GIS

ArcGIS Core SDK - Technical Documentation

Comprehensive Technical Guide & Data Flow Architecture

v4.33.6

Table of Contents

System Architecture

High-Level System Architecture

Presentation Layer
React + TypeScript
ArcGIS Core SDK
Tailwind CSS

Business Logic
Zustand Stores
Context Providers
Event Emitter

Data Services
ArcGIS Online
External APIs
WebSocket Services

Core Components

Component	Purpose	Technology
MapView	Main map rendering and interaction	ArcGIS Core SDK
Layer Management	Dynamic layer addition/removal	Custom Hooks + ArcGIS
State Management	Application state and data flow	Zustand + Context API
UI Components	Interactive tools and controls	React + Radix UI

🔄 Complete Data Flow & Rendering

Main Data Flow Pipeline

1. User Input — Layer selection, configuration, and user interactions →
2. Data Validation — Input validation and data structure verification →
3. ArcGIS Processing — Layer conversion and ArcGIS SDK integration →
4. WebGL Rendering — GPU-accelerated map rendering and display →
5. Interactive Display — User interaction handling and real-time updates

Layer Management Flow

How Users Apply and Remove Layers

1. Layer Selection
User selects layer from catalog or uploads file
- ArcGIS Online layers
- File uploads (CSV, GeoJSON)
- Web services

→

2. Layer Processing
Layer data is processed and validated
- Geometry validation
- Attribute processing
- Coordinate system conversion

→

3. ArcGIS Integration
Layer is converted to ArcGIS format
- FeatureLayer creation
- Renderer configuration
- Popup template setup

→

4. Map Rendering
Layer is added to map and rendered
- WebGL rendering
- Layer ordering
- Performance optimization

Data Flow Architecture

```
// Data Flow Example - Layer Management const useMapLayers = ({ view, appliedLayers }) => { // 1. Process layer changes const processLayerChanges = async () => { const { added, removed, updated } = findLayerDifferences(oldLayers, newLayers); // 2. Remove old layers removed.forEach(layer => view.map.remove(layer)); // 3. Add new layers for (const layerItem of added) { const layer = await createLayerByType(layerItem); view.map.add(layer); } // 4. Update existing layers updated.forEach(layer => updateLayerProperties(layer)); }; // 5. Reorder layers for proper rendering reorderLayers(view.map); };
```

Layer Types Supported

Supported Layer Types:
- **Feature Layer:** Vector data with attributes and popups
- **Map Image Layer:** Raster-based map services
- **Vector Tile Layer:** Optimized vector tiles
- **Imagery Layer:** Satellite and aerial imagery
- **Group Layer:** Organized layer collections

Layer Management Flow

Layer Application & Management Flow

1. Layer Selection
User selects layer from catalog or uploads file
- ArcGIS Online layers
- File uploads (CSV, GeoJSON)
- Web services

→

2. Layer Processing
Layer data is processed and validated
- Geometry validation
- Attribute processing
- Coordinate system conversion

→

3. ArcGIS Integration
Layer is converted to ArcGIS format
- FeatureLayer creation
- Renderer configuration
- Popup template setup

→

4. Map Rendering
Layer is added to map and rendered
- WebGL rendering
- Layer ordering
- Performance optimization

Layer Management Hook Flow

```
// Layer Management Hook - Data Flow export function useMapLayers({ view, appliedLayers }) { const [isMapLoading, setIsMapLoading] = useState(false); // Process layer changes efficiently const processLayerChanges = async () => { const { added, removed, updated } = findLayerDifferences(oldLayers, newLayers); // Batch operations for performance await Promise.all([ ...removed.map(layer => removeLayer(layer)), ...added.map(layer => addLayer(layer)), ...updated.map(layer => updateLayer(layer)) ]); }; return { isMapLoading }; }
```

ArcGIS SDK Data Flow

ArcGIS Core SDK Data Processing Flow

User Input — Layer selection & configuration
↓
Data Validation — Geometry & attribute validation
↓
ArcGIS Processing — Layer conversion & optimization
↓
WebGL Rendering — GPU-accelerated rendering
↓
Interactive Display — User interaction & updates

ArcGIS Data Processing Pipeline

```
// ArcGIS Core SDK Data Flow const createLayerByType = async (layerItem: AppliedLayer) => { const layerType = layerItem.layerType || layerItem.type || ""; switch (layerType) { case "Feature Layer": // 1. Create FeatureLayer instance const featureLayer = new FeatureLayer({ url: layerItem.url, title: layerItem.title, visible: layerItem.visibility ?? true, outFields: ["*"], popupEnabled: layerItem.popupEnabled ?? true }); // 2. Load layer data await featureLayer.load(); // 3. Configure popup template if (layerItem.popupTemplate) { featureLayer.popupTemplate = layerItem.popupTemplate; } // 4. Set renderer if specified if (layerItem.renderer) { featureLayer.renderer = layerItem.renderer; } return featureLayer; case "Map Image Layer": // Handle Map Image Layer creation return new MapImageLayer({ url: layerItem.url, title: layerItem.title, visible: layerItem.visibility ?? true }); default: throw new Error(`Unsupported layer type: ${layerType}`); } };
```

