🚀 ProMan: Complete Development Roadmap

📋 Project Overview

Extension Name: ProMan (Prompt Manager)
Target Platform: Firefox Browser Extension
Primary Use Case: Gemini AI prompt management with universal input field support
Tech Stack: React, WebExtension APIs, Local Storage
Development Time: ~12-15 days (2-3 hours per day)
Skill Level: Beginner-friendly with AI assistance

🛠️ Technology Stack

Core Technologies

- **React 18** - UI framework
- **WebExtension API** (Firefox Manifest V3)
- **localStorage** - Data persistence
- **CSS Modules / Tailwind CSS** - Styling
- **Webpack / Vite** - Build tooling

Development Tools

- **Node.js & npm** - Package management
- **web-ext** - Firefox extension testing tool
- **ESLint** - Code quality
- **Git** - Version control

Key APIs & Libraries

- `browser.storage.local` - Extension storage
- `browser.contextMenus` - Right-click menu
- `browser.commands` - Keyboard shortcuts
- `browser.tabs` - Tab management
- Content Scripts - Page interaction
- React hooks (`useState`, `useEffect`, `useRef`)

📦 Project Structure

```
proman/
├── manifest.json                 # Extension configuration
├── package.json                  # Dependencies
├── webpack.config.js             # Build configuration
├── src/
│   ├── background/
│   │   └── background.js          # Background script
│   ├── content/
│   │   ├── content.js             # Content script
│   │   └── content.css            # Content styling
│   ├── popup/
│   │   ├── Popup.jsx              # Main popup component
│   │   ├── popup.html             # Popup HTML
│   │   └── popup.css              # Popup styling
│   ├── components/
│   │   ├── PromptSelector.jsx     # /p triggered selector
│   │   ├── PromptList.jsx         # Prompt library view
│   │   ├── PromptEditor.jsx       # Create/edit prompts
│   │   ├── SearchBar.jsx          # Search functionality
│   │   └── ExportImport.jsx       # Data management
│   ├── utils/
│   │   ├── storage.js             # Storage helpers
│   │   ├── prompts.js             # Prompt operations
│   │   └── constants.js           # App constants
│   └── styles/
│       └── global.css             # Global styles
├── public/
│   └── icons/                     # Extension icons
└── dist/                          # Build output
```

🗓️ Development Roadmap (15 Days)

Phase 1: Foundation & Setup (Days 1-2)

Day 1: Environment Setup & Project Initialization

Duration: 2-3 hours
Difficulty: ⭐⭐☆☆☆

Tasks:

1. Install Node.js and npm
2. Create project directory structure
3. Initialize npm project
4. Install dependencies
5. Create basic manifest.json
6. Set up build configuration

Deliverables:

- Working project structure
- Basic manifest.json with permissions
- Build script that creates dist folder

LLM Prompt for Day 1:

```
I'm creating a Firefox extension called "ProMan" for prompt management.
Help me set up the initial project:

1. Create a package.json with these dependencies:
   - React 18
   - Webpack 5 with necessary loaders
   - web-ext for testing
   - Babel for JSX transpilation

2. Create a webpack.config.js that:
   - Bundles React components
   - Handles CSS modules
   - Creates separate bundles for popup, content script, and background script
   - Outputs to a 'dist' folder
   - Supports development and production modes

3. Create a manifest.json (Manifest V3) for Firefox with:
   - Name: "ProMan - Prompt Manager"
   - Permissions: storage, activeTab, contextMenus
   - Content script that runs on all URLs
   - Background service worker
   - Browser action with popup
   - Keyboard commands for Ctrl+Shift+P and Ctrl+Shift+S

Include clear comments explaining each configuration option.
```

Day 2: Storage System & Data Models

Duration: 2-3 hours
Difficulty: ⭐⭐⭐☆☆

Tasks:

1. Create storage utility functions
2. Define prompt data structure
3. Implement CRUD operations for prompts
4. Add error handling for storage operations
5. Create sample data for testing

Deliverables:

- `storage.js` with all storage helpers
- `prompts.js` with prompt management logic
- Test data structure

Data Model:

```
{
  id: "unique-id-123",
  title: "Code Review Prompt",
  content: "Please review this code for...",
  tags: ["coding", "review"],
  createdAt: timestamp,
  updatedAt: timestamp,
  usageCount: 0
}
```

LLM Prompt for Day 2:

```
Create a storage management system for my Firefox extension prompt manager:

1. Create storage.js with functions:
   - savePrompt(prompt) - saves/updates a prompt
   - getPrompts() - retrieves all prompts
   - getPromptById(id) - retrieves single prompt
   - deletePrompt(id) - removes a prompt
   - searchPrompts(query) - searches by title/tags/content
   - getAllTags() - returns unique tags list

2. Use browser.storage.local API (Firefox WebExtension)
3. Each prompt should have: id, title, content, tags[], createdAt, updatedAt,
usageCount
4. Include error handling and console logging
5. Add helper function to generate unique IDs
6. Create exportPrompts() and importPrompts(data) for JSON export/import

Make the code modular, well-commented, and beginner-friendly.
```

