Requirements Document
Sprint 3

Polish: AI-Integrated Document Editor

UNT's Best

Walid Esmael
Matthew Norman
Arnav Verma
Mohamed Babiker

Date: December 3, 2025

Revision History

Date	Section / Task	Contributor(s)	Details
09/30/2025	Project Overview	Mohamed Babiker, Arnav Verma	Wrote complete overview covering product vision, Azure architecture, and user centered design; refined document flow and tone.
10/01/2025	Problem Statement	Mohamed Babiker	Authored full section on AI tool fragmentation and Polish’s integrated workflow; edited for clarity and coherence.
10/02/2025	System Requirements	Arnav Verma, Walid Esmael	Defined cloud stack and platform components; Arnav consolidated and aligned entries with architectural design.
10/04/2025	User Profile	Mathew Norman	Composed target audience, pain points, and goals; ensured consistency with UI/UX intent.
10/06/2025	List of Features, Functional Requirements, Non Functional Requirements	Arnav Verma	Drafted all core features (F1–F10), authored and prioritized user stories (R1–R10), and defined NF1–NF6.
10/08/2025 – 10/10/2025	Sprint 1 Scope and Progress	Mohamed Babiker	Outlined sprint goals, deliverables, and highlights; Arnav refined final wording and presentation.
10/09/2025 – 10/11/2025	Testing Summary	Walid Esmael	Authored full section detailing objectives, approach, environment, risks, and outcomes.
10/10/2025 – 10/12/2025	Challenges and Resolutions	Arnav Verma, Mohamed Babiker	Documented async and UI animation issues; Mohamed validated fixes during testing.

Date	Section / Task	Contributor(s)	Details
10/12/2025	Sprint Velocity and Collaboration	Mathew Norman	Compiled sprint metrics, collaboration process, and tool usage summary.
11/16/2025	JWT Authentication Integration, OAuth Integration	Arnav Verma	Implemented RS256 JWT validation middleware, route protection, and claims based authorization.
12/01/2025	LLMService Implementation	Mathew Norman	Built real Azure OpenAI interactions including streaming edits, summarization, and template generation.
12/04/2025	References and Formatting / Final Integration	Walid Esmael, Mohamed Babiker	Final proofreading, formatting, and integration of all sections into the approved deliverable.

1. Project Overview

Polish is a cloud-based, AI-integrated document editor designed to eliminate the inefficiencies of fragmented workflows between AI writing tools like ChatGPT or Grammarly and traditional editors such as Microsoft Word or Google Docs. By unifying AI-driven content generation, real-time editing, and multi-format export into a single web application, it prevents disruptions from copying, pasting, and reformatting. Users can upload documents in DOCX, PDF, or LaTeX formats, interact with AI via natural language prompts for rewrites, structural changes, or formatting, and see dynamic updates in the editor with visual tracking and autosave every 30 seconds. The platform emphasizes responsiveness, accessibility per WCAG standards, and seamless versioning for recoverability.

Built as a Single Page Application (SPA) using React.js and TypeScript on the frontend, with a Node.js and Express backend, Polish is hosted on Microsoft Azure for scalability and reliability. The backend coordinates microservice-style modules:

1. AIService integrates with Azure OpenAI for text generation
2. FileService manages uploads and storage in Azure Blob
3. VersionService tracks modifications in Azure SQL or Cosmos DB
4. ExportService handles outputs in PDF, DOCX, or LaTeX
5. Authentication: OAuth login with Google or GitHub and JWT based authorization

Additional services like Azure Key Vault for credential protection and Application Insights for performance monitoring ensure 99.9% uptime, low latency, and observability. Data design features a normalized relational schema for entities like Users, Documents, and Versions, with large files offloaded to blob storage for efficiency.

Guided by a user-centered philosophy, Polish minimizes context-switching to enhance creative flow, allowing commands like "make this section two columns" or "rewrite in a professional tone" with instant application. Development follows an agile sprint methodology, with initial focus on architecture, UI, mock AI, and testing for cross-browser compatibility. Ultimately, Polish redefines AI-era document editing as an intelligent, integrated workspace that boosts productivity and precision for tasks like resumes, research papers, or cover letters.

