PR Builder TUI - Complete Visual Guide

This document provides a comprehensive visual walkthrough of the PR Builder Terminal User Interface (TUI), showing all screens and functionality with detailed explanations.

Overview

The PR Builder TUI is built using **Bubbletea**, a Go framework for building terminal user interfaces based on the Elm architecture. The interface provides an interactive way to manage PR description generation, offering keyboard-driven navigation and real-time feedback.

Architecture

The TUI follows the **Model-View-Controller (MVC)** pattern as implemented by Bubbletea:

- **Model**: Contains all application state (selected file index, generation status, error state)
- **Update**: Pure function that handles messages (key presses, async events) and returns new state
- **View**: Pure function that renders the current state to a string

This architecture ensures predictable state management and makes the UI easy to reason about and test.

Screen 1: Main File List View

The main screen is the primary interface where users spend most of their time. It displays the list of changed files and provides navigation and selection controls.

Visual Layout

```
╭──────────────────────────────────────────────────────────────────────────────
│
│
│                        PR DESCRIPTION GENERATOR
│
│
│
│  feature/user-auth → master
│
│
│
│  Files: 3 | Tokens: 793
│
│
│
│  CHANGED FILES
│
│
───────────────────────────────────────────────────────────────────────────┤
│  ▶ [✓] src/auth/login.ts                      +12  -1   (204t)
│
│     [✓] src/auth/middleware.ts               +29  -1   (255t)
│
│     [✓] tests/auth.test.ts                   +28  -3   (334t)
│
│
│
│
───────────────────────────────────────────────────────────────────────────┤
│  [↑↓/jk] Navigate  [Space] Toggle  [G] Generate  [Q] Quit
│
│
│
╰──────────────────────────────────────────────────────────────────────────────
```

Components Explained

Header Section

The header displays the application title in bold, primary-colored text, centered for visual prominence. This immediately identifies the tool to the user.

Branch Information

The branch line shows the source branch (where changes are) and target branch (where PR will merge to) in a clear `source → target` format. This uses success-colored text (green) to indicate active branches.

Example: `feature/user-auth → master`

Statistics Bar

The stats line provides quick metrics about the current session:

- **Files**: Total number of visible files (after filters are applied)
- **Tokens**: Total token count for all included files (important for LLM context limits)

Example: `Files: 3 | Tokens: 793`

The token count is displayed in warning color (yellow) to draw attention to context window usage.

File List Section

The main content area displays all changed files in a scrollable list. Each file entry shows:

1. **Selection Indicator**: `▶` symbol marks the currently selected file
2. **Inclusion Checkbox**:
    - `[✓]` = File is included in PR description generation
    - `[ ]` = File is excluded from generation
3. **File Path**: Full relative path from repository root
4. **Change Statistics**:
    - `+N` = Lines added (green)

    - `-N` = Lines deleted (red)
    - `(Nt)` = Token count for this file (yellow)

Visual States:

- **Selected file**: Highlighted with primary color (blue), bold text, and `▶` indicator
- **Unselected files**: Normal text with indentation for alignment

Help Bar

The bottom help bar shows available keyboard shortcuts in a concise format:

- Key names in bold primary color
- Action descriptions in muted text
- Separated by spacing for readability

Functionality

Navigation

Users can move through the file list using two sets of keybindings:

- **Arrow keys**: `↑` and `↓` for up/down movement
- **Vim keys**: `j` (down) and `k` (up) for keyboard-centric users

The selection wraps at boundaries (stays at first/last item when reaching edges).

File Toggling

Pressing `Space` on a selected file toggles its inclusion status:

- Included files (`[✓]`) will be sent to the LLM for description generation
- Excluded files (`[ ]`) are ignored during generation
- The session is automatically saved after each toggle

This allows fine-grained control over which changes appear in the PR description.