Phase 2: Core UI Components (Days 3-5)

Day 3: Popup Interface & Prompt List

Duration: 2-3 hours
Difficulty: ⭐⭐⭐☆☆

Tasks:

1. Create popup HTML structure
2. Build main Popup.jsx component
3. Create PromptList.jsx to display prompts
4. Add basic styling with CSS
5. Connect to storage to load prompts

Deliverables:

- Working popup that opens when clicking extension icon
- List view of all saved prompts
- Basic dark/light theme support

LLM Prompt for Day 3:

```
Create the main popup interface for my prompt manager extension:

1. Create popup.html that loads React and the Popup component

2. Create Popup.jsx (React component) with:
   - Header with title and "Add Prompt" button
   - PromptList component to display all prompts
   - useState to manage prompts array
   - useEffect to load prompts from storage on mount
   - Basic error handling

3. Create PromptList.jsx that:
   - Takes prompts array as prop
   - Maps over prompts and displays them as cards
   - Shows title, first 50 chars of content, and tags
   - Has edit and delete buttons for each prompt
   - Shows empty state when no prompts exist

4. Add popup.css with:
   - Clean, modern design
   - CSS variables for light/dark mode
   - Card-based layout for prompts
   - Responsive spacing

Use modern React patterns (functional components, hooks). Include helpful
comments.
```

Day 4: Prompt Editor Component

Duration: 2-3 hours
Difficulty: ⭐⭐⭐☆☆

Tasks:

1. Create PromptEditor.jsx component
2. Build form with title, content, tags inputs
3. Implement save/cancel functionality
4. Add form validation
5. Support both create and edit modes

Deliverables:

- Working form to create/edit prompts
- Tag input with comma-separated values
- Validation for required fields

LLM Prompt for Day 4:

```
Create a PromptEditor component for creating/editing prompts:

1. Create PromptEditor.jsx that:
   - Takes props: prompt (optional, for editing), onSave, onCancel
   - Has form inputs for: title, content (textarea), tags
   - Uses controlled components with useState
   - Tags input accepts comma-separated values
   - Has Save and Cancel buttons
   - Validates that title and content are not empty
   - Calls onSave with prompt object when submitted

2. Features needed:
   - Auto-focus on title input when opened
   - Character counter for content (helpful but not limiting)
   - Tag pills display below input showing parsed tags
   - Handle both "create new" and "edit existing" modes
   - Clear form after successful save

3. Styling:
   - Full-width modal/overlay style
   - Large textarea for content (at least 200px height)
   - Clean, accessible form design
   - Visual feedback for validation errors

Include detailed comments explaining the component logic.
```

Day 5: Search & Filter Functionality

Duration: 2-3 hours
Difficulty: ⭐⭐⭐☆☆

Tasks:

1. Create SearchBar.jsx component
2. Implement real-time search filtering
3. Add tag filter chips
4. Sort prompts by recent/usage/alphabetical
5. Integrate search with PromptList

Deliverables:

- Working search bar with instant results
- Tag-based filtering
- Sort options

LLM Prompt for Day 5:

```
Create search and filter functionality for the prompt manager:

1. Create SearchBar.jsx component that:
   - Has a text input for searching
   - Shows a dropdown/chips for tag filtering
   - Has sort options: Recent, Most Used, A-Z
   - Emits onChange events to parent with search criteria
   - Shows clear button when search is active
   - Debounces search input (300ms) for performance

2. Update Popup.jsx to:
   - Add SearchBar component above PromptList
   - Implement filtering logic that searches across title, content, and tags
   - Apply tag filters when selected
   - Apply sorting based on selected option
   - Show result count when searching

3. Create a utility function filterAndSortPrompts() that:
   - Takes prompts array and search criteria
   - Returns filtered and sorted results
   - Handles case-insensitive search
   - Supports multiple tag filters (AND logic)

Make the search feel instant and smooth. Add helpful comments.
```

Phase 3: Content Script & Input Detection (Days 6-8)

Day 6: Content Script Foundation

Duration: 2-3 hours
Difficulty: ⭐⭐⭐⭐☆

Tasks:

1. Create content script file
2. Detect all input fields and textareas on page

3. Listen for `/p` trigger in input fields
4. Implement keyboard event handling
5. Test on multiple websites including Gemini

Deliverables:

- Content script that runs on all pages
- Detection of `/p` trigger
- Console logging to verify detection works

LLM Prompt for Day 6:

```
Create a content script for detecting prompt triggers in input fields:

1. Create content.js that:
   - Listens for keydown events on the entire document
   - Detects when user types "/p" in any input field or textarea
   - Identifies the currently focused input element
   - Stores reference to the input element for later use
   - Prevents the "/p" from being entered in the input
   - Logs to console when trigger detected (for debugging)

2. Handle edge cases:
   - Detect contenteditable divs (used by Gemini and modern chat UIs)
   - Work with both input/textarea and contenteditable elements
   - Don't trigger if user types "//p" or "/P" (only lowercase "/p")
   - Only trigger at start of line or after whitespace

3. Create helper functions:
   - isInputElement(element) - checks if element accepts text input
   - getCurrentCursorPosition(element) - gets cursor position
   - insertTextAtCursor(element, text) - inserts text at cursor

4. Add special detection for Gemini:
   - Identify Gemini's main prompt input
   - Identify Gemini's system instruction input
   - Use data attributes or unique selectors

Include detailed comments explaining the event handling logic.
```

