System Design Document: Azure DevOps TUI App

1. Overview

This document outlines the high-level system design for a Terminal User Interface (TUI) application built in Go, leveraging the Azure DevOps (ADO) Go SDK and the Bubble Tea framework. The app will provide in-depth management of user stories (work items), tasks, pull requests (PRs), and pipeline monitoring. Key focuses include:

- **Core Functionality**: Interactive dashboard for ADO resources, with filtering and management capabilities.
- **Target Users**: Developers, project managers, and DevOps engineers who prefer terminal-based tools for efficiency.
- **Tech Stack**:
  - Language: Go (for performance and concurrency).
  - TUI Framework: Bubble Tea (for building interactive, stateful UIs with models, updates, and views).
  - ADO Integration: `azure-devops-go-api` SDK (thin wrapper over ADO REST APIs).
  - Styling: Lip Gloss (for enhanced terminal rendering, e.g., colors, borders).
- **Assumptions**: Users have an ADO Personal Access Token (PAT) for authentication. The app runs locally in a terminal supporting ANSI colors (e.g., iTerm, Windows Terminal).

The design emphasizes modularity, asynchronous data fetching to avoid blocking the UI, and keyboard-driven navigation for a seamless experience.

2. Requirements

Functional Requirements

- **Dashboard View**:
  - Display overviews of ADO organizations, projects, repos, boards, and teams.
  - Real-time summaries (e.g., open work items, active PRs, running pipelines).
- **Work Item Management (User Stories and Tasks)**:
  - List, view, create, update, and delete work items (e.g., user stories, tasks, bugs).
  - In-depth filtering: By state (e.g., New, Active, Resolved), priority, assignee, iteration, area path, tags, or custom queries using ADO's WIQL (Work Item Query Language).
  - Sorting options (e.g., by creation date, severity).
  - Detailed views: Show fields, comments, attachments, history, and relations (e.g., parent-child tasks).
- **PR Management**:
  - Browse PRs across repos: List open/closed PRs with filters (e.g., by author, reviewer, status, branch).
  - Actions: View diffs, approve/reject, add comments, merge (if permissions allow).
  - Notifications: Poll for updates on assigned PRs.
- **Pipeline Monitoring**:
  - Monitor build and release pipelines: List runs, view logs, statuses (e.g., succeeded, failed, in-progress).
  - Filtering: By pipeline name, branch, date, or outcome.
  - Real-time polling: Refresh statuses periodically without manual intervention.
  - Actions: Trigger re-runs or cancellations.

- **General Features**:
  - Search across resources (e.g., global search for work items or PRs).
  - Configuration: Store PAT, organization URL, and preferences (e.g., poll intervals) in a local config file (e.g., YAML).
  - Error Handling: Graceful retries for API failures, offline mode with cached data.
  - Help System: Built-in keybinding cheatsheet.