Generation Trigger

Pressing `G` initiates the PR description generation process:

- Collects all included files
- Applies active filters
- Adds any context items
- Sends to the API with the configured prompt
- Transitions to the generating screen

Exit

Pressing `Q` or `Ctrl+C` exits the TUI and returns to the shell.

State Management

The main screen model maintains:

- `selectedIndex`: Currently highlighted file (0-based index)
- `controller`: Reference to core controller with session data
- `width`, `height`: Terminal dimensions for responsive rendering

When the user navigates or toggles files, the Update function:

1. Receives the key press message
2. Updates the appropriate state field
3. Optionally triggers a command (like auto-save)
4. Returns the new model

The View function then re-renders based on the new state.

Screen 2: File Navigation States

The file list supports multiple selection states to show user interaction feedback.

State 1: First File Selected

```
╭──────────────────────────────────────────────────────────────────────
│
│
│                    PR DESCRIPTION GENERATOR
│
│
│
│  feature/user-auth → master
│
│
│
│  Files: 3 | Tokens: 793
│
│
│
│  CHANGED FILES
│
│
├──────────────────────────────────────────────────────────────────┤
│  ▶ [✓] src/auth/login.ts                  +12  -1   (204t)
│
│     [✓] src/auth/middleware.ts            +29  -1   (255t)
│
│     [✓] tests/auth.test.ts                +28  -3   (334t)
│
│
│
│
├──────────────────────────────────────────────────────────────────┤
│  [↑↓/jk] Navigate  [Space] Toggle  [G] Generate  [Q] Quit
│
│
│
╰──────────────────────────────────────────────────────────────────────
```

Behavior: The first file (`src/auth/login.ts`) is selected, indicated by the `▶` marker. This is the initial state when the TUI launches.

State 2: Middle File Selected

```
╭──────────────────────────────────────────────────────────────────────────
│
│
│                       PR DESCRIPTION GENERATOR
│
│
│
│  feature/user-auth → master
│
│
│
│  Files: 3 | Tokens: 793
│
│
│
│  CHANGED FILES
│
│
├──────────────────────────────────────────────────────────────────────┤
│      [✓] src/auth/login.ts                    +12  -1   (204t)
│
│  ▶ [✓] src/auth/middleware.ts                 +29  -1   (255t)
│
│      [✓] tests/auth.test.ts                   +28  -3   (334t)
│
│
│
│
├──────────────────────────────────────────────────────────────────────┤
│  [↑↓/jk] Navigate  [Space] Toggle  [G] Generate  [Q] Quit
│
│
│
╰──────────────────────────────────────────────────────────────────────────
```

Behavior: User pressed `↓` or `j` to move to the second file. The `▶` marker moves down, and the previously selected file returns to normal styling.

State 3: Last File Selected

```
╭──────────────────────────────────────────────────────────────────────────
│
│
│                      PR DESCRIPTION GENERATOR
│
│
│
│ feature/user-auth → master
│
│
│
│ Files: 3 | Tokens: 793
│
│
│
│ CHANGED FILES
│
│
──────────────────────────────────────────────────────────────────────────│
│     [✓] src/auth/login.ts                      +12  -1   (204t)
│
│     [✓] src/auth/middleware.ts                 +29  -1   (255t)
│
│ ▶ [✓] tests/auth.test.ts                       +28  -3   (334t)
│
│
│
│
──────────────────────────────────────────────────────────────────────────│
│ [↑↓/jk] Navigate  [Space] Toggle  [G] Generate  [Q] Quit
│
│
│
╰──────────────────────────────────────────────────────────────────────────
```

Behavior: User continued pressing `↓` or `j` to reach the last file. Further down presses have no effect (selection stays at last item).

Screen 3: File Toggle States

Users can toggle file inclusion to control which files are included in the PR description generation.

