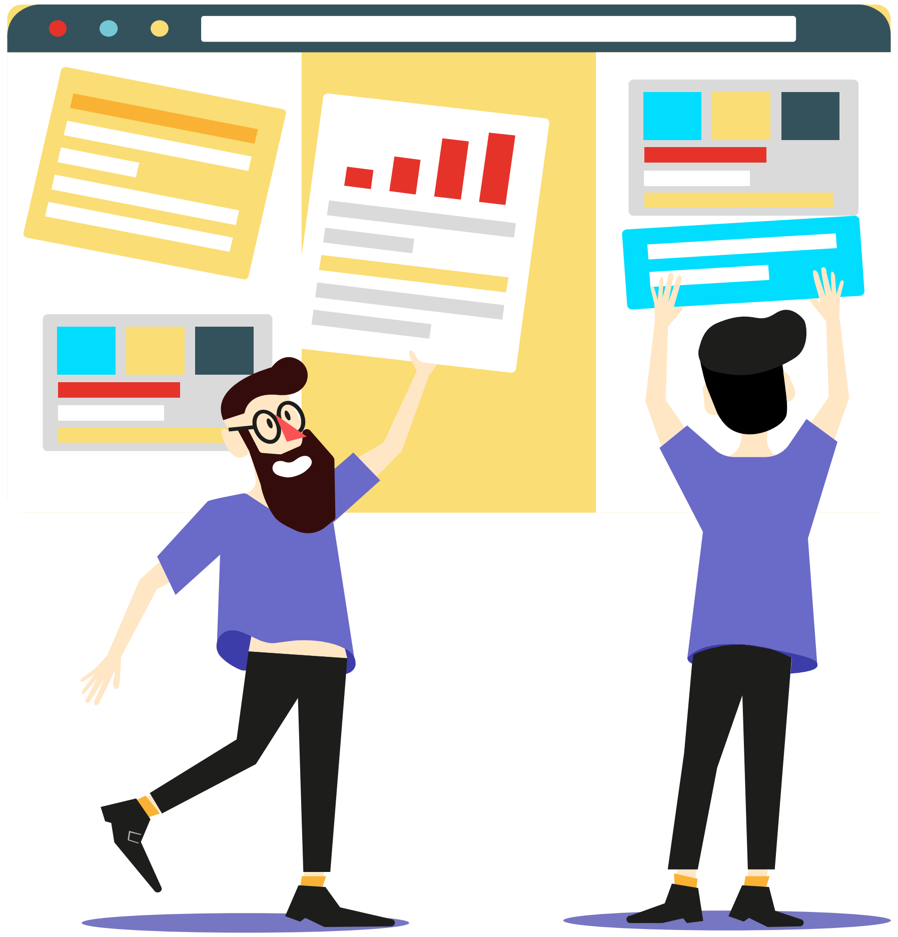
Web Framework Development Tutorial

React Version

Danqing Shi

CONTENTS

1. Introduction to Web Framework
2. Learn the Basics
3. How to Develop An Application with Framework
4. State Management
5. Third-Party Libraries
6. Homework



/01

Introduction to Web Framework

What is front-end web framework?

Popular front-end web frameworks.

Why we need these frameworks?

What is Web Framework?

Web Framework: A web framework (WF) or web application framework (WAF) is a software framework that is designed to support the development of web applications including web services, web resources, and web APIs. Web frameworks provide a standard way to build and deploy web applications on the World Wide Web. Web frameworks aim to automate the overhead associated with common activities performed in web development.

Single-Page Application (**SPA**): A single-page application (SPA) is a web application or web site that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server. This approach avoids interruption of the user experience between successive pages, making the application behave more like a desktop application.

Web browser JavaScript Frameworks: Web browser JavaScript frameworks, such as AngularJS, React and Vue.js have adopted SPA principles. They can be used as a base in the development of web applications.

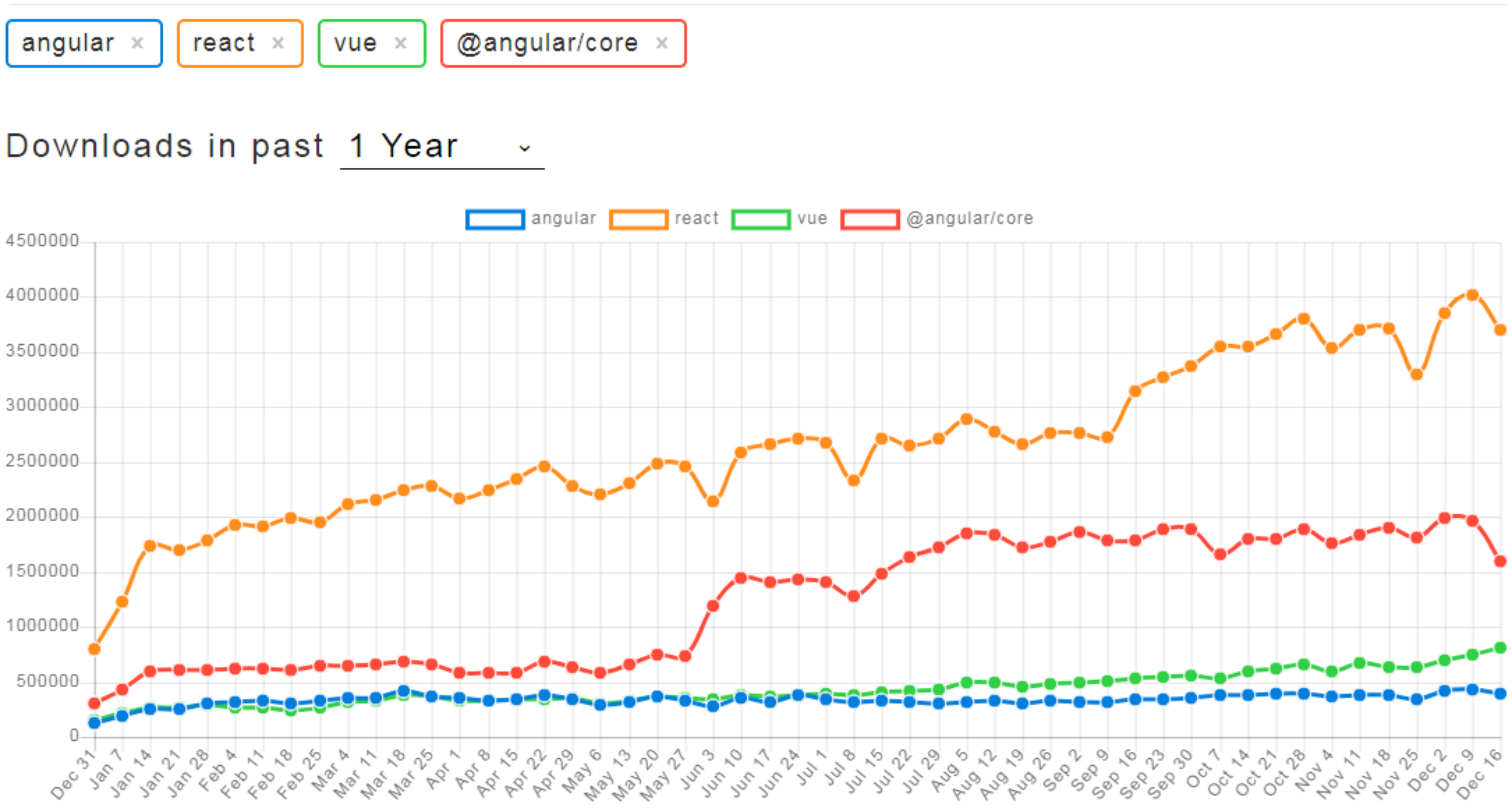
https://en.wikipedia.org/wiki/Web_framework; https://en.wikipedia.org/wiki/Single-page_application ;

Popular Front-end Web Framework

React
Facebook
2013-3
MVC

Vue
Evan You
2014-2
MVVM

Angular
Google
2010-10
MVW



npm trend

https://github.com/HankBass/front-end-frameworks-comparison

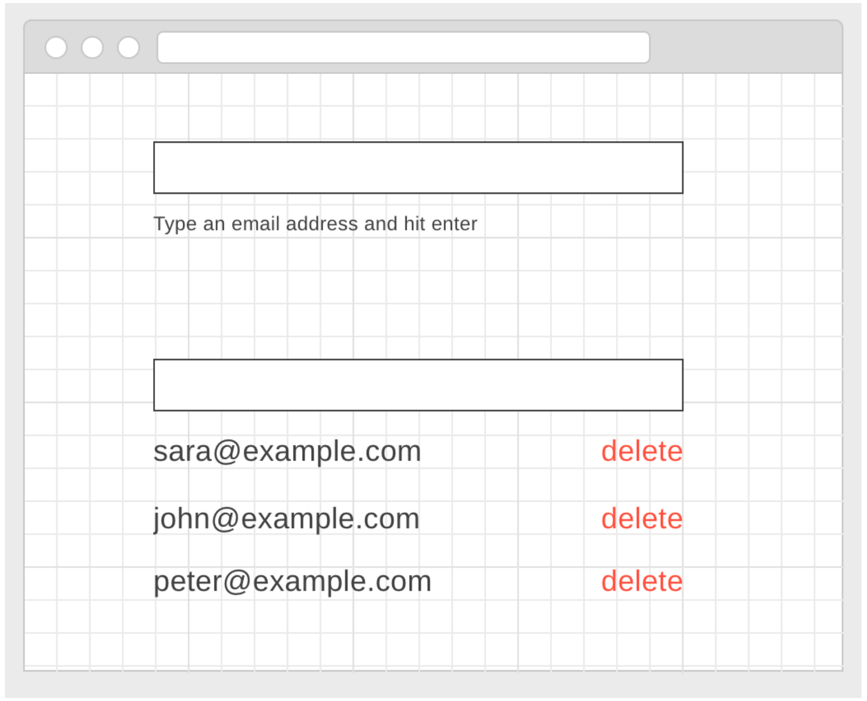
Why We Need Web Frameworks?