Day 7: Prompt Selector Overlay

Duration: 2-3 hours
Difficulty: ⭐⭐⭐⭐☆

Tasks:

1. Create PromptSelector.jsx component
2. Build overlay that appears near input field
3. Display filtered list of prompts
4. Implement keyboard navigation (arrow keys, enter)
5. Position overlay intelligently near cursor

Deliverables:

- Overlay component that appears when `/p` is typed
- Keyboard navigation working

- Click to select prompt

LLM Prompt for Day 7:

```
Create an in-page prompt selector overlay component:

1. Create PromptSelector.jsx that:
   - Renders as an absolutely positioned overlay
   - Shows a filtered list of prompts
   - Has a search input at the top
   - Supports keyboard navigation (↑↓ arrows, Enter to select, Esc to close)
   - Highlights currently selected item
   - Emits onSelect event with chosen prompt
   - Can be positioned near the cursor/input field

2. In content.js, add logic to:
   - When "/p" is detected, inject the React overlay into the page
   - Create a React root and render PromptSelector
   - Pass available prompts from storage
   - Calculate position near the input field (avoid going off-screen)
   - Handle selection: insert prompt content into the input field
   - Clean up and remove overlay after selection or Esc

3. Create content.css for the overlay:
   - High z-index to appear above all page content
   - Dropdown/modal style with shadow
   - Maximum height with scroll for long lists
   - Smooth animations for appear/disappear
   - Adapts to light/dark mode

4. Handle text insertion:
   - For input/textarea: set value and dispatch input event
   - For contenteditable: insert at cursor position
   - Maintain cursor position after insertion

This is complex! Include step-by-step comments and error handling.
```

Day 8: Keyboard Shortcuts

Duration: 2-3 hours
Difficulty: ⭐⭐⭐☆☆

Tasks:

1. Implement Ctrl+Shift+P shortcut for prompt selector
2. Implement Ctrl+Shift+S for quick save
3. Handle Mac vs Windows key differences
4. Test shortcuts don't conflict with browser/page shortcuts

Deliverables:

- Both keyboard shortcuts working
- Cross-platform compatibility (Ctrl vs Command)

LLM Prompt for Day 8:

```
Implement keyboard shortcuts for the extension:

1. In manifest.json, define commands:
   - "open-prompt-selector": Ctrl+Shift+P (Cmd+Shift+P on Mac)
   - "quick-save-prompt": Ctrl+Shift+S (Cmd+Shift+S on Mac)

2. In background.js:
   - Listen for browser.commands.onCommand
   - When "open-prompt-selector" is triggered:
     * Send message to content script to show prompt selector
     * Include the active tab ID
   - When "quick-save-prompt" is triggered:
     * Send message to content script to save selected text
     * If no text selected, show notification

3. In content.js:
   - Listen for messages from background script
   - On "open-prompt-selector": show the PromptSelector overlay
   - On "quick-save-prompt":
     * Get currently selected text on page
     * If text exists, open a mini save dialog
     * Pre-fill content with selected text
     * Save to storage with auto-generated title

4. Create a notification system:
   - Show temporary toast notifications
   - Success message when prompt saved
   - Error messages if something fails

Include cross-browser considerations and clear comments.
```

Phase 4: Advanced Features (Days 9-11)

Day 9: Context Menu (Right-Click) Integration

Duration: 2-3 hours
Difficulty: ⭐⭐⭐☆☆

Tasks:

1. Add context menu item in background script
2. Handle "Save as Prompt" on text selection
3. Open quick save dialog with pre-filled content
4. Test on various websites

Deliverables:

- Right-click menu option for selected text
- Quick save flow working

LLM Prompt for Day 9:

```
Implement right-click context menu for saving prompts:

1. In background.js:
   - Create context menu item using browser.contextMenus.create()
   - Title: "Save as Prompt"
   - Context: "selection" (only show when text is selected)
   - Listen for browser.contextMenus.onClicked
   - When clicked, send message to content script with selected text

2. In content.js:
   - Listen for messages from background script
   - When "save-selection" message received:
     * Create a small modal/dialog for saving
     * Pre-fill content with the selected text
     * Ask user for title and tags
     * Save to storage when confirmed
     * Show success notification

3. Create a QuickSaveDialog component:
   - Lightweight modal that appears on the page
   - Simple form: title input, tags input, content textarea (pre-filled)
   - Save and Cancel buttons
   - Positioned in center of viewport with overlay
   - Auto-focus on title input

4. Add permissions check:
   - Ensure manifest.json has "contextMenus" permission
   - Handle cases where selection API might not work

Include error handling and user feedback for all steps.
```

Day 10: Import/Export Functionality

Duration: 2-3 hours
Difficulty: ⭐⭐⭐☆☆

Tasks:

