Product Design

Team UNT's Best

Walid Esmael, Matthew Norman,
Arnav Verma, Mohamed Babiker

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, 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 Updated ER Diagrams Updated Information Architecture Diagram	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 Blob Storage – 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 Diagram(s)

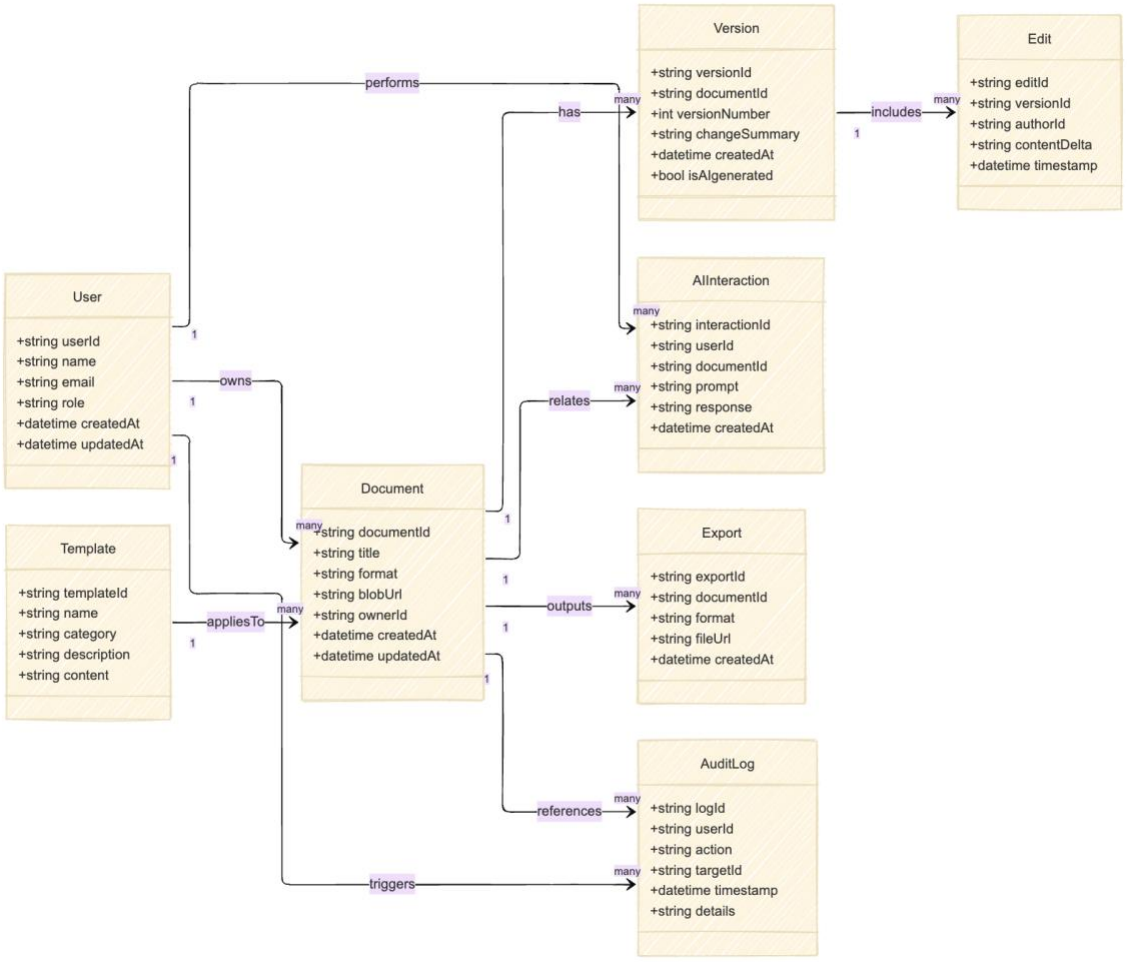

Fig 1.1 Core Backend Services Class Diagram

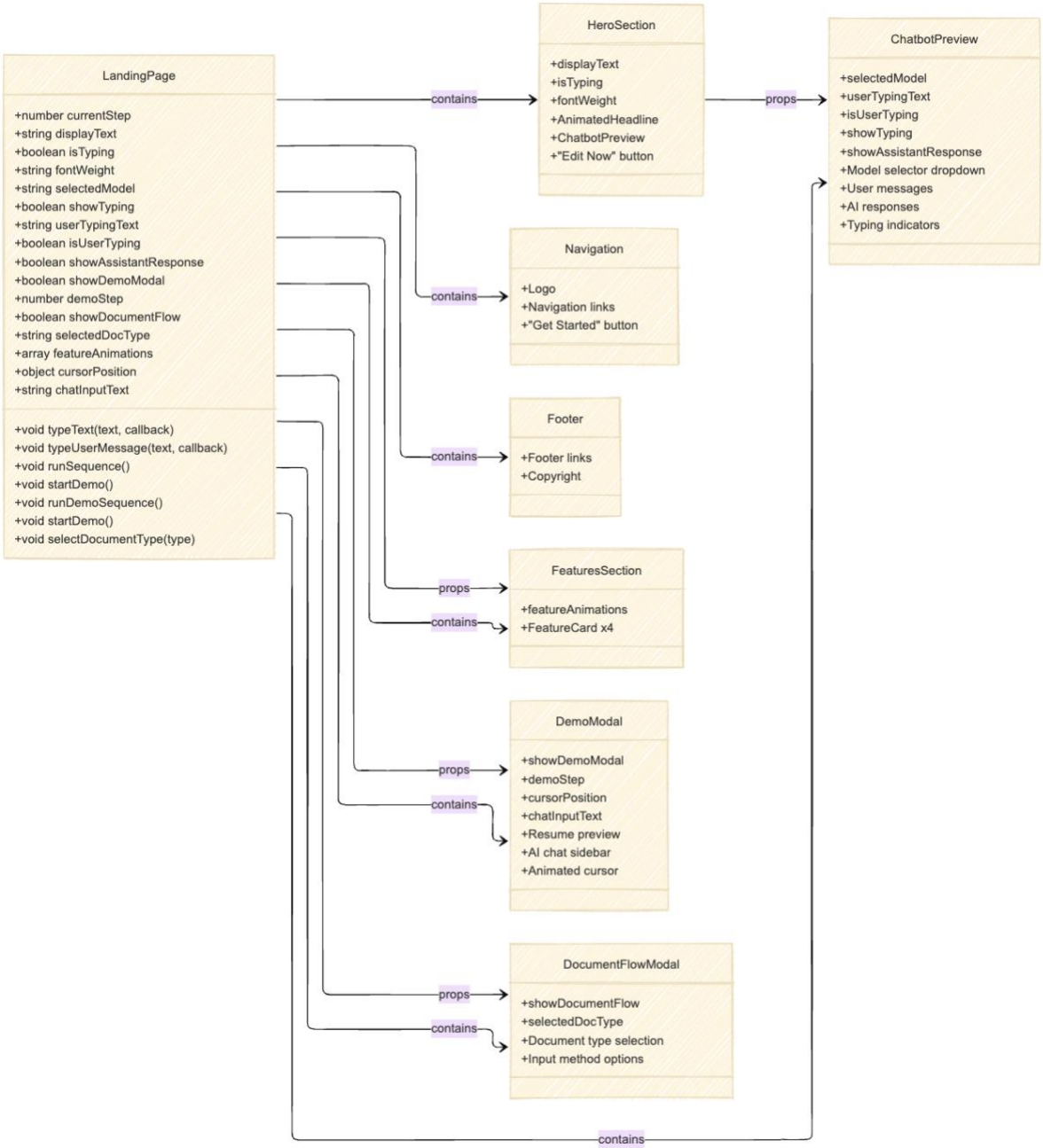

Fig 1.2 Front end class diagram

ER Diagram(s)