Advantages

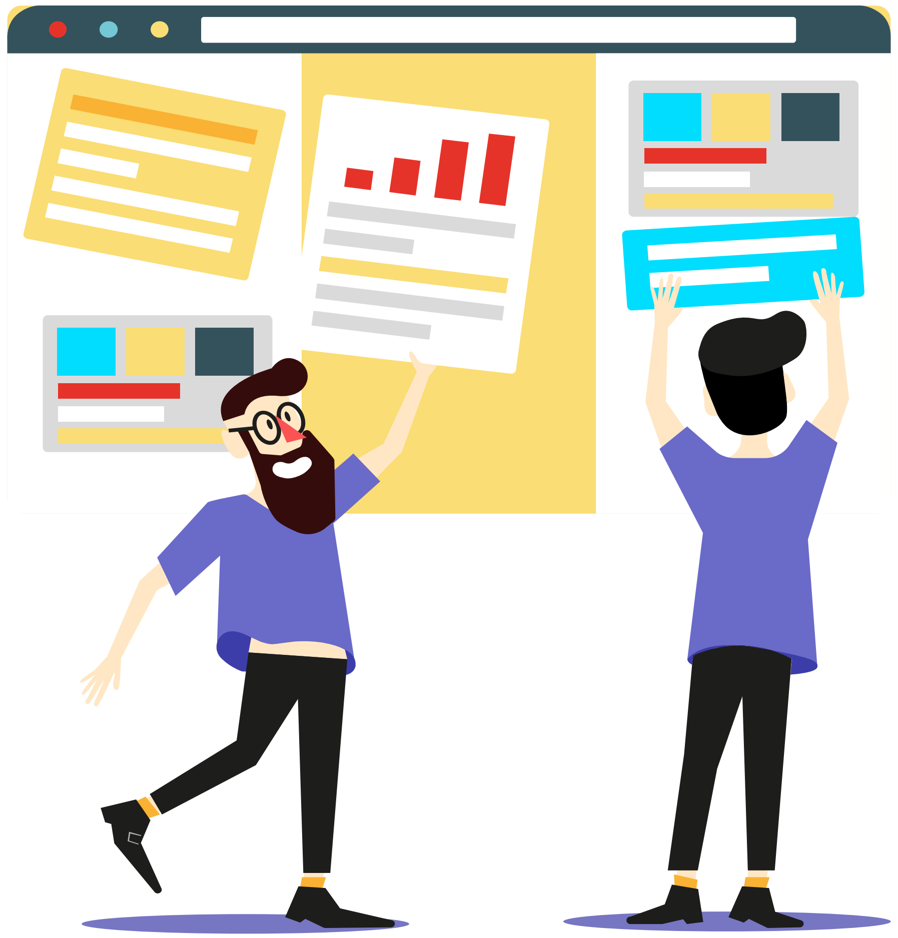
- They are based on components;
- They have a strong community;
- They have plenty of third party libraries to deal with things;
- They have useful third party components;
- They have browser extensions that help debugging things;
- They are good for doing single page applications;

Why We Need Web Frameworks?

The deepest reason why modern JavaScript frameworks exist ?
The biggest problem is always **updating the UI on every state change**.



- The main problem modern JavaScript frameworks solve is keeping the UI in sync with the state.
- It is not possible to write complex, efficient and easy to maintain UIs with Vanilla JavaScript.



/02

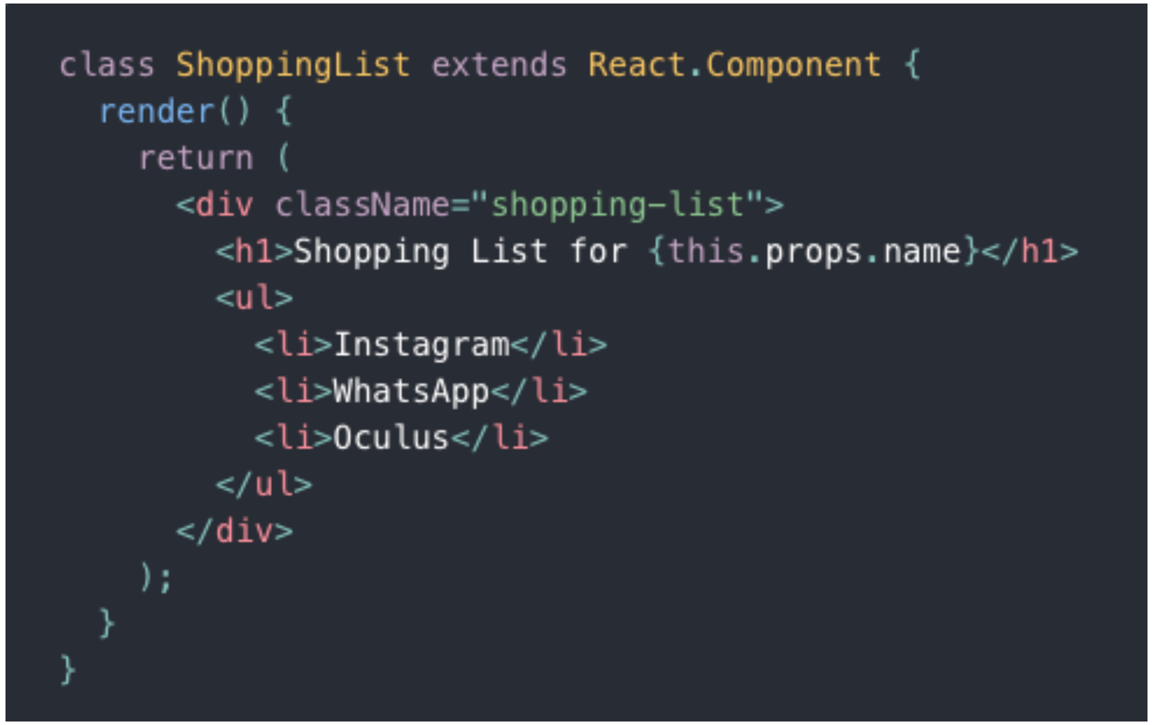
Learn the Basics

What is React?

Component LifeCycle

What is React?

- React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called “components”.
- We’ll start with **React.Component** subclasses:
- We’ll get to the funny XML-like tags soon. We use components to tell React what we want to see on the screen. When our data changes, React will efficiently update and re-render our components.
- Here, ShoppingList is a **React component class**, or **React component type**. A component takes in parameters, called props (short for “properties”), and returns a hierarchy of views to display via the render method.

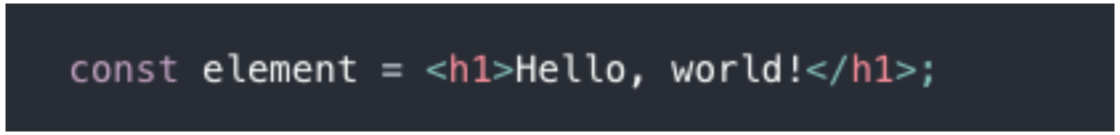


```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}
```

demo-1
// Example usage: <ShoppingList name="Mark" />

Introducing JSX

- It is called JSX, and it is a syntax extension to JavaScript. We recommend using it with React to describe what the UI should look like. JSX may remind you of a template language, but it comes with the full power of JavaScript.

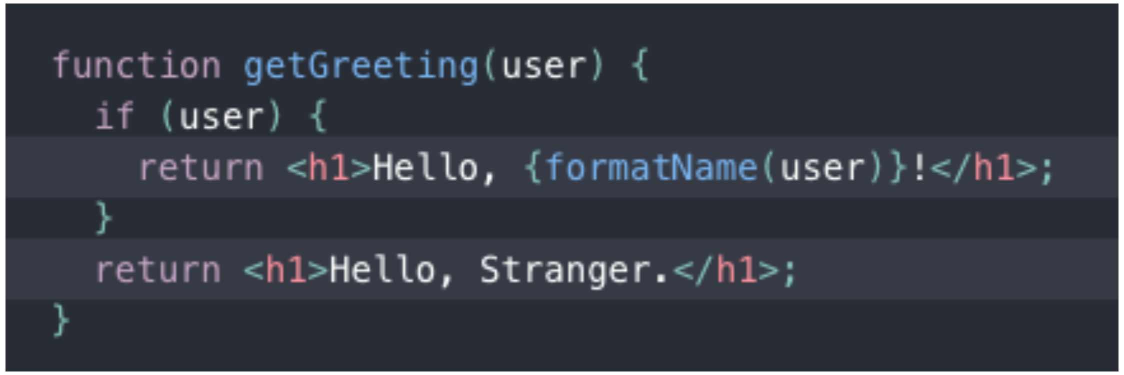


```
const element = <h1>Hello, world!</h1>;
```

This funny tag syntax is neither a string nor HTML.

Why JSX?

- React embraces the fact that rendering logic is inherently coupled with other UI logic: how events are handled, how the state changes over time, and how the data is prepared for display.
- React doesn't require using JSX, but most people find it helpful as a visual aid when working with UI inside the JavaScript code. It also allows React to show more useful error and warning messages.



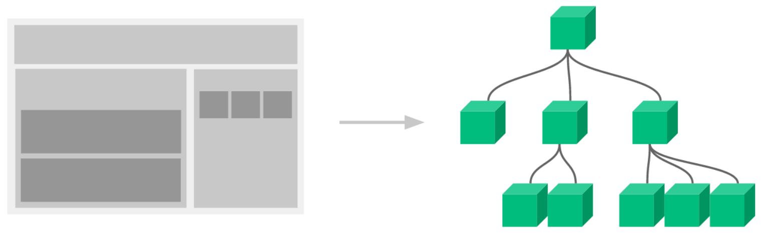
```
function getGreeting(user) {
  if (user) {
    return <h1>Hello, {formatName(user)}!</h1>;
  }
  return <h1>Hello, Stranger.</h1>;
}
```

