



# КАК ПЕРЕСТАТЬ БЫТЬ ФРЭЙМВОРК РАЗРАБОТЧИКОМ?

## FRONTEND TECHLEAD



1. +8 лет в разработке
2. Опыт в стартапах и корпорациях
3. Люблю чистую архитектуру и писать тесты
4. Занимаюсь обучением и менторством
5. Люблю походы и кататься на эндуро мотоциклах

# ЧТО БУДЕТ В ДОКЛАДЕ?

- Разберем типичный код на react и его архитектуру
- Обсудим проблемы
- Сделаем рефакторинг, порисуем схемы
- Напишем юнит тесты
- Плюсы, минусы и полезные практики

# **ПОЧЕМУ ВООБЩЕ РОДИЛСЯ ДОКЛАД?**





# ОБНОВЛЕНИЕ ИНСТРУМЕНТОВ СТАЛО НОРМОЙ



# SELECTORS/ACTIONS HELL



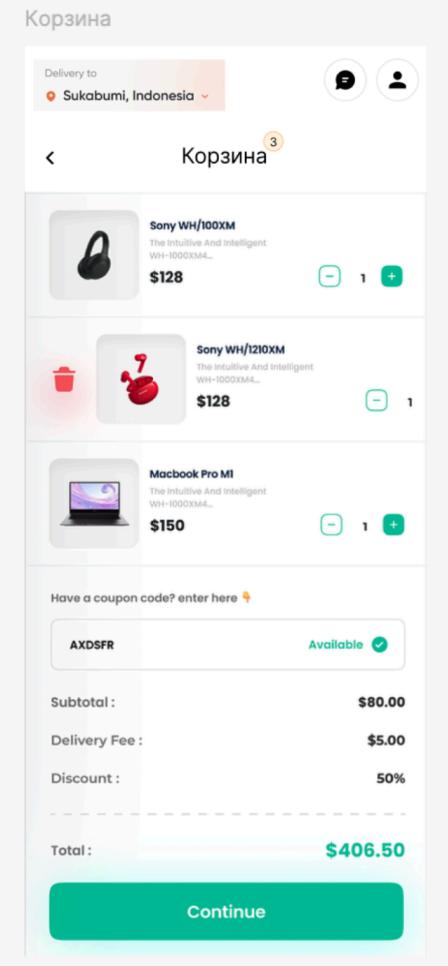
```
✓ selectors
  > __dataMocks__
  > __tests__
  ✓ entities
    > __dataStubs__
    > __tests__
    TS getCarrierNames.ts
    TS getCarriersOfOffer.ts
    TS getEntities.ts
    TS getFlight.ts
    TS getFlightCarriers.ts
    TS getFlights.ts
    TS getFlightsOfOffer.ts
    TS getFlightsSegments.ts
    TS getOffer.ts
    TS getOffers.ts
    TS getOffersCount.ts
    TS getOffersFromEntities.ts
    TS getSearchId.ts
    TS getSegment.ts
    TS getSegmentsOfFlight.ts
    TS index.ts
    TS types.ts
```

```
✓ actions
  > aggregateFlightsAndOffers
  > sorting
  ✓ streaming
    > __dataMocks__
    > __tests__
    TS normalizeStreamingResponse.ts
    TS prepareConsumerContextForStreamin...
    TS storeNormalizedData.ts
    TS streaming.ts
    TS streamingNormalizer.ts
    TS streamingParams.ts
    TS utils.ts
```

