

# Arborist UI

CAD Application Interface

TypeScript Implementation Plan

Trunk → Tree → Branch → Leaf Architecture

Design System & Implementation Reference

January 2026  
Version 1.0

## Abstract

This document provides a complete TypeScript implementation plan for the Arborist UI system—a CAD application interface using tree-themed nomenclature that aligns with Git version control concepts. The architecture organizes work into Trunks (workspaces), Trees (repositories), Branches (versions), and Leaves (design files). This plan covers core types, state management, UI components, modal systems, plugin architecture, and a phased implementation timeline.

## Contents

<b>I Conceptual Foundation</b>	<b>3</b>
<b>1 Tree-Themed Architecture</b>	<b>3</b>
1.1 Naming Convention & Git Alignment . . . . .	3
1.2 Extended Vocabulary . . . . .	3
1.3 Visual Hierarchy . . . . .	3
<b>2 Application Layout</b>	<b>3</b>
2.1 UI Structure . . . . .	4
<b>II Type System</b>	<b>4</b>
<b>3 Core Domain Types</b>	<b>4</b>
3.1 Arborist Types . . . . .	4
3.2 Layer Types . . . . .	6
<b>4 Application State</b>	<b>8</b>
4.1 State Shape . . . . .	8
4.2 Action Types . . . . .	11

<b>III State Management</b>	<b>12</b>
<b>5 Context Architecture</b>	<b>12</b>
5.1 App Context . . . . .	12
5.2 Specialized Hooks . . . . .	13
 <b>IV UI Components</b>	 <b>20</b>
<b>6 Ribbon Component</b>	<b>20</b>
<b>7 Sidebar Components</b>	<b>24</b>
7.1 Trunk Selector . . . . .	24
7.2 Tree & Branch Selector . . . . .	27
7.3 Layers Panel . . . . .	30
<b>8 Modal Components</b>	<b>32</b>
8.1 Base Modal . . . . .	32
 <b>V Plugin System</b>	 <b>35</b>
<b>9 Plugin Architecture</b>	<b>35</b>
9.1 Plugin Types . . . . .	35
9.2 Plugin Context . . . . .	37
 <b>VI Implementation Timeline</b>	 <b>42</b>
<b>10 Phased Development Plan</b>	<b>42</b>
<b>11 File Structure</b>	<b>44</b>
<b>12 Summary</b>	<b>45</b>

# Part I

## Conceptual Foundation

### 1 Tree-Themed Architecture

#### 1.1 Naming Convention & Git Alignment

Arborist	Git Equivalent	Icon	Description
Trunk	Workspace	x	Root container for all Trees
Tree	Repository	x	Project folder with version control
Branch	Git Branch	x	Version/variant of a Tree
Leaf	File (blob)	x	Individual design document

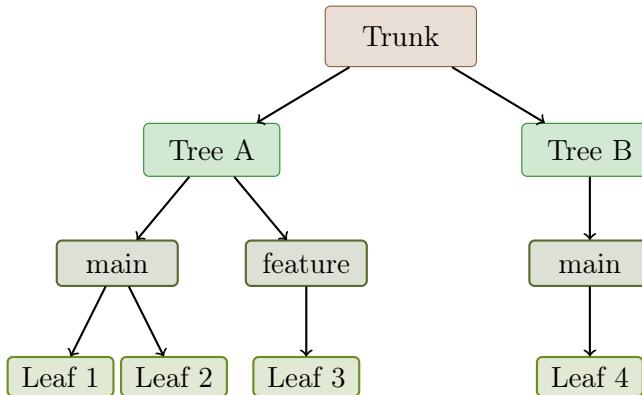
Table 1: Core Naming Convention

#### 1.2 Extended Vocabulary

Term	Use	Description
Root	Config	Project settings, manifest files
Seed	Template	Starter project template
Sapling	New Project	Empty or newly created Tree
Graft	Merge	Combining Branches
Prune	Delete	Removing old Branches
Ring	History	Version history (like tree rings)
Bark	Metadata	Project info, thumbnails
Sap	Sync	Data flow between devices
Canopy	Export	Published/exported view
Grove	Collection	Group of related Trees

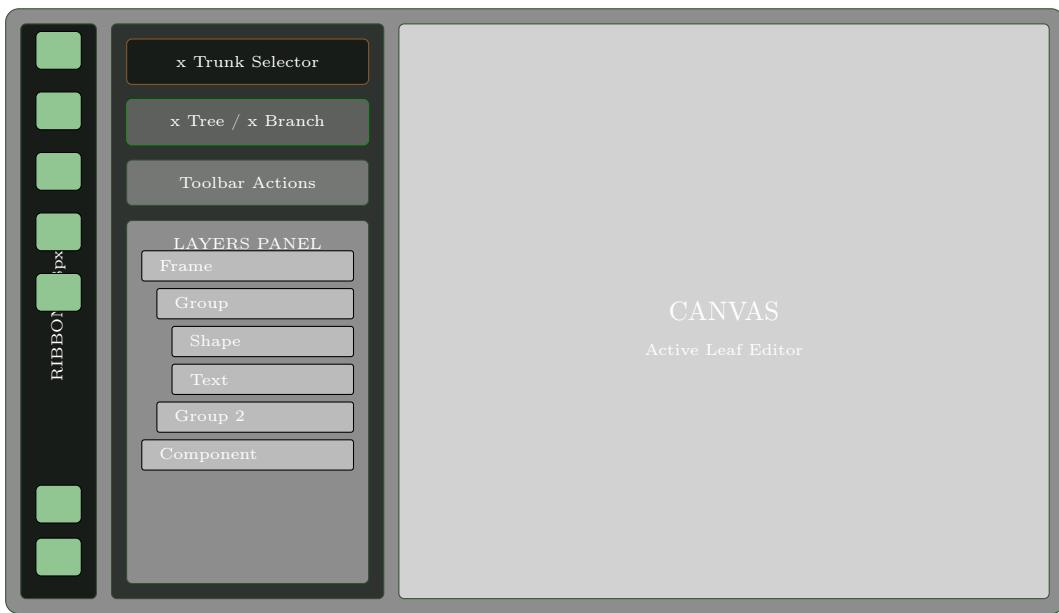
Table 2: Extended Vocabulary

#### 1.3 Visual Hierarchy



### 2 Application Layout

## 2.1 UI Structure



1. Ribbon

2. Sidebar

3. Canvas

1. **Ribbon** (48px) — Navigation icons, plugin actions, settings, help
2. **Sidebar** (240–400px) — Trunk/Tree/Branch selectors, toolbar, layers panel
3. **Canvas** (flex) — Active Leaf editor, design surface

## Part II

# Type System

### 3 Core Domain Types

#### 3.1 Arborist Types

```

1 // src/types/arborist.ts
2
3 /**
4  * Trunk - Workspace containing multiple Trees
5  * Analogous to a workspace or collection of repositories
6 */
7 export interface Trunk {
8   id: string;
9   name: string;
10  trees: Tree[];
11  settings: TrunkSettings;
12  createdAt: number;
13  lastOpenedAt: number;
14 }
15
16 export interface TrunkSettings {
17   theme: 'dark' | 'light' | 'system';

```

```
18     accentColor: string;
19     sidebarWidth: number;
20     autoSave: boolean;
21     autoSaveInterval: number;
22     syncEnabled: boolean;
23 }
24
25 /**
26 * Tree - Repository/Project containing Branches
27 * Represents a complete project with version history
28 */
29 export interface Tree {
30     id: string;
31     name: string;
32     path: string; // Local path or remote URL
33     branches: Branch[];
34     currentBranchId: string;
35     defaultBranchId: string; // Usually 'main'
36     remotes: GitRemote[];
37     bark: TreeBark; // Metadata
38     isCloudSync: boolean;
39     createdAt: number;
40     lastModifiedAt: number;
41 }
42
43 export interface TreeBark {
44     description: string;
45     thumbnail?: string;
46     tags: string[];
47     author: string;
48     license?: string;
49 }
50
51 export interface GitRemote {
52     name: string; // e.g., "origin"
53     url: string;
54     type: 'github' | 'gitlab' | 'bitbucket' | 'self-hosted';
55 }
56
57 /**
58 * Branch - Version/variant of a Tree
59 * Direct mapping to Git branch concept
60 */
61 export interface Branch {
62     id: string;
63     name: string; // e.g., "main", "feature/dark-mode"
64     gitRef: string; // Git reference
65     leaves: Leaf[];
66     parentBranchId: string | null; // For branch visualization
67     lastCommit: Commit | null;
68     isProtected: boolean;
69     createdAt: number;
70     lastModifiedAt: number;
71 }
72
73 export interface Commit {
74     hash: string;
```

```

75     shortHash: string;
76     message: string;
77     author: string;
78     email: string;
79     timestamp: number;
80     parentHashes: string[];
81   }
82
83   /**
84    * Leaf - Individual design document
85    * The atomic unit of work, containing layers
86   */
87   export interface Leaf {
88     id: string;
89     name: string;
90     type: LeafType;
91     layers: Layer[];
92     canvas: CanvasSettings;
93     thumbnail?: string;
94     createdAt: number;
95     lastModifiedAt: number;
96   }
97
98   export type LeafType =
99     | 'design' // Standard design document
100    | 'component' // Reusable component
101    | 'asset' // Static asset
102    | 'prototype'; // Interactive prototype
103
104  export interface CanvasSettings {
105    width: number;
106    height: number;
107    backgroundColor: string;
108    gridEnabled: boolean;
109    gridSize: number;
110    snapToGrid: boolean;
111    rulerEnabled: boolean;
112  }

