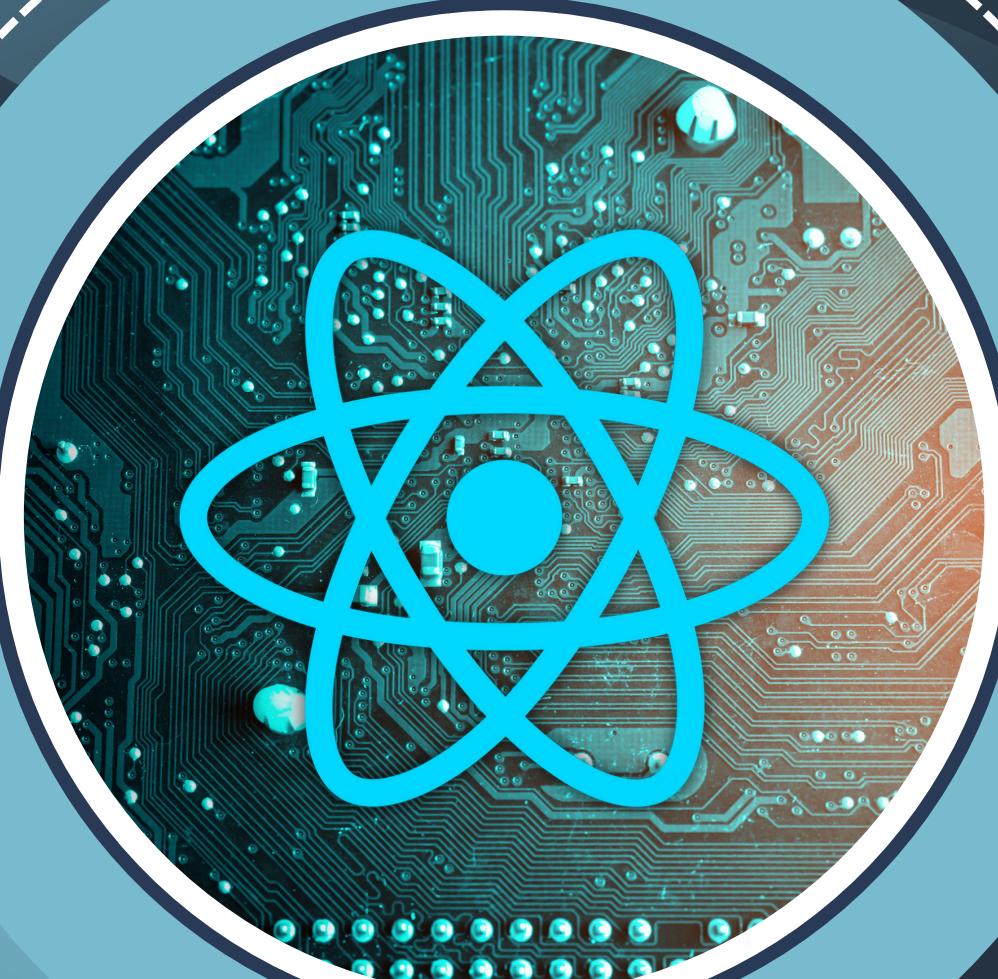


Adapter

دیزاین پترن هادر React

Structural Patterns

به زبان ساده همراه با مثال های عملی
کاربرد دیزاین پترن ها برای برنامه نویسان React



Structural Patterns

Adapter

الگوی Adapter برای هماهنگ کردن رابط های ناسازگار استفاده می شود. این الگو با ارائه یک "واسط" یا "رابط تبدیل"، امکان کار کردن دو سیستم ناسازگار را فراهم می کند. از این الگو می توان برای ایجاد کامپونت های قابل استفاده مجدد یا یکپارچه سازی کتابخانه ها استفاده کرد.

فرض کنید می خواهیم یک سبد خرید بسازیم اما بجای اینکه در کد سبد خرید به صورت مستقیم از Local Storage یا Cookie برای ذخیره سازی اطلاعات استفاده کنیم، به بخش سبد خرید یک Adapter پاس دهیم و در آینده با عوض کردن Adapter ها به راحتی بتوانیم نحوه ذخیره سازی اطلاعات را تغییر دهیم.

بخش IProduct

یک Interface با زیر های id و qty (quantity) را به شکل زیر می سازیم.



IProduct



```
export interface IProduct {  
    id: number;  
    qty: number;  
}
```

بخش IMemoryAdapter

این Interface بیانگر آن است که هر Adapter فارغ از اینکه به چه شکلی پیاده سازی شده باشد باید داری دو متده load برای دریافت اطلاعات از حافظه و save برای ذخیره سازی اطلاعات در حافظه باشد.