1. Create ExportImport.jsx component
2. Implement JSON export with download
3. Implement JSON import with validation
4. Add backup/restore functionality
5. Handle merge vs replace on import

Deliverables:

- Export button creates downloadable JSON file
- Import button accepts JSON and validates
- User can choose to merge or replace existing prompts

LLM Prompt for Day 10:

```
Create import/export functionality for prompt data:

1. Create ExportImport.jsx component with:
   - "Export All Prompts" button
   - "Import Prompts" button with file input
   - Display statistics (total prompts, tags, etc.)
   - Option to "Replace All" or "Merge" on import

2. Export functionality:
   - Get all prompts from storage
   - Create JSON object with metadata (version, exportDate, count)
   - Generate downloadable JSON file
   - Filename: "proman-prompts-YYYY-MM-DD.json"
   - Use Blob and URL.createObjectURL for download

3. Import functionality:
   - Accept JSON file upload
   - Validate file structure
   - Check for required fields in each prompt
   - Show preview of what will be imported
   - Handle two modes:
     * Replace: Clear existing prompts, import new ones
     * Merge: Keep existing, add new (skip duplicates by title)
   - Show success message with import summary

4. Add data validation:
   - Ensure imported data has correct structure
   - Sanitize inputs to prevent issues
   - Handle errors gracefully with user-friendly messages
   - Create backup before replacing all data

5. Add this component to the Popup with a Settings/Data Management tab

Include detailed comments and error handling at each step.
```

Day 11: Theme System & Polish

Duration: 2-3 hours
Difficulty: ⭐⭐☆☆☆

Tasks:

1. Implement system theme detection
2. Add CSS variables for light/dark mode
3. Apply consistent theming across all components
4. Add smooth transitions between themes
5. Test in both modes

Deliverables:

- Automatic theme switching based on system preference
- Consistent styling across popup and overlays
- Smooth visual experience

LLM Prompt for Day 11:

```
Implement automatic light/dark theme system:

1. Create theme detection utility:
   - Detect system preference using window.matchMedia
   - Listen for theme changes in real-time
   - Apply theme class to root elements

2. Update all CSS files to use CSS variables:
   - Define color variables for light and dark modes
   - Example variables: --bg-primary, --text-primary, --border-color, --accent-
color
   - Use prefers-color-scheme media query

3. Theme variables to define:
   Light mode:
   - Background: #ffffff, #f5f5f5
   - Text: #1a1a1a, #666666
   - Borders: #e0e0e0
   - Accent: #4285f4

   Dark mode:
   - Background: #1e1e1e, #2d2d2d
   - Text: #e0e0e0, #a0a0a0
   - Borders: #404040
   - Accent: #8ab4f8

4. Update all components:
   - Replace hardcoded colors with CSS variables
   - Ensure overlays/modals have proper backdrop
   - Test readability in both modes
   - Add smooth transitions (200ms) for theme changes

5. Special handling for content script overlays:
   - Detect page theme (light/dark)
   - Match overlay theme to page or use contrast
   - Ensure overlay is always readable

Apply the theme system consistently across popup.css, content.css, and all
component styles.
```

Phase 5: Testing & Refinement (Days 12-14)

Day 12: Gemini-Specific Testing & Optimization

Duration: 2-3 hours
Difficulty: ⭐⭐⭐⭐☆

Tasks:

1. Test extension on Gemini's main prompt input
2. Test on Gemini's system instruction input
3. Handle Gemini-specific UI quirks
4. Optimize selector positioning for Gemini
5. Test prompt insertion and verify it works correctly

Deliverables:

- Extension working flawlessly on both Gemini inputs
- Documented any special handling needed
- Optimized UX for Gemini workflow

LLM Prompt for Day 12:

```
Optimize the extension for Google Gemini:

1. Analyze Gemini's input elements:
   - Main chat input: Identify the correct selector
   - System instruction input: Identify the correct selector
   - Determine if they use contenteditable or other elements
   - Check for any special attributes or classes

2. Create Gemini-specific handlers in content.js:
   - Function detectGeminiInputs() that finds both inputs
   - Enhanced text insertion for Gemini's editor
   - Handle any special Gemini editor behavior
   - Preserve formatting if Gemini supports it
```

```
3. Test scenarios to implement:
   - Insert prompt in empty input
   - Insert prompt with existing text
   - Insert prompt at cursor position (middle of text)
   - Trigger from keyboard shortcut on Gemini
   - Save Gemini text as new prompt

4. Positioning optimization:
   - Make overlay appear in optimal position for Gemini's layout
   - Ensure overlay doesn't cover important UI elements
   - Handle Gemini's responsive design (different screen sizes)

5. Create fallback mechanisms:
   - If Gemini changes their DOM structure, try multiple selectors
   - Graceful degradation if specific targeting fails
   - User notification if insertion fails

Document Gemini's specific selectors and any workarounds needed.
Visit gemini.google.com to inspect the actual DOM structure.
```

Day 13: Bug Fixes & Edge Cases

Duration: 2-3 hours
Difficulty: ⭐⭐⭐☆☆

Tasks:

1. Test on multiple websites (ChatGPT, Claude, etc.)
2. Fix any bugs discovered during testing
3. Handle edge cases (empty states, long content, special characters)
4. Improve error messages
5. Add loading states where needed

Deliverables:

- Bug fix list with resolutions
- Improved error handling
- Better UX for edge cases

LLM Prompt for Day 13:

```
Help me create a comprehensive testing and bug-fixing plan:

1. Create a test checklist covering:
   - All core features (save, edit, delete, search, export, import)
   - All trigger methods (/p, shortcuts, context menu)
   - Different input types (input, textarea, contenteditable)
   - Different websites (Gemini, ChatGPT, Claude, generic text fields)
   - Edge cases: empty prompts, very long prompts (10,000+ chars), special
characters
   - Performance: 100+ prompts, rapid searching, quick save/load

2. Common bugs to check for:
   - Storage race conditions
   - Memory leaks from event listeners
   - Overlay not cleaning up properly
   - Keyboard shortcuts conflicting
   - Content script injection failures
   - Theme not applying correctly
   - Import validation failures

3. Create error handling improvements:
   - Try-catch blocks around all storage operations
   - User-friendly error messages (no technical jargon)
   - Graceful degradation when features fail
   - Console logging for debugging (with clear prefixes)

4. Add loading states:
   - Show spinner when loading prompts in popup
   - Disable buttons during save operations
   - Show feedback during import/export
   - Debounce rapid user actions

5. Create a debug mode:
   - Log all major operations to console
   - Add data-debug attributes for testing
   - Create helper function to dump current state

Provide code snippets for common bug fixes and error handling patterns.
```

Day 14: Documentation & Packaging

Duration: 2-3 hours
Difficulty: ⭐⭐☆☆☆

Tasks:

1. Write comprehensive README.md
2. Create user guide with screenshots
3. Add inline code comments
4. Create installation instructions
5. Prepare for Firefox Add-ons submission
6. Build final production version

Deliverables:

- Complete README with features, installation, usage
- User guide document
- Packaged .zip file ready for Firefox Add-ons
- Release notes

LLM Prompt for Day 14:

```
Help me create documentation and package the extension:

1. Create a comprehensive README.md including:
   - Project title and description
   - Features list with emojis
   - Screenshots (placeholder descriptions)
   - Installation instructions (from source and from Firefox Add-ons)
   - Usage guide:
     * How to create prompts
     * How to use /p trigger
     * How to use keyboard shortcuts
     * How to import/export
   - Gemini-specific tips
   - Keyboard shortcuts reference table
   - Troubleshooting section
   - Privacy policy (local storage only)
   - Contributing guidelines
   - License (suggest MIT)

2. Create USER_GUIDE.md with:
   - Step-by-step tutorials for each feature
   - Best practices for organizing prompts
   - Tips for effective prompt management
   - FAQ section

3. Pre-submission checklist for Firefox Add-ons:
   - Verify manifest.json has all required fields
   - Add description, homepage_url, author
   - Ensure icons exist in 48x48 and 96x96
   - Remove console.logs in production build
   - Test in clean Firefox profile
   - Verify all permissions are justified

4. Create build script for production:
   - Webpack production mode
   - Minify code
   - Remove source maps
   - Create .zip file with only necessary files
   - Exclude node_modules, src files, .git

5. Create CHANGELOG.md for version 1.0.0

Provide templates for all documentation files.
```

Phase 6: Polish & Launch (Day 15)

Day 15: Final Testing & Deployment

Duration: 2-3 hours
Difficulty: ⭐⭐☆☆☆

Tasks:

1. Final round of testing on Gemini
2. Test on fresh Firefox profile
3. Get feedback from 2-3 test users
4. Make final adjustments
5. Submit to Firefox Add-ons (or prepare for self-distribution)
6. Create demo video or GIF

Deliverables:

- Fully tested extension
- Submitted to Firefox Add-ons (or ready for distribution)
- Demo materials

LLM Prompt for Day 15:

```
Help me prepare for launch:
```

```
1. Final testing checklist:
   - Install extension in clean Firefox profile
   - Test every feature systematically
   - Verify Gemini integration works perfectly
   - Check performance with realistic data (50+ prompts)
   - Test import/export with sample data
   - Verify all keyboard shortcuts work
   - Test on both Windows and Mac (if possible)

2. Create sample data for demos:
   - Generate 10-15 realistic example prompts
   - Include various categories (coding, writing, analysis)
   - Create sample export file users can import

3. Firefox Add-ons submission process:
   - Create developer account at addons.mozilla.org
   - Prepare extension listing:
     * Catchy title and subtitle
     * Detailed description (200+ words)
     * Feature highlights
     * Screenshots (3-5 images)
     * Privacy policy explanation
   - Upload .zip file
   - Fill out questionnaire about permissions
   - Submit for review

4. Alternative: Self-distribution instructions:
   - How to load unpacked extension in Firefox
   - How to install from .xpi file
   - Hosting considerations for updates

5. Create demo materials:
   - GIF showing /p trigger in action
   - GIF showing keyboard shortcuts
   - GIF showing save from context menu
   - Short video (1-2 min) showing full workflow

6. Post-launch plan:
   - How to gather user feedback
   - Version update strategy
   - Bug report template

Provide guidance for the submission process and marketing the extension.
```