demo-2
JSX and Conditional Render

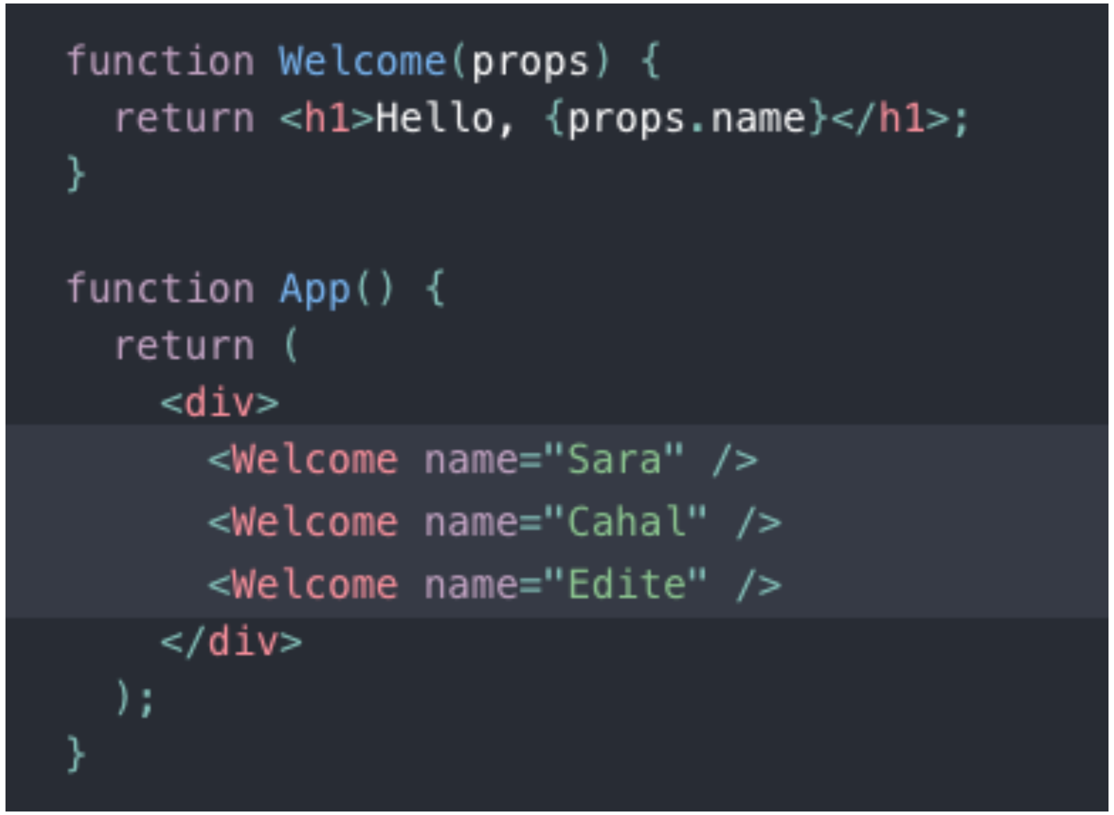
https://reactjs.org/docs/introducing-jsx.html

Composing Components

- Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail. A button, a form, a dialog, a screen: in React apps, all those are commonly expressed as components.
- Typically, new React apps have a single App component at the very top. However, if you integrate React into an existing app, you might start bottom-up with a small component like Button and gradually work your way to the top of the view hierarchy.



For example, we can create an App component that renders Welcome many times:.



```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}
```

demo-3

https://reactjs.org/docs/components-and-props.html

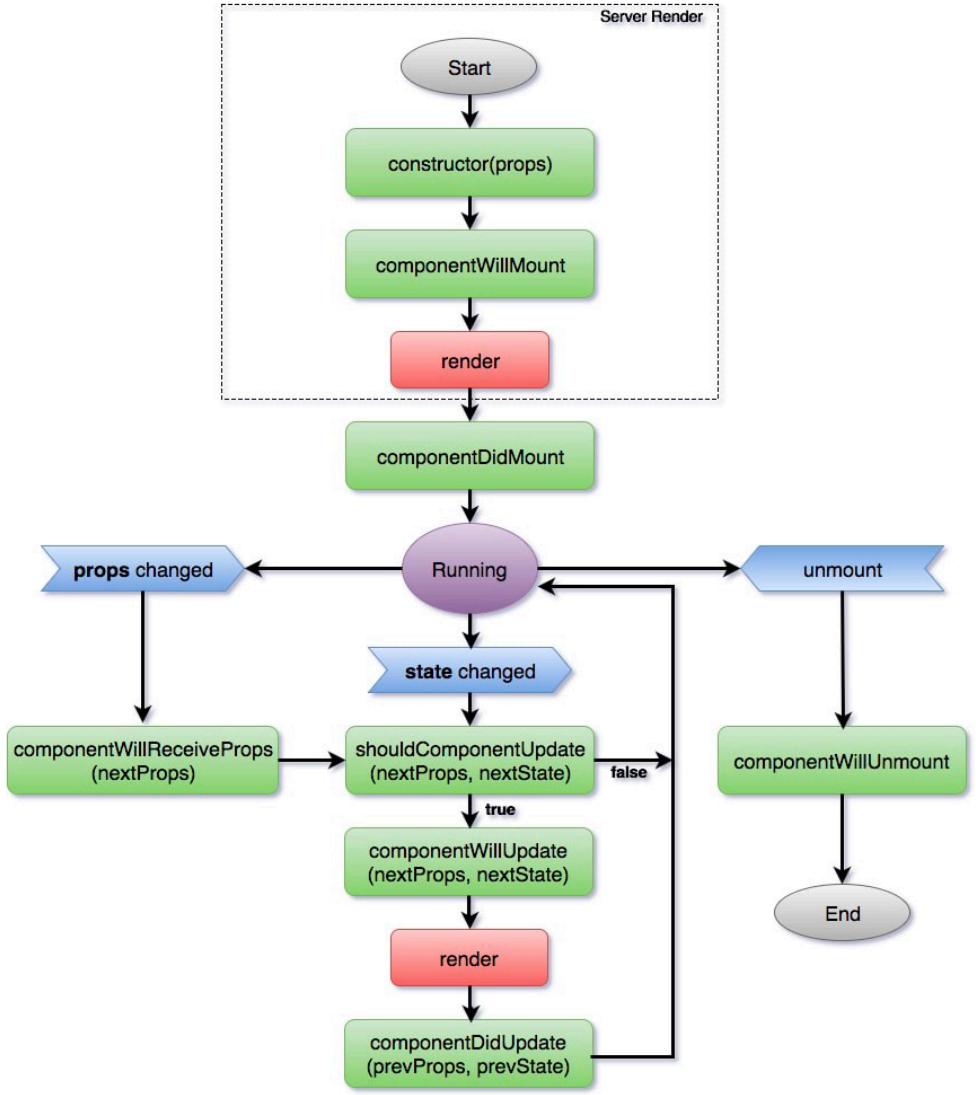
Component Lifecycle

Mounting
- constructor()
- componentWillMount()
- render()
- componentDidMount()

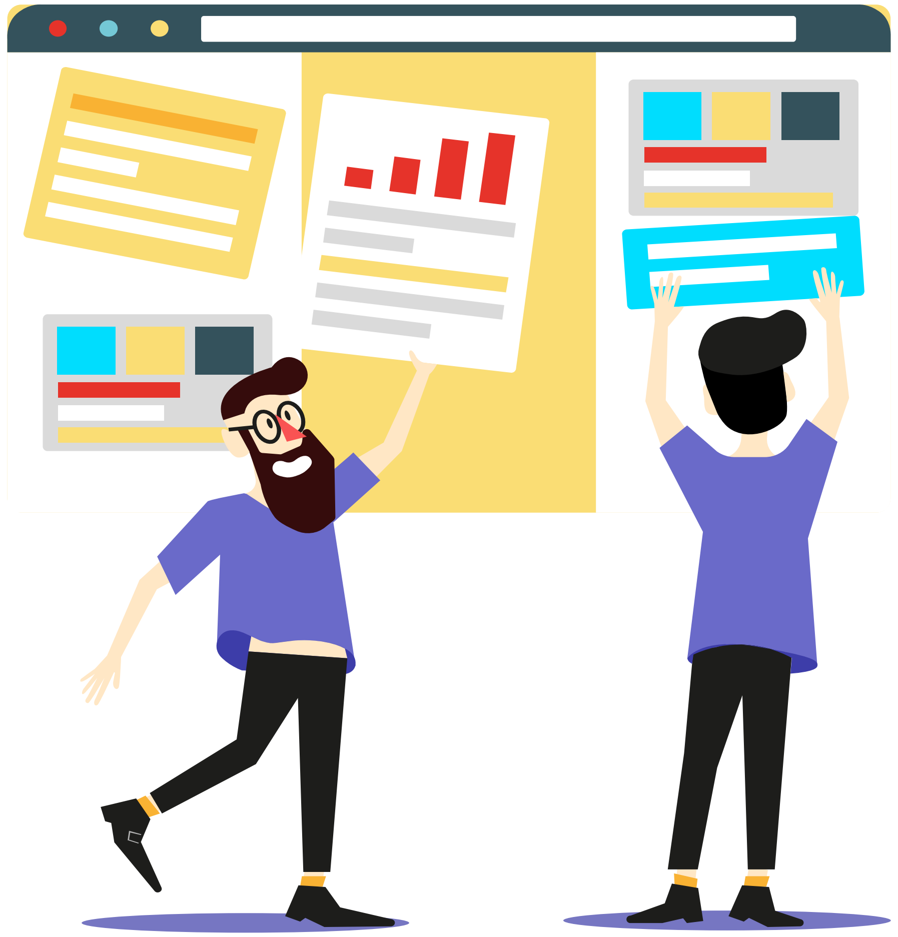
Updating
- componentWillReceiveProps()
- shouldComponentUpdate()
- componentWillUpdate()
- render()
- componentDidUpdate()

Unmounting
- componentWillUnmount()

demo-4


Server Render
Start
constructor(props)
componentWillMount
render
componentDidMount
props changed
Running
unmount
state changed
componentWillReceiveProps (nextProps)
shouldComponentUpdate (nextProps, nextState)
false
true
componentWillUnmount
componentWillUpdate (nextProps, nextState)
End
render
componentDidUpdate (prevProps, prevState)

https://levelup.gitconnected.com/componentdidmakesense-react-lifecycle-explanation-393dcb19e459



/03

Develop A Web Application

What is front-end web framework?

Thinking in React

How to develop a web application with framework?

Thinking in React

"**React is, in our opinion, the premier way to build big, fast Web apps with JavaScript. It has scaled very well for us at Facebook and Instagram.**" **– Facebook**

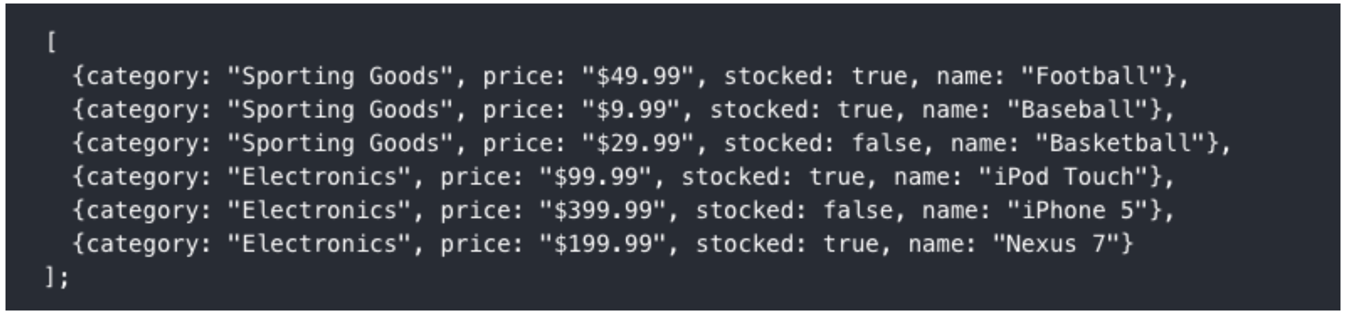
One of the many great parts of React is how it makes you think about apps as you build them.

