**Feature Specification: Dark Mode Implementation**

**1. Introduction**

* **1.1 Feature Name:** Dark Mode
* **1.2 Feature ID:** FEAT-001 (Internal tracking ID)
* **1.3 Version:** 1.0
* **1.4 Date:** June 12, 2025
* **1.5 Author(s):** Maximo Valencia
* **1.6 Stakeholders:** Product Management, UI/UX Team, Engineering Team, QA Team, Marketing.
* **1.7 Category:** Frontend

**2. Overview**

This feature aims to provide users with an alternative visual theme for the application interface, specifically a "Dark Mode." Dark Mode will invert the color scheme, typically using a dark background with light text and elements, to reduce eye strain in low-light environments and offer a personalized viewing experience. This feature will be accessible via user settings.

**3. Goals and Objectives**

* **3.1 Primary Goal:** To improve user experience by offering a customizable visual theme that reduces eye strain.
* **3.2 Objectives:**
  + Implement a toggle in user settings to switch between Light and Dark Mode.
  + Ensure all primary UI elements (headers, footers, navigation, content areas, buttons, forms) render correctly in Dark Mode.
  + Maintain readability and usability across all components.
  + Persist the user's selected theme across sessions.
  + Support both manual selection and system preference (e.g., prefers-color-scheme).

**4. User Stories**

* As a user, I want to be able to switch to a Dark Mode theme so that I can reduce eye strain, especially during evening use.
* As a user, I want my chosen theme (Light or Dark) to be remembered when I close and re-open the application.
* As a user, I want the application to respect my operating system's dark mode preference if I haven't manually selected a theme.
* As a developer, I want a clear and maintainable way to manage the application's visual themes.

**5. Functional Requirements**

* **FR-5.1 Theme Toggle:**
  + A dedicated toggle switch for "Theme" (with options "Light" and "Dark") must be present in the User Profile/Settings section.
  + Clicking the toggle must instantly apply the selected theme across the entire application.
* **FR-5.2 Theme Persistence:**
  + The user's theme preference must be stored locally (e.g., localStorage) and applied automatically upon subsequent visits.
  + If no preference is stored, the application should default to prefers-color-scheme if supported by the browser/OS. Otherwise, default to Light Mode.
* **FR-5.3 UI Element Styling:**
  + All core application components (see Section 7) must be styled appropriately for Dark Mode, ensuring sufficient contrast and readability.
  + Interactive elements (buttons, links, form fields) must clearly indicate their states (hover, active, disabled) in both themes.
* **FR-5.4 Image/Icon Handling:**
  + Images and icons should either adapt to the dark background or be sufficiently visible without becoming distracting. Consider using SVG icons that can be styled with CSS. For raster images, evaluate whether a darker version is necessary or if a slight desaturation is sufficient.
* **FR-5.5 Third-Party Integrations:**
  + Assess and address styling for any embedded third-party widgets or components to ensure they blend reasonably well with the chosen theme. (E.g., if using a chat widget, check its dark mode support).

**6. Non-Functional Requirements**

* **NFR-6.1 Performance:** Switching themes should be instant and not introduce noticeable latency or page reloads.
* **NFR-6.2 Accessibility:**
  + Ensure sufficient color contrast ratios (WCAG 2.1 AA or AAA) for both Light and Dark modes.
  + Avoid using color as the sole means of conveying information.
* **NFR-6.3 Maintainability:** The styling solution should be modular and easily extensible for future theme enhancements. (e.g., CSS variables, CSS-in-JS theming).
* **NFR-6.4 Cross-Browser Compatibility:** Support for modern browsers (Chrome, Firefox, Safari, Edge) on desktop and mobile.