# ПОЯВИЛИСЬ ПРОБЛЕМЫ

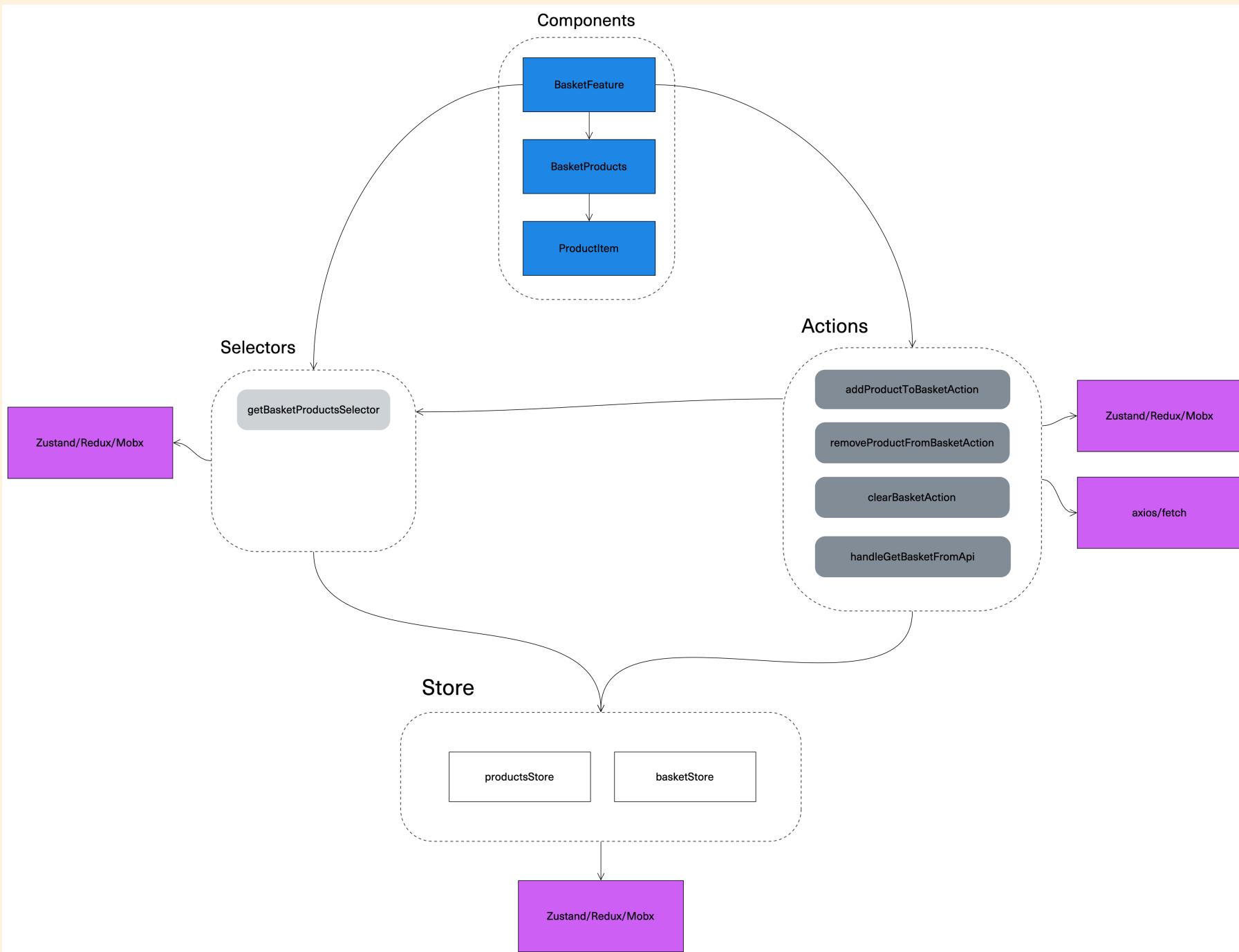
- Сложно вносить изменения
- Сложно разрабатывать тесты
- Сложно обновлять инструменты

**ВОЗЬМЕМ ЗА ОСНОВУ ПРИМЕР С  
МАРКЕТПЛЕЙСОМ**



- Добавить товар в корзину
- Удалить товар из корзины
- Сохранять в браузерное хранилище
- Синхронизировать корзину с API
- Расширять корзину функционалом

# ТИПИЧНАЯ АРХИТЕКТУРА ТАКОГО ПРОЕКТА



**СЛОЖНО РЕФАКТОРИТЬ**

```
1
2 export const addToCartAction = async (productId) => {
3     // Получаем продукт по ID
4     const product = productStore.getProductById(productId);
5     // Получаем количество продукта в корзине
6     const productInBasket = basketStore.getProductCount(productId)
7
8     // Проверяем, что товар еще доступен на складе
9     const noAvailableAmount = executeAction(checkProductAvailable,
10    if (noAvailableAmount) return;
11
12    // Добавляем в корзину
13    dispatch(ADD_TO_CART, { ...product, quantity: 1 })
14
15    // Обновляем localStorage
16    localStorage.setItem('cart', basketStore.getState());
17
18    // Синхронизация с API
19    await syncCartWithAPI();
20};
```