Core ArcGIS Functionality

🎯 Main ArcGIS Features
- **Interactive Mapping:** Pan, zoom, and navigate with smooth WebGL rendering
- **Layer Management:** Add, remove, reorder, and configure multiple layer types
- **Data Visualization:** Custom renderers, popups, and labeling
- **Spatial Analysis:** Query features, measure distances, and analyze spatial relationships
- **Coordinate Systems:** Support for multiple projections and transformations
- **Performance Optimization:** Level-of-detail rendering and viewport culling

User Layer Operations

➕ Adding Layers
- Browse ArcGIS Online catalog
- Upload CSV/GeoJSON files
- Add web services (WMS, WFS)
- Import from ArcGIS Pro projects
- Create custom graphics layers

⚙️ Configuring Layers
- Set visibility and opacity
- Configure popup templates
- Apply custom renderers
- Set scale dependencies
- Configure labeling

🗑️ Removing Layers
- Remove individual layers
- Clear all layers
- Reset to default state
- Save layer configurations
- Export layer settings

🔄 Managing Layers
- Reorder layer stacking
- Group related layers
- Apply filters and queries
- Set layer permissions
- Monitor layer performance

Data Transformation Process

Data Transformation Steps:
- **Input Validation:** Check data format and structure
- **Coordinate Conversion:** Transform to Web Mercator (EPSG:3857)
- **Attribute Mapping:** Map fields to ArcGIS schema
- **Geometry Processing:** Validate and optimize geometries
- **Renderer Setup:** Configure visual representation
- **Performance Optimization:** Implement level-of-detail and culling

Technology Stack

Frontend Framework	Mapping SDK	Styling	State Management	Build Tool	Package Manager
React 18 + TypeScript	ArcGIS Core v4.33.6	Tailwind CSS	Zustand + Context API	Vite	npm

Dependencies

```
{ "dependencies": { "@arcgis/core": "^4.33.6", "react": "^18.2.0", "react-dom": "^18.2.0", "zustand": "^4.4.0", "@tanstack/react-query": "^5.0.0", "@radix-ui/react-popover": "^1.0.0" }, "devDependencies": { "typescript": "^5.0.0", "vite": "^5.0.0", "tailwindcss": "^3.3.0", "@types/react": "^18.2.0" } }
```

🚀 Application Structure & Entry Points

📁 Project File Structure

Core Application Files
- **src/main.tsx** - Application entry point with providers
- **src/App.tsx** - Main application component and routing
- **src/modules/map/MapView.tsx** - Core map component
- **src/hooks/useMapLayers.ts** - Layer management hook
- **src/store/** - State management with Zustand
- **src/components/** - Reusable UI components

🎯 Application Entry Points

```
// 1. MAIN ENTRY POINT - src/main.tsx import React from 'react'; import ReactDOM from 'react-dom/client'; import { BrowserRouter } from 'react-router-dom'; import { QueryClient, QueryClientProvider } from '@tanstack/react-query'; import App from './App'; const queryClient = new QueryClient({ defaultOptions: { queries: { staleTime: 5 * 60 * 1000, retry: 1 } } }); const Root = () => ( ); ReactDOM.createRoot(document.getElementById('root')!).render();
```

🗺️ Map Component Structure

```
// 2. MAP COMPONENT - src/modules/map/MapView.tsx const MapComponent = ({ isMapExpanded, isProfileLoading, setIsMapExpanded }) => { const [view, setView] = useState(null); const [appliedLayers, setAppliedLayers] = useState([]); // ArcGIS Core SDK initialization useEffect(() => { const defaultMap = new WebMap({ portalItem: { id: DEFAULT_WEBMAP_ID } }); const newView = new MapView({ container: mapDiv.current, map: defaultMap, constraints: { minZoom: mapConstants.MIN_ZOOM, maxZoom: mapConstants.MAX_ZOOM } }); setView(newView); }, []); return (
); };
```

🔧 Layer Management Hook

```
// 3. LAYER MANAGEMENT - src/hooks/useMapLayers.ts export function useMapLayers({ view, appliedLayers }) { const [isMapLoading, setIsMapLoading] = useState(false); // Process layer changes efficiently const processLayerChanges = async () => { const { added, removed, updated } = findLayerDifferences(oldLayers, newLayers); // Batch operations for performance await Promise.all([ ...removed.map(layer => removeLayer(layer)), ...added.map(layer => addLayer(layer)), ...updated.map(layer => updateLayer(layer)) ]); }; return { isMapLoading }; }
```

