

**A10 HTTP Redux Solution**

***Engineering Design Document***

Version 1.0a

2018/02/5

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Author / Editor | Revision | Date | Comments |
| Roll, Chris | 1.0a | 2/5/2018 | For drop 3 release |

Contents

[Goal and Objective 4](#_Toc505678180)

[A10 Redux HTTP Workflow 4](#_Toc505678181)

[Feature Summary 5](#_Toc505678182)

[Terminology 6](#_Toc505678183)

[Container-ID 6](#_Toc505678184)

[Observable 6](#_Toc505678185)

[Epic 7](#_Toc505678186)

[Inbound Action 8](#_Toc505678187)

[Outbound Action 8](#_Toc505678188)

[Dispatcher 9](#_Toc505678189)

[A10 HTTP Actions 9](#_Toc505678190)

[Inbound Actions 9](#_Toc505678191)

[Outbound Actions 11](#_Toc505678192)

[Dispatchers - A10Dispatchers 12](#_Toc505678193)

[A10Dispatchers.httpRequest() 12](#_Toc505678194)

[A10Dispatchers.cancelHTTPRequest() 16](#_Toc505678195)

[Middlewares 16](#_Toc505678196)

[A10HTTPEpic 16](#_Toc505678197)

[Interceptor of A10HTTPEpic 20](#_Toc505678198)

[A10GCEpic 23](#_Toc505678199)

[Reducers 25](#_Toc505678200)

[A10DataReducer 25](#_Toc505678201)

[A10GCReducer 27](#_Toc505678202)

[Namespace 29](#_Toc505678203)

[Setup namespace in Global Config 29](#_Toc505678204)

[Retrieve namespace in A10Container 30](#_Toc505678205)

[Using A10 HTTP Redux Solution in The Real World 31](#_Toc505678206)

[Design Principle 31](#_Toc505678207)

[References 39](#_Toc505678208)

# 

# Goal and Objective

1. GUI Framework provides reusable Redux actions for asynchronous HTTP data processing.
2. Developer can get a variety of HTTP status actions such as successful HTTP response, HTTP error handling, and has the ability to cancel previous HTTP requests.
3. GUI Framework manages the life cycle of data in the Redux store for HTTP responses and takes care of their garbage collection when the containers get unmounted.

# A10 Redux HTTP Workflow

1. Fetch data via HTTP

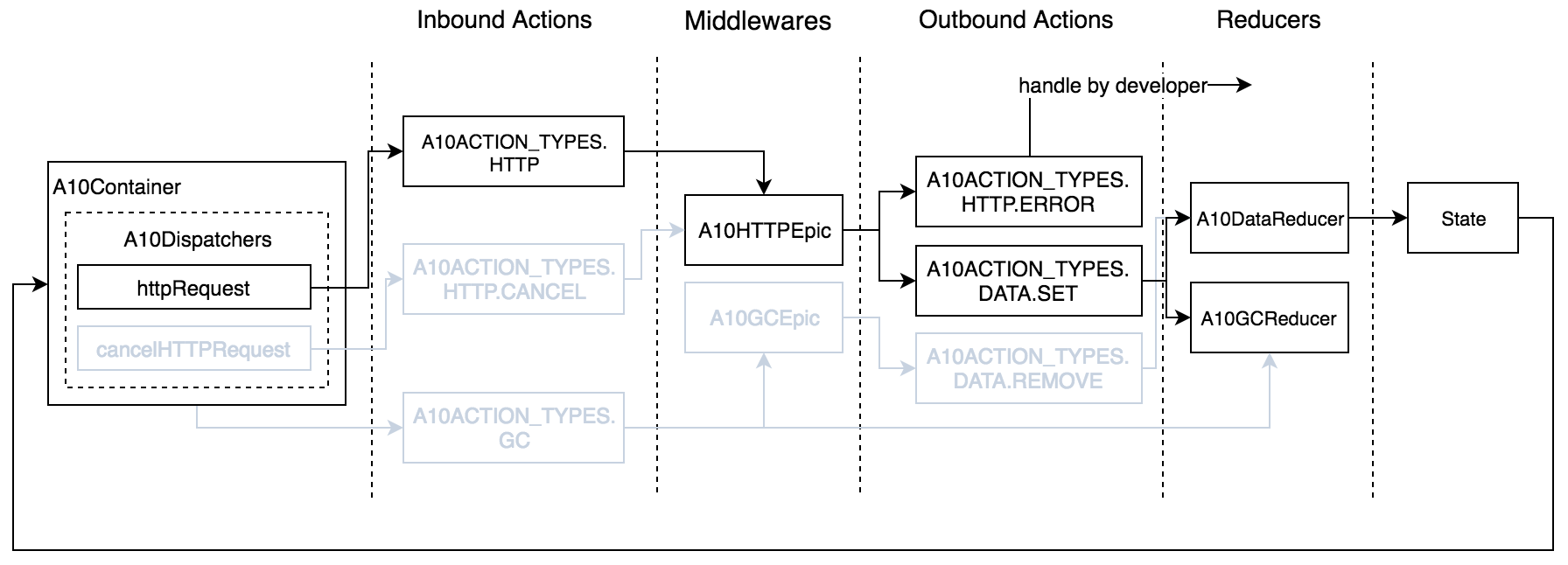


Figure 1 the workflow of a10 http redux – HTTP

1. Cancel HTTP request

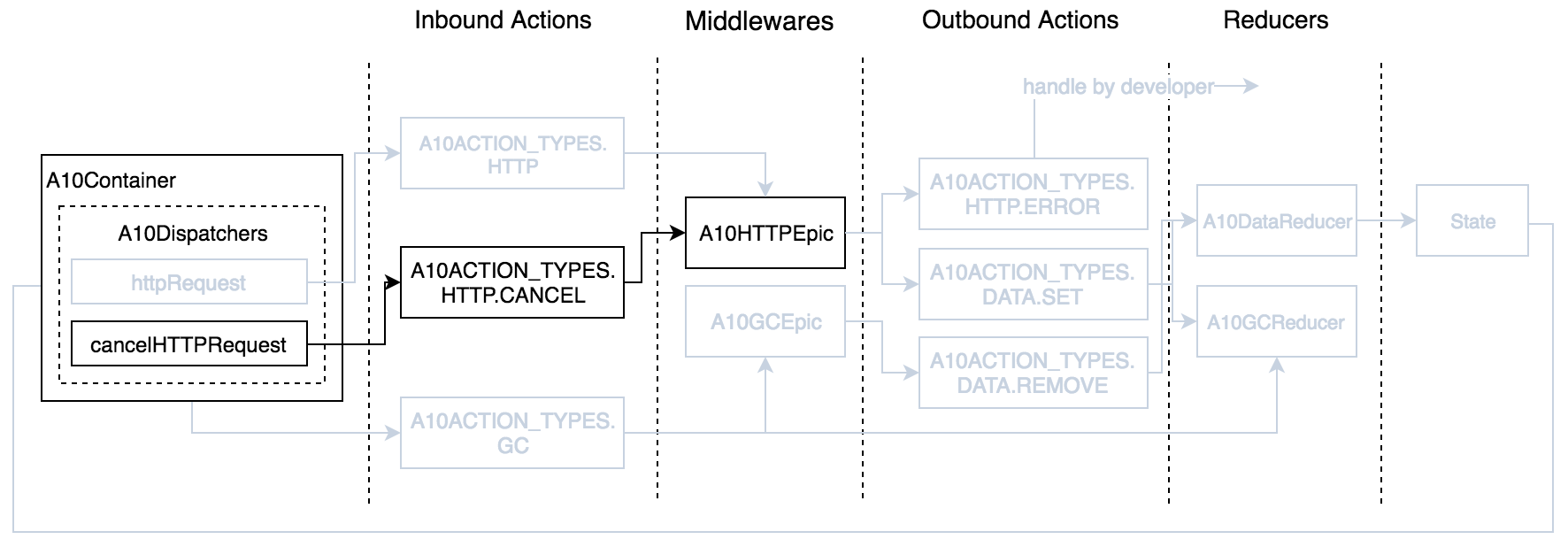


Figure 2 the workflow of a10 http redux - HTTP.CANCEL

1. Release state data when container is to be unmounted

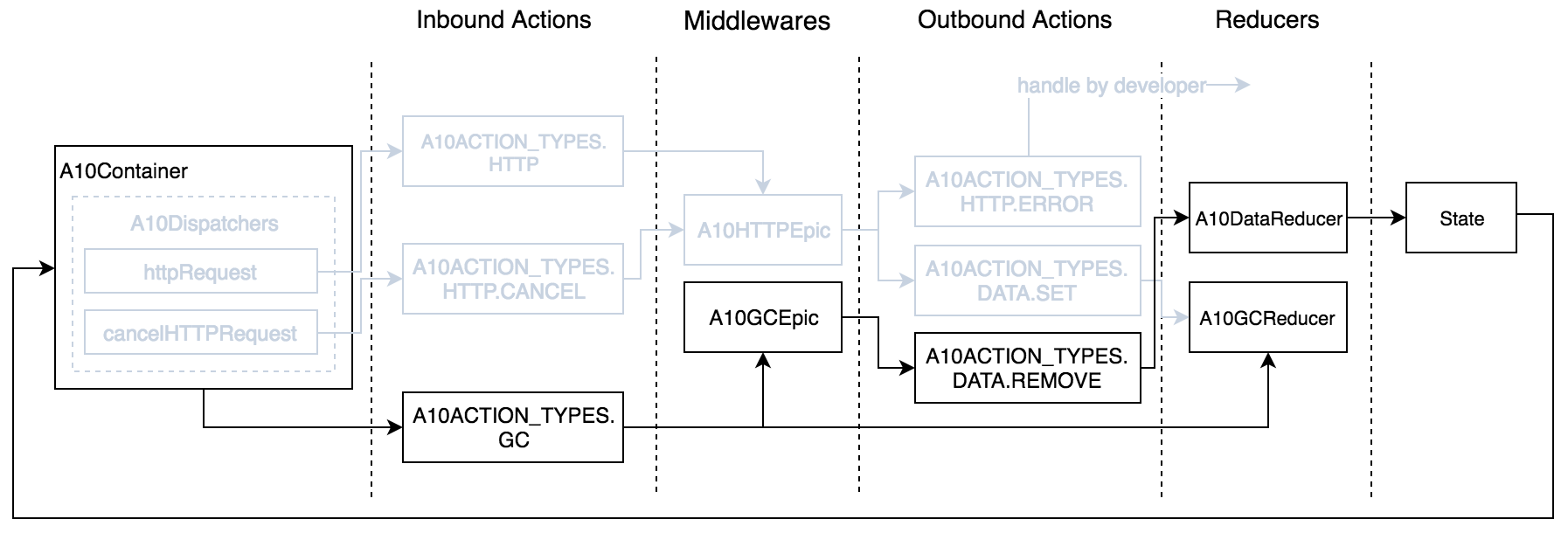


Figure 3 the workflow of a10 http redux - GC

# Feature Summary

GUI Framework provides reusable Redux actions for asynchronous HTTP request based on Redux + Redux-Observable. This facility helps developers to retrieve asynchronous data via HTTP with ease.

All React-Redux applications need to communicate with an HTTP server where a set of common problems frequently arise while making HTTP requests - Error handling, Request cancellation, and State data management. The following are our proposals for dealing with these issues.

1. **Generic Error Handling**

GUI Framework will catch every HTTP error and program error at runtime. It analyzes the error, and dispatches out a Redux action with complete error information and appropriate error message (JSON syntax or pure text) that could be customized by app developer if needed.

1. **Request Cancellation**  
   Every HTTP request is able to be aborted as long as developer dispatches out a cancel action. However, if the request may affect server state change, then the server state is undefined. This cancellation facility is often used when switching pages or to stop UI operations (mainly HTTP GET requests).
2. **State Data Management**

Most developers may face issues arising from state data management in Redux store. Redux will not purge unused state data in the store by default, which may introduce unnecessary memory consumption.

GUI Framework will delete unused state data in the Redux store automatically when containers are unmounted, thereby reducing burden on developers.

# Terminology

## Container-ID

It is a unique ID assigned by GUI Framework automatically for every A10Container.

## Observable

This term comes from ReactiveX, and can be thought of as a representation of any set of values over any amount of time. This is a fundamental concept of ReactiveX.   
In asynchronous programming as typified in ReactiveX, many instructions may execute in parallel and their results are later captured, in arbitrary order, by “observers” which are interested in gathering the results of the executions. Rather than calling a method, you define ***a mechanism for retrieving and transforming the data***, in the form of an “Observable”, and subscribe an observer to it, at which point the said mechanism fires into action with the observer standing sentry to capture and respond to its emissions whenever they are ready.