🎯 Success Metrics

Technical Milestones

- ✅ Extension installs without errors
- ✅ All features work on Gemini (primary target)
- ✅ /p trigger responds within 100ms
- ✅ Popup loads in <200ms with 100 prompts
- ✅ No console errors in production
- ✅ Passes Firefox Add-ons review

User Experience Goals

- ✅ New user can create first prompt in <30 seconds
- ✅ Prompt insertion feels instant
- ✅ Search returns results as you type
- ✅ UI adapts to system theme automatically
- ✅ Export/import preserves all data perfectly

🚨 Common Pitfalls & Solutions

Pitfall 1: Content Script Not Injecting

Solution: Verify manifest permissions, check for CSP conflicts, add timeout retry logic

Pitfall 2: Overlay Not Positioning Correctly

Solution: Calculate position relative to viewport, handle page scroll, add boundary detection

Pitfall 3: Storage Data Corruption

Solution: Always validate before saving, implement data versioning, create automatic backups

Pitfall 4: Keyboard Shortcuts Not Working

Solution: Check for conflicts with browser shortcuts, test on clean profile, verify manifest commands

Pitfall 5: React Not Rendering in Content Script

Solution: Create isolated root, avoid conflicts with page's React, use Shadow DOM if needed

---

📚 Learning Resources

Essential Reading

- **MDN WebExtensions API**: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions
- **React Documentation**: https://react.dev
- **Firefox Extension Workshop**: https://extensionworkshop.com

Helpful Tutorials

- Building a Firefox extension with React
- Content scripts vs background scripts explained
- Working with storage in extensions
- Manifest V3 migration guide

Community Support

- r/firefox (Reddit)
- Stack Overflow (tag: firefox-addon)
- Firefox Add-ons discourse

🔄 Future Enhancements (Post-MVP)

Version 1.1

- Prompt variables ({{date}}, {{selection}})
- Folder organization
- Prompt templates

Version 1.2

- Multi-language support
- Prompt statistics and analytics
- Quick edit mode

Version 1.3

- Prompt chaining/sequences
- Advanced search with regex
- Custom keyboard shortcuts

Version 2.0

- Cloud sync option
- Community prompt library
- AI-powered prompt suggestions
- Chrome extension port

🎓 Tips for Success

For the Developer

1. **Test early, test often** - Don't wait until Day 14 to test on Gemini

2. **Commit code daily** - Use Git from Day 1, commit at end of each session

3. **Ask specific questions** - When stuck, give the AI clear context and error messages

4. **Start simple** - Get basic version working before adding complexity

5. **Take breaks** - Step away when frustrated, fresh eyes find bugs faster

When Using AI Assistance

1. **Show your code** - Always include relevant code when asking for help
2. **Describe what you tried** - Explain your debugging attempts
3. **Ask for explanations** - Don't just copy code, understand why it works
4. **Request alternatives** - Ask "What's another way to do this?"
5. **Validate suggestions** - Test AI-generated code thoroughly

Debugging Strategies

1. **Use console.log liberally** - Add logs at each step to trace execution
2. **Check the browser console** - Firefox Dev Tools (F12) is your friend
3. **Test in isolation** - Comment out code to isolate the problem
4. **Read error messages carefully** - They usually tell you exactly what's wrong
5. **Use web-ext run** - Test in temporary Firefox profile to avoid contamination

---

🛠️ Development Commands Reference

```
# Initial setup
npm install

# Development mode with auto-reload
npm run dev
# or
web-ext run --source-dir ./dist

# Build for production
npm run build

# Package for distribution
npm run package
# Creates: proman-v1.0.0.zip

# Lint code
npm run lint

# Clean build directory
npm run clean
```

📋 Daily Checklist Template

Use this at the start of each day:

```
## Day X: [Feature Name]

### Before Starting
- [ ] Read the day's tasks and LLM prompt
- [ ] Review yesterday's code
- [ ] Open Firefox Dev Tools
- [ ] Have Gemini open for testing

### During Development
- [ ] Copy LLM prompt and get initial code
- [ ] Understand the code before implementing
- [ ] Test each function as you write it
- [ ] Add console.logs for debugging
- [ ] Commit code with descriptive message

### Before Finishing
- [ ] Test the day's feature thoroughly
- [ ] Document any issues encountered
- [ ] Update TODO list for next day
- [ ] Push code to Git
- [ ] Write quick note: "What worked, what didn't"
```

🎨 Design Guidelines

Color Palette

Light Mode

```
--bg-primary: #ffffff
--bg-secondary: #f5f5f5
--bg-hover: #e8e8e8
--text-primary: #1a1a1a
--text-secondary: #666666
--border: #e0e0e0
--accent: #4285f4
--accent-hover: #3367d6
--success: #34a853
--error: #ea4335
```

