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

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

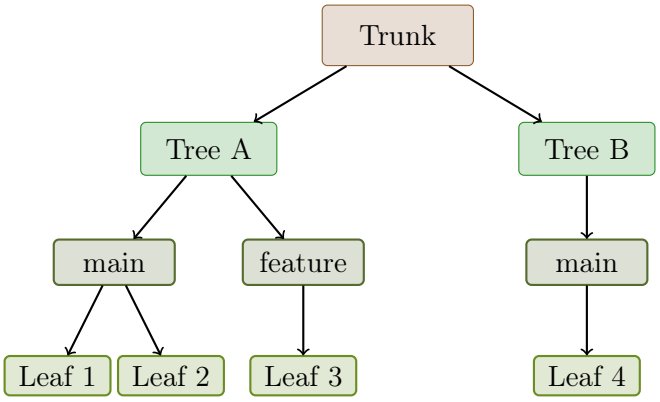
Table 1: Core Naming Convention

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

```typescript
// src/types/arborist.ts

/**
 * Trunk - Workspace containing multiple Trees
 * Analogous to a workspace or collection of repositories
 */
export interface Trunk {
  id: string;
  name: string;
  trees: Tree[];
  settings: TrunkSettings;
  createdAt: number;
  lastOpenedAt: number;
}

export interface TrunkSettings {
  theme: 'dark' | 'light' | 'system';
```

```typescript
  accentColor: string;
  sidebarWidth: number;
  autoSave: boolean;
  autoSaveInterval: number;
  syncEnabled: boolean;
}

/**
 * Tree - Repository/Project containing Branches
 * Represents a complete project with version history
 */
export interface Tree {
  id: string;
  name: string;
  path: string;                    // Local path or remote URL
  branches: Branch[];
  currentBranchId: string;
  defaultBranchId: string;         // Usually 'main'
  remotes: GitRemote[];
  bark: TreeBark;                  // Metadata
  isCloudSync: boolean;
  createdAt: number;
  lastModifiedAt: number;
}

export interface TreeBark {
  description: string;
  thumbnail?: string;
  tags: string[];
  author: string;
  license?: string;
}

export interface GitRemote {
  name: string;                    // e.g., "origin"
  url: string;
  type: 'github' | 'gitlab' | 'bitbucket' | 'self-hosted';
}

/**
 * Branch - Version/variant of a Tree
 * Direct mapping to Git branch concept
 */
export interface Branch {
  id: string;
  name: string;                    // e.g., "main", "feature/dark-
      mode"
  gitRef: string;                  // Git reference
  leaves: Leaf[];
  parentBranchId: string | null;   // For branch visualization
  lastCommit: Commit | null;
  isProtected: boolean;
  createdAt: number;
  lastModifiedAt: number;
}

export interface Commit {
  hash: string;
```

```
  shortHash: string;
  message: string;
  author: string;
  email: string;
  timestamp: number;
  parentHashes: string[];
}

/**
 * Leaf - Individual design document
 * The atomic unit of work, containing layers
 */
export interface Leaf {
  id: string;
  name: string;
  type: LeafType;
  layers: Layer[];
  canvas: CanvasSettings;
  thumbnail?: string;
  createdAt: number;
  lastModifiedAt: number;
}

export type LeafType =
  | 'design'      // Standard design document
  | 'component'   // Reusable component
  | 'asset'       // Static asset
  | 'prototype';  // Interactive prototype

export interface CanvasSettings {
  width: number;
  height: number;
  backgroundColor: string;
  gridEnabled: boolean;
  gridSize: number;
  snapToGrid: boolean;
  rulerEnabled: boolean;
}
```

3.2 Layer Types

```
// src/types/layers.ts

/**
 * Layer - Element within a Leaf
 * Hierarchical structure for design elements
 */
export interface Layer {
  id: string;
  name: string;
  type: LayerType;
  visible: boolean;
  locked: boolean;
  opacity: number;
  blendMode: BlendMode;
  parentId: string | null;
```

```typescript
  childIds: string[];
  order: number;                    // Sort order within parent
  transform: Transform;
  constraints: Constraints;
}

export type LayerType =
  | 'frame'
  | 'group'
  | 'rectangle'
  | 'ellipse'
  | 'polygon'
  | 'path'
  | 'text'
  | 'image'
  | 'component'
  | 'instance';

export type BlendMode =
  | 'normal'
  | 'multiply'
  | 'screen'
  | 'overlay'
  | 'darken'
  | 'lighten';

export interface Transform {
  x: number;
  y: number;
  width: number;
  height: number;
  rotation: number;
  scaleX: number;
  scaleY: number;
}

export interface Constraints {
  horizontal: 'left' | 'right' | 'center' | 'scale' | 'left-right';
  vertical: 'top' | 'bottom' | 'center' | 'scale' | 'top-bottom';
}

// Type-specific properties
export interface FrameLayer extends Layer {
  type: 'frame';
  fill: Fill[];
  stroke: Stroke[];
  cornerRadius: number | [number, number, number, number];
  clipContent: boolean;
  layoutMode: 'none' | 'horizontal' | 'vertical';
  layoutProps?: AutoLayoutProps;
}

export interface TextLayer extends Layer {
  type: 'text';
  content: string;
  fontFamily: string;
  fontSize: number;
  fontWeight: number;
```

```
  lineHeight: number | 'auto';
  letterSpacing: number;
  textAlign: 'left' | 'center' | 'right' | 'justify';
  fill: Fill[];
}

export interface ImageLayer extends Layer {
  type: 'image';
  src: string;
  fit: 'fill' | 'fit' | 'crop' | 'tile';
}

export interface Fill {
  type: 'solid' | 'gradient' | 'image';
  color?: string;
  opacity?: number;
  gradient?: Gradient;
}

export interface Stroke {
  color: string;
  width: number;
  position: 'inside' | 'center' | 'outside';
  dashPattern?: number[];
}

export interface Gradient {
  type: 'linear' | 'radial' | 'angular';
  stops: GradientStop[];
  angle?: number;
}

export interface GradientStop {
  position: number;
  color: string;
}

export interface AutoLayoutProps {
  direction: 'horizontal' | 'vertical';
  gap: number;
  paddingTop: number;
  paddingRight: number;
  paddingBottom: number;
  paddingLeft: number;
  alignItems: 'start' | 'center' | 'end' | 'stretch';
  justifyContent: 'start' | 'center' | 'end' | 'space-between';
}
```

4 Application State

4.1 State Shape

```
// src/types/state.ts

export interface AppState {
```

```typescript
  // Navigation state
  navigation: NavigationState;

  // UI state
  ui: UIState;

  // Editor state (current Leaf)
  editor: EditorState | null;

  // Plugin state
  plugins: PluginsState;

  // User state
  user: UserState;
}

export interface NavigationState {
  // Current location in the tree
  currentTrunkId: string | null;
  currentTreeId: string | null;
  currentBranchId: string | null;
  currentLeafId: string | null;

  // Data
  trunks: Trunk[];
  recentTreeIds: string[];
  recentLeafIds: string[];
}

export interface UIState {
  // Sidebar
  sidebarOpen: boolean;
  sidebarWidth: number;
  activePanel: 'layers' | 'assets' | 'components' | 'history';

  // Modals
  modals: {
    settings: boolean;
    help: boolean;
    payment: boolean;
    trunkManager: boolean;
    branchManager: boolean;
    exportDialog: boolean;
  };

  // Theme
  theme: 'dark' | 'light' | 'system';
  resolvedTheme: 'dark' | 'light';
  accentColor: string;

  // Notifications
  notifications: Notification[];
}

export interface EditorState {
  // Current Leaf data
  leaf: Leaf;

