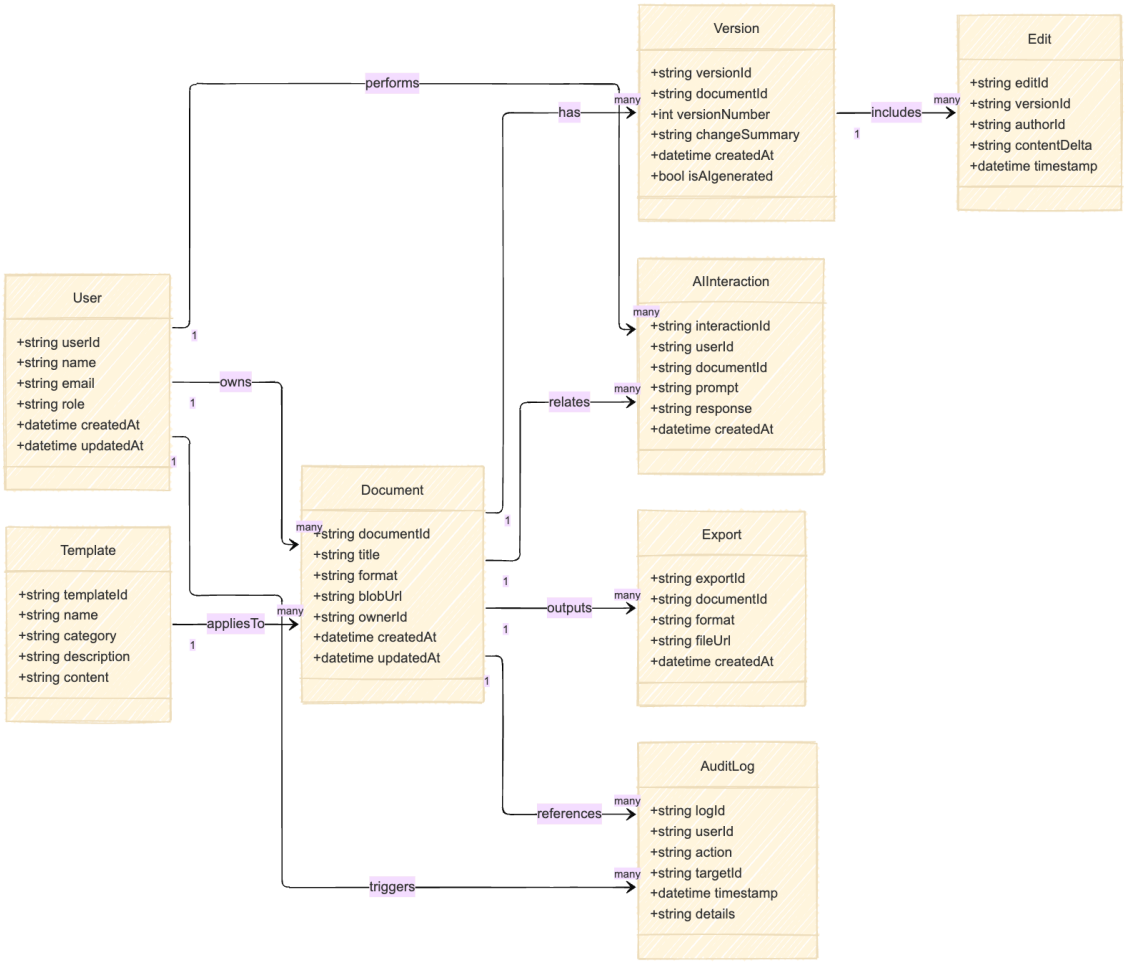
- FileService manages uploads, saves files in Azure Blob Storage, and transforms documents.
- produces front end preview materials.

- AIService Notifies Azure OpenAI of user prompts for document editing and text production.
- uses Azure SignalR to stream AI replies to the frontend.
- Document versions are tracked and committed by VersionService.
- allows autosave and restore and stores both manual and AI modifications.
- ExportService preserves the original formatting while converting documents into the desired formats.
- TemplateService applies AI generated templates to documents and provides a list of available templates.
- AuthService uses Azure AD B2C to manage user login and authorization.

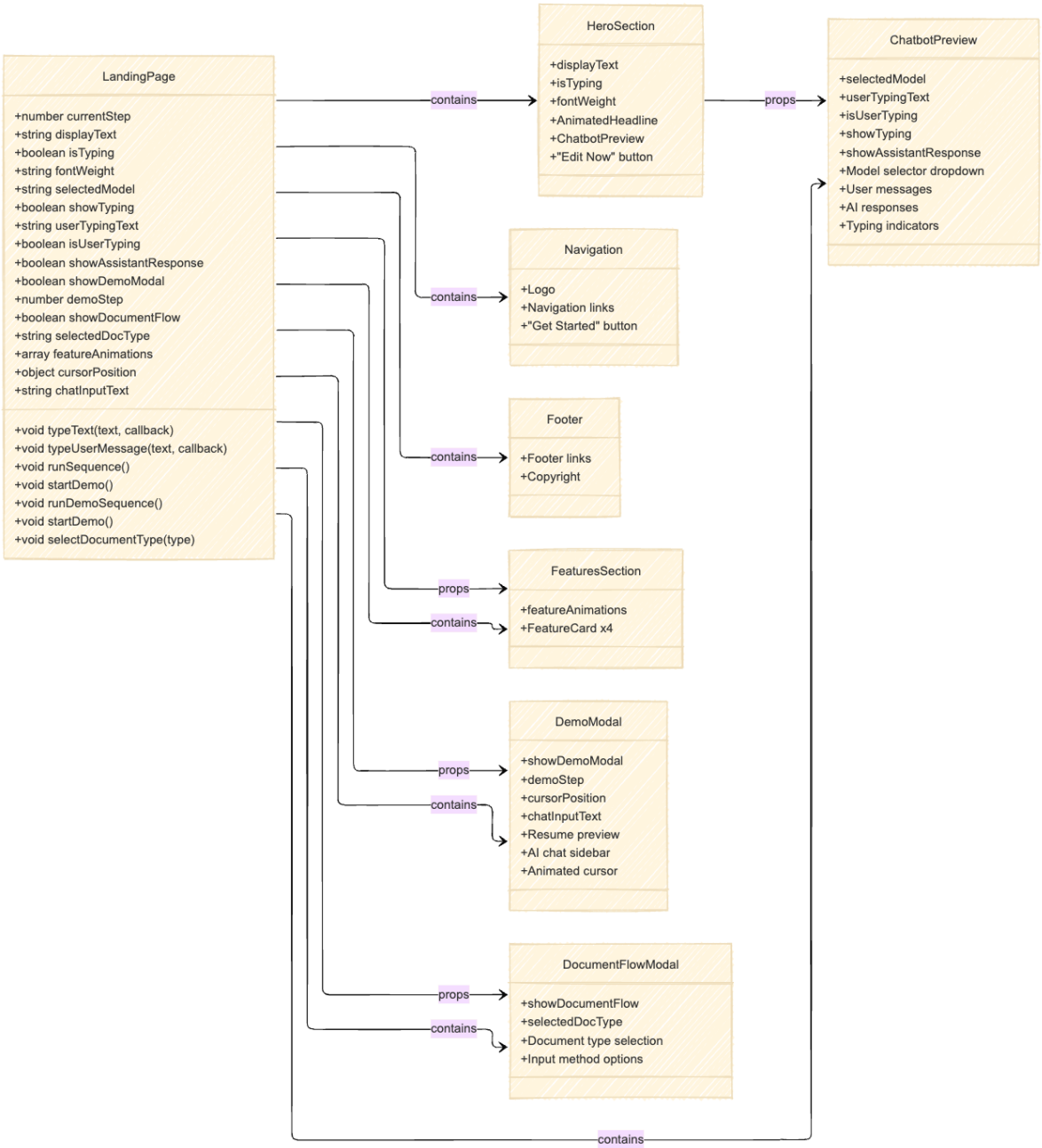
**Storage & Observability**

- Azure Storage Accounts – Stores originals, converted files, previews, and exports.
- Azure Cosmos DB / Azure SQL – Manages users, documents, versions, and changes.
- Azure Key Vault – Protects API keys and secrets.
- Azure Application Insights – Logs performance, errors, and latency metrics.