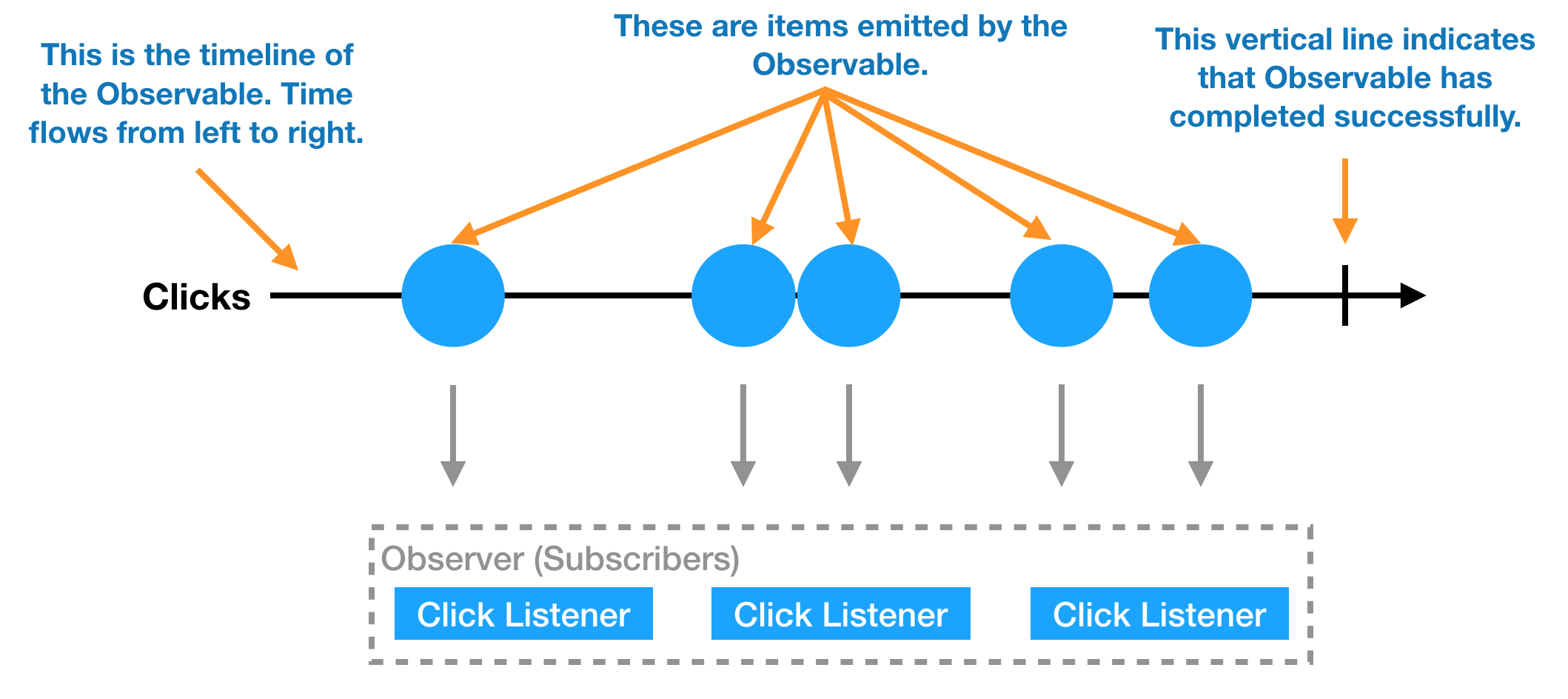


Figure 4 Observable IN REACTIVEX

## Epic

GUI Framework uses **redux-observable** as a middleware in Redux. This middleware uses Epic as a core primitive unit to handle the asynchronous thing. Epic is a function that takes a stream of actions and returns a stream of actions.

Epics run alongside the normal Redux dispatch channel, after the reducers have already received them--so you cannot "swallow" an incoming action. Actions always run through your reducers ***before*** your Epics even receive them.

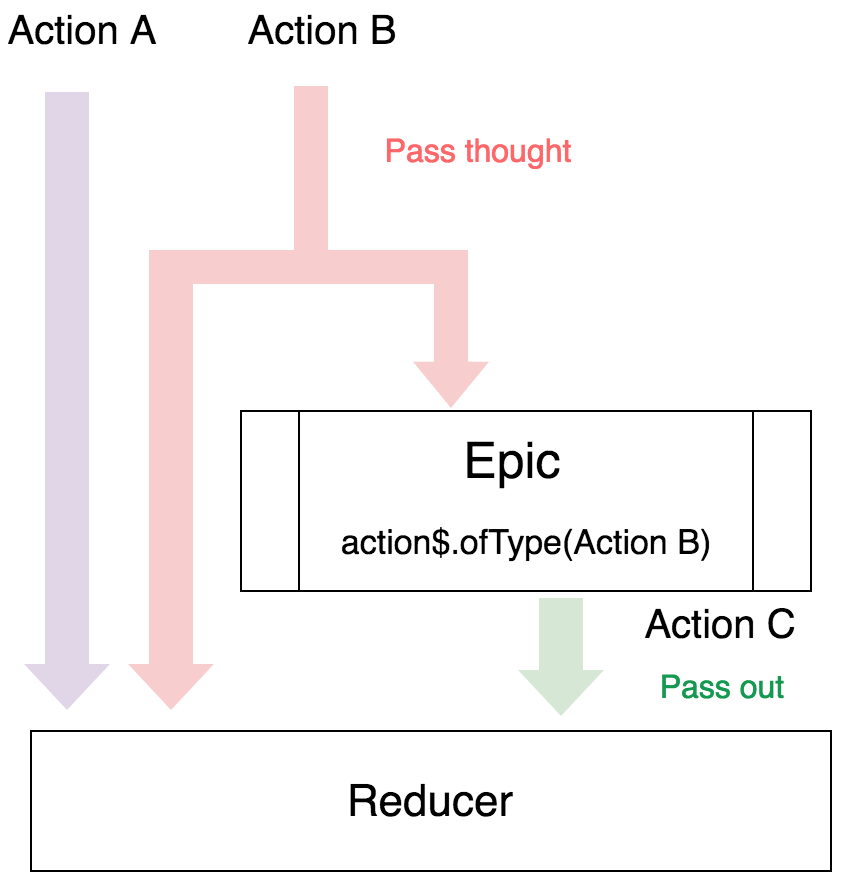


Figure 5 Epic's action workflow

## Inbound Action

An action emitted from A10Container; each action represents a new operation to the data.

## Outbound Action

An action emitted from the Epic after receiving an inbound action and processing its data.

## Dispatcher

A function that dispatches an action.

# A10 HTTP Actions

## Inbound Actions

GUI Framework provides three inbound actions for HTTP solution:

1. **A10ACTION\_TYPES.HTTP**

The action for HTTP request. Developers are encouraged to use the dispatcher **A10Dispatchers.httpRequest()** (provided by GUI Framework in detail below) to request HTTP data instead of dispatching this action yourself.

Developers can get **A10Dispatchers** in A10Container’s props.

Options for A10ACTION\_TYPES.HTTP

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Description | Optional |
| namespace | String | String[] | The key path for creating state data structure; it can be hierarchical, e.g., "foo.bar" or ["foo", "bar"]. |  |
| request | Object | Object[] | Function | The HTTP request configuration. This option supports three types - config object, multiple configs array, or a function returning a promise object.  GUI Framework uses [axios](https://github.com/axios/axios) as the default HTTP client. The config options available should be as same as axios' config. |  |
| interceptor | Object | A set of callbacks for capturing outbound actions. Developer can inject or even replace his/her customized actions via this option. | v |
| container-id | String <**private**> | An UID provided by GUI Framework for every A10Container. The UID will be passed in the option when calling **A10Dispatchers.httpRequest()**. |  |

1. **A10ACTION\_TYPES.HTTP.CANCEL**

This action is for canceling a HTTP request and is often used while switching pages or interrupting UI operations.

Developers are encouraged to use the dispatcher **A10Dispatchers.cancelHTTPRequest(),** provided by the GUI Framework to cancel an HTTP request, instead of creating an A10ACTION\_TYPES.HTTP.CANCEL action directly.

Options for A10ACTION\_TYPES.HTTP.CANCEL

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Description | Optional |
| namespace | String | string[] | A key to cancel request(s).  This should match the namespace for the requests you want to cancel.  If it is null, all namespaces corresponding to the container-id will be cancelled. | v |
| container-id | String <**private**> | An UID provided from the GUI Framework. | v |

Note: GUI Framework will cancel all pending HTTP if options parameter is omitted.

1. **A10ACTION\_TYPES.DATA.GC**

GUI Framework will dispatch this GC (Garbage Collection) action when a container gets unmounted. GUI Framework cleans up A10DataReducer’s state data in Redux store afterwards. It is always executed by default.

Options for A10ACTION\_TYPES.DATA.GC

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Description | Optional |
| container-id | String | An UID provided by framework |  |

## Outbound Actions

Epic will pass out a new action after receiving an inbound action; we call the passed-out action outbound action.

GUI Framework provides three outbound actions.

1. **A10ACTION\_TYPES.HTTP.ERROR**

This action will be emitted when a HTTP error or program error is encountered.

Options for A10ACTION\_TYPES.HTTP.ERROR

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Description | Optional |
| namespace | String[] | A key provided by developer from an A10 HTTP request action. |  |
| response | Object | An error response object. This is provided from axios.  https://github.com/axios/axios#handling-errors. |  |

1. **A10ACTION\_TYPES.DATA.SET**

A10HTTPEpic will dispatch this action after an HTTP request is successful. A10DataReducer and A10GCReducer will receive A10ACTION\_TYPES.DATA.SET and then update the store state. A10DataReducer is responsible for saving HTTP response data, whereas A10GCReducer is only in charge of recording namespace.

Options for A10ACTION\_TYPES.DATA.SET

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Description | Optional |
| namespace | String[] | A key provided by developer from an A10 HTTP request action. |  |
| data | Any | A data to store in A10DataReducer. |  |
| container-id | String | An UID provided from GUI Framework |  |

1. **A10ACTION\_TYPES.DATA.REMOVE**

This action is used to clear data in Redux store.

Options for A10ACTION\_TYPES.DATA.REMOVE

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Description | Optional |
| namespace | String[] | A key provided by developer from an A10 HTTP request action. |  |
| container-id | String | An UID provided from GUI Framework | v |

# Dispatchers - A10Dispatchers

GUI Framework provides two dispatchers in every A10Container for two inbound actions types - **A10ACTION\_TYPES.HTTP** and **A10ACTION\_TYPES.HTTP.CANCEL**.

We recommend developers to use the dispatchers we provide instead of dispatching the actions themselves.