```

## 3.2 Layer Types

```

1 // src/types/layers.ts
2
3 /**
4  * Layer - Element within a Leaf
5  * Hierarchical structure for design elements
6 */
7 export interface Layer {
8   id: string;
9   name: string;
10  type: LayerType;
11  visible: boolean;
12  locked: boolean;
13  opacity: number;
14  blendMode: BlendMode;
15  parentId: string | null;

```

```
16     childIds: string[];  
17     order: number; // Sort order within parent  
18     transform: Transform;  
19     constraints: Constraints;  
20   }  
21  
22 export type LayerType =  
23   | 'frame'  
24   | 'group'  
25   | 'rectangle'  
26   | 'ellipse'  
27   | 'polygon'  
28   | 'path'  
29   | 'text'  
30   | 'image'  
31   | 'component'  
32   | 'instance';  
33  
34 export type BlendMode =  
35   | 'normal'  
36   | 'multiply'  
37   | 'screen'  
38   | 'overlay'  
39   | 'darken'  
40   | 'lighten';  
41  
42 export interface Transform {  
43   x: number;  
44   y: number;  
45   width: number;  
46   height: number;  
47   rotation: number;  
48   scaleX: number;  
49   scaleY: number;  
50 }  
51  
52 export interface Constraints {  
53   horizontal: 'left' | 'right' | 'center' | 'scale' | 'left-right';  
54   vertical: 'top' | 'bottom' | 'center' | 'scale' | 'top-bottom';  
55 }  
56 // Type-specific properties  
57 export interface FrameLayer extends Layer {  
58   type: 'frame';  
59   fill: Fill[];  
60   stroke: Stroke[];  
61   cornerRadius: number | [number, number, number, number];  
62   clipContent: boolean;  
63   layoutMode: 'none' | 'horizontal' | 'vertical';  
64   layoutProps?: AutoLayoutProps;  
65 }  
66  
67  
68 export interface TextLayer extends Layer {  
69   type: 'text';  
70   content: string;  
71   fontFamily: string;  
72   fontSize: number;  
73   fontWeight: number;
```

```

74   lineHeight: number | 'auto';
75   letterSpacing: number;
76   textAlign: 'left' | 'center' | 'right' | 'justify';
77   fill: Fill[];
78 }
79
80 export interface ImageLayer extends Layer {
81   type: 'image';
82   src: string;
83   fit: 'fill' | 'fit' | 'crop' | 'tile';
84 }
85
86 export interface Fill {
87   type: 'solid' | 'gradient' | 'image';
88   color?: string;
89   opacity?: number;
90   gradient?: Gradient;
91 }
92
93 export interface Stroke {
94   color: string;
95   width: number;
96   position: 'inside' | 'center' | 'outside';
97   dashPattern?: number[];
98 }
99
100 export interface Gradient {
101   type: 'linear' | 'radial' | 'angular';
102   stops: GradientStop[];
103   angle?: number;
104 }
105
106 export interface GradientStop {
107   position: number;
108   color: string;
109 }
110
111 export interface AutoLayoutProps {
112   direction: 'horizontal' | 'vertical';
113   gap: number;
114   paddingTop: number;
115   paddingRight: number;
116   paddingBottom: number;
117   paddingLeft: number;
118   alignItems: 'start' | 'center' | 'end' | 'stretch';
119   justifyContent: 'start' | 'center' | 'end' | 'space-between';
120 }

```

## 4 Application State

### 4.1 State Shape

```

1 // src/types/state.ts
2
3 export interface AppState {

```

```
4   // Navigation state
5   navigation: NavigationState;
6
7   // UI state
8   ui: UIState;
9
10  // Editor state (current Leaf)
11  editor: EditorState | null;
12
13  // Plugin state
14  plugins: PluginsState;
15
16  // User state
17  user: UserState;
18 }
19
20 export interface NavigationState {
21   // Current location in the tree
22   currentTrunkId: string | null;
23   currentTreeId: string | null;
24   currentBranchId: string | null;
25   currentLeafId: string | null;
26
27   // Data
28   trunks: Trunk[];
29   recentTreeIds: string[];
30   recentLeafIds: string[];
31 }
32
33 export interface UIState {
34   // Sidebar
35   sidebarOpen: boolean;
36   sidebarWidth: number;
37   activePanel: 'layers' | 'assets' | 'components' | 'history';
38
39   // Modals
40   modals: {
41     settings: boolean;
42     help: boolean;
43     payment: boolean;
44     trunkManager: boolean;
45     branchManager: boolean;
46     exportDialog: boolean;
47   };
48
49   // Theme
50   theme: 'dark' | 'light' | 'system';
51   resolvedTheme: 'dark' | 'light';
52   accentColor: string;
53
54   // Notifications
55   notifications: Notification[];
56 }
57
58 export interface EditorState {
59   // Current Leaf data
60   leaf: Leaf;
61 }
```

```
62      // Selection
63      selectedLayerIds: string[];
64      hoveredLayerId: string | null;
65
66      // Layers panel
67      expandedLayerIds: string[];
68
69      // Viewport
70      viewport: {
71          x: number;
72          y: number;
73          zoom: number;
74      };
75
76      // Tool state
77      activeTool: ToolType;
78      toolOptions: Record<string, unknown>;
79
80      // History
81      canUndo: boolean;
82      canRedo: boolean;
83  }
84
85  export type ToolType =
86  | 'select'
87  | 'frame'
88  | 'rectangle'
89  | 'ellipse'
90  | 'polygon'
91  | 'pen'
92  | 'text'
93  | 'hand'
94  | 'zoom';
95
96  export interface PluginsState {
97      installed: PluginManifest[];
98      enabled: string[];
99      settings: Record<string, unknown>;
100 }
101
102 export interface UserState {
103     isAuthenticated: boolean;
104     profile: UserProfile | null;
105     subscription: SubscriptionTier;
106     syncStatus: 'idle' | 'syncing' | 'error';
107 }
108
109 export interface UserProfile {
110     id: string;
111     email: string;
112     name: string;
113     avatarUrl?: string;
114 }
115
116 export type SubscriptionTier = 'free' | 'pro' | 'team' | 'enterprise'
117 ;
118 export interface Notification {
```

```

119     id: string;
120     type: 'info' | 'success' | 'warning' | 'error';
121     message: string;
122     duration?: number;
123     dismissible: boolean;
124 }

```

## 4.2 Action Types

```

1 // src/types/actions.ts
2
3 export type AppAction =
4   // Navigation actions
5   | { type: 'NAV_SET_TRUNK'; payload: string }
6   | { type: 'NAV_SET_TREE'; payload: string }
7   | { type: 'NAV_SET_BRANCH'; payload: string }
8   | { type: 'NAV_SET_LEAF'; payload: string }
9   | { type: 'NAV_LOAD_TRUNKS'; payload: Trunk[] }
10  | { type: 'NAV_CREATE_TRUNK'; payload: { name: string } }
11  | { type: 'NAV_DELETE_TRUNK'; payload: string }
12  | { type: 'NAV_CREATE_TREE'; payload: { trunkId: string; name: string } }
13  | { type: 'NAV_DELETE_TREE'; payload: string }
14  | { type: 'NAV_CREATE_BRANCH'; payload: { treeId: string; name: string; fromBranchId: string } }
15  | { type: 'NAV_DELETE_BRANCH'; payload: string }
16  | { type: 'NAV_CREATE_LEAF'; payload: { branchId: string; name: string; type: LeafType } }
17  | { type: 'NAV_DELETE_LEAF'; payload: string }
18
19   // UI actions
20   | { type: 'UI_TOGGLE_SIDEBAR' }
21   | { type: 'UI_SET_SIDEBAR_WIDTH'; payload: number }
22   | { type: 'UI_SET_ACTIVE_PANEL'; payload: UIState['activePanel'] }
23   | { type: 'UI_OPEN_MODAL'; payload: keyof UIState['modals'] }
24   | { type: 'UI_CLOSE_MODAL'; payload: keyof UIState['modals'] }
25   | { type: 'UI_SET_THEME'; payload: 'dark' | 'light' | 'system' }
26   | { type: 'UI_ADD_NOTIFICATION'; payload: Omit<Notification, 'id'> }
27   | { type: 'UI_DISMISS_NOTIFICATION'; payload: string }
28
29   // Editor actions
30   | { type: 'EDITOR_LOAD_LEAF'; payload: Leaf }
31   | { type: 'EDITOR_UNLOAD_LEAF' }
32   | { type: 'EDITOR_SELECT_LAYERS'; payload: { ids: string[]; mode: 'replace' | 'add' | 'toggle' } }
33   | { type: 'EDITOR_CLEAR_SELECTION' }
34   | { type: 'EDITOR_TOGGLE_LAYER_EXPAND'; payload: string }
35   | { type: 'EDITOR_SET_VIEWPORT'; payload: Partial<EditorState['viewport']> }
36   | { type: 'EDITOR_SET_TOOL'; payload: ToolType }
37
38   // Layer actions
39   | { type: 'LAYER_CREATE'; payload: { type: LayerType; parentId?: string } }
40   | { type: 'LAYER_DELETE'; payload: string[] }