---

2. Problem Statement

In recent years, AI writing tools have become powerful but fragmented. Professionals and students often rely on large models such as ChatGPT or Grammarly to help them draft, edit, or polish their work yet these tools live outside the environments where actual documents are created. Users must copy text between browser tabs, paste outputs into Word or Google Docs, and then manually repair formatting errors introduced during the process. This constant back-and-forth breaks concentration, disrupts workflow rhythm, and increases the risk of version errors or data loss.

Traditional document editors, meanwhile, remain isolated from modern AI capabilities. While they support templates and grammar checks, they lack real time, context-aware assistance that understands both content and layout. When writers want AI to adjust tone, rephrase paragraphs, or re-structure a résumé section, they must leave the document environment entirely turning what should be a seamless creative flow into a disjointed sequence of microtasks. For time-constrained users like job seekers or students preparing submissions, this fragmentation costs efficiency and creativity.

Polish addresses this disconnect by embedding AI directly into the document editing workflow. Instead of switching tools, users can interact with intelligent models inside a single, responsive interface that preserves formatting, tracks edits, and exports instantly to common formats like DOCX, PDF, and LaTeX. The problem, therefore, is not simply “making AI write better,” but removing the barriers that prevent people from using AI naturally while they write. By integrating real-time AI prompting, live previews, and formatting preservation, Polish aims to unify the currently scattered stages of professional document creation into one cohesive, intelligent workspace.

---

3. System Requirements

Category	Requirement
Cloud Platform	Microsoft Azure App Service
Frontend	React.js & TypeScript
Backend	Node.js & Express
Database	Azure Cosmos DB / Azure SQL
AI Integration	Azure OpenAI Service (Imagine Cup compliant)
Storage	Azure Blob Storage
Authentication	Azure Active Directory B2C
Real-time Updates	Azure SignalR Service
Monitoring	Azure Application Insights
Security	Azure Key Vault for secret management
Browsers Supported	Chrome 90, Firefox 88, Safari, Edge
Minimum RAM	4 GB (local dev environment)
Connectivity	Stable Internet connection required

4. User Profile

- **Audience:** Students, professionals, and job seekers creating resumes, cover letters, or reports.
- **Skill Level:** Basic-to-intermediate computer literacy.
- **Pain Point:** Time lost switching between AI tools and editors.
- **Goal:** Efficient, integrated document creation.

5. List of Features

ID	Feature	Description
F1	AI-integrated document editor	In-editor AI prompting with live results.
F2	Universal file support	Import DOCX, PDF, LaTeX with format preservation.
F3	Live editing & visual feedback	Real-time AI changes via SignalR streaming.
F4	Change tracking & version control	Autosave every 30 s and version history.
F5	Instant multi-format export	Export to DOCX, PDF, LaTeX in one click.
F6	AI template generation	Resume/cover letter templates from prompts.
F7	Dynamic format adjustment	Natural-language layout tweaks (“make skills section two-column”).
F8	Cross-platform access	Browser-based UI works on Windows/macOS/Linux.
F9	User authentication	Secure login via Azure AD B2C.
F10	Analytics and telemetry	Usage and performance insights via App Insights.
F11	OAuth login	Users can sign in using Google or GitHub and receive a secure JWT session.
F12	Upload service	Handles file uploads, validation, and Blob Storage persistence.
F13	LLM integration service	Provides AI powered writing assistance, summaries, templates, and live token streaming.

6. Functional Requirements (User Stories)

ID	User Story	Priority
R1	As a user, I can upload DOCX, PDF, or LaTeX files to begin editing without format loss.	1
R2	As a user, I can prompt AI directly inside the editor to receive contextual assistance.	1
R3	As a user, I see AI changes render instantly with visual diff feedback.	1
R4	As a user, I can export my document in multiple formats with one click.	1
R5	As a user, I can track AI and manual edits for review and rollback.	1
R6	As a user, I can retain original formatting after AI edits.	1
R7	As a user, I can generate custom templates (e.g., “modern software resume”).	2

ID	User Story	Priority
R8	As a user, I can modify layout using natural-language prompts.	2
R9	As a user, I can access documents from any device.	2
R10	As a user, I can maintain separate document versions.	3
R11	As a user, I can log in using Google or GitHub so that I can access my documents securely.	1
R12	As a user, my requests are authorized using JWT so that only permitted actions can be performed.`	1
R13	As a user, I can upload documents that are validated and stored for editing.	1
R14	As a user, I can receive AI generated suggestions, edits, and templates from the LLM Service.	

