SPEC-001-Discord Markdown Formatter SPA

Background

Goal. Build a single-page web app that helps users compose and preview Discord-compatible text using Markdown (bold, italics, underline, strikethrough, headers, lists, code blocks, block quotes, spoilers, masked links, syntax highlighting) and Discord's inline timestamp syntax. The app should generate the exact characters users can paste into Discord with no surprises.

Why now. Discord's official "Markdown Text 101" covers the basics, but users still bounce between docs and trial-and-error. A focused SPA with live preview, snippet templates, and timestamp helpers reduces friction for everyday users, moderators, and content creators.

Primary users. - Everyday Discord users who want quick formatting without remembering syntax. - Community mods/staff drafting announcements with consistent style. - Bot authors/content creators needing reproducible snippets for docs and posts.

Scope (initial). - Live editor + preview rendering as Discord would show it. - One-click copy of the raw Markdown/Discord text. - Timestamp builder (all supported styles, timezone-aware, relative/absolute forms). - Quick-reference panel and examples.

Non-goals (initial). - Logging into Discord or posting messages on users' behalf. - Multi-file projects or long-form documents. - Full WYSIWYG rich-text editing; we stay text-first with helper controls.

Assumptions. - SPA will be public, no authentication. - Mobile and desktop responsive layout. - The preview should emulate Discord formatting closely using client-side parsing; we will not rely on Discord APIs for rendering.

Requirements

Prioritization: MoSCoW (optimize for casual users; creators/mods supported)

Must Have

- Live split-view **Editor** (left) and **Preview** (right) with real-time render of Discord Markdown (bold, italic, underline, strikethrough, spoilers, code span, code block, block quote, lists, headers-like text, masked links, emojis, mentions rendered as plain text, inline math not required).
- **Timestamp Builder**: UI to insert Discord timestamps `<t:epoch[:style]>` for absolute and relative times; presets for common styles (t, T, d, D, f, F, R); timezone selection; copy-to-clipboard of the token.
- **Code Blocks**: language selector for syntax highlight hints (e.g., `` ```js ``, `` ```py ``, `` ```json ``, `` ```ts ``).
- **Quick Reference** panel summarizing supported Markdown and timestamp styles with examples.
- **One-click Copy** of full message and per-snippet copy; clipboard API fallback.

- **No-Account SPA**: runs fully client-side; no message is sent to any server by default.
- **Responsive** layout for mobile (≤375px), tablet, desktop.
- **Accessibility**: keyboard navigation for all controls; WCAG AA color contrast; screen-reader friendly labels.
- **Undo/Redo & Local Draft Persistence** via localStorage.
- **Performance**: initial load ≤150KB JS (gzipped) for MVP, TTI ≤2s on mid-tier mobile.
- **Supported Browsers**: latest Chrome, Edge, Firefox, Safari (last 2 major versions).

Should Have

- **Template Gallery**: common patterns (announcement, rules, spoiler reveal, poll-style list) with insert buttons.
- **Inline toolbar** with quick format buttons (B, I, U, S, `code`, `code block`, spoiler, quote, list).
- **Keyboard Shortcuts** for common formats (e.g., Ctrl/Cmd+B/I/U, fenced code block toggle).
- **Shareable Gist** export/import (manual copy/paste JSON only; no backend required).
- **Light/Dark theme** auto-detect with manual toggle.
- **i18n-ready** structure (copy in en-US for MVP).

Could Have

- **PWA installability** with offline support (static assets + local drafts).
- **Timestamp presets** ("Next Friday 6pm", "In 24 hours") that compute epoch.
- **Image placeholder** helper (explains that images must be uploaded in Discord, not embedded via Markdown).
- **Link validator** for masked links.
- **Export** to PNG of the rendered preview (approximate look, not exact Discord CSS).

Won’t Have (MVP)

- Discord OAuth or sending messages to channels.
- Rich text WYSIWYG that hides the Markdown—text-first UX is intentional.
- Multi-user collaboration.
- Custom fonts/themes beyond light/dark.

Non-Functional

- **Privacy-by-default**: no analytics or remote logging in MVP; optional toggle for privacy-preserving telemetry later.
- **Security**: sanitize preview to avoid XSS from code blocks and links; no HTML injection.
- **Reliability**: no backend dependency; app functional entirely offline after first load if PWA enabled.