IMemoryAdapter



```
export interface IMemoryAdapter<T> {
    load(): T[];
    save(items: T[]): void;
}
```

بخش CartService

بخش سبد خرید را به شکلی توسعه می دهیم که هیچ وابستگی به شیوه ذخیره سازی اطلاعات وجود نداشته باشد، به عبارتی در این بخش هیچ کاری نداریم که اطلاعات را می خواهیم در Local Storage یا Cookie ذخیره کنیم.



```
import { IMemoryAdapter } from "./Adapters/memory-adapter.interface";
import { IProduct } from "./product.interface";

export class CartService {
  private items: IProduct[] = [];
  private memoryAdapter: IMemoryAdapter<IProduct>;

  constructor(storageAdapter: IMemoryAdapter<IProduct>) {
    this.memoryAdapter = storageAdapter;
    this.items = this.memoryAdapter.load();
  }

  add(id: number, qty: number): void {}

  get(): IProduct[] {}

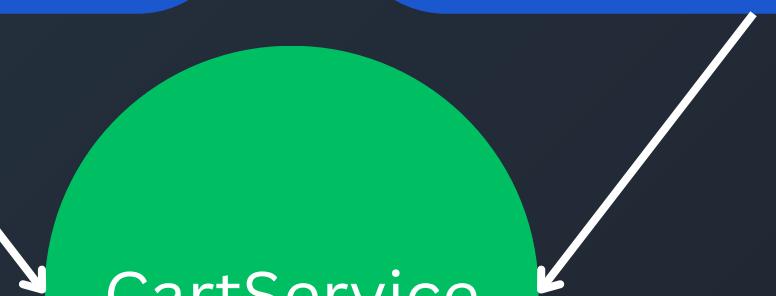
  remove(id: number): void {}

  increaseQuantity(id: number, qty: number): void {}
}
```

CookieAdapter

LocalStorageAdapter

CartService



بخش CartService - Methods

از چهار Method زیر برای کار با سرویس سبد خرید استفاده می‌کنیم.



CartService - Methods

```
add(id: number, qty: number): void {
  const existingProduct = this.items.find(item => item.id === id);
  if (existingProduct) {
    existingProduct.qty += qty;
  } else {
    this.items.push({ id, qty });
  }
  this.memoryAdapter.save(this.items);
}

get(): IProduct[] {
  return this.items;
}

remove(id: number): void {
  this.items = this.items.filter(item => item.id !== id);
  this.memoryAdapter.save(this.items);
}

increaseQuantity(id: number, qty: number): void {
  const product = this.items.find(item => item.id === id);
  if (product) {
    product.qty += qty;
    this.memoryAdapter.save(this.items);
  } else {
    throw new Error(`Product with id ${id} not found`);
  }
}
```

add

افزودن آیتم جدید

get

دربافت اطلاعات

remove

حذف یک آیتم

increaseQuantity

افزایش مقدار یک آیتم

پیاده سازی اولین Adapter

اولین Adapter خود را که قرار است با استفاده از Cookie عملیات مدیریت حافظه را انجام دهد را با نام زیر می سازیم.



CookieAdapter

```
import { IMemoryAdapter } from "./memory-adapter.interface";

export class CookieAdapter<T> implements IMemoryAdapter<T> {
  private cookieKey: string;
  private expirationDays: number;

  constructor(cookieKey: string, expirationDays: number) {
    this.cookieKey = cookieKey;
    this.expirationDays = expirationDays;
  }

  private setCookie(name: string, value: string, days: number): void {
    const expires = new Date(Date.now() + days * 24 * 60 * 60 * 1000).toUTCString();
    document.cookie = `${name}=${encodeURIComponent(value)}; expires=${expires}; path=/`;
  }

  private getCookie(name: string): string | null {
    const cookies = document.cookie.split('; ').reduce((acc, cookie) => {
      const [key, val] = cookie.split('=');
      acc[key] = decodeURIComponent(val);
      return acc;
    }, {} as Record<string, string>);
    return cookies[name] || null;
  }

  load(): T[] {
    const data = this.getCookie(this.cookieKey);
    return data ? (JSON.parse(data) as T[]) : [];
  }

  save(items: T[]): void {
    this.setCookie(this.cookieKey, JSON.stringify(items), this.expirationDays);
  }
}
```

CookieAdapter

load

save

پیاده سازی دومین Adapter

دومین Adapter خود را که قرار است با استفاده از Local Storage عملیات مدیریت حافظه را انجام دهد را با نام LocalStorageAdapter و به شکل زیر می سازیم.



LocalStorageAdapter

```
import { IMemoryAdapter } from "./memory-adapter.interface";

export class LocalStorageAdapter<T> implements IMemoryAdapter<T> {
  private storageKey: string;

  constructor(storageKey: string) {
    this.storageKey = storageKey;
  }

  load(): T[] {
    const data = localStorage.getItem(this.storageKey);
    return data ? (JSON.parse(data) as T[]) : [];
  }

  save(items: T[]): void {
    localStorage.setItem(this.storageKey, JSON.stringify(items));
  }
}
```

