2024

INTEGRATING ARCGIS AND REDUX USING MIDDLEWARE

Vishnu Vardhan Reddy Rapuru
Michigan Technological University, vrapuru@mtu.edu

INTEGRATING ARCGIS AND REDUX USING MIDDLEWARE

By

Vishnu Vardhan Reddy Rapuru

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY

2024

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Computer Science.

Department of Computer Science

Report Advisor :	*Robert L Pastel*
Committee Member :	*Gorkem Asilioglu*
Committee Member :	*Donald J Lafreniere*
Department Chair :	*Zhenlin Wang*

Table of Contents

Author Contribution Statement

The practical execution of the project was a joint effort between Vishnu Rapuru and Ram Sudda, who worked in tandem to bring the project to fruition. The composition of this report, however, was undertaken independently by myself, encapsulating the collective work and findings derived from our collaborative endeavor.

Acknowledgements

I would like to express my sincere gratitude to Parsharam Reddy Sudda, who collaborated on various aspects of this project. While I am the author of this report, Ram's contributions were instrumental in the successful implementation of the integrated ArcGIS and Redux solution. His dedication, insights, and collaborative spirit significantly enriched the development process, and I appreciate the teamwork that has led to the completion of this project.

Abstract

The integration of ArcGIS with Redux through middleware presents a novel approach to managing state in geospatial applications. This report outlines the process and benefits of combining ArcGIS's robust mapping and analytics capabilities with Redux's predictable state container for JavaScript apps. It begins with an introduction to both technologies, followed by a detailed discussion on the architecture design, focusing on the role of middleware as the linchpin in this integration[1]. The paper highlights the benefits, such as improved state management and application performance, and addresses the challenges encountered during the integration process. Implementation details are provided, including the setup of the Redux store and the specific middleware used for ArcGIS integration. The paper concludes with a look at potential future enhancements, emphasizing the scalability and maintainability of this integration method. This Report serves as a guide for developers seeking to enhance their geospatial applications with advanced state management techniques.

I. Introduction

The integration of Redux with ArcGIS has been explored to enhance the capabilities of web applications by combining the predictable state management of Redux with the powerful geospatial features of ArcGIS. A notable example is the work shared by Esri on building modern web apps with the ArcGIS JavaScript API, where Redux plays a crucial role in managing application state, leading to fewer bugs and more predictable behavior [1]. Additionally, Esri provides a boilerplate on GitHub for an ArcGIS JS API 4.x app using React and Redux, which includes middleware for ArcGIS Online authentication and SceneView management [2]. This boilerplate serves as a foundation for developers to build robust web scene viewers with integrated state management.

This report emphasizes the benefits of this integration, such as total separation of data and presentation, reusable UI components, and enhanced performance. It also discusses the architectural patterns, like encapsulating the ArcGIS JS API in middleware to keep non-data objects out of the store and facilitate easier testing. The use of Redux DevTools Extension for live state browsing and dispatching actions, as well as Hot Module Replacement for seamless updates during development, are highlighted as key features that improve the development experience[2].

Background

The KeTT(Keweenaw Time Traveller) application is a tool that allows users to explore maps and share stories related to the Copper Country history, powered by Esri modern. It integrates ArcGIS and Redux to leverage the strengths of both technologies. However, the current version of the KeTT application does not utilize middleware, which leads to following issues:

- **State Management Contradicting React Principles:** The MapView is embedded within the Map component, which prevents the Redux reducer from controlling MapView interactions and state updates. As a result, the application relies on querying the DOM to determine the state, which goes against the React principle of having the app state dictate the UI [5][4].
- **Tight Coupling of Actions and Components:** Actions are dispatched directly within components, leading to a tight coupling between them. This design limits the reusability of actions across different parts of the application and makes it difficult to manage state in a scalable way[6][5].
- **Inefficient Map Rendering:** Every time a new time-period or timeline range is selected, the entire map component is re-rendered. This is not only inefficient but also unnecessary since only the tile layer above the basemap needs to be updated.

