CZ3003 Final Assignment - Redux

Name: Ernest Ang Cheng Han
Matriculation Number: U1921310H
Date of Submission: 20/10/2021
Lab group: TDDP1

TABLE OF CONTENTS

1. Background of problem - State Management

With the rise of complicated server-side (e.g. Token Caching, Server Side Response handling) and client-side requirements (e.g. Pagination, Spinners, Active Routing, Selected Tabs) when designing our applications, the need for a clear, consistent and predictable convention of managing the state of our entire application arises urgently.

However, this is no easy feat, take for instance an e-commerce platform that you have patronized before, such as Shopee or Lazada: such applications should provide authentication features and most even allow authorization and access from external identity providers (Google, Facebook, OAuth2.0 etc.). On top of managing server-side authentication, proper management of client-side active sessions is necessary especially to prevent security flaws within its design (e.g. Cross Site Request Forgery). When it comes to the actual functionality, a bright and appealing UI design is necessary with multiple ways of sorting through items (e.g. price, number of views). Some organisations even leverage artificial intelligence in order to recommend items to users based on their search patterns and history. After searching for items, users should be allowed to select the size and color if applicable for their desired item before proceeding to confirm and checkout the specific items they have collected in their cart and the list of features seem to go on and on without even considering complicated functions for payment and tracking of orders. The reality is that without a proper and clear definition of the actions that are going on behind the interface of our applications, it is easy for us to lose control of our data and state. Worse still, it would be even harder to debug and reproduce errors occurring in production as well as add new features to our already complex.

Computer Scientists diagnose one of the major complexities that developers fail to handle is the combination of data mutation and data asynchronicity. Data mutation refers to changes to an application’s state while data asynchronicity refers to the unpredictable and non-sequential nature in which data mutation occurs in applications. The non-deterministic nature of data asynchronicity along with the opaqueness of multiple data mutation to a system makes it easy for programs to lose control over the data it handles, the actions it is supposed to perform and the subsequent presentation of data on its interface. This could result in introducing bugs to software systems and if not fixed promptly, it could crash applications, frustrating users and costing firms millions in revenue or even lives in some instances.

2. Introduction to Redux

Redux is an open-sourced state management tool for modern JavaScript applications, leveraged upon by multiple firms ranging from social media giants such as Instagram (Facebook), to financial and investment firms such as Robinhood.

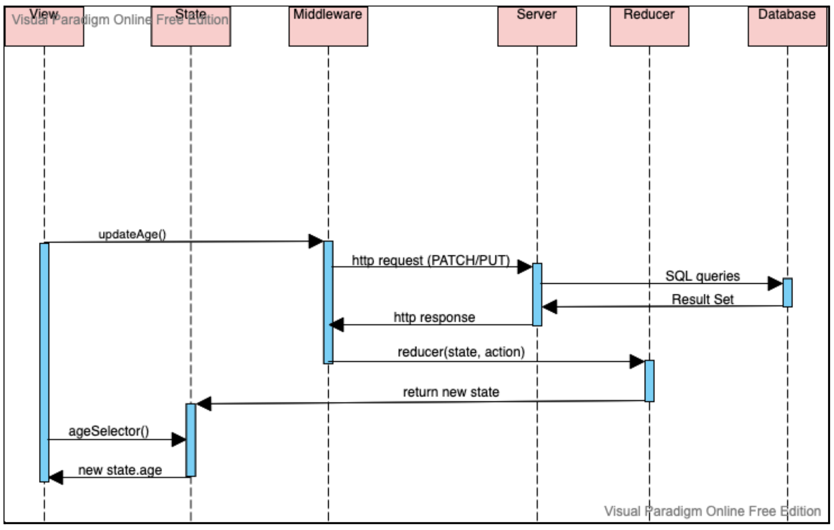
Hence, developers Dan Abramov and Andrew Clarke conceptualised and released Redux in 2015 at the ReactEurope conference. Aimed at making state mutations anticipatable by imposing certain restrictions at when and how updates could happen, Redux containerised an application's data and state while ensuring consistency and predictability in the way these information are handled and updated.