Here is the instance for using A10Dispatchers in an A10Container:

Const { A10Dispatchers } = this.props

## A10Dispatchers.httpRequest()

Interface for httpRequest()

|  |
| --- |
| function httpRequest(options): Promise |

Options for httpRequest()

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Description | Optional |
| namespace | String | string[] | The key path for creating state data structure, it can be nested. e.g., "foo.bar" or ["foo", "bar"]. |  |
| request | Object | Object[] | Function | The HTTP request configuration. This option supports three types - config object, multiple configs array, or a function returning a promise object.  GUI Framework uses [axios](https://github.com/axios/axios) as default HTTP client. The config options should be same as axios' config. |  |
| interceptor | Object | A set of callbacks for capturing outbound actions. Developer can inject or even replace his/her customized actions via this option. | v |

Examples of using options.request:

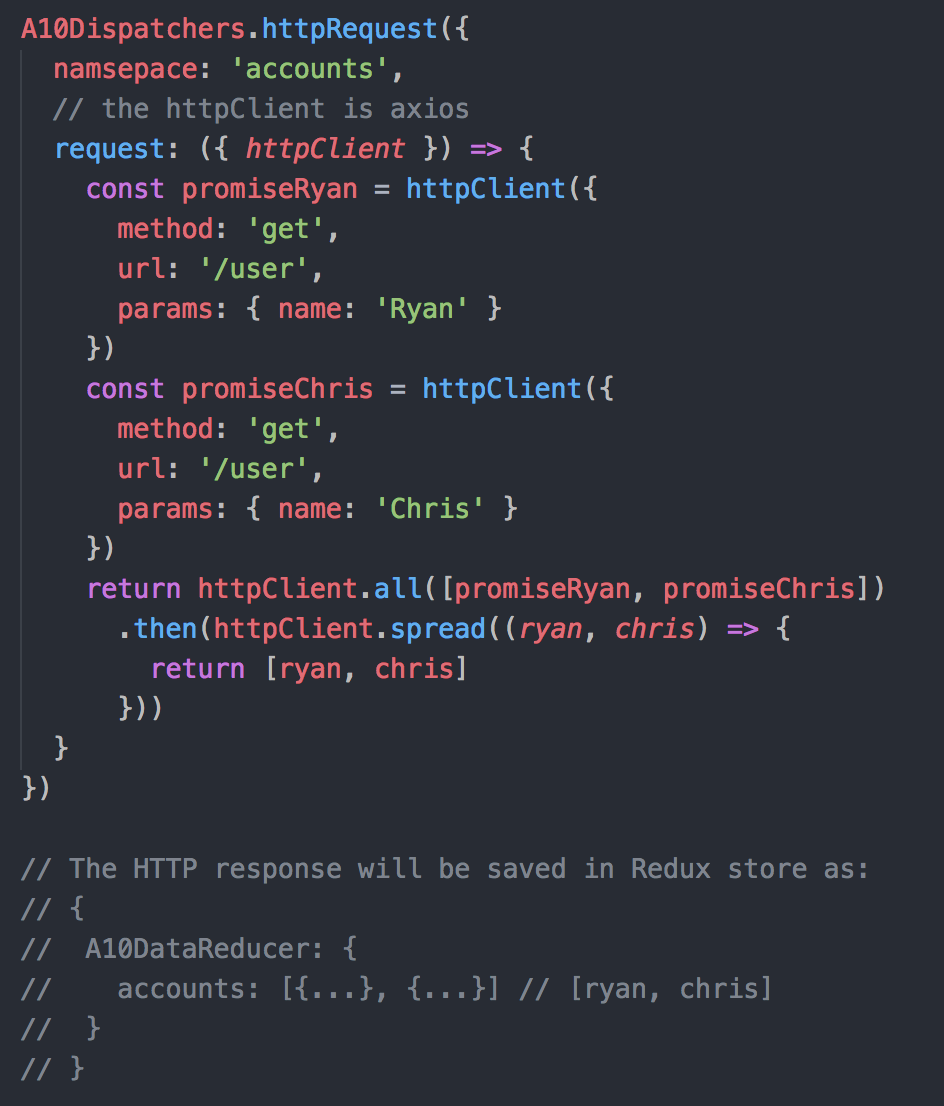
1. Single HTTP request



1. Multiple concurrent HTTP requests (Based on axios.all)



1. Customized HTTP request



For more axios usage please refer to https://github.com/axios/axios.

## A10Dispatchers.cancelHTTPRequest()

Interface for cancelHTTPRequest()

|  |
| --- |
| function cancelHTTPRequest(options): void |

Options for cancelHTTPRequest()

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Description | Optional |
| namespace | String | string[] | This should match the namespace for the requests you want to cancel.  If it is null, all namespaces corresponding to the container-id will be cancelled. |  |
| all | Boolean | Set true to cancel all HTTP requests that have not returned. Default is false. | v |

# Middlewares

## A10HTTPEpic

A10HTTPEpic is responsible for HTTP request processing and generic error handling. This Epic captures "A10ACTION\_TYPES.HTTP" action and then performs the following steps.

1. Process the request information that was provided by developer and fetch the data.
2. Try to intercept action "A10ACTION\_TYPES.HTTP.CANCEL". Once this action is received, use the options provided by developer to determine which HTTP request(s) should be canceled.
3. If the HTTP request result is successful, A10HTTPEpic will return another action "A10ACTION\_TYPES.DATA.SET" to reducer to save data to store.
4. If the HTTP request fails, A10HTTPEpic will return another action "A10ACTION\_TYPES.HTTP.ERROR". Developer needs to capture this action in their reducer because GUI Framework does not handle this.