---

7. Non-Functional Requirements

ID	Requirement	Description
NF1	Performance	AI responses and processing ≤ 3 s for smooth workflow.
NF2	Reliability	99.9 % uptime via Azure HA infrastructure.
NF3	Security	All data encrypted in transit and at rest (Azure Key Vault and AD B2C).
NF4	Usability	User adapts within 2 minutes without training.
NF5	Formatting Accuracy	Preserve 100 % layout during AI edits and exports.
NF6	Cross-Platform	Identical functionality across Windows/macOS/Linux.
NF7	Security	All authenticated requests must verify JWT signatures and enforce role based access control.
NF8	AI Performance	Streaming responses from the LLM Service must begin within 1 second.
NF9	Upload Performance	Uploads up to 5 MB must complete within 2 seconds.

---

8. Sprint Scope and Progress

Goal ID	Description	Status
G1	Implement landing page with animations and navigation.	Complete
G2	Develop split-screen editor (UI and mock AI chat).	Complete
G3	Simulate AI responses and chat animations.	Complete
G4	Create export modal for DOCX/PDF/LaTeX downloads.	Complete
G5	Add responsive and accessibility features.	Complete
G6	Set up mock API routes (/api/chat, /api/export).	Complete
G7	Begin basic test suite for UI and API mocks.	Complete
G8	Setup for backend integration (Sprint 2 target).	Complete
G9	Implement OAuth login with Google and GitHub	Complete
G10	Add JWT authorization for protected routes and editor access	Complete
G11	Build LLM Service for AI suggestions, summaries, and templates	Complete
G12	Add Upload Service for DOCX, PDF, and LaTeX with Blob Storage	Complete
G13	Expand automated tests for authentication, uploads, and LLM	In Progress
G14	Begin partial backend API implementation from OpenAPI specification	In Progress

Completed Deliverables

• OAuth login flow using Google and GitHub
• Backend generated JWT with roles and permissions
• JWT middleware for route protection and editor access
• LLM Service supporting summaries, suggestions, and streaming
• Upload Service integrated with Azure Blob Storage
• Additional automated tests for authentication, uploads, and AI operations
• Postman Collection updated with Sprint 3 endpoints
• Partial API progress completed (subset of 47 endpoints)

Not Completed

• Full OpenAPI endpoint coverage
• Complete collaborative editing API and real time integration

Sprint 3 Highlights

• Unified authentication system replaced vendor based Auth0
• End to end JWT security across document, LLM, and upload routes
• Successful LLM streaming and AI content generation
• Improved backend structure with modular services

• Expanded test coverage for critical backend flows

1. Testing Summary

Testing Objectives

- Confirm smooth end-to-end flow: upload → edit → export.
- Validate integration of real-time AI assistance and template generation.
- Ensure cross-platform stability, fast response (<3s), and secure operation.
- Maintain layout accuracy and accessibility compliance (WCAG 2.1).

Test Approach

- **Methodology:** Agile-based iterative testing over three sprints.
- **Sprint 1:** UI stability, AI prompting, live editing, file export.
- **Sprint 2:** Formatting reliability, change tracking, template customization.
- **Sprint 3:** Scalability, performance, and accessibility validation.
- **Testing Types:** Regression, functional, UI, performance, accessibility, and security.
- **Automation Tools:** Playwright and aXe introduced in Sprint 2 for regression and accessibility testing.

Test Environment

- Cloud-based QA setup mirroring Microsoft Azure production deployment.
- Testing across multiple browsers (Chrome, Firefox, Safari, Edge) and screen sizes (320–1920 px).
- Focus on maintaining performance consistency and responsive design.

Key Features Tested

- Landing Page: Animation timing, modal behavior, responsive layout.
- Chatbot: Animation order, context retention, response timing.
- Editor Page: Layout rendering, AI chat integration, export functionality.
- API Routes: Request handling, latency validation, error resilience.
- Accessibility: Keyboard navigation, color contrast, screen reader support.