# Class Diagrams




Core Backend Services Class Diagram




Front end class diagram

# ER Diagram(s)

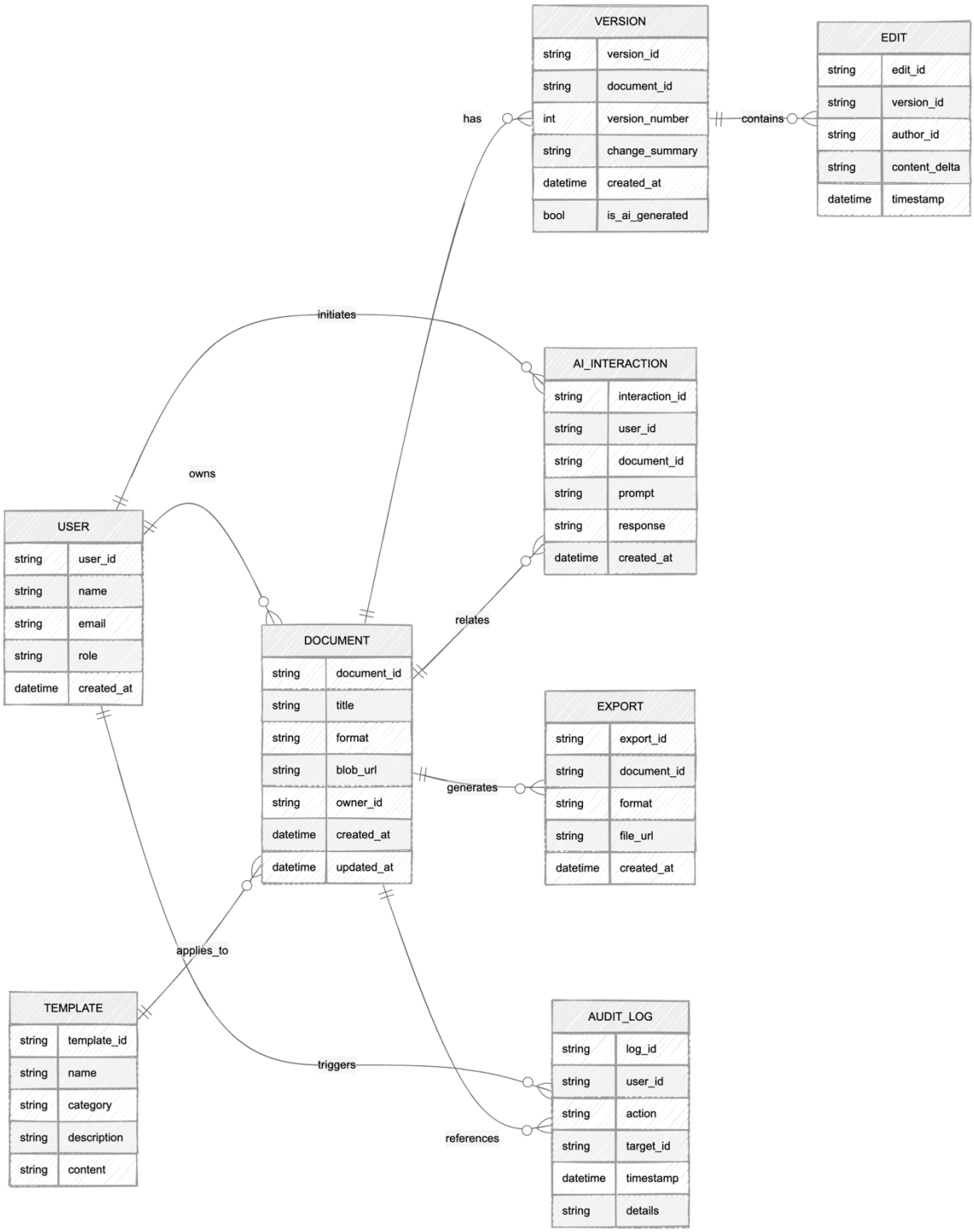



Fig 1.3 ER Diagram

**Data Flow Explanation:** In the case of a user signing in with Azure AD B2C, the profile is determined in the users table Documents allow the user to create or upload various Documents, which contain metadata, like title, format, and a bloburl reference to the file in Azure Blob Storage. A document may have numerous Versions, which are defined as the saved versions of the document made either by human hand or by AI revisions. All the changes are recorded in the Edits table and every AI prompt and response are traced in AIInteractions.The completed documents are exported to such formats as DOCX, PDF, or LaTeX and stored in the Exports table. Templates are used to store reusable resume and cover letter templates and Audit_Logs keep track of the important actions taken, including uploads, AI prompts, and exports, in order to ensure transparency.This relational model is reliable with the help of Azure SQL Database, can be used in real time collaboration with the help of SignalR and is secure since only the metadata was stored in SQL while large files are managed in Blob Storage.

**Consistency Notes:**
Normalization: Schema is a follow up of 3NF to remove redundancy.
Foreign Keys: Impose inter table referential integrity.
Timestamps: Timestamps are included in all tables (creation and update).
Cloud Integration Files stored in Azure Blob storage.
Scalability: Scalability, real time editing, AI assistance, and version control.