The entire state of an application is defined within a single object known as the store in Redux. Data within this store is read-only and the only way of mutation is via emitting an action by the store itself. Actions are objects which describe the mutation to take place, however, given that actions are just objects, they alone are merely ways of describing the type of mutation to occur in the application's store and have no actual function to mutate the data. Hence, reducers are used to take these actions as inputs and with the current state of the application, perform the respective mutations described by the actions before updating the state if necessary.

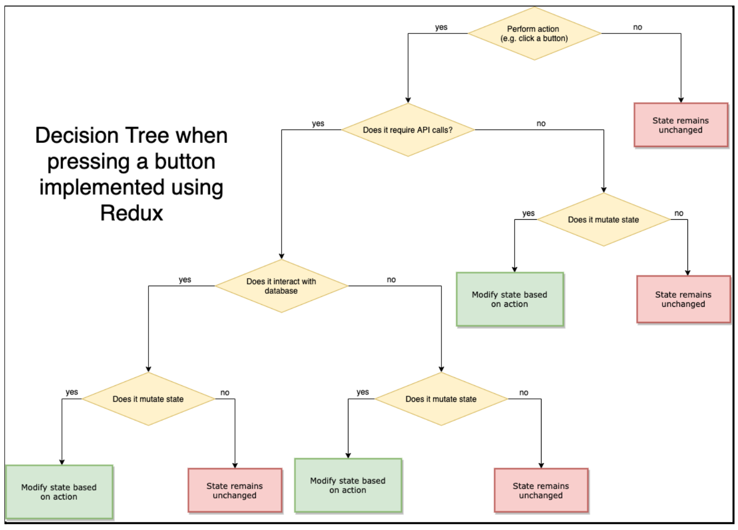
The immutable nature of a centralized application state and the restrictions imposed on data mutation only via actions through reducers thereby outlines a clear fashion and predictable way of data mutation which thereby provides a feasible and easy way to handle this prominent issue of data asynchronicity and data mutation.

Redux is also greatly supported with a wide variety of features, one of which is the Redux DevTools, an extension on many popular browsers such as Mozilla Firefox and Google Chrome which visualizes the application state and of course any changes to it at any point of time. This makes it easy to log and trace the mutations to your application state at any point of time. It also comes with a "Time-Travel Debugging" feature which records the execution of your program which gives developers the flexibility to replay data mutations to the application state backwards and/or forwards. There is also a great variety of redux system add-ons such as middlewares for analytics and websockets as well as functional testing frameworks.