Risks and Mitigations

Risk	**Impact**	**Mitigation**
AI response delays	High	Caching, API optimization, performance monitoring
Data security breach	High	Role-based access control, encryption, security audits
Formatting inconsistencies	Medium	Regression and format-preservation tests
Azure downtime	Medium	Failover mechanisms, autosave buffers
Low user adoption	Low	Feedback-driven design, onboarding guides

Expected Outcomes

- Consistent and stable performance across devices and browsers.
- Seamless AI-assisted editing without data or formatting loss.
- Successful export to DOCX, PDF, and LaTeX.
- Compliance with WCAG 2.1 accessibility standards.
- Improved confidence in release readiness after each sprint.

---

10. Current Challenges and Resolutions

Issue	Impact	Resolution
Animation timing race between user and AI typing	Out-of-order chat responses	Add async queue control.
Export dialog latency on Firefox	UI stutter	Switch to async/await with loading state.

Sprint Velocity & Collaboration

- **Planned Tasks:** 10
- **Completed:** 8 fully and 2 partially
- **Team Tools:** Trello Board, Figma for UI mockups, and VS Code
- **Communication:** Weekly meetings and group testing sessions for bug review and demo preparation.

11. Extended Functional Requirements (R11–R14)

ID	User Story (As a…)	I want to…	So that…	Priority	Acceptance Criteria
R11	Collaborative editor	See real-time cursors and presence of other users	I can co-author documents without conflicts	Must	SignalR broadcasts cursor position every 100ms; user avatars appear in VersionSidebar
R12	AI user	Receive streaming AI responses token-by-token	I see edits appear as if typed by a human	Must	/llm/stream returns text/event-stream; frontend renders chunks with 60fps
R13	Academic user	Export to LaTeX with full BibTeX bibliography support	My thesis/references preserve formatting	Should	POST /docs/{id}/export with format: latex and includeBibTeX: true returns valid .tex + .bib
R14	Any user	Start from AI-generated templates (resume, cover letter, research paper, grant proposal)	I overcome blank page syndrome	Should	/llm/templates returns 15+ categorized starters; DocumentCreateRequest.templateId applies instantly

12. Extended Non-Functional Requirements (NF7–NF9) – Sprint 1 Final

ID	Category	Requirement	Metric	Verification Method
[illegible]	Accessibility	Full WCAG 2.1 Level AA compliance	0 failures in axe-core + NVDA/VoiceOver	11/11 test cases passed (see Test Tracker)
NF8	Real-time Performance	WebSocket/SignalR latency under 100ms (95th percentile)	≤100ms P95	Application Insights + Chrome DevTools
NF9	API Availability	99.95% uptime excluding maintenance	<4.38h downtime/year	Azure Monitor SLA + Auth0 + Cosmos DB combined

13. System Architecture Updates – Auth & Database

13.1 Authentication Provider

- **Replaced Azure AD B2C** with **Auth0** (decision: better React SDK, free tier, social login)
- Auth0 Application: polish-app-prod
- JWT validation: RS256, audience https://api.polish.app
- Custom claims: permissions: [editor, viewer, admin]

13.2 Database – Azure Cosmos DB (NoSQL)

- Account: polish-cosmos-db
- Database: PolishDB
- Containers (6):
    1. Users (partition: /id)
    2. Documents (partition: /ownerId)
    3. Versions (partition: /documentId, TTL: 30 days for autosaves)
    4. AIInteractions (partition: /documentId)
    5. Templates (singleton)
    6. AuditLogs (partition: /date)
- Change Feed → Azure Functions → SignalR broadcast (real-time updates)
- RU/s: 400 baseline → 6000 burst
- Indexes: /title, /updatedAt, composite /ownerId + updatedAt

14. API Design

- Full OpenAPI 3.1.0 specification committed: openapi.yml
- Live interactive docs for API on Swagger UI
- Rate limiting: 100 req/min per user

15. JWT Authentication Architecture

The authentication layer has been redesigned to eliminate all third-party identity providers. Instead of Azure AD B2C or Auth0, the backend now issues and validates its own JWT tokens using keys stored securely in Azure Key Vault. This change provides full control over the identity model, simplifies integration, and aligns the platform with enterprise self-managed security demands.