```

```

41 | { type: 'LAYER_UPDATE'; payload: { id: string; changes: Partial<
42 |   Layer> } }
43 | { type: 'LAYER_RENAME'; payload: { id: string; name: string } }
44 | { type: 'LAYER_REORDER'; payload: { id: string; targetId: string;
45 |   position: 'before' | 'after' | 'inside' } }
46 | { type: 'LAYER_TOGGLE_VISIBILITY'; payload: string }
47 | { type: 'LAYER_TOGGLE_LOCK'; payload: string }
48 | { type: 'LAYER_DUPLICATE'; payload: string[] }
49 | { type: 'LAYER_GROUP'; payload: string[] }
50 | { type: 'LAYER_UNGROUP'; payload: string }
51 // Plugin actions
52 | { type: 'PLUGIN_INSTALL'; payload: PluginManifest }
53 | { type: 'PLUGIN_UNINSTALL'; payload: string }
54 | { type: 'PLUGIN_ENABLE'; payload: string }
55 | { type: 'PLUGIN_DISABLE'; payload: string }
56 // User actions
57 | { type: 'USER_LOGIN'; payload: UserProfile }
58 | { type: 'USER_LOGOUT' }
59 | { type: 'USER_SET_SUBSCRIPTION'; payload: SubscriptionTier };

```

## Part III

# State Management

## 5 Context Architecture

### 5.1 App Context

```

1 // src/contexts/AppContext.tsx
2
3 import { createContext, useContext, useReducer, ReactNode, Dispatch }
4   from 'react';
5 import { AppState, AppAction } from '../types/state';
6 import { appReducer } from '../reducers/appReducer';
7 import { initialState } from '../reducers/initialState';
8
9 interface AppContextValue {
10   state: AppState;
11   dispatch: Dispatch<AppAction>;
12 }
13
14 const AppContext = createContext<AppContextValue | null>(null);
15
16 interface AppProviderProps {
17   children: ReactNode;
18 }
19
20 export function AppProvider({ children }: AppProviderProps): JSX.
21   Element {
22   const [state, dispatch] = useReducer(appReducer, initialState);
23
24   return (

```

```

23     <AppContext.Provider value={{ state, dispatch }}>
24         {children}
25     </AppContext.Provider>
26   );
27 }
28
29 export function useAppContext(): ApplicationContextValue {
30   const context = useContext(AppContext);
31   if (!context) {
32     throw new Error('useAppContext must be used within AppProvider');
33   }
34   return context;
35 }

```

## 5.2 Specialized Hooks

```

1 // src/hooks/useNavigation.ts
2
3 import { useCallback, useMemo } from 'react';
4 import { useAppContext } from '../contexts/AppContext';
5 import { Trunk, Tree, Branch, Leaf, LeafType } from '../types/
    arborist';
6
7 export interface UseNavigationReturn {
8   // Current selections
9   currentTrunk: Trunk | null;
10  currentTree: Tree | null;
11  currentBranch: Branch | null;
12  currentLeaf: Leaf | null;
13
14  // Data
15  trunks: Trunk[];
16  recentTrees: Tree[];
17
18  // Actions
19  setTrunk: (id: string) => void;
20  setTree: (id: string) => void;
21  setBranch: (id: string) => void;
22  setLeaf: (id: string) => void;
23
24  // CRUD
25  createTrunk: (name: string) => void;
26  deleteTrunk: (id: string) => void;
27  createTree: (name: string) => void;
28  deleteTree: (id: string) => void;
29  createBranch: (name: string, fromBranchId?: string) => void;
30  deleteBranch: (id: string) => void;
31  createLeaf: (name: string, type: LeafType) => void;
32  deleteLeaf: (id: string) => void;
33 }
34
35 export function useNavigation(): UseNavigationReturn {
36   const { state, dispatch } = useAppContext();
37   const { navigation } = state;
38
39   // Memoized current selections

```

```

40   const currentTrunk = useMemo(() =>
41     navigation.trunks.find(t => t.id === navigation.currentTrunkId)
42       ?? null,
43     [navigation.trunks, navigation.currentTrunkId]
44   );
45
46   const currentTree = useMemo(() =>
47     currentTrunk?.trees.find(t => t.id === navigation.currentTreeId)
48       ?? null,
49     [currentTrunk, navigation.currentTreeId]
50   );
51
52   const currentBranch = useMemo(() =>
53     currentTree?.branches.find(b => b.id === navigation.
54       currentBranchId) ?? null,
55     [currentTree, navigation.currentBranchId]
56   );
57
58   const currentLeaf = useMemo(() =>
59     currentBranch?.leaves.find(l => l.id === navigation.currentLeafId
60       ) ?? null,
61     [currentBranch, navigation.currentLeafId]
62   );
63
64   // Actions
65   const setTrunk = useCallback((id: string) => {
66     dispatch({ type: 'NAV_SET_TRUNK', payload: id });
67   }, [dispatch]);
68
69   const setTree = useCallback((id: string) => {
70     dispatch({ type: 'NAV_SET_TREE', payload: id });
71   }, [dispatch]);
72
73   const setBranch = useCallback((id: string) => {
74     dispatch({ type: 'NAV_SET_BRANCH', payload: id });
75   }, [dispatch]);
76
77   const createTrunk = useCallback((name: string) => {
78     dispatch({ type: 'NAV_CREATE_TRUNK', payload: { name } });
79   }, [dispatch]);
80
81   const deleteTrunk = useCallback((id: string) => {
82     dispatch({ type: 'NAV_DELETE_TRUNK', payload: id });
83   }, [dispatch]);
84
85   const createTree = useCallback((name: string) => {
86     if (!navigation.currentTrunkId) return;
87     dispatch({ type: 'NAV_CREATE_TREE', payload: {
88       trunkId: navigation.currentTrunkId,
89       name
90     }});
91   }, [dispatch, navigation.currentTrunkId]);
92
93   const deleteTree = useCallback((id: string) => {

```

```

94     dispatch({ type: 'NAV_DELETE_TREE', payload: id });
95 }, [dispatch]);
96
97 const createBranch = useCallback((name: string, fromBranchId?: string) => {
98   if (!navigation.currentTreeId) return;
99   dispatch({ type: 'NAV_CREATE_BRANCH', payload: {
100     treeId: navigation.currentTreeId,
101     name,
102     fromBranchId: fromBranchId ?? navigation.currentBranchId ?? '',
103   }});
104 }, [dispatch, navigation.currentTreeId, navigation.currentBranchId]);
105
106 const deleteBranch = useCallback((id: string) => {
107   dispatch({ type: 'NAV_DELETE_BRANCH', payload: id });
108 }, [dispatch]);
109
110 const createLeaf = useCallback((name: string, type: LeafType) => {
111   if (!navigation.currentBranchId) return;
112   dispatch({ type: 'NAV_CREATE_LEAF', payload: {
113     branchId: navigation.currentBranchId,
114     name,
115     type
116   }});
117 }, [dispatch, navigation.currentBranchId]);
118
119 const deleteLeaf = useCallback((id: string) => {
120   dispatch({ type: 'NAV_DELETE_LEAF', payload: id });
121 }, [dispatch]);
122
123 // Recent trees
124 const recentTrees = useMemo(() => {
125   const allTrees = navigation.trunks.flatMap(t => t.trees);
126   return navigation.recentTreeIds
127     .map(id => allTrees.find(t => t.id === id))
128     .filter((t): t is Tree => t !== undefined);
129 }, [navigation.trunks, navigation.recentTreeIds]);
130
131 return {
132   currentTrunk,
133   currentTree,
134   currentBranch,
135   currentLeaf,
136   trunks: navigation.trunks,
137   recentTrees,
138   setTrunk,
139   setTree,
140   setBranch,
141   setLeaf,
142   createTrunk,
143   deleteTrunk,
144   createTree,
145   deleteTree,
146   createBranch,
147   deleteBranch,
148   createLeaf,
149   deleteLeaf,

```

```
150     };
151 }
```

```
1 // src/hooks/useUI.ts
2
3 import { useCallback } from 'react';
4 import { useApplicationContext } from '../contexts/AppContext';
5 import { UIState } from '../types/state';
6
7 export interface UseUIReturn {
8     // State
9     sidebarOpen: boolean;
10    sidebarWidth: number;
11    activePanel: UIState['activePanel'];
12    theme: UIState['theme'];
13    resolvedTheme: UIState['resolvedTheme'];
14    modals: UIState['modals'];
15    notifications: UIState['notifications'];
16
17     // Actions
18    toggleSidebar: () => void;
19    setSidebarWidth: (width: number) => void;
20    setActivePanel: (panel: UIState['activePanel']) => void;
21    setTheme: (theme: UIState['theme']) => void;
22    openModal: (modal: keyof UIState['modals']) => void;
23    closeModal: (modal: keyof UIState['modals']) => void;
24    notify: (notification: { type: 'info' | 'success' | 'warning' | 'error'; message: string; duration?: number }) => void;
25    dismissNotification: (id: string) => void;
26 }
27
28 export function useUI(): UseUIReturn {
29     const { state, dispatch } = useApplicationContext();
30     const { ui } = state;
31
32     const toggleSidebar = useCallback(() => {
33         dispatch({ type: 'UI_TOGGLE_SIDEBAR' });
34     }, [dispatch]);
35
36     const setSidebarWidth = useCallback((width: number) => {
37         dispatch({ type: 'UI_SET_SIDEBAR_WIDTH', payload: width });
38     }, [dispatch]);
39
40     const setActivePanel = useCallback((panel: UIState['activePanel']) => {
41         dispatch({ type: 'UI_SET_ACTIVE_PANEL', payload: panel });
42     }, [dispatch]);
43
44     const setTheme = useCallback((theme: UIState['theme']) => {
45         dispatch({ type: 'UI_SET_THEME', payload: theme });
46     }, [dispatch]);
47
48     const openModal = useCallback((modal: keyof UIState['modals']) => {
49         dispatch({ type: 'UI_OPEN_MODAL', payload: modal });
50     }, [dispatch]);
51
52     const closeModal = useCallback((modal: keyof UIState['modals']) =>
```

```

53     {
54       dispatch({ type: 'UI_CLOSE_MODAL', payload: modal });
55     }, [dispatch]);
56 
57   const notify = useCallback((notification: {
58     type: 'info' | 'success' | 'warning' | 'error';
59     message: string;
60     duration?: number
61   }) => {
62     dispatch({
63       type: 'UI_ADD_NOTIFICATION',
64       payload: { ...notification, dismissible: true }
65     });
66   }, [dispatch]);
67 
68   const dismissNotification = useCallback((id: string) => {
69     dispatch({ type: 'UI_DISMISS_NOTIFICATION', payload: id });
70   }, [dispatch]);
71 
72   return {
73     sidebarOpen: ui.sidebarOpen,
74     sidebarWidth: ui.sidebarWidth,
75     activePanel: ui.activePanel,
76     theme: ui.theme,
77     resolvedTheme: ui.resolvedTheme,
78     modals: ui.modals,
79     notifications: ui.notifications,
80     toggleSidebar,
81     setSidebarWidth,
82     setActivePanel,
83     setTheme,
84     openModal,
85     closeModal,
86     notify,
87     dismissNotification,
88   };
89 }