**7. UI/UX Specifications**

* **7.1 Theme Toggle Placement:** The theme toggle will be located within the "Settings" menu, under a new "Appearance" sub-section.
* **7.2 Visual Design:** Refer to the attached Figma/Sketch designs for specific color palettes, typography, and component styling for both Light and Dark modes. (Link to design assets: [e.g., Figma Link])
* **7.3 Component-Specific Adaptations:**
  + **Navigation Bar:** Background color will be a dark shade, text will be light, active links will have a distinct accent color.
  + **Content Areas:** Background will be a dark gray, body text a light off-white.
  + **Buttons:** Primary buttons will have a dark background with light text in Dark Mode, secondary buttons will have a light background with dark text.
  + **Forms:** Input fields will have dark backgrounds and light text, with clear focus states.
  + **Tables:** Alternating row colors should maintain readability.
  + **Charts/Graphs:** Colors used in data visualization should be re-evaluated for contrast and clarity in Dark Mode.
* **7.4 Animations/Transitions:** Smooth transitions (e.g., transition: background-color 0.3s ease;) should be applied when switching themes to enhance the visual experience.

**8. Technical Design & Implementation Details (High-Level)**

* **8.1 Technology Stack:** React, JavaScript, TypeScript, Tailwind CSS
* **8.2 Theming Approach:**
  + Utilize CSS Variables (Custom Properties) for defining colors. This allows for easy swapping of values based on a parent class or data attribute on the <body> or <html> element.
  + Alternatively, if using a CSS-in-JS library, define theme objects that can be dynamically passed.
* **8.3 Theme State Management:**
  + Store the current theme in a React Context/Redux store for global access.
  + Persist the theme preference using localStorage (or similar client-side storage).
* **8.4 System Preference Detection:** Use window.matchMedia('(prefers-color-scheme: dark)') to detect OS dark mode preference.

**9. Test Cases (Examples)**

* **TC-9.1:** Verify theme toggle functionality:
  + Toggle from Light to Dark.
  + Toggle from Dark to Light.
  + Verify all UI elements update correctly.
* **TC-9.2:** Verify theme persistence:
  + Select Dark Mode, close and reopen the application. Verify Dark Mode is active.
  + Select Light Mode, close and reopen the application. Verify Light Mode is active.
* **TC-9.3:** Verify prefers-color-scheme integration:
  + No local preference set. Set OS to Dark Mode. Verify application defaults to Dark Mode.
  + No local preference set. Set OS to Light Mode. Verify application defaults to Light Mode.
* **TC-9.4:** Verify contrast ratios using a WCAG checker in both modes.
* **TC-9.5:** Verify accessibility with keyboard navigation and screen readers in both modes.
* **TC-9.6:** Check component rendering for all key pages/modules in both themes.

**10. Open Questions / Dependencies**

* How should embedded iframes (e.g., YouTube videos) be handled? (May require specific API calls or external libraries if no native dark mode support).
* Are there any specific third-party libraries that *do not* support dark mode, and what's our fallback strategy?
* Should we offer more than two themes in the future (e.g., "Sepia" or custom color themes)? (Out of scope for V1 but good to consider for architecture).
* What is the specific behavior if a user's prefers-color-scheme changes *while* the app is open, and they have no explicit preference set?

**11. Future Enhancements (Out of Scope for V1)**

* Automatic theme switching based on time of day.
* Support for multiple custom themes.
* Theme editor for users.

**Feature Specification: Advanced Search & Filtering**

**1. Introduction**

* **1.1 Feature Name:** Advanced Search & Filtering
* **1.2 Feature ID:** FEAT-003
* **1.3 Version:** 1.0
* **1.4 Date:** June 12, 2025
* **1.5 Author(s):** Maximo Valencia
* **1.6 Stakeholders:** Product Management, UI/UX Team, Engineering Team, Data Science (if complex search), Customers.
* **1.7 Category:** Backend, Database

**2. Overview**

This feature will enhance the application's search capabilities by introducing advanced filtering options and improving the relevance of search results. Users will be able to refine their searches based on multiple criteria, leading to more precise and efficient data discovery within the application.

**3. Goals and Objectives**

* **3.1 Primary Goal:** To enable users to quickly and accurately find specific information within the application's data.
* **3.2 Objectives:**
  + Provide dedicated filter categories (e.g., by date range, status, category, assigned user).
  + Allow combination of multiple filters.
  + Implement a clear and intuitive search interface.
  + Improve search result relevance for keyword searches.
  + Enable users to save common search queries.