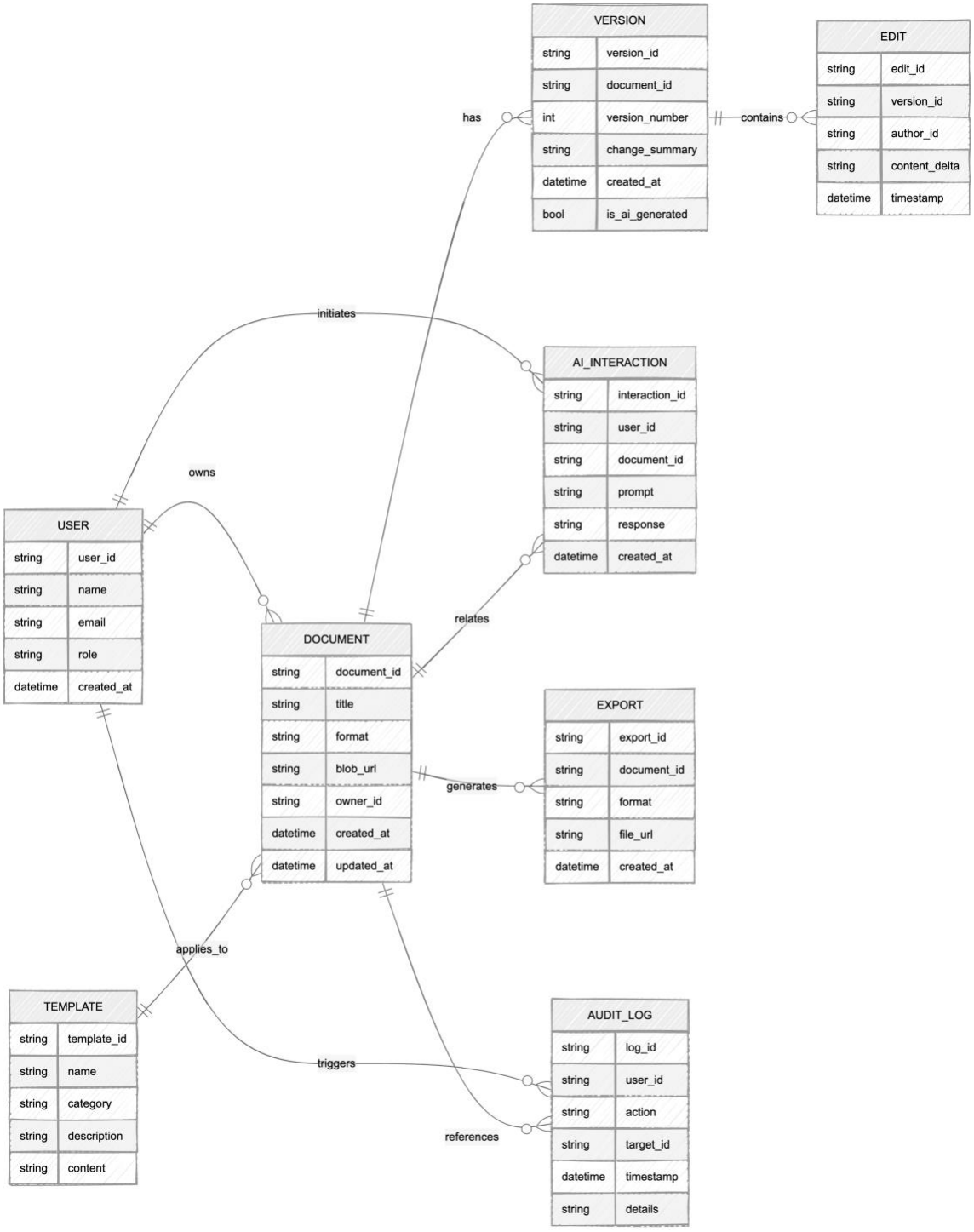

Fig 1.3 ER Diagram

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