```

```

1 // src/hooks/useLayers.ts
2 
3 import { useCallback, useMemo } from 'react';
4 import { useContext } from '../contexts/AppContext';
5 import { Layer, LayerType } from '../types/layers';
6 import { buildLayerTree, LayerTreeNode } from '../utils/layerTree';
7 
8 export interface UseLayersReturn {
9   // Data
10  layers: Layer[];
11  layerTree: LayerTreeNode[];
12  selectedIds: string[];
13  expandedIds: string[];
14  selectedLayers: Layer[];
15 
16  // Selection
17  selectLayers: (ids: string[], mode?: 'replace' | 'add' | 'toggle')
18    => void;
19  clearSelection: () => void;

```

```

19    selectAll: () => void;
20
21     // Expansion
22     toggleExpand: (id: string) => void;
23     expandAll: () => void;
24     collapseAll: () => void;
25
26     // CRUD
27     createLayer: (type: LayerType, parentId?: string) => void;
28     deleteSelected: () => void;
29     duplicateSelected: () => void;
30
31     // Modification
32     renameLayer: (id: string, name: string) => void;
33     toggleVisibility: (id: string) => void;
34     toggleLock: (id: string) => void;
35     reorderLayer: (id: string, targetId: string, position: 'before' | 'after' | 'inside') => void;
36
37     // Grouping
38     groupSelected: () => void;
39     ungroupLayer: (id: string) => void;
40 }
41
42 export function useLayers(): UseLayersReturn {
43     const { state, dispatch } = useAppContext();
44
45     const layers = state.editor?.leaf.layers ?? [];
46     const selectedIds = state.editor?.selectedLayerIds ?? [];
47     const expandedIds = state.editor?.expandedLayerIds ?? [];
48
49     const layerTree = useMemo(() => buildLayerTree(layers), [layers]);
50
51     const selectedLayers = useMemo(() =>
52         layers.filter(l => selectedIds.includes(l.id)),
53         [layers, selectedIds]
54     );
55
56     // Selection
57     const selectLayers = useCallback((ids: string[], mode: 'replace' | 'add' | 'toggle' = 'replace') => {
58         dispatch({ type: 'EDITOR_SELECT_LAYERS', payload: { ids, mode } });
59     }, [dispatch]);
60
61     const clearSelection = useCallback(() => {
62         dispatch({ type: 'EDITOR_CLEAR_SELECTION' });
63     }, [dispatch]);
64
65     const selectAll = useCallback(() => {
66         dispatch({ type: 'EDITOR_SELECT_LAYERS', payload: { ids: layers.map(l => l.id), mode: 'replace' } });
67     }, [dispatch, layers]);
68
69     // Expansion
70     const toggleExpand = useCallback((id: string) => {
71         dispatch({ type: 'EDITOR_TOGGLE_LAYER_EXPAND', payload: id });
72     }, [dispatch]);

```

```
73 const expandAll = useCallback(() => {
74   const groupIds = layers.filter(l => l.childIds.length > 0).map(l
75     => l.id);
76   groupIds.forEach(id => {
77     if (!expandedIds.includes(id)) {
78       dispatch({ type: 'EDITOR_TOGGLE_LAYER_EXPAND', payload: id })
79     }
80   });
81 }, [dispatch, layers, expandedIds]);
82
83 const collapseAll = useCallback(() => {
84   expandedIds.forEach(id => {
85     dispatch({ type: 'EDITOR_TOGGLE_LAYER_EXPAND', payload: id });
86   });
87 }, [dispatch, expandedIds]);
88
89 // CRUD
90 const createLayer = useCallback((type: LayerType, parentId?: string
91   ) => {
92   dispatch({ type: 'LAYER_CREATE', payload: { type, parentId } });
93 }, [dispatch]);
94
95 const deleteSelected = useCallback(() => {
96   if (selectedIds.length > 0) {
97     dispatch({ type: 'LAYER_DELETE', payload: selectedIds });
98   }
99 }, [dispatch, selectedIds]);
100
101 const duplicateSelected = useCallback(() => {
102   if (selectedIds.length > 0) {
103     dispatch({ type: 'LAYER_DUPLICATE', payload: selectedIds });
104   }
105 }, [dispatch, selectedIds]);
106
107 // Modification
108 const renameLayer = useCallback((id: string, name: string) => {
109   dispatch({ type: 'LAYER_RENAME', payload: { id, name } });
110 }, [dispatch]);
111
112 const toggleVisibility = useCallback((id: string) => {
113   dispatch({ type: 'LAYER_TOGGLE_VISIBILITY', payload: id });
114 }, [dispatch]);
115
116 const toggleLock = useCallback((id: string) => {
117   dispatch({ type: 'LAYER_TOGGLE_LOCK', payload: id });
118 }, [dispatch]);
119
120 const reorderLayer = useCallback(
121   (id: string,
122    targetId: string,
123    position: 'before' | 'after' | 'inside'
124   ) => {
125   dispatch({ type: 'LAYER_REORDER', payload: { id, targetId,
126     position } });
127 }, [dispatch]);
```

```

127    // Grouping
128    const groupSelected = useCallback(() => {
129      if (selectedIds.length > 1) {
130        dispatch({ type: 'LAYER_GROUP', payload: selectedIds });
131      }
132    }, [dispatch, selectedIds]);
133
134    const ungroupLayer = useCallback((id: string) => {
135      dispatch({ type: 'LAYER_UNGROUP', payload: id });
136    }, [dispatch]);
137
138    return {
139      layers,
140      layerTree,
141      selectedIds,
142      expandedIds,
143      selectedLayers,
144      selectLayers,
145      clearSelection,
146     selectAll,
147      toggleExpand,
148      expandAll,
149      collapseAll,
150      createLayer,
151      deleteSelected,
152      duplicateSelected,
153      renameLayer,
154      toggleVisibility,
155      toggleLock,
156      reorderLayer,
157      groupSelected,
158      ungroupLayer,
159    };
160  }

```

## Part IV

# UI Components

### 6 Ribbon Component

```

1 // src/components/Ribbon/Ribbon.tsx
2
3 import { useState, useCallback, MouseEvent } from 'react';
4 import { useUI } from '../../hooks/useUI';
5 import { usePlugins } from '../../hooks/usePlugins';
6 import { RibbonIcon } from './RibbonIcon';
7 import { RibbonContextMenu } from './RibbonContextMenu';
8 import {
9   PanelLeftClose,
10  PanelLeftOpen,
11  Layers,
12  Search,
13  GitBranch,

```

```

14     Package,
15     History,
16     Settings,
17     HelpCircle,
18 } from 'lucide-react';
19 import styles from './Ribbon.module.css';
20
21 interface RibbonAction {
22   id: string;
23   icon: React.ComponentType<{ size?: number }>;
24   tooltip: string;
25   onClick: () => void;
26   isActive?: boolean;
27 }
28
29 export function Ribbon(): JSX.Element {
30   const {
31     sidebarOpen,
32     toggleSidebar,
33     activePanel,
34     setActivePanel,
35     openModal
36   } = useUI();
37   const { pluginRibbonActions } = usePlugins();
38
39   const [contextMenu, setContextMenu] = useState<{ x: number; y:
40   number } | null>(null);
41   const [hiddenActions, setHiddenActions] = useState<Set<string>>(new
42   Set());
43
44   // Core navigation actions
45   const coreActions: RibbonAction[] = [
46     {
47       id: 'layers',
48       icon: Layers,
49       tooltip: 'Layers (L)',
50       onClick: () => setActivePanel('layers'),
51       isActive: activePanel === 'layers',
52     },
53     {
54       id: 'assets',
55       icon: Package,
56       tooltip: 'Assets',
57       onClick: () => setActivePanel('assets'),
58       isActive: activePanel === 'assets',
59     },
60     {
61       id: 'components',
62       icon: GitBranch,
63       tooltip: 'Components',
64       onClick: () => setActivePanel('components'),
65       isActive: activePanel === 'components',
66     },
67     {
68       id: 'history',
69       icon: History,
70       tooltip: 'Version History',
71       onClick: () => setActivePanel('history'),
72     },
73   ];

