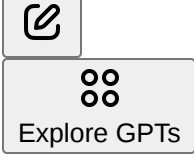




ChatGPT



Today

- 1. [MobX React TS Structure](#)
- 2. [RootStore vs Individual Stores](#)
- 3. [MobX React TS Structure](#)
- 4. [Best React State Management](#)
- 5. [Efficient React State Management](#)
- 6. [Stripe Payment Integration Setup](#)
- 7. [What is Stripe](#)

Previous 7 Days

- 1. [Redux vs Zustand Comparison](#)
- 2. [Twilio Free Services Overview](#)
- 3. [React TS Node SMS](#)
- 4. [React TS CRUD App](#)
- 5. [Three Column Card Layout](#)
- 6. [Auth0 vs JWT](#)

Previous 30 Days

- 1. [New Label Mumps Inquiry](#)
- 2. [Outlook Alert Issues](#)
- 3. [Java Backend Learning Timeline](#)

July

- 1. [Node.js vs Java API](#)
- 2. [Merge Rows in Excel](#)
- 3. [Ignore First Word MUMPS](#)

June

- 1. [React vs React Native](#)
- 2. [View File Name Extensions](#)
- 3. [Create Gmail group in Contacts](#)
- 4. [Redux vs MobX: Comparison](#)
- 5. [Pagination in MUMPS](#)
- 6. [View Last Global in Web](#)
- 7. [Zmailsend Cache Email Routine](#)
- 8. [Outlook No Popup Notifications](#)
- 9. [Lamborghini Car Prices USA](#)

Upgrade planGet GPT-4, DALL-E, and more



ChatGPT

GIVE ME THE BEST STRUCTURE FOR MOBX REACT TS WITH 3 STORES



Memory updated

Here's a well-organized structure for a MobX-React TypeScript project with three stores. This structure ensures scalability, maintainability, and clean separation of concerns.

1. Project Structure

```
javascript
Copy code
/src
|-- /components      // Reusable components (e.g., buttons, inputs)
|-- /common          // Components specific to certain pages or features
|-- /specific
|-- /stores
|  |-- /RootStore.ts  // Aggregates all the stores
|  |-- /StoreA.ts     // First store
|  |-- /StoreB.ts     // Second store
|  |-- /StoreC.ts     // Third store
|-- /pages
|  |-- /PageA         // Components related to Page A
|  |-- /PageB         // Components related to Page B
|  |-- /PageC         // Components related to Page C
|-- /hooks            // Custom hooks
|-- /types            // TypeScript types and interfaces
|-- /utils            // Utility functions
|-- App.tsx          // Main App component
|-- index.tsx        // Entry point
```

2. Detailed Breakdown

```
typescript
Copy code
import StoreA from './StoreA';
import StoreB from './StoreB';
import StoreC from './StoreC';

class RootStore {
  storeA: StoreA;
  storeB: StoreB;
  storeC: StoreC;

  constructor() {
    this.storeA = new StoreA(this);
    this.storeB = new StoreB(this);
    this.storeC = new StoreC(this);
  }
}

const rootStore = new RootStore();
export default rootStore;
export const StoreContext = React.createContext(rootStore);
```

```
typescript
Copy code
import { makeAutoObservable } from 'mobx';
import RootStore from './RootStore'; // For cross-store communication

class StoreA {
  rootStore: RootStore;
  data: string = '';

  constructor(rootStore: RootStore) {
    this.rootStore = rootStore;
    makeAutoObservable(this);
  }

  setData(data: string) {
    this.data = data;
  }
}

export default StoreA;
```

```
typescript
Copy code
import { makeAutoObservable } from 'mobx';
import RootStore from './RootStore';

class StoreB {
  rootStore: RootStore;
  count: number = 0;

  constructor(rootStore: RootStore) {
    this.rootStore = rootStore;
    makeAutoObservable(this);
  }

  increment() {
    this.count += 1;
  }
}

export default StoreB;
```

```
typescript
Copy code
import { makeAutoObservable } from 'mobx';
import RootStore from './RootStore';

class StoreC {
  rootStore: RootStore;
  items: string[] = [];

  constructor(rootStore: RootStore) {
    this.rootStore = rootStore;
    makeAutoObservable(this);
  }

  addItem(item: string) {
    this.items.push(item);
  }
}

export default StoreC;
```

3. Using Stores in Components

```
typescript
Copy code
import React, { useContext } from 'react';
import { observer } from 'mobx-react-lite';
import { StoreContext } from '../stores/RootStore';

const MyComponent: React.FC = observer(() => {
  const { storeA, storeB, storeC } = useContext(StoreContext);

  return (
    <div>
      <h2>{storeA.data}</h2>
      <button onClick={() => storeB.increment()}>Increment: {storeB.count}</button>
      <ul>
        {storeC.items.map((item, index) => (
          <li key={index}>{item}</li>
        ))}
      </ul>
    </div>
  );
});

export default MyComponent;
```

4. Custom Hooks

Create custom hooks if you find yourself repeatedly accessing stores in your components:

```
typescript
Copy code
import { useContext } from 'react';
import { StoreContext } from '../stores/RootStore';

export const useStore = () => useContext(StoreContext);
```

```
typescript
Copy code
import React from 'react';
import { observer } from 'mobx-react-lite';
import { useStore } from '../hooks/useStore';

const MyComponent: React.FC = observer(() => {
  const { storeA, storeB, storeC } = useStore();

  // Same as before...
});
```

5. Types and Interfaces

Place shared types and interfaces in the /types folder for easy reuse and consistency across the project.

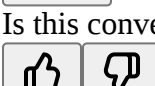
6. Utilities

Put utility functions in the /utils folder, keeping the business logic clean and separated from UI concerns.

This structure is designed to keep your MobX-React-TS project modular, scalable, and easy to maintain.



Is this conversation helpful so far?



You've hit the Free plan limit for GPT-4o. Responses will use our basic model until your limit resets after 2:15 PM.

Get Plus



Message ChatGPT



ChatGPT can make mistakes. Check important info.