```

```typescript
  // Selection
  selectedLayerIds: string[];
  hoveredLayerId: string | null;

  // Layers panel
  expandedLayerIds: string[];

  // Viewport
  viewport: {
    x: number;
    y: number;
    zoom: number;
  };

  // Tool state
  activeTool: ToolType;
  toolOptions: Record<string, unknown>;

  // History
  canUndo: boolean;
  canRedo: boolean;
}

export type ToolType =
  | 'select'
  | 'frame'
  | 'rectangle'
  | 'ellipse'
  | 'polygon'
  | 'pen'
  | 'text'
  | 'hand'
  | 'zoom';

export interface PluginsState {
  installed: PluginManifest[];
  enabled: string[];
  settings: Record<string, unknown>;
}

export interface UserState {
  isAuthenticated: boolean;
  profile: UserProfile | null;
  subscription: SubscriptionTier;
  syncStatus: 'idle' | 'syncing' | 'error';
}

export interface UserProfile {
  id: string;
  email: string;
  name: string;
  avatarUrl?: string;
}

export type SubscriptionTier = 'free' | 'pro' | 'team' | 'enterprise'
    ;

export interface Notification {
```

```
  id: string;
  type: 'info' | 'success' | 'warning' | 'error';
  message: string;
  duration?: number;
  dismissible: boolean;
}
```

4.2 Action Types

```
// src/types/actions.ts

export type AppAction =
  // Navigation actions
  | { type: 'NAV_SET_TRUNK'; payload: string }
  | { type: 'NAV_SET_TREE'; payload: string }
  | { type: 'NAV_SET_BRANCH'; payload: string }
  | { type: 'NAV_SET_LEAF'; payload: string }
  | { type: 'NAV_LOAD_TRUNKS'; payload: Trunk[] }
  | { type: 'NAV_CREATE_TRUNK'; payload: { name: string } }
  | { type: 'NAV_DELETE_TRUNK'; payload: string }
  | { type: 'NAV_CREATE_TREE'; payload: { trunkId: string; name: string } }
  | { type: 'NAV_DELETE_TREE'; payload: string }
  | { type: 'NAV_CREATE_BRANCH'; payload: { treeId: string; name: string; fromBranchId: string } }
  | { type: 'NAV_DELETE_BRANCH'; payload: string }
  | { type: 'NAV_CREATE_LEAF'; payload: { branchId: string; name: string; type: LeafType } }
  | { type: 'NAV_DELETE_LEAF'; payload: string }

  // UI actions
  | { type: 'UI_TOGGLE_SIDEBAR' }
  | { type: 'UI_SET_SIDEBAR_WIDTH'; payload: number }
  | { type: 'UI_SET_ACTIVE_PANEL'; payload: UIState['activePanel'] }
  | { type: 'UI_OPEN_MODAL'; payload: keyof UIState['modals'] }
  | { type: 'UI_CLOSE_MODAL'; payload: keyof UIState['modals'] }
  | { type: 'UI_SET_THEME'; payload: 'dark' | 'light' | 'system' }
  | { type: 'UI_ADD_NOTIFICATION'; payload: Omit<Notification, 'id'> }
  | { type: 'UI_DISMISS_NOTIFICATION'; payload: string }

  // Editor actions
  | { type: 'EDITOR_LOAD_LEAF'; payload: Leaf }
  | { type: 'EDITOR_UNLOAD_LEAF' }
  | { type: 'EDITOR_SELECT_LAYERS'; payload: { ids: string[]; mode: 'replace' | 'add' | 'toggle' } }
  | { type: 'EDITOR_CLEAR_SELECTION' }
  | { type: 'EDITOR_TOGGLE_LAYER_EXPAND'; payload: string }
  | { type: 'EDITOR_SET_VIEWPORT'; payload: Partial<EditorState['viewport']> }
  | { type: 'EDITOR_SET_TOOL'; payload: ToolType }

  // Layer actions
  | { type: 'LAYER_CREATE'; payload: { type: LayerType; parentId?: string } }
  | { type: 'LAYER_DELETE'; payload: string[] }
```

```
  | { type: 'LAYER_UPDATE'; payload: { id: string; changes: Partial<Layer> } }
  | { type: 'LAYER_RENAME'; payload: { id: string; name: string } }
  | { type: 'LAYER_REORDER'; payload: { id: string; targetId: string; position: 'before' | 'after' | 'inside' } }
  | { type: 'LAYER_TOGGLE_VISIBILITY'; payload: string }
  | { type: 'LAYER_TOGGLE_LOCK'; payload: string }
  | { type: 'LAYER_DUPLICATE'; payload: string[] }
  | { type: 'LAYER_GROUP'; payload: string[] }
  | { type: 'LAYER_UNGROUP'; payload: string }

  // Plugin actions
  | { type: 'PLUGIN_INSTALL'; payload: PluginManifest }
  | { type: 'PLUGIN_UNINSTALL'; payload: string }
  | { type: 'PLUGIN_ENABLE'; payload: string }
  | { type: 'PLUGIN_DISABLE'; payload: string }

  // User actions
  | { type: 'USER_LOGIN'; payload: UserProfile }
  | { type: 'USER_LOGOUT' }
  | { type: 'USER_SET_SUBSCRIPTION'; payload: SubscriptionTier };
```

Part III
State Management

5 Context Architecture

5.1 App Context

```
// src/contexts/AppContext.tsx

import { createContext, useContext, useReducer, ReactNode, Dispatch } from 'react';
import { AppState, AppAction } from '../types/state';
import { appReducer } from '../reducers/appReducer';
import { initialState } from '../reducers/initialState';

interface AppContextValue {
  state: AppState;
  dispatch: Dispatch<AppAction>;
}

const AppContext = createContext<AppContextValue | null>(null);

interface AppProviderProps {
  children: ReactNode;
}

export function AppProvider({ children }: AppProviderProps): JSX.Element {
  const [state, dispatch] = useReducer(appReducer, initialState);

  return (
```

```
      <AppContext.Provider value={{ state, dispatch }}>
        {children}
      </AppContext.Provider>
    );
}

export function useAppContext(): AppContextValue {
  const context = useContext(AppContext);
  if (!context) {
    throw new Error('useAppContext must be used within AppProvider');
  }
  return context;
}
```