```

```

70         isActive: activePanel === 'history',
71     },
72     [
73         {
74             id: 'search',
75             icon: Search,
76             tooltip: 'Search (Ctrl+F)',
77             onClick: () => {/* Open search */},
78         },
79     ];
80
81     // System actions (always at bottom)
82     const systemActions: RibbonAction[] = [
83         {
84             id: 'help',
85             icon: HelpCircle,
86             tooltip: 'Help & Support',
87             onClick: () => openModal('help'),
88         },
89         {
90             id: 'settings',
91             icon: Settings,
92             tooltip: 'Settings (Ctrl+,)',
93             onClick: () => openModal('settings'),
94         },
95     ];
96
97     // Combine with plugin actions
98     const allTopActions = [
99         ...coreActions.filter(a => !hiddenActions.has(a.id)),
100        ...pluginRibbonActions.filter(a => !hiddenActions.has(a.id)),
101    ];
102
103     const handleContextMenu = useCallback((e: MouseEvent) => {
104         e.preventDefault();
105         setContextMenu({ x: e.clientX, y: e.clientY });
106     }, []);
107
108     const toggleActionVisibility = useCallback((id: string) => {
109         setHiddenActions(prev => {
110             const next = new Set(prev);
111             if (next.has(id)) {
112                 next.delete(id);
113             } else {
114                 next.add(id);
115             }
116             return next;
117         });
118     }, []);
119
120     return (
121         <nav
122             className={styles.ribbon}
123             onContextMenu={handleContextMenu}
124             aria-label="Main navigation"
125         >
126             {/* Sidebar toggle */}
127             <div className={styles.toggleSection}>
128                 <RibbonIcon

```

```

128         icon={sidebarOpen ? PanelLeftClose : PanelLeftOpen}
129         tooltip={sidebarOpen ? 'Collapse sidebar' : 'Expand sidebar'
130           }
131         onClick={toggleSidebar}
132       />
133     </div>
134
135     {/* Top section - customizable actions */}
136   <div className={styles.topSection}>
137     {allTopActions.map(action => (
138       <RibbonIcon
139         key={action.id}
140         icon={action.icon}
141         tooltip={action.tooltip}
142         onClick={action.onClick}
143         isActive={action.isActive}
144       />
145     )))
146   </div>
147
148   {/* Spacer */}
149   <div className={styles.spacer} />
150
151   {/* Bottom section - system actions */}
152   <div className={styles.bottomSection}>
153     {systemActions.map(action => (
154       <RibbonIcon
155         key={action.id}
156         icon={action.icon}
157         tooltip={action.tooltip}
158         onClick={action.onClick}
159       />
160     )))
161   </div>
162
163   {/* Context menu */}
164   {contextMenu && (
165     <RibbonContextMenu
166       position={contextMenu}
167       actions={[...coreActions, ...pluginRibbonActions]}
168       hiddenIds={hiddenActions}
169       onToggle={toggleActionVisibility}
170       onClose={() => setContextMenu(null)}
171     />
172   )}
173 </nav>
174 );
175 }

```

```

1 // src/components/Ribbon/RibbonIcon.tsx
2
3 import { useState, useRef, ComponentType, useCallback } from 'react';
4 import { Tooltip } from '../ui/Tooltip';
5 import styles from './RibbonIcon.module.css';
6
7 interface RibbonIconProps {
8   icon: ComponentType<{ size?: number; className?: string }>;

```

```

 9   tooltip: string;
10  onClick: () => void;
11  isActive?: boolean;
12  disabled?: boolean;
13 }
14
15 export function RibbonIcon({
16   icon,
17   tooltip,
18   onClick,
19   isActive = false,
20   disabled = false,
21 }: RibbonIconProps): JSX.Element {
22   const [showTooltip, setShowTooltip] = useState(false);
23   const buttonRef = useRef<HTMLButtonElement>(null);
24
25   const handleMouseEnter = useCallback(() => setShowTooltip(true),
26     []);
26   const handleMouseLeave = useCallback(() => setShowTooltip(false),
27     []);
28
29   return (
30     <div className={styles.container}>
31       <button
32         ref={buttonRef}
33         type="button"
34         className={`${styles.button} ${isActive ? styles.active : ''}`}
35         onClick={onClick}
36         disabled={disabled}
37         onMouseEnter={handleMouseEnter}
38         onMouseLeave={handleMouseLeave}
39         aria-label={tooltip}
40         aria-pressed={isActive}
41       >
42         <Icon size={20} className={styles.icon} />
43         {isActive && <span className={styles.indicator} aria-hidden="true" />}
44       </button>
45
46       <Tooltip
47         visible={showTooltip && !disabled}
48         targetRef={buttonRef}
49         position="right"
50       >
51         {tooltip}
52       </Tooltip>
53     </div>
54   );
55 }

```

## 7 Sidebar Components

### 7.1 Trunk Selector

```

1 // src/components/Sidebar/TrunkSelector.tsx
2
3 import { useState, useRef, useEffect, useCallback } from 'react';
4 import { useNavigation } from '../../../../../hooks/useNavigation';
5 import { useUI } from '../../../../../hooks/useUI';
6 import {
7   ChevronDown,
8   Plus,
9   FolderTree,
10  Cloud,
11  Settings2,
12  Check
13 } from 'lucide-react';
14 import styles from './TrunkSelector.module.css';
15
16 export function TrunkSelector(): JSX.Element {
17   const {
18     trunks,
19     currentTrunk,
20     setTrunk,
21     createTrunk
22   } = useNavigation();
23   const { openModal } = useUI();
24
25   const [isOpen, setIsOpen] = useState(false);
26   const dropdownRef = useRef<HTMLDivElement>(null);
27
28   // Close on outside click
29   useEffect(() => {
30     function handleClickOutside(event: MouseEvent): void {
31       if (dropdownRef.current && !dropdownRef.current.contains(event.target as Node)) {
32         setIsOpen(false);
33       }
34     }
35     document.addEventListener('mousedown', handleClickOutside);
36     return () => document.removeEventListener('mousedown', handleClickOutside);
37   }, []);
38
39   const handleSelect = useCallback((id: string) => {
40     setTrunk(id);
41     setIsOpen(false);
42   }, [setTrunk]);
43
44   const handleCreate = useCallback(() => {
45     const name = prompt('Trunk name:');
46     if (name?.trim()) {
47       createTrunk(name.trim());
48     }
49     setIsOpen(false);
50   }, [createTrunk]);
51
52   const handleManage = useCallback(() => {
53     setIsOpen(false);
54     openModal('trunkManager');
55   }, [openModal]);
56

```

```

57     return (
58       <div className={styles.container} ref={dropdownRef}>
59         <button
60           type="button"
61           className={styles.selector}
62           onClick={() => setIsOpen(!isOpen)}
63           aria-expanded={isOpen}
64           aria-haspopup="listbox"
65         >
66           <div className={styles.trunkInfo}>
67             <FolderTree size={16} className={styles.icon} />
68             <span className={styles.name}>
69               {currentTrunk?.name ?? 'Select Trunk'}
70             </span>
71           </div>
72           <ChevronDown
73             size={16}
74             className={`${styles.chevron} ${isOpen ? styles.open : ''}`}
75           >
76         </button>
77
78       {isOpen && (
79         <div className={styles.dropdown} role="listbox">
80           {/* Trunk list */}
81           {trunks.length > 0 && (
82             <>
83               <div className={styles.sectionLabel}>Your Trunks</div>
84               {trunks.map(trunk => (
85                 <button
86                   key={trunk.id}
87                   type="button"
88                   className={`${styles.option} ${trunk.id ===
89                     currentTrunk?.id ? styles.active : ''}`}
89                   onClick={() => handleSelect(trunk.id)}
90                   role="option"
91                   aria-selected={trunk.id === currentTrunk?.id}
92                 >
93                   {trunk.settings.syncEnabled ? (
94                     <Cloud size={14} />
95                   ) : (
96                     <FolderTree size={14} />
97                   )}
98                   <span className={styles.optionName}>{trunk.name}</
98                     span>
99                   {trunk.id === currentTrunk?.id && (
100                     <Check size={14} className={styles.checkIcon} />
101                   )}
102                 </button>
103               ))}
104               <div className={styles.divider} />
105             </>
106           )})
107
108         {/* Actions */}
109         <button type="button" className={styles.option} onClick={
110           handleCreate}>
111           <Plus size={14} />

```

```

111         <span>New Trunk</span>
112     </button>
113     <button type="button" className={styles.option} onClick={
114         handleManage}
115         <Settings2 size={14} />
116         <span>Manage Trunks...</span>
117     </button>
118     </div>
119   </div>
120 };
121 }

```

## 7.2 Tree & Branch Selector

```

1 // src/components/Sidebar/TreeBranchSelector.tsx
2
3 import { useState, useCallback } from 'react';
4 import { useNavigation } from '../../../../../hooks/useNavigation';
5 import { useUI } from '../../../../../hooks/useUI';
6 import {
7   ChevronDown,
8   TreeDeciduous,
9   GitBranch,
10  Plus,
11  GitMerge,
12  Trash2
13 } from 'lucide-react';
14 import styles from './TreeBranchSelector.module.css';
15
16 export function TreeBranchSelector(): JSX.Element {
17   const {
18     currentTrunk,
19     currentTree,
20     currentBranch,
21     setTree,
22     setBranch,
23     createTree,
24     createBranch,
25     deleteBranch
26   } = useNavigation();
27   const { openModal, notify } = useUI();
28
29   const [treeDropdownOpen, setTreeDropdownOpen] = useState(false);
30   const [branchDropdownOpen, setBranchDropdownOpen] = useState(false)
31   ;
32
33   const trees = currentTrunk?.trees ?? [];
34   const branches = currentTree?.branches ?? [];
35
36   const handleTreeSelect = useCallback((id: string) => {
37     setTree(id);
38     setTreeDropdownOpen(false);
39   }, [setTree]);
40
41   const handleBranchSelect = useCallback((id: string) => {

