ChatGPT

Product Requirements Document: Slack Conversation to Decision Card Plugin (MVP)

Overview

This document outlines the requirements for a WordPress plugin that converts Slack-style conversations into summarized "Decision Cards" on a WordPress site. The plugin will allow users (e.g., team leaders or project managers) to paste a Slack conversation transcript into an admin interface, then use AI (via OpenAI-compatible APIs) to generate a summary of the discussion and extract key action items. The result will be saved as a structured WordPress post (custom post type) capturing the decision's status (Proposed, Approved, or Rejected), the owner responsible, and a due date for any follow-up. This helps teams document important decisions made in Slack, ensuring none of the context or commitments are lost once the Slack thread fades [1]. By using a custom post type for Decision Cards, we keep this content separate and structured, improving organization and clarity on the site [2]. Each Decision Card will have its own custom fields and layout, making it easy to review decisions without cluttering regular blog posts [3].

Goals

- **Capture Decision Discussions:** Provide a simple way to record decisions and outcomes from Slack (or Slack-like) conversations into WordPress, preserving key context and agreed action items.
- **Automate Summarization:** Leverage AI (OpenAI or compatible models) to **summarize lengthy conversations** and highlight main points and tasks, saving users time compared to writing summaries manually.
- **Structured Documentation:** Store decisions as structured content (custom post type with fields for status, owner, due date) so that decision records are consistent and easy to sort or filter (e.g., find all "Approved" decisions or all decisions owned by a certain person).
- **Ease of Use:** Ensure the plugin is **standalone, easy to install, and use**. Users should be able to activate the plugin, enter an API key, and start generating Decision Cards without complex setup or coding. The interface should be minimal and intuitive (paste text and click a button).
- **Reliability and Clarity:** The generated summaries should be clear and concise. Include basic error handling so if the AI fails or the input is invalid, the user gets a helpful message rather than a broken experience. Logging should be in place to help troubleshoot issues during this MVP stage.
- **No Slack API Dependency (MVP):** Focus on **simulated input** for now – the user will manually paste conversation text. This avoids needing a Slack API integration or OAuth in the MVP, keeping the scope small while still addressing the core problem of documenting decisions.

User Stories

- **As a Team Member (User),** I want to quickly create a Decision Card from a Slack conversation so that I can document important decisions without manually writing a summary.
- **As a Project Manager (User),** I want the plugin to extract the key points and action items from the discussion so that the Decision Card highlights what was decided and who is responsible for what.

- **As a Content Editor (User),** I want to be able to set or adjust the decision’s status (Proposed/Approved/Rejected), assign an owner, and set a due date, so that the context of the decision (outcome and next steps) is clear to stakeholders.
- **As a WordPress Admin,** I want to easily configure the plugin (for example, enter my OpenAI API key once) and use it via a simple interface, so that even non-technical team members can generate decision summaries with minimal training.
- **As a WordPress Admin,** I want the plugin to handle errors gracefully (e.g., API failures or invalid input) by showing error messages or logs, so that I can understand issues and ensure the Decision Card gets created correctly.

Features & Functionality

- **Custom Post Type – “Decision Card”:** The plugin registers a custom post type (e.g., `decision_card`) to store decision records. This keeps Decision Cards separate from regular posts and pages [2]. Each Decision Card post will include custom fields for **Status** (Proposed, Approved, Rejected), **Owner** (person responsible for the decision or action), and **Due Date** (deadline for any action items). These fields can be stored as post meta (custom fields) and shown prominently in the post.
- **Admin Interface for Input:** In the WordPress admin (likely under a new menu item “Decision Cards” or a Tools submenu), provide an interface with a **large textarea** input where the user can paste the Slack conversation transcript. This conversation text is free-form (e.g., a series of messages prefixed by usernames/timestamps). The interface will be kept minimal – just instructions and the input field – to keep the user experience simple.
- **“Generate Summary” Action:** A **Generate Summary** button triggers the AI summarization. When clicked, the plugin will take the conversation text from the textarea and call the OpenAI (or OpenRouter) API to generate two things: (1) a **concise summary** of the conversation focusing on the decision made or discussed, and (2) a **list of action items** (bullet points of tasks or follow-ups mentioned in the convo). The API call will happen server-side (to avoid exposing API keys and to handle large text). A loading indicator or message will inform the user that the summary is being generated.
- **AI-Generated Summary & Action Items:** The OpenAI API’s response will be parsed to extract the summary and action items. The plugin will then create a **new “Decision Card” post** in WordPress with that content. The post content template may include a section for the summary and a section for the action items (for example, the content might read: **Summary:** ... AI-generated summary ... **Action Items:** … bullet list ...). Alternatively, the summary could be saved as the post content and the action items as a separate structured field or appended after the summary in content. For MVP, simply inserting both into the post content (with proper formatting) is acceptable.
- **Metadata Fields Population:** The plugin will also populate the custom fields for Status, Owner, and Due Date. For the MVP, these could be left for the user to fill in manually after generation (with default values like Status = “Proposed” if not provided). Optionally, the generation form can include inputs for these fields so the user sets them before clicking “Generate Summary”. For example, a dropdown for Status, a text field for Owner, and a date picker for Due Date. If provided, the plugin uses these values when creating the post.
- **Post Creation and Feedback:** Once the Decision Card is created (with summary and action items), the user should get feedback. For instance, after generation the plugin might redirect the user to the newly created post’s edit screen (as a draft) so they can review and publish it. This allows the user to make any edits (e.g., adjust wording or fill in any missing metadata) before publishing. If the design chooses to show the results first, it could display the summary and actions on the generation page for review with a “Save as New Post” confirmation button. However, to simplify, automatically saving as a draft post is a straightforward MVP approach.

