PRD: AI Action-Items Converter

PRD Title: AI Text-to-Action Item Converter

Owner: Team CodeCrafters (A. Venkata Hari Priya , U. Ujwala , A. Rmaya)

Date: December 25, 2025

Version: 1.0 (MVP)

Stakeholders: Buildathon Judges, Mentors, End Users

1) Overview

- **What are we building?:** A Client-Side Progressive Web App (PWA) that uses AI logic to instantly convert unstructured meeting notes or voice transcripts into structured Action Items with deadlines, owners, and smart priorities.
- **Why now?:** Remote work has increased Zoom fatigue. Employees waste hours re-watching calls just to find simple tasks. We need a tool that parses tasks "before the coffee gets cold," without privacy risks or complex setups.

2) Problem Statement

- **Problem:** After a 45-minute video call, action items get buried in verbal conversations. Reviewing the entire recording to find who promised what is time-consuming and error-prone.
- **Who is affected (persona):** Project Managers, Developers, Students, and Corporate Teams who attend multiple meetings daily.
- **Impact / pain:** Missed deadlines, ambiguity over task ownership ("I thought *you* were doing that"), and wasted productivity reviewing transcripts.

3) Goals & Success Metrics

- **Primary goal(s):** Reduce the time required to extract actionable tasks from raw text from ~15 minutes to under 5 seconds.
- **KPIs (with target numbers):**
    1. **Parsing Accuracy:** >90% success rate in identifying Assignee and Due Date.
    2. **Processing Speed:** Generate report in <200ms for inputs up to 100 lines.
- **Guardrails (must not worsen):** Privacy must be absolute. No user data can be sent to external servers for processing.

4) Users / Personas

- **Primary persona:** "The Busy PM" – Juggles 5+ meetings a day, needs quick summaries immediately after a call.
- **Secondary persona(s):** "The Student" – Needs to track group project assignments and deadlines from chaotic group chats.
- **Key needs / jobs-to-be-done:**
    1. Input messy text quickly (Paste or Voice).
    2. Visualize deadlines on a timeline.
    3. Set instant timers for urgent tasks.

5) Scope

- **In scope:**
    - Text-to-Action parsing logic (Regex-based).
    - Voice-to-Text input (Web Speech API).
    - Priority auto-detection (High/Medium/Low).
    - Visual Timeline generation.
    - **Local Launcher:** Python script to ease testing.
    - **PWA functionality:** manifest.json included for installability.
- **Out of scope:**
    - Real-time audio recording of full hour-long meetings.
    - Backend database integration (MongoDB/SQL).
    - Automatic Email Sending (User manually copies text to email).

6) User Journey / Use Cases

- **Top use cases (3-5):**
    1. **Post-Meeting Cleanup:** User pastes a raw transcript and instantly sees a visual timeline of tasks.
    2. **Voice Note Taking:** User dictates tasks while walking ("Remind me to call John tomorrow").
    3. **Urgent Deadline Tracking:** User sets a 30-minute browser alert for a "High Priority" task.
- **Happy path (steps):**
    1. User runs python app.py and opens the app.

2. User logs in (credentials saved locally).
3. Pastes text into the input box.
4. Clicks "Generate Action Items."
5. Reviews structured cards (Red/Yellow/Green priorities).
6. Clicks "Copy Email" and pastes the summary into their email client.

- **Edge cases:**
    - Input text contains no names or dates (System defaults to "Unassigned" and "No Date").
    - User opens file directly without Python (Voice features may be disabled by browser security).

7) Requirements (Functional)

R1: Text Parsing Engine

- **Description:** The system must analyze text strings to extract "Assignee" (Who), "Task" (What), and "Deadline" (When).
- **Priority:** P0 (Critical)
- **Acceptance criteria:** Input "John to fix bug by Friday" must output -> {Assignee: John, Task: fix bug, Due: Friday, Priority: Medium}.

R2: Smart Priority Logic

- **Description:** Algorithms must assign priority levels based on keywords.
- **Priority:** P1
- **Acceptance criteria:**
    - "Today/Urgent" = High (Red).
    - "Tomorrow/Next Week" = Medium (Yellow).
    - No date = Low (Green).

R3: Localhost Launcher

- **Description:** A simple Python script must serve the static files to simulate a server environment.
- **Priority:** P1
- **Acceptance criteria:** Running python app.py must launch the default browser to the correct localhost URL.

8) Non-Functional Requirements

- **Performance:** Parsing must occur in real-time (<200ms) on client devices.
- **Security & Privacy:** Zero-knowledge architecture. All data remains in LocalStorage; nothing is transmitted to the cloud.
- **Reliability:** Core parsing features must work offline.
- **Compatibility:** Optimized for Google Chrome (Desktop & Android).

9) Data & Analytics

- **Events to track:** N/A (Privacy-focused app does not track user analytics).
- **Dashboards/Reports:** User Dashboard shows a personal list of generated tasks stored in LocalStorage.

10) Dependencies & Assumptions

- **Dependencies:**
  - Browser Web Speech API (for Voice features).
  - Python 3 (for the local launcher script).
- **Assumptions:** Users will provide inputs in English using a standard Subject-Verb-Object structure.

---

11) Risks & Mitigations

- **Risk:** The parsing logic (Regex) might fail on complex sentences.
- **Mitigation:** Provide a clear "Example Format" placeholder text to guide users toward the supported input structure.

12) Rollout Plan

- **Phases:**
  - **Alpha:** Internal testing on Localhost.
  - **Beta:** Submission to Buildathon judges.
- **Rollback plan:** Revert to previous HTML/JS commit.

13) Open Questions

- **Q1:** Should we integrate a third-party LLM (like Gemini API) in V2 for better context understanding?
- **Q2:** Can we add a feature to export tasks directly to a .CSV file?

Tech Stack & Deployment (Instructions for Judges)

- **Frontend:** HTML5, CSS3, JavaScript (Vanilla ES6).
- **Architecture:** Client-Side Progressive Web App (PWA).
- **AI/NLP Logic:** JavaScript Regular Expressions (RegEx) for pattern matching and entity extraction.
- **Voice Integration:** Web Speech API (Native Browser Support).
- **Storage:** LocalStorage (Browser-based persistence).
- **Local Server:** Python (app.py) included to serve files and enable secure browser features (Mic/Audio).

How to Run This Project:

1. Download the project folder.
2. Run the Python launcher: python app.py
3. The app will automatically open in your default browser at http://localhost:8000.
4. **Note:** Using this launcher is recommended over clicking the HTML file directly, as it ensures the **Voice Recognition** features have the required security permissions.