Sprint 1 Requirements Document

Project Name: Polish: AI-Integrated Document Editor
Team Name: UNT's Best
Team Members: Walid Esmael • Matthew Norman • Arnav Verma • Mohamed Babiker
Date: October 18, 2025

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 (User Stories), Non-Functional Requirements	Arnav Verma	Drafted all ten core features (F1–F10) with detailed descriptions and linkage to requirements. Authored and prioritized all user stories (R1–R10); aligned with Test Plan and sprint deliverables. Defined NF1–NF6 (performance, reliability, usability, and cross-platform); ensured technical alignment.
10/08/2025 – 10/10/2025	Sprint 1 Scope & Progress	Mohamed Babiker	Outlined sprint goals (G1–G8), 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, test environment, risks, and outcomes based on Test Plan.
10/10/2025 – 10/12/2025	Challenges & 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 & Collaboration	Mathew Norman	Compiled sprint metrics, collaboration process, and tool usage summary.
10/18/2025	References & Formatting / Final Integration	Arnav Verma	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. AuthService uses Azure Active Directory B2C for secure authentication.

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 micro-tasks. 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

Category	Requirement
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.

---

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
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

---

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.

---

8. Sprint 1 Scope and Progress

Goal ID	**Description**	**Status**
G1	Implement landing page with animations and navigation.	Complete

Goal ID	Description	Status
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.	Incomplete
G6	Set up mock API routes (/api/chat, /api/export).	In Progress
G7	Begin basic test suite for UI and API mocks.	Complete
G8	Lay groundwork for backend integration (Sprint 2 target).	In Progress

Completed Deliverables

1. **Functional Web Prototype** – Frontend built with React and TypeScript deploys locally and demonstrates all UI workflows.
2. **Landing Page Animations** – Hero typing sequence, feature wave effect, and responsive navigation.
3. **Interactive Chat System** – Simulated AI prompt-response cycle using mock API data.
4. **Document Upload & Preview (Static Mocks)** – Supports DOCX, PDF, and LaTeX samples.
5. **Export Modal & Download Simulation** – Triggers fake downloads to demonstrate export flow.
6. **Accessibility Baseline** – Keyboard navigation, ARIA labels, and contrast testing implemented.
7. **Performance Testing Metrics** – Average load time 1.3 s; mock AI response 2.8 s; animation ≈ 58 fps.
8. **Sprint 1 Documentation Package** – Requirements, Design Doc, User Profile, and Test Plan compiled and approved.

Sprint Highlights

- **Consistent Design Language:** The UI now reflects a unified visual style across landing, editor, and chat sections.
- **Proof of Concept:** Demonstrates feasibility of in-editor AI communication with real-time visual feedback.
- **Stable Architecture:** Mocked API structure mirrors future Azure services, simplifying next integration phase.
- **Cross-Platform Responsiveness:** Verified on Windows, macOS, and Linux browsers (Chrome, Firefox, Edge, Safari).
- **Usability Focus:** Minimal learning curve users can upload, prompt, and export within two minutes of onboarding.

- Azure OpenAI live endpoint mocked locally.

- Version control logic under development.

---

9. 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:** GitHub Projects (Board and Issues), Figma for UI mockups, and VS Code for shared development.
- **Communication:** Weekly meetings and group testing sessions for bug review and demo preparation.

Sprint 2 Plan (Preview)

1. Integrate real Azure OpenAI API for document editing and text generation.
2. Implement autosave and version control via **VersionService** and **Cosmos DB**.
3. Connect Azure Blob Storage for secure file management.
4. Finalize authentication flow with Azure AD B2C.
5. Add template generation and structure adjustment features.
6. Begin unit testing and Cypress end-to-end automation.

---

12. References

- *Azure OpenAI Service Documentation*
- *React and TypeScript Guides (React.dev)*
- *Microsoft Azure App Service Docs*
- *WCAG 2.1 Accessibility Standards*

Approval Signatures

Name	Role	Signature Date
Instructor / TA	Sponsor	
Arnav Verma	Team Member	
Walid Esmael	Team Member	
Matthew Norman	Team Member	
Mohamed Babiker	Team Member	