State 1: All Files Included

```
╭──────────────────────────────────────────────────────────────────────────
│
│
│                        PR DESCRIPTION GENERATOR
│
│
│
│  feature/user-auth → master
│
│
│
│  Files: 3 | Tokens: 793
│
│
│
│  CHANGED FILES
│
│
├──────────────────────────────────────────────────────────────────────────┤
│  ▶ [✓] src/auth/login.ts                      +12  -1   (204t)
│
│     [✓] src/auth/middleware.ts               +29  -1   (255t)
│
│     [✓] tests/auth.test.ts                   +28  -3   (334t)
│
│
│
├──────────────────────────────────────────────────────────────────────────┤
│  [↑↓/jk] Navigate  [Space] Toggle  [G] Generate  [Q] Quit
│
│
│
╰──────────────────────────────────────────────────────────────────────────
```

State: Initial state with all files marked with `[✓]`, indicating they will all be included in generation.

State 2: First File Excluded

```
╭──────────────────────────────────────────────────────────────────────────────
│
│
│                      PR DESCRIPTION GENERATOR
│
│
│
│  feature/user-auth → master
│
│
│
│  Files: 3 | Tokens: 589
│
│
│
│  CHANGED FILES
│
│
├──────────────────────────────────────────────────────────────────────────┤
│  ▶ [ ] src/auth/login.ts                             +12  -1   (204t)
│
│     [✓] src/auth/middleware.ts                      +29  -1   (255t)
│
│     [✓] tests/auth.test.ts                          +28  -3   (334t)
│
│
│
│
├──────────────────────────────────────────────────────────────────────────┤
│  [↑↓/jk] Navigate  [Space] Toggle  [G] Generate  [Q] Quit
│
│
│
╰──────────────────────────────────────────────────────────────────────────────
```

Action: User pressed `Space` on the first file.

Changes:

- Checkbox changed from `[✓]` to `[ ]`
- Token count decreased from 793 to 589 (204 tokens removed)
- Session automatically saved to `.pr-builder/session.yaml`

State 3: Multiple Files Excluded

```
╭──────────────────────────────────────────────────────────────────────────────
│
│
│                       PR DESCRIPTION GENERATOR
│
│
│
│  feature/user-auth → master
│
│
│
│  Files: 3 | Tokens: 334
│
│
│
│  CHANGED FILES
│
│
├───────────────────────────────────────────────────────────────────────────┤
│      [ ] src/auth/login.ts                    +12  -1   (204t)
│
│      [ ] src/auth/middleware.ts               +29  -1   (255t)
│
│  ▶ [✓] tests/auth.test.ts                      +28  -3   (334t)
│
│
│
│
├───────────────────────────────────────────────────────────────────────────┤
│  [↑↓/jk] Navigate  [Space] Toggle  [G] Generate  [Q] Quit
│
│
│
╰──────────────────────────────────────────────────────────────────────────────
```

Action: User navigated to second file and pressed `Space` again.

Changes:

- Two files now excluded: `[ ]`
- Only one file included: `[✓]`
- Token count further reduced to 334
- Only the test file will be included in generation

State 4: Re-including Files

```
╭──────────────────────────────────────────────────────────────────────────
│
│
│                       PR DESCRIPTION GENERATOR
│
│
│
│  feature/user-auth → master
│
│
│
│  Files: 3 | Tokens: 589
│
│
│
│  CHANGED FILES
│
│
──────────────────────────────────────────────────────────────────────────│
│  ▶ [ ] src/auth/login.ts                         +12  -1   (204t)
│
│      [✓] src/auth/middleware.ts                  +29  -1   (255t)
│
│      [✓] tests/auth.test.ts                      +28  -3   (334t)
│
│
│
│
──────────────────────────────────────────────────────────────────────────│
│  [↑↓/jk] Navigate  [Space] Toggle  [G] Generate  [Q] Quit
│
│
│
╰──────────────────────────────────────────────────────────────────────────
```

Action: User navigated back to second file and pressed `Space` to re-include it.