In this part, I'll walk you through the thought process of building a searchable product data table using React.

https://reactjs.org/docs/thinking-in-react.html

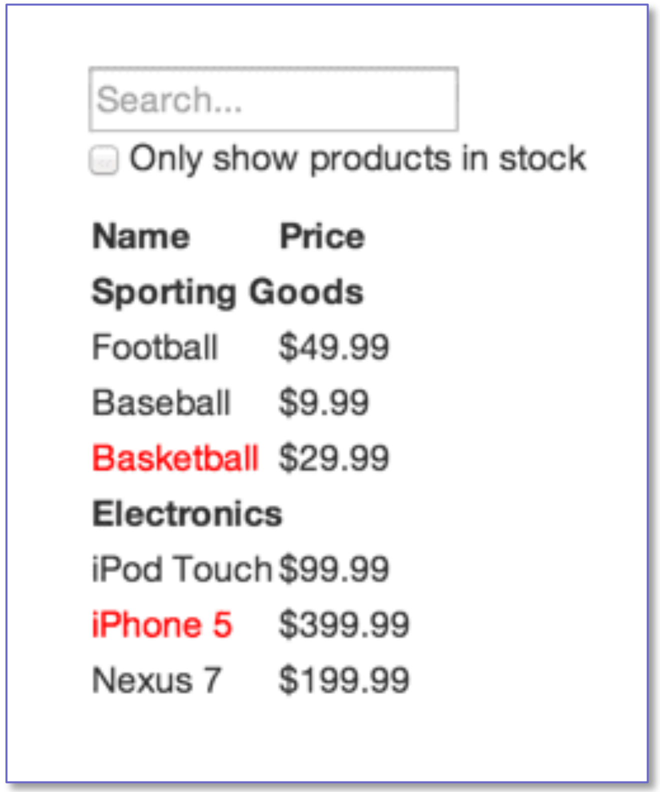
1. Prototyping

Imagine that we already have a JSON data, and we can prototype the searchable product data table.



```
[
  {category: "Sporting Goods", price: "$49.99", stocked: true, name: "Football"},
  {category: "Sporting Goods", price: "$9.99", stocked: true, name: "Baseball"},
  {category: "Sporting Goods", price: "$29.99", stocked: false, name: "Basketball"},
  {category: "Electronics", price: "$99.99", stocked: true, name: "iPod Touch"},
  {category: "Electronics", price: "$399.99", stocked: false, name: "iPhone 5"},
  {category: "Electronics", price: "$199.99", stocked: true, name: "Nexus 7"}
];
```

JSON data



Search...

☐ Only show products in stock

Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99

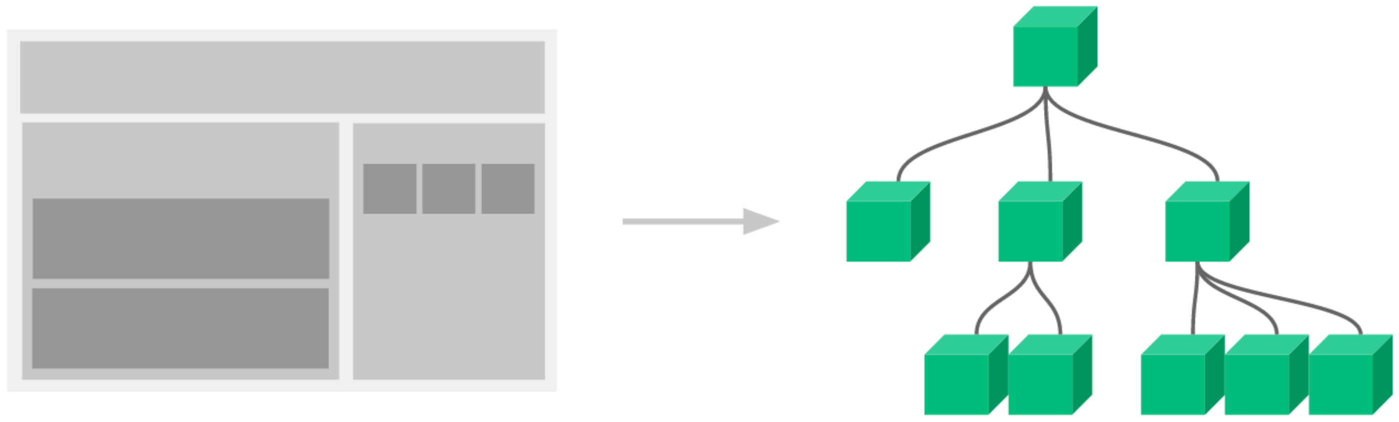
Designed Prototype

https://reactjs.org/docs/thinking-in-react.html

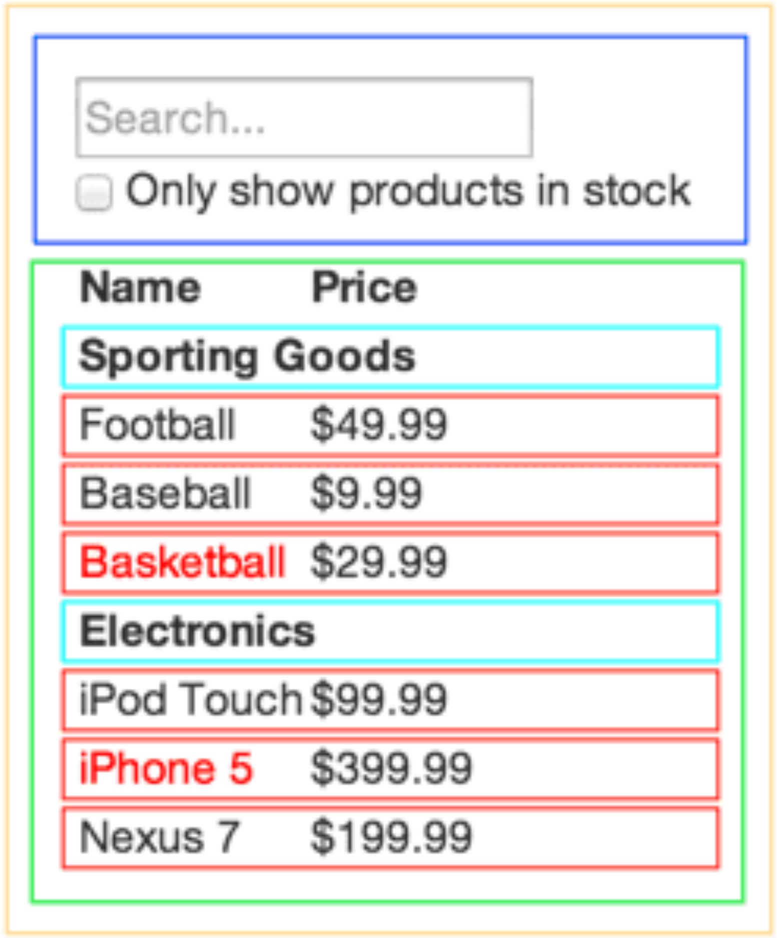
2. Composing with Components

Break The UI Into A Component Hierarchy

The first thing you'll want to do is to draw boxes around every component (and subcomponent) in the mock and give them all names. If you're working with a designer, they may have already done this, so go talk to them! Their Photoshop layer names may end up being the names of your React components!



2. Composing with Components


Search...

Only show products in stock

Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99

You'll see here that we have five components in our app. We've italicized the data each component represents.

1. **FilterableProductTable (orange):** contains the entirety of the example
2. **SearchBar (blue):** receives all *user input*
3. **ProductTable (green):** displays and filters the *data collection* based on *user input*
4. **ProductCategoryRow (turquoise):** displays a heading for each *category*
5. **ProductRow (red):** displays a row for each *product*

2. Composing with Components

Now that we've identified the components in our mock, let's arrange them into a hierarchy.

Components that appear within another component in the mock should appear as a child in the **hierarchy**.

- `FilterableProductTable`
    - `SearchBar`
    - `ProductTable`
        - `ProductCategoryRow`
        - `ProductRow`

3. Build Static Components

- Now that you have your component hierarchy, it's time to implement your app. The easiest way is to build a version that takes your data model and renders the UI but has no interactivity. It's best to decouple these processes because building a static version requires a lot of typing and no thinking, and adding interactivity requires a lot of thinking and not a lot of typing.
- To build a static version of your app that renders your data model, you'll want to build components that reuse other components and pass data using *props*. *props* are a way of passing data from parent to child. If you're familiar with the concept of *state*, **don't use state at all** to build this static version. State is reserved only for interactivity, that is, data that changes over time. Since this is a static version of the app, you don't need it.

don't use state at all

At the end of this step, you'll have a library of reusable components that render your data model.

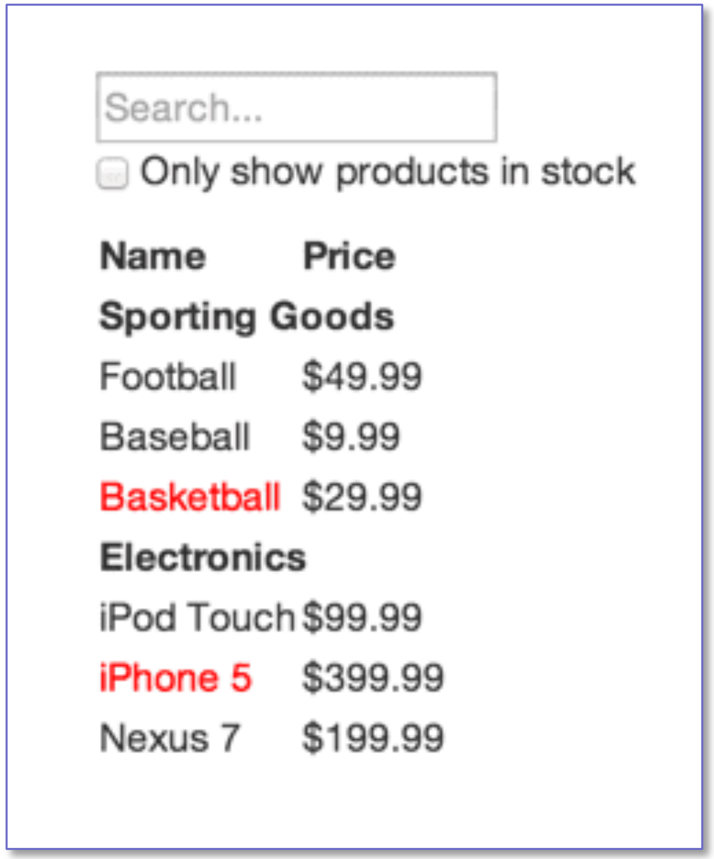
React's **one-way data flow** (also called *one-way binding*) keeps everything modular and fast.