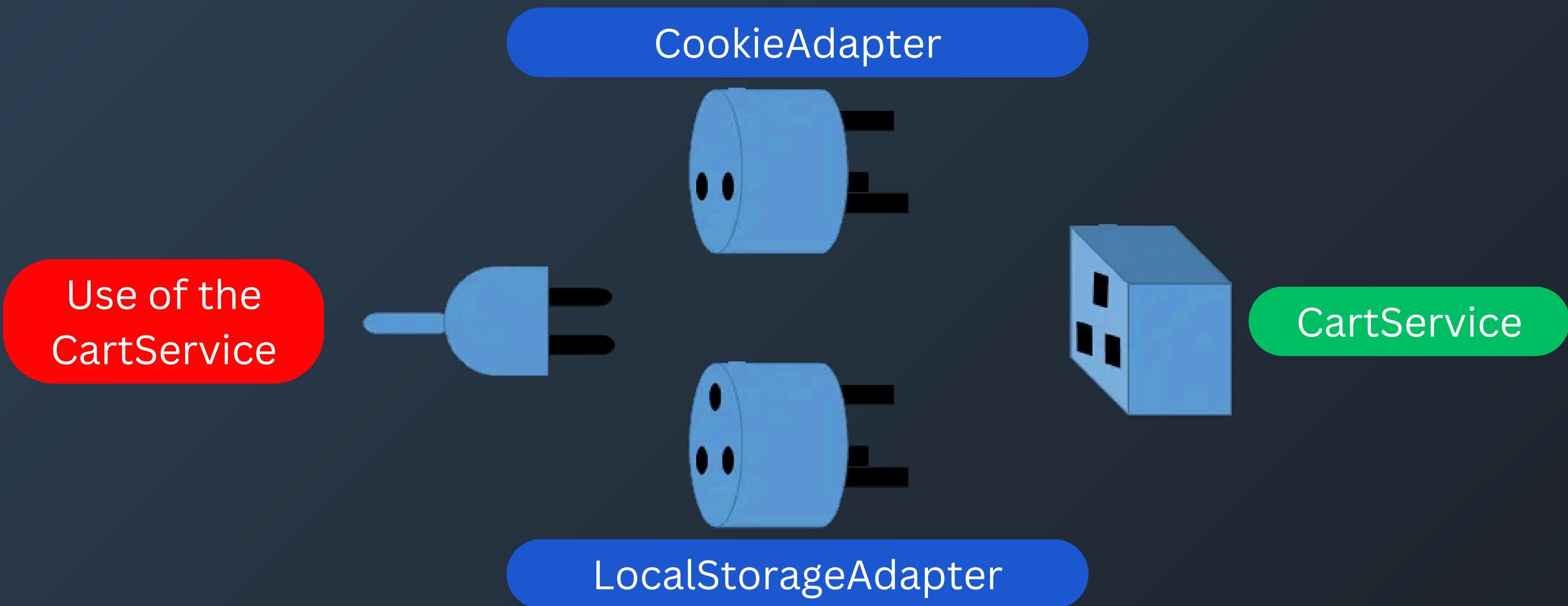
LocalStorageAdapter

load

save

جمع‌بندی Adapter‌ها

اکنون ما دو Adapter ساخته ایم که هر دو دارای دو Method به نام‌های `load` و `save` هستند. حال می‌توانیم به راحتی هر یک از این Adapter‌ها را به CartService بدهیم و نوع حافظه را بر اساس نیازمندی سیستم به سادگی تغییر دهیم.



طرز استفاده از CartService

حال می توانیم به شکل زیر هر یک از Adapter های موجود را به CartService بدهیم تا از آن جهت عملیات مدیریت حافظه استفاده کند.



AdapterDesignPattern

```
● ● ●  
import { useEffect, useState } from "react";
import { CartService } from "./pattern/cart.service";
import { IProduct } from "./pattern/product.interface";
import { CookieAdapter } from "./pattern/Adapters/cookie.adapter";
import { LocalStorageAdapter } from "./pattern/Adapters/local-storage.adapter";

const AdapterDesignPattern = () => {
  const cookieAdapter = new CookieAdapter<IProduct>('cartItems', 7);
  const localStorageAdapter = new LocalStorageAdapter<IProduct>('cartItems');
  const cartService = new CartService(cookieAdapter);

  const [cart, setCart] = useState<IProduct[]>([]);
  useEffect(() => {
    setCart(cartService.get());
  }, [cart]);
  const handleClick = () => {
    cartService.add(1, 1);
  };
  return (
    <div>
      <button onClick={handleClick}>Add</button>
      {cart.length > 0 ? (
        <ul>
          {cart.map((item, index) => (
            <li key={index}>
              ID: {item.id}, Quantity: {item.qty}
            </li>
          )));
        </ul>
      ) : (
        <p>Empty!</p>
      )}
    </div>
  );
}

export default AdapterDesignPattern;
```

```
● ● ●  
<div>
  <button onClick={handleClick}>Add</button>
  {cart.length > 0 ? (
    <ul>
      {cart.map((item, index) => (
        <li key={index}>
          ID: {item.id}, Quantity: {item.qty}
        </li>
      )));
    </ul>
  ) : (
    <p>Empty!</p>
  )}
</div>
```