Changes:

- Second file toggled back to `[✓]`
- Token count increased to 589

- Toggle is bidirectional - pressing `Space` again would exclude it

Implementation Detail: The toggle operation calls `controller.ToggleFileInclusion(index)` which flips the boolean flag in the domain model, then automatically saves the session to persist the change.

Screen 4: Generating Screen

When the user presses `G` to generate a PR description, the TUI transitions to a loading screen that provides feedback during the asynchronous operation.

Visual Layout

```
╭────────────────────────────────────────────────────────────────────────────
│
│
│                      GENERATING PR DESCRIPTION
│
│
│
│  Analyzing changes and generating description...
│
│
│
│                                 ⠋
│
│
│
│  This may take a few seconds...
│
│
│
╰────────────────────────────────────────────────────────────────────────────
```

Components Explained

Title

The title changes from “PR DESCRIPTION GENERATOR” to “GENERATING PR DESCRIPTION” to indicate the active process. This provides immediate visual feedback that the action was triggered.

Status Message

The main message “Analyzing changes and generating description…” informs the user about what’s happening. This is important because the operation may take several seconds depending on API response time.

Loading Spinner

The centered spinner character (`⠋`) provides animated feedback. In a real terminal, this would cycle through different Braille characters to create a spinning animation:

- `⠋ ⠙ ⠹ ⠸ ⠼ ⠴ ⠦ ⠧ ⠇ ⠏`

The spinner is rendered in success color (green) to indicate active processing.

Wait Message

The bottom message “This may take a few seconds…” sets expectations about timing and reassures the user that the application hasn’t frozen.

Functionality

Async Operation

The generation process is implemented as a Bubbletea **Command**, which is a function that returns a message:

```
func (m *Model) generateCmd() tea.Cmd {
    return func() tea.Msg {
        description, err := m.controller.GenerateDescription()
        return generateCompleteMsg{
            description: description,
            err:         err,
        }
    }
}
```

This runs in a goroutine, allowing the UI to remain responsive (though we ignore input during generation).

State Management

When generation starts:

1. User presses `G` in main screen
2. Update function sets `m.generating = true`
3. Update returns the `generateCmd()` command
4. View function checks `m.generating` and renders loading screen
5. Bubbletea executes the command asynchronously
6. When complete, command returns `generateCompleteMsg`
7. Update receives the message and transitions to result screen

Input Blocking

During generation, all keyboard input is ignored except for the completion message. This prevents users from accidentally triggering multiple generations or navigating away during processing.

Technical Implementation

The loading screen demonstrates several Bubbletea patterns:

State Flag Pattern: Using a boolean flag (`generating`) to control which view is rendered.

Command Pattern: Async operations return Commands that eventually send messages back to Update.

Message Pattern: Custom message types (`generateCompleteMsg`) carry data from async operations.

Centered Layout: Using `lipgloss.PlaceHorizontal()` to center the spinner for visual appeal.

Screen 5: Result Screen

After generation completes successfully, the TUI displays the generated PR description in a formatted view.

Visual Layout

```
╭──────────────────────────────────────────────────────────────────────────────
│
│
│                    GENERATED PR DESCRIPTION
│
│
│
│  ✓ Description generated successfully!
│
│
│
│
┌──────────────────────────────────────────────────────────────────────────┐ │
│  │ # Pull Request: feature/user-auth → master
│ │
│  │
│ │
│  │ ## Summary
│ │
│  │ This PR includes changes across 3 files, implementing new features
│ │
│  │ and improvements.
│ │
│  │
│ │
│  │ ## Changes
│ │
│  │ - **src/auth/login.ts**: +12 lines, -1 lines
│ │
│  │ - **src/auth/middleware.ts**: +29 lines, -1 lines
│ │
│  │ - **tests/auth.test.ts**: +28 lines, -3 lines
│ │
│  │
│ │
│  │ ## Key Changes
│ │
│  │ - Updated login.ts with new functionality
│ │
│  │ - Improved middleware with better error handling
│ │
│  │ - Enhanced authentication system with improved security measures
│ │
│  │
│ │
│  │ ## Testing
│ │
│  │ - All existing tests pass
│ │
│  │ - New tests added for changed functionality
│ │
│  │ - Manual testing completed for critical paths
│ │
│  │
│ │
│  │ ## Breaking Changes
│ │
│  │ None
│ │
│
└─────────────────────────────────────────────────────────────────────────┘ │
│
│
│  [Esc] Back  [Q] Quit
│
│
│
╰─────────────────────────────────────────────────────────────────────────────
```