To address these issues, integrating Redux Middleware can provide a centralized way to handle actions, allowing for greater reusability and easier management of state transitions. Middleware can also optimize rendering by initializing the basemap once and updating only the necessary layers upon action dispatches. Furthermore, by abstracting MapView interactions into middleware, the application can adhere to React's unidirectional data flow, where the state is the single source of truth for the UI.

II. Understanding ArcGIS, ReactJs, and Redux

ArcGIS

ArcGIS is a comprehensive platform developed by Esri for mapping, spatial analysis, and geospatial data visualization. It's widely used for creating interactive maps and geospatial applications[13]. Major features of ArcGIS are:

- **Geospatial Data:** ArcGIS works with geographic and spatial data, such as maps, layers, and features.
- **Web Mapping:** ArcGIS enables web-based mapping, allowing users to access maps via web browsers or mobile apps.
- **Spatial Analysis:** It supports various spatial analysis operations, including proximity analysis, geocoding, and routing.
- **Customization:** Developers can extend ArcGIS functionality through APIs and SDKs.

ReactJs

ReactJS, commonly known as React, is a free and open-source front-end JavaScript library used for building user interfaces, particularly for single-page applications. It allows developers to create large web applications that can change data, without reloading the page. Its key feature is the ability to build components, which are small, reusable pieces of code that return a React element to be rendered to the page. React's declarative nature makes it easy to predict and debug, and it efficiently updates and renders just the right components when data changes. Developed by Facebook, React has gained immense popularity due to its simplicity, performance, and scalability[15].

- **Component Reusability:** React's component-based structure enables the creation of reusable UI components, facilitating the development of modular and scalable applications.
- **Virtual DOM:** React's virtual DOM efficiently updates the UI by only re-rendering components that have changed, optimizing performance in applications with dynamic data updates, such as ArcGIS maps.
- **State Management:** React provides built-in state management capabilities, which can be augmented with libraries like Redux for managing complex application states, including ArcGIS data.

Redux

Redux is an open-source JavaScript library for managing application state in a predictable and centralized manner. It's commonly used with React but can be integrated with other libraries or frameworks[14]. Major Components of Redux are:

- **Store:** Redux stores the entire application state in a single JavaScript object.
- **Actions:** Actions are plain objects describing changes to the state. They're dispatched to the store.
- **Reducers:** Reducers specify how the state changes in response to actions. They are pure functions.
- **Dispatch:** Dispatching an action is the process of sending it to the Redux store.

III. Architecture Design

Overview of Middleware

Redux middleware provides a way to interact with actions that have been dispatched to the store before they reach the reducer. Middleware functions can perform tasks such as logging, crash reporting, performing asynchronous requests, or modifying actions. Essentially, middleware acts as a third-party extension point between dispatching an action and the moment it reaches the reducer [7].

The Webmap Middleware that we implemented in our project is a specialized type of middleware tailored for integrating ArcGIS capabilities with a Redux-managed React application. By intercepting actions and handling the ArcGIS API calls within the middleware, it streamlines the process of updating the map view and managing geospatial data. This approach minimizes the need for multiple dispatches and allows for a more centralized and efficient handling of state changes related to the map, which can be particularly beneficial in complex applications where map interactions are frequent and varied [2].

Role of Middleware in Integrating ArcGIS and Redux

The ArcGIS.mapView object is created and configured in the middleware, and it is used to display and manipulate the map within our application.

The middleware imports action types (e.g.: INIT_SCENE, SET_CENTER, SET_PLACE, SET_PORTAL_URL) and a selector (selectedTimeline) from Redux action creators and reducers. These action types represent various actions that can be dispatched in response to user interactions.

The selectedTimeline selector captures and updates the user's timeline selection. It is first used by the middleware to render the ArcGIS map with the selected timeline's data, and then it is stored in the Redux store for reference by other components, such as TimelineTitle, to provide a consistent user experience based on the selected timeline.