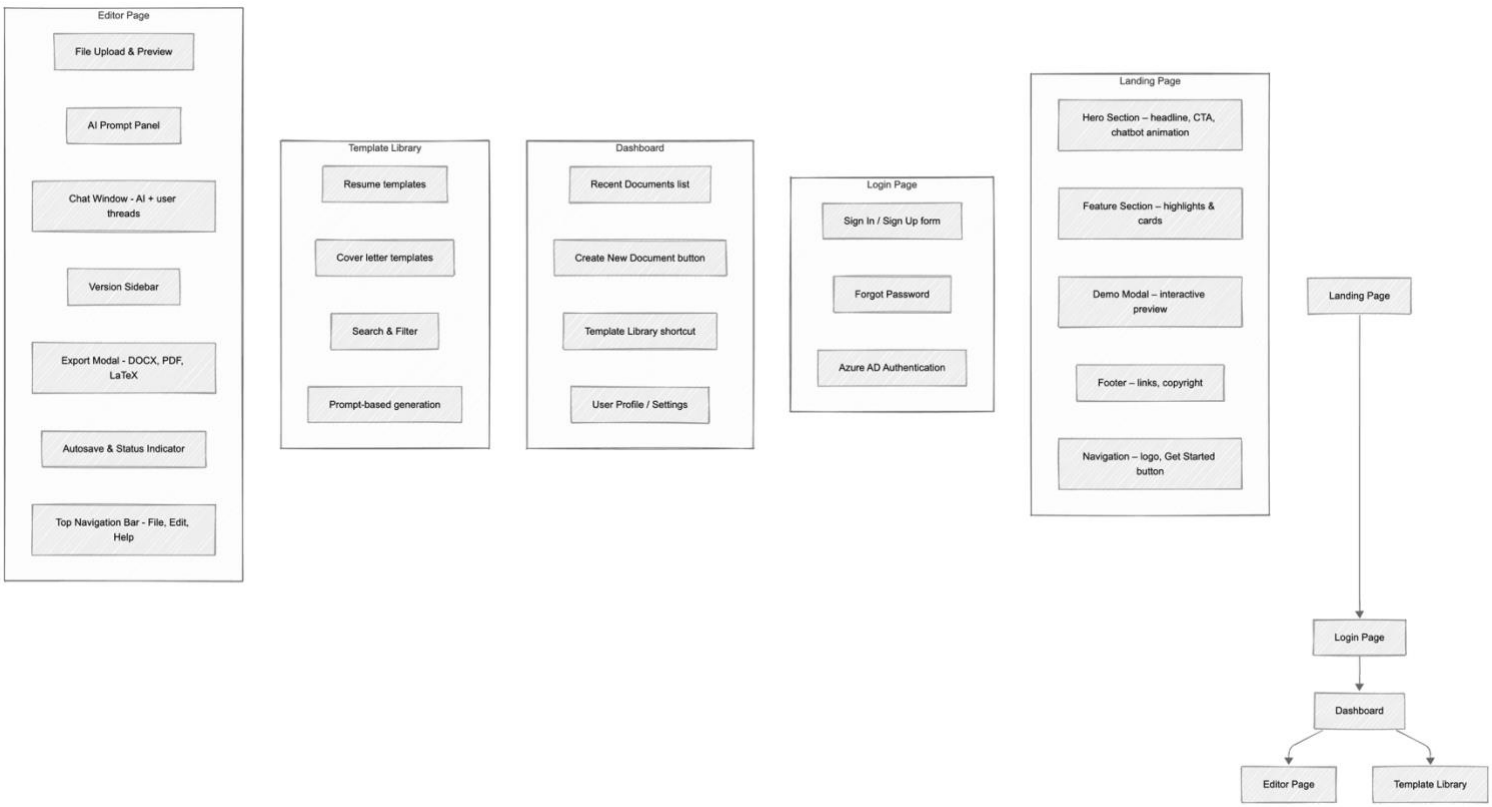

Fig. 1.4 Information Architecture Diagram

User Interface Wireframe(s)/Screenshot(s)