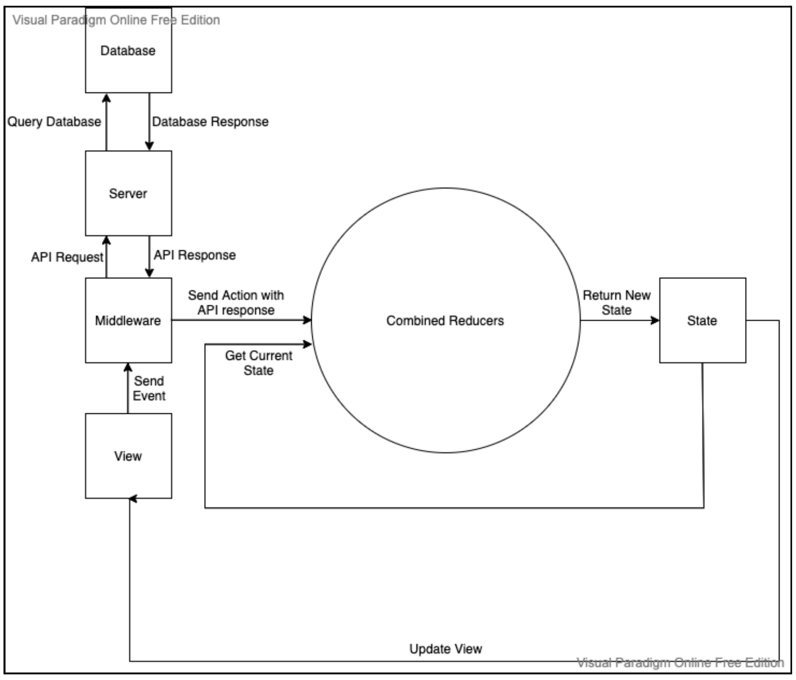
3. Analysis Models



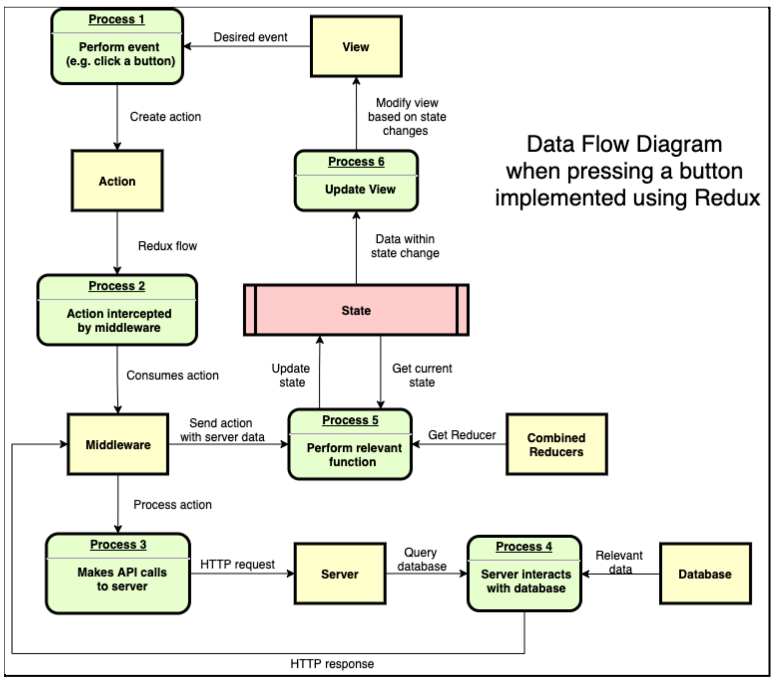
Sequence Diagram for Redux Flow


Decision Tree when pressing a button implemented using Redux
Perform action (e.g. click a button)
yes
no
State remains unchanged
Does it require API calls?
yes
no
Does it mutate state
yes
no
Modify state based on action
State remains unchanged
Does it interact with database
yes
no
Does it mutate state
yes
no
Modify state based on action
State remains unchanged
Does it mutate state
yes
no
Modify state based on action
State remains unchanged