Dark Mode

```
--bg-primary: #1e1e1e
--bg-secondary: #2d2d2d
--bg-hover: #3a3a3a
--text-primary: #e0e0e0
--text-secondary: #a0a0a0
--border: #404040
--accent: #8ab4f8
--accent-hover: #aecbfa
--success: #81c995
--error: #f28b82
```

Typography

- **Headings**: System font stack (`-apple-system, BlinkMacSystemFont, "Segoe UI", Roboto`)
- **Body**: 14px for readability
- **Code**: `"Consolas", "Monaco", monospace`

Spacing Scale

- XS: 4px
- SM: 8px
- MD: 16px
- LG: 24px
- XL: 32px

Component Patterns

- **Cards**: 8px border-radius, subtle shadow
- **Buttons**: 6px border-radius, clear hover states
- **Inputs**: Full-width, 36px height, focus ring
- **Modal overlays**: Semi-transparent backdrop, centered content

---

🔐 Security Considerations

Data Privacy

- ✅ All data stored locally (no external servers)
- ✅ No analytics or tracking
- ✅ No network requests except to extension store
- ✅ Export files are unencrypted JSON (user owns data)

Content Script Safety

- ✅ Don't execute arbitrary code from prompts
- ✅ Sanitize user input before display
- ✅ Use textContent instead of innerHTML where possible
- ✅ Validate all imported data

Permissions Justification

```
{
  "storage": "Required to save prompts locally",
  "activeTab": "Required to insert prompts into input fields",
  "contextMenus": "Required for right-click 'Save as Prompt' feature"
}
```

📊 Project Timeline Visualization

```
Week 1: Foundation
├── Day 1-2: Setup & Storage ████████
├── Day 3-5: UI Components   ████████████
└── Day 6-7: Content Script  ████████

Week 2: Features & Polish
├── Day 8-9:  Shortcuts      ████████
├── Day 10-11: Import/Export ████████
├── Day 12-13: Testing       ████████
└── Day 14-15: Launch Prep   ████████
```

🐛 Troubleshooting Guide

Issue: Extension won't load

```
Possible causes:
1. Manifest syntax error → Check JSON validity
2. Missing required fields → Verify manifest version, name, version
3. Invalid permissions → Check spelling of permission names

Solution: Run `web-ext lint` to validate
```

Issue: Content script not detecting inputs

```
Possible causes:
1. Script injecting too early → Add delay or use DOMContentLoaded
2. Input is in iframe → Check frame permissions
3. Input uses Shadow DOM → Adjust selectors

Solution: Add logging to verify injection timing
```

Issue: Prompts not saving

```
Possible causes:
1. Storage quota exceeded → Check used space
2. Invalid data format → Validate before save
3. Storage permission missing → Check manifest

Solution: Wrap saves in try-catch, log errors
```

Issue: Overlay appears behind page content

```
Possible causes:
1. Page has high z-index → Increase overlay z-index (9999)
2. CSS isolation issue → Use !important sparingly
3. Positioned incorrectly → Check position: fixed vs absolute

Solution: Inspect computed styles in dev tools
```

Issue: React not rendering

```
Possible causes:
1. React not loaded → Check script injection order
2. Babel not transpiling → Verify webpack config
3. JSX syntax error → Check console for errors

Solution: Test with simple <div>Hello</div> first
```

💡 Pro Tips

Efficiency Hacks

1. **Snippet library**: Save common code patterns for reuse
2. **Hot reload**: Use webpack dev server for instant updates
3. **Dev shortcuts**: Create npm scripts for common tasks

4. **Browser profiles**: Keep separate profile for extension testing
5. **Dual monitors**: Code on one, test on the other

Code Quality

1. **DRY principle**: Extract repeated code into functions
2. **Named functions**: Better than anonymous for debugging
3. **Early returns**: Reduce nesting with guard clauses
4. **Const by default**: Only use let when reassignment needed
5. **Comments for "why"**: Code shows "what", comments explain "why"

Git Workflow

```
# Daily commits
git add .
git commit -m "Day X: Implemented [feature]"

# Feature branches (optional for solo dev)
git checkout -b feature/prompt-selector
git merge feature/prompt-selector

# Tag releases
git tag v1.0.0
```

📞 Getting Help

When You're Stuck

Before asking for help:

1. Read the error message completely
2. Check browser console for additional errors
3. Verify your code matches the expected structure
4. Try the simplest version that could work
5. Google the exact error message

When asking AI for help:

```
I'm working on [specific feature] in my Firefox extension.
I'm trying to [expected behavior].
Instead, [actual behavior is happening].

Here's the relevant code:
[paste code]

Here's the error:
[paste error message]

What I've tried:
- [attempt 1]
- [attempt 2]

What could be causing this?
```

Useful Resources by Phase

Days 1-2 (Setup)

- WebExtension Getting Started: https://mzl.la/3xYZ
- React + Webpack Tutorial
- Understanding package.json

Days 3-5 (UI)

- React Hooks Documentation
- CSS Grid/Flexbox Guide
- Component composition patterns

Days 6-8 (Content Scripts)