# Information Architecture

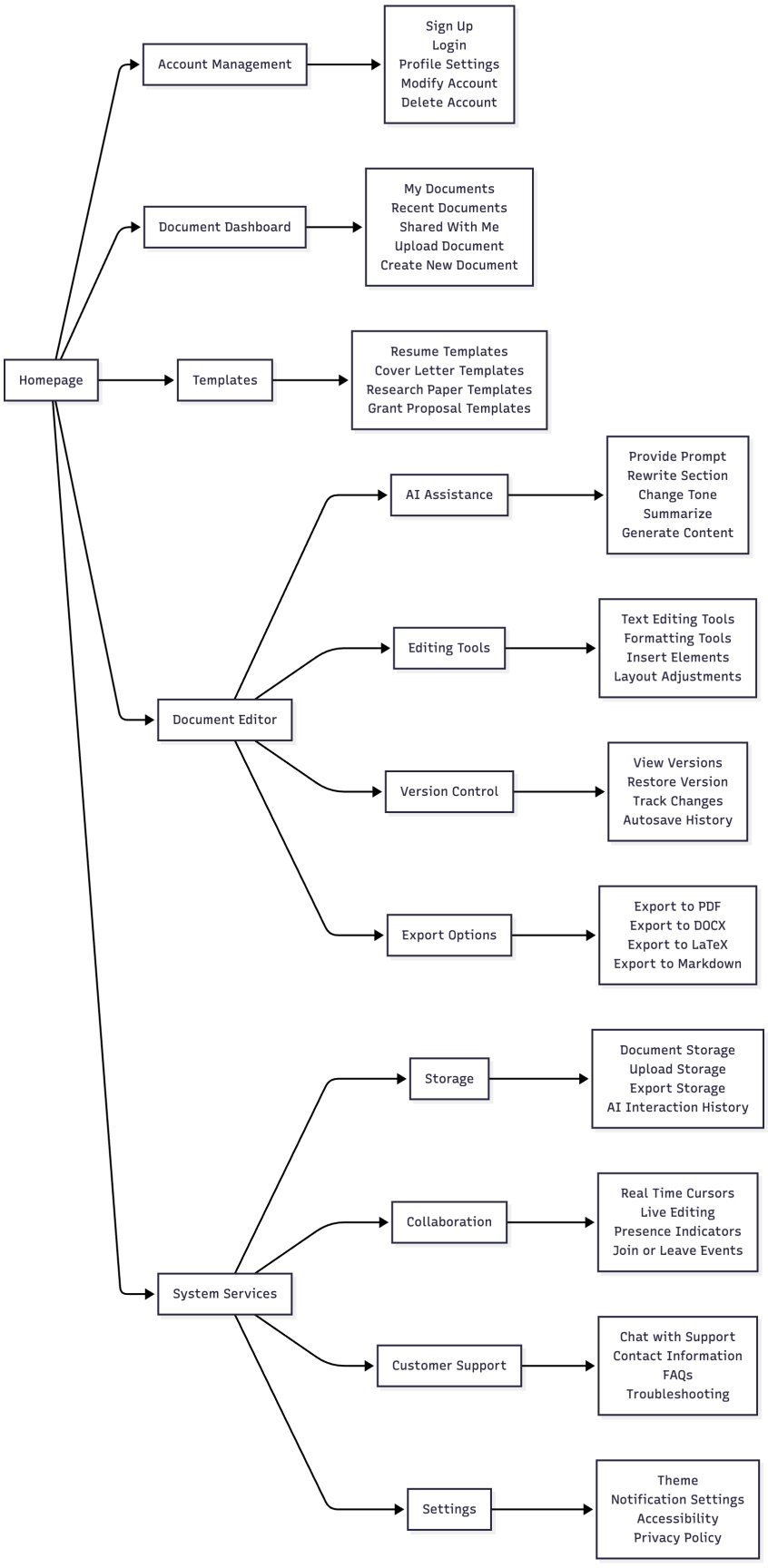

Homepage
Account Management
Sign Up
Login
Profile Settings
Modify Account
Delete Account
Document Dashboard
My Documents
Recent Documents
Shared With Me
Upload Document
Create New Document
Templates
Resume Templates
Cover Letter Templates
Research Paper Templates
Grant Proposal Templates
Document Editor
AI Assistance
Provide Prompt
Rewrite Section
Change Tone
Summarize
Generate Content
Editing Tools
Text Editing Tools
Formatting Tools
Insert Elements
Layout Adjustments
Version Control
View Versions
Restore Version
Track Changes
Autosave History
Export Options
Export to PDF
Export to DOCX
Export to LaTeX
Export to Markdown
System Services
Storage
Document Storage
Upload Storage
Export Storage
AI Interaction History
Collaboration
Real Time Cursors
Live Editing
Presence Indicators
Join or Leave Events
Customer Support
Chat with Support
Contact Information
FAQs
Troubleshooting
Settings
Theme
Notification Settings
Accessibility
Privacy Policy