4. Add States

To make your UI interactive, you need to be able to trigger changes to your underlying data model. React achieves this with **state**.

To build your app correctly, you first need to think of the minimal set of mutable state that your app needs. The key here is DRY: Don't Repeat Yourself. Figure out the absolute minimal representation of the state your application needs and compute everything else you need on-demand.

For example, if you're building a TODO list, keep an array of the TODO items around; don't keep a separate state variable for the count. Instead, when you want to render the TODO count, take the length of the TODO items array.



Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99

4. Add States

Step 1: Think of all of the pieces of data in our example application. We have:

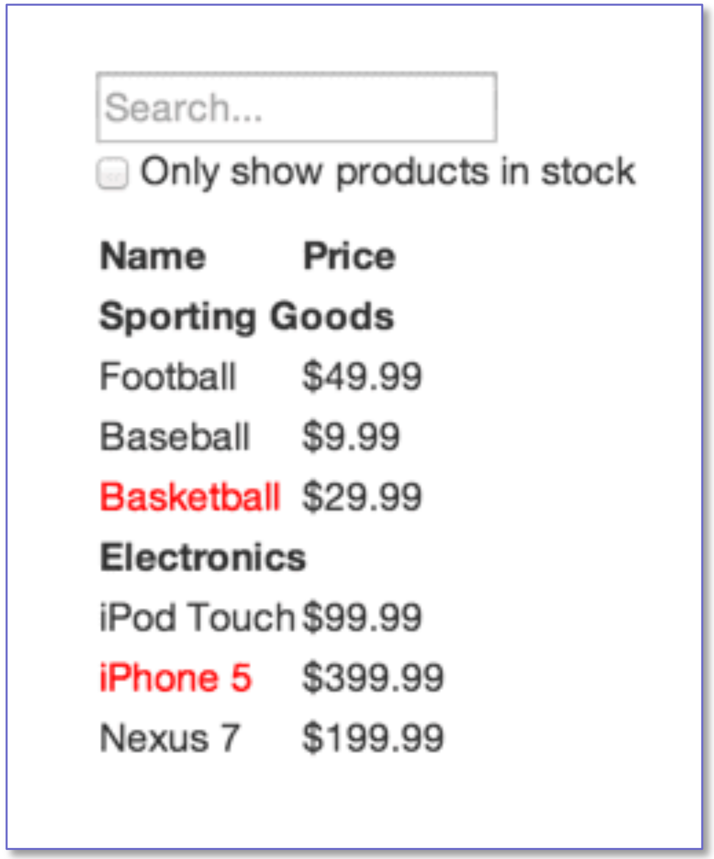
- The original list of products
- The search text the user has entered
- The value of the checkbox
- The filtered list of products

Step 2: Let's go through each one and figure out which one is state. Ask three questions about each piece of data:

- Is it passed in from a parent via props? If so, it probably isn't state.
- Does it remain unchanged over time? If so, it probably isn't state.
- Can you compute it based on any other state or props in your component? If so, it isn't state.

So finally, our state is:

- The search text the user has entered
- The value of the checkbox



Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99

https://reactjs.org/docs/thinking-in-react.html

5. Data Flow

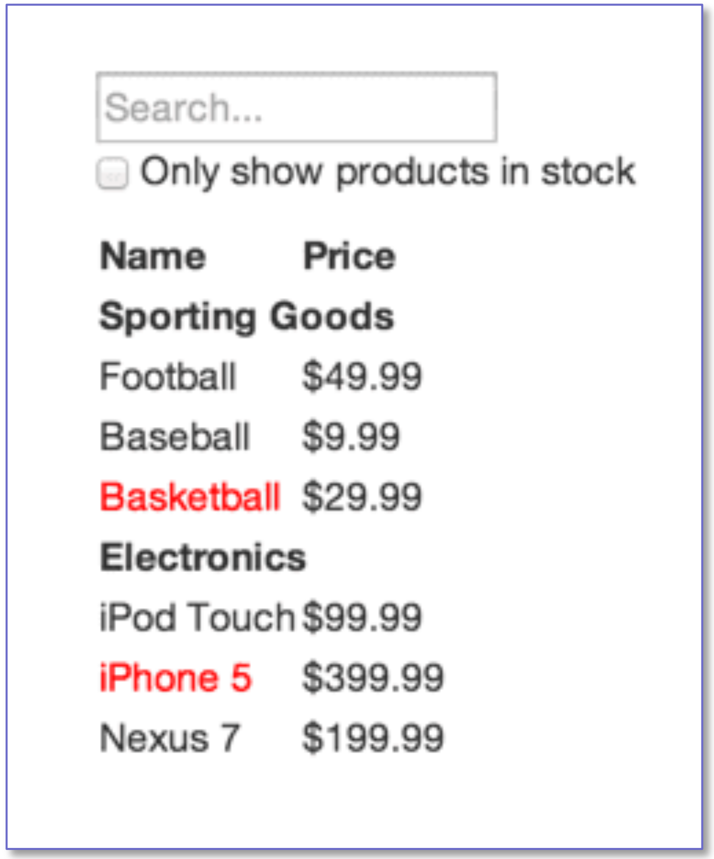
Let’s think about what we want to happen:

We want to make sure that whenever the user changes the form, we update the state to reflect the user input.

Since components should only update their own state, **FilterableProductTable** will pass callbacks to **SearchBar** that will fire whenever the state should be updated.

We can use the **onChange** event on the inputs to be notified of it.

The callbacks passed by **FilterableProductTable** will call **setState**(), and the app will be updated.

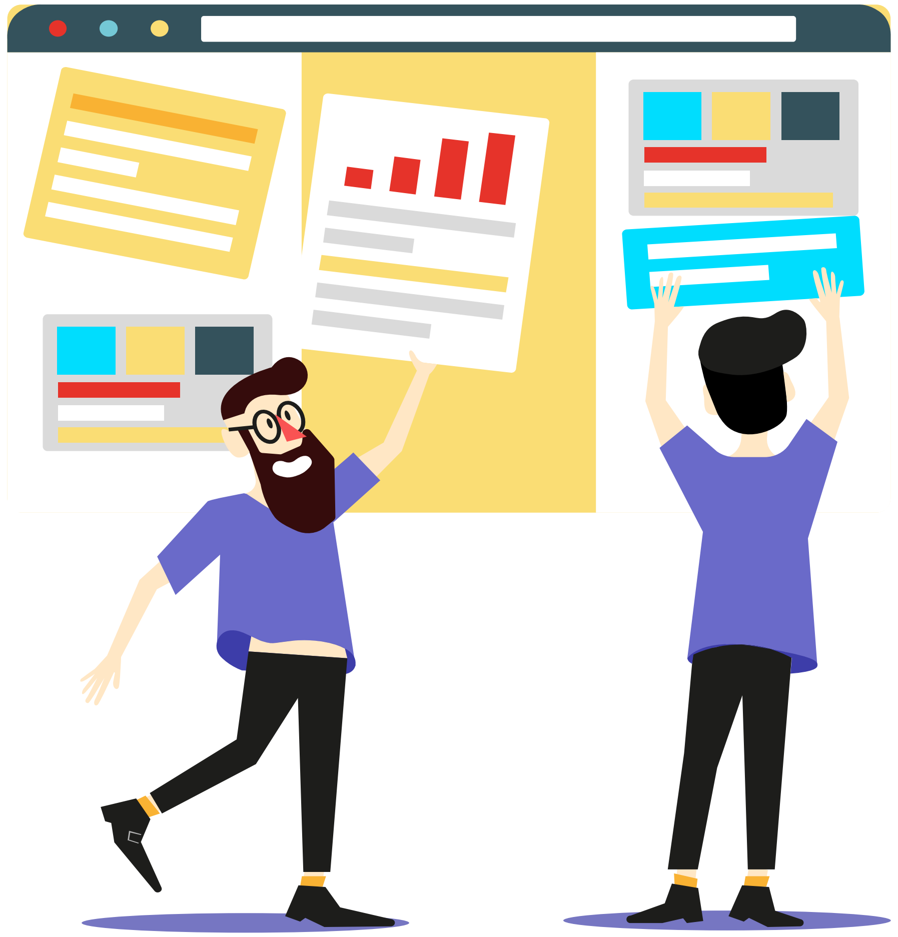


Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99

6. Finish! Practice!

Homework 1:

Design and make your own Shopping List or TODO List application.