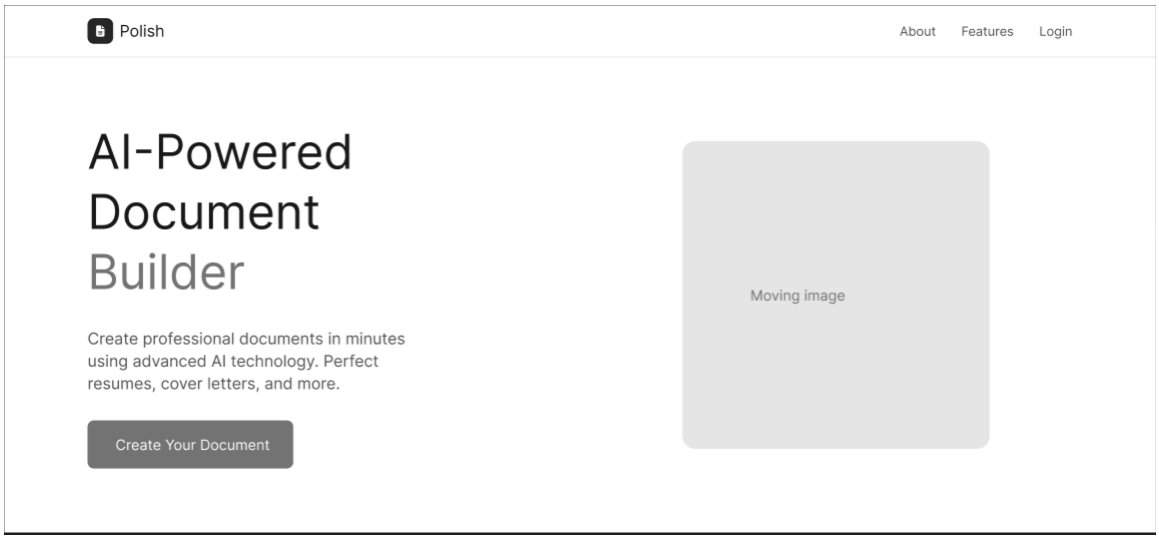

Fig. 2.1 Landing Page

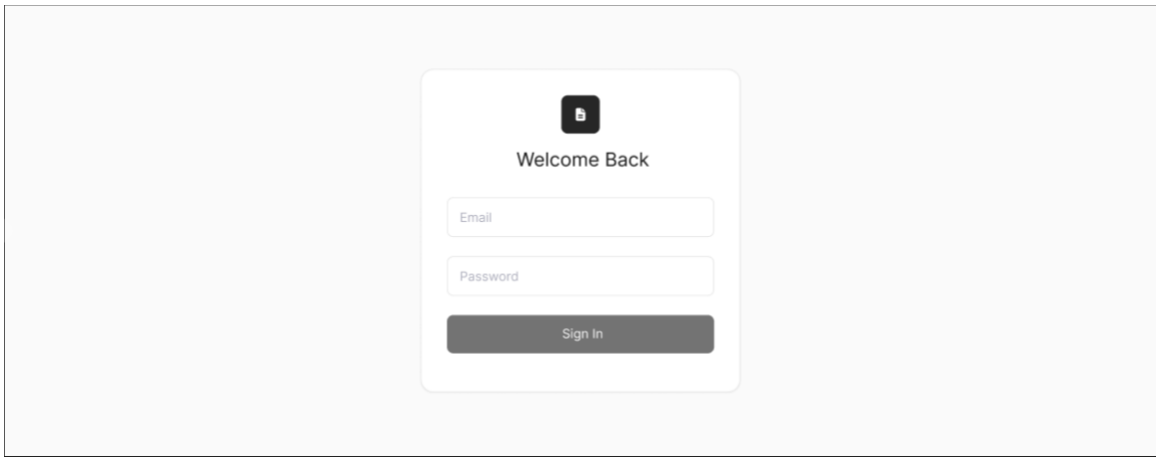

Fig. 2.2 Login Screen

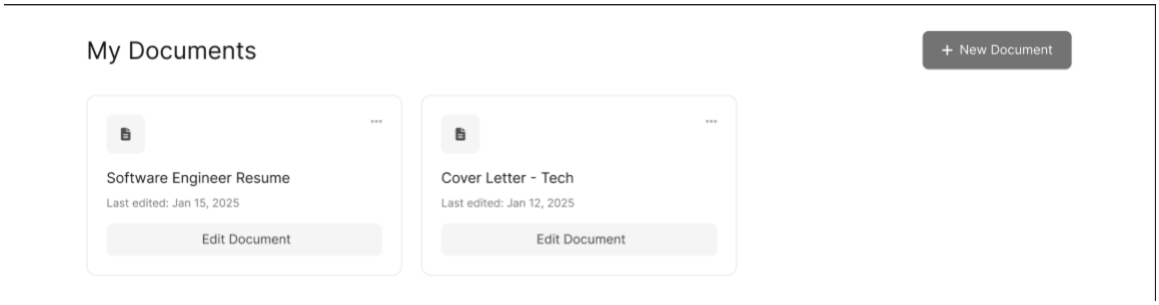

Fig. 2.3. Document Selection