- **OpenAI API Integration:** The plugin will integrate with OpenAI-compatible APIs to perform text generation. It should accept an API key (from OpenAI or OpenRouter) via a settings page or a config file. All API calls (for generating summaries) are made from the server side using WordPress HTTP functions (e.g., `wp_remote_post`). The conversation text is sent to OpenAI’s **Chat Completion** endpoint [4] along with a prompt instructing it to summarize and list action items. The plugin should parse the API response (likely JSON) to retrieve the generated text.
- **Basic Error Handling:** The plugin will include error handling for common failure cases:
- If the OpenAI API request fails (network error, API error, or invalid API key), the user should see a friendly error message in the UI (e.g., “Summary generation failed: please check your API key and internet connection”). The error should not result in a half-created post; no Decision Card post will be saved in this case to avoid clutter.
- If the conversation text is empty or too short, the plugin can warn the user that there isn’t enough content to summarize (preventing a wasted API call).
- If the AI returns an incomplete or malformed response, the plugin should handle that gracefully (perhaps show an error, or include whatever was returned and flag it).
- **Logging:** Implement a simple logging mechanism to record events and errors. For instance, log the API request and response status (but **not** the full conversation text or summary content, to avoid storing sensitive info unnecessarily). Logging could be done to a file in the plugin directory or via WordPress’s debug log (`error_log()` function) for developers to troubleshoot. In case of AI errors, the log might include the error code or message from OpenAI. This is especially useful during the MVP phase to identify issues.
- **Minimal Design / Styling:** The admin UI will use WordPress default styles as much as possible (to keep it lightweight). We won’t build a complex UI – just a simple form. No frontend interface is required since Decision Cards will be viewed as normal posts on the site (or within the admin). The plugin should be **standalone** (no dependency on other plugins) and keep the footprint small.
- **Security Considerations:** Only users with appropriate permissions (e.g., Editors or Administrators) should be able to generate Decision Cards, since it involves creating posts. The input is a pasted conversation – the plugin should sanitize this input appropriately when inserting into the post (e.g., strip disallowed HTML) to avoid any potential script injection, though in most cases it will be plain text. API keys should be stored securely (in the database via an options page) and never exposed on the client side.

UI Wireframe Description

The plugin’s admin UI will be simple and focused. When a user navigates to the **“Generate Decision Card”** page (accessible via the WordPress admin menu, e.g. under **Decision Cards** or **Tools**):

- **Header/Title:** The page title might be “New Decision Card” or “Slack Conversation Summary”. A brief description could be included under it: e.g., “Paste a Slack conversation transcript below and click ‘Generate Summary’ to create a Decision Card.” This guides the user on how to use the tool.
- **Conversation Textarea:** A large **multiline textarea** input labeled “Conversation Transcript” is the primary field. The user will paste the entire Slack conversation here. This field should be big enough (maybe 10-15 lines visible by default, scrollable) to accommodate long conversations. For example, the text might look like multiple lines prefixed with usernames and messages, which the user copies from Slack. (No special formatting or rich text needed; plain text is fine.)
- **Metadata Inputs (optional):** Below the conversation box, there may be fields for the structured metadata:
- **Status:** A dropdown or radio buttons with options Proposed, Approved, Rejected. The user can select the decision status. By default, it could start as “Proposed” (or a blank that forces choice). This indicates the outcome of the decision being discussed.