Method

High-Level Architecture

- **SPA stack**: React 19 (no Server Components), Vite build, TypeScript, TailwindCSS for UI, Zustand for small state, DOMPurify for sanitizing preview HTML, markdown-it for parsing, and Prism (or Highlight.js) for code block styling. Luxon + @vvo/tzdb for timestamp/timezone features.

- **No backend** in MVP. All logic client-side. Optional analytics via Plausible (self-hosted/cloud), **off by default**; toggle persists in localStorage.
- **Theming**: dark-first with a neutral palette; subtle Discord-like spacing/compactness without brand assets.

Core Modules

1) **Editor** - Textarea (or code editor) with inline toolbar actions that wrap selections with Markdown tokens: `**` `*` `__` `~~` `` ` `` `> ` `- ` `||` etc. - Keyboard shortcuts: Cmd/Ctrl+B/I/U/S, fenced code block toggle (```), quote (Cmd/Ctrl+Shift+.). - Snippet templates insert at cursor (announcement, rules, poll list, spoiler reveal, code sample). - Draft persistence (localStorage with versions).

2) **Parser & Preview** - Uses **markdown-it** configured to match Discord's subset and quirks: - Disable raw HTML (`html: false`). - Enable autolink/strikethrough/blockquote/list/code_fence/code_inline. - Custom plugin to treat `__underline__`, `||spoiler||`, masked links `[text](url)`; preserve triple backticks with a language hint. - Post-process code blocks to attach Prism/HLJS classes for highlighting. - Sanitize rendered HTML via **DOMPurify** with a tight allowlist; no iframes, scripts, events.

3) **Timestamp Builder** - UI inputs: date, time, timezone, style selector (t, T, d, D, f, F, R), relative/absolute toggle, preview. - Output token: `<t:UNIX[:STYLE]>` inserted at cursor and copyable. - Helpers: common presets ("Now", "+5m", "Tonight 7 PM", "Next Friday 18:00").

4) **Quick Reference** - Collapsible panel listing syntax cheatsheet and timestamp styles with live examples driven by current time.

5) **Analytics (optional)** - Tiny switch in settings. When enabled, load Plausible script with `data-domain` and record minimal events (`pageview`, `copy_full`, `copy_snippet`, `insert_timestamp`).

6) **Accessibility & i18n** - All controls reachable by keyboard; ARIA labels; prefers-reduced-motion; translation keys stored in a JSON map with en-US shipped.

State Model (Zustand)

```ts
export type CodeLang = 'js'|'ts'|'py'|'json'|'bash'|'md';
export interface Draft {
  id: string;            // nanoid
  title: string;
  content: string;       // markdown
  updatedAt: number;     // epoch ms
}
export interface AppState {
  content: string;
  selection: { start: number; end: number } | null;
  codeLang: CodeLang;
  theme: 'dark'|'light'|'system';
  tz: string;            // IANA tz, e.g., 'America/New_York'
```

```
  analyticsEnabled: boolean; // persisted
  drafts: Record<string, Draft>;
}
```

Local Persistence (localStorage)

- `dmf.content:v1` → current editor content
- `dmf.settings:v1` → `{ theme, tz, analyticsEnabled }`
- `dmf.drafts:v1` → array of `Draft`
- Migrations: `dmf.mig:v1` to track schema version.

Timestamp Algorithm (absolute & relative)

```
Input: date, time, timezone (IANA), style (optional)
1. Use Luxon to construct DateTime from inputs in selected timezone.
2. Convert to seconds since Unix epoch: `Math.floor(dt.toUTC().toSeconds())`.
3. If relative: compute delta from now and produce `<t:epoch:R>`.
4. If absolute: produce `<t:epoch>` or `<t:epoch:STYLE>`.
5. Insert token into editor at cursor.
```

Markdown Action Algorithms (selection wrappers)

- **Wrap**: given token(s) `startToken`, `endToken` (same if symmetric), wrap current selection; if selection already wrapped, **unwrap**.
- **Block**: for lists/quotes/fences, split selection by lines and prefix with `-`, `>`, or add fenced block with language.