```

```

41     setBranch(id);
42     setBranchDropdownOpen(false);
43   }, [setBranch]);
44
45   const handleCreateTree = useCallback(() => {
46     const name = prompt('Tree name:');
47     if (name?.trim()) {
48       createTree(name.trim());
49       notify({ type: 'success', message: 'Created Tree: ${name}' });
50     }
51     setTreeDropdownOpen(false);
52   }, [createTree, notify]);
53
54   const handleCreateBranch = useCallback(() => {
55     const name = prompt('Branch name:');
56     if (name?.trim()) {
57       createBranch(name.trim());
58       notify({ type: 'success', message: 'Created Branch: ${name}' })
59       ;
60     }
61     setBranchDropdownOpen(false);
62   }, [createBranch, notify]);
63
63   const handleDeleteBranch = useCallback((id: string, e: React.
64     MouseEvent) => {
64     e.stopPropagation();
65     const branch = branches.find(b => b.id === id);
66     if (branch?.isProtected) {
67       notify({ type: 'error', message: 'Cannot delete protected
68         branch' });
69     }
70     if (confirm(`Delete branch "${branch?.name}"?`)) {
71       deleteBranch(id);
72       notify({ type: 'success', message: 'Branch deleted' });
73     }
74   }, [branches, deleteBranch, notify]);
75
76   if (!currentTrunk) {
77     return (
78       <div className={styles.placeholder}>
79         Select a Trunk to continue
80       </div>
81     );
82   }
83
84   return (
85     <div className={styles.container}>
86       /* Tree selector */
87       <div className={styles.selectorGroup}>
88         <button
89           type="button"
90           className={styles.selector}
91           onClick={() => setTreeDropdownOpen(!treeDropdownOpen)}
92           aria-expanded={treeDropdownOpen}
93         >
94           <TreeDeciduous size={14} className={styles.icon} />
95           <span className={styles.label}>

```

```
96     {currentTree?.name ?? 'Select Tree'}
```

```
97   </span>
98   <ChevronDown size={14} className={styles.chevron} />
99 </button>
```

```
100
101 {treeDropdownOpen && (
102   <div className={styles.dropdown}>
103     {trees.map(tree => (
104       <button
105         key={tree.id}
106         type="button"
107         className={'${styles.option} ${tree.id ===
108           currentTree?.id ? styles.active : ''}'}
109         onClick={() => handleTreeSelect(tree.id)}
110       >
111         <TreeDeciduous size={12} />
112         <span>{tree.name}</span>
113       </button>
114     ))}
115   <div className={styles.divider} />
116   <button type="button" className={styles.option} onClick={

117     handleCreateTree}>
118     <Plus size={12} />
119     <span>New Tree</span>
120   </button>
121 </div>
122 )
123 /* Branch selector */
124 {currentTree && (
125   <div className={styles.selectorGroup}>
126     <button
127       type="button"
128       className={styles.selector}
129       onClick={() => setBranchDropdownOpen(!branchDropdownOpen)
130         }
131       aria-expanded={branchDropdownOpen}
132     >
133       <GitBranch size={14} className={styles.icon} />
134       <span className={styles.label}>
135         {currentBranch?.name ?? 'Select Branch'}
136       </span>
137       <ChevronDown size={14} className={styles.chevron} />
138     </button>
139
140   {branchDropdownOpen && (
141     <div className={styles.dropdown}>
142       {branches.map(branch => (
143         <button
144           key={branch.id}
145           type="button"
146           className={'${styles.option} ${branch.id ===
147             currentBranch?.id ? styles.active : ''}'}
148           onClick={() => handleBranchSelect(branch.id)}
149         >
150           <GitBranch size={12} />
151           <span>{branch.name}</span>
152         </button>
153       ))}
154     </div>
155   )}
156 </div>
157
158 /* User selector */
159 {userDropdownOpen && (
160   <div className={styles.dropdown}>
161     {users.map(user => (
162       <button
163         key={user.id}
164         type="button"
165         className={'${styles.option} ${user.id ===
166           currentUser?.id ? styles.active : ''}'}
167         onClick={() => handleUserSelect(user.id)}
168       >
169         <GitUser size={14} />
170         <span>{user.name}</span>
171       </button>
172     ))}
173     <ChevronDown size={14} className={styles.chevron} />
174   </div>
175 )}
```

```

150         {branch.isProtected && (
151             <span className={styles.badge}>protected</span>
152         )}
153         {!branch.isProtected && (
154             <button
155                 type="button"
156                 className={styles.deleteBtn}
157                 onClick={(e) => handleDeleteBranch(branch.id, e)}
158                 aria-label={'Delete ${branch.name}'}
159             >
160                 <Trash2 size={12} />
161             </button>
162         )}
163     </button>
164 )
165 <div className={styles.divider} />
166 <button type="button" className={styles.option} onClick
167     ={handleCreateBranch}>
168     <Plus size={12} />
169     <span>New Branch</span>
170 </button>
171 <button
172     type="button"
173     className={styles.option}
174     onClick={() => openModal('branchManager')}
175     >
176         <GitMerge size={12} />
177         <span>Merge Branches...</span>
178     </button>
179 </div>
180     )}
181   )
182 </div>
183 );
184 }

```

### 7.3 Layers Panel

```

1 // src/components/Sidebar/LayersPanel/LayersPanel.tsx
2
3 import { useCallback } from 'react';
4 import { useLayers } from '../../../../../hooks/useLayers';
5 import { LayerTreeItem } from './LayerTreeItem';
6 import { LayerTreeNode } from '../../../../../utils/layerTree';
7 import styles from './LayersPanel.module.css';
8
9 export function LayersPanel(): JSX.Element {
10     const {
11         layerTree,
12         selectedIds,
13         expandedIds,
14         selectLayers,
15         toggleExpand,
16         toggleVisibility,

```

```
17     toggleLock,
18     renameLayer,
19     reorderLayer,
20   } = useLayers();
21
22   const handleSelect = useCallback((
23     id: string,
24     event: React.MouseEvent
25   ) => {
26     const mode = event.ctrlKey || event.metaKey
27       ? 'toggle'
28       : event.shiftKey
29       ? 'add'
30       : 'replace';
31     selectLayers([id], mode);
32   }, [selectLayers]);
33
34   const renderNode = useCallback((node: LayerTreeNode, depth: number) => {
35     const isSelected = selectedIds.includes(node.layer.id);
36     const isExpanded = expandedIds.includes(node.layer.id);
37     const hasChildren = node.children.length > 0;
38
39     return (
40       <div key={node.layer.id} role="treeitem" aria-selected={
41         isSelected}>
42         <LayerTreeItem
43           layer={node.layer}
44           depth={depth}
45           isSelected={isSelected}
46           isExpanded={isExpanded}
47           hasChildren={hasChildren}
48           onSelect={handleSelect}
49           onToggleExpand={toggleExpand}
50           onToggleVisibility={toggleVisibility}
51           onToggleLock={toggleLock}
52           onRename={renameLayer}
53           onReorder={reorderLayer}
54         />
55
56         {hasChildren && isExpanded && (
57           <div role="group" className={styles.children}>
58             {node.children.map(child => renderNode(child, depth + 1))
59               }
60             </div>
61           )}
62         </div>
63       );
64     }, [
65       selectedIds,
66       expandedIds,
67       handleSelect,
68       toggleExpand,
69       toggleVisibility,
70       toggleLock,
71       renameLayer,
72       reorderLayer
73     ]);
74 
```

```

72     return (
73       <div className={styles.panel}>
74         <div className={styles.header}>
75           <span className={styles.title}>Layers</span>
76           <span className={styles.count}>
77             {layerTree.reduce((acc, n) => acc + countNodes(n), 0)}
78           </span>
79         </div>
80       </div>
81
82       <div
83         className={styles.tree}
84         role="tree"
85         aria-label="Layer hierarchy"
86       >
87         {layerTree.length === 0 ? (
88           <div className={styles.empty}>
89             No layers yet. Create one to get started.
90           </div>
91         ) : (
92           layerTree.map(node => renderNode(node, 0))
93         )}
94       </div>
95     </div>
96   );
97 }
98
99 function countNodes(node: LayerTreeNode): number {
100   return 1 + node.children.reduce((acc, child) => acc + countNodes(
101     child), 0);
102 }
```

## 8 Modal Components

### 8.1 Base Modal

```

1 // src/components/Modal/Modal.tsx
2
3 import {
4   useEffect,
5   useCallback,
6   useRef,
7   ReactNode,
8   KeyboardEvent
9 } from 'react';
10 import { createPortal } from 'react-dom';
11 import { X } from 'lucide-react';
12 import styles from './Modal.module.css';
13
14 interface ModalProps {
15   isOpen: boolean;
16   onClose: () => void;
17   title: string;
18   children: ReactNode;
19   size?: 'small' | 'medium' | 'large' | 'fullscreen';

```

```

20     showCloseButton?: boolean;
21     closeOnOverlay?: boolean;
22     closeOnEscape?: boolean;
23   }
24
25   export function Modal({
26     isOpen,
27     onClose,
28     title,
29     children,
30     size = 'medium',
31     showCloseButton = true,
32     closeOnOverlay = true,
33     closeOnEscape = true,
34   }: ModalProps): JSX.Element | null {
35     const modalRef = useRef<HTMLDivElement>(null);
36     const previousActiveElement = useRef<HTMLElement | null>(null);
37
38     // Escape key handler
39     useEffect(() => {
40       if (!isOpen || !closeOnEscape) return;
41
42       function handleKeyDown(e: globalThis.KeyboardEvent): void {
43         if (e.key === 'Escape') {
44           onClose();
45         }
46       }
47
48       document.addEventListener('keydown', handleKeyDown);
49       return () => document.removeEventListener('keydown',
50         handleKeyDown);
51     }, [isOpen, closeOnEscape, onClose]);
52
53     // Focus management and scroll lock
54     useEffect(() => {
55       if (isOpen) {
56         previousActiveElement.current = document.activeElement as
57           HTMLElement;
58         document.body.style.overflow = 'hidden';
59         modalRef.current?.focus();
60       } else {
61         document.body.style.overflow = '';
62         previousActiveElement.current?.focus();
63       }
64
65       return () => {
66         document.body.style.overflow = '';
67       };
68     }, [isOpen]);
69
70     const handleOverlayClick = useCallback((e: React.MouseEvent) => {
71       if (closeOnOverlay && e.target === e.currentTarget) {
72         onClose();
73       }
74     }, [closeOnOverlay, onClose]);
75
76     const handleKeyDown = useCallback((e: KeyboardEvent<HTMLDivElement
77       >) => {