Fig. 1.4 Information Architecture Diagram

# User Interface Wireframe(s)/Screenshot(s)

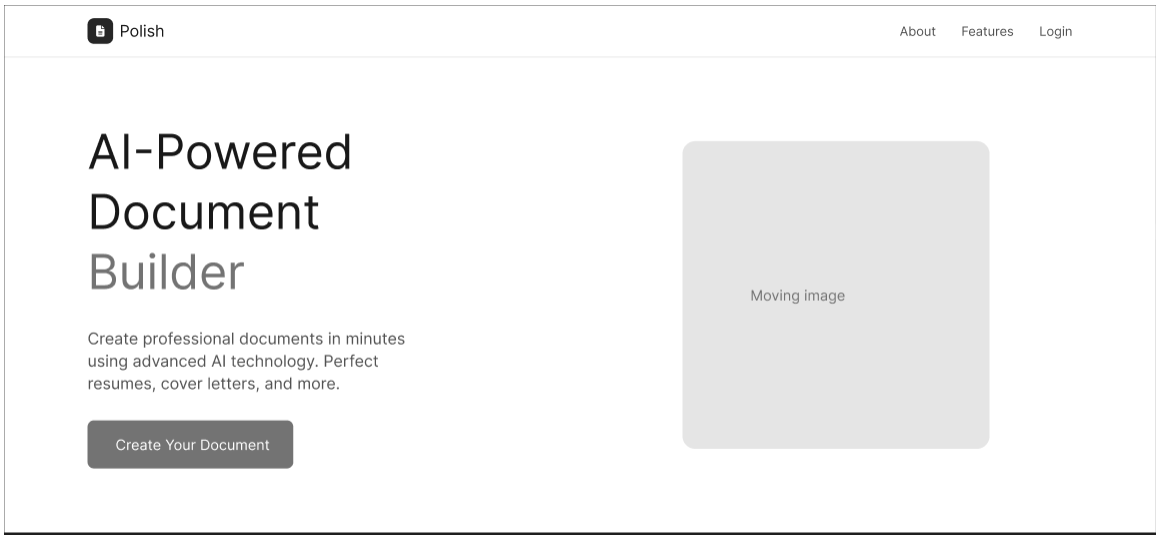



Fig. 2.1 Landing Page

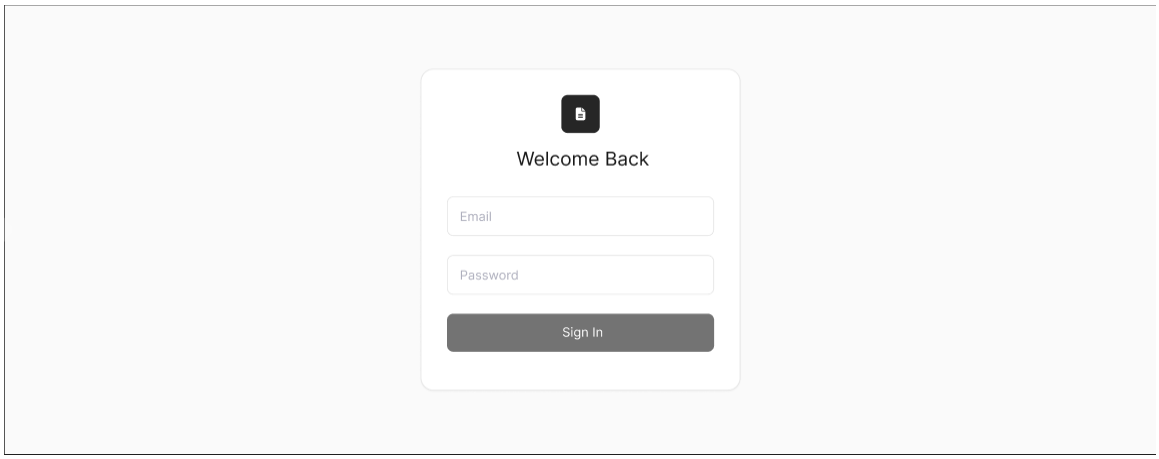



Fig. 2.2 Login Screen

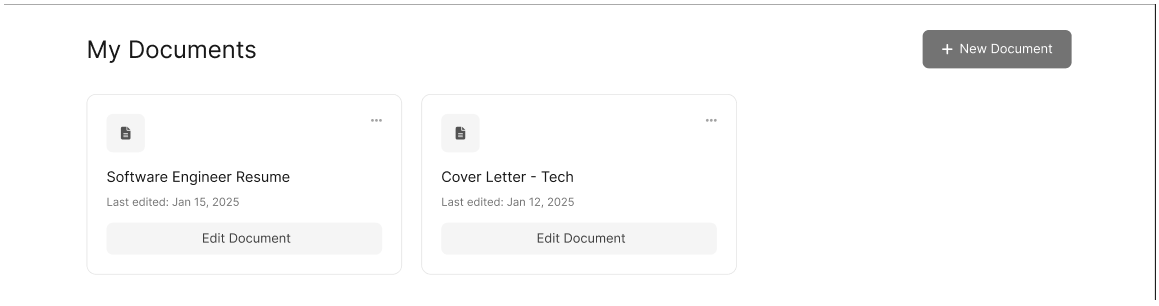



Fig. 2.3. Document Selection

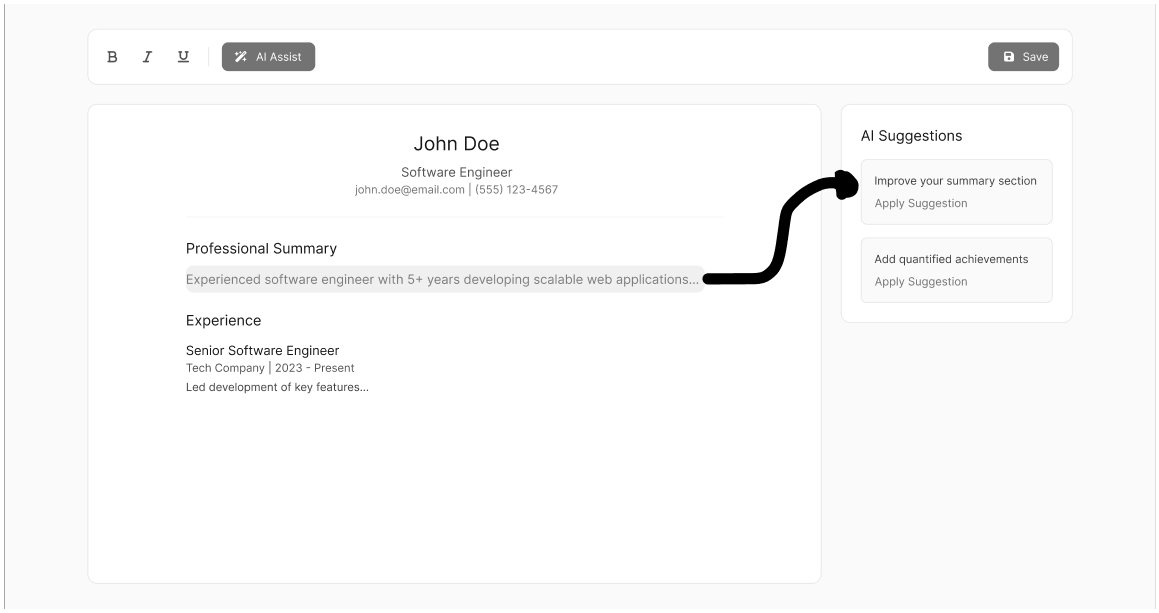


Fig.2.4. Document Editing

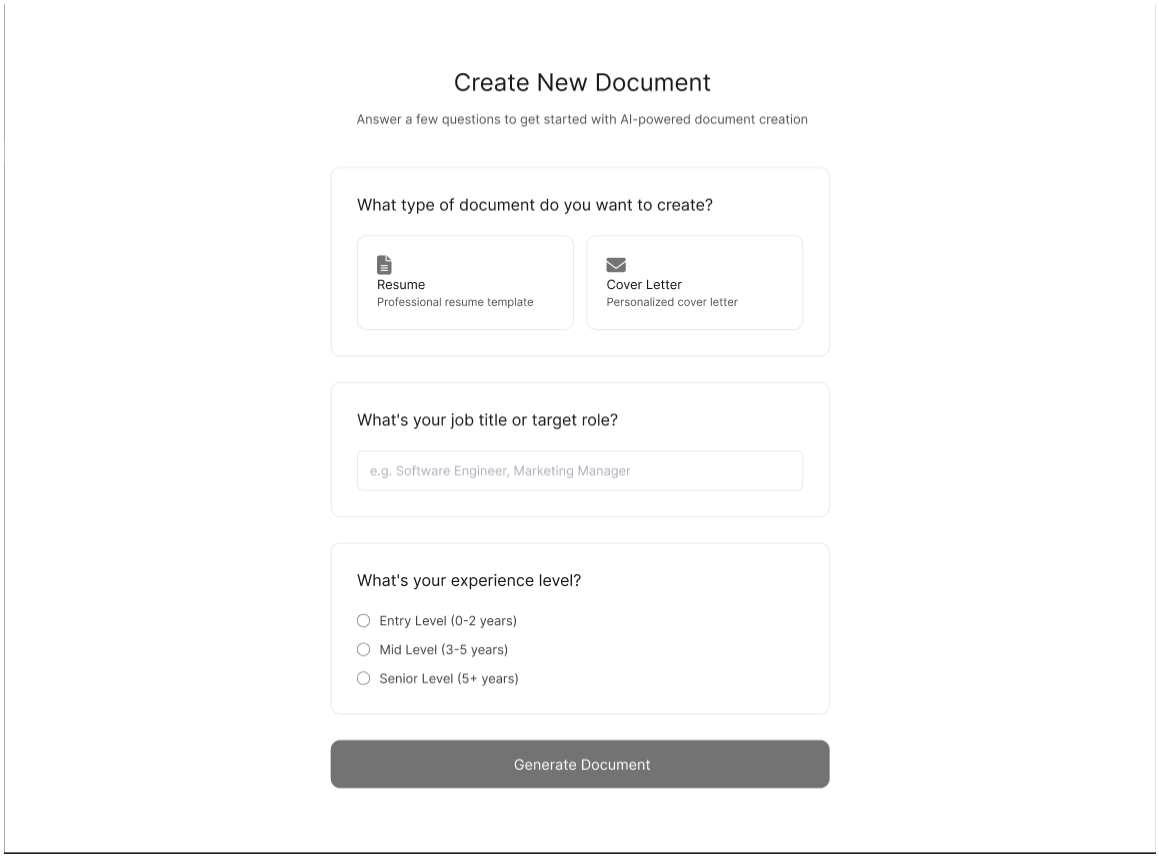



Fig. 2.5. Create new document

# Design Summary

Polish is a cloud based, AI assisted document editor built as a **Single Page Application (SPA)** using **React and TypeScript**. It connects to a **Node.js and Express** backend hosted on **Azure App Service**, integrating multiple Azure services to handle editing, versioning, authentication, storage, and export.

The frontend manages all user interactions uploading files, submitting prompts, applying AI edits, and exporting final documents while the backend coordinates the actual processing and communication with Azure OpenAI, Blob Storage, and the database. Through **Azure SignalR**, edits and AI responses stream to the user interface in real time, creating an interactive editing experience that feels instantaneous.

Data flow follows a clear route:

**Browser → SPA (HTTPS/WebSocket) → Express API → Azure Services (OpenAI, Blob, SQL, SignalR, AD B2C, Key Vault, App Insights).**

This structure ensures both scalability and security while maintaining low latency (<3 s per request) and high availability (99.9% uptime target).

On the frontend, components like *FileUpload*, *EditorCanvas*, *AIPromptPanel*, *VersionSidebar*, and *ExportModal* each manage a distinct part of the user journey from import to AI assisted editing to export. On the backend, modular services such as *FileService*, *AIService*, *VersionService*, *ExportService*, *TemplateService*, and *AuthService* isolate functionality for maintainability and faster iteration.

The system's **architecture and data design** follow normalization and security best practices. Files are stored in Azure Blob Storage, metadata and change logs reside in Azure SQL, and authentication runs through Azure AD B2C. Monitoring and metrics are captured via Azure Application Insights.