**4. User Stories**

* As a user, I want to filter my results by date range so I can find items from a specific period.
* As a user, I want to filter by item status (e.g., "Open", "Closed", "Pending") to quickly see relevant tasks.
* As a user, I want to search for keywords *and* apply filters simultaneously to narrow down results.
* As a user, I want to save my frequently used filter combinations so I don't have to set them up every time.
* As a user, I want to see how many results are returned based on my search and filter criteria.

**5. Functional Requirements**

* **FR-5.1 Search Bar Enhancement:** The existing universal search bar must integrate with the new filtering mechanism.
* **FR-5.2 Filter Panel:** A collapsible or expandable filter panel must be available next to search results.
* **FR-5.3 Filter Types:** Implement the following filter types for relevant data models (e.g., "Tasks", "Documents"):
  + **Text Search:** Keyword search across relevant fields.
  + **Date Range:** From (date) and To (date) picker.
  + **Dropdown/Multi-Select:** For predefined categories (e.g., Status: Open, In Progress, Done; Type: Bug, Feature, Task).
  + **User Assignment:** Select one or more users.
* **FR-5.4 Filter Combination:** Users must be able to select multiple filters across different categories (AND logic by default).
* **FR-5.5 Clear Filters:** A "Clear All Filters" button must reset all active filters.
* **FR-5.6 Saved Searches:** Users can save their current search query and filter combination for future access.
  + Provide an input field to name the saved search.
  + List saved searches in a dropdown or dedicated section.
* **FR-5.7 Result Count:** Display the number of results matching the current search and filter criteria.
* **FR-5.8 URL Parameter Integration:** Search and filter parameters should be reflected in the URL for shareability and bookmarking.

**6. Non-Functional Requirements**

* **NFR-6.1 Performance:** Search and filter operations must return results within 1-2 seconds, even for large datasets.
* **NFR-6.2 Scalability:** The search infrastructure should be able to handle increasing data volumes and user queries.
* **NFR-6.3 Accuracy:** Search results must be highly relevant to the queried terms and filters.
* **NFR-6.4 Security:** Ensure proper sanitization of search inputs to prevent injection attacks.

**7. UI/UX Specifications**

* **7.1 Filter Panel Layout:** Clearly organized filter sections, potentially with accordions for each filter category.
* **7.2 Input Controls:** Use appropriate UI components for each filter type (e.g., calendar pickers for dates, multi-select dropdowns).
* **7.3 Active Filters Display:** Clearly show which filters are currently applied (e.g., "chips" or tags above the results, with individual clear options).
* **7.4 Empty State:** A clear message and suggestions when no results are found for a given search/filter.
* **7.5 Loading Indicator:** Visual feedback (spinner/skeleton loader) during search operations.

**8. Technical Design & Implementation Details (High-Level)**

* **8.1 Technology Stack:** JavaScript, Node.js, SQL
* **8.2 Backend Search Engine:** Implement a robust search backend (e.g., Elasticsearch, Algolia, custom SQL-based full-text search) for efficient querying and indexing.
* **8.3 API Endpoints:** Dedicated API endpoints for searching and filtering, accepting various parameters.
* **8.4 Frontend State Management:** Manage search query and filter states in the frontend (e.g., React Context, Redux, Zustand).
* **8.5 URL Sync:** Utilize browser history API (pushState, replaceState) to update URL parameters without full page reloads.

**9. Test Cases (Examples)**

* **TC-9.1:** Verify search by keyword returns relevant results.
* **TC-9.2:** Apply a single filter (e.g., by status) and verify results are narrowed correctly.
* **TC-9.3:** Apply multiple filters (e.g., keyword + status + date range) and verify combined logic.
* **TC-9.4:** Verify "Clear All Filters" resets the search and displays all original results.
* **TC-9.5:** Save a search query, log out, log in, and verify the saved search is available and loads correctly.
* **TC-9.6:** Test performance with a large number of items in the database.

**10. Open Questions / Dependencies**

* Which specific data models will be searchable and filterable in V1?
* What fields within those models should be indexed for full-text search?
* Do we need fuzzy searching or typo tolerance? (Out of scope for V1, but good to consider).
* What is the maximum number of saved searches a user can have?