Components Explained

Success Header

The title changes to “GENERATED PR DESCRIPTION” and includes a success indicator (`✓`) with the message “Description generated successfully!” in green. This provides clear positive feedback that the operation completed.

Description Box

The generated description is displayed in a bordered box using `lipgloss.NewStyle().Border(lipgloss.NormalBorder())`. This visually separates the content from the UI chrome and makes it clear what was generated.

The box has:

- Normal border style (single-line characters)
- Border color matching the theme (dark gray)
- Padding for readability
- Fixed width to prevent line wrapping issues

Markdown Content

The description itself is formatted as Markdown, which is the standard format for PR descriptions on platforms like GitHub. The structure typically includes:

Title: `# Pull Request: source → target`

- Clearly identifies the branches involved
- Uses Markdown heading syntax

Summary Section: `## Summary`

- High-level overview of the changes
- Typically 1-3 sentences

Changes Section: `## Changes`

- Bulleted list of modified files
- Includes line addition/deletion counts
- Uses bold formatting for file names

Key Changes Section: `## Key Changes`

- Detailed description of what was modified
- Focuses on the "what" and "why"
- Bulleted for easy scanning

Testing Section: `## Testing`

- How the changes were validated
- Test coverage information
- Manual testing notes

Breaking Changes Section: `## Breaking Changes`

- Lists any API or behavior changes
- Critical for semantic versioning
- “None” if no breaking changes

Navigation Help

The help bar shows simplified options:

- `[Esc]` - Return to main screen to make adjustments
- `[Q]` - Quit the application

Functionality

Result Display

When the `generateCompleteMsg` is received:

1. Update checks if there was an error
2. If successful, sets `m.generated = true` and stores description
3. View renders the result screen
4. Description is wrapped in a styled box

Navigation Back

Pressing `Esc` returns to the main screen:

1. Sets `m.generated = false`
2. Clears the generated description
3. Returns to file list view
4. User can adjust files and regenerate

Copy Workflow

While the TUI doesn’t provide built-in copy functionality, users can:

1. Note the description

2. Exit the TUI
3. Run `pr-builder generate -o pr.md` to save to file
4. Or use the session file to regenerate via CLI

This demonstrates the power of the dual-interface design - users can switch between TUI and CLI as needed.

Error Handling

If generation fails, the screen would show:

```
╭──────────────────────────────────────────────────────────────────────────
│
│
│                         GENERATED PR DESCRIPTION
│
│
│
│  ✗ Error generating description
│
│
│
│
┌──────────────────────────────────────────────────────────────────────┐ │
│  │ Error: API request failed: connection timeout
│ │
│  │
│ │
│  │ Please check your connection and try again.
│ │
│
└──────────────────────────────────────────────────────────────────────┘ │
│
│
│  [Esc] Back  [Q] Quit
│
│
│
╰──────────────────────────────────────────────────────────────────────────
```

The error is displayed in red with a `✗` indicator, and the error message is shown in the box instead of the description.

Screen 6: Error States

The TUI handles various error conditions gracefully with clear visual feedback.

Error State 1: Initialization Error

```
╭──────────────────────────────────────────────────────────────────────────────
│
│
│  ✗ Error: failed to initialize: not a git repository
│
│
│
│  Press Q to quit
│
│
│
╰──────────────────────────────────────────────────────────────────────────────
```

Trigger: Running the TUI outside a git repository or with invalid path.