This distributed yet tightly integrated architecture allows Polish to deliver:

- **Real time AI assistance** for document improvement
- **Reliable autosave and version control**
- **Seamless export across formats (DOCX, PDF, LaTeX)**
- **Enterprise grade security** with managed secrets and identity
- **Cross platform accessibility** through a web based interface

In essence, Polish unifies the efficiency of AI with the familiarity of a word processor, transforming how users write, refine, and deliver documents all within a secure, scalable Azure environment.

# Design Rationale

Initial Exploration

At the start of the project, the team explored different ways to build an AI powered document editor that could handle real time collaboration, text generation, and secure storage. We began with a simple web app that called external APIs for text generation, but as the design matured, we realized that **scalability, latency, and privacy** would quickly become major issues. This led us to design a **cloud integrated system** that could manage AI assistance, file processing, authentication, and export through a unified backend.

Platform Selection

We evaluated **Google Cloud**, **AWS**, and **Azure** for hosting and AI integration. While all three offered strong capabilities, **Azure** was chosen because it provided direct access to:

- **Azure OpenAI** for text generation and editing
- **Azure SignalR** for real time streaming
- **Azure AD B2C** for secure authentication and user management

Using these within one ecosystem reduced integration overhead and long term maintenance risks.

Backend Design

We compared **Node.js/Express** with **Python/FastAPI**. Although Python was familiar for machine learning tasks, **Node.js** was chosen because:

- Its **event driven, non-blocking** design supports concurrent streaming.
- Shared **TypeScript** code across frontend and backend simplified debugging.
- It ensured **low latency performance** for WebSocket based operations.

The trade off was the need to handle asynchronous code carefully, but responsiveness and modularity outweighed that cost.

Frontend Framework

For the frontend, we compared **React**, **Angular**, and **Vue**.

**React** was selected for its:

- **Component based structure**, fitting our modular UI (FileUploader, EditorCanvas, AIPromptPanel)
- Strong **ecosystem and tooling**
- Easy integration with **state management and WebSocket communication**

Its main drawback a steeper learning curve and complex setup was acceptable given the long term flexibility it provides.

Data and Storage

We decided to separate storage for scalability and performance:

- **Azure Blob Storage:** for large files and exports
- **Azure SQL / Cosmos DB:** for metadata, versions, and audit logs

This avoided performance bottlenecks and reduced storage costs. Alternatives like storing files directly in the database were rejected due to inefficiency. Using managed Azure services increased reliability at the cost of slightly higher setup complexity.

Authentication and Security

Instead of building a custom user system, we integrated **Azure AD B2C**, which offers:

- Built in **identity management and MFA**
- **Compliance** with enterprise security standards
- Seamless connection with other Azure services

We later added **Azure Key Vault** for secret management and **Application Insights** for observability, improving security and monitoring across all services.

Trade offs and Final Design

Each choice balanced control, cost, and simplicity. We rejected:

- **Monolithic designs**, which limited scalability
- **Serverless setups**, which introduced cold start latency

The final design is **modular, cloud native, and adaptable**, ensuring performance under 3 seconds per request, while supporting future expansion such as custom AI models or on premise deployments.

---

# 6. API Design - OpenAPI 3.1.0

## Overview

The backend API is fully described using **OpenAPI 3.1.0** with **47 production-grade endpoints**. Interactive documentation:

- Swagger UI
- ReDoc

## 6.2 Key Features

- **Auth0 JWT** (RS256) with custom claims: permissions, role
- **Server-Sent Events (SSE)** with **SignalR fallback** for token streaming
- **Operational Transformation–ready PATCH** at PATCH /docs/{id} using JSON Patch + OT
- **Rate limiting**: 100 req/min per user, 10 LLM calls/min
- **CORS**: https://polish.app, http://localhost:3000
- **Auto-generated SDKs**: TypeScript-Fetch, Python, Go, Java (OpenAPI Generator)

## 6.3 Endpoint Summary (Selected Highlights)

| Method | Endpoint | Purpose | Real-time |
|---|---|---|---|
| **POST** | /llm/stream | Token-by-token AI response (SSE) | Yes |
| **PATCH** | /docs/{id} | Real-time collaborative edits | Yes |
| **POST** | /docs/{id}/export | PDF / DOCX / LaTeX / Markdown export (+BibTeX) | |
| **GET** | /realtime/negotiate | SignalR connection upgrade | Yes |
| **POST** | /webhooks | Register webhook for event notifications | |
| **GET** | /webhooks | List registered webhooks | |
| **GET** | /docs | List user documents | |
| **POST** | /docs | Create a new document | |
| **GET** | /docs/{id} | Retrieve specific document | |
| **DELETE** | /docs/{id} | Delete document | |
| **GET** | /docs/{id}/versions | List all document versions | |
| **POST** | /docs/{id}/versions/{versionId}/restore | Restore specific version | |
| **POST** | /llm/suggest | Get AI writing suggestions | |
| **POST** | /llm/summarize | Summarize document | |
| **GET** | /llm/templates | Fetch AI-generated template starters | |
| **GET** | /auth/me | Get current user profile (Auth0) | |
| **POST** | /auth/logout | Invalidate session (client-side) | |
| **GET** | /audit | Admin activity logs | |

Full specification: openapi.yaml committed to main branch.

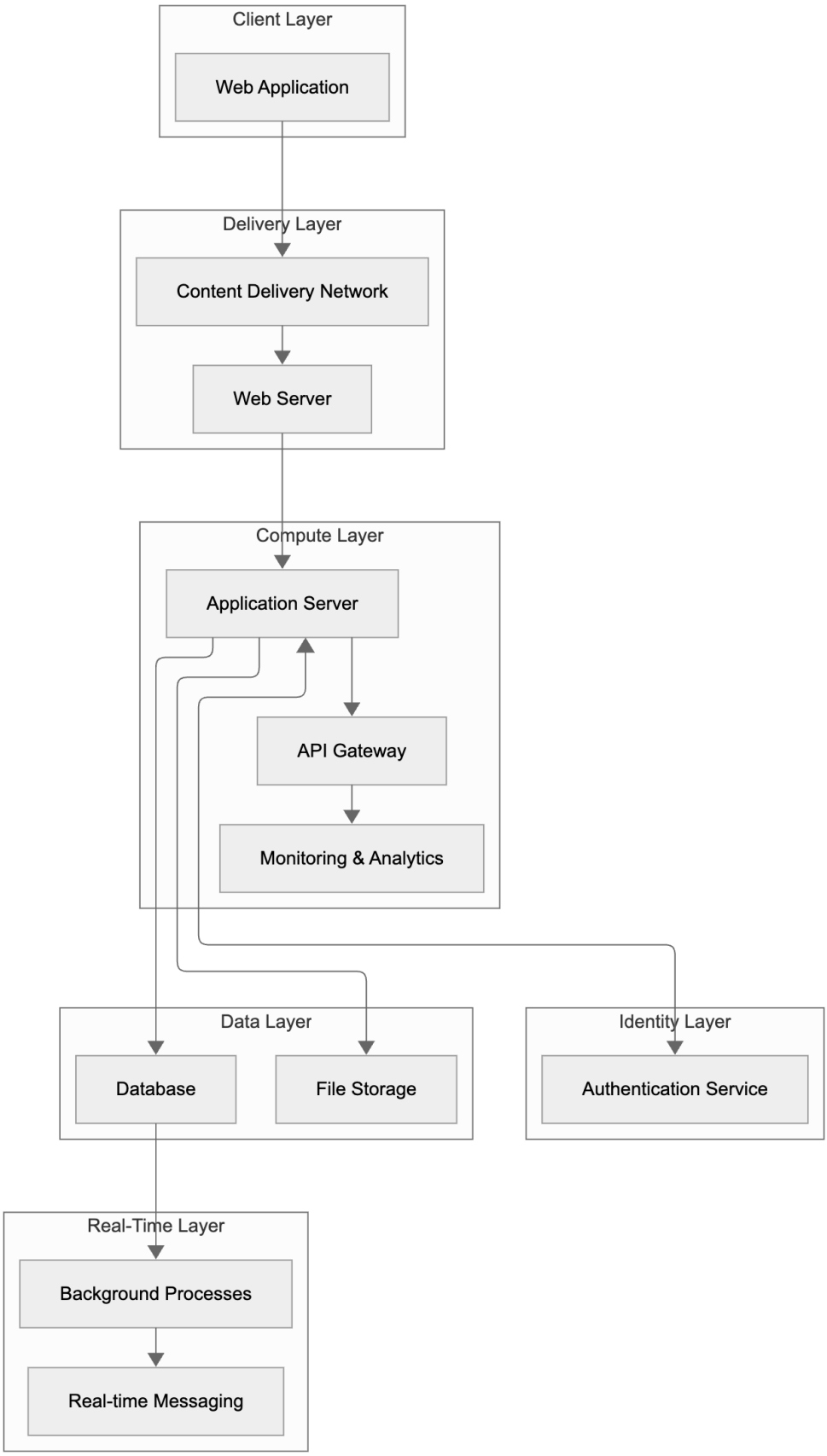
# 7. Database Design Azure Cosmos DB (NoSQL)