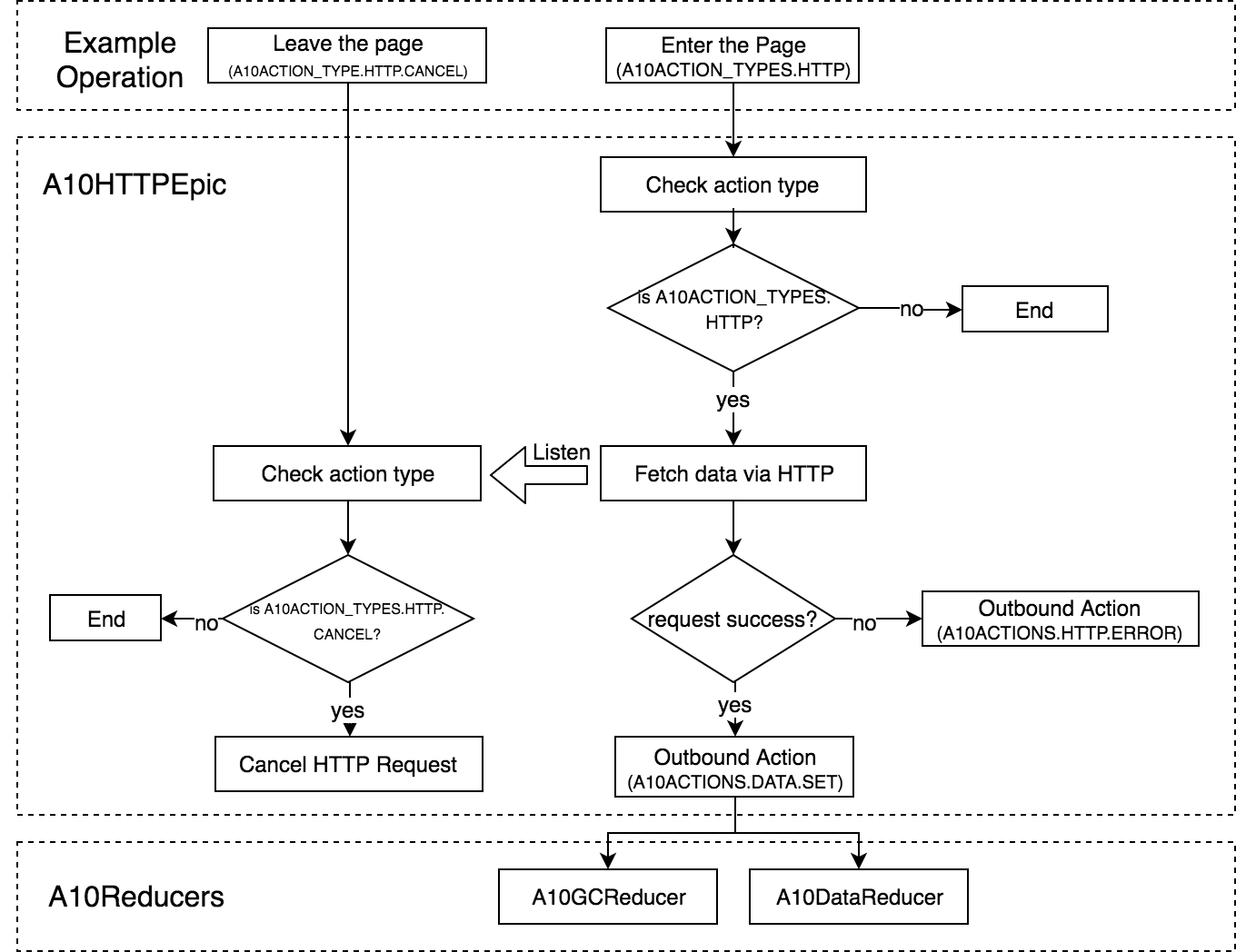


Figure 6 Workflow of A10httpepic

An App Container sample code using A10Dispatchers.requestHTTP

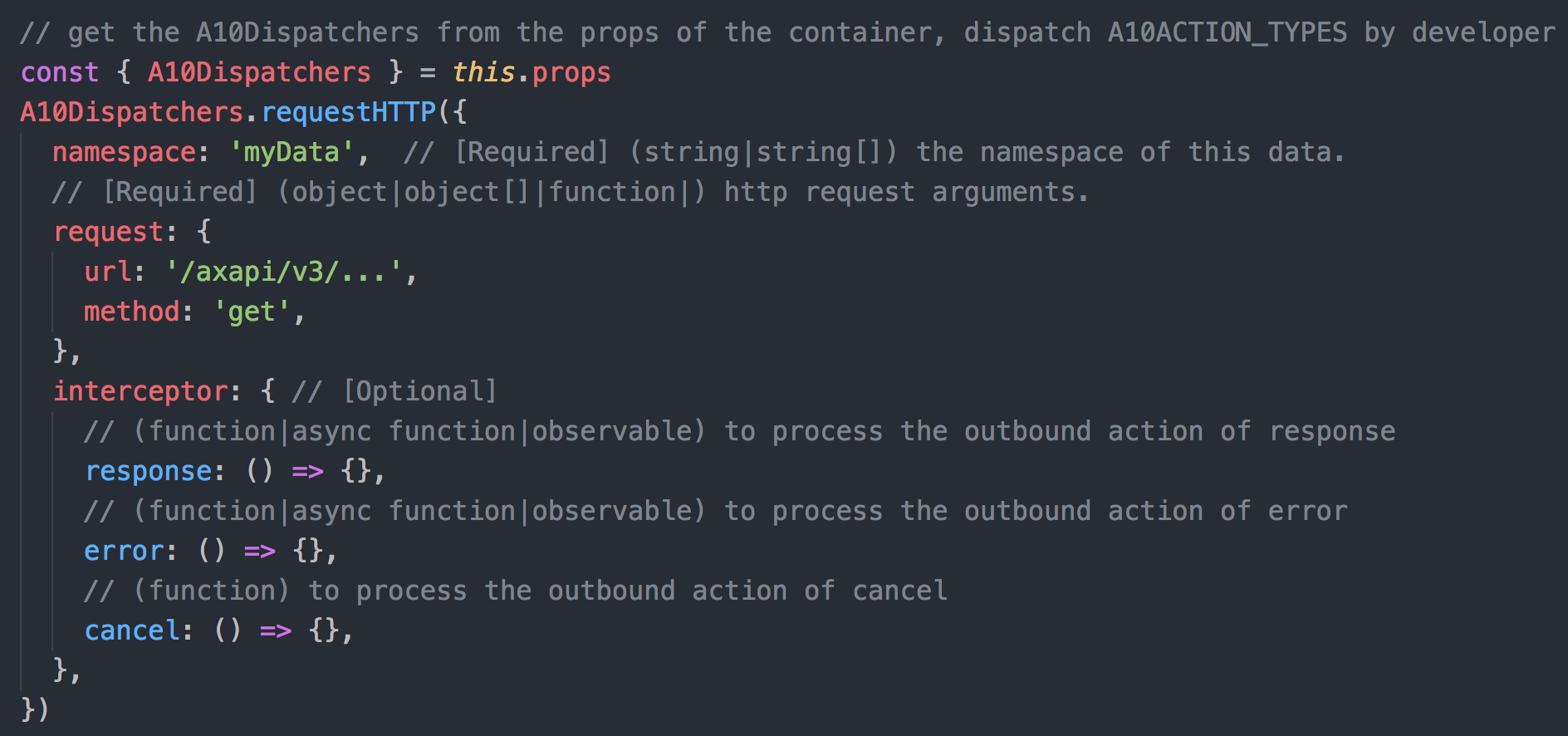


Figure 7 Dispatching a http action

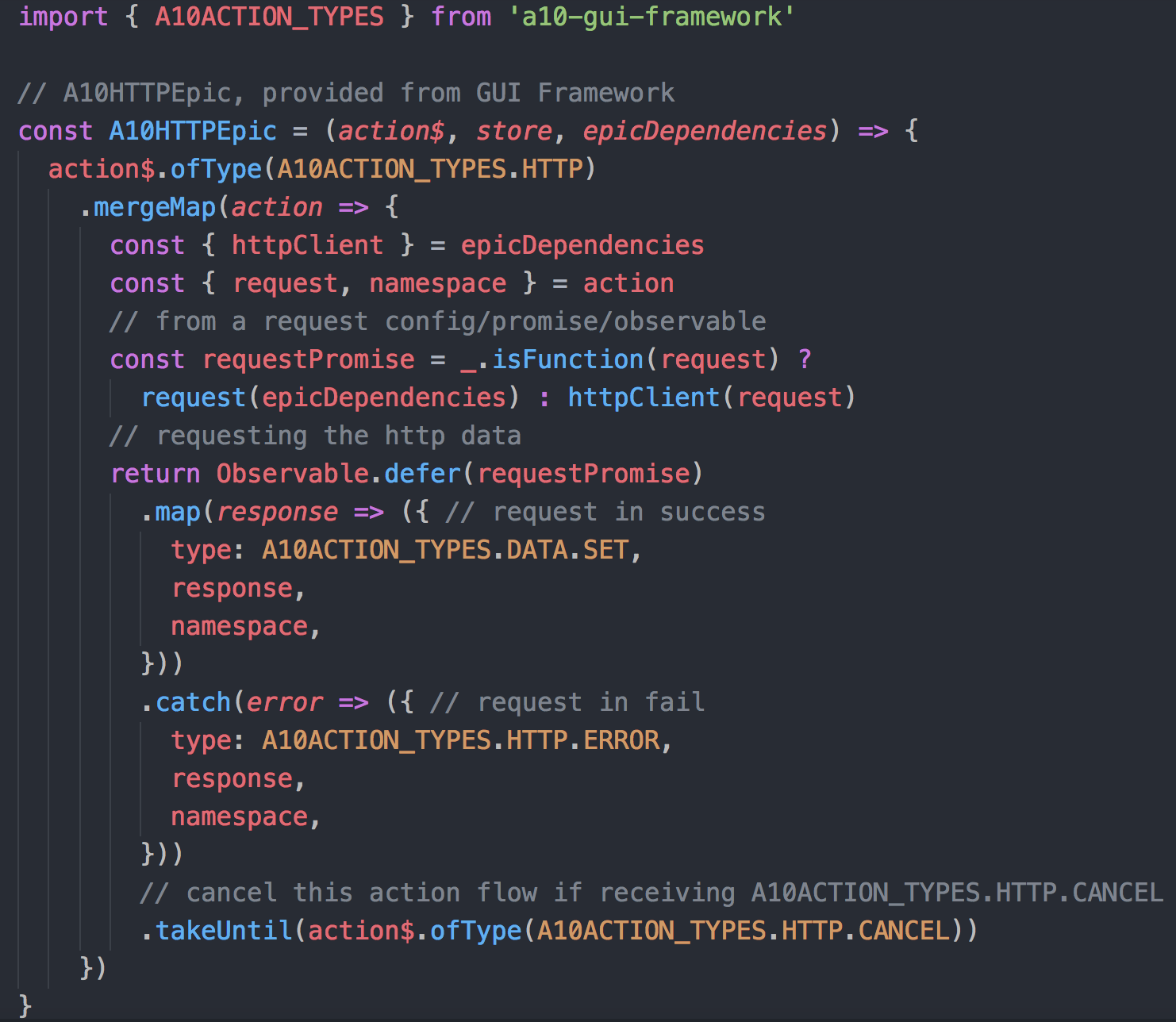
A10HTTPEpic code in GUI Framework (pseudo code)

Figure 8 The epic of A10HTTP in redux middleware

### Interceptor of A10HTTPEpic

Developers may not be satisfied with the simple outbound actions provided by A10HTTPEpic and may instead want to have finer control in outbound actions of an Epic.

To fulfill the above goal, A10HTTPEpic provides an interceptor for developers wanting to control all the flows of A10HTTPEpic. The interceptor can intercept all outbound actions "error", "response", and "cancel". Developers are able to not only modify but also chain (or replace via observable) any outbound action that they want to control as long as they pass a parameter "interceptor" to the action A10HTTP.

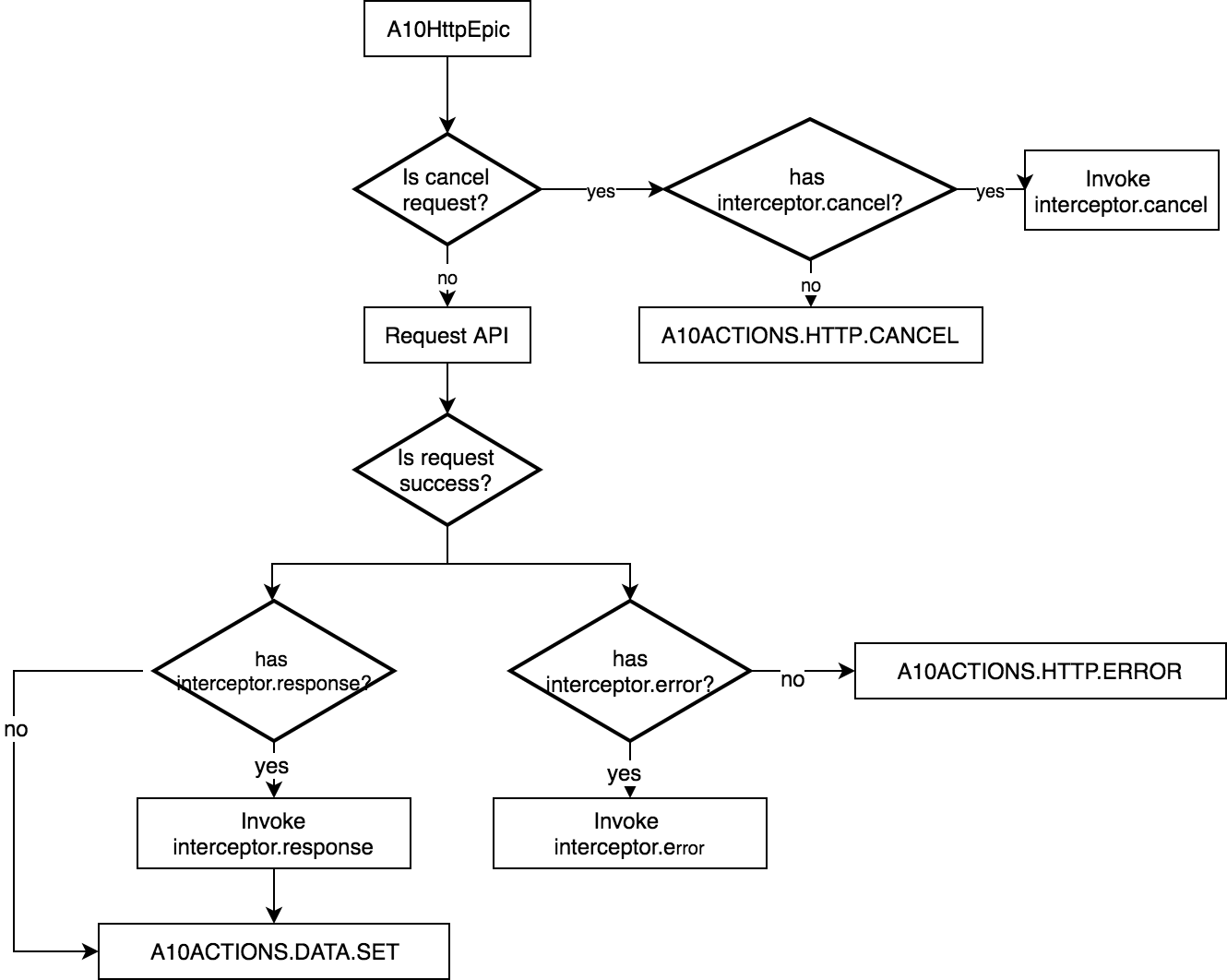


Figure 9 INTERCEPTOR workflow in A10HttpEPIC

The interface of the interceptor.

|  |  |  |
| --- | --- | --- |
| Interceptor Name | Description | Overridden Outbound Action |
| Error | Process the response data if the http request failed. | A10ACTION\_TYPES.HTTP.ERROR |
| Response | Deal with the response data if the request is in success. | A10ACTION\_TYPES.DATA.SET |
| Cancel | Modify or chain the cancel action. | A10ACTION\_TYPES.HTTP.CANCEL |

An example of interceptor usage.

Figure 10 Simple example of interceptor usage

## A10GCEpic

This is a middleware for garbage collection in Redux store. A10GCEpic gets the container-id from action **A10ACTION\_TYPES.DATA.GC** and checks the data whether it belongs to the container-id. A10GCEpic will dispatch out an action notifying A10DataReducer to clean up the data If the data is not referenced by any container.

Here are the steps when A10GCEpic receives action A10ACTION\_TYPES.DATA.GC:

1. Look for all namespaces with corresponding container-id.
2. To check the count of each namespace and determine what data should be removed, if the count is zero then continue to Step 3.
3. Send **A10ACTION\_TYPES.DATA.REMOVE** action to remove the unused HTTP response data from the reducer

In the flow chart below, A10GCReducer always decrease the count of namespace before A10GCEpic starts.

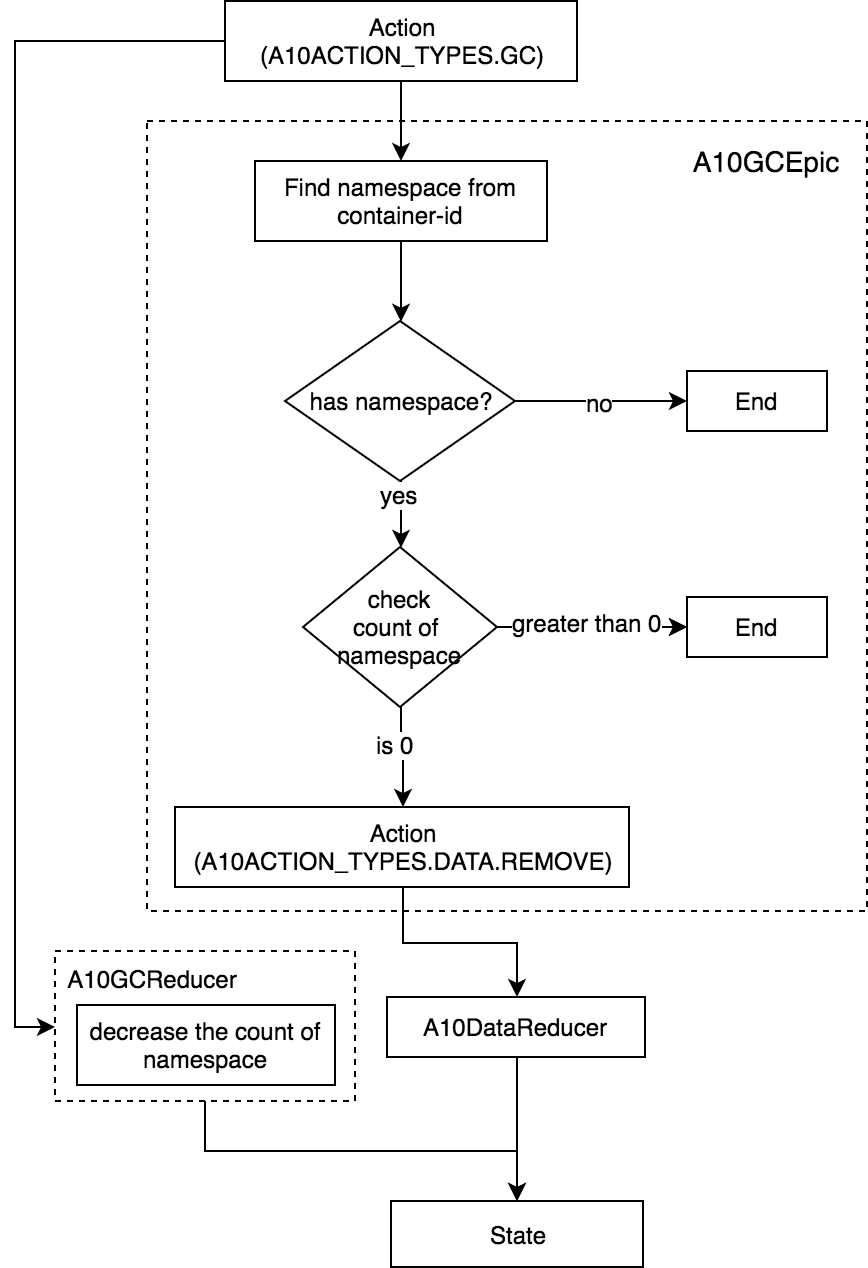


Figure 11 GARBAGE COLLECTION for HTTP DATA

# Reducers

Framework has two internal reducers, one is to store state data, the other is to remove unused state data.

## A10DataReducer

A10DataReducer uses **Immutable.js** to store data from actions of type **A10ACTION\_TYPES.DATA.SET**, where each action contains a namespace and a piece of data. A10DataReducer uses the namespace as key to store the data into Redux store.

NOTE: The state of A10Reducer is an **immutable object**. Developer must use immutable API to retrieve the data in A10Reducer.

A10DataReducer code in GUI Framework (pseudo code).