- Content Scripts API Reference
- DOM Event Handling
- Input element manipulation

Days 9-11 (Features)

- Context Menus API
- File handling in browser
- CSS custom properties

Days 12-15 (Testing)

- Browser testing strategies
- Firefox Add-ons Review Criteria
- Creating extension screenshots

🎯 Definition of Done

For Each Feature

- [ ] Code is written and commented
- [ ] Feature works in Firefox (latest version)
- [ ] No console errors when using feature
- [ ] Tested on Gemini specifically
- [ ] Edge cases handled gracefully
- [ ] Committed to Git with clear message

For the Complete Project

- [ ] All core features implemented
- [ ] Works flawlessly on Gemini (both inputs)
- [ ] Works on at least 3 other sites
- [ ] All keyboard shortcuts functional
- [ ] Import/export tested with real data
- [ ] Theme switches smoothly

- [ ] No memory leaks after extended use
- [ ] README and documentation complete
- [ ] Extension packaged and tested from .zip
- [ ] Ready for Firefox Add-ons submission

📈 Metrics to Track

Development Metrics

- Lines of code written per day
- Bugs fixed vs new bugs found
- Time spent on each phase
- Features completed vs planned

Performance Metrics

- Extension load time: < 200ms
- Search response time: < 50ms
- Overlay appearance time: < 100ms
- Storage operation time: < 50ms

Quality Metrics

- Console errors: 0 in production
- User-reported bugs: Track and fix
- Test coverage: Manual testing checklist
- Code review findings: Self-review each day

🎓 Learning Outcomes

By completing this project, you will learn:

Technical Skills

- ✅ Modern JavaScript (ES6+, async/await, modules)
- ✅ React fundamentals (components, hooks, state management)
- ✅ WebExtension APIs (storage, content scripts, messaging)
- ✅ DOM manipulation and event handling
- ✅ CSS (layouts, theming, responsive design)
- ✅ Build tools (Webpack, npm scripts)
- ✅ Git version control
- ✅ Browser Developer Tools

Soft Skills

- ✅ Breaking down complex projects

- ✅ Following technical specifications
- ✅ Debugging systematically
- ✅ Reading and understanding documentation
- ✅ Managing scope and priorities
- ✅ Testing and quality assurance

Product Skills

- ✅ User experience design
- ✅ Feature prioritization
- ✅ Edge case consideration
- ✅ Performance optimization
- ✅ Deployment and distribution

🚀 Bonus Challenges

After completing the MVP, try these:

Easy (1-2 hours each)

1. **Prompt shortcuts**: Assign custom shortcuts to favorite prompts
2. **Recent prompts**: Show last 5 used prompts for quick access
3. **Duplicate detection**: Warn when creating similar prompts
4. **Bulk operations**: Select multiple prompts to delete/export

Medium (3-4 hours each)

1. **Prompt templates**: Create prompts with `{{placeholders}}`
2. **Folders/Categories**: Organize beyond just tags
3. **Keyboard-only mode**: Navigate entire UI without mouse
4. **Undo/Redo**: For prompt edits and deletions

Hard (5+ hours each)

1. **Chrome port**: Make it work in Chrome
2. **Sync service**: Optional cloud backup via GitHub Gist
3. **Prompt sharing**: Export single prompt as shareable link
4. **AI suggestions**: Suggest tags or improvements to prompts

---

📝 Final Notes

Remember

- **Progress > Perfection**: Get it working, then make it better
- **User first**: Every decision should improve the user experience
- **Privacy matters**: Users trust you with their data

- **Have fun**: This should be enjoyable, not stressful

When You Finish

1. **Celebrate**: You built a real, useful tool!
2. **Share**: Show it to friends, post on Reddit/Twitter
3. **Iterate**: Gather feedback and improve
4. **Reflect**: Write about what you learned
5. **Next project**: Apply these skills to something new

Stay Connected

- Consider creating a GitHub repository
- Write a blog post about your experience
- Help others who are building similar projects
- Keep learning and building!

📜 License Suggestion

🎉 You're Ready to Build!

This roadmap gives you everything you need to create **ProMan**, a production-ready Firefox extension for managing prompts. The structure is designed to:

- **Build incrementally**: Each day adds one clear feature
- **Learn progressively**: Concepts build on each other
- **Stay motivated**: See visible progress daily
- **Avoid overwhelm**: Manageable chunks with clear goals
- **Enable AI assistance**: Detailed prompts for every step

Your Next Steps

1. **Bookmark this roadmap** - You'll reference it daily
2. **Set up your workspace** - Clean desk, good chair, dev tools ready
3. **Schedule your time** - Block 2-3 hours per day for 15 days
4. **Start Day 1 tomorrow** - Don't wait for the perfect moment
5. **Trust the process** - Follow the plan, even when it feels hard

Good Luck! 🚀

You've got this. In just 15 days, you'll have built something real, something useful, and something you can be proud of. The skills you learn will serve you for years to come.

Now go build something amazing! 💪

Need help along the way? Just come back with specific questions, error messages, or code snippets, and we'll debug together. Happy coding! 🎨