## 7.1 Containers & Partition Strategy

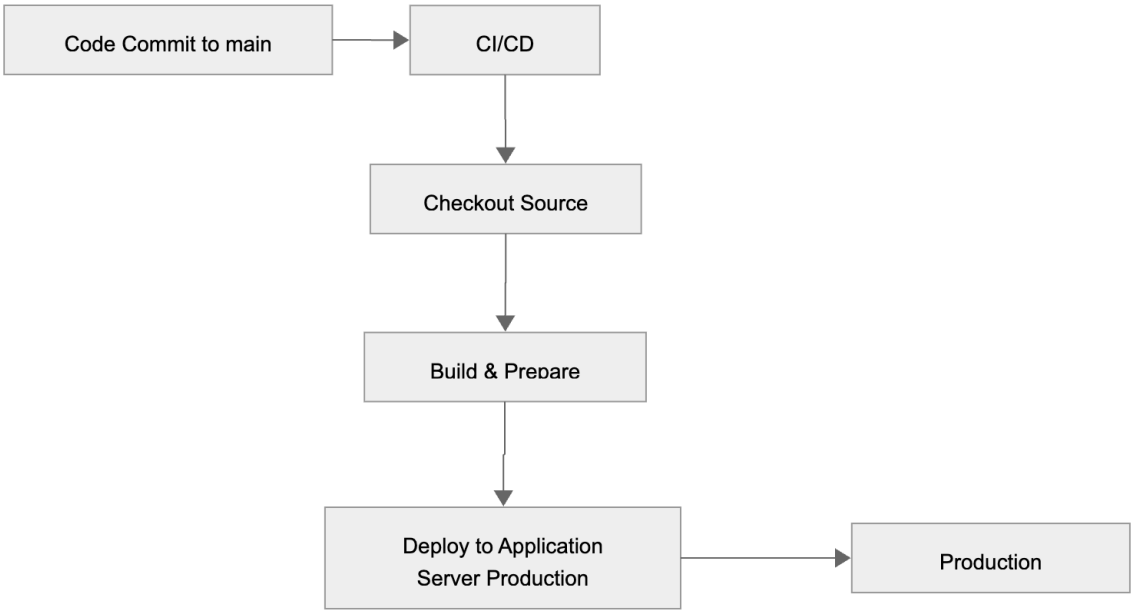
| Container | Partition Key | TTL | RU/s | Indexes |
|---|---|---|---|---|
| Users | /id | | 400 | email (unique) |
| Documents | /ownerId | | 1200 | /title, /updatedAt |
| Versions | /documentId | 30 days | 800 | /timestamp |
| AIInteractions | /documentId | 90 days | 600 | /model, /tokensUsed |
| Templates | /category | | 400 | singleton per category |
| AuditLogs | /date | 365 days | 1000 | /userId, /action |

# 10. Deployment Topology & CI/CD

## 10.1 Architecture Diagram


Client Layer
Web Application
Delivery Layer
Content Delivery Network
Web Server
Compute Layer
Application Server
API Gateway
Monitoring & Analytics
Data Layer
Database
File Storage
Identity Layer
Authentication Service
Real-Time Layer
Background Processes
Real-time Messaging