Display: Simple error message in red with clear instruction to exit.

Error State 2: Session Load Error

```
╭──────────────────────────────────────────────────────────────────────────
│
│
│  ✗ Error: failed to load session: invalid YAML format
│
│
│
│  Press Q to quit
│
│
│
╰──────────────────────────────────────────────────────────────────────────
```

Trigger: Corrupted session file in `.pr-builder/session.yaml`.

Display: Error message explaining the issue. User can exit, fix the file, and restart.

Error State 3: Generation Error

As shown in the previous section, generation errors are displayed on the result screen with the error message in a box, allowing the user to return to the main screen and try again.

Keyboard Shortcuts Reference

Main Screen

Key	Action	Description
`↑` or `k`	Navigate Up	Move selection to previous file
`↓` or `j`	Navigate Down	Move selection to next file
`Space`	Toggle File	Include/exclude selected file from generation
`g` or `G`	Generate	Start PR description generation
`q` or `Q`	Quit	Exit the TUI
`Ctrl+C`	Force Quit	Immediately exit the application

Result Screen

Key	Action	Description
`Esc`	Go Back	Return to main file list screen
`q` or `Q`	Quit	Exit the TUI
`Ctrl+C`	Force Quit	Immediately exit the application

Generating Screen

Key	Action	Description
(All input ignored)	Wait	Wait for generation to complete

Technical Architecture

Bubbletea Model Structure

The TUI model contains all necessary state:

```go
type Model struct {
    controller     *controller.Controller  // Core business logic
    width          int                     // Terminal width
    height         int                     // Terminal height
    selectedIndex  int                     // Currently selected file
    err            error                   // Error state
    generating     bool                    // Generation in progress
    generated      bool                    // Generation complete
    generatedDesc  string                  // Generated description
}
```

State Machine

The TUI operates as a state machine with three primary states:

```
┌──────────┐
│   Main   │ ◄──── Initial state
└──────────┘
     │
     │ Press G
     ▼
┌──────────┐
│Generating│ ◄──── Async operation
└──────────┘
     │
     │ Complete
     ▼
┌──────────┐
│  Result  │ ◄──── Display output
└──────────┘
     │
     │ Press Esc
     ▼
┌──────────┐
│   Main   │ ◄──── Back to start
└──────────┘
```

Message Flow

The Bubbletea message passing system handles all events:

Key Press Messages:

```
tea.KeyMsg{String: "j"} → Update → selectedIndex++
tea.KeyMsg{String: " "} → Update → ToggleFile → SaveSession
tea.KeyMsg{String: "g"} → Update → generating=true → generateCmd()
```

Async Messages:

```
generateCompleteMsg{description, err} → Update → generated=true
```

Window Messages:

```
tea.WindowSizeMsg{Width, Height} → Update → width, height
```

Styling System

The TUI uses **Lipgloss** for all styling, with a centralized style definition:

Color Palette:

- Primary (Blue): Highlights, titles, selected items
- Success (Green): Positive feedback, additions
- Error (Red): Errors, deletions
- Warning (Yellow): Token counts, cautions
- Muted (Gray): Help text, secondary info
- Border (Dark Gray): Boxes and separators

Style Components:

- `TitleStyle`: Bold, primary color, centered

- `SelectedItemStyle`: Bold, primary color, left-padded
- `UnselectedItemStyle`: Normal, indented
- `BorderStyle`: Rounded border, padded
- `BoxStyle`: Normal border, minimal padding
- `HelpStyle`: Muted color, bottom-aligned

Responsive Design

The TUI adapts to terminal size:

- Receives `tea.WindowSizeMsg` on resize
- Stores dimensions in model
- Uses dimensions for centering and wrapping
- Minimum viable size: 80x24 (standard terminal)