- **Owner:** A small text input (or dropdown of users) where the user can specify who is the owner or responsible person for this decision or task. In MVP, a simple text field is fine (the user might type a name).
- **Due Date:** A date picker or text field for a deadline or due date related to the decision’s action items. (In MVP, even a text field accepting a date string is sufficient, but a date picker UI would improve usability). These fields are optional in the MVP interface – the user could also fill them in later by editing the created post – but including them here streamlines the workflow.
- **Generate Summary Button:** A prominent **button** labeled “Generate Summary” is placed below the inputs. When clicked, it triggers the summarization process. The button might be styled in WordPress admin style (e.g., primary button class). If using a standard form submission, clicking it will submit the form to the plugin for processing. If using AJAX, clicking it will trigger a JavaScript call. In either case, the user should get feedback that something is happening.
- **Loading Indicator:** Upon clicking the button, the UI should give a visual cue that the summary is being generated. For example, disable the button and show a spinner or a message like “Generating summary, please wait…”. This prevents the user from clicking multiple times and sets expectation that it may take a few seconds (depending on API response time).
- **Post-Creation Behavior:** After the OpenAI API returns the summary and the plugin creates the Decision Card post, the UI will either:
- **Redirect to the Post Editor:** e.g., open the newly created Decision Card in edit mode. The user would then see the post title (which might default to something like “Decision Card – {Date}” or a snippet of the summary), the content filled in with the summary and action items, and the custom fields (Status, Owner, Due Date) populated. The user can make final edits and hit Publish. This approach leverages WordPress’s native post editing UI for review.

- **Show a Preview on the same page:** e.g., display the generated summary and action items right on the form page (perhaps in a preview box below the button) along with a message “Success! Decision Card created as Draft.” and a link to view/edit it. The user could click the link to go to the editor, or if the output is satisfactory, maybe a “Publish Now” button could be offered.
  For MVP simplicity, the redirect to the post editor (draft) is straightforward and uses WordPress defaults for editing content.

- **Error Messages:** If something went wrong (for example, the API call failed, or the user forgot to paste text), the page will display an error notice in red. For instance, “Error: Unable to generate summary. Please check your API key and try again.” This notice appears near the top of the page or near the button. The user can correct the issue (e.g., update their API settings or ensure they pasted text) and try again.

Visual Layout: All these elements would be arranged in a single column, similar to a basic form: instructions, textarea, optional fields, then the button. The design will use standard WP admin styles (e.g., form fields with regular styling, the primary button style for submission). This minimal design ensures the focus is on functionality and clarity. No elaborate styling or JavaScript frameworks will be used in the MVP to keep it lightweight.

Technical Requirements

- **WordPress Compatibility:** The plugin will be developed for WordPress 6.x and above (ensuring compatibility with the latest WordPress at time of development). It should use WordPress best practices (PHP 7 or 8 compatible code, no deprecated functions) and adhere to coding standards to ensure maintainability. The plugin is installed like any other (via upload or the plugin repository in the future) and should activate without errors.

- **Programming Language:** Primarily PHP (since it’ s a WordPress plugin). We will use minimal JavaScript for any dynamic behaviors like AJAX calls or UI niceties (if needed for loading indicators, etc.), but the core logic (API call, post creation) is in PHP on the server side.
- **Custom Post Type Implementation:** Use the WordPress `register_post_type()` function to create the `decision_card` post type at plugin initialization. This post type should support custom fields (for metadata) and have its own menu in the admin (or be under an existing menu). The post type can use the standard WordPress editor for content, but we’ ll be programmatically creating content for it. We might set `'public' => false, 'show_ui' => true` for this post type, meaning it’ s managed in admin but not necessarily meant to be publicly queryable like blog posts (depending on whether these Decision Cards should appear on the public site or just internal).
- **Custom Fields:** Implement the Status, Owner, Due Date as custom fields (post meta). We can register these meta keys (using `register_post_meta` for proper sanitization, or simply use them directly). In the post edit screen, these could appear in a meta box for Decision Cards. For MVP, a simple meta box or the default “custom fields” UI can be used to edit them. (If time permits, we can add a small meta box with dropdown for status and date picker for due date in the post editor).
- **Admin Page & Form Handling:** The plugin will add an admin page (likely via `add_menu_page` or `add_submenu_page`) for the “Generate Summary” interface. This page will use a standard HTML form. On submission, a PHP handler (within the plugin, hooked via admin_post or a dedicated page handler) will:
- Validate user capabilities (ensure the user can create posts).
- Fetch the conversation text and any metadata from the POST request.
- Call the OpenAI API (using curl or `wp_remote_post` to the appropriate endpoint).
- Receive and parse the response.
- Create a new `decision_card` post (using `wp_insert_post`), with post_status as ‘draft’ (or ‘publish’ if we want it live immediately – but draft is safer for review). Fill the post content with the formatted summary and action items. Save the meta fields (status, owner, due date) using `update_post_meta`.
- Redirect to the post editor or back to the form with a success message.
  We must ensure this process is secure: use nonces in the form to prevent CSRF, sanitize input from the form (though it’ s mostly free text which we’ ll pass to API and then to post – we should strip any `<script>` tags just in case).
- **OpenAI API Integration:** We will integrate with OpenAI via their REST API. Specifically, we plan to use the **Chat Completions API** (endpoint `/v1/chat/completions`) since the Slack conversation is a dialog and the AI is well-suited to process conversation context [5] . We’ ll need to include the Authorization header with the API key for each request. The plugin should allow configuration of the API key: likely via a settings page where an admin enters their OpenAI API key (or an OpenRouter API key/endpoint). This key would be stored in the WordPress database (option table) and retrieved for use in API calls. Ensure this key is not exposed publicly (it will only be used server-side).
- The request format: The plugin will construct a prompt or messages array for the chat API. For example, it might send a system message like “You are an assistant that summarizes Slack conversations into a concise decision summary and action items.” and a user message containing the pasted conversation text. It will then request a completion.
- Model: By default, use `gpt-3.5-turbo` for speed and cost-efficiency. The plugin might allow the model to be configured (somewhere in settings) if the user wants to use `gpt-4` or any OpenAI-compatible model via OpenRouter. But MVP can hard-code a model like GPT-3.5.
- Response Handling: The OpenAI API will return a JSON with the assistant’ s reply. The plugin will extract the text. Ideally, we’ ll design the prompt such that the AI’ s answer is already structured