Figure 12 The reducer of A10DataReducer

The mapStateToProps function in the container

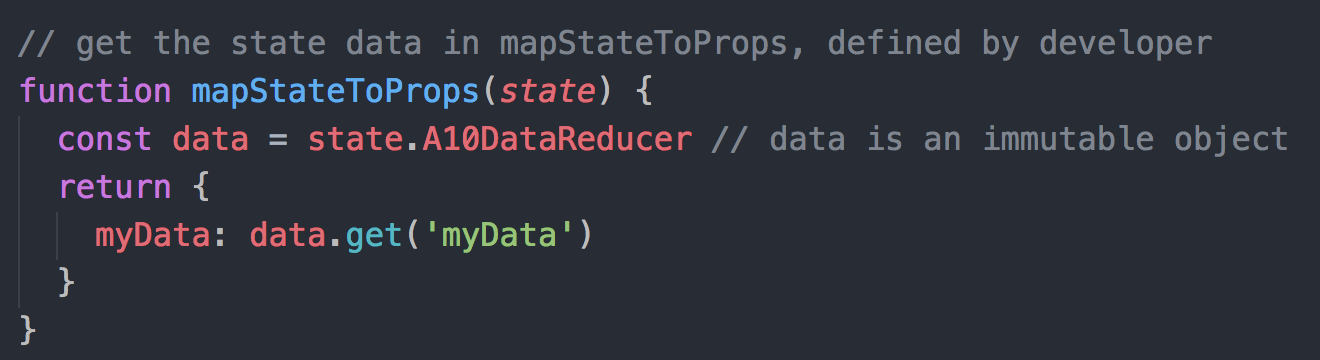


Figure 13 Get the state data in mapStateToProps

Here is a sample data structure of A10DataReducer. The namespaces in below state data are "slb.virtual-server.vs1" and "cgn":



Figure 14 The data structure of A10DataReducer in Redux store

## A10GCReducer

After A10HTTPEpic dispatched A10ACTION\_TYPES.DATA.SET action, A10GCReducer will initialize the namespace name/count table and save to store state.

The table is later used by **A10GCEpic** to determine which state data should be removed if it is unused.



Figure 15 namespace Collection

A10GCReducer contains two data structures: the first is the container to namespace table, i.e., nsOwnerRecord, and the second is namespace to reference count table, i.e., nsUsage.

Here is the example that demonstrates the data structure of A10GCReducer. It contains "slb.virtual-server.vs1" and "cgn" namespaces, where in "slb.virtual-server.vs1" is used for "Container A", "cgn" is used for "Container A" and "Container B".

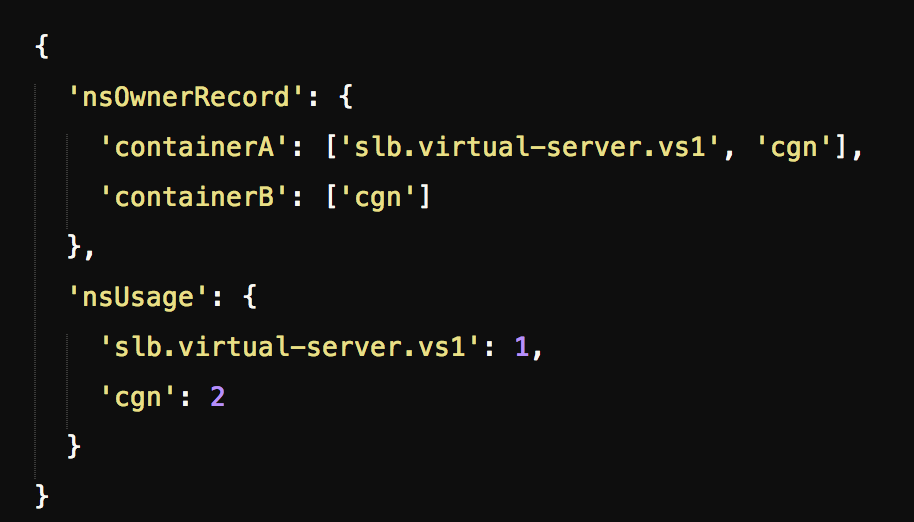


Figure 16 A10GCRReducer Structure

# Namespace

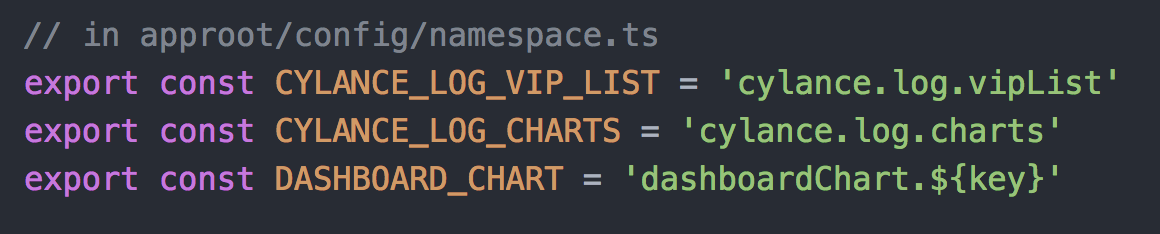
As we mentioned above, namespace is a key path for creating state data structure. We highly recommend organizing all of HTTP action namespaces into a file for easier debugging and maintenance.

## Setup namespace in Global Config

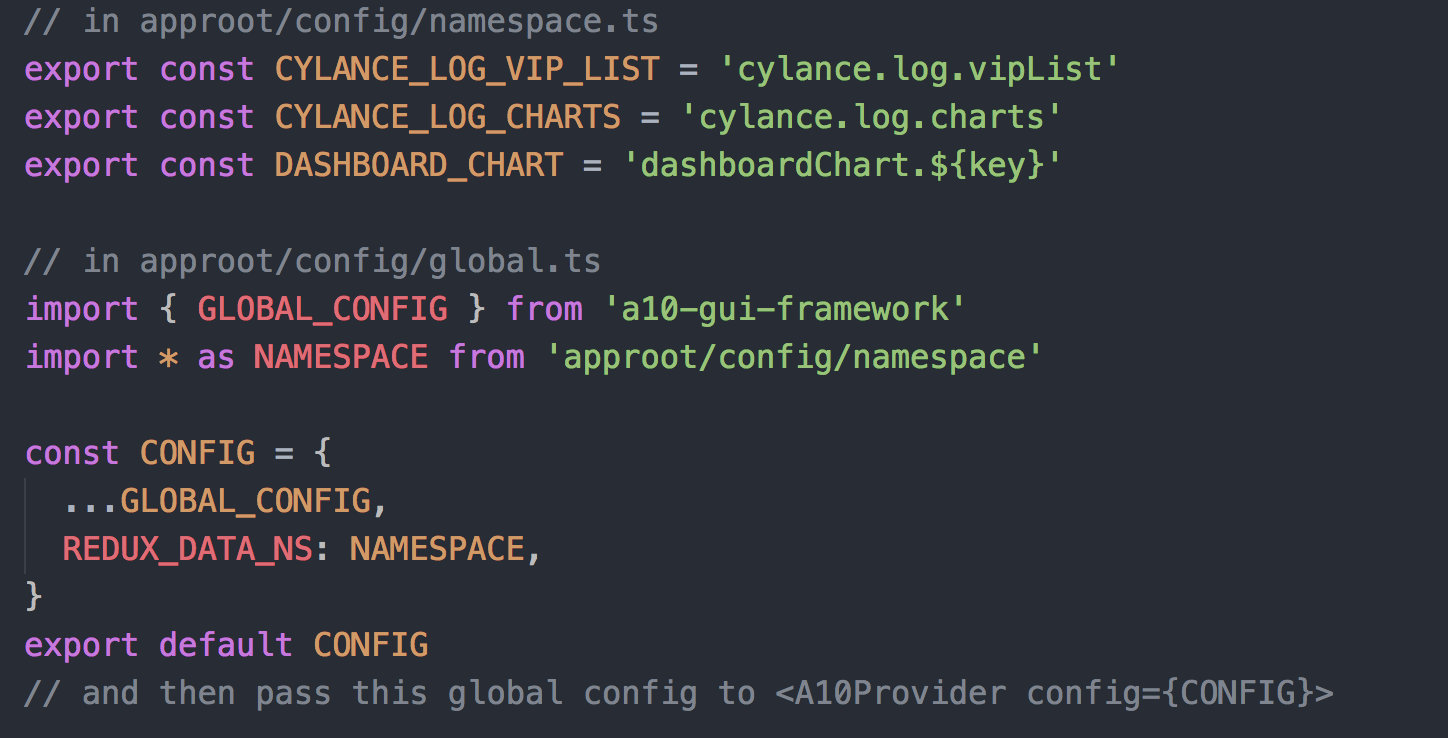
GUI Framework offers a facility for checking namespace conflicts as long as developer places namespace in global config **REDUX\_DATA\_NS**. Conflict checking is enabled by setting **DEBUG** to **true** in global config.

The following example is our recommended approach for setting up your HTTP namespaces:

1. Define your namespaces in a config file.



1. Assign all of namespaces in REDUX\_DATA\_NS.

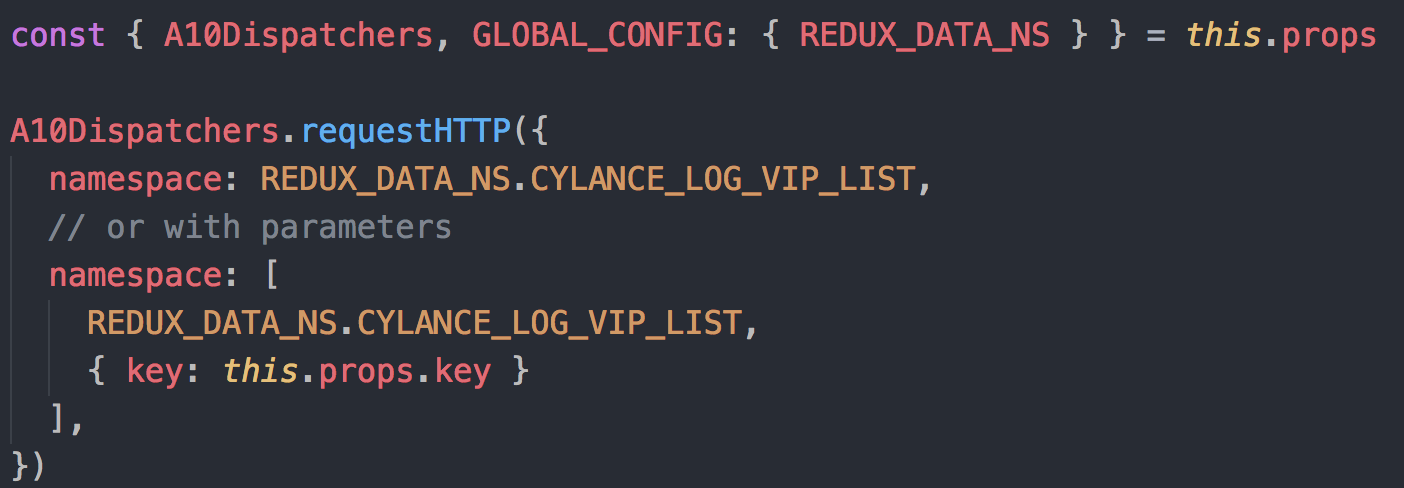


## Retrieve namespace in A10Container

Developer gets global config from A10Container props and REDUX\_DATA\_NS from global config.