Decision Tree Diagram for Redux Flow

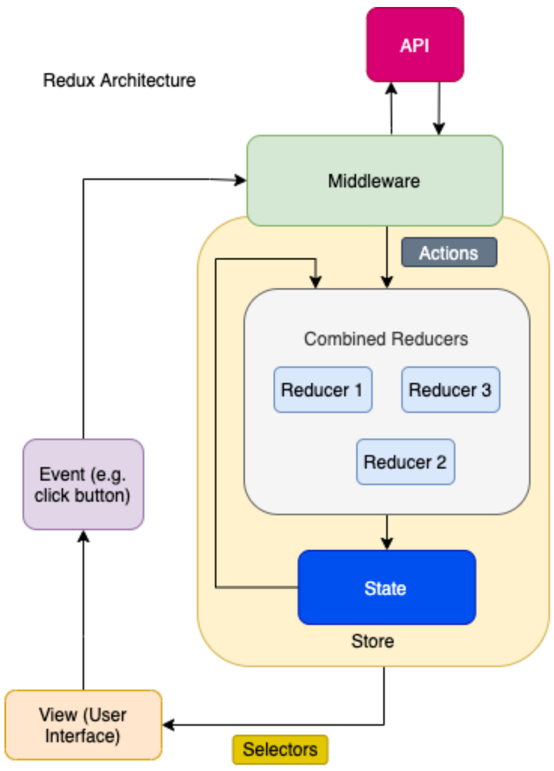


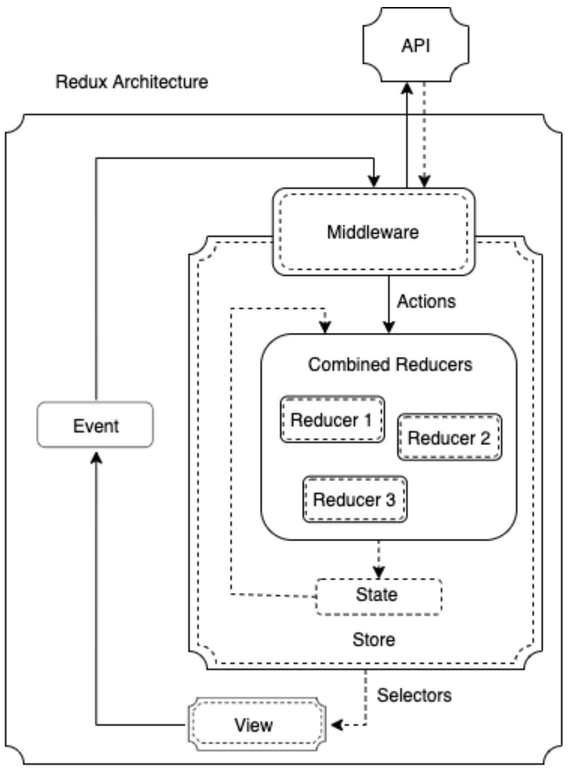
Context Diagram for Redux Flow



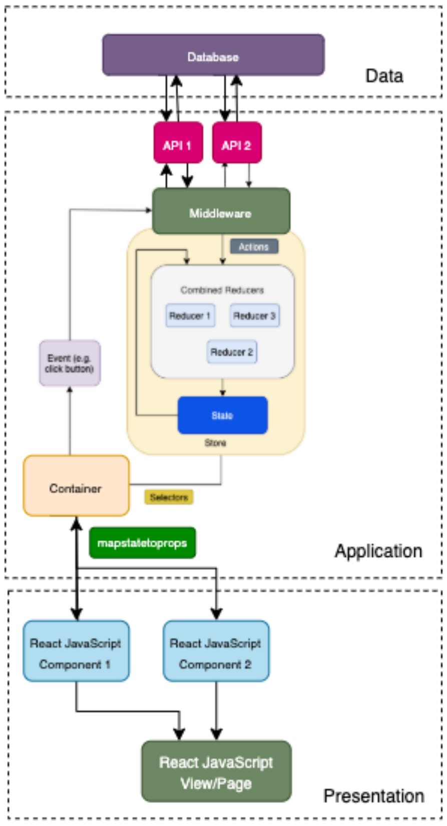
Data Flow Diagram for Redux Flow

4. Software Architecture

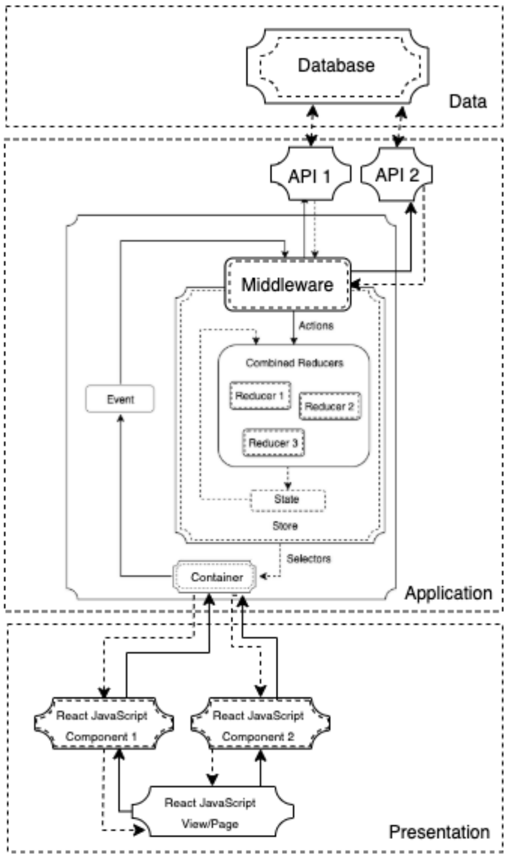




Redux with React JavaScript in 3 layer architecture



Redux with React JavaScript in 3 layer architecture



5. Software Quality Attributes

Integrity: Immutable Read-Only nature of the store ensures that the data and state of the application encapsulated within the store ensures that mutations that occur can only be done via actions and reducers, thereby ensuring data integrity and unauthorised changes to the store from other parts of the application.