5.2 Specialized Hooks

```
// src/hooks/useNavigation.ts

import { useCallback, useMemo } from 'react';
import { useAppContext } from '../contexts/AppContext';
import { Trunk, Tree, Branch, Leaf, LeafType } from '../types/
    arborist';

export interface UseNavigationReturn {
  // Current selections
  currentTrunk: Trunk | null;
  currentTree: Tree | null;
  currentBranch: Branch | null;
  currentLeaf: Leaf | null;

  // Data
  trunks: Trunk[];
  recentTrees: Tree[];

  // Actions
  setTrunk: (id: string) => void;
  setTree: (id: string) => void;
  setBranch: (id: string) => void;
  setLeaf: (id: string) => void;

  // CRUD
  createTrunk: (name: string) => void;
  deleteTrunk: (id: string) => void;
  createTree: (name: string) => void;
  deleteTree: (id: string) => void;
  createBranch: (name: string, fromBranchId?: string) => void;
  deleteBranch: (id: string) => void;
  createLeaf: (name: string, type: LeafType) => void;
  deleteLeaf: (id: string) => void;
}

export function useNavigation(): UseNavigationReturn {
  const { state, dispatch } = useAppContext();
  const { navigation } = state;

  // Memoized current selections
```

```typescript
  const currentTrunk = useMemo(() =>
    navigation.trunks.find(t => t.id === navigation.currentTrunkId)
       ?? null,
    [navigation.trunks, navigation.currentTrunkId]
  );

  const currentTree = useMemo(() =>
    currentTrunk?.trees.find(t => t.id === navigation.currentTreeId)
       ?? null,
    [currentTrunk, navigation.currentTreeId]
  );

  const currentBranch = useMemo(() =>
    currentTree?.branches.find(b => b.id === navigation.
       currentBranchId) ?? null,
    [currentTree, navigation.currentBranchId]
  );

  const currentLeaf = useMemo(() =>
    currentBranch?.leaves.find(l => l.id === navigation.currentLeafId
       ) ?? null,
    [currentBranch, navigation.currentLeafId]
  );

  // Actions
  const setTrunk = useCallback((id: string) => {
    dispatch({ type: 'NAV_SET_TRUNK', payload: id });
  }, [dispatch]);

  const setTree = useCallback((id: string) => {
    dispatch({ type: 'NAV_SET_TREE', payload: id });
  }, [dispatch]);

  const setBranch = useCallback((id: string) => {
    dispatch({ type: 'NAV_SET_BRANCH', payload: id });
  }, [dispatch]);

  const setLeaf = useCallback((id: string) => {
    dispatch({ type: 'NAV_SET_LEAF', payload: id });
  }, [dispatch]);

  const createTrunk = useCallback((name: string) => {
    dispatch({ type: 'NAV_CREATE_TRUNK', payload: { name } });
  }, [dispatch]);

  const deleteTrunk = useCallback((id: string) => {
    dispatch({ type: 'NAV_DELETE_TRUNK', payload: id });
  }, [dispatch]);

  const createTree = useCallback((name: string) => {
    if (!navigation.currentTrunkId) return;
    dispatch({ type: 'NAV_CREATE_TREE', payload: {
      trunkId: navigation.currentTrunkId,
      name
    }});
  }, [dispatch, navigation.currentTrunkId]);

  const deleteTree = useCallback((id: string) => {
```

```
    dispatch({ type: 'NAV_DELETE_TREE', payload: id });
  }, [dispatch]);

  const createBranch = useCallback((name: string, fromBranchId?: string) => {
    if (!navigation.currentTreeId) return;
    dispatch({ type: 'NAV_CREATE_BRANCH', payload: {
      treeId: navigation.currentTreeId,
      name,
      fromBranchId: fromBranchId ?? navigation.currentBranchId ?? ''
    }});
  }, [dispatch, navigation.currentTreeId, navigation.currentBranchId]);

  const deleteBranch = useCallback((id: string) => {
    dispatch({ type: 'NAV_DELETE_BRANCH', payload: id });
  }, [dispatch]);

  const createLeaf = useCallback((name: string, type: LeafType) => {
    if (!navigation.currentBranchId) return;
    dispatch({ type: 'NAV_CREATE_LEAF', payload: {
      branchId: navigation.currentBranchId,
      name,
      type
    }});
  }, [dispatch, navigation.currentBranchId]);

  const deleteLeaf = useCallback((id: string) => {
    dispatch({ type: 'NAV_DELETE_LEAF', payload: id });
  }, [dispatch]);

  // Recent trees
  const recentTrees = useMemo(() => {
    const allTrees = navigation.trunks.flatMap(t => t.trees);
    return navigation.recentTreeIds
      .map(id => allTrees.find(t => t.id === id))
      .filter((t): t is Tree => t !== undefined);
  }, [navigation.trunks, navigation.recentTreeIds]);

  return {
    currentTrunk,
    currentTree,
    currentBranch,
    currentLeaf,
    trunks: navigation.trunks,
    recentTrees,
    setTrunk,
    setTree,
    setBranch,
    setLeaf,
    createTrunk,
    deleteTrunk,
    createTree,
    deleteTree,
    createBranch,
    deleteBranch,
    createLeaf,
    deleteLeaf,
```

```
  };
}
```

```
// src/hooks/useUI.ts

import { useCallback } from 'react';
import { useAppContext } from '../contexts/AppContext';
import { UIState } from '../types/state';

export interface UseUIReturn {
  // State
  sidebarOpen: boolean;
  sidebarWidth: number;
  activePanel: UIState['activePanel'];
  theme: UIState['theme'];
  resolvedTheme: UIState['resolvedTheme'];
  modals: UIState['modals'];
  notifications: UIState['notifications'];

  // Actions
  toggleSidebar: () => void;
  setSidebarWidth: (width: number) => void;
  setActivePanel: (panel: UIState['activePanel']) => void;
  setTheme: (theme: UIState['theme']) => void;
  openModal: (modal: keyof UIState['modals']) => void;
  closeModal: (modal: keyof UIState['modals']) => void;
  notify: (notification: { type: 'info' | 'success' | 'warning' | '
      error'; message: string; duration?: number }) => void;
  dismissNotification: (id: string) => void;
}

export function useUI(): UseUIReturn {
  const { state, dispatch } = useAppContext();
  const { ui } = state;