Security Model

- No HTML input accepted; markdown-it `html:false`.
- Sanitize preview output using DOMPurify with an allowlist of tags: `a, strong, em, s, u, code, pre, span, ul, ol, li, blockquote`.
- Escape masked link URLs; prohibit `javascript:` and data URIs except `data:image` when rendering (but images are informational only and not rendered inline to match Discord).
- Do **not** use React Server Components; stay client-only to avoid RSC vulnerabilities. Lock React to patched 19.2.x+.

Component Diagram (PlantUML)

```
@startuml
skinparam componentStyle rectangle
component "Editor" as Editor
component "Toolbar" as Toolbar
component "Preview (markdown-it + DOMPurify + Prism)" as Preview
component "Timestamp Builder" as TS
component "Quick Reference" as Ref
```

```
component "Settings (Theme/TZ/Analytics)" as Settings
component "Zustand Store" as Store
component "LocalStorage" as LS
component "Plausible (optional)" as Plausible

Editor --> Store
Toolbar --> Editor
TS --> Editor
Store --> Preview
Store --> Settings
Settings --> Store
Store --> LS
Preview --> DOMPurify
Preview --> Prism
Settings ..> Plausible : load script if enabled
Ref --> Editor
@enduml
```

Sequence: "Insert Timestamp"

```
@startuml
actor User
participant "Timestamp Builder" as TS
participant "Luxon" as Luxon
participant "Editor" as Editor

User -> TS: select date/time/tz & style
TS -> Luxon: compute epoch seconds (UTC)
Luxon --> TS: epoch
TS -> Editor: insert `<t:epoch[:STYLE]>` at cursor
Editor -> Editor: update state & cursor
@enduml
```

Styling & UX Notes

- **Dark-first**; system theme detection; toggle in header.
- Compact spacing, monospace for code, readable defaults (14–15px body, 13px monospace).
- Copy buttons adjacent to editor and timestamp builder.
- Mobile: vertical stack (Editor → Toolbar → Preview), sticky toolbar; desktop: split pane with resizable gutter.

Library Choices (justification)

- **markdown-it**: fast, pluggable, configurable subset matching; custom rules for Discord-only bits.
- **Prism or Highlight.js**: Prism is lightweight and extensible; HLJS offers autodetect. Start with Prism and ship only 5 languages; allow switching if autodetect requested later.

- **Luxon + @vvo/tzdb**: reliable IANA timezone handling and up-to-date tz data list.
- **DOMPurify**: hardened sanitizer with good defaults.
- **Zustand**: simple and small global state without boilerplate.
- **Vite**: fast DX and small bundles.

Implementation

Tech Stack & Versions (locked for MVP)

- **React 19.2.4**, **TypeScript 5.6+**, **Vite 7.3.1**, **Zustand 5.0.10**, **markdown-it 14.1.0**, **PrismJS 1.30.0**, **DOMPurify 3.3.1**, **Luxon 3.7.2**, **@vvo/tzdb 6.198.0**, **TailwindCSS 3.4+**, **Zod 4.3+**.

Project Structure

```
/discord-markdown-formatter
  ├─ src/
  │  ├─ app/
  │  │  ├─ App.tsx
  │  │  └─ routes.tsx (single route for SPA)
  │  ├─ components/
  │  │  ├─ Editor.tsx
  │  │  ├─ Toolbar.tsx
  │  │  ├─ Preview.tsx
  │  │  ├─ TimestampBuilder.tsx
  │  │  ├─ QuickReference.tsx
  │  │  ├─ Settings.tsx
  │  │  └─ CopyButton.tsx
  │  ├─ lib/
  │  │  ├─ markdown.ts (markdown-it + plugins + Prism hook)
  │  │  ├─ sanitize.ts (DOMPurify config)
  │  │  ├─ selection.ts (wrap/unwrap helpers)
  │  │  ├─ time.ts (Luxon helpers)
  │  │  └─ storage.ts (localStorage API + migrations)
  │  ├─ store/
  │  │  └─ useAppStore.ts
  │  ├─ styles/
  │  │  └─ prism.css (only needed languages)
  │  ├─ assets/
  │  └─ index.tsx
  ├─ public/
  │  └─ favicon.svg
  ├─ package.json
  ├─ vite.config.ts
  ├─ tailwind.config.ts
  └─ README.md
```