Reliability: Redux only allows for one central store to maintain the entire state of the application, meaning that there is a single source of truth and the data being read from the store across the entire application is consistent and reliable.

Interoperability & Portability: Redux is compatible and interoperable with many modern frontend frameworks used today such as Angular, Vue, Ember, Svelte and React. Redux is also the inspiration of many state management derivatives optimized to the framework used, such as NgRx for Angular which leverages the technologies of Redux and RxJs, as well as Vuex for Vue.

Reusability: Redux is compatible with many softwares built on the foundation of Component Driven Development (CDD) frameworks which result in reusable modular components such as React-Redux, making these functional components integrated with Redux and hence Redux in itself - reusable, accelerating the development process without having to rewrite elements and components of a program while decoupling and simplifying code bases.

Testability: Redux supports and can be tested with any test runner such as Jest and Enzyme. The “@redux/toolkit” library also comes with a variety of methods and functions which supports the testing of reducers and mocking of actions dispatched asynchronously such as “createAction” and “createAsyncThunk”.

Maintainability, Usability & Installability: Redux is a widely used open-source software leveraged upon by organisations by the likes of Amazon and Facebook (Instagram). With extremely detailed documentation, plethora of tutorials and courses curated by industry professionals, along with a great and huge community of developers and contributors maintaining the software, Redux is a rather simple to use software with many theoretical, practical and technical guides and documentation available. Additionally, Redux is lightweight (~2KB) and easy to install using Node Package Manager (NPM) which is compatible on most desktop operating systems (e.g. Windows, MacOSX, Linux).

Flexibility, Correctness & Robustness: The nature and architecture of Redux makes it easy for developers to scale and add functionalities via actions and reducers while ensuring its correctness as well as providing tracing and bugging capabilities in the unlikely scenario of unanticipated data mutations, unlikely given the nature of Redux and how it is able to resolve data mutation and data asynchronicity thereby providing a clear way of predicting changes to the application state. The structure of outlining data mutations via actions and handling the business logic via reducers also ensures the correct functionality of each reducer when consuming a particular action and subsequently the correct updating of the application state and the relevant views.

Efficiency: In terms of performance and consumption of system resources, Redux requires resources to process actions in any used middleware and reducers, as well as notifying subscribers (if any) after the actions have been dispatched by the store, before updating the view and any components based on any mutations made to the data of the state. According to its documentation and studies made by multiple developers and computer scientists, there seems to be nothing inherently slow or inefficient about the design of Redux and how it is subsequently used in applications and deployed into production. Additionally, derivative libraries are built to heavily optimize the respective framework's performance. For instance, React-Redux cuts down on unnecessary re-renders and updates to components in the view.

Overall, Redux seems to be a product and system which hits most of the requirements and concerns illustrated by Software Quality Attributes. Based on the McCall Factor Model, Redux seems to be highly revisable, operable and transitable.

6. Discussion and Reflection

In order to evaluate Redux as a technical solution to modern day software engineering woes and problems, we need to evaluate its alternatives. Redux is actually just a slight modification from a design pattern, Flux, which was architected by developers at Facebook to overcome the supposed issues faced by the very conventional Model-View-Controller (MVC) design pattern commonly used in developing clients and servers. Later on, we see alternative state management tools such as MobX, each with their own respective benefits and detriments for evaluation.

6.1 Model-View-Controller

Proposed in 1976 by Computer Scientist Trygve Reenskaug, the Model-View-Controller (MVC) design pattern was and is still used today to design and develop Single Page and Multi Page applications while meeting complicated software requirements. By abstracting and separating a program into 3 separate layers, Model which handles data structures, View which handles the User Interface, and Controller which takes user input, manipulates models and updates the view, it simplifies the process of development while meeting the above mentioned complicated needs of softwares. For instance, a developer could work on UI and other client side features without having to worry about the business logic that goes behind manipulating the information and data that is presented on the interface.