  const toggleSidebar = useCallback(() => {
    dispatch({ type: 'UI_TOGGLE_SIDEBAR' });
  }, [dispatch]);

  const setSidebarWidth = useCallback((width: number) => {
    dispatch({ type: 'UI_SET_SIDEBAR_WIDTH', payload: width });
  }, [dispatch]);

  const setActivePanel = useCallback((panel: UIState['activePanel'])
     => {
    dispatch({ type: 'UI_SET_ACTIVE_PANEL', payload: panel });
  }, [dispatch]);

  const setTheme = useCallback((theme: UIState['theme']) => {
    dispatch({ type: 'UI_SET_THEME', payload: theme });
  }, [dispatch]);

  const openModal = useCallback((modal: keyof UIState['modals']) => {
    dispatch({ type: 'UI_OPEN_MODAL', payload: modal });
  }, [dispatch]);

  const closeModal = useCallback((modal: keyof UIState['modals']) =>
```

```
      {
    dispatch({ type: 'UI_CLOSE_MODAL', payload: modal });
  }, [dispatch]);

  const notify = useCallback((notification: {
    type: 'info' | 'success' | 'warning' | 'error';
    message: string;
    duration?: number
  }) => {
    dispatch({
      type: 'UI_ADD_NOTIFICATION',
      payload: { ...notification, dismissible: true }
    });
  }, [dispatch]);

  const dismissNotification = useCallback((id: string) => {
    dispatch({ type: 'UI_DISMISS_NOTIFICATION', payload: id });
  }, [dispatch]);

  return {
    sidebarOpen: ui.sidebarOpen,
    sidebarWidth: ui.sidebarWidth,
    activePanel: ui.activePanel,
    theme: ui.theme,
    resolvedTheme: ui.resolvedTheme,
    modals: ui.modals,
    notifications: ui.notifications,
    toggleSidebar,
    setSidebarWidth,
    setActivePanel,
    setTheme,
    openModal,
    closeModal,
    notify,
    dismissNotification,
  };
}
```

```
// src/hooks/useLayers.ts

import { useCallback, useMemo } from 'react';
import { useAppContext } from '../contexts/AppContext';
import { Layer, LayerType } from '../types/layers';
import { buildLayerTree, LayerTreeNode } from '../utils/layerTree';

export interface UseLayersReturn {
  // Data
  layers: Layer[];
  layerTree: LayerTreeNode[];
  selectedIds: string[];
  expandedIds: string[];
  selectedLayers: Layer[];

  // Selection
  selectLayers: (ids: string[], mode?: 'replace' | 'add' | 'toggle')
     => void;
  clearSelection: () => void;
```

```typescript
  selectAll: () => void;

  // Expansion
  toggleExpand: (id: string) => void;
  expandAll: () => void;
  collapseAll: () => void;

  // CRUD
  createLayer: (type: LayerType, parentId?: string) => void;
  deleteSelected: () => void;
  duplicateSelected: () => void;

  // Modification
  renameLayer: (id: string, name: string) => void;
  toggleVisibility: (id: string) => void;
  toggleLock: (id: string) => void;
  reorderLayer: (id: string, targetId: string, position: 'before' | 'after' | 'inside') => void;

  // Grouping
  groupSelected: () => void;
  ungroupLayer: (id: string) => void;
}

export function useLayers(): UseLayersReturn {
  const { state, dispatch } = useAppContext();

  const layers = state.editor?.leaf.layers ?? [];
  const selectedIds = state.editor?.selectedLayerIds ?? [];
  const expandedIds = state.editor?.expandedLayerIds ?? [];

  const layerTree = useMemo(() => buildLayerTree(layers), [layers]);

  const selectedLayers = useMemo(() =>
    layers.filter(l => selectedIds.includes(l.id)),
    [layers, selectedIds]
  );

  // Selection
  const selectLayers = useCallback((ids: string[], mode: 'replace' | 'add' | 'toggle' = 'replace') => {
    dispatch({ type: 'EDITOR_SELECT_LAYERS', payload: { ids, mode } });
  }, [dispatch]);

  const clearSelection = useCallback(() => {
    dispatch({ type: 'EDITOR_CLEAR_SELECTION' });
  }, [dispatch]);

  const selectAll = useCallback(() => {
    dispatch({ type: 'EDITOR_SELECT_LAYERS', payload: { ids: layers.map(l => l.id), mode: 'replace' } });
  }, [dispatch, layers]);

  // Expansion
  const toggleExpand = useCallback((id: string) => {
    dispatch({ type: 'EDITOR_TOGGLE_LAYER_EXPAND', payload: id });
  }, [dispatch]);
```

```typescript

  const expandAll = useCallback(() => {
    const groupIds = layers.filter(l => l.childIds.length > 0).map(l => l.id);
    groupIds.forEach(id => {
      if (!expandedIds.includes(id)) {
        dispatch({ type: 'EDITOR_TOGGLE_LAYER_EXPAND', payload: id });
      }
    });
  }, [dispatch, layers, expandedIds]);

  const collapseAll = useCallback(() => {
    expandedIds.forEach(id => {
      dispatch({ type: 'EDITOR_TOGGLE_LAYER_EXPAND', payload: id });
    });
  }, [dispatch, expandedIds]);

  // CRUD
  const createLayer = useCallback((type: LayerType, parentId?: string) => {
    dispatch({ type: 'LAYER_CREATE', payload: { type, parentId } });
  }, [dispatch]);

  const deleteSelected = useCallback(() => {
    if (selectedIds.length > 0) {
      dispatch({ type: 'LAYER_DELETE', payload: selectedIds });
    }
  }, [dispatch, selectedIds]);

  const duplicateSelected = useCallback(() => {
    if (selectedIds.length > 0) {
      dispatch({ type: 'LAYER_DUPLICATE', payload: selectedIds });
    }
  }, [dispatch, selectedIds]);

  // Modification
  const renameLayer = useCallback((id: string, name: string) => {
    dispatch({ type: 'LAYER_RENAME', payload: { id, name } });
  }, [dispatch]);

  const toggleVisibility = useCallback((id: string) => {
    dispatch({ type: 'LAYER_TOGGLE_VISIBILITY', payload: id });
  }, [dispatch]);

  const toggleLock = useCallback((id: string) => {
    dispatch({ type: 'LAYER_TOGGLE_LOCK', payload: id });
  }, [dispatch]);

  const reorderLayer = useCallback((
    id: string,
    targetId: string,
    position: 'before' | 'after' | 'inside'
  ) => {
    dispatch({ type: 'LAYER_REORDER', payload: { id, targetId, position } });
  }, [dispatch]);