/04

State Management

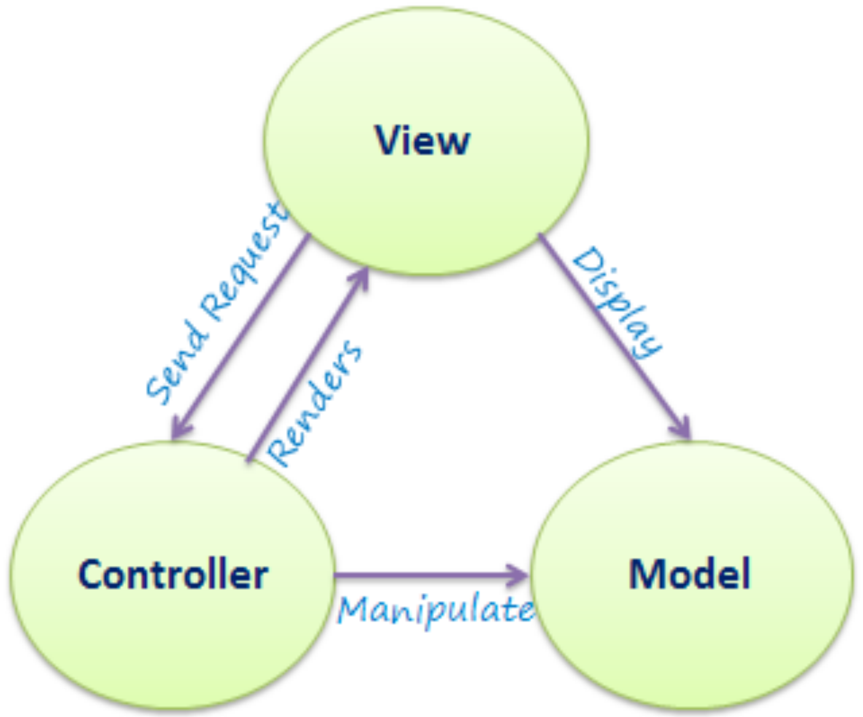
State management refers to the management of the state of one or more user interface controls such as text fields, OK buttons, etc. in a graphical user interface. In this user interface programming technique, the state of one UI control depends on the state of other UI controls.

Model-View-Controller MVC

The Problem With Traditional MVC Architecture

The Model-View-Controller (MVC) pattern is familar to most front-end web developers nowadays. This pattern describes a separation between the data (model), the presentation (view) and the application logic (controller). This ensures that your application is built in a structured way and that you achieve a separation of concerns.

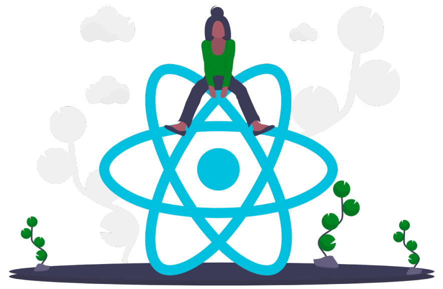
However the disadvantage is that you’re loosing control of your data flow. In general the data flow is bi-directional. The user input in one component can affect other components and vice versa. Controlling the flow of data and making sure that all user interface components update accordingly is an error-prone task.



Flux

What is Flux?
Flux is the application architecture for building client-side web applications.

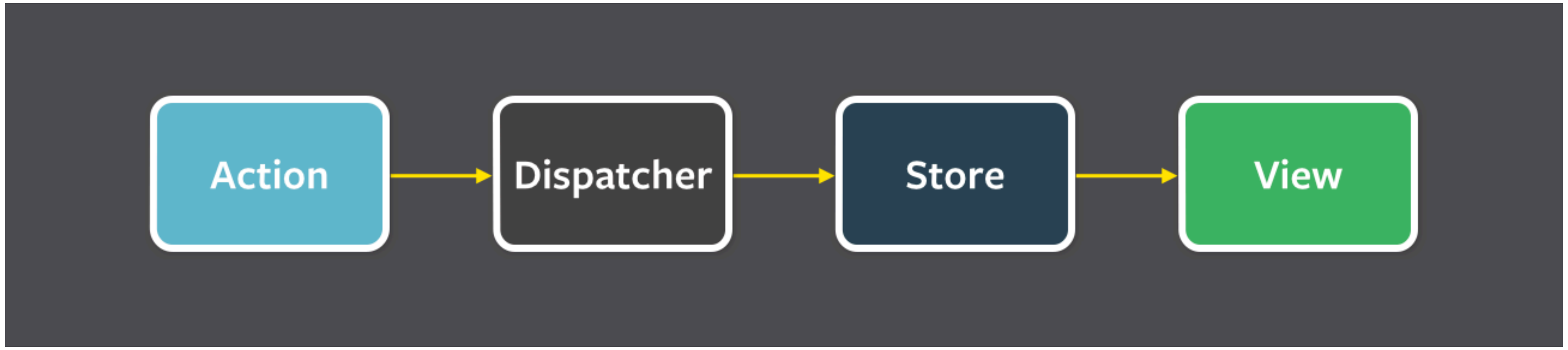
What does it do?
It complements React's composable view components by utilizing a unidirectional data flow.



How do I use it?
It's more of a pattern rather than a formal framework, and you can start using Flux immediately without a lot of new code.

Flux - Structure and Data Flow

Data in a Flux application flows in a single direction:

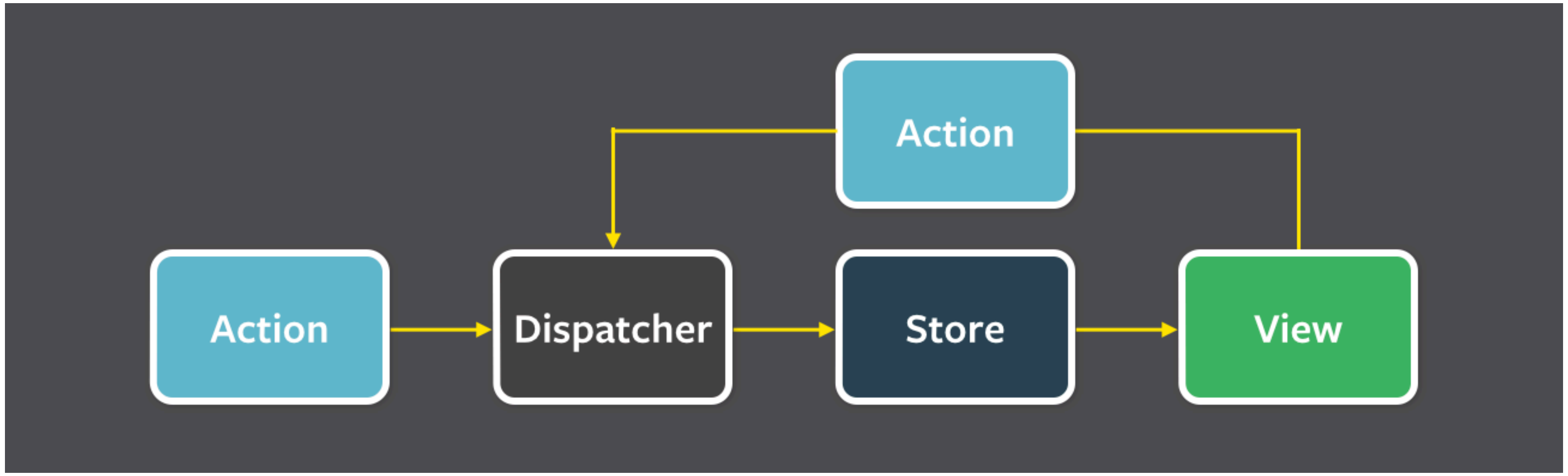


A unidirectional data flow is central to the Flux pattern, and the above diagram should be **the primary mental model for the Flux programmer**. The dispatcher, stores and views are independent nodes with distinct inputs and outputs. The actions are simple objects containing the new data and an identifying *type* property.

https://facebook.github.io/flux/

Flux - Structure and Data Flow

The views may cause a new action to be propagated through the system in response to user interactions:

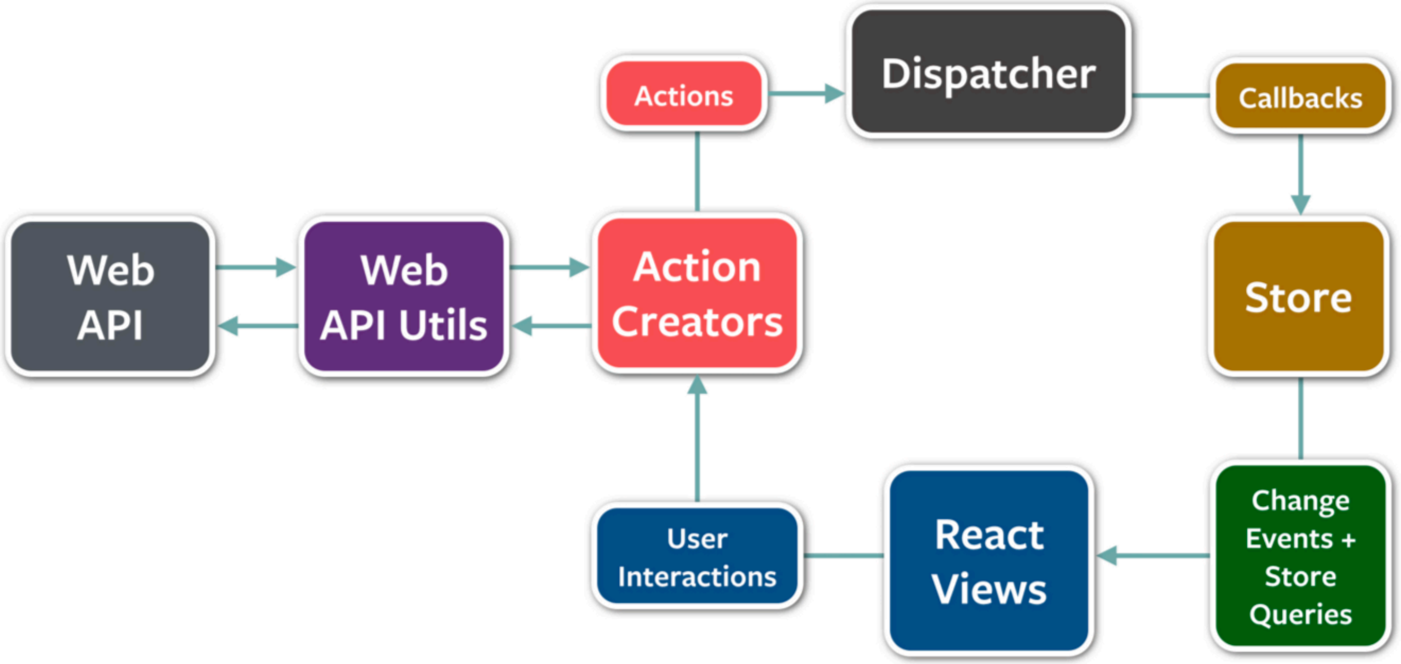


All data flows through the dispatcher as a central hub. Actions are provided to the dispatcher in an *action creator* method, and most often originate from user interactions with the views.
The dispatcher then invokes the callbacks that the stores have registered with it, dispatching actions to all stores.
Within their registered callbacks, stores respond to whichever actions are relevant to the state they maintain. The stores then emit a *change* event to alert the controller-views that a change to the data layer has occurred.