**Feature Specification: Multi-Factor Authentication (MFA)**

**1. Introduction**

* **1.1 Feature Name:** Multi-Factor Authentication (MFA)
* **1.2 Feature ID:** FEAT-005
* **1.3 Version:** 1.0
* **1.4 Date:** June 12, 2025
* **1.5 Author(s):** Maximo Valencia
* **1.6 Stakeholders:** Product Management, Engineering Team, Security Team, Legal/Compliance (if applicable).
* **1.7 Category:** Backend, Security

**2. Overview**

This feature will enhance the security of user accounts by implementing Multi-Factor Authentication (MFA). Users will be able to enable MFA, requiring a second verification step (e.g., TOTP authenticator app or SMS code) in addition to their password, thereby significantly reducing the risk of unauthorized access.

**3. Goals and Objectives**

* **3.1 Primary Goal:** To significantly improve the security posture of user accounts against unauthorized access.
* **3.2 Objectives:**
  + Provide users with the option to enable/disable MFA.
  + Support Time-based One-Time Password (TOTP) applications (e.g., Google Authenticator, Authy).
  + Generate and allow users to store recovery codes for emergency access.
  + Guide users through the MFA setup process clearly and securely.

**4. User Stories**

* As a security-conscious user, I want to enable MFA on my account to protect it from unauthorized access.
* As a user, I want to use my preferred authenticator app (e.g., Google Authenticator) for MFA.
* As a user, I want to be provided with recovery codes in case I lose access to my MFA device.
* As a user, I want a clear and easy process to set up and manage MFA.
* As an administrator, I want to ensure our application meets industry security standards.

**5. Functional Requirements**

* **FR-5.1 MFA Enrollment:**
  + A "Multi-Factor Authentication" section in User Settings.
  + A button to "Enable MFA."
  + Upon clicking, initiate a guided flow for MFA setup (see UI/UX).
  + Display a QR code and a secret key for TOTP app configuration.
  + Require the user to verify the setup by entering a code from their authenticator app.
* **FR-5.2 Login Flow Integration:**
  + If MFA is enabled, after successful password entry, prompt the user for an MFA code.
  + Verify the entered MFA code against the stored secret.
* **FR-5.3 Recovery Codes:**
  + Generate a set of unique, single-use recovery codes during MFA setup.
  + Display these codes to the user with a strong recommendation to save them securely.
  + Allow users to generate new recovery codes (invalidating old ones) from settings.
* **FR-5.4 MFA Disabling:**
  + Allow users to disable MFA from their settings, requiring password re-entry for security.
* **FR-5.5 Session Management:**
  + Consider "remember me" options (e.g., "Don't ask for 30 days on this device") if appropriate, ensuring proper security precautions.
* **FR-5.6 Brute-Force Protection:** Implement rate limiting on MFA code verification attempts.

**6. Non-Functional Requirements**

* **NFR-6.1 Security:**
  + Store MFA secrets securely (e.g., encrypted in the database).
  + Ensure all communications during MFA setup and verification are encrypted (HTTPS).
  + Adhere to best practices for TOTP implementation (time synchronization, cryptographic strength).
* **NFR-6.2 Usability:** The MFA setup process should be as straightforward as possible to encourage adoption.
* **NFR-6.3 Performance:** MFA verification should not add significant latency to the login process.
* **NFR-6.4 Audit Logging:** Log successful and failed MFA attempts.

**7. UI/UX Specifications**

* **7.1 Settings Section:** A dedicated and clearly labeled MFA section within user settings.
* **7.2 Setup Wizard:** A multi-step wizard for MFA enrollment:
  + Step 1: Introduction to MFA.
  + Step 2: Display QR code and secret key.
  + Step 3: Verification (enter code from app).
  + Step 4: Display and confirm recovery codes.
  + Step 5: Completion message.
* **7.3 Login Prompt:** A dedicated input field for the MFA code during login.
* **7.4 Error Handling:** Clear error messages for invalid codes or attempts.
* **7.5 Instructions:** Concise instructions for using authenticator apps and managing recovery codes.