```

```
  // Grouping
  const groupSelected = useCallback(() => {
    if (selectedIds.length > 1) {
      dispatch({ type: 'LAYER_GROUP', payload: selectedIds });
    }
  }, [dispatch, selectedIds]);

  const ungroupLayer = useCallback((id: string) => {
    dispatch({ type: 'LAYER_UNGROUP', payload: id });
  }, [dispatch]);

  return {
    layers,
    layerTree,
    selectedIds,
    expandedIds,
    selectedLayers,
    selectLayers,
    clearSelection,
    selectAll,
    toggleExpand,
    expandAll,
    collapseAll,
    createLayer,
    deleteSelected,
    duplicateSelected,
    renameLayer,
    toggleVisibility,
    toggleLock,
    reorderLayer,
    groupSelected,
    ungroupLayer,
  };
}
```

Part IV
UI Components

6 Ribbon Component

```
// src/components/Ribbon/Ribbon.tsx

import { useState, useCallback, MouseEvent } from 'react';
import { useUI } from '../../hooks/useUI';
import { usePlugins } from '../../hooks/usePlugins';
import { RibbonIcon } from './RibbonIcon';
import { RibbonContextMenu } from './RibbonContextMenu';
import {
  PanelLeftClose,
  PanelLeftOpen,
  Layers,
  Search,
  GitBranch,
```

```typescript
  Package,
  History,
  Settings,
  HelpCircle,
} from 'lucide-react';
import styles from './Ribbon.module.css';

interface RibbonAction {
  id: string;
  icon: React.ComponentType<{ size?: number }>;
  tooltip: string;
  onClick: () => void;
  isActive?: boolean;
}

export function Ribbon(): JSX.Element {
  const {
    sidebarOpen,
    toggleSidebar,
    activePanel,
    setActivePanel,
    openModal
  } = useUI();
  const { pluginRibbonActions } = usePlugins();

  const [contextMenu, setContextMenu] = useState<{ x: number; y:
     number } | null>(null);
  const [hiddenActions, setHiddenActions] = useState<Set<string>>(new
      Set());

  // Core navigation actions
  const coreActions: RibbonAction[] = [
    {
      id: 'layers',
      icon: Layers,
      tooltip: 'Layers (L)',
      onClick: () => setActivePanel('layers'),
      isActive: activePanel === 'layers',
    },
    {
      id: 'assets',
      icon: Package,
      tooltip: 'Assets',
      onClick: () => setActivePanel('assets'),
      isActive: activePanel === 'assets',
    },
    {
      id: 'components',
      icon: GitBranch,
      tooltip: 'Components',
      onClick: () => setActivePanel('components'),
      isActive: activePanel === 'components',
    },
    {
      id: 'history',
      icon: History,
      tooltip: 'Version History',
      onClick: () => setActivePanel('history'),
```

```
      isActive: activePanel === 'history',
    },
    {
      id: 'search',
      icon: Search,
      tooltip: 'Search (Ctrl+F)',
      onClick: () => {/* Open search */},
    },
  ];

  // System actions (always at bottom)
  const systemActions: RibbonAction[] = [
    {
      id: 'help',
      icon: HelpCircle,
      tooltip: 'Help & Support',
      onClick: () => openModal('help'),
    },
    {
      id: 'settings',
      icon: Settings,
      tooltip: 'Settings (Ctrl+,)',
      onClick: () => openModal('settings'),
    },
  ];

  // Combine with plugin actions
  const allTopActions = [
    ...coreActions.filter(a => !hiddenActions.has(a.id)),
    ...pluginRibbonActions.filter(a => !hiddenActions.has(a.id)),
  ];

  const handleContextMenu = useCallback((e: MouseEvent) => {
    e.preventDefault();
    setContextMenu({ x: e.clientX, y: e.clientY });
  }, []);

  const toggleActionVisibility = useCallback((id: string) => {
    setHiddenActions(prev => {
      const next = new Set(prev);
      if (next.has(id)) {
        next.delete(id);
      } else {
        next.add(id);
      }
      return next;
    });
  }, []);

  return (
    <nav
      className={styles.ribbon}
      onContextMenu={handleContextMenu}
      aria-label="Main navigation"
    >
      {/* Sidebar toggle */}
      <div className={styles.toggleSection}>
        <RibbonIcon
```

```
            icon={sidebarOpen ? PanelLeftClose : PanelLeftOpen}
            tooltip={sidebarOpen ? 'Collapse sidebar' : 'Expand sidebar'}
            onClick={toggleSidebar}
          />
        </div>

        {/* Top section - customizable actions */}
        <div className={styles.topSection}>
          {allTopActions.map(action => (
            <RibbonIcon
              key={action.id}
              icon={action.icon}
              tooltip={action.tooltip}
              onClick={action.onClick}
              isActive={action.isActive}
            />
          ))}
        </div>

        {/* Spacer */}
        <div className={styles.spacer} />

        {/* Bottom section - system actions */}
        <div className={styles.bottomSection}>
          {systemActions.map(action => (
            <RibbonIcon
              key={action.id}
              icon={action.icon}
              tooltip={action.tooltip}
              onClick={action.onClick}
            />
          ))}
        </div>

        {/* Context menu */}
        {contextMenu && (
          <RibbonContextMenu
            position={contextMenu}
            actions={[...coreActions, ...pluginRibbonActions]}
            hiddenIds={hiddenActions}
            onToggle={toggleActionVisibility}
            onClose={() => setContextMenu(null)}
          />
        )}
      </nav>
    );
}
```

```
// src/components/Ribbon/RibbonIcon.tsx

import { useState, useRef, ComponentType, useCallback } from 'react';
import { Tooltip } from '../ui/Tooltip';
import styles from './RibbonIcon.module.css';

interface RibbonIconProps {
  icon: ComponentType<{ size?: number; className?: string }>;
```

```
  tooltip: string;
  onClick: () => void;
  isActive?: boolean;
  disabled?: boolean;
}

export function RibbonIcon({
  icon: Icon,
  tooltip,
  onClick,
  isActive = false,
  disabled = false,
}: RibbonIconProps): JSX.Element {
  const [showTooltip, setShowTooltip] = useState(false);
  const buttonRef = useRef<HTMLButtonElement>(null);

  const handleMouseEnter = useCallback(() => setShowTooltip(true),
     []);
  const handleMouseLeave = useCallback(() => setShowTooltip(false),
     []);

  return (
    <div className={styles.container}>
      <button
        ref={buttonRef}
        type="button"
        className={`${styles.button} ${isActive ? styles.active : ''}
           `}
        onClick={onClick}
        disabled={disabled}
        onMouseEnter={handleMouseEnter}
        onMouseLeave={handleMouseLeave}
        aria-label={tooltip}
        aria-pressed={isActive}
      >
        <Icon size={20} className={styles.icon} />
        {isActive && <span className={styles.indicator} aria-hidden="
           true" />}
      </button>

      <Tooltip
        visible={showTooltip && !disabled}
        targetRef={buttonRef}
        position="right"
      >
        {tooltip}
      </Tooltip>
    </div>
  );
}
```

7 Sidebar Components

7.1 Trunk Selector

```tsx
// src/components/Sidebar/TrunkSelector.tsx

import { useState, useRef, useEffect, useCallback } from 'react';
import { useNavigation } from '../../hooks/useNavigation';
import { useUI } from '../../hooks/useUI';
import {
  ChevronDown,
  Plus,
  FolderTree,
  Cloud,
  Settings2,
  Check
} from 'lucide-react';
import styles from './TrunkSelector.module.css';

export function TrunkSelector(): JSX.Element {
  const {
    trunks,
    currentTrunk,
    setTrunk,
    createTrunk
  } = useNavigation();
  const { openModal } = useUI();

  const [isOpen, setIsOpen] = useState(false);
  const dropdownRef = useRef<HTMLDivElement>(null);

  // Close on outside click
  useEffect(() => {
    function handleClickOutside(event: MouseEvent): void {
      if (dropdownRef.current && !dropdownRef.current.contains(event.
          target as Node)) {
        setIsOpen(false);
      }
    }
    document.addEventListener('mousedown', handleClickOutside);
    return () => document.removeEventListener('mousedown',
        handleClickOutside);
  }, []);

  const handleSelect = useCallback((id: string) => {
    setTrunk(id);
    setIsOpen(false);
  }, [setTrunk]);

  const handleCreate = useCallback(() => {
    const name = prompt('Trunk name:');
    if (name?.trim()) {
      createTrunk(name.trim());
    }
    setIsOpen(false);
  }, [createTrunk]);

  const handleManage = useCallback(() => {
    setIsOpen(false);
    openModal('trunkManager');
  }, [openModal]);