Dependencies

```
{
  "dependencies": {
    "react": "19.2.4",
    "react-dom": "19.2.4",
    "zustand": "5.0.10",
    "markdown-it": "14.1.0",
    "prismjs": "1.30.0",
    "dompurify": "3.3.1",
    "luxon": "3.7.2",
    "@vvo/tzdb": "6.198.0",
    "zod": "4.3.6"
  },
  "devDependencies": {
    "typescript": "^5.6.0",
    "vite": "7.3.1",
    "@types/dompurify": "^3.0.5",
    "@types/luxon": "^3.7.1",
    "tailwindcss": "^3.4.0",
    "postcss": "^8.4.0",
    "autoprefixer": "^10.4.0"
  }
}
```

Markdown Engine (src/lib/markdown.ts)

```
import MarkdownIt from 'markdown-it';
import DOMPurify from 'dompurify';
import Prism from 'prismjs';
import 'prismjs/components/prism-javascript';
import 'prismjs/components/prism-typescript';
import 'prismjs/components/prism-python';
import 'prismjs/components/prism-json';
import 'prismjs/components/prism-bash';

export const md = new MarkdownIt({
  html: false,
  linkify: true,
  typographer: false,
  breaks: false,
  highlight: (code, lang) => {
    try {
      if (lang && Prism.languages[lang]) {
        return `<pre class="language-${lang}"><code>` +
          Prism.highlight(code, Prism.languages[lang], lang) +
```

```
          '</code></pre>';
      }
    } catch {}
    return `<pre class="language-none"><code>${md.utils.escapeHtml(code)}</
code></pre>`;
  }
});

// Custom rules to emulate Discord quirks
// underline via __text__
const underlineRule = (state) => {/* simplified placeholder; implement with
inline rule */};
// spoiler via ||text|| -> wrap in <span class="spoiler">text</span>
const spoilerRule = (state) => {/* inline rule */};

// Register plugins/rules here
// md.inline.ruler.before('emphasis', 'underline', underlineRule);
// md.inline.ruler.before('emphasis', 'spoiler', spoilerRule);

export function renderSafe(markdown: string) {
  const dirty = md.render(markdown);
  // Sanitize output; tight allowlist
  return DOMPurify.sanitize(dirty, {
    USE_PROFILES: { html: true },
    ALLOWED_TAGS:
['a','strong','em','s','u','code','pre','span','ul','ol','li','blockquote','p','br'],
    ALLOWED_ATTR: ['class','href','rel','target','data-language'],
    FORBID_ATTR: ['on*'],
    FORBID_TAGS: ['img','iframe','style']
  });
}
```

Selection Helpers (src/lib/selection.ts)

```
export function toggleWrap(src: string, start: number, end: number, token:
string) {
  const before = src.slice(0, start);
  const sel = src.slice(start, end);
  const after = src.slice(end);
  const wrapped = src.slice(start - token.length, end + token.length);
  const already = before.endsWith(token) && after.startsWith(token);
  if (already) return before.slice(0, -token.length) + sel +
after.slice(token.length);
  return before + token + sel + token + after;
}
```

Timestamp Helpers (src/lib/time.ts)

```ts
import { DateTime } from 'luxon';
export type Style = 't'|'T'|'d'|'D'|'f'|'F'|'R'|undefined;
export function toEpochSeconds(dtISO: string, tz: string) {
  const dt = DateTime.fromISO(dtISO, { zone: tz });
  return Math.floor(dt.toUTC().toSeconds());
}
export function token(epoch: number, style?: Style) {
  return style ? `<t:${epoch}:${style}>` : `<t:${epoch}>`;
}
```

Components (key excerpts)

- **Editor.tsx**: `<textarea>` with controlled value from Zustand; on toolbar actions, use `toggleWrap` and update selection positions.
- **Toolbar.tsx**: buttons for **B**, *I*, **U**, ~~S~~, `code`, `code block`, quote, list, spoiler; timestamp button opens builder.
- **Preview.tsx**: `dangerouslySetInnerHTML={{__html: renderSafe(content)}}` (sanitized output only).
- **TimestampBuilder.tsx**: date/time pickers + timezone select (from `@vvo/tzdb`), style dropdown, **Insert** & **Copy**.
- **Settings.tsx**: theme toggle (dark/light/system), timezone default, **Analytics** toggle (off by default).