(e.g., contains a summary paragraph and then "Action Items:" list). The plugin may do minor post-processing, like adding Markdown bullets to the action items if needed, or splitting the text by a delimiter if we ask the AI to output JSON or a special format (the latter is more complex; MVP can rely on plain text and trust the format to an extent).

- **API Usage Constraints:** Be mindful of token limits and rate limits:
- The plugin should enforce a reasonable size limit on the input conversation text (for example, if a Slack thread is extremely long). OpenAI's GPT-3.5 model has a context window of around 4,000 tokens (which is roughly ~3,000 words of input before it can't handle more) [6]. If the user inputs a conversation that's too large (e.g., many thousands of words), we may need to warn or truncate it to avoid API errors. For MVP, simply warning the user or truncating the input (and noting that in the summary) could be done.
- Rate limiting: The OpenAI API key may have a rate limit (requests per minute). In normal use, it's unlikely a single user generates summaries in rapid succession, but the plugin should not spawn simultaneous API calls from one click. We'll ensure one request per click. (If we later allow bulk operations, we'd need to queue them, but not in MVP.)
- Cost considerations: Each API call costs tokens (input + output). The plugin should aim to keep the prompt succinct and request a reasonably sized summary (not too long) to minimize cost. Perhaps instruct the model to keep the summary to a few paragraphs and action items list concise. We assume the user understands they pay OpenAI for usage; we will note this in documentation ("Requires an OpenAI API key with an active subscription or credits").
- **Dependencies:** No heavy external dependencies beyond the OpenAI API. We will use WordPress built-in functions (for HTTP, for UI, etc.) as much as possible. If using OpenRouter, the endpoint and API key would be configurable, but no additional library is needed if their API is OpenAI-compatible (the same HTTP call structure). We won't use client-side libraries or frameworks (no jQuery beyond WP default, no React etc., unless needed for admin UI which is unlikely for this simple form).
- **Testing Requirements:** The plugin should be tested with various conversation inputs to ensure the output is formatted correctly in the post. Also test the error flows (e.g., put an invalid API key and see that an error is shown, test with no input, test with extremely long text). WordPress's environment (PHP version, memory limits) should handle the typical API call (the largest cost will be memory for handling the JSON response and the text). We should document any special requirements like needing PHP curl enabled (WordPress usually has that).
- **Performance:** Given this is a synchronous process (user clicks and waits for summary), we should be mindful of not doing anything overly slow aside from the API call itself. The API call might take a couple of seconds depending on the conversation length. If it becomes an issue, using AJAX to make the call could prevent the page from timing out. For MVP, a normal form post that might take a few seconds is acceptable, but we'll set a reasonable timeout on the HTTP request (maybe ~10-15 seconds) and handle a timeout gracefully (show an error suggesting the user try again). Logging the time taken can help gauge performance.

OpenAI API Usage Pattern and Constraints

- **API Choice and Model:** The plugin will utilize the **OpenAI Chat Completions API** (or an equivalent OpenAI-compatible endpoint via OpenRouter) to generate the summary and action items. The Chat Completions API is suitable because it can handle conversational context and instructions in one call [5]. We will likely use `gpt-3.5-turbo` as the default model due to its balance of capability and cost, but ensure the integration is modular enough to switch to another model or provider if needed (OpenRouter could route to alternatives).
- **Prompt Design:** The prompt sent to the API will instruct the model to produce the desired output format. For example, the system message might say: "You are a tool that summarizes Slack conversations into a decision record. Include a brief summary of the discussion (focusing on the

decision and rationale) and then list any action items with bullet points. Use a friendly but professional tone. If a decision status or outcome is clear, note it." The user message will then contain the raw conversation text that was pasted. By structuring the prompt, we aim to get a clear **Summary** and **Action Items** list in the response. We might ask the model to prefix the action list with a heading like "Action Items:" for easier parsing.