```

```
  return (
    <div className={styles.container} ref={dropdownRef}>
      <button
        type="button"
        className={styles.selector}
        onClick={() => setIsOpen(!isOpen)}
        aria-expanded={isOpen}
        aria-haspopup="listbox"
      >
        <div className={styles.trunkInfo}>
          <FolderTree size={16} className={styles.icon} />
          <span className={styles.name}>
            {currentTrunk?.name ?? 'Select Trunk'}
          </span>
        </div>
        <ChevronDown
          size={16}
          className={`${styles.chevron} ${isOpen ? styles.open : ''}`
        />
      </button>

      {isOpen && (
        <div className={styles.dropdown} role="listbox">
          {/* Trunk list */}
          {trunks.length > 0 && (
            <>
              <div className={styles.sectionLabel}>Your Trunks</div>
              {trunks.map(trunk => (
                <button
                  key={trunk.id}
                  type="button"
                  className={`${styles.option} ${trunk.id ===
                    currentTrunk?.id ? styles.active : ''}`}
                  onClick={() => handleSelect(trunk.id)}
                  role="option"
                  aria-selected={trunk.id === currentTrunk?.id}
                >
                  {trunk.settings.syncEnabled ? (
                    <Cloud size={14} />
                  ) : (
                    <FolderTree size={14} />
                  )}
                  <span className={styles.optionName}>{trunk.name}</
                    span>
                  {trunk.id === currentTrunk?.id && (
                    <Check size={14} className={styles.checkIcon} />
                  )}
                </button>
              ))}
              <div className={styles.divider} />
            </>
          )}

          {/* Actions */}
          <button type="button" className={styles.option} onClick={
            handleCreate}>
            <Plus size={14} />
```

```
              <span>New Trunk</span>
            </button>
            <button type="button" className={styles.option} onClick={
                handleManage}>
              <Settings2 size={14} />
              <span>Manage Trunks...</span>
            </button>
          </div>
        )}
      </div>
    );
}
```

7.2 Tree & Branch Selector

```
// src/components/Sidebar/TreeBranchSelector.tsx

import { useState, useCallback } from 'react';
import { useNavigation } from '../../hooks/useNavigation';
import { useUI } from '../../hooks/useUI';
import {
  ChevronDown,
  TreeDeciduous,
  GitBranch,
  Plus,
  GitMerge,
  Trash2
} from 'lucide-react';
import styles from './TreeBranchSelector.module.css';

export function TreeBranchSelector(): JSX.Element {
  const {
    currentTrunk,
    currentTree,
    currentBranch,
    setTree,
    setBranch,
    createTree,
    createBranch,
    deleteBranch
  } = useNavigation();
  const { openModal, notify } = useUI();

  const [treeDropdownOpen, setTreeDropdownOpen] = useState(false);
  const [branchDropdownOpen, setBranchDropdownOpen] = useState(false)
      ;

  const trees = currentTrunk?.trees ?? [];
  const branches = currentTree?.branches ?? [];

  const handleTreeSelect = useCallback((id: string) => {
    setTree(id);
    setTreeDropdownOpen(false);
  }, [setTree]);

  const handleBranchSelect = useCallback((id: string) => {
```

```
    setBranch(id);
    setBranchDropdownOpen(false);
  }, [setBranch]);

  const handleCreateTree = useCallback(() => {
    const name = prompt('Tree name:');
    if (name?.trim()) {
      createTree(name.trim());
      notify({ type: 'success', message: `Created Tree: ${name}` });
    }
    setTreeDropdownOpen(false);
  }, [createTree, notify]);

  const handleCreateBranch = useCallback(() => {
    const name = prompt('Branch name:');
    if (name?.trim()) {
      createBranch(name.trim());
      notify({ type: 'success', message: `Created Branch: ${name}` })
        ;
    }
    setBranchDropdownOpen(false);
  }, [createBranch, notify]);

  const handleDeleteBranch = useCallback((id: string, e: React.
    MouseEvent) => {
    e.stopPropagation();
    const branch = branches.find(b => b.id === id);
    if (branch?.isProtected) {
      notify({ type: 'error', message: 'Cannot delete protected
        branch' });
      return;
    }
    if (confirm(`Delete branch "${branch?.name}"?`)) {
      deleteBranch(id);
      notify({ type: 'success', message: 'Branch deleted' });
    }
  }, [branches, deleteBranch, notify]);

  if (!currentTrunk) {
    return (
      <div className={styles.placeholder}>
        Select a Trunk to continue
      </div>
    );
  }

  return (
    <div className={styles.container}>
      {/* Tree selector */}
      <div className={styles.selectorGroup}>
        <button
          type="button"
          className={styles.selector}
          onClick={() => setTreeDropdownOpen(!treeDropdownOpen)}
          aria-expanded={treeDropdownOpen}
        >
          <TreeDeciduous size={14} className={styles.icon} />
          <span className={styles.label}>
```

```
            {currentTree?.name ?? 'Select Tree'}
          </span>
          <ChevronDown size={14} className={styles.chevron} />
        </button>

        {treeDropdownOpen && (
          <div className={styles.dropdown}>
            {trees.map(tree => (
              <button
                key={tree.id}
                type="button"
                className={`${styles.option} ${tree.id ===
                   currentTree?.id ? styles.active : ''}`}
                onClick={() => handleTreeSelect(tree.id)}
              >
                <TreeDeciduous size={12} />
                <span>{tree.name}</span>
              </button>
            ))}
            <div className={styles.divider} />
            <button type="button" className={styles.option} onClick={
               handleCreateTree}>
              <Plus size={12} />
              <span>New Tree</span>
            </button>
          </div>
        )}
      </div>

      {/* Branch selector */}
      {currentTree && (
        <div className={styles.selectorGroup}>
          <button
            type="button"
            className={styles.selector}
            onClick={() => setBranchDropdownOpen(!branchDropdownOpen)
               }
            aria-expanded={branchDropdownOpen}
          >
            <GitBranch size={14} className={styles.icon} />
            <span className={styles.label}>
              {currentBranch?.name ?? 'Select Branch'}
            </span>
            <ChevronDown size={14} className={styles.chevron} />
          </button>

          {branchDropdownOpen && (
            <div className={styles.dropdown}>
              {branches.map(branch => (
                <button
                  key={branch.id}
                  type="button"
                  className={`${styles.option} ${branch.id ===
                     currentBranch?.id ? styles.active : ''}`}
                  onClick={() => handleBranchSelect(branch.id)}
                >
                  <GitBranch size={12} />
                  <span>{branch.name}</span>
```

```
                    {branch.isProtected && (
                      <span className={styles.badge}>protected</span>
                    )}
                    {!branch.isProtected && (
                      <button
                        type="button"
                        className={styles.deleteBtn}
                        onClick={(e) => handleDeleteBranch(branch.id, e
                           )}
                        aria-label={`Delete ${branch.name}`}
                      >
                        <Trash2 size={12} />
                      </button>
                    )}
                  </button>
                ))}
                <div className={styles.divider} />
                <button type="button" className={styles.option} onClick
                   ={handleCreateBranch}>
                  <Plus size={12} />
                  <span>New Branch</span>
                </button>
                <button
                  type="button"
                  className={styles.option}
                  onClick={() => openModal('branchManager')}
                >
                  <GitMerge size={12} />
                  <span>Merge Branches...</span>
                </button>
              </div>
            )}
          </div>
        )}
      </div>
    );
  }
```