Redux

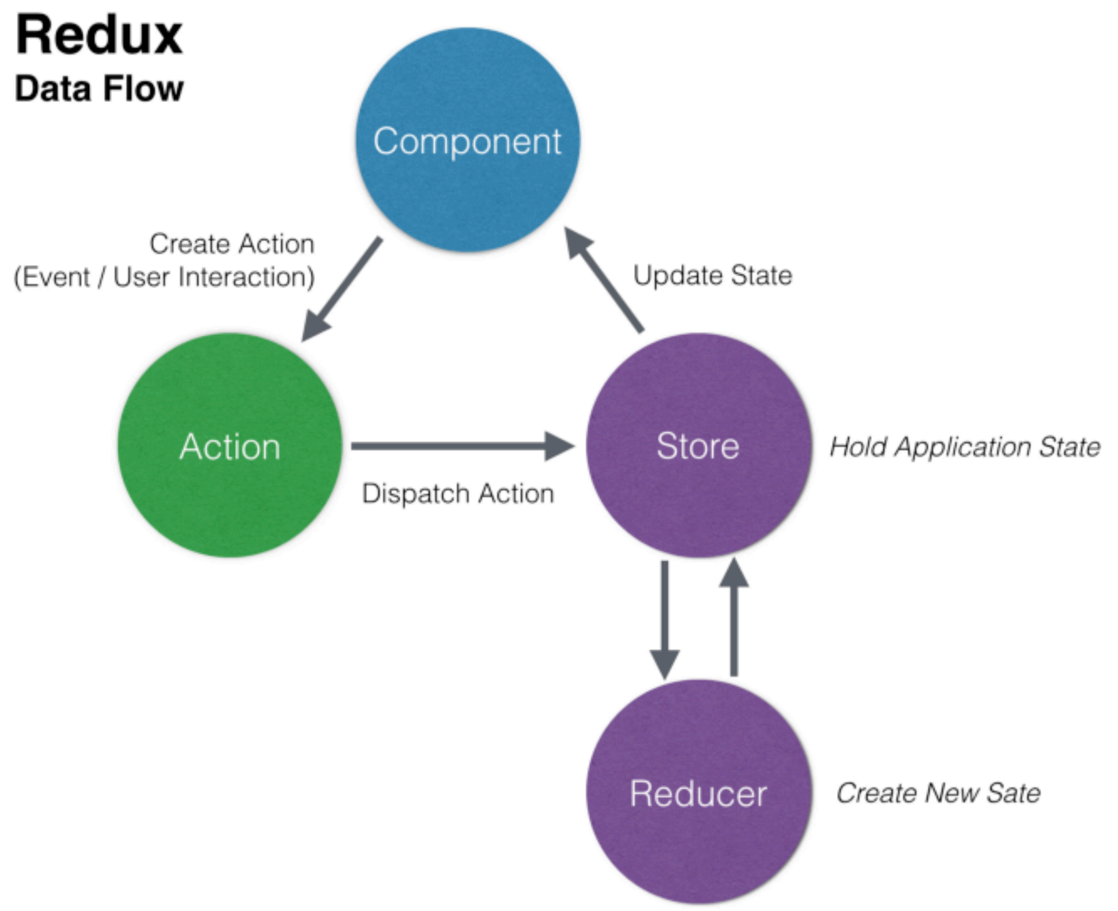
Flux is a pattern and Redux is a library.

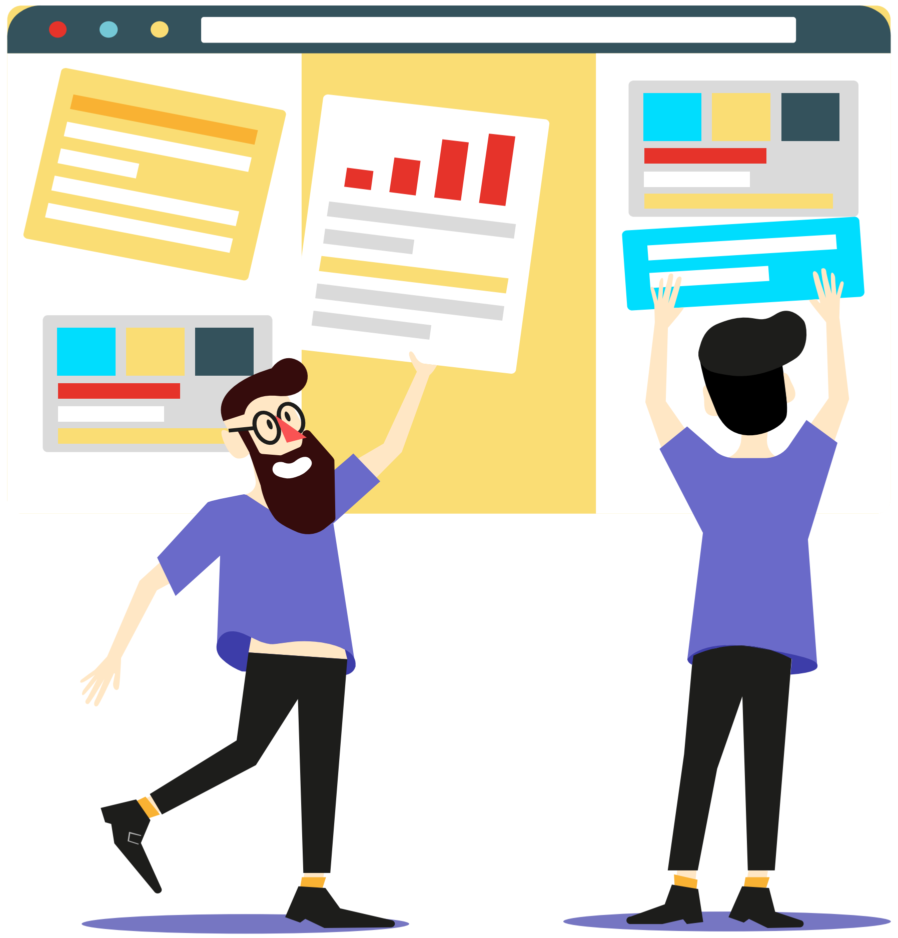
By using Redux we're solving this problems by introducing a central data store in our application. The store contains the state of the application and is the source of truth for components. By using the store concept you do not need to synchronize state between components manually. Instead you can fully rely on the Redux store at any time.



Redux

- **Actions**: actions are used to send information from the application to the store. Sending information to the store is needed to change the application state after a user interaction, internal events or API calls.
- **Reducers**: reducers are the most important building block and it's important to understand the concept. Reducers are pure JavaScript functions that take the current application state and an action object and return a new application state.
- **Store:** The store is the central objects that holds the state of the application.





/05

Third-Party Libraries

UI Design Library

Third-party libraries recommendation

UI Design Library

Ant Design (By Ant Finance)
Element UI (By Ele.me)

Ant Design

A design system with values of Nature and Determinacy for better user experience of enterprise applications

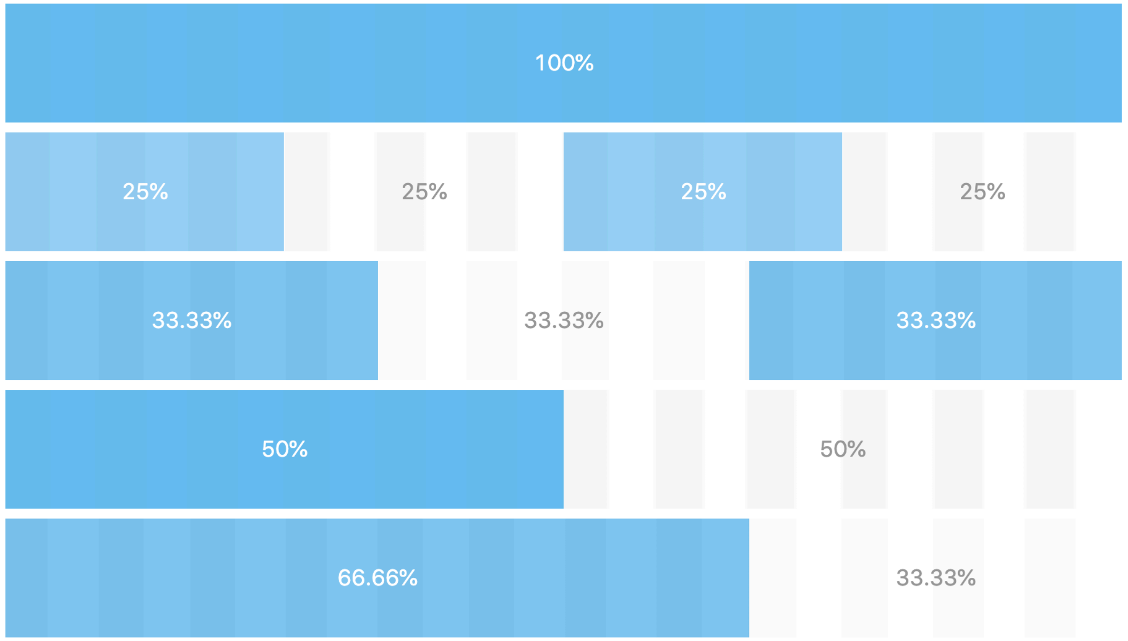
A Desktop UI Library

Element, a Vue 2.0 based component library for developers, designers and product managers

https://element.eleme.cn; https://ant.design

UI Design Library - Grid

In the grid system, we define the frame outside the information area based on row and column, to ensure that every area can have stable arrangement.
Following is a brief look at how it works:
Establish a set of column in the horizontal space defined by row (abbreviated col)
Your content elements should be placed directly in the col, and only col should be placed directly in row
The column grid system is a value of 1-24 to represent its range spans. For example, three columns of equal width can be created by .col-8 (span=8).
If the sum of col spans in a row are more than 24, then the overflowing col as a whole will start a new line arrangement.

