SENG 429 Project Proposal

Prompt Press

1. Introduction & Overview

Prompt Press is a simple web app for drafting, refining, and publishing blog posts. The site focuses on two spaces: a public Feed for finished work and a Profile workspace for drafts. The goal is a clean path from **idea → draft → optional targeted edits → publish**.

2. Scope

2.1. Pages

• **Feed:** Public, read-only view of published posts. Cards open a full article dialog.

• **Profile:** Private workspace showing your non-published posts (e.g., statuses: pending/created/error).

• **Admin:** Manage tags/metadata, moderate comments, and view simple stats.

2.2. Core Flow (Create → Edit → Publish)

Create:

• Enter a short prompt. The system (AI) generates a draft article in markdown.

• You can publish immediately or edit first.

Edit (optional, targeted):

• Open the draft in Profile. Select (**highlight**) the exact text you want to change.

• Provide a short instruction for how to change it (e.g., “make this more concise”).

• The system updates **only** the selected part and shows the revised draft.

• Can be repeated as needed.

Publish:

• When satisfied - with or without edits - publish the draft.

• The post moves to the public Feed.

3. User Roles

• **Anonymous Reader:** Browse the Feed, open full posts, basic search/filter.

• **Author:** Create drafts, run optional targeted edits (highlight + how), publish posts, delete own drafts.

• **Admin:** Manage tags/metadata, moderate comments, view basic stats.

4. Status

Each post carries a small status to make the flow obvious:

- **pending** - actively generating or re-drafting
- **created** - draft ready to review
- **published** - visible in the Feed
- **error** - an operation failed; message shown for quick fixes

• Publishing sets a timestamp; later edits produce a new draft state.

5. Data Model (Entities)

• **Blog**:{ prompt, aiResult?, status, errorMessage?, publishedAt, createdAt, updatedAt, author }

• **User**:{ email, name, role: "admin" | "author", passwordHash, createdAt }

• **Comment**: { blog, user|null, text, status: "visible"|"hidden", createdAt }

• **Tag**: { name, slug, createdAt }

• **Revision**: blog, user, what, how, createdAt } (records AI edit intents)

6. System Requirements

6.1. Functional Requirements

• **FR-1 Create:** Create a draft from a short prompt; draft saved with status/timestamps.

• **FR-2 Read (Feed):** Anyone views published posts; open full content dialog.

• **FR-3 Read (Profile):** Authors view non-published posts (pending/created/error).

• **FR-4 Edit (Optional, Targeted):** Highlight selection + provide “how”; only that text is updated.

• **FR-5 Publish:** Authors can publish at any time; post appears in Feed with a publish timestamp.

• **FR-6 Status Lifecycle:** pending → created → published (+ error with message).

• **FR-7 Manage:** Authors delete own drafts; Admin manages tags/comments and basic metadata.

• **FR-8 Validation & Errors:** Forms validate required fields; API returns structured JSON errors.

6.2. Non-Functional Requirements

• **NFR-1 Usability:** Two clear spaces (Feed, Profile), readable typography, obvious buttons, keyboard-friendly dialogs.

• **NFR-2 Clarity:** Inline helper text/toasts; status badges; "Publish" available immediately after creation.

• **NFR-3 Performance:** Responsive lists/dialogs; lightweight refetch/polling for freshness.

• **NFR-4 Reliability:** Deterministic, selection-scoped edits; proper HTTP codes (400/404/422/500).

• **NFR-5 Security & Config:** Env-based configuration; role checks for admin operations.

• **NFR-6 Maintainability:** Modular client/server structure, typed models, clean service boundaries.

• **NFR-7 Extensibility:** Room for tags, comments, search, stats without disrupting core flows.

7. Architecture

• **Stack**: MERN (MongoDB, Express/Node, React/Next.js, TypeScript)

• **Frontend**: Routes for Feed, Profile, Admin; dialogs for Create/Read/Edit; forms with validation; lightweight polling.

• **Backend**: REST API; schema validation (e.g., zod); selection-scoped AI updates; status transitions; CORS.

• **Data**: MongoDB collections for Blog, User, Comment, Tag, Revision (optional: Tagging).

• **Structure**: /client (frontend) and /server (backend); env files for config.