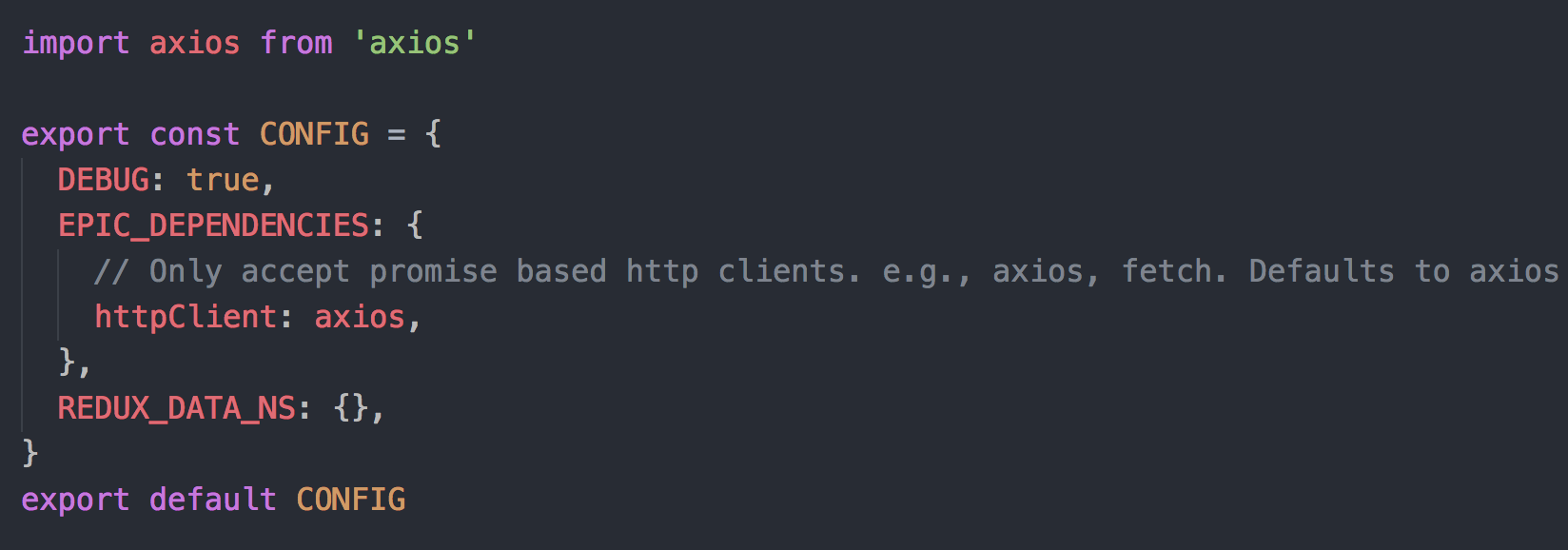
const { GLOBAL\_CONFIG } = this.props

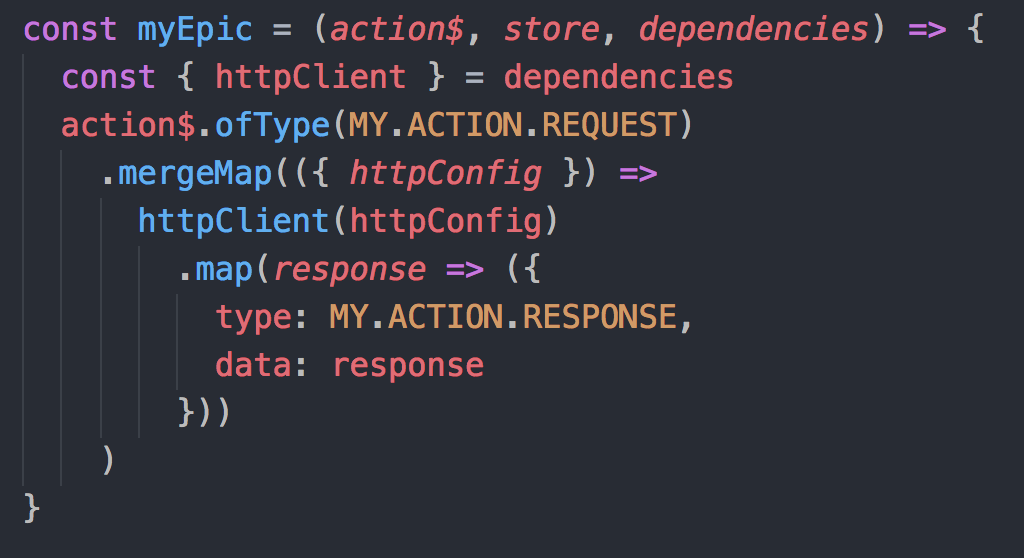
const { REDUX\_DATA\_NS } = GLOBAL\_CONFIG

An example of getting namespace from REDUX\_DATA\_NS in an A10Container.

# Epic Dependencies

Injecting your dependencies into your Epics can help with testing. Developer can put dependencies definition in Global Config. By default, GUI Framework uses "axios" as httpClient.



Get httpClient from epic dependencies.

# Using A10 HTTP Redux Solution in The Real World

Cylance Dashboard is a sample App that illustrates how A10 HTTP Redux Solution works in a composite container.

## Design Principle

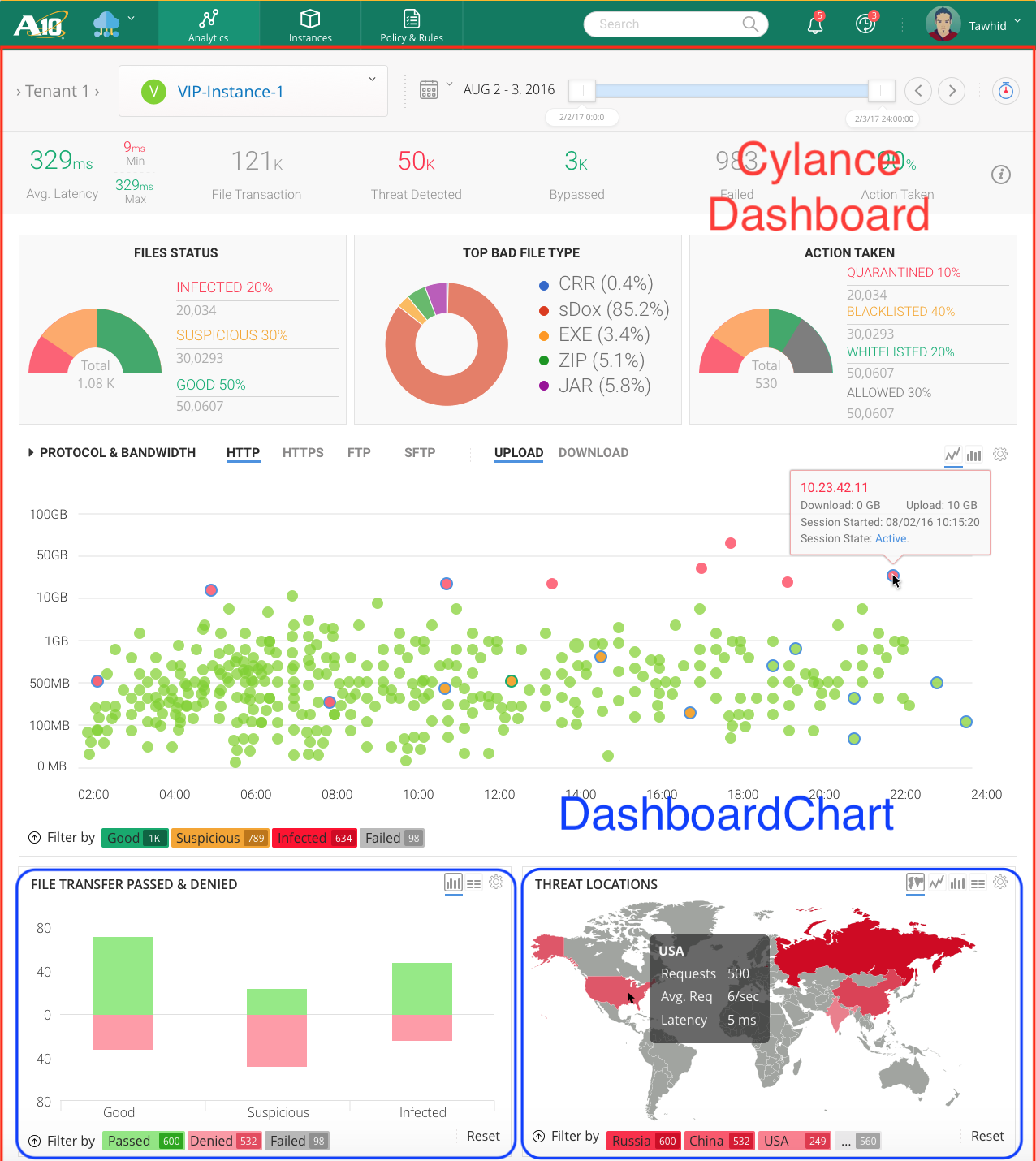
There are two containers in the Cylance App, i.e., CylanceDashboard and DashboardChart. DashboardChart is able to be reused in other Apps or containers. CylanceDashboard is a parent of DashboardChart. "File Transfer Passed & Denied" and "Threat Locations" are instances of DashboardChart.