Components of the Integration Architecture

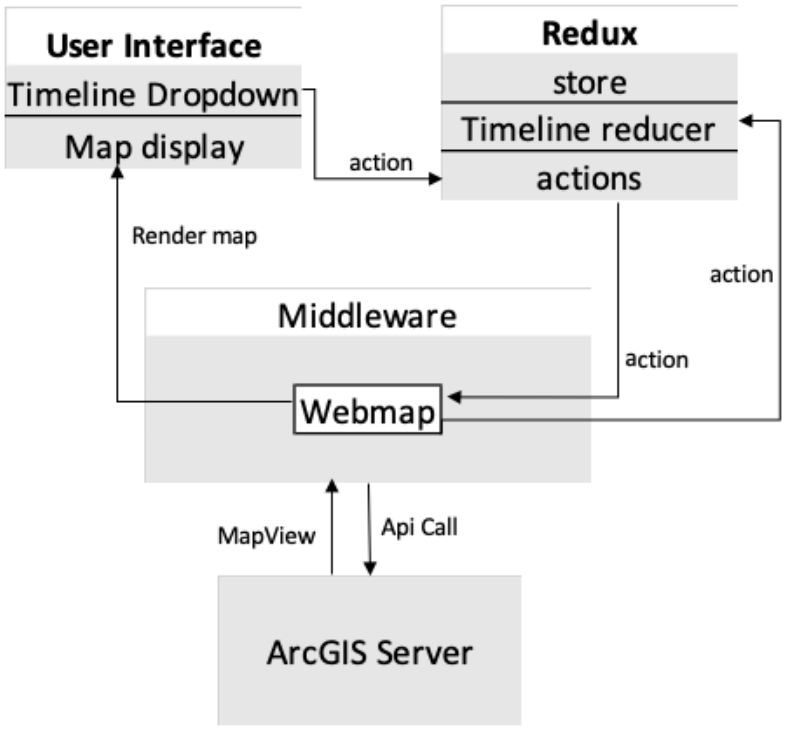

Figure 1: Architecture for Integration of ArcGIS with Redux using Middleware

Redux Store

The Redux store stores the application state, including data and information that components need to render. The ArcGIS.mapView object is a part of the application's functionality allowing you to configure and control how geographic data is displayed to the user using middleware to display the desired geospatial information. it is not stored within the store itself.

Storing the ArcGIS.mapView object directly in the Redux store is not a common pattern because the map view is typically treated as part of the view layer of our application, while Redux manages the application's state layer. The middleware facilitates rendering mapView on our React Application but does not store the view itself in the Redux store.

Middleware

The middleware initializes a ArcGIS object, which serves as a container for ArcGIS-related objects and data. This object allows you to maintain a reference to key ArcGIS components and objects throughout the middleware's execution.

The middleware itself is a function that takes the store as its first argument, returning another function that takes next as its argument, which, in turn, takes action as its argument. This function structure is typical for Redux middleware.

```
// Middleware
export const webMap = store => (next) => (action) => {
   // Middleware logic here
   // ...
   next(action);
// Pass the action to the next reducer TimelinePicker in TimelineSlice
};
```

ArcGIS API Middleware

The ArcGIS API provides access to a wide range of geospatial capabilities and services. Middleware calls ArcGIS API to retrieve map data, perform geospatial operations, and update

map layers. Also uses ArcGIS servers and services to fetch geospatial information requested by the React application.

IV. Implementation Details

Setting up the Redux Store

1) Defining state Structure

State structure, in the context of Redux and similar state management libraries, refers to the way the application's data is organized and stored. It defines the shape and structure of the entire application's state, which includes all the data that your application needs to maintain and manipulate. State structure typically consists of nested objects and arrays to represent different aspects of the application's data.