- **Handling Long Inputs:** As noted, OpenAI models have a context limit. GPT-3.5 can handle roughly 4096 tokens total (input + output), which is about ~8 pages of text. If a conversation is longer, the model won't see beyond the limit. For MVP, we assume users will input reasonably sized conversations (maybe Slack threads of a few dozen messages). If needed, we will trim the input from the start or advise the user to shorten it. (A future enhancement could chunk the conversation and summarize iteratively, but that's out of scope now.) We will document this limitation.
- **Response Parsing:** We expect the API to return a single message (the assistant's reply) containing the summary and action items. The plugin will parse this text. Since it's natural language, the safest approach is to insert it directly into the post content. If we strictly require splitting summary vs. action items, we could look for a delimiter or known phrase. For instance, if we instructed the model to output "**Summary:** ... **Action Items:** ...", the plugin can split on "Action Items:" to separate the two. However, this parsing might be brittle if the model's output varies. MVP approach: treat the entire output as one block of content (which inherently has a summary paragraph and a list). This is simpler and avoids parsing errors.
- **API Errors & Retries:** The plugin should handle certain API issues. If the API returns an error (HTTP non-200 response or an error field in JSON), the plugin will catch that. Common issues might be invalid API key (401), rate limit exceeded (429), or content filter (if the conversation had disallowed content). In such cases, we do **not** retry endlessly. We show an error to the user as described. If it's a rate limit or temporary issue, the user can manually retry by pressing the button again. If it's an invalid key or content issue, the user must fix the input or key. We won't implement automatic retries in MVP to keep logic simple.
- **Costs and Usage Monitoring:** Each summary generation uses the user's OpenAI API credits. The plugin will not enforce limits itself (that's between user and OpenAI), but we will make sure not to accidentally call the API multiple times per request. In case of UI issues (like double-click on the button), we should prevent duplicate submissions (disable button on first click) to avoid multiple charges. We might include in documentation or UI a note like "Each summary generation will use your OpenAI API and may incur charges."
- **OpenRouter Compatibility:** If using OpenRouter (an API that can forward to multiple models with an OpenAI-compatible interface), the plugin should allow configuring the base API endpoint and API key for that. Essentially, this means not hardcoding the API URL to `api.openai.com` but letting it be configurable. Since OpenRouter's API is very similar to OpenAI's [7], minimal adjustments are needed (mostly the endpoint URL and maybe an additional header). This flexibility can be mentioned in documentation (e.g., "To use an alternative AI API (such as OpenRouter), enter its endpoint URL in the settings"). For MVP, primary testing will be with OpenAI's own API.
- **Security & Privacy:** When using the API, we will send the conversation text to OpenAI's servers. We should make the user aware that this data will be processed by a third-party (OpenAI or others) and they should avoid inputting extremely sensitive information. This can be mentioned in a disclaimer. (Given Slack conversations may be internal, this is important to note in documentation: using the plugin implies agreement to send that content to the API provider). We will rely on OpenAI's encryption (HTTPS) for transit security. The plugin itself will not store the conversation text anywhere permanently except in logs if we choose to log it (but we decided not to log full content for privacy). Only the resulting summary and items are stored in WordPress.