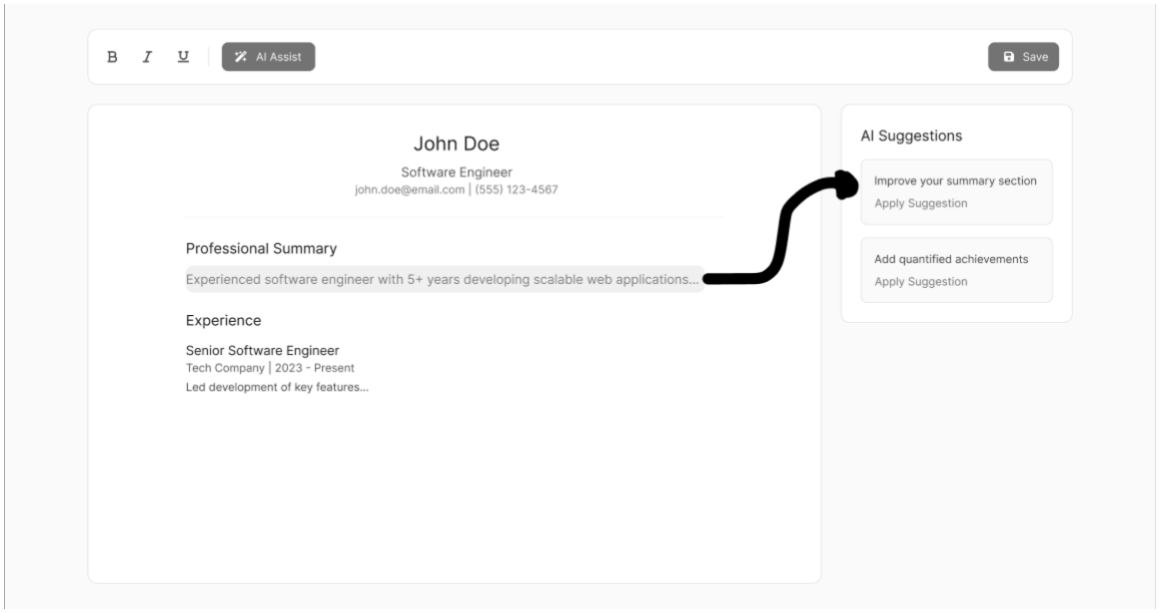

Fig.2.4. Document Editing

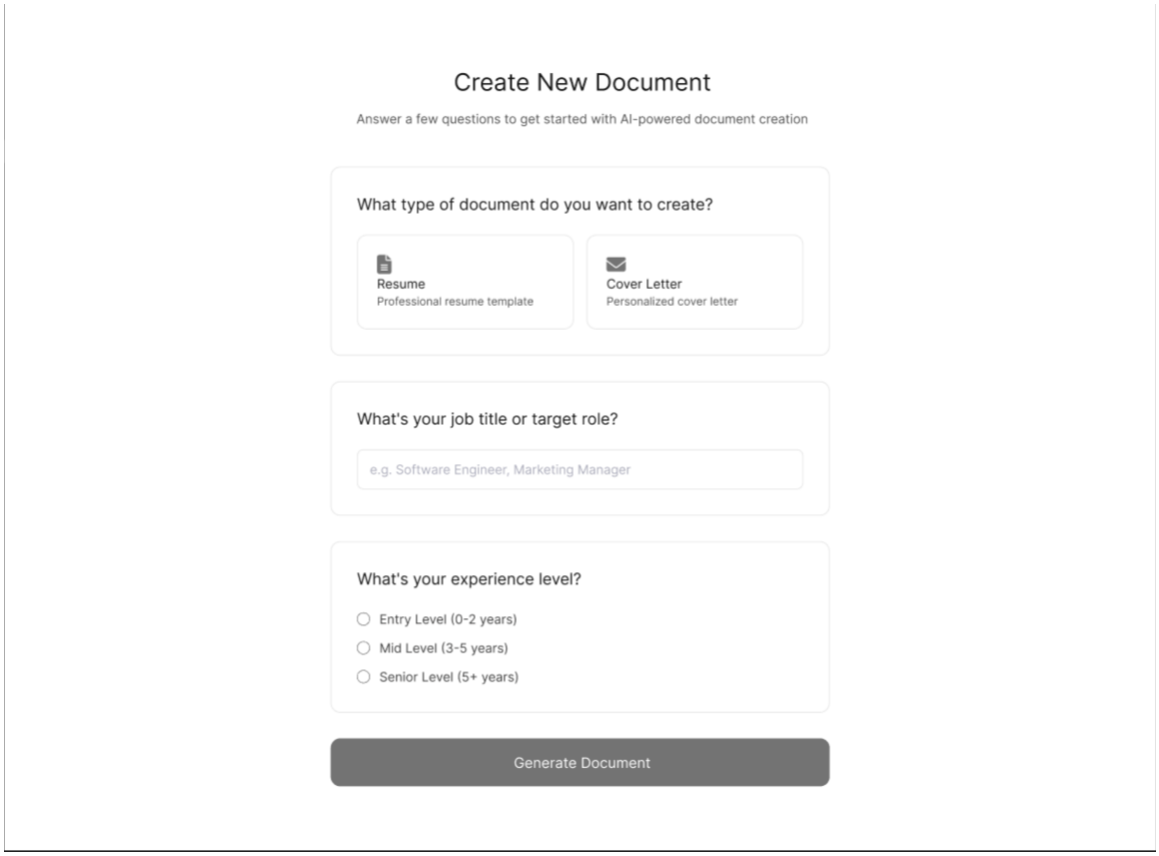

Fig. 2.5. Create new document

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.