**8. Technical Design & Implementation Details (High-Level)**

* **8.1 Technology Stack:** OAuth 2.0, JavaScript
* **8.2 Backend MFA Library:** Utilize a robust server-side library for TOTP generation and verification (e.g., speakeasy for Node.js, PyOTP for Python, GoogleAuthenticator for Java/C#).
* **8.3 Database Schema:** Add fields to the users table for mfa\_secret (encrypted), mfa\_enabled (boolean), and a separate table for recovery\_codes.
* **8.4 API Endpoints:**
  + /api/user/mfa/enable
  + /api/user/mfa/verify (for setup and login)
  + /api/user/mfa/recovery-codes/generate
  + /api/user/mfa/disable
* **8.5 Client-Side QR Code Generation:** Use a client-side library to render the QR code from the secret key.

**9. Test Cases (Examples)**

* **TC-9.1:** Verify MFA setup process is completed successfully, and user can log in with MFA.
* **TC-9.2:** Verify MFA code validation during login (correct code works, incorrect code fails).
* **TC-9.3:** Test login with a recovery code after MFA device loss (simulated).
* **TC-9.4:** Verify disabling MFA works and user can log in with just password afterward.
* **TC-9.5:** Attempt multiple invalid MFA codes to ensure rate limiting is active.
* **TC-9.6:** Verify QR code is scannable by common authenticator apps.

**10. Open Questions / Dependencies**

* Will we support SMS-based MFA in the future? (Out of scope for V1).
* What is the desired recovery code expiration policy?
* How will we handle users who lose *both* their MFA device *and* their recovery codes? (Account recovery process, likely a separate feature).

**Feature Specification: Email Notifications for Key Events**

**1. Introduction**

* **1.1 Feature Name:** Email Notifications for Key Events
* **1.2 Feature ID:** FEAT-004
* **1.3 Version:** 1.0
* **1.4 Date:** June 12, 2025
* **1.5 Author(s):** Maximo Valencia
* **1.6 Stakeholders:** Product Management, Engineering Team, Marketing, Customer Success.
* **1.7 Category:** Backend

**2. Overview**

This feature will enable the application to send automated email notifications to users for significant events, such as task assignments, comments on their items, or status changes. Users will have granular control over which types of notifications they wish to receive.

**3. Goals and Objectives**

* **3.1 Primary Goal:** To keep users informed about critical updates and activities related to their work within the application.
* **3.2 Objectives:**
  + Send timely email notifications for predefined event types.
  + Provide users with preferences to enable/disable specific notification types.
  + Ensure emails are well-formatted, clear, and include direct links to the relevant content in the application.
  + Track email delivery status.

**4. User Stories**

* As a user, I want to receive an email when a new task is assigned to me so I don't miss important work.
* As a user, I want to receive an email when someone comments on an item I own or am following so I can stay engaged.
* As a user, I want to be able to turn off email notifications for certain types of events if I find them too frequent.
* As an administrator, I want to know if notification emails are being sent successfully.

**5. Functional Requirements**

* **FR-5.1 Notification Event Triggers:** Implement triggers for the following events (initial list, extendable):
  + New task assigned to me.
  + Comment added to my item.
  + Status change on an item I own or follow.
  + (Add more as needed based on application context)
* **FR-5.2 User Notification Preferences:**
  + A new section in User Settings will allow users to toggle each notification type (e.g., "Email me for new assignments: [On/Off]").
  + Changes to preferences must be saved instantly.
* **FR-5.3 Email Content:**
  + Emails must include a clear subject line indicating the event type.
  + Email body should contain relevant details (e.g., item name, initiator, new status, snippet of comment).
  + A direct link to the relevant item/page within the application must be included.
  + A link to "Manage your notification preferences" must be included in the footer of every email.
* **FR-5.4 Email Templating:** Use a templating system for emails to ensure consistency and ease of updates.
* **FR-5.5 Rate Limiting:** Implement rate limiting to prevent overwhelming users with too many emails in a short period.
* **FR-5.6 Delivery Status:** Log the status of email sends (e.g., sent, failed, opened - if supported by email service).