7.3 Layers Panel

```
// src/components/Sidebar/LayersPanel/LayersPanel.tsx

import { useCallback } from 'react';
import { useLayers } from '../../../hooks/useLayers';
import { LayerTreeItem } from './LayerTreeItem';
import { LayerTreeNode } from '../../../utils/layerTree';
import styles from './LayersPanel.module.css';

export function LayersPanel(): JSX.Element {
  const {
    layerTree,
    selectedIds,
    expandedIds,
    selectLayers,
    toggleExpand,
    toggleVisibility,
```

```
    toggleLock,
    renameLayer,
    reorderLayer,
  } = useLayers();

  const handleSelect = useCallback((
    id: string,
    event: React.MouseEvent
  ) => {
    const mode = event.ctrlKey || event.metaKey
      ? 'toggle'
      : event.shiftKey
        ? 'add'
        : 'replace';
    selectLayers([id], mode);
  }, [selectLayers]);

  const renderNode = useCallback((node: LayerTreeNode, depth: number)
      => {
    const isSelected = selectedIds.includes(node.layer.id);
    const isExpanded = expandedIds.includes(node.layer.id);
    const hasChildren = node.children.length > 0;

    return (
      <div key={node.layer.id} role="treeitem" aria-selected={
          isSelected}>
        <LayerTreeItem
          layer={node.layer}
          depth={depth}
          isSelected={isSelected}
          isExpanded={isExpanded}
          hasChildren={hasChildren}
          onSelect={handleSelect}
          onToggleExpand={toggleExpand}
          onToggleVisibility={toggleVisibility}
          onToggleLock={toggleLock}
          onRename={renameLayer}
          onReorder={reorderLayer}
        />

        {hasChildren && isExpanded && (
          <div role="group" className={styles.children}>
            {node.children.map(child => renderNode(child, depth + 1))
                }
          </div>
        )}
      </div>
    );
  }, [
    selectedIds,
    expandedIds,
    handleSelect,
    toggleExpand,
    toggleVisibility,
    toggleLock,
    renameLayer,
    reorderLayer
  ]);
```

```
72
73    return (
74      <div className={styles.panel}>
75        <div className={styles.header}>
76          <span className={styles.title}>Layers</span>
77          <span className={styles.count}>
78            {layerTree.reduce((acc, n) => acc + countNodes(n), 0)}
79          </span>
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(
        child), 0);
101 }
```

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';
```

```typescript
  showCloseButton?: boolean;
  closeOnOverlay?: boolean;
  closeOnEscape?: boolean;
}

export function Modal({
  isOpen,
  onClose,
  title,
  children,
  size = 'medium',
  showCloseButton = true,
  closeOnOverlay = true,
  closeOnEscape = true,
}: ModalProps): JSX.Element | null {
  const modalRef = useRef<HTMLDivElement>(null);
  const previousActiveElement = useRef<HTMLElement | null>(null);

  // Escape key handler
  useEffect(() => {
    if (!isOpen || !closeOnEscape) return;

    function handleKeyDown(e: globalThis.KeyboardEvent): void {
      if (e.key === 'Escape') {
        onClose();
      }
    }

    document.addEventListener('keydown', handleKeyDown);
    return () => document.removeEventListener('keydown',
       handleKeyDown);
  }, [isOpen, closeOnEscape, onClose]);

  // Focus management and scroll lock
  useEffect(() => {
    if (isOpen) {
      previousActiveElement.current = document.activeElement as
         HTMLElement;
      document.body.style.overflow = 'hidden';
      modalRef.current?.focus();
    } else {
      document.body.style.overflow = '';
      previousActiveElement.current?.focus();
    }

    return () => {
      document.body.style.overflow = '';
    };
  }, [isOpen]);

  const handleOverlayClick = useCallback((e: React.MouseEvent) => {
    if (closeOnOverlay && e.target === e.currentTarget) {
      onClose();
    }
  }, [closeOnOverlay, onClose]);

  const handleKeyDown = useCallback((e: KeyboardEvent<HTMLDivElement
     >) => {
```

```
    // Focus trap
    if (e.key === 'Tab') {
      const focusable = modalRef.current?.querySelectorAll<
          HTMLElement>(
        'button, [href], input, select, textarea, [tabindex]:not([
            tabindex="-1"])'
      );

      if (!focusable || focusable.length === 0) return;

      const first = focusable[0];
      const last = focusable[focusable.length - 1];

      if (e.shiftKey && document.activeElement === first) {
        e.preventDefault();
        last.focus();
      } else if (!e.shiftKey && document.activeElement === last) {
        e.preventDefault();
        first.focus();
      }
    }
  }, []);

  if (!isOpen) return null;

  const modalContent = (
    <div
      className={styles.overlay}
      onClick={handleOverlayClick}
      aria-modal="true"
      role="dialog"
      aria-labelledby="modal-title"
    >
      <div
        ref={modalRef}
        className={`${styles.modal} ${styles[size]}`}
        tabIndex={-1}
        onKeyDown={handleKeyDown}
      >
        <header className={styles.header}>
          <h2 id="modal-title" className={styles.title}>{title}</h2>
          {showCloseButton && (
            <button
              type="button"
              className={styles.closeButton}
              onClick={onClose}
              aria-label="Close modal"
            >
              <X size={18} />
            </button>
          )}
        </header>

        <div className={styles.content}>
          {children}
        </div>
      </div>
    </div>
```

```
  );

  // Render to portal
  const portalRoot = document.getElementById('modal-root') ??
      document.body;
  return createPortal(modalContent, portalRoot);
}
```

Part V
Plugin System

9 Plugin Architecture

9.1 Plugin Types

```
// src/types/plugin.ts

import { ComponentType } from 'react';
import { Layer, LayerType } from './layers';

/**
 * Plugin manifest - metadata loaded from plugin.json
 */
export interface PluginManifest {
  id: string;
  name: string;
  version: string;
  description: string;
  author: string;
  authorUrl?: string;
  minAppVersion: string;
  main: string;
  repository?: string;
  license?: string;
}

/**
 * Plugin lifecycle interface
 */
export interface Plugin {
  onload(): void | Promise<void>;
  onunload(): void | Promise<void>;
}

/**
 * Plugin API - provided to plugins for app integration
 */
export interface PluginAPI {
  // Ribbon
  addRibbonIcon(
    id: string,
    icon: ComponentType<{ size?: number }>,
    tooltip: string,
```

```typescript
    callback: () => void
  ): () => void;

  // Toolbar
  addToolbarAction(
    id: string,
    icon: ComponentType<{ size?: number }>,
    tooltip: string,
    callback: () => void,
    options?: ToolbarActionOptions
  ): () => void;

  // Commands
  registerCommand(command: Command): () => void;
  executeCommand(id: string): void;

  // Settings
  registerSettingTab(tab: SettingTab): () => void;

  // Storage
  loadData<T = unknown>(): Promise<T | null>;
  saveData<T = unknown>(data: T): Promise<void>;

  // Events
  on<K extends keyof PluginEvents>(
    event: K,
    callback: PluginEvents[K]
  ): () => void;

  // Layer operations
  getSelectedLayers(): Layer[];
  selectLayers(ids: string[]): void;
  createLayer(type: LayerType, props?: Partial<Layer>): string;
  updateLayer(id: string, changes: Partial<Layer>): void;
  deleteLayer(id: string): void;

  // UI
  showModal(component: ComponentType<{ onClose: () => void }>): void;
  showNotice(message: string, type?: 'info' | 'success' | 'warning' |
      'error'): void;
  showConfirm(message: string): Promise<boolean>;
}

export interface ToolbarActionOptions {
  position?: number;
  disabled?: boolean;
  separator?: 'before' | 'after';
}

export interface Command {
  id: string;
  name: string;
  description?: string;
  hotkeys?: string[];
  callback: () => void;
  checkCallback?: () => boolean; // For conditional enablement
}

```