```

```

75      // Focus trap
76      if (e.key === 'Tab') {
77          const focusable = modalRef.current?.querySelectorAll<
78              HTMLElement>(
79                  'button, [href], input, select, textarea, [tabindex]:not([
80                      tabindex="-1"] )',
81              );
82
83          if (!focusable || focusable.length === 0) return;
84
85          const first = focusable[0];
86          const last = focusable[focusable.length - 1];
87
88          if (e.shiftKey && document.activeElement === first) {
89              e.preventDefault();
90              last.focus();
91          } else if (!e.shiftKey && document.activeElement === last) {
92              e.preventDefault();
93              first.focus();
94          }
95      },
96      [],
97  );
98
99  if (!isOpen) return null;
100
101  const modalContent = (
102      <div
103          className={styles.overlay}
104          onClick={handleOverlayClick}
105          aria-modal="true"
106          role="dialog"
107          aria-labelledby="modal-title"
108      >
109          <div
110              ref={modalRef}
111              className={'${styles.modal} ${styles[size]}'}
112              tabIndex={-1}
113              onKeyDown={handleKeyDown}
114          >
115              <header className={styles.header}>
116                  <h2 id="modal-title" className={styles.title}>{title}</h2>
117                  {showCloseButton && (
118                      <button
119                          type="button"
120                          className={styles.closeButton}
121                          onClick={onClose}
122                          aria-label="Close modal"
123                      >
124                          <X size={18} />
125                      </button>
126                  )}
127              </header>
128
129              <div className={styles.content}>
130                  {children}
131              </div>
132          </div>
133      </div>
134  );

```

```

131     );
132
133     // Render to portal
134     const portalRoot = document.getElementById('modal-root') ??
135         document.body;
136     return createPortal(modalContent, portalRoot);
137 }
```

## Part V

# Plugin System

## 9 Plugin Architecture

### 9.1 Plugin Types

```

1 // src/types/plugin.ts
2
3 import { ComponentType } from 'react';
4 import { Layer, LayerType } from './layers';
5
6 /**
7  * Plugin manifest - metadata loaded from plugin.json
8 */
9 export interface PluginManifest {
10   id: string;
11   name: string;
12   version: string;
13   description: string;
14   author: string;
15   authorUrl?: string;
16   minAppVersion: string;
17   main: string;
18   repository?: string;
19   license?: string;
20 }
21
22 /**
23  * Plugin lifecycle interface
24 */
25 export interface Plugin {
26   onload(): void | Promise<void>;
27   onunload(): void | Promise<void>;
28 }
29
30 /**
31  * Plugin API - provided to plugins for app integration
32 */
33 export interface PluginAPI {
34   // Ribbon
35   addRibbonIcon(
36     id: string,
37     icon: ComponentType<{ size?: number }>,
38     tooltip: string,
```

```

39     callback: () => void
40   ): () => void;
41
42   // Toolbar
43   addToolbarAction(
44     id: string,
45     icon: ComponentType<{ size?: number }>,
46     tooltip: string,
47     callback: () => void,
48     options?: ToolbarActionOptions
49   ): () => void;
50
51   // Commands
52   registerCommand(command: Command): () => void;
53   executeCommand(id: string): void;
54
55   // Settings
56   registerSettingTab(tab: SettingTab): () => void;
57
58   // Storage
59   loadData<T = unknown>(): Promise<T | null>;
60   saveData<T = unknown>(data: T): Promise<void>;
61
62   // Events
63   on<K extends keyof PluginEvents>(
64     event: K,
65     callback: PluginEvents[K]
66   ): () => void;
67
68   // Layer operations
69   getSelectedLayers(): Layer[];
70   selectLayers(ids: string[]): void;
71   createLayer(type: LayerType, props?: Partial<Layer>): string;
72   updateLayer(id: string, changes: Partial<Layer>): void;
73   deleteLayer(id: string): void;
74
75   // UI
76   showModal(component: ComponentType<{ onClose: () => void }>): void;
77   showNotice(message: string, type?: 'info' | 'success' | 'warning' |
78     'error'): void;
79   showConfirm(message: string): Promise<boolean>;
80 }
81
82 export interface ToolbarActionOptions {
83   position?: number;
84   disabled?: boolean;
85   separator?: 'before' | 'after';
86 }
87
88 export interface Command {
89   id: string;
90   name: string;
91   description?: string;
92   hotkeys?: string[];
93   callback: () => void;
94   checkCallback?: () => boolean; // For conditional enablement
95 }
```

```

96  export interface SettingTab {
97    id: string;
98    name: string;
99    icon: ComponentType<{ size?: number }>;
100   component: ComponentType;
101 }
102
103 export interface PluginEvents {
104   'layer:created': (layer: Layer) => void;
105   'layer:updated': (layer: Layer, changes: Partial<Layer>) => void;
106   'layer:deleted': (id: string) => void;
107   'selection:changed': (ids: string[]) => void;
108   'leaf:opened': (leafId: string) => void;
109   'leaf:closed': () => void;
110   'branch:switched': (branchId: string) => void;
111   'tree:switched': (treeId: string) => void;
112 }

```

## 9.2 Plugin Context

```

1 // src/contexts/PluginContext.tsx
2
3 import {
4   createContext,
5   useContext,
6   useState,
7   useCallback,
8   useEffect,
9   ReactNode,
10  ComponentType
11 } from 'react';
12 import { Plugin, PluginManifest, PluginAPI, Command, SettingTab }
13   from '../types/plugin';
13 import { useAppContext } from './AppContext';
14 import { Layer, LayerType } from '../types/layers';
15
16 interface RibbonAction {
17   id: string;
18   pluginId: string;
19   icon: ComponentType<{ size?: number }>;
20   tooltip: string;
21   onClick: () => void;
22 }
23
24 interface ToolbarAction {
25   id: string;
26   pluginId: string;
27   icon: ComponentType<{ size?: number }>;
28   tooltip: string;
29   onClick: () => void;
30   disabled?: boolean;
31   position?: number;
32 }
33
34 interface PluginInstance {
35   manifest: PluginManifest;

```

```

36     instance: Plugin;
37   }
38
39   interface PluginContextValue {
40     plugins: Map<string, PluginInstance>;
41     pluginRibbonActions: RibbonAction[];
42     pluginToolbarActions: ToolbarAction[];
43     pluginCommands: Command[];
44     pluginSettingTabs: SettingTab[];
45     loadPlugin: (manifest: PluginManifest) => Promise<void>;
46     unloadPlugin: (id: string) => Promise<void>;
47     isPluginEnabled: (id: string) => boolean;
48   }
49
50 const PluginContext = createContext<PluginContextValue | null>(null);
51
52 interface PluginProviderProps {
53   children: ReactNode;
54 }
55
56 export function PluginProvider({ children }: PluginProviderProps): JSX.Element {
57   const { state, dispatch } = useAppContext();
58
59   const [plugins] = useState(() => new Map<string, PluginInstance>())
60   ;
61   const [ribbonActions, setRibbonActions] = useState<RibbonAction[]>([]);
62   const [toolbarActions, setToolbarActions] = useState<ToolbarAction[]>([]);
63   const [commands, setCommands] = useState<Command[]>([]);
64   const [settingTabs, setSettingTabs] = useState<SettingTab[]>([]);
65
66   // Create API for a specific plugin
67   const createPluginAPI = useCallback((pluginId: string): PluginAPI => {
68     const fullId = (id: string) => `${pluginId}:${id}`;
69
70     return {
71       // Ribbon
72       addRibbonIcon(id, icon, tooltip, callback) {
73         const actionId = fullId(id);
74         const action: RibbonAction = {
75           id: actionId,
76           pluginId,
77           icon,
78           tooltip,
79           onClick: callback
80         };
81         setRibbonActions(prev => [...prev, action]);
82         return () => setRibbonActions(prev => prev.filter(a => a.id
83           !== actionId));
84       },
85       // Toolbar
86       addToolbarAction(id, icon, tooltip, callback, options) {
87         const actionId = fullId(id);
88         const action: ToolbarAction = {

```

```

88         id: actionId,
89         pluginId,
90         icon,
91         tooltip,
92         onClick: callback,
93         disabled: options?.disabled,
94         position: options?.position,
95     };
96     setToolbarActions(prev => {
97         const next = [...prev, action];
98         if (options?.position !== undefined) {
99             next.sort((a, b) => (a.position ?? 999) - (b.position ?? 999));
100        }
101        return next;
102    });
103    return () => setToolbarActions(prev => prev.filter(a => a.id
104        !== actionId));
105 },
106
107 // Commands
108 registerCommand(command) {
109     const cmd = { ...command, id: fullId(command.id) };
110     setCommands(prev => [...prev, cmd]);
111     return () => setCommands(prev => prev.filter(c => c.id !==
112         cmd.id));
113 },
114
115 executeCommand(id) {
116     const cmd = commands.find(c => c.id === fullId(id));
117     if (cmd && (!cmd.checkCallback || cmd.checkCallback())) {
118         cmd.callback();
119     }
120 },
121
122 // Settings
123 registerSettingTab(tab) {
124     const t = { ...tab, id: fullId(tab.id) };
125     setSettingTabs(prev => [...prev, t]);
126     return () => setSettingTabs(prev => prev.filter(st => st.id
127         !== t.id));
128 },
129
130 // Storage
131 async loadData<T>() {
132     const key = `plugin:${pluginId}:data`;
133     try {
134         const data = localStorage.getItem(key);
135         return data ? JSON.parse(data) as T : null;
136     } catch {
137         return null;
138     }
139 }
140
141 async saveData<T>(data: T) {
142     const key = `plugin:${pluginId}:data`;
143     localStorage.setItem(key, JSON.stringify(data));
144 }