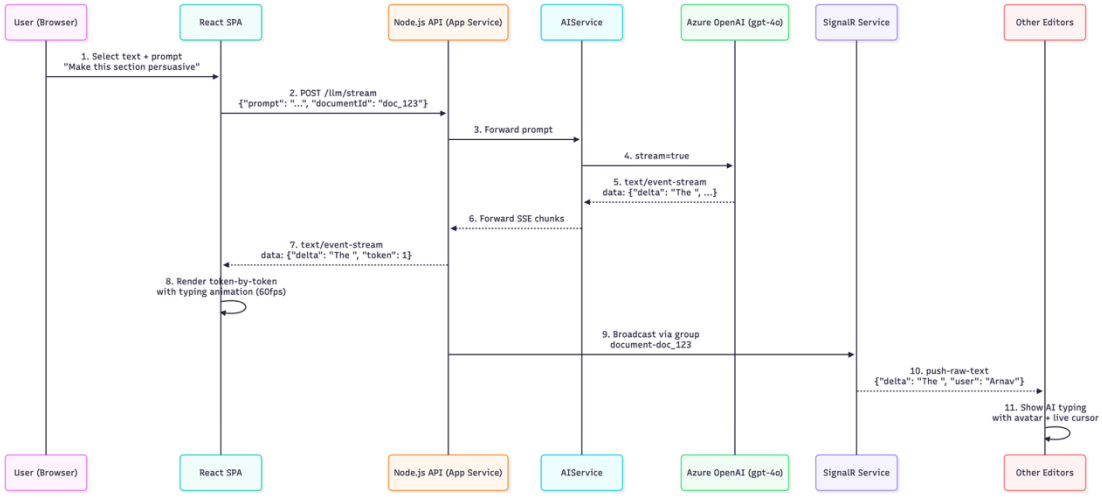
## 10.2 CI/CD Pipeline



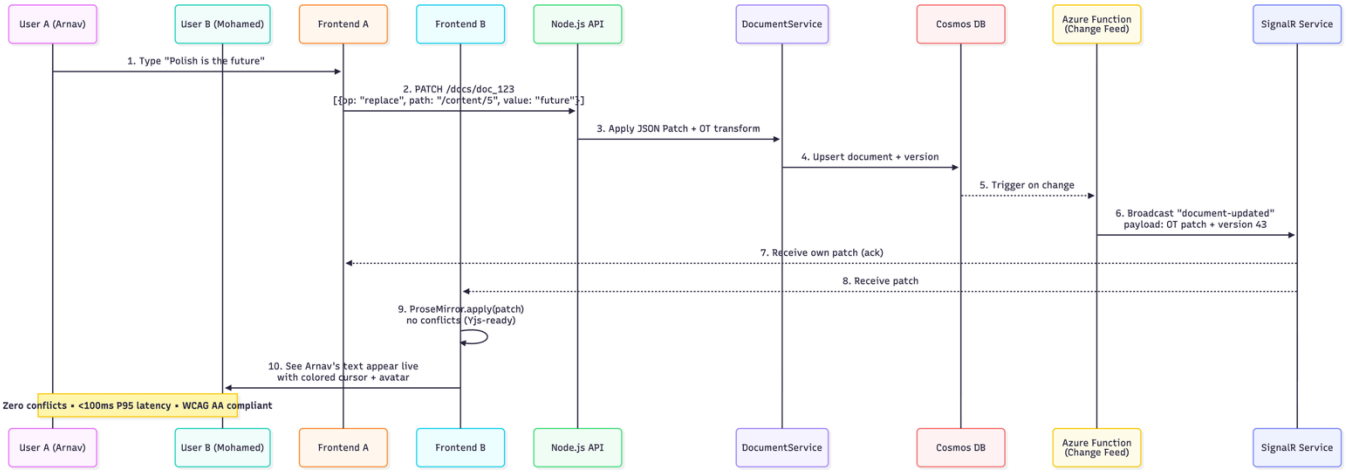

# Sequence Diagrams

## 8. Real-Time Architecture

## 8.1 Sequence Diagram  AI Streaming (Fig 8.1)

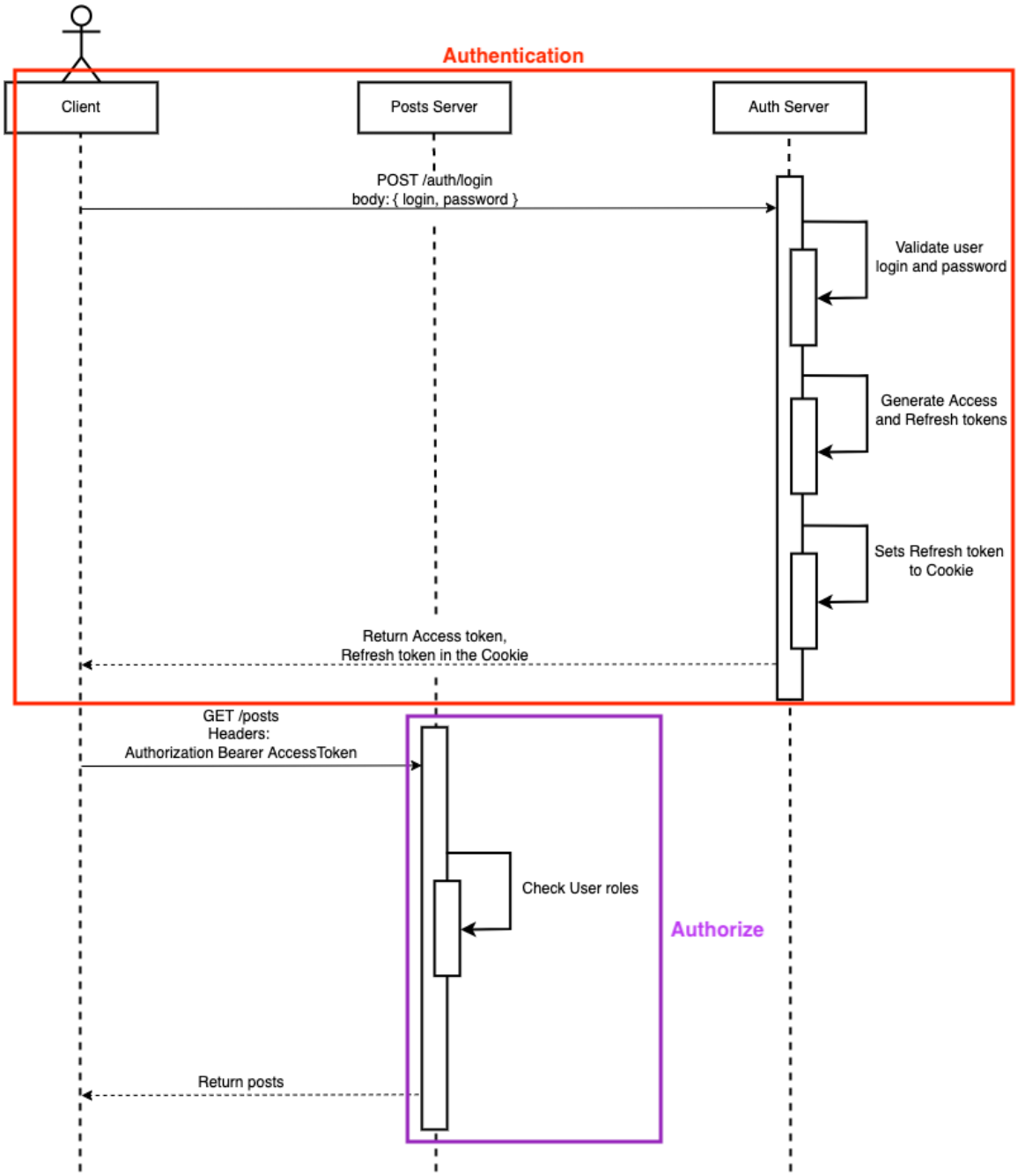


## 8.2 Sequence Diagram  Collaborative Editing (Fig 8.2)




# 11. JWT Authentication

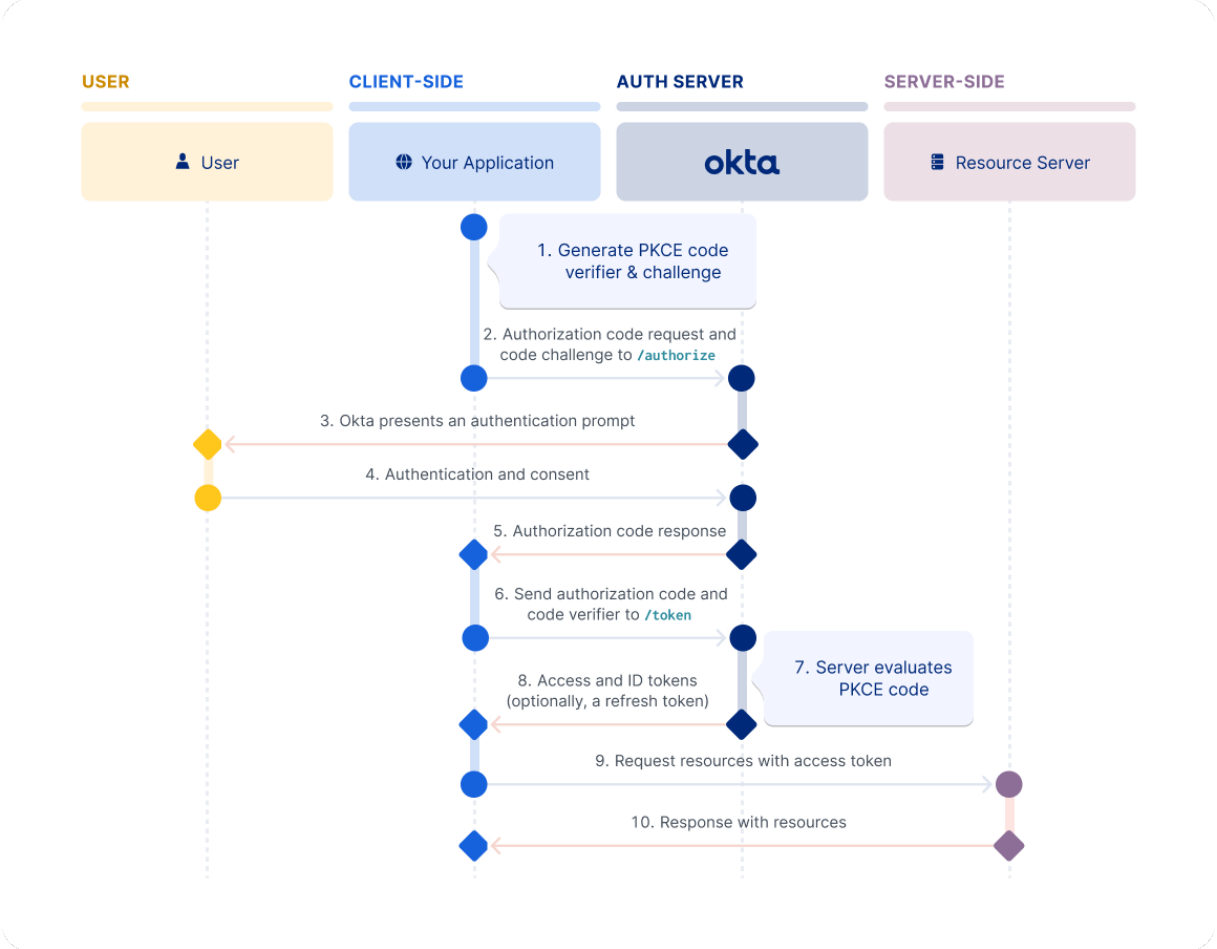
A user receives a token that represents their identity and access rights. This sequence shows the steps a user goes through when logging in. The user provides their login details, the system confirms their identity, and a token is created for them. This token is returned and acts as their “pass” inside the platform. Whenever they perform actions like editing or uploading documents, this token proves who they are.