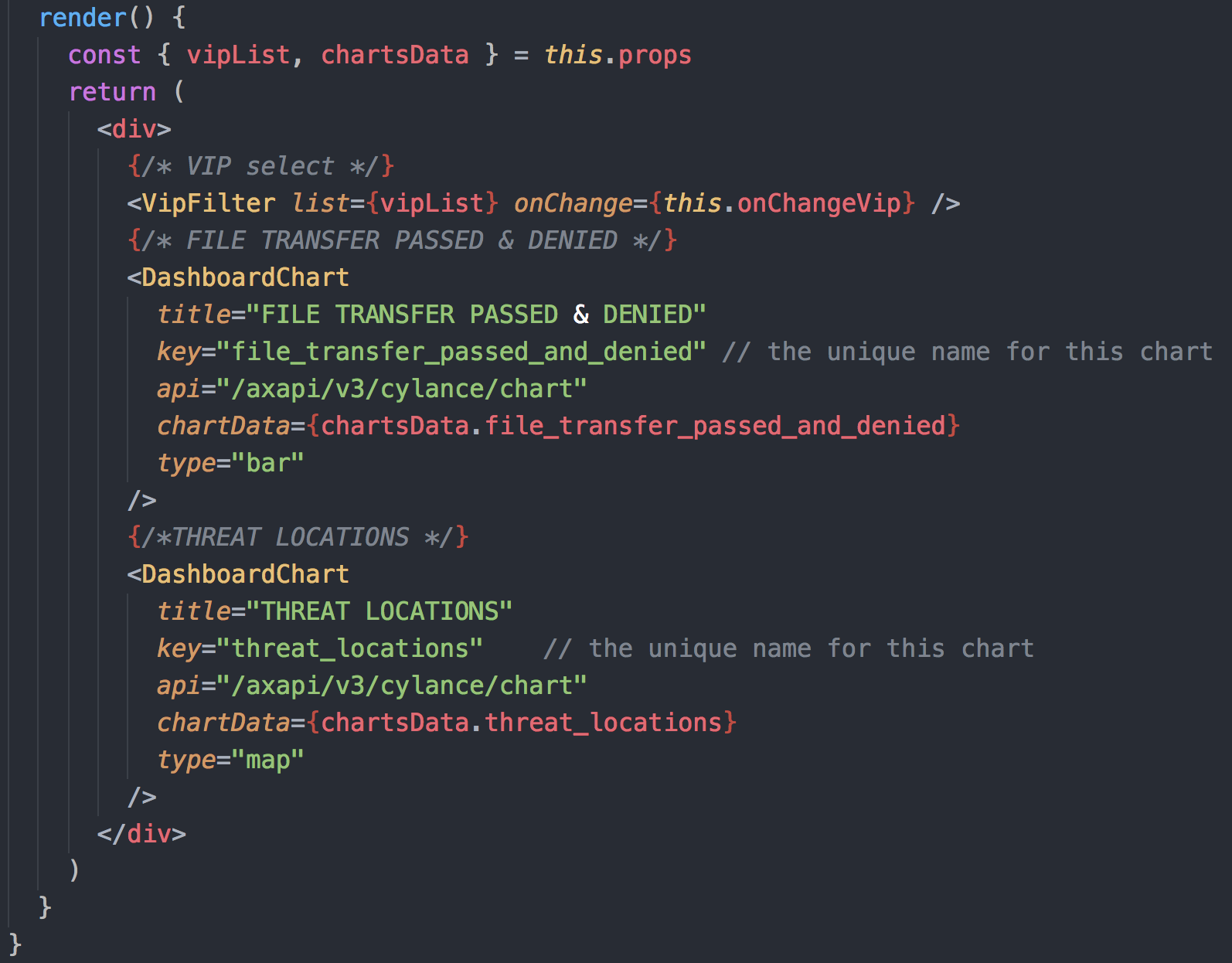
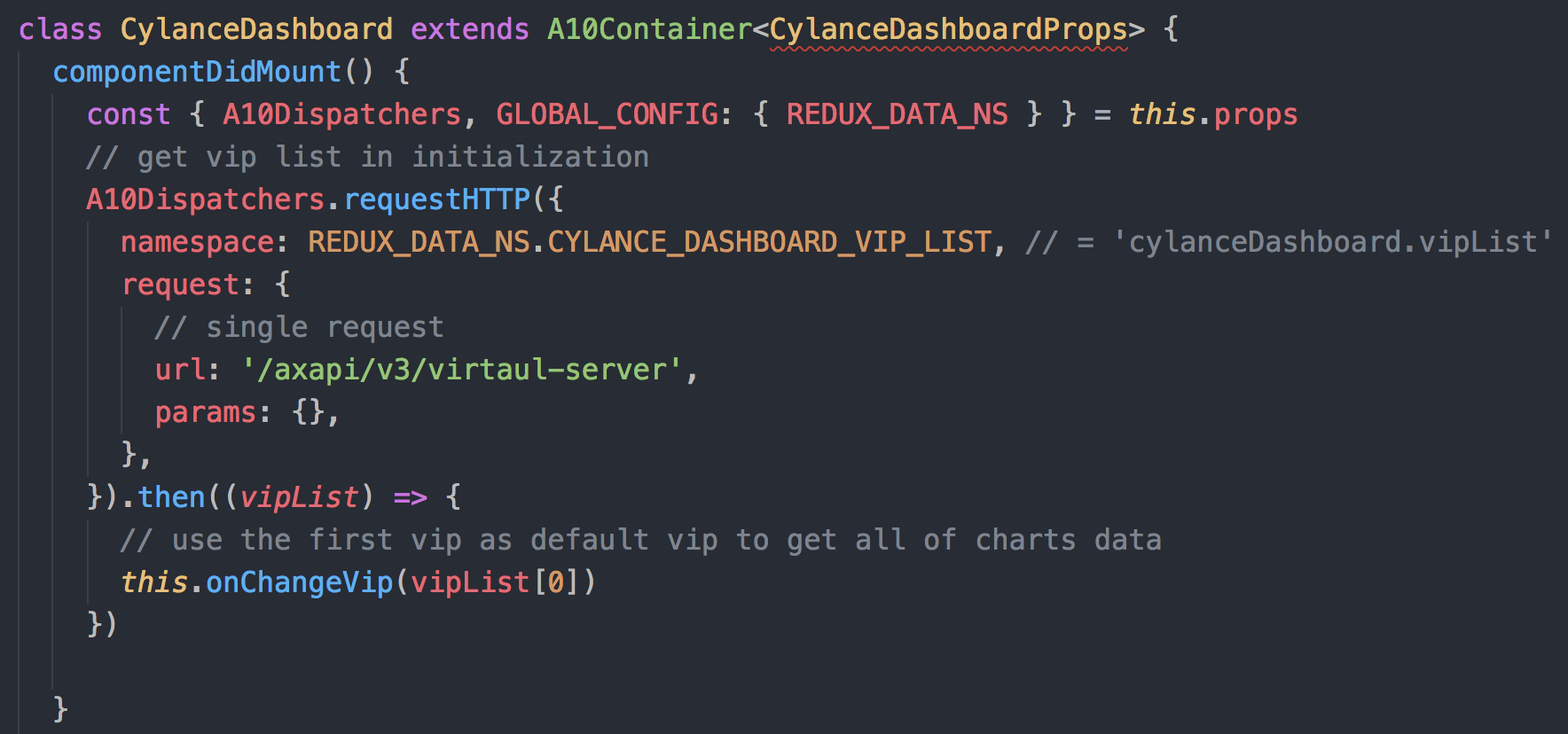
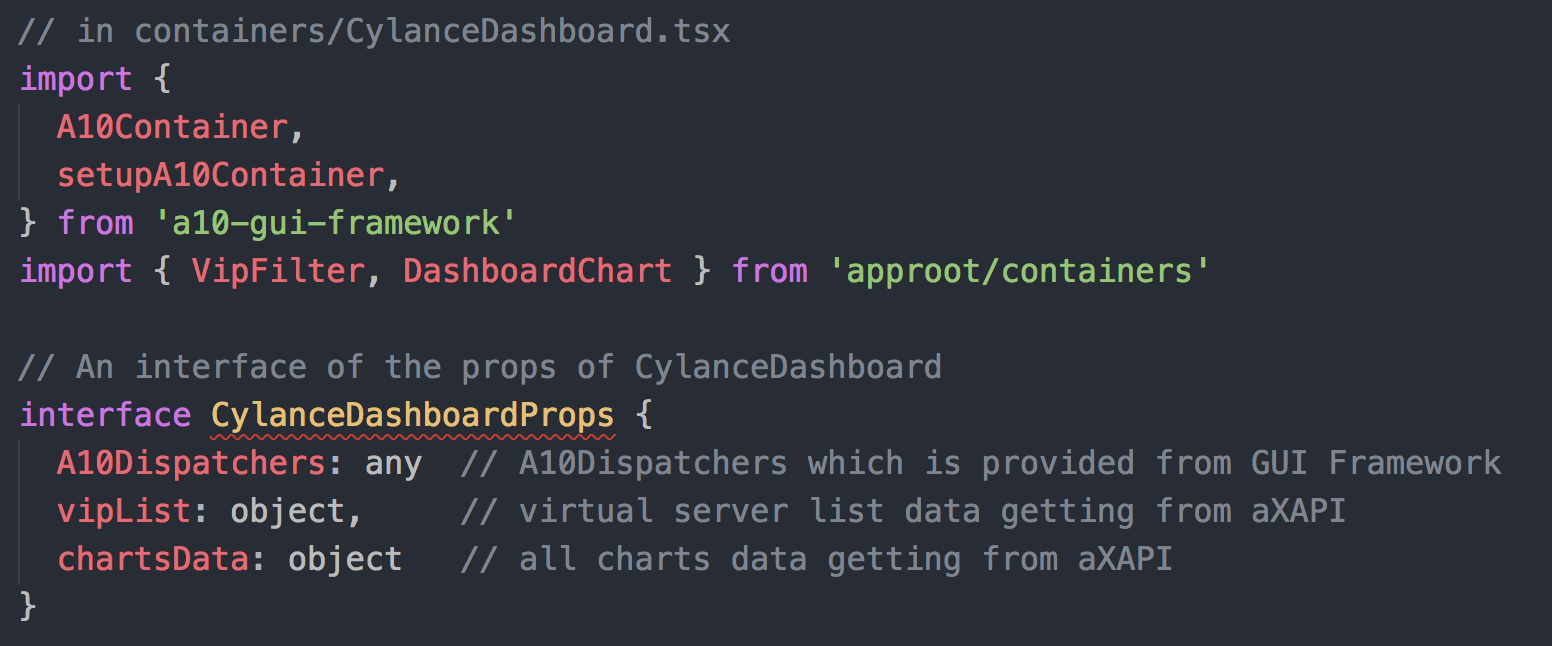
**CylanceDashboard workflow:**

1. Get a vip list by dispatching A10Dispatchers.requestHTTP during initialization.
2. Get all charts data by dispatching A10Dispatchers.requestHTTP and then broadcast to the two DashboardCharts.
3. Update charts data if user changes vip.