**6. Non-Functional Requirements**

* **NFR-6.1 Deliverability:** High email deliverability rate (avoiding spam folders).
* **NFR-6.2 Scalability:** The notification system must handle a growing number of users and events.
* **NFR-6.3 Timeliness:** Notifications should be sent within minutes (or near real-time) of the triggering event.
* **NFR-6.4 Security:** Email content should be sanitized to prevent XSS or other vulnerabilities.

**7. UI/UX Specifications**

* **7.1 Settings Interface:** Clear, organized toggles in user settings.
* **7.2 Email Design:** Professional, branded email templates that are responsive across different email clients.
* **7.3 Email Content Clarity:** Concise and actionable language in emails.

**8. Technical Design & Implementation Details (High-Level)**

* **8.1 Technology Stack:** Python, Node.js, Java
* **8.2 Email Service Provider (ESP) Integration:** Integrate with a reliable ESP (e.g., SendGrid, Mailgun, AWS SES).
* **8.3 Asynchronous Processing:** Use a message queue (e.g., RabbitMQ, Kafka, Redis Queue) and background workers/jobs to process notification events and send emails asynchronously.
* **8.4 Event-Driven Architecture:** Emit events when specific actions occur (e.g., task.assigned, comment.created, item.status\_changed).
* **8.5 Database Schema Changes:** Add a notification\_preferences table or extend the users table to store user choices.

**9. Test Cases (Examples)**

* **TC-9.1:** Create a new task and assign it to another user; verify they receive the assignment email.
* **TC-9.2:** Add a comment to an item owned by another user; verify the owner receives the comment email.
* **TC-9.3:** As a user, disable "New Assignment" emails. Assign a task to that user; verify no email is received.
* **TC-9.4:** Verify clicking the link in the email navigates to the correct item in the application.
* **TC-9.5:** Test email rendering across various email clients (Gmail, Outlook, Apple Mail).

**10. Open Questions / Dependencies**

* What are the exact wording and content for each email template?
* Which specific ESP will be used?
* Do we need a "digest" option for less frequent email updates? (Out of scope for V1).

## **Feature Specification: User Feedback Mechanism**

**1. Introduction**

* **1.1 Feature Name:** User Feedback Mechanism
* **1.2 Feature ID:** FEAT-006
* **1.3 Version:** 1.0
* **1.4 Date:** June 12, 2025
* **1.5 Author(s):** Maximo Valencia
* **1.6 Stakeholders:** Product Management, UI/UX Team, Engineering Team, QA Team, Customer Success, Marketing.
* **1.7 Category:** Frontend, Backend, Database

**2. Overview**

This feature will implement a direct and easily accessible way for users to submit feedback (suggestions, bug reports, general comments) directly within the application. The goal is to streamline the feedback collection process, making it simpler for users to contribute and easier for our team to categorize and respond to their input.

**3. Goals and Objectives**

* **3.1 Primary Goal:** To establish an effective channel for continuous user feedback, enabling iterative product improvement.
* **3.2 Objectives:**
  + Provide a persistent and easily discoverable "Feedback" entry point within the application.
  + Allow users to submit different types of feedback (bug, suggestion, general comment).
  + Collect essential user and context information automatically with each submission.
  + Integrate feedback submissions with a central tracking system for review and action.
  + Provide users with confirmation of their submission.

**4. User Stories**

* As a user, I want an easy way to report a bug I encounter so the development team can fix it.
* As a user, I want to submit a suggestion for a new feature so my ideas can be considered.
* As a user, I want to provide general comments about my experience without having to leave the application.
* As a product manager, I want to receive organized user feedback in a central location to inform product decisions.
* As a developer, I want clear, contextual information (e.g., browser, OS) when a bug is reported to aid in debugging.