JWT Authorization Sequence Diagram

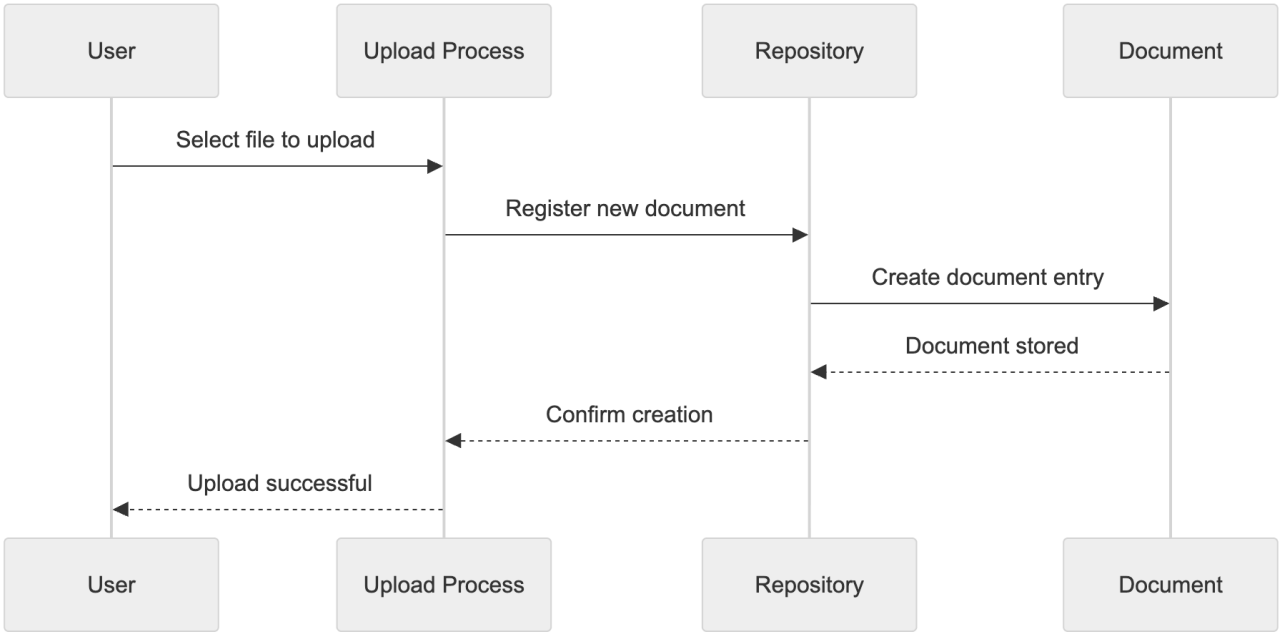
# 12. OAuth2 Authentication

The user logs in using a trusted external provider, like google, github, facebook etc.



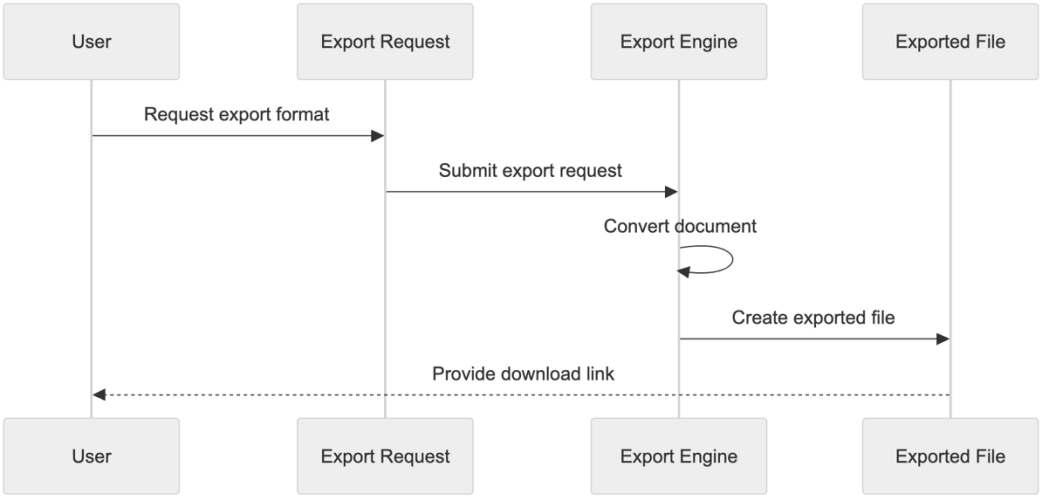

OAuth2 Sequence Diagram

# Document Upload to Azure Cosmos DB



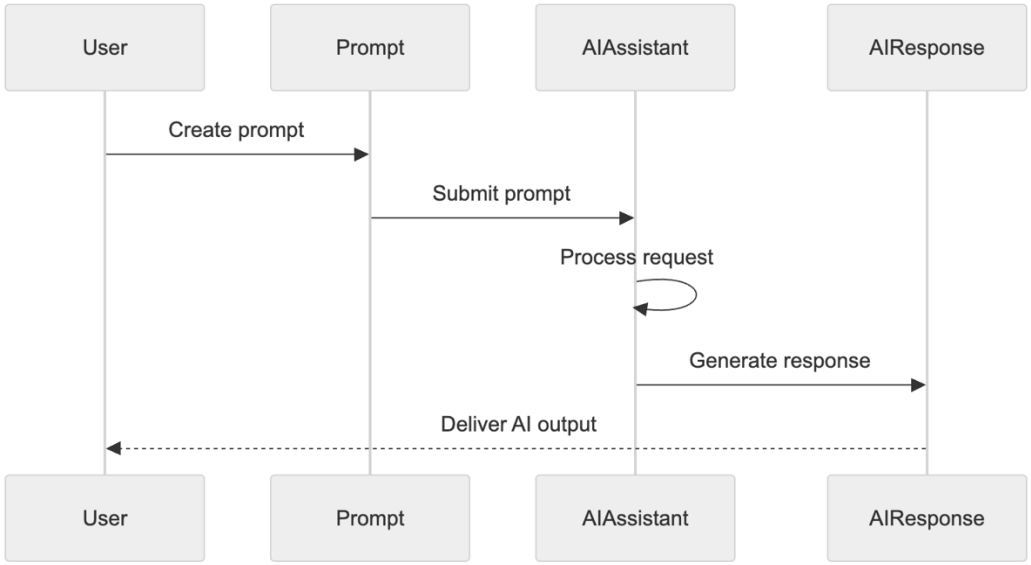

Document Upload to CosmosDB

# 13. Export Process




Export Process Sequence Diagram

# 14. LLM Service




LLM Service Sequence Diagram