```

```
142     // Events
143     on(event, callback) {
144         // Event system implementation
145         return () => {};
146     },
147
148     // Layer operations
149     getSelectedLayers() {
150         return state.editor?.leaf.layers.filter(
151             l => state.editor?.selectedLayerIds.includes(l.id)
152         ) ?? [];
153     },
154
155     selectLayers(ids) {
156         dispatch({ type: 'EDITOR_SELECT_LAYERS', payload: { ids, mode
157             : 'replace' } });
158     },
159
160     createLayer(type: LayerType, props?: Partial<Layer>) {
161         const id = crypto.randomUUID();
162         dispatch({ type: 'LAYER_CREATE', payload: { type, ...props } });
163         return id;
164     },
165
166     updateLayer(id, changes) {
167         dispatch({ type: 'LAYER_UPDATE', payload: { id, changes } });
168     },
169
170     deleteLayer(id) {
171         dispatch({ type: 'LAYER_DELETE', payload: [id] });
172     },
173
174     // UI
175     showModal(component) {
176         // Modal implementation
177     },
178
179     showNotice(message, type = 'info') {
180         dispatch({
181             type: 'UI_ADD_NOTIFICATION',
182             payload: { type, message, dismissible: true, duration: 3000
183                 }
184         });
185
186         async showConfirm(message) {
187             return window.confirm(message);
188         },
189     };
190 }, [state, dispatch, commands]);
191
192 // Load plugin
193 const loadPlugin = useCallback(async (manifest: PluginManifest) =>
194 {
195     if (plugins.has(manifest.id)) {
196         console.warn(`Plugin ${manifest.id} already loaded`);
197     }
198 }
```

```

196         return;
197     }
198
199     try {
200         const module = await import(/* @vite-ignore */ manifest.main);
201         const PluginClass = module.default as new (api: PluginAPI) =>
202             Plugin;
203
204         const api = createPluginAPI(manifest.id);
205         const instance = new PluginClass(api);
206
207         await instance.onload();
208
209         plugins.set(manifest.id, { manifest, instance });
210         dispatch({ type: 'PLUGIN_INSTALL', payload: manifest });
211
212         console.log(`Plugin loaded: ${manifest.name} v${manifest.version}`);
213     } catch (error) {
214         console.error(`Failed to load plugin ${manifest.id}:`, error);
215         throw error;
216     }
217 }, [createPluginAPI, dispatch, plugins]);
218
219 // Unload plugin
220 const unloadPlugin = useCallback(async (id: string) => {
221     const plugin = plugins.get(id);
222     if (!plugin) return;
223
224     try {
225         await plugin.instance.onunload();
226     } catch (error) {
227         console.error(`Error unloading plugin ${id}:`, error);
228     }
229
230     // Clean up registrations
231     setRibbonActions(prev => prev.filter(a => a.pluginId !== id));
232     setToolbarActions(prev => prev.filter(a => a.pluginId !== id));
233     setCommands(prev => prev.filter(c => !c.id.startsWith(`${id}:`)));
234     ;
235     setSettingTabs(prev => prev.filter(t => !t.id.startsWith(`${id}:`)));
236
237     plugins.delete(id);
238     dispatch({ type: 'PLUGIN_UNINSTALL', payload: id });
239
240     console.log(`Plugin unloaded: ${id}`);
241 }, [dispatch, plugins]);
242
243 const isPluginEnabled = useCallback((id: string) => {
244     return state.plugins.enabled.includes(id);
245 }, [state.plugins.enabled]);
246
247 const value: PluginContextValue = {
248     plugins,
249     pluginRibbonActions: ribbonActions,
250     pluginToolbarActions: toolbarActions,
251     pluginCommands: commands,

```

```

250     pluginSettingTabs: settingTabs,
251     loadPlugin,
252     unloadPlugin,
253     isPluginEnabled,
254   );
255
256   return (
257     <PluginContext.Provider value={value}>
258       {children}
259     </PluginContext.Provider>
260   );
261 }
262
263 export function usePlugins(): PluginContextValue {
264   const context = useContext(PluginContext);
265   if (!context) {
266     throw new Error('usePlugins must be used within PluginProvider');
267   }
268   return context;
269 }

```

## Part VI

# Implementation Timeline

## 10 Phased Development Plan

### Phase 1: Foundation (Weeks 1-2)

**Goal:** Core type system and state management

- Define all TypeScript types (arborist.ts, layers.ts, state.ts, actions.ts)
- Implement ApplicationContext and reducer skeleton
- Create useNavigation, useUI, useLayers hooks
- Set up CSS variables and theme system
- Create utility functions (layerTree, id generation)

**Deliverable:** Type-safe state management foundation

### Phase 2: Shell Components (Weeks 3-4)

**Goal:** Application layout and navigation

- Implement AppShell layout component
- Build Ribbon with RibbonIcon components
- Create TrunkSelector dropdown
- Create TreeBranchSelector component
- Implement sidebar resize functionality
- Add keyboard navigation support

**Deliverable:** Navigable application shell

### Phase 3: Layers Panel (Weeks 5-6)

**Goal:** Full layer management

- Build LayersPanel with tree view
- Implement LayerTreeItem with all interactions
- Add multi-select (Ctrl+Click, Shift+Click)
- Implement drag-and-drop reordering
- Add inline rename functionality
- Implement visibility/lock toggles
- Add context menu for layer actions

**Deliverable:** Fully functional layers panel

### Phase 4: Modal System (Weeks 7-8)

**Goal:** Complete modal infrastructure

- Build base Modal component with portal
- Implement SettingsModal with tabs
- Create HelpModal with keyboard shortcuts
- Build PaymentModal with plan selection
- Add TrunkManagerModal
- Implement BranchManagerModal (merge UI)
- Add ExportDialog

**Deliverable:** All modal dialogs functional

### Phase 5: Plugin System (Weeks 9-10)

**Goal:** Extensibility infrastructure

- Define Plugin interface and PluginAPI
- Implement PluginContext and loader
- Build plugin settings UI
- Create example plugin (export, theme, etc.)
- Add command palette integration
- Document plugin development guide

**Deliverable:** Working plugin architecture

### Phase 6: Polish & Integration (Weeks 11-12)

**Goal:** Production readiness

- Add keyboard shortcuts throughout
- Implement undo/redo system
- Add loading states and error boundaries
- Accessibility audit (ARIA, focus management)
- Performance optimization (memoization, virtualization)
- Write integration tests
- Documentation and comments

**Deliverable:** Production-ready UI system

## 11 File Structure

```
1  src/
2      components/
3          AppShell/
4              AppShell.tsx
5              AppShell.module.css
6          Ribbon/
7              Ribbon.tsx
8              Ribbon.module.css
9              RibbonIcon.tsx
10             RibbonIcon.module.css
11             RibbonContextMenu.tsx
12         Sidebar/
13             Sidebar.tsx
14             Sidebar.module.css
15             TrunkSelector.tsx
16             TrunkSelector.module.css
17             TreeBranchSelector.tsx
18             TreeBranchSelector.module.css
19             Toolbar.tsx
20             Toolbar.module.css
21             LayersPanel/
22                 LayersPanel.tsx
23                 LayersPanel.module.css
24                 LayerTreeItem.tsx
25                 LayerTreeItem.module.css
26         Canvas/
27             Canvas.tsx
28         Modal/
29             Modal.tsx
30             Modal.module.css
31             SettingsModal/
32                 HelpModal/
33                 PaymentModal/
34                 TrunkManagerModal/
35                 BranchManagerModal/
36             ui/
37                 Tooltip.tsx
38                 Select.tsx
39                 Switch.tsx
40                 Slider.tsx
41                 Button.tsx
42                 Input.tsx
43             contexts/
44                 ApplicationContext.tsx
45                 PluginContext.tsx
46         hooks/
47             useNavigation.ts
48             useUI.ts
49             useLayers.ts
50             usePlugins.ts
51             useKeyboard.ts
```

```

52      reducers/
53          appReducer.ts
54          navigationReducer.ts
55          uiReducer.ts
56          editorReducer.ts
57          initialState.ts
58      types/
59          arborist.ts
60          layers.ts
61          state.ts
62          actions.ts
63          plugin.ts
64      utils/
65          layerTree.ts
66          ids.ts
67          storage.ts
68      styles/
69          variables.css
70          reset.css
71          global.css
72      plugins/
73          example-plugin/
74              manifest.json
75              main.ts
76
    App.tsx

```

## 12 Summary

Component	Purpose	Phase
Type System	Trunk/Tree/Branch/Leaf + Layer types	1
State Management	AppContext, reducers, hooks	1
Ribbon	Navigation icons, plugin slots	2
TrunkSelector	Workspace switching	2
TreeBranchSelector	Project/version navigation	2
LayersPanel	Hierarchical layer tree	3
Modal System	Settings, Help, Payment dialogs	4
Plugin System	Extensibility API	5

Table 3: Implementation Summary

---

## Trunk → Tree → Branch → Leaf

*A forest of creativity, version-controlled.*

---