📊 State Management Structure

```
// 4. STATE MANAGEMENT - src/store/mapStore.ts export const useMapStore = create((set, get) => ({ center: { lat: 0, lng: 0 }, zoom: 10, baseMapId: null, appliedLayersData: "", parcelPins: [], updateMapState: (zoom, center, boundingBox, mapScale) => set({ zoom, center, boundingBox, mapScale }), addParcelPin: (parcelId, lat, lon, attributes) => set(state => ({ parcelPins: [...state.parcelPins, { parcelId, lat, lon, attributes }] })) }));
```

Layer Management Hook

```
// Custom Hook for Layer Management export function useMapLayers({ view, appliedLayers }) { const [isMapLoading, setIsMapLoading] = useState(false); // Process layer changes efficiently const processLayerChanges = async () => { const { added, removed, updated } = findLayerDifferences(oldLayers, newLayers); // Batch operations for performance await Promise.all([ ...removed.map(layer => removeLayer(layer)), ...added.map(layer => addLayer(layer)), ...updated.map(layer => updateLayer(layer)) ]); }; return { isMapLoading }; }
```

State Management

```
// Zustand Store for Map State export const useMapStore = create((set, get) => ({ center: { lat: 0, lng: 0 }, zoom: 10, baseMapId: null, appliedLayersData: "", parcelPins: [], updateMapState: (zoom, center, boundingBox, mapScale) => set({ zoom, center, boundingBox, mapScale }), addParcelPin: (parcelId, lat, lon, attributes) => set(state => ({ parcelPins: [...state.parcelPins, { parcelId, lat, lon, attributes }] })) }));
```

👥 User Workflow & Experience

Complete User Journey

🚀 Getting Started
- User logs in to the application
- Default map loads with basemap
- User sees available tools and layers
- Initial view shows default extent
- User can start exploring immediately

🗂️ Adding Layers
- Click "Add Layers" button
- Browse ArcGIS Online catalog
- Search for specific layers
- Preview layer content
- Add layer to map

⚙️ Configuring Layers
- Right-click layer for options
- Adjust transparency and visibility
- Configure popup information
- Set layer filters and queries
- Customize rendering styles

🔍 Interactive Analysis
- Click features for detailed info
- Measure distances and areas
- Query features by attributes
- Export data and results
- Save analysis configurations

🎯 Key User Interactions

```
// User Interaction Examples const handleLayerInteraction = { // Layer visibility toggle toggleVisibility: (layerId) => { const layer = view.map.findLayerById(layerId); layer.visible = !layer.visible; }, // Layer opacity adjustment setOpacity: (layerId, opacity) => { const layer = view.map.findLayerById(layerId); layer.opacity = opacity; }, // Feature selection selectFeature: (feature) => { view.highlight(feature); showPopup(feature); }, // Map navigation navigateTo: (extent) => { view.goTo(extent, { duration: 1000 }); } };
```

Setup & Configuration

Prerequisites
- Node.js 18+ and npm
- ArcGIS Developer Account
- Valid API Key for ArcGIS Services
- Modern web browser with ES6+ support

Installation Steps

```
# 1. Clone the repository git clone [repository-url] cd project-name # 2. Install dependencies npm install # 3. Configure environment variables cp .env.example .env.local # 4. Add your ArcGIS API key VITE_ARCGIS_API_KEY=your_api_key_here VITE_DEFAULT_WEBMAP_ID=your_webmap_id # 5. Start development server npm run dev
```

Environment Configuration

```
# Required Environment Variables VITE_ARCGIS_API_KEY=your_arcgis_api_key VITE_DEFAULT_WEBMAP_ID=your_default_webmap_id VITE_API_BASE_URL=your_api_base_url VITE_GOOGLE_API_KEY=your_google_api_key
```

API Reference

ArcGIS Core SDK Integration

```
// Core ArcGIS imports import MapView from "@arcgis/core/views/MapView"; import WebMap from "@arcgis/core/WebMap"; import FeatureLayer from "@arcgis/core/layers/FeatureLayer"; import PopupTemplate from "@arcgis/core/PopupTemplate"; import Graphic from "@arcgis/core/Graphic"; // Map initialization const mapView = new MapView({ container: "mapDiv", map: new WebMap({ portalItem: { id: "webmapId" } }), zoom: 10, center: [-118.244, 34.052] });
```

Custom Hooks API

Hook	Purpose	Returns
useMapLayers	Layer management and rendering	{ isMapLoading }
useMapStore	Map state management	Map state and actions
useLayerFilters	Layer filtering capabilities	Filter functions
useMapExtentFilter	Map extent filtering	Extent filter functions