Assumptions & Limitations

- **Manual Input (No Live Slack Connection):** We assume users can obtain the conversation text (e.g., by copy-pasting from Slack). The plugin does not fetch data from Slack’s API in this MVP. Thus, the conversation formatting might be plain text and could include user names, timestamps, etc., as copied. The AI should handle informal formatting reasonably, but any non-text elements (images, attachments) or complex threading structure from Slack won’t be captured. The summarization is only as good as the text provided.
- **Quality of AI Summary:** The accuracy and usefulness of the summary and action items depend on the AI model and the input. The AI might occasionally miss nuances or context, especially if the conversation is very long or complex [6]. It might also **hallucinate** details that weren’t in the conversation (though less likely in a summarization task). We assume the user will review the output; this is not an authoritative record until a human verifies it. In other words, the Decision Card is a draft generated by AI. The user should edit any inaccuracies. Past experiences have shown that AI summaries of Slack chats can omit important details or misrepresent the tone, so caution is advised [8].
- **Trust and Verification:** Related to quality, we assume users will not blindly trust the AI output. Especially for critical decisions, the summary should be cross-checked with the actual conversation. The plugin does not provide a way to automatically verify the summary against the source; that’s up to the user. This is a limitation of current AI – as one prototype developer noted, you may still need to read the Slack messages if you doubt the summary [8]. Our plugin aims to save time, but not to replace human judgment.
- **Single Conversation Scope:** Each generation is assumed to be one self-contained Slack thread or conversation. The plugin does not merge multiple separate conversations, and it doesn’t handle continuous monitoring. It’s a one-time snapshot. If a decision was discussed in bits across multiple Slack channels, the user would have to compile that manually before input.
- **Structured Fields Population:** We assume that determining Status, Owner, Due Date might sometimes require human judgment. The AI might detect these in text (for example, it might infer “Approved” if the conversation clearly says a proposal was agreed, or pick up a date mentioned). However, we are not fully relying on AI to set these fields correctly in MVP. It’s the user’s responsibility to choose the appropriate status and fill any missing metadata. The plugin doesn’t validate the Owner or Due Date beyond basic format (e.g., due date should be a date).
- **No Concurrent Summaries:** We assume usage will be one summary at a time. The plugin doesn’t design for queueing multiple requests or high traffic. If two admins coincidentally used it at the same time, each would just make separate API calls. This is fine given the use case (likely low frequency, internal usage).
- **Error Handling Limits:** While we implement basic error handling, the plugin might not catch every edge case. For instance, if OpenAI changes its API format or if network is very slow, the user might experience a timeout without a graceful message (though we set timeouts). Also, logging is basic; in some hosting environments, file write permissions might block logging. We assume the WordPress site has typical capabilities (curl access, ability to connect to external APIs). If the site is on a server without external internet, the plugin cannot function (an obvious limitation given it needs API access).
- **User Interface Simplicity:** The MVP UI is very simple, which is intentional. This means certain conveniences are not present: e.g., there’s no rich text editor for the conversation (just plain textarea), no preview of how the summary will look (aside from after generation), and no multi-language UI (assuming users are okay with the interface in English). We assume an English Slack conversation and summary by default, though the OpenAI API can handle other languages if input is different. Non-English usage hasn’t been explicitly tested in MVP.
- **Security Assumptions:** We assume that only trusted users (admins/editors) have access to this feature, so we do not implement heavy input sanitization beyond protecting against technical

issues. The biggest risk is if a user pastes malicious content (like a script tag) and the plugin inserts it into a post unsanitized. To mitigate, we will sanitize or at least use `wp_insert_post` which applies some default filtering. But because these Decision Cards are likely internal records, this risk is low. Nonetheless, it’ s a limitation that the plugin doesn’ t thoroughly sanitize AI output (if the AI were ever manipulated to output unsafe HTML, theoretically). Using `wp_kses_post` on the output could be a quick solution (restrict to basic HTML). We assume the OpenAI output will mostly be plain text with perhaps markdown-like formatting which in HTML we can allow.
- **Logging Privacy:** If logging is enabled, we will avoid logging conversation text or AI output by default. We might log meta-info (timestamp, user who triggered, success/fail). This is to avoid storing potentially sensitive conversation details in logs. We assume admins can enable more verbose logging if debugging, but that’ s manual. This limitation is a conscious choice to respect privacy.

Future Considerations

Looking beyond the MVP, there are several enhancements and features we could consider:

- **Direct Slack Integration:** In the future, the plugin could connect to Slack via its API. For example, a user could authenticate a Slack workspace and then select a channel or thread from within the WordPress admin. The plugin could fetch the messages automatically, eliminating the copy-paste step. This might involve using Slack’ s Web API or Events API to retrieve conversation history. It could even be triggered via Slack (e.g., a Slack slash command or button that sends a thread to WordPress). This would make capturing decisions even smoother once the initial MVP proves useful.
- **Real-time Slack App:** Building on integration, a Slack bot or app could be developed in tandem. For instance, reacting with a specific emoji (like :bookmark:) in Slack could signal our system to grab that thread, summarize it, and post to WordPress (similar to the Zapier workflow [1], but wholly within our ecosystem). This would require a server endpoint to receive Slack events and invoke our plugin’ s logic.
- **Enhanced AI Capabilities:** We could refine the AI prompts or use more advanced models to improve summary quality. For example, using GPT-4 for more complex discussions might yield better results (though at higher cost). We might also allow custom prompts – giving users the ability to tweak what instructions the AI follows. Perhaps different teams might want a different summary style (e.g., very brief vs. detailed, or including direct quotes for record). Future versions could have settings for tone or length of summary.
- **Fine-tuning or Custom Models:** For organizations concerned about data privacy or wanting consistent outputs, fine-tuning a model or using an on-premise model could be explored. OpenAI allows fine-tuning on GPT-3. This could be used to train on some example Slack conversations and summaries to guide style. Alternatively, open-source LLMs could be integrated via OpenRouter or other means, so that data never leaves the company servers. This is a longer-term possibility once we identify the needs (and if OpenAI API costs are a barrier, an open model might be more cost-effective).
- **Action Items Integration:** The list of action items could be more than just text. In future, we might integrate with task management tools (similar to how the Zapier example created Asana tasks). For instance, an integration with plugins like [Project management in WP] or external tools (Trello, Asana, Jira) could allow one-click conversion of action items into tasks. At minimum, we could turn each action item into a checklist within the WordPress post (using a block or custom field) so that as tasks are completed, someone could check them off. This would make Decision Cards actionable, not just static records.
- **Frontend and Notifications:** Currently, Decision Cards are stored in WordPress which could be an internal site or public. In future, if these are meant for internal use, we might create a special