**5. Functional Requirements**

* **FR-5.1 Feedback Entry Point:** A floating "Feedback" button or a prominent link in the footer/help menu must be present on all application pages.
* **FR-5.2 Feedback Form:** Clicking the entry point must open a modal or dedicated page with a feedback submission form.
* **FR-5.3 Feedback Type Selection:** The form must include a dropdown or radio buttons for users to select the feedback type:
  + **Bug Report:** For reporting technical issues.
  + **Feature Suggestion:** For proposing new features or enhancements.
  + **General Comment:** For anything else.
* **FR-5.4 Text Input:** A multi-line text area for the user to describe their feedback (minimum 50 characters, maximum 1000 characters).
* **FR-5.5 Optional Screenshot/Attachment (V2, but consider for future):** Ability to attach screenshots or files to submissions. (Out of scope for V1 but good to mention for future planning).
* **FR-5.6 Automatic Data Collection:** Automatically capture and include the following with each submission (invisible to the user):
  + User ID (if logged in)
  + Current URL / page path
  + Browser name and version
  + Operating System
  + Timestamp of submission
* **FR-5.7 Submission Confirmation:** Upon successful submission, display a confirmation message to the user.
* **FR-5.8 Integration with Tracking System:** Submitted feedback must be sent to a designated internal system (e.g., Jira, Asana, Zendesk Support) as a new issue/ticket, populating relevant fields.

**6. Non-Functional Requirements**

* **NFR-6.1 Security:** User-submitted content must be sanitized to prevent XSS or other injection attacks.
* **NFR-6.2 Performance:** The feedback form should load quickly and not disrupt the user's current workflow.
* **NFR-6.3 Data Integrity:** Ensure reliable delivery of feedback to the tracking system.
* **NFR-6.4 Scalability:** The system should handle a potentially high volume of feedback submissions.

**7. UI/UX Specifications**

* **7.1 Feedback Button:** A small, consistent, and easily recognizable "Feedback" button, potentially floating on the side of the screen or fixed in the footer.
* **7.2 Modal/Form Layout:** A clean, intuitive form within a modal that doesn't obscure the entire page.
* **7.3 Clear Instructions:** Concise labels and placeholder text for form fields.
* **7.4 Confirmation Message:** A friendly and reassuring message after submission, thanking the user for their input.
* **7.5 Error Handling:** Clear validation messages for invalid or missing input.
* **7.6 Branding:** Consistent branding with the rest of the application's design.

**8. Technical Design & Implementation Details (High-Level)**

* **8.1 Technology Stack:** Javascript, Node.js, Python, Java
* **8.2 Frontend Component:** Develop a reusable React/Vue/Angular component for the feedback button and modal form.
* **8.3 API Endpoint:** Create a dedicated API endpoint (e.g., /api/feedback/submit) that accepts the feedback data.
* **8.4 Backend Integration:**
  + When the API endpoint receives data, it will perform validation.
  + Then, it will use the SDK/API of the chosen tracking system (e.g., Jira API, Zendesk API) to create a new ticket.
  + Map the feedback type to a corresponding issue type or tag in the tracking system.
  + Include automatic data as custom fields or part of the ticket description.
* **8.5 Error Logging:** Implement robust error logging for failed feedback submissions to the tracking system.

**9. Test Cases (Examples)**

* **TC-9.1:** Verify clicking the "Feedback" button opens the form correctly.
* **TC-9.2:** Submit a "Bug Report" with valid text; verify confirmation message and ticket creation in the tracking system with correct type and auto-collected data.
* **TC-9.3:** Submit a "Feature Suggestion" with valid text; verify confirmation message and ticket creation.
* **TC-9.4:** Submit a "General Comment" with valid text; verify confirmation message and ticket creation.
* **TC-9.5:** Attempt to submit with empty text field; verify validation error message.
* **TC-9.6:** Verify the auto-collected data (User ID, URL, Browser, OS) is accurately captured and sent.
* **TC-9.7:** Test responsiveness of the feedback form across different device sizes.

**10. Open Questions / Dependencies**

* Which specific internal tracking system (Jira, Asana, Zendesk, etc.) will feedback be sent to?
* What are the exact fields and mappings required for the chosen tracking system?
* What is the desired response time for acknowledging or acting on submitted feedback? (More of a process question, but relevant).
* Should users be able to receive replies to their feedback via email? (Out of scope for V1, but a common follow-up request).