```typescript
export interface SettingTab {
  id: string;
  name: string;
  icon: ComponentType<{ size?: number }>;
  component: ComponentType;
}

export interface PluginEvents {
  'layer:created': (layer: Layer) => void;
  'layer:updated': (layer: Layer, changes: Partial<Layer>) => void;
  'layer:deleted': (id: string) => void;
  'selection:changed': (ids: string[]) => void;
  'leaf:opened': (leafId: string) => void;
  'leaf:closed': () => void;
  'branch:switched': (branchId: string) => void;
  'tree:switched': (treeId: string) => void;
}
```

9.2 Plugin Context

```typescript
// src/contexts/PluginContext.tsx

import {
  createContext,
  useContext,
  useState,
  useCallback,
  useEffect,
  ReactNode,
  ComponentType
} from 'react';
import { Plugin, PluginManifest, PluginAPI, Command, SettingTab }
    from '../types/plugin';
import { useAppContext } from './AppContext';
import { Layer, LayerType } from '../types/layers';

interface RibbonAction {
  id: string;
  pluginId: string;
  icon: ComponentType<{ size?: number }>;
  tooltip: string;
  onClick: () => void;
}

interface ToolbarAction {
  id: string;
  pluginId: string;
  icon: ComponentType<{ size?: number }>;
  tooltip: string;
  onClick: () => void;
  disabled?: boolean;
  position?: number;
}

interface PluginInstance {
  manifest: PluginManifest;
```

```
  instance: Plugin;
}

interface PluginContextValue {
  plugins: Map<string, PluginInstance>;
  pluginRibbonActions: RibbonAction[];
  pluginToolbarActions: ToolbarAction[];
  pluginCommands: Command[];
  pluginSettingTabs: SettingTab[];
  loadPlugin: (manifest: PluginManifest) => Promise<void>;
  unloadPlugin: (id: string) => Promise<void>;
  isPluginEnabled: (id: string) => boolean;
}

const PluginContext = createContext<PluginContextValue | null>(null);

interface PluginProviderProps {
  children: ReactNode;
}

export function PluginProvider({ children }: PluginProviderProps):
    JSX.Element {
  const { state, dispatch } = useAppContext();

  const [plugins] = useState(() => new Map<string, PluginInstance>())
     ;
  const [ribbonActions, setRibbonActions] = useState<RibbonAction
     []>([]);
  const [toolbarActions, setToolbarActions] = useState<ToolbarAction
     []>([]);
  const [commands, setCommands] = useState<Command[]>([]);
  const [settingTabs, setSettingTabs] = useState<SettingTab[]>([]);

  // Create API for a specific plugin
  const createPluginAPI = useCallback((pluginId: string): PluginAPI
     => {
    const fullId = (id: string) => `${pluginId}:${id}`;

    return {
      // Ribbon
      addRibbonIcon(id, icon, tooltip, callback) {
        const actionId = fullId(id);
        const action: RibbonAction = {
          id: actionId,
          pluginId,
          icon,
          tooltip,
          onClick: callback
        };
        setRibbonActions(prev => [...prev, action]);
        return () => setRibbonActions(prev => prev.filter(a => a.id
           !== actionId));
      },

      // Toolbar
      addToolbarAction(id, icon, tooltip, callback, options) {
        const actionId = fullId(id);
        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
                !== actionId));
104       },
105
106       // Commands
107       registerCommand(command) {
108         const cmd = { ...command, id: fullId(command.id) };
109         setCommands(prev => [...prev, cmd]);
110         return () => setCommands(prev => prev.filter(c => c.id !==
                cmd.id));
111       },
112
113       executeCommand(id) {
114         const cmd = commands.find(c => c.id === fullId(id));
115         if (cmd && (!cmd.checkCallback || cmd.checkCallback())) {
116           cmd.callback();
117         }
118       },
119
120       // Settings
121       registerSettingTab(tab) {
122         const t = { ...tab, id: fullId(tab.id) };
123         setSettingTabs(prev => [...prev, t]);
124         return () => setSettingTabs(prev => prev.filter(st => st.id
                !== t.id));
125       },
126
127       // Storage
128       async loadData<T>() {
129         const key = `plugin:${pluginId}:data`;
130         try {
131           const data = localStorage.getItem(key);
132           return data ? JSON.parse(data) as T : null;
133         } catch {
134           return null;
135         }
136       },
137
138       async saveData<T>(data: T) {
139         const key = `plugin:${pluginId}:data`;
140         localStorage.setItem(key, JSON.stringify(data));
141       },
```

```
142
143       // Events
144       on(event, callback) {
145         // Event system implementation
146         return () => {};
147       },
148
149       // Layer operations
150       getSelectedLayers() {
151         return state.editor?.leaf.layers.filter(
152           l => state.editor?.selectedLayerIds.includes(l.id)
153         ) ?? [];
154       },
155
156       selectLayers(ids) {
157         dispatch({ type: 'EDITOR_SELECT_LAYERS', payload: { ids, mode
                : '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     if (plugins.has(manifest.id)) {
195       console.warn(`Plugin ${manifest.id} already loaded`);
```

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

```
    pluginSettingTabs: settingTabs,
    loadPlugin,
    unloadPlugin,
    isPluginEnabled,
  };

  return (
    <PluginContext.Provider value={value}>
      {children}
    </PluginContext.Provider>
  );
}

export function usePlugins(): PluginContextValue {
  const context = useContext(PluginContext);
  if (!context) {
    throw new Error('usePlugins must be used within PluginProvider');
  }
  return context;
}
```

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 AppContext 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

```
src/
        components/
            AppShell/
                AppShell.tsx
                AppShell.module.css
            Ribbon/
                Ribbon.tsx
                Ribbon.module.css
                RibbonIcon.tsx
                RibbonIcon.module.css
                RibbonContextMenu.tsx
            Sidebar/
                Sidebar.tsx
                Sidebar.module.css
                TrunkSelector.tsx
                TrunkSelector.module.css
                TreeBranchSelector.tsx
                TreeBranchSelector.module.css
                Toolbar.tsx
                Toolbar.module.css
                LayersPanel/
                    LayersPanel.tsx
                    LayersPanel.module.css
                    LayerTreeItem.tsx
                    LayerTreeItem.module.css
            Canvas/
                Canvas.tsx
            Modal/
                Modal.tsx
                Modal.module.css
                SettingsModal/
                HelpModal/
                PaymentModal/
                TrunkManagerModal/
                BranchManagerModal/
            ui/
                Tooltip.tsx
                Select.tsx
                Switch.tsx
                Slider.tsx
                Button.tsx
                Input.tsx
        contexts/
            AppContext.tsx
            PluginContext.tsx
        hooks/
            useNavigation.ts
            useUI.ts
            useLayers.ts
            usePlugins.ts
            useKeyboard.ts
```

```
        reducers/
            appReducer.ts
            navigationReducer.ts
            uiReducer.ts
            editorReducer.ts
            initialState.ts
        types/
            arborist.ts
            layers.ts
            state.ts
            actions.ts
            plugin.ts
        utils/
            layerTree.ts
            ids.ts
            storage.ts
        styles/
            variables.css
            reset.css
            global.css
        plugins/
            example-plugin/
                manifest.json
                main.ts
        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.