archive page or dashboard listing all Decision Cards, with filters by status, owner, etc., making it a "Decision Log" for the team. Additionally, we could implement notifications – e.g., when a Decision Card is published, automatically notify the Owner or interested parties (perhaps via email or even posting back to Slack a link to the Decision Card). This closes the loop between Slack discussion and formal documentation.
- **Improved UI/UX:** The MVP admin UI is minimal. We can enhance it with a more polished design: use WordPress's modern components (React-based UI via Gutenberg components) for a smoother experience, add modals or progress bars for the generation process, etc. A preview of the summary before saving could be useful so users can regenerate if they don't like it. Also, in the post editor, a custom Gutenberg block template for Decision Cards could structure the content (one block for summary, one for action items list, and fields for metadata) – making editing easier.
- **Multi-Conversational Context:** In some decisions, context might span multiple Slack threads or follow-ups that happen later. Future versions might allow selecting multiple inputs or updating an existing Decision Card with new information. For example, if a proposed decision was later approved in a separate conversation, the plugin could append that update to the original Decision Card (rather than creating a separate card). This would require identifying or linking related conversations – an advanced feature perhaps using conversation IDs if integrated with Slack.
- **Analytics & Metrics:** Over time, it might be useful to see analytics like how many Decision Cards are created, average time saved, or which statuses are most common (are most proposals approved or rejected?). While not core to functionality, providing some reporting could demonstrate the plugin's value to management (e.g., "We documented 15 decisions this month. Owner Alice has 5 upcoming action items due."). This could tie into the custom post type by querying the data.
- **Localization and Internationalization:** As adoption grows, we'd internationalize the plugin (make it translatable) and possibly localize any AI prompts if needed. Also, if teams conduct discussions in languages other than English, we'd want to ensure the AI summarization works in those languages (OpenAI supports many languages, but we'd want to test and maybe adjust prompts).
- **Error Handling Improvements:** Implement more robust fallback strategies. For example, if OpenAI is down or returns an error, maybe automatically try an alternative model or service if configured. Or cache results to avoid re-calling for the same input (though that's less likely since each input is unique conversation).
- **Security Enhancements:** If in future the plugin is used in multi-user environments, we might add role-based access. Maybe only certain roles can generate Decision Cards, or a moderation workflow (e.g., an AI-generated Decision Card stays in draft until someone approves it for publication to ensure no sensitive info is accidentally made public).

In summary, the MVP lays the foundation by focusing on the core use case (paste text -> get summary -> save post). Future iterations can deepen integration with Slack, enhance the AI and UX, and make the tool more collaborative and intelligent.

Milestone Breakdown (1-Week Build)

Day 1: Plugin Setup & Custom Post Type – Initialize a new WordPress plugin project. Define the basics: plugin header, structure, and version control (if any). Implement the `register_post_type('decision_card', ...)` in the plugin activation or init hook to create the custom post type for Decision Cards (with fields for status, owner, due date prepared). Ensure the post type appears in the WP admin menu and is editable. Test that you can manually create a Decision Card post (without AI content yet) and that custom fields (could use the default Custom Fields meta box initially) can be added. This day establishes the data structure in WordPress.

Day 2: Admin Interface & Settings – Create the admin page for the plugin. Using WordPress admin menus, add a page under an appropriate menu (e.g., top-level “Decision Cards” or under Settings/ Tools). Develop the HTML form with a textarea for the conversation and inputs for status/owner/due date. Also, create a basic Settings section where an admin can input their OpenAI API key (this could be a separate page or a section on the same page, stored in an option). Implement nonces and form handling structure (e.g., set up an `admin_post` hook or a page handler function to process the form submission). By end of Day 2, the UI should be mostly functional (except the AI call): e.g., the form submission can dump/print the input data or create a dummy post. This is tested without AI – perhaps just create a draft Decision Card with the raw conversation content to ensure the flow from form to post works.

Day 3: OpenAI API Integration – Integrate the OpenAI API call. Install any HTTP library if needed (or just use `wp_remote_post`). Write the code to take the conversation text and call the OpenAI Chat Completion API. Hard-code a prompt for now in the code (we can refine prompt later). Handle the API response: parse the JSON and extract the summary text. For testing, use a sample conversation and check that the API responds with a reasonable summary. This likely requires using an actual OpenAI API key in a development environment to verify. Also implement reading the API key from settings (so it’s not hardcoded). By end of Day 3, the end-to-end generation should work in a rudimentary way: user pastes text, clicks generate, the plugin contacts OpenAI and gets a summary. For now, maybe just display the summary on the page to verify it (you can later integrate it into post creation).