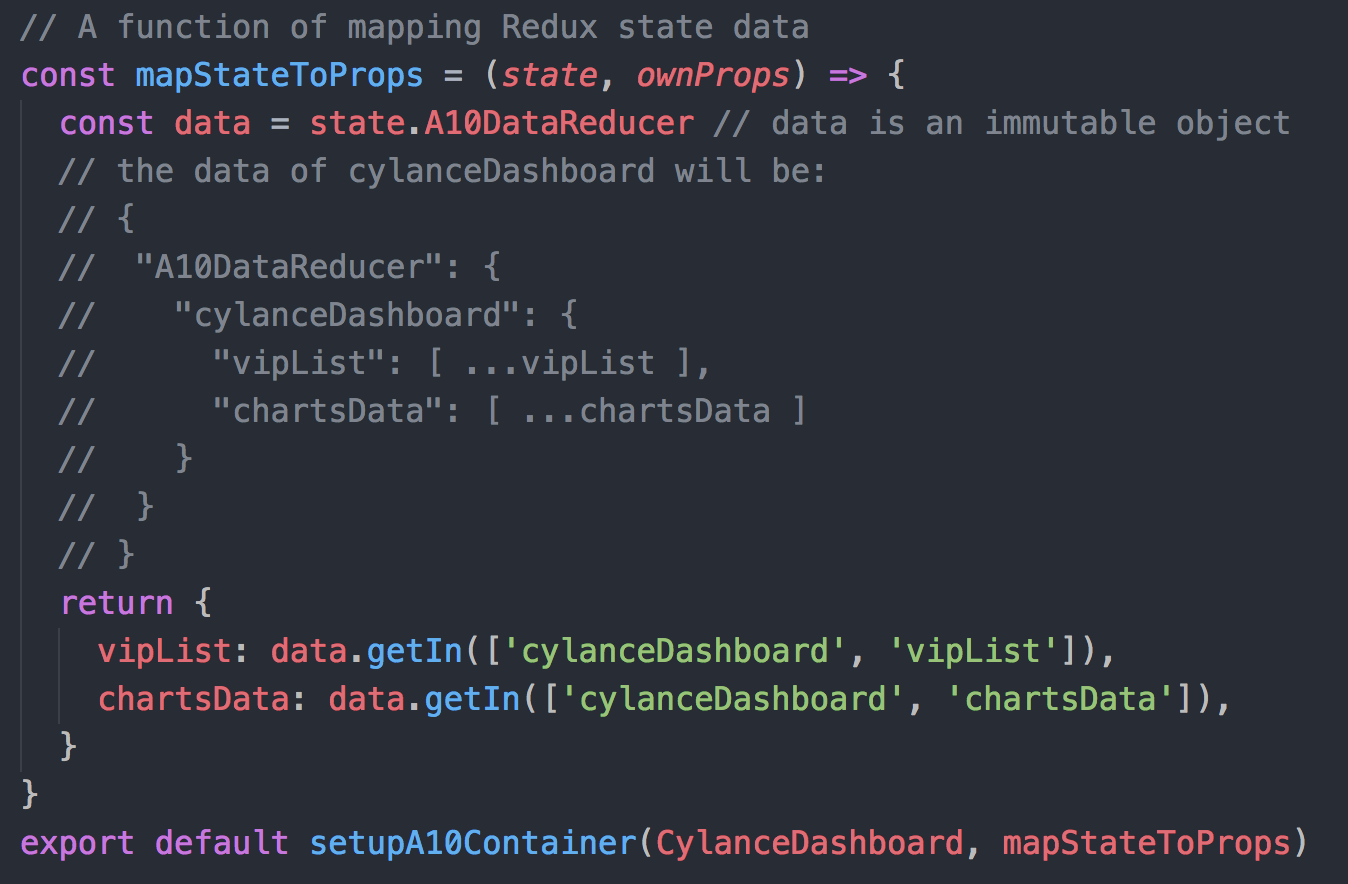
**DashboardChart workflow:**

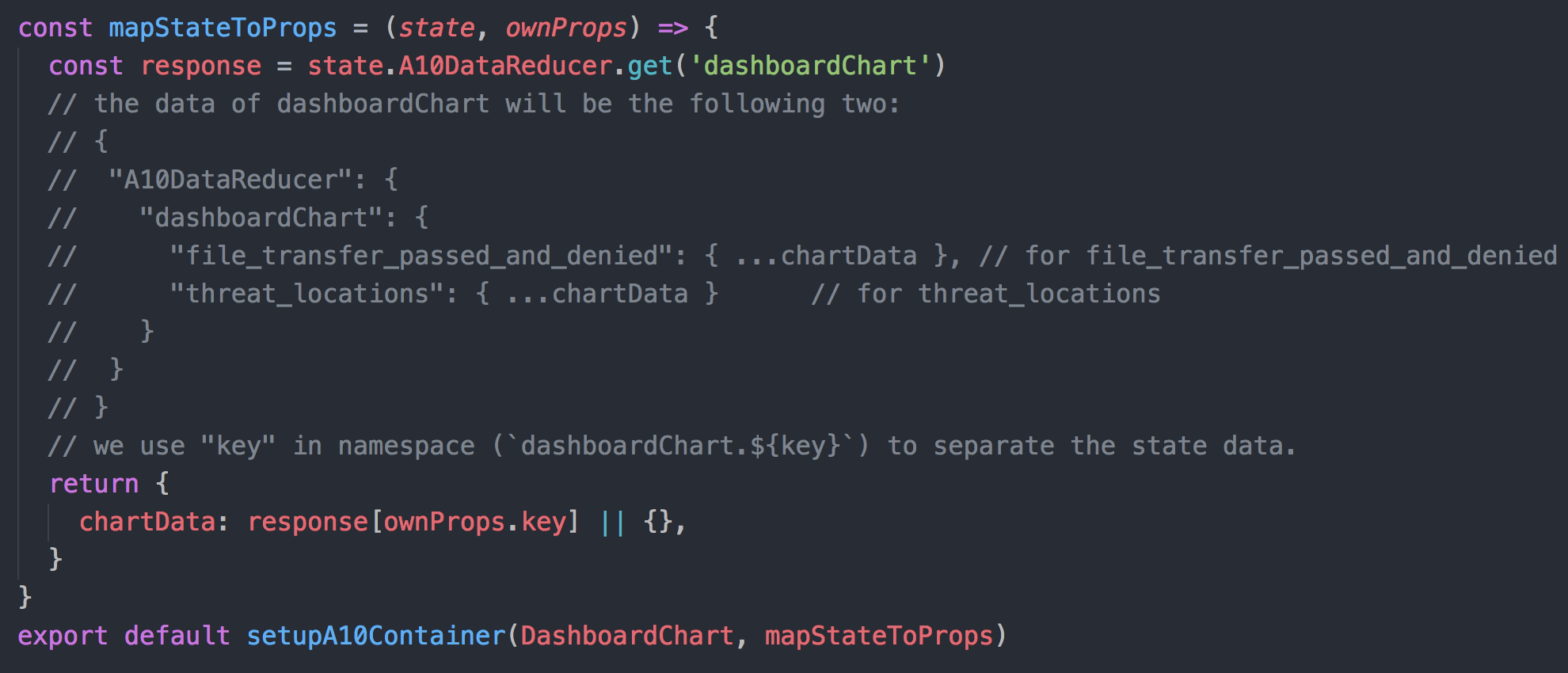
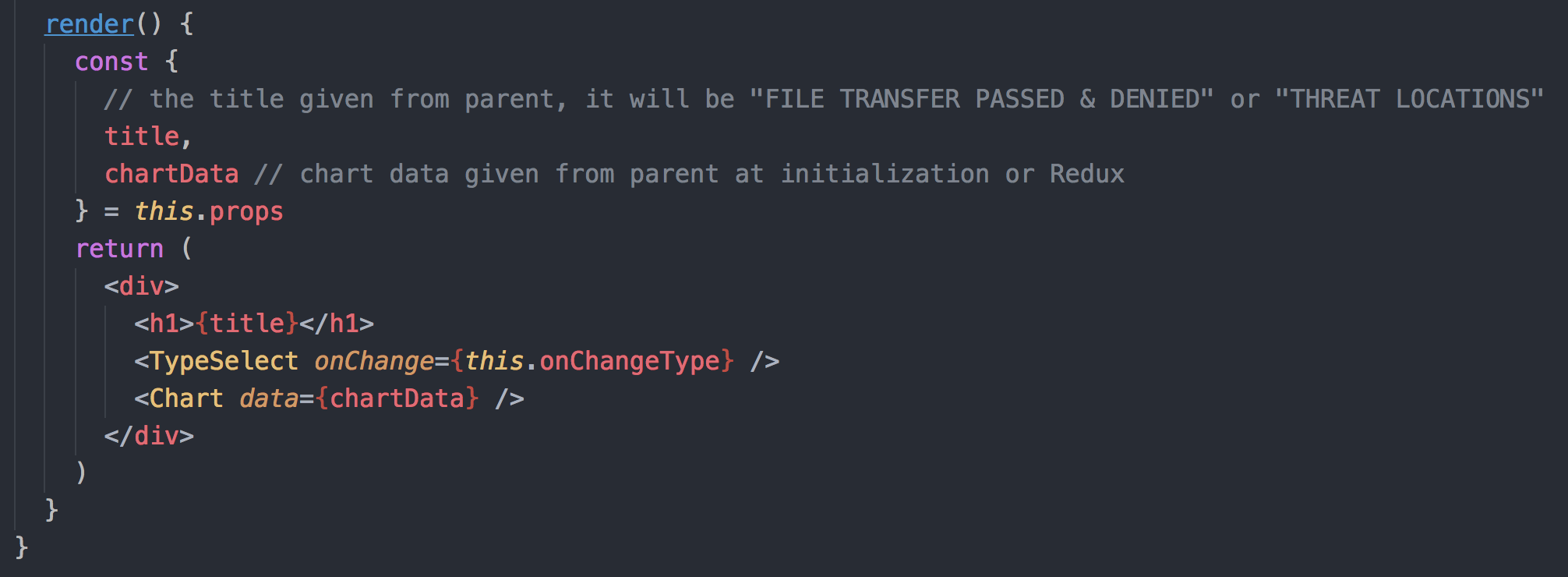
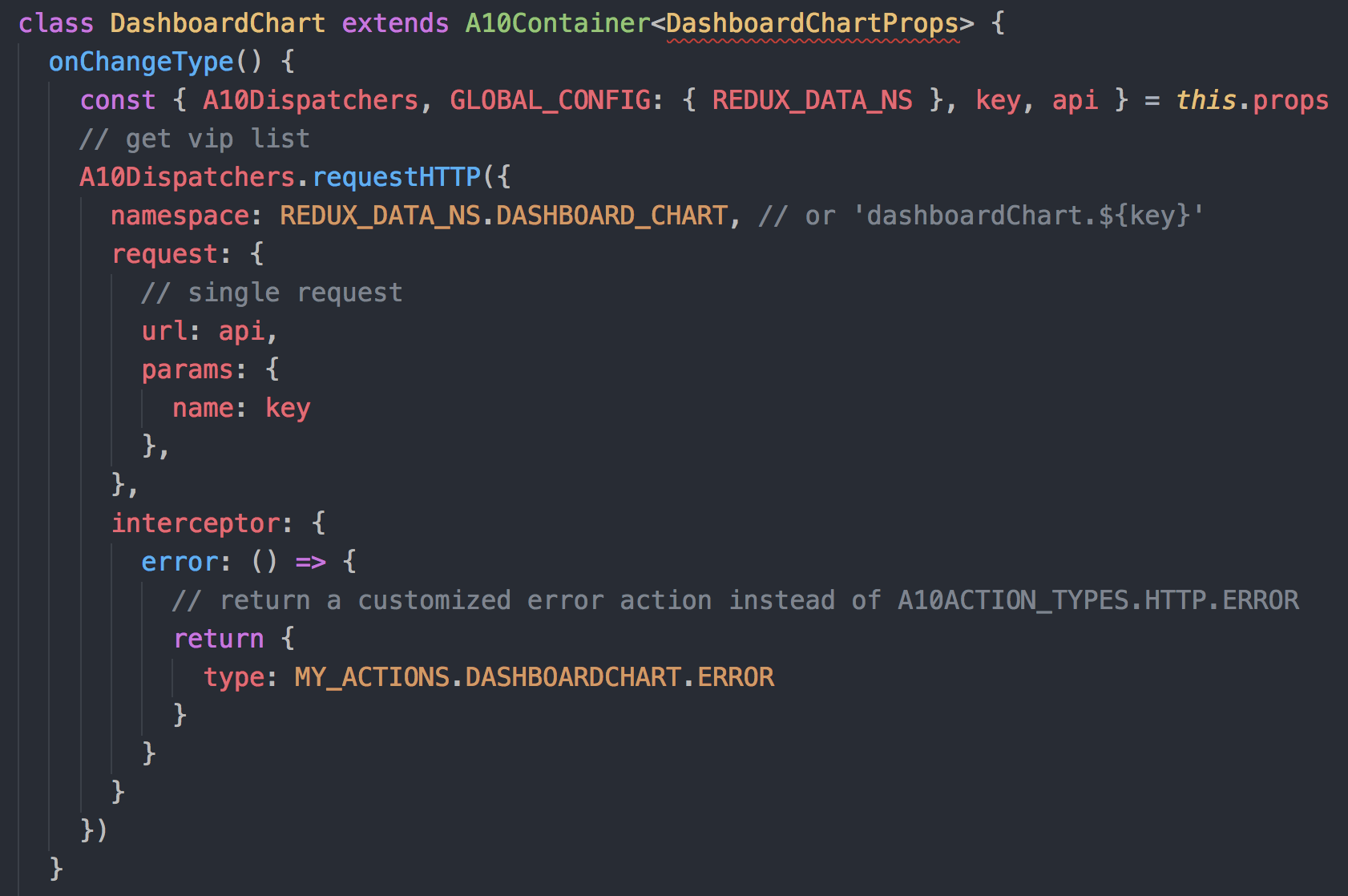
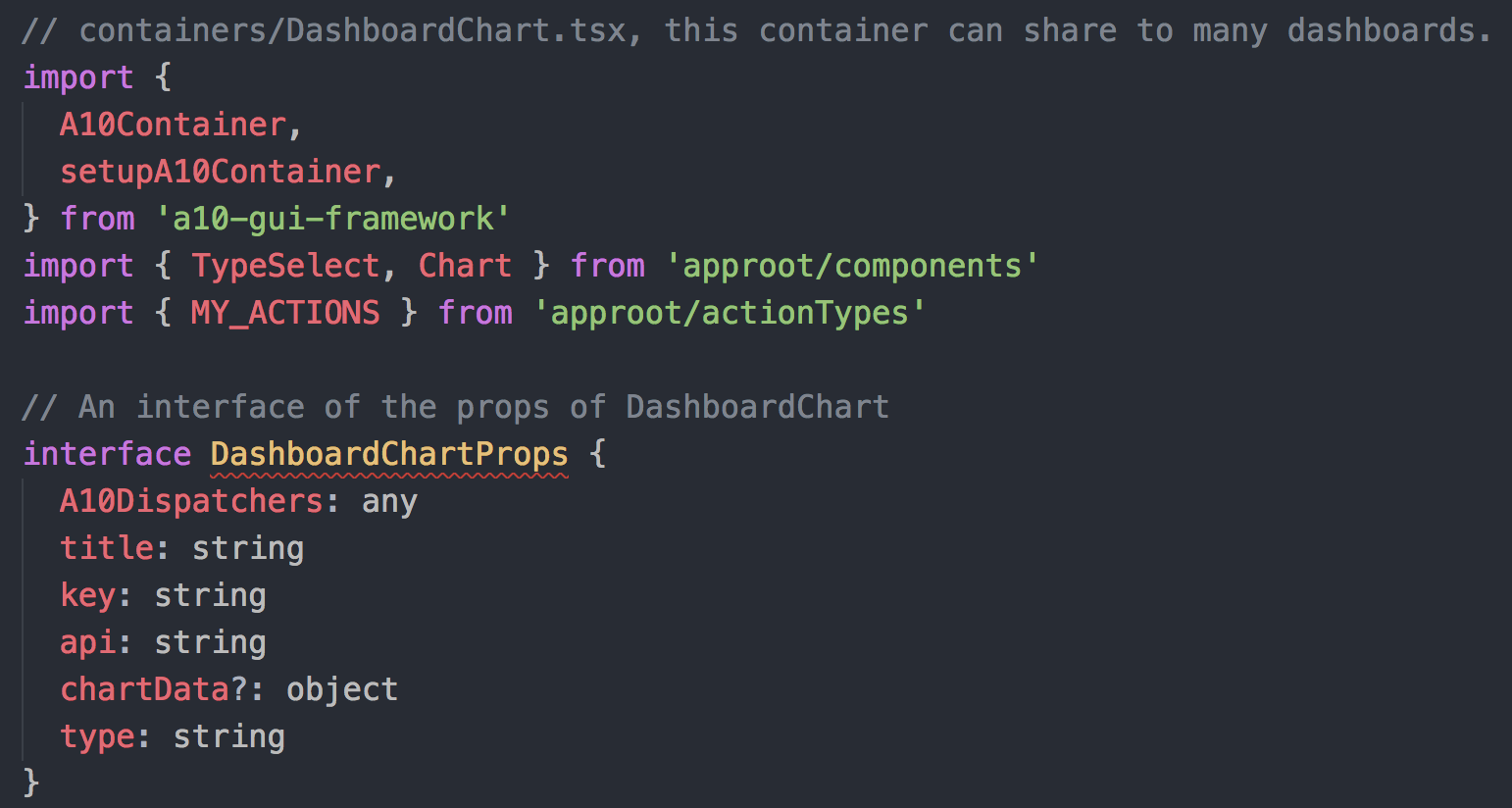
1. Render chart graphics during initialization or when user changes the vip.
2. Re-render chart data by dispatching A10Dispatchers.requestHTTP after changing chart type or filter.



Cylance Dashboard App:

MapStateToProps saves data from store state to a specific container props.

Developer can add business logic before data is saved from state store to container props in mapStateToProps.

DashboardChart for two charts:

# References

<https://reactjs.org/>

<https://redux.js.org/>

<https://facebook.github.io/immutable-js/>

<https://github.com/redux-observable/redux-observable>

<https://github.com/Reactive-Extensions/RxJS>