At the top level, the application is organized into three major objects:

```
Store {
  timeline: {},
  errors: {},
  ArcGISstate: {}
 }
```

- **Timeline:** This part of the state is dedicated to storing timeline-related data. It specifically stores the selectedTimeline which is used to call the ArcGIS API in the middleware and then saved in the store.

```
const Timeline = () => {
        useEffect(() => {
              handleOnChangeTimeline();
        }, []);
```

```
        const handleOnChangeTimeLine = (year) =>{
                dispatch(selectedTimeline(payload))
        }
}
```

- **Errors:** This part of the state manages error-related states in our Redux store.
- **ArcGISstate:** This part of the state is designated for storing data related to the ArcGIS integration in our application. The ArcGISstate object may contain various properties and data specific to ArcGIS functionalities. This data could include information about Center coordinates integrated into our application.

2) Creating Reducers and actions

To describe the actions and reducers in our application, there are three slices that we use in our application: ArcGISSlice, TimelineSlice, and errorSlice.

Reducers:

- **ArcGISReducer:** Manages state related to ArcGIS integration.
- **TimelinePicker:** Manages state related to timeline data and selection.
- **Error:** Manages error-related state.

Extra Reducers:

These reducers handle the states associated with the asynchronous ‘fetchTimelineData’ action, which are pending, fulfilled, and rejected states.

Actions:

- **ArcGISLoadMap:** This action updates the ‘mapId’ property in the ArcGIS state with the value provided in the payload. It’s used to load a specific map in the ArcGIS integration.
- **selectedTimeline:** This action updates the ‘selected time’ property in the timeline state with values provided in the payload. It is used to select a specific timeline for display.
- **clearTimelineData:** This action clears the ‘timeline data’ property in the timeline state, typically used when you need to reset or clear the timeline data.
- **error:** This action sets the error state to the value provided in the payload. It’s used to handle and display errors in the application.
- **remove error:** This action clears the error state, typically used when an error needs to be removed or reset.

Implementing Middleware for ArcGIS Integration

A. Handling Asynchronous actions

fetchTimelineData: This async action is used for fetching timeline data from a specified URL. It has pending, fulfilled, and rejected actions automatically generated by the Redux Toolkit’s ‘createAsyncThunk’. These actions are used to manage the loading state and handle data retrieval.

B. Dispatching ArcGIS related actions

INIT_SCENE Action: (role of action) This action is dispatched to initialize the map scene by creating a map view (MapView) and attaching it to an HTML container element (DIV).

A MapView instance is a 2D view of a Map within the application. It is responsible for displaying the map and its various layers, handling user interactions like zooming and panning, and providing access to the map’s data. When you store a MapView instance in the

application’s state or context, it becomes a central element that other components and actions can reference and manipulate. This allows for a cohesive and interactive mapping experience, as the MapView instance acts as the canvas where geographical data is rendered. It facilitates actions such as setting the map’s center, switching base maps, and adding layers, making it an essential part of the application’s geospatial functionality1.

By incorporating the MapView instance as a core aspect of the INIT_SCENE action, it ensures that the map’s state is readily accessible and modifiable by the application’s other components. This integration allows for a dynamic and responsive mapping interface, where changes to the map are efficiently managed and reflected in the user interface. The MapView’s persistence within the application’s state is crucial for maintaining a consistent and interactive mapping environment, enabling a seamless user experience as they interact with the map and its features.

SET_CENTER Action: This action is dispatched to set the map's centre and zoom level.

SET_PORTAL_URL Action: This action is dispatched to set the URL of a tile layer based on the provided URL in the action payload.

SET_PLACE Action: This action is dispatched when a user searches for a place. It uses the ArcGIS Geocoding service to find address candidates for the specified place.

Process of Integrating ArcGIS and Redux

Dispatching actions from components

We have a Timeline component that presents various timelines to the user. When the user selects a timeline, the selectedTimeline action is dispatched with a payload containing information about the selected timeline to the webMap middleware.

```
const Timeline = () => {
```

```
  const dispatch = useDispatch();
  const handleOnChangeTimeLine = (year) => {
// ...
      // Dispatch the selectedTimeline action with payload
      dispatch(selectedTimeline({ payload }));
    }
  };
```

Middleware Processing of actions

The dispatched selectedTimeline action passes through the middleware layer before reaching the Redux store. In our middleware setup, the webMap middleware intercepts these actions.