Day 4: Post Creation & Content Formatting – Now that the AI response is coming in, format the content properly and create the Decision Card post with it. Define how the summary and action items will appear: likely as a formatted string. Implement any parsing if needed (e.g., inserting "Summary:" label or splitting action items). Use `wp_insert_post` to create a new `decision_card` post with this content. Also set the post meta for status/owner/due date from the form inputs. Save as draft status initially. After creation, either redirect the user to the edit screen or show a success message with a link – implement whichever is chosen. Test this thoroughly: after generation, go to the post and see that content and meta are correctly saved. Make adjustments to formatting (maybe the summary is too long – adjust prompt, or action items format need bullets). Also handle edge cases: if API returned nothing or garbage, ensure the post isn’t created or is flagged. By end of Day 4, a user should be able to go through the normal flow and end up with a drafted Decision Card post populated by AI content.

Day 5: Error Handling, Logging & Polish – Clean up and harden the plugin. Implement the error messages in the UI: for example, if the API key is missing or invalid, catch the API error and add an admin notice on the page. If the conversation text is empty, prevent calling the API and show a validation error (“Please paste a conversation”). Add logging for important events: use `error_log` or create a log file to record successes/failures (with timestamps). Make sure the API key is not logged. Do a security review: add `sanitize_textarea_field` or similar on input, ensure output is inserted safely (consider using `wp_kses` on the AI output to strip any HTML that might be problematic). Test the UI/UX: does the button disable on click? If not using AJAX, maybe add a small script to disable it to prevent double submission. Ensure the settings page (API key entry) works and the key is retrievable (maybe test with a dummy key if not using a real call). Write basic usage instructions (could be in readme or the settings page itself). By end of Day 5, the plugin should be user-ready for an MVP: functioning as expected and handling errors gracefully.

Optional / Buffer: **Day 6: Internal Testing & Feedback** – Use this day to let a few team members test the plugin and try various conversations. Gather feedback on the summary quality and any UI confusion. Fix any bugs discovered (e.g., meta fields not saving correctly, or a formatting issue in certain cases). Tweak the AI prompt if summaries are not satisfying (maybe the summary is too verbose or missing info – adjust instructions). Ensure that the plugin works on a fresh WordPress install (try installing it on a test site from scratch, with only this plugin). Verify no PHP errors or warnings in the log. This day is about quality assurance.

Day 7: Final Adjustments & Documentation – With testing done, finalize documentation: prepare a README.md or similar explaining how to install, where to find the interface, and how to get an OpenAI API key. Note the limitations (context length, etc.) so users are aware. Clean up the code (remove any debug var_dumps, ensure comments are there for clarity). If planning to release, assign a version number 0.1.0. Do a last run-through of the entire flow as a user and make sure everything is smooth. This day buffers any remaining tasks and ensures the product is deliverable.

Each of these milestones ensures that by the end of the week, we have a fully working MVP plugin that meets the requirements: a standalone, easy-to-use WordPress plugin that transforms a pasted Slack conversation into a neatly summarized Decision Card post with minimal user effort. All core features (AI integration, custom post storage, basic UI, error handling) are implemented within this one-week timeline.

Sources:
[1] Zapier example of summarizing a Slack thread with ChatGPT and using it to create a task (illustrates value of capturing context)
[2] [3] WPBeginner on custom post types, explaining how they organize content with custom fields for better structure
[5] WPGetAPI documentation referencing OpenAI’s Chat Completion endpoint (used for generating chat-based summaries)
[6] Taylor Hughes’ experience: summarization usefulness depends on conversation length (too short or too long conversations pose challenges)
[8] Caution from a Slack-GPT summary prototype: AI summaries can’t always be fully trusted and may require checking against original messages.

---

[1] Summarize a Slack thread with ChatGPT and create a task for it in Asana
https://zapier.com/apps/asana/integrations/slack/1545912/summarize-a-slack-thread-with-chatgpt-and-create-a-task-for-it-in-asana

[2] [3] How to Create Custom Post Types in WordPress
https://www.wpbeginner.com/wp-tutorials/how-to-create-custom-post-types-in-wordpress/

[4] [5] Connect WordPress to OpenAI - WPGetAPI
https://wpgetapi.com/docs/connect-wordpress-to-openai-chatgpt/

[6] [8] Get GPT-3 To Read Your Slack, So You Don’t Have To | by Taylor Hughes | Medium
https://medium.com/@taylorhughes/get-gpt-3-to-read-your-slack-so-you-dont-have-to-2482fd2f8fdf

[7] OpenAI: o3 – Run with an API - OpenRouter
https://openrouter.ai/openai/o3/api