Construed in 1976, MVC has since then evolved into improved variants such as HMVC, MVA, MVVM, and MVP. This architecture is also the basis and foundation of many softwares used today: Rails (Ruby), Laravel (PHP), and Django (Python).

6.2 Flux

In 2014, developers and engineers from Facebook proposed Flux at their annual F8 conference. Engineering Manager Tom Occhino opined that systems designed using MVC become complicated very quickly, lowering scalability. This issue worsens exponentially on the addition of new features, making the code “fragile and unpredictable.” This was becoming a serious problem for developers new to a certain codebase as they were afraid to touch the code lest they might break something, slowing down development and onboarding new engineers to a project. Software Engineer Jing Chen further added that MVC is good for small applications, but the complexity explodes when many models and their corresponding views are added to a system, making it difficult to understand and debug especially given the possible bidirectional data flow between models and views.

Hence, a team of engineers proposed: Flux, a system architecture that promotes single directional data flow through an application. Similar to Redux, Flux contains: Stores which contain all the application’s data (one store per type of application data), a Dispatcher per store which replaces MVC’s Controller*,* deciding how a Store is to be updated when an Action is triggered. The View is also updated when the Store changes. This ensures a unidirectional flow of data between a system’s components. Even as there can be multiple Stores or Views, the system can be seen as having only one Store and one View since the data is flowing only one way and the different Stores and Views do not directly affect each other.

6.3 MobX

Slightly after a year Flux proposed, Facebook developer Michel Westrate and his team released the first version of MobX. At its core, Mobx consists of: Data Stores with the state of the application, Observables to track mutations to application state via implicit subscriptions, Derivations which are values computed from the Observable State, Reactions which are actions that can be performed from the current Observable State (usually I/O related), mainly ensuring that the DOM is updated and any necessary network requests are automatically sent out and handled. Finally, there are actions which alter the state of the application.

6.4 Evaluation

	MVC	Flux
Components	Model, View, Controller	Action, Dispatcher, View, Store
Data Flow	Bidirectional	Unidirectional
Data/State	Multiple Models	Multiple Stores
Structure	No fixed data structure	No fixed data structure
Dispatcher	None	Single dispatcher to which all actions pass through
Debugging	Difficult with multiple controllers handling multiple models while updating multiple views, in addition to the bidirectional flow of data	Single dispatcher makes it easy for debugging since exact actions which are dispatched is transparent
Business Logic	Controller	Store
Mutability	Not applicable	Immutable Store

	Redux	MobX
Components	Store, View, Action, Reducer, Middleware (Add-ons)	Store, State, Observables, Derivations, Reactions, Actions
Data Flow	Unidirectional	Unidirectional
State	Single Central Store	Multiple States
Data Structure	Single Object	No fixed data structure
Dispatcher	None	None
Debugging	Single store makes debugging easy since mutations to the central store are very transparent (Single Source of truth). Multiple add-ons (Redux DevTools) and extensions available which make debugging easy.	Little support available for MobX makes it debugging difficult
Business Logic	Reducer	Store
Mutability	Immutable Store	Mutable Store

6.5 Overall

Each state management tool comes with its pros and cons when being used to design modern-day applications with heavy client-side and server-side requirements. The conventional and still relevant MVC architecture is the foundation of many applications and softwares, utilized across technology giants such as Microsoft. Yet, the rise of other design patterns and frameworks to handle state management tools gives us the flexibility yet dilemma in selecting the best to design our applications. I believe that amongst the above-mentioned technologies, Redux is the best and the most desirable tool which can be used to solve the development woes of software engineering and the conflict of data mutation and asynchronicity. The unidirectional flow of data along with an immutable store acting as a single source of truth for the entire state of the application, added on with the huge plethora of community support and software extensions, makes it the perfect tool for development without having to struggle with state management issues and slow development processes which engineers, team leads and managers would otherwise have to deal with as their products scales with multiple features being added in every update or release.