Integration with Core

The TUI is a thin presentation layer over the core controller:

Controller Interface

```go
type Controller struct {
    data *domain.PRData
    // ... services
}

func (c *Controller) Initialize(targetBranch string) error
func (c *Controller) GetData() *domain.PRData
func (c *Controller) ToggleFileInclusion(index int) error
func (c *Controller) GenerateDescription() (string, error)
func (c *Controller) SaveSession(path string) error
func (c *Controller) LoadSession(path string) error
```

Data Flow

```
User Input → TUI Model → Controller → Domain Model
                                    → Services (Git, API)
                                    → Session (YAML)
```

Session Synchronization

Every state-changing operation in the TUI:

1. Calls the appropriate controller method
2. Controller updates the domain model
3. Controller saves the session to disk
4. TUI re-renders with new state

This ensures the TUI and CLI stay in sync - users can exit the TUI, run CLI commands, and return to the TUI with consistent state.

Comparison: TUI vs CLI

When to Use TUI

- **Exploring changes**: Navigate and review files interactively
- **Fine-tuning**: Toggle files while seeing token counts update
- **Quick iterations**: Generate, review, adjust, regenerate
- **Learning**: Visual feedback helps understand the tool

When to Use CLI

- **Automation**: Script PR generation in CI/CD
- **Batch operations**: Process multiple repositories
- **Remote sessions**: SSH without terminal emulation
- **Integration**: Pipe output to other tools

Shared Benefits

- Same session file format
- Consistent behavior
- Interchangeable workflows
- No lock-in to one interface

Best Practices

Effective TUI Usage

Start with defaults: Launch the TUI and review the initial file selection before making changes.

Use token counts: Monitor the token count to stay within LLM context limits (typically 4K-8K tokens).

Toggle strategically: Exclude generated files, lock files, and large refactors that don't need detailed description.

Iterate quickly: Generate, review, adjust files, regenerate. The TUI makes this cycle fast.

Save sessions: Use the CLI to save custom sessions for different PR types (features, bugfixes, refactors).

Keyboard Efficiency

Learn vim keys: `j/k` navigation is faster than arrow keys for frequent users.

Use Space liberally: Toggle files on/off to experiment with different contexts.

Quick generation: Press `g` as soon as you've selected files - generation is fast.

Escape to adjust: If the description isn't quite right, press `Esc` and adjust file selection.

Future Enhancements

Planned Features

Filter Editor Screen: Interactive screen to add/edit/remove filters without CLI.

Context Editor Screen: Add files and notes as context through the TUI.

Prompt Editor Screen: Select presets or write custom prompts interactively.

Diff Viewer: Show actual code changes for selected file.

Search: Filter file list by name or path pattern.

Scrolling: Handle repositories with 100+ changed files.

Undo/Redo: Step backward/forward through session changes.

Technical Improvements

Use Bubbles Components: Replace custom list with `bubbles/list` for better UX.

Add Viewport: Use `bubbles/viewport` for scrolling long content.

Spinner Animation: Use `bubbles/spinner` for proper animated loading.

Progress Bar: Show generation progress if API supports streaming.

Help Screen: Dedicated overlay with all keyboard shortcuts.

Themes: Support light/dark themes and custom color schemes.

Conclusion

The PR Builder TUI demonstrates professional terminal UI development with Bubbletea. The clean architecture, responsive design, and thoughtful UX create a tool that's both powerful and pleasant to use.

The session-based design ensures the TUI and CLI work together seamlessly, giving users the flexibility to choose the right interface for each task. The comprehensive keyboard shortcuts and visual feedback make the TUI efficient for experienced users while remaining accessible to newcomers.

By following Bubbletea’s Elm architecture and maintaining clear separation between presentation and business logic, the TUI is maintainable, testable, and extensible. Future enhancements can be added without disrupting the core functionality.

This TUI serves as both a functional tool and a reference implementation for building professional terminal applications in Go.