Data Flow Design - >
Use a client serve model where the frontend communicates with backend for data operations
Apply pagination and lazy loading to handle large datasets efficiently

FetchData Retrieve only 100 records per request to minimize memory usage

ProcessData Perform sorting , searching and aggregation operations on the server side to reduce load on frontend side

Render Data Display the Data in the UI using a flatlist which is using pagination for smooth user experience

Component Architecture

Structure:

Container Components : Manage state and logic for example DataContainer

Presentational Components : Render UI and recieve Data through props

On high level we can assume achitecture like this

App-> DataContainer
SearchBar
SortControls
DataTable
Pagination Controls

State Management With Observer Pattern

We will use a state management library for example Redux to manage the applications state

and also will implement the Observer patter to update the UI component when the state changes

What is observer pattern ?

It is a design pattern use to establish a one to many dependancy between objects so
that when one object changes its state all its dependents are notified and updated automatically by consuming the change that occured
Redux uses this only if the state in store is changed the components that subscribe to that state re-renders and updates their data

Redux uses actions,reducers,epics(middleware),model and service based files
to update the store efficently

Performance Optimization

Techniques :

We Can use Fast Image to load images in a increasing of quality manner according to the connection bandwidth

Debouncing : We can debounce the search input to avoid frequent network calls

Memoization : Use memoization techniques to avoid redudant calculations and renders
For - using react memo and react.memo(making pure components)

Interaction Design

Usability

Provide a responsive and intuitive UI with clear controls for sorting , searching and pagination

We can even ensure accessibility by prop of accessiblitiy =true in our components
so that even visually impaired people can interact with our app smoothly

Components:
- Search Bar:Input for filtering data
- SortControls:Buttons or dropdowns
- DataTable:Table to display the data with virtual scrolling
- Pagination:To fetch limited data in a go and only fetch and append to our state when users scrolls the list

Scaling :
Ensuring the backend can handle high volumes of data and concurrent requests.
Use a microservice architecture for the backend to scale individual services as needed

Implement caching mechanisms to reduce database load as such redis

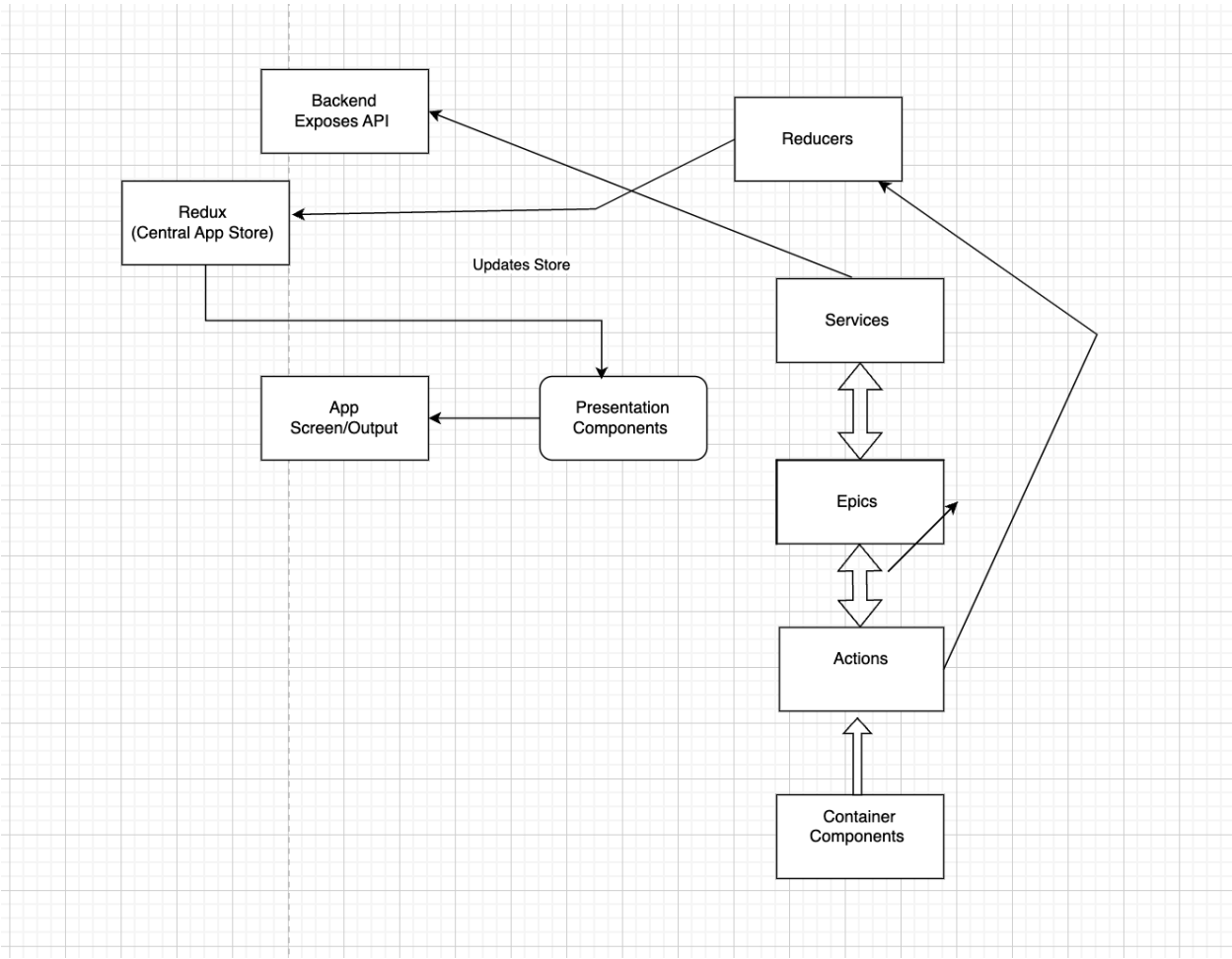
FrontEnd:
Modularize the code to allow adding new features without significant changes to existing code

Use Code splitting and lazy loading to reduce the initial load time of the application

Clarity

Code Clarity ->
We will write a clean module and well documented code to ensure maintainability
Also we will write tdd of the feature before starting to code

Follow proper naming conventions



So Here i will explain the flow of this diagram

Container component dispatches actions like fetch data this action is caught by the middleware which is epics , here im using rxjs as a middleware
This Epic will have the code to call the service , service will use a 3rd party like axios to make an api call to the backend server , the response we get from the call is taken and from epic itself We call another action of success or error based on the response you got
This action will call the reducer and update the store data, now all the presentation components which were using this data will re-render and updates their UI
And hence the screen gets populated by the latest data