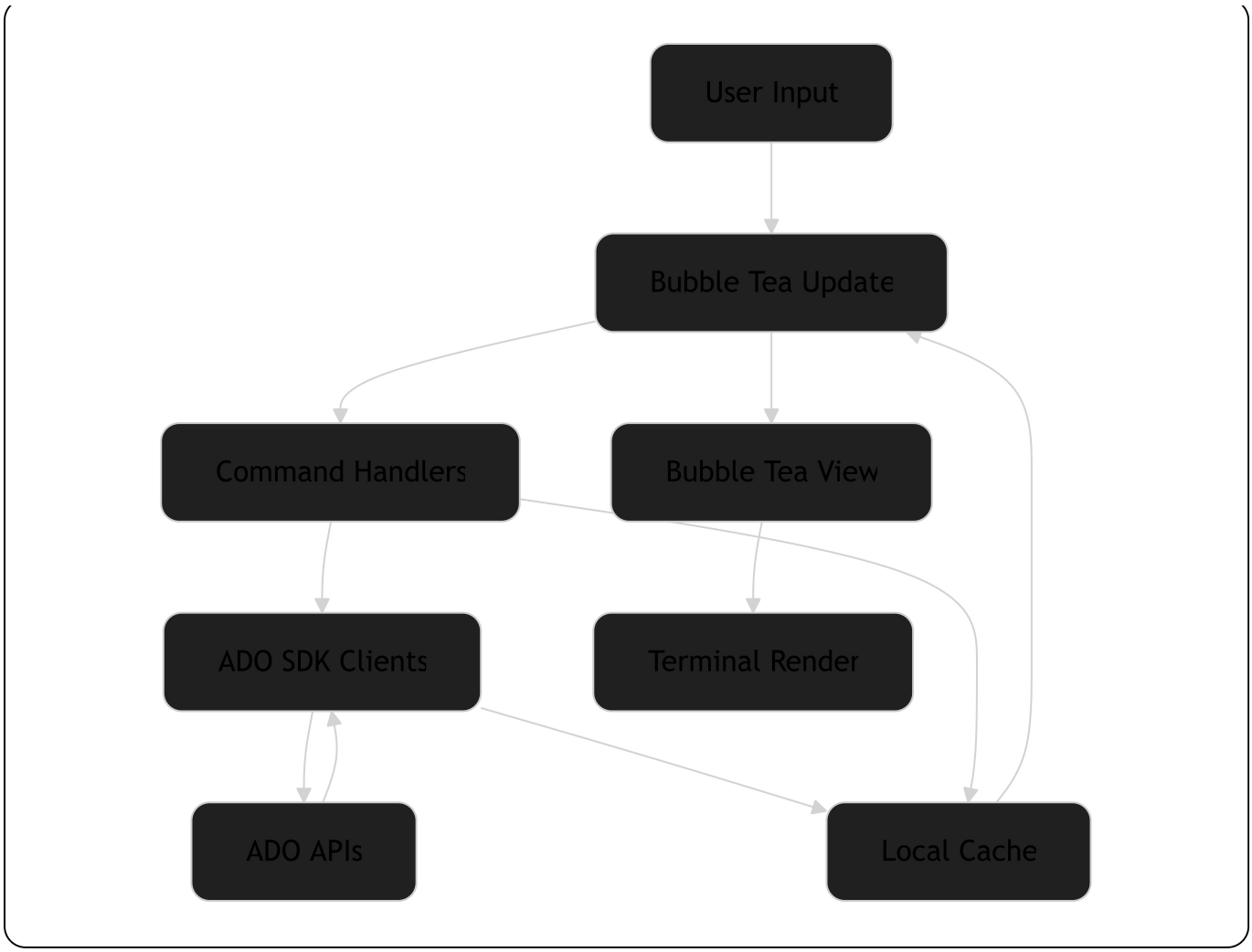
Non-Functional Requirements

- **Performance**: API calls should be asynchronous; UI updates < 1s for most operations.
- **Usability**: Fully keyboard-navigable; support for mouse if terminal allows.
- **Security**: PAT stored securely (e.g., using keyring package); no plaintext storage.
- **Scalability**: Handle large ADO instances (e.g., thousands of work items) via pagination in SDK calls.
- **Extensibility**: Modular design for adding features like test plan integration.

3. High-Level Architecture

The app follows Bubble Tea's Elm-inspired architecture (Model-View-Update pattern), integrated with the ADO SDK for data operations.

- **Layers**:
  - **UI Layer**: Bubble Tea components (e.g., lists, viewports, text inputs) for rendering and user input.
  - **Business Logic Layer**: Handlers for commands (e.g., fetch data, apply filters).
  - **Data Layer**: ADO SDK clients for API interactions; local cache (e.g., using BoltDB or in-memory) for offline/faster access.
  - **Config/Storage Layer**: Persistent storage for user settings.
- **Data Flow**:
  1. User launches app → Loads config → Authenticates with ADO SDK.
  2. Initial fetch: Async SDK calls to populate dashboard data.
  3. User interacts (e.g., filters work items) → Bubble Tea sends messages → Update function processes (e.g., applies filter, re-fetches if needed) → View renders updated UI.
  4. Background tasks: Goroutines for polling (e.g., pipeline statuses every 30s).
- **Concurrency**: Use Go channels and goroutines for non-blocking API calls. Bubble Tea's `tea.Cmd` handles async operations elegantly.



4. Key Components

4.1 Models (Bubble Tea)

- **Root Model**: Manages app state (e.g., current view: dashboard, work items, PRs, pipelines).
- **Sub-Models**:
  - **DashboardModel**: Aggregates summaries from other models.
  - **WorkItemModel**: Handles listing/filtering with a table or list component. State includes filters (map[string]string), paginated results.
  - **PRModel**: Similar to WorkItemModel, with actions like approve (calls SDK's `pullrequest.UpdatePullRequest`).
  - **PipelineModel**: Uses progress bars/spinners for statuses; timer for polling.

4.2 SDK Integration

- Initialize client: `connection := azuredevops.NewPatConnection(orgURL, pat)`
- Clients: Use specific clients like `workitemtracking.NewClient(connection)` for work items, `git.NewClient` for PRs/repos, `build.NewClient` for pipelines.
- Filtering: For work items, use `workitemtracking.GetWorkItemsWithQuery` with WIQL for complex filters.
- Pagination: Handle with SDK's built-in support (e.g., top/skip parameters).
- Async Wrapper: Wrap SDK calls in goroutines, returning `tea.Cmd` for Bubble Tea integration.

4.3 UI Elements (Bubble Tea + Lip Gloss)

- **Layouts**: Tabbed or viewport-based navigation (e.g., left sidebar for menu, main area for content).
- **Components**:
  - Lists/Tables: For displaying filtered work items/PRs (e.g., bubbletable or custom list).
  - Forms: For creating/updating items (textinput, textarea).
  - Progress: For pipeline monitoring (bubbles/progress).
  - Filtering UI: Modal with inputs for criteria; apply dynamically without full refresh.
- **Keybindings**: Global (e.g., Ctrl+C quit, / search) and context-specific (e.g., a approve PR).

4.4 Data Models

- Map ADO entities to Go structs (SDK provides these, e.g., `workitemtracking.WorkItem`).
- Custom: `FilteredList` struct with items, current page, total count.
- Cache: Key-value store (e.g., project ID → serialized data) with TTL for freshness.

5. Implementation Plan

- **Phase 1**: Setup skeleton app with auth and dashboard.
- **Phase 2**: Implement work item management with filtering.
- **Phase 3**: Add PR and pipeline features.
- **Phase 4**: Polish UI, add polling, error handling.
- **Testing**: Unit tests for handlers; integration tests with mock SDK (e.g., using httptest for API responses).
- **Dependencies**: `go get` for Bubble Tea, Lip Gloss, ADO SDK, and extras like `github.com/charmbracelet/bubbles` for components.

6. Potential Challenges and Mitigations

- **API Rate Limits**: Implement exponential backoff in retries.
- **Large Data Sets**: Use lazy loading and virtualization in lists.
- **Terminal Compatibility**: Test on multiple terminals; fallback to basic rendering.
- **Security**: Encrypt PAT; prompt for it if not stored.
- **Extensibility**: Use interfaces for models to allow plugins.

This design provides a solid foundation for a powerful, terminal-native ADO tool. If you'd like to dive deeper into code snippets, specific diagrams, or iterate on features, let me know!