Authentication and authorization now operate purely through backend issued access tokens, cryptographically signed using an asymmetric RSA key pair. All user identity, roles, and permissions are stored in Azure Cosmos DB as part of the Users collection documented in Sprint 2’s schema.

15.1 Token Structure and Claims

Every token issued by the backend follows this structure
• Algorithm RS256 using a private signing key stored in Azure Key Vault.

• Public key served through a JWKS style endpoint for client-side verification.
• Token lifetime configurable per environment (default 60 minutes).
Each token contains three groups of claims

Standard claims

iss, sub, aud, iat, exp

Application permission claims

polish.permissions gives granular access such as
• doc.read
• doc.write
• ai.use
• webhook.manage

Role claims

polish.role gives one of
• student
• user
• admin

Roles are sourced from the Users container in Cosmos DB and are updated at login.

No external OAuth profile or upstream identity attributes exist. All identity information is internal.

15.2 Authentication and Token Issuance Flow

A new backend-controlled login process has been implemented

1. User submits email/password to /auth/login.
2. Backend verifies credentials against hashed values stored in Cosmos DB (Users container).
3. Backend loads user profile, role, permissions.
4. Backend signs a JWT with RS256 using the private key from Key Vault.
5. JWT returned to client for inclusion in Authorization header.

Password hashing uses PBKDF2 or bcrypt with configurable salt rounds. Failed logins return uniform 401 responses to avoid information leakage.

15.3 Validation Middleware

A full JWT lifecycle validation layer now protects every secured endpoint. The middleware performs

• Extract bearer token from Authorization header.
• Retrieve cached RSA public key.
• Verify signature, expiration, audience.
• Decode claims and attach to req.user.
• Enforce permission requirements for each endpoint.
• Enforce admin key requirement for elevated operations such as audit access.

Failure modes
• 401 for missing, expired, or malformed tokens.
• 403 for valid tokens lacking required permissions.

This middleware replaces the earlier AD B2C and Auth0 validation logic referenced in Sprint 2.

15.4 Integration Across API Endpoints

All protected endpoints now require a backend issued JWT including

• /docs and /docs/{id}
• /llm/* for real time AI interactions
• /realtime/negotiate for SignalR sessions
• /webhooks for registering external listeners
• /audit for admin actions

Access control is determined by the permission claims in the token.

Example
doc.write is required for editing
ai.use is required for LLM operations
admin role plus X Admin Key is required for audit endpoints

This security model now governs document creation, editing, autosave, versioning, and all AI calls.

15.5 User Storage and Identity Model

User records are stored in the Cosmos DB Users container (id, email, role, permissions, hashed password).

This container previously held Auth0 identities but has been updated to store native credentials and internal identity metadata.

Fields include
• id (UUID)
• email
• hashedPassword
• role
• permissions
• createdAt, updatedAt

The security model is fully owned by the backend and no longer depends on upstream identity metadata.

15.6 Security and Performance Advantages

Migrating away from Auth0 and AD B2C yields
• Zero dependency on third party identity providers.
• No external login redirect flows.
• Faster login (local verification only).
• Lower latency because there is no remote JWKS lookup.
• Total control over claims, roles, and permission expansion.
• Ability to implement enterprise specific RBAC in future sprints.
• Full portability for self hosted or on premise deployments.

RS256 asymmetric cryptography ensures tokens cannot be forged since private key never leaves Key Vault.

15.7 Summary

JWT is now the sole security mechanism for all API interactions.
Identity, authentication, and authorization are entirely backend managed, allowing complete control over user lifecycle, claim design, and system level security.

This new architecture replaces Auth0 everywhere in the design and requirement documents and aligns the system with the move toward a fully self contained Azure backend.

---

16. Editor Backend Service

The Editor backend is now a dedicated service that orchestrates document loading, patch application, collaborative consistency, autosaving, and versioning. It complements the real time flow shown in Section 8.2.

16.1 Document Load Pipeline

When the editor opens /docs/{id}, the backend performs
• Permission check ensuring the user is owner or collaborator
• Fetch document metadata and content from Cosmos DB
• Load the latest committed version number
• Return the ProseMirror JSON structure to the client
This ensures the editor is initialized with the canonical state.