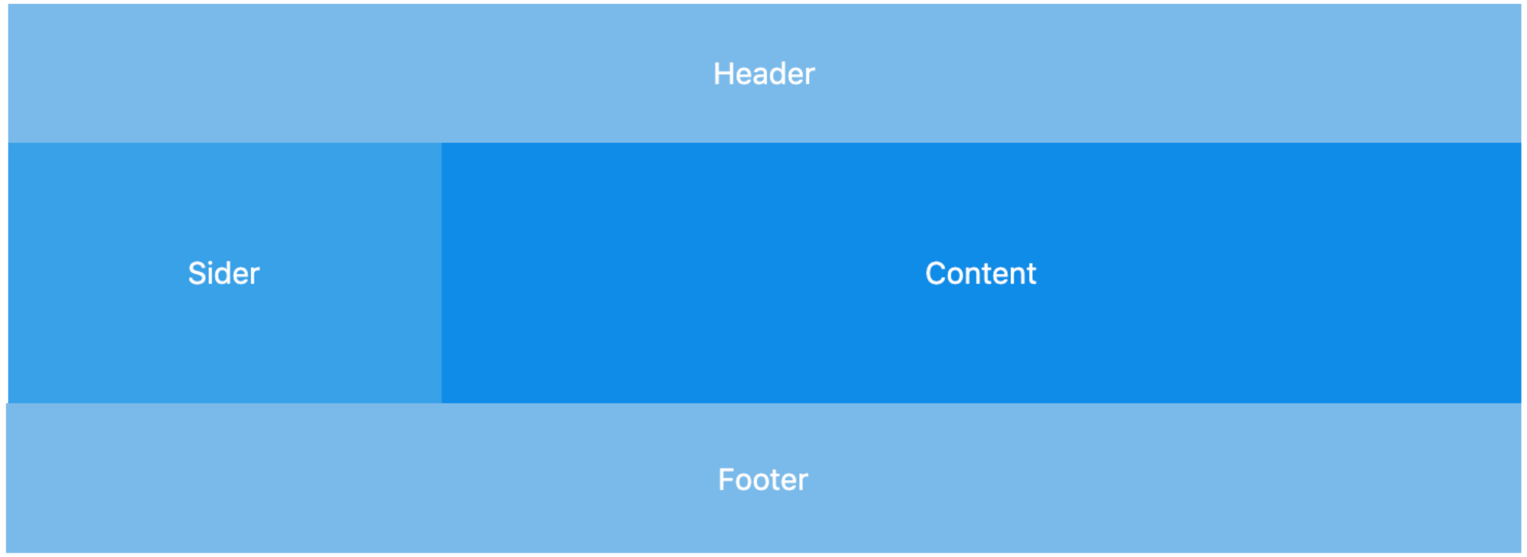


https://ant.design/components/grid/

UI Design Library - Layout

Handling the overall layout of a page.

Layout Components: Layout, Header, Sider, Content, Footer.



```
<Layout>
  <Header>Header</Header>
  <Layout>
    <Sider>Sider</Sider>
    <Content>Content</Content>
  </Layout>
  <Footer>Footer</Footer>
</Layout>
```

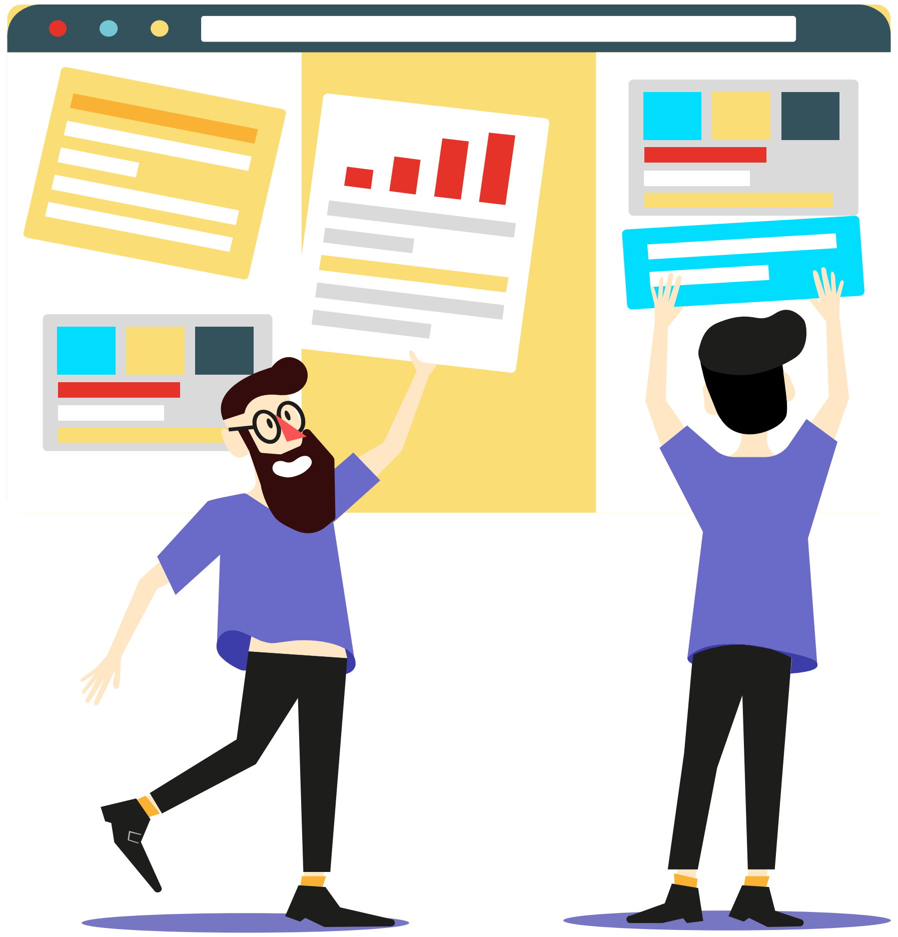
https://ant.design/components/layout/

Libraries Recommendation

Router	react-router
Layout	@rebass/grid react-blocks react-flexbox-grid
Drag and drop	react-beautiful-dnd react-dnd react-sortable-hoc
Code Editor	react-codemirror2 react-monaco-editor
Rich Text Editor	react-quill braft-editor
JSON Viewer	react-json-view
Color Picker	rc-color-picker react-color
Media Query	react-responsive react-media
Copy to clipboard	react-copy-to-clipboard
Document head manager	react-helmet react-document-title
Icons	react-fa react-icons
QR Code	qrcode.react
Charts	BizCharts recharts victory
Visual Graph Editor	GGEditor
Top Progress Bar	nprogress

i18n	react-intl
Code highlight	react-syntax-highlighter
Markdown renderer	react-markdown
Infinite Scroll	react-virtualized antd-table-infinity
Map	react-google-maps google-map-react react-amap
Context Menu	react-contextmenu react-contexify
Emoji	emoji-mart
Split View	react-split-pane
Image Crop	react-image-crop
Trend Lines	react-sparklines
Keywords highlight	react-highlight-words
Animation	react-move Ant Motion react-spring

https://ant.design/docs/react/recommendation



/06

Homework

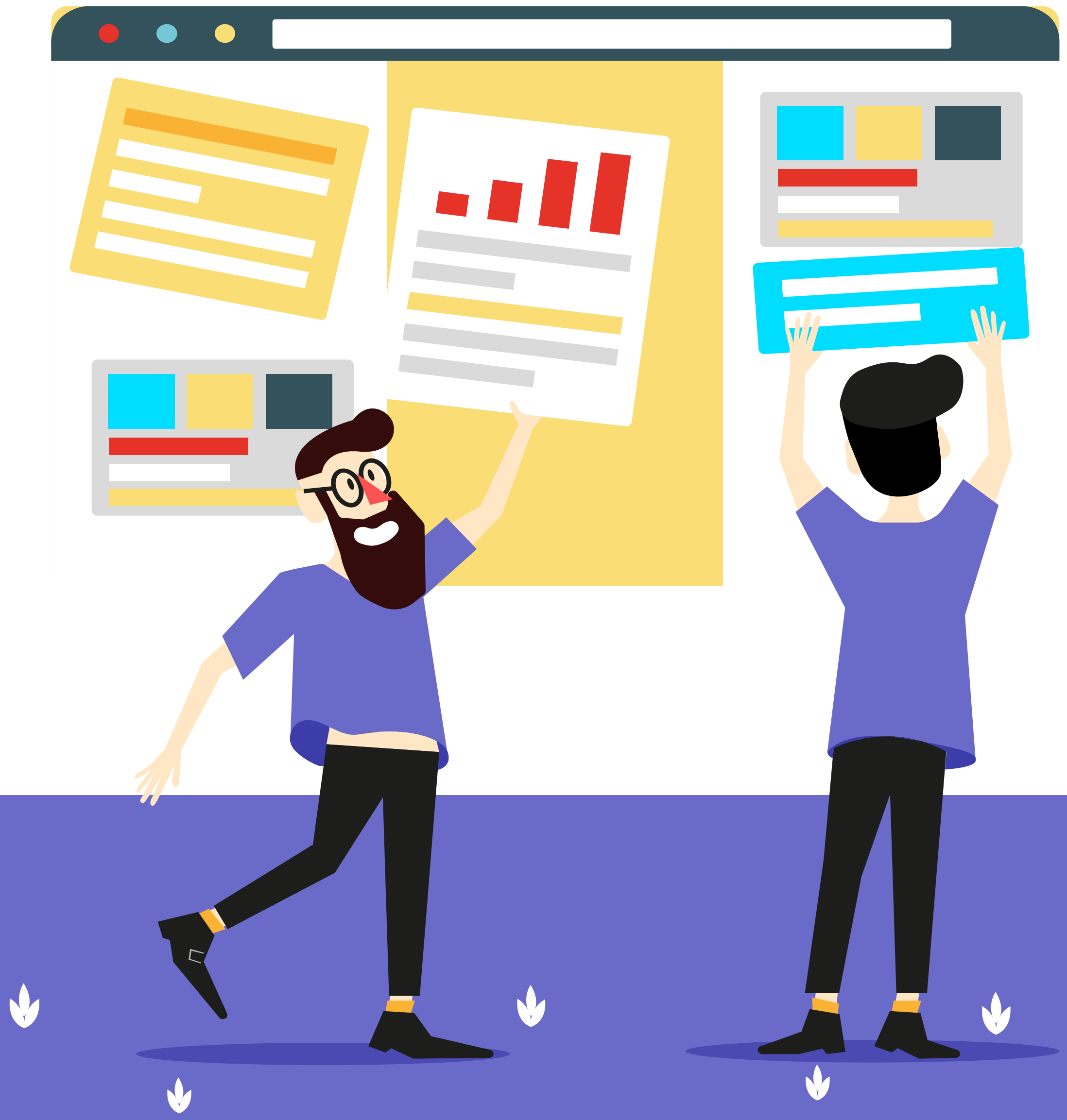
Homework

Homework 1:

Design and make your own Shopping List or TODO List application based on Part 3.

Homework 2:

Design and make a notebook application based on Part 4 & Part 5.



Web Framework Development Tutorial

Danqing Shi
Tongji University