Inside the webMap middleware, there is a switch statement that checks the type of each intercepted action. Depending on the action type, different processing steps are performed. For example:

- INIT_SCENE: Initializes the ArcGIS scene and map view if there isn't an existing basemap, using data from selectedTimeline.
- SET_CENTER: Modifies the map view's center point and zoom level.
- SET_PORTAL_URL: Updates the feature layer (Tile Layer) on top of the basemap according to given URL, which could change features or markers based on the selected time period.
- SET_PLACE: Geocodes a specified place and updates the map view to center on this location.

Interfacing with ArcGIS API

After processing the action, the middleware utilizes ArcGIS API, which makes HTTP requests to ArcGIS services, initializing map views, adding layers, or performing geospatial operations. In the context of our middleware, we are configuring various aspects of the ArcGIS map view, such as the base map, location widget, and housing layer.

Updating Redux store with selectedTimeline data

The selectedTimeline action, which contains information about the selected timeline, is passed from the middleware to Timeline reducer to update the store. This ensures that the selected timeline information is stored in the Redux store for further use.

V. Benefits and Challenges

Benefits of Integrating ArcGIS and Redux

Integrating ArcGIS and Redux offers several benefits like streamlining the coding process, making modifications more manageable, and facilitating long-term code maintenance by enforcing clear patterns and practices. This combination enhances code quality and developer productivity.

- **Easy to code**

Redux provides a clear and organized structure for managing the application state by enforcing a predictable pattern for updating and accessing data, which simplifies the coding process.

Redux encourages a modular approach to our application, including ArcGIS integration, which can be encapsulated within separate reducers, actions, and middleware, which makes code more manageable and easier to reason about [1].

- **Easy to Modify Code**

Redux facilitates consistent and structured state updates through reducers. When changes are needed, developers can make modifications within well-defined reducers, ensuring that updates are applied consistently throughout the application [9].

The structured nature of Redux allows developers to pinpoint issues within specific reducers, actions, or middleware, reducing the time and effort required to identify and fix problems [10].

Redux's flexibility allows for code modifications without any significant disruptions to extend functionality or introduce new features while maintaining existing code.

- **Easy Maintenance**

Redux promotes a consistent and predictable way of managing state, which makes it easier for development teams to maintain the codebase collectively, as everyone follows the same conventions and patterns [11].

By providing a structured approach to state management, Redux helps prevent the accumulation of technical debt with organized and maintainable code, reducing the likelihood of costly refactoring or rewrites.

The Redux ecosystem has robust documentation and a supportive community that offers best practices and guidance. This wealth of resources aids in maintaining code quality and addressing maintenance challenges effectively.

Challenges Faced During Integration

- **Data Synchronization**

Integrating ArcGIS with Redux often involves managing complex geospatial data, which can change frequently due to user interactions or real-world updates[1]. Ensuring that the data displayed on the map remains synchronized with the application's state in Redux can be challenging.

- **Handling Complex Interactions**

Single-user interaction on the map requires the complexity of user interactions with the map to increase, which requires applications to perform tasks like drawing polygons, querying data, or navigating between different map views. Coding for multiple interactions while ensuring a clean and maintainable codebase can be a significant challenge [12].

- **Overcoming UI Interaction Challenges**

A notable challenge during development involved the positioning of components above the map, which occasionally obscured user interactions with MapView. The solution to this problem was to employ a flexbox approach using Tailwind CSS, enabling precise component placement on the page and ensuring that the map view's usability remained unaffected by any overlapping UI elements. This solution provided an effective balance between user interface design and map interactivity.