16.2 Operational Transformation Patching

The endpoint PATCH /docs/{id} processes incremental edits from the client. Patches conform to the JSON Patch format defined in the OpenAPI schema.

The backend handles
• Validation: patch operations must target valid paths
• Conflict resolution: last writer wins with cursor reconciliation
• Transformation: normalizing incoming patches into OT compatible ops
• Broadcasting: sending the confirmed patch to all connected clients through SignalR

These rules ensure consistency even under concurrent editing.

16.3 Autosave and Version Control

The editor autosaves every thirty seconds or after a series of edits. The backend
• Computes change count
• Increments version number in Cosmos DB
• Writes to the Versions container (7.1)
• Publishes a Change Feed event for real time subscribers

This preserves a reliable timeline of edits and allows users to restore past versions.

16.4 Collaboration Features

The Editor backend supports
• Cursor tracking broadcasts every 100 ms
• Live collaborator presence
• Multi user edit queues
• Patch replay for late joining collaborators

As described in Section 8.3, these updates appear in the editor via colored carets and presence indicators.

16.5 Error Recovery and Consistency Guarantees

If a patch fails
• The client receives a structured error
• The backend replays the last stable state from Cosmos DB
• The client re syncs with the canonical document

This prevents desynchronization between editor clients.

---

17. LLM Service Architecture

The LLM Service is a new backend subsystem that centralizes AI operations and connects directly with the editor for real time machine assistance. It matches the OpenAPI LLM endpoints.

17.1 Unified LLM Interface

The service offers a unified interface for

• Streaming completions (POST /llm/stream)
• Writer suggestions (POST /llm/suggest)
• Summaries (POST /llm/summarize)
• Template retrieval (GET /llm/templates)

All routes require JWT authentication.

17.2 Streaming Pipeline

For streaming endpoints the service
• Accepts prompt and metadata
• Calls Azure OpenAI's gpt-4o through the Node SDK
• Wraps tokens into SSE events
• Sends deltas to the client in real time
• Falls back to SignalR if the browser does not support SSE

This provides low latency feedback essential for live writing assistance.

17.3 Safety, Filtering, and Guardrails

Before interacting with the model
• Inputs pass through content sanitation
• Oversized prompts are chunked or truncated
• Potentially unsafe outputs are filtered
• Token usage is tracked and stored in AIInteractions container
This ensures predictable behavior under all usage conditions.

17.4 Cost and Rate Controls

The service enforces
• 10 LLM calls per minute per user
• Max token budget per request
• Automatic rejection for runaway prompts
• Logging of model, tokens consumed, and latency

This guards against accidental or malicious resource consumption.

17.5 Document Aware AI Assistance

When used within the editor, the LLM service may
• Read the current document content
• Execute scope aware edits
• Generate structured diffs
• Return them in a format compatible with the Editor backend’s OT engine
This allows the editor to highlight AI suggested insertions or replacements.

17.6 Templates and Starter Generation

The service supports AI generated resume, cover letter, and multi section templates by
• Loading curated prompt templates
• Generating structured outputs
• Caching results to reduce repeated inference

18. Sprint Velocity & Artifacts Delivered

Metric	Value
Story Points Planned	42
Story Points Completed	42 (100%)
Bugs Found → Fixed	3 → 0 open
Lines of Code (backend + frontend)	4,829
GitHub Copilot Acceptance Rate	92%

Final Deliverables
Requirements Document – 11 pages (this version)

1. **Product Design Document** – 15 pages, Revision 8
2. **OpenAPI 3.1.0 Specification** – 47 endpoints, Swagger UI live
3. **Postman Collection + Environment** – 112 requests
4. **Accessibility Test Matrix** – 11/11 GREEN
5. **Cosmos DB Dashboard Screenshots** – 6 containers + change feed
6. **Auth0 Configuration Export** – rules, connections, API

19. References

- OpenAPI Specification: https://github.com/arnav-verma/polish/blob/main/openapi.yaml
- Swagger UI: https://api.polish.app/docs
- Auth0 Dashboard: https://manage.auth0.com/dashboard/us/polish-app-prod
- Azure Portal: https://portal.azure.com/#resource/subscriptions/…/resourceGroups/polish-rg