Styling

- Tailwind with custom tokens: `--bg`, `--fg`, `--muted`, `--accent` to approximate Discord feel without copying. Monospace for code (`ui-monospace, SFMono-Regular, Menlo, Monaco, Consolas, 'Liberation Mono', 'Courier New', monospace`).
- Spoiler CSS: `.spoiler { background: var(--muted); color: transparent; border-radius: .25rem } .spoiler:hover, .spoiler:focus { color: inherit }`.

Security & Privacy

- Sanitization via DOMPurify; no HTML parsing in markdown-it; blocked protocols (`javascript:`, `vbscript:`, `data:` except images—though images not rendered to match Discord text constraints).
- Clipboard copy uses `navigator.clipboard.writeText` with fallback to hidden `<textarea>`.
- Analytics toggle: when **ON**, dynamically import Plausible script and send only aggregated events; no cookies; no per-user IDs. Default **OFF** and persisted in localStorage.

Performance

- Bundle split: core app + async chunk for analytics + async chunk for Quick Reference examples.
- Tree-shake Prism to only ship selected languages; load others on demand.
- Target ≤**150KB gzipped** JS initial.

Testing

- Unit: selection helpers, timestamp helpers (Jest + @testing-library/react).
- E2E: core flows (playwright) — typing, wrapping, code block highlight, timestamp insertion, copy.
- Accessibility: `@axe-core/playwright` smoke checks.

Build & Deploy

- `vite build` producing static assets.
- Host on Netlify, Vercel, or GitHub Pages.
- Optional PWA: add Workbox for asset precache + offline drafts.

Example Acceptance Criteria (MVP)

- User types Markdown and sees preview within 50ms of pause.
- Inserting `<t:...>` via builder yields correct relative/absolute preview examples.
- Copy buttons copy exact source; no invisible characters.
- Code blocks for js/ts/py/json/bash highlight correctly.

Milestones

M0 – Project Setup (1 week) - Repo scaffold, linting/formatting, CI, Tailwind, Zustand store, theme switch.

M1 – Editor & Preview (1.5 weeks) - Markdown engine + sanitizer + split pane + copy buttons; initial toolbar actions.

M2 – Code Blocks & Prism (1 week) - Prism wired with js/ts/py/json/bash; language picker; lazy-load additional languages.

M3 – Timestamp Builder (1 week) - Absolute/relative tokens, timezone select, presets; Quick Reference entries.

M4 – Templates & Shortcuts (0.5 week) - Insertable message templates; keyboard shortcuts; drafts persistence.

M5 – Hardening & A11y (0.5 week) - a11y pass, sanitizer allowlist review, unit/E2E tests, performance budget.

M6 – Optional Analytics Toggle (0.5 week) - Settings switch, deferred script load, event wiring.

M7 – Launch (0.5 week) - Docs, version lock, deploy to Netlify/Vercel, post-launch QA.

Gathering Results

Success Metrics

- **Usability**: time-to-first-correctly-formatted message < 2 minutes for new users.

- **Accuracy**: 0 known discrepancies between preview and Discord rendering for supported syntax.
- **Performance**: TTI ≤ 2s on Moto G Power (emulated), interaction latency < 100ms.
- **Adoption**: ≥ 500 unique users in first month (if analytics enabled by users).

QA Playbook

- Markdown parity tests using a corpus of examples from Discord's docs; visual diffs per release.
- Timestamp correctness tests across 10 timezones, DST boundaries, and relative clock skew ±2 minutes.
- Security scans: dependency audit, DOMPurify config review, link protocol tests.

Post-Launch

- Collect user feedback via GitHub Issues; triage weekly.
- Add languages as requested (Prism components lazy-loaded per need).
- Evaluate PWA/offline and PNG export based on user demand.

Need Professional Help in Developing Your Architecture?

Please contact me at sammuti.com :)