# СЛОЖНО ТЕСТИРОВАТЬ

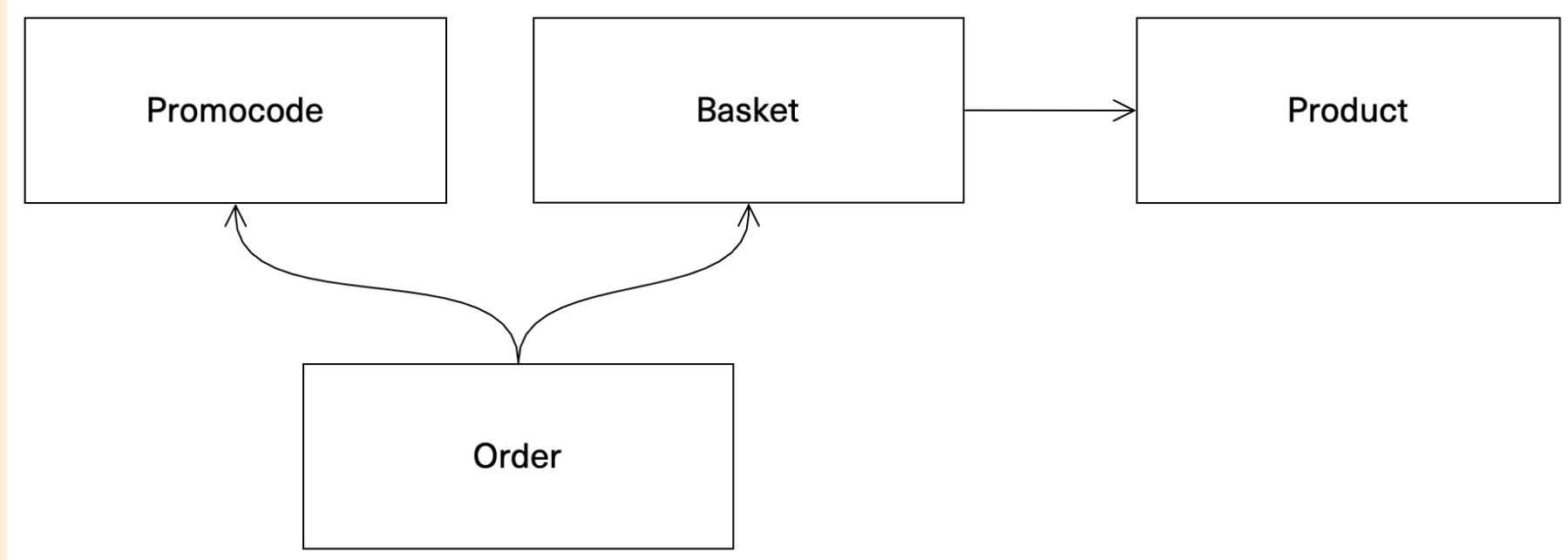
- Приходиться мокать инструменты
- Тесты разрабатывать сложно и долго
- Тесты нестабильны
- Слабая мотивация разрабатывать тесты

```
1
2 it('Проверка добавления товара в корзину', async () => {
3     const context = mockApplicationContext({
4         stores: [productStoreMock, baskStoreMock],
5         services: [localStorageMock, apiMock]
6     })
7     // Мокируем ответ от API
8     context.api.mockResponseOnce(JSON.stringify({ success: true }))
9
10    await runAction(addToCartAction(productId));
11
12    // Проверка, что товар в корзине
13    expect(basketStore.getState()).toHaveBeenCalledWith(jsonToCheck)
14
15    // Проверка обновления localStorage
16    expect(localStorageMock).toHaveBeenCalledWith('cart', expect.anything())
17
18    // Проверка вызова API
19    expect(apiMock).toHaveBeenCalledWith('addProduct', { productId })
20});
```