VI. Future Enhancements

As the project evolves, several key enhancements and extensions are identified to elevate the capabilities and user experience:

1. Complete Integration of ARCGIS with React and npm Module Development:

- Develop a dedicated npm module that abstracts integrating ARCGIS with React applications. As there is currently no official React support for ARCGIS, this initiative would empower React developers to effortlessly incorporate geospatial functionalities into their applications. The npm module not only facilitates integration but also allows users to conveniently download and incorporate the necessary software components, enhancing accessibility and usability.
- An npm module refers to a packaged collection of JavaScript files and other resources, encapsulated to perform a specific set of functionalities.
- Define a clear and comprehensive API(Application Programming Interface) for the npm module to facilitate straightforward integration and usage.(API essentially refers to a set of functions, methods, and rules that allow one piece of software to interact with another.)
- Provide thorough documentation, including installation instructions, API references, and usage examples, to ensure accessibility and ease of adoption.

2. Integration of a Reset Button:

- Introduce a reset button feature to enable users to revert to the default state effortlessly, especially after performing various actions within the application.

- Ensure the reset button is strategically placed, easily accessible, and clearly labelled for user convenience.
- The introduction of a reset button offers a nuanced approach to resetting the application state without the need for a complete application reload. Unlike a traditional reload, which involves fetching data from APIs and rendering the entire map again, the reset button can be implemented to selectively reset specific aspects of the state while minimizing unnecessary data calls and rendering processes.

These proposed enhancements aim to not only expand the maintainability and robustness of the application but also improve its accessibility, usability, and overall user satisfaction. This also ensures that the underlying code remains not only user-friendly but also sustainably structured, facilitating long-term reliability and scalability for our application.

VII. References

1. React Redux: Building Modern Web Apps with the ArcGIS JS API – https://www.esri.com/arcgis-blog/products/3d-gis/3d-gis/react-redux-building-modern-web-apps-with-the-arcgis-js-api/
2. Boiler Plate on GitHub for an ArcGIS JS API 4.x app – https://github.com/Esri/react-redux-js4
3. Keweenaw Time Traveller Web-application https://kett.geospatialresearch.mtu.edu/
4. React Managing State – https://react.dev/learn/managing-state
5. Keweenaw Time Travel Explorer Code Documentation – https://github.com/Keweenaw-Time-Traveler/ktt-app/tree/docs/docs/js
6. React Redux: Performance considerations when dispatching multiple actions – https://medium.com/unsplash/react-redux-performance-considerations-when-dispatching-multiple-actions-5162047bf8a6
7. Middleware - https://redux.js.org/understanding/history-and-design/middleware
8. Redux Middleware – what it is and How to Build it from scratch – https://www.freecodecamp.org/news/what-is-redux-middleware-and-how-to-create-one-from-scratch/
9. Redux Fundamentals, Part 3: State, Actions, and Reducers – https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers
10. Right way to update state in redux reducers – https://stackoverflow.com/questions/36031590/right-way-to-update-state-in-redux-reducers
11. How to organize your React + Redux Codebase – https://www.pluralsight.com/resources/blog/guides/how-to-organize-your-react--redux-codebase
12. Chapter 6. Handling Complex side effects – https://livebook.manning.com/book/redux-in-action/chapter-6/10
13. ArcGIS Documentation:

- ArcGIS for Developers: https://developers.arcgis.com/
- ArcGIS REST API: https://developers.arcgis.com/rest/

14. Redux Documentation:

    - Official Redux Documentation: https://redux.js.org/introduction/getting-started

15. React Documentation:

    - React Official Documentation: https://reactjs.org/docs/getting-started.html

16. Middleware and Redux Toolkit:

    - Redux Toolkit Middleware: https://redux-toolkit.js.org/api/createMiddleware
    - Understanding Redux Middleware: https://redux.js.org/understanding/history-and-design/middleware

17. Related Blog Posts and Articles:

    - Using ArcGIS in a React Application: https://developers.arcgis.com/javascript/latest/sample-code/intro-mapview/index.html
    - Managing Complex UI with Flexbox and React: https://css-tricks.com/using-flexbox/

18. Redux and State Management:
    - "Redux in Action" by Marc Garreau and Will Faurot

19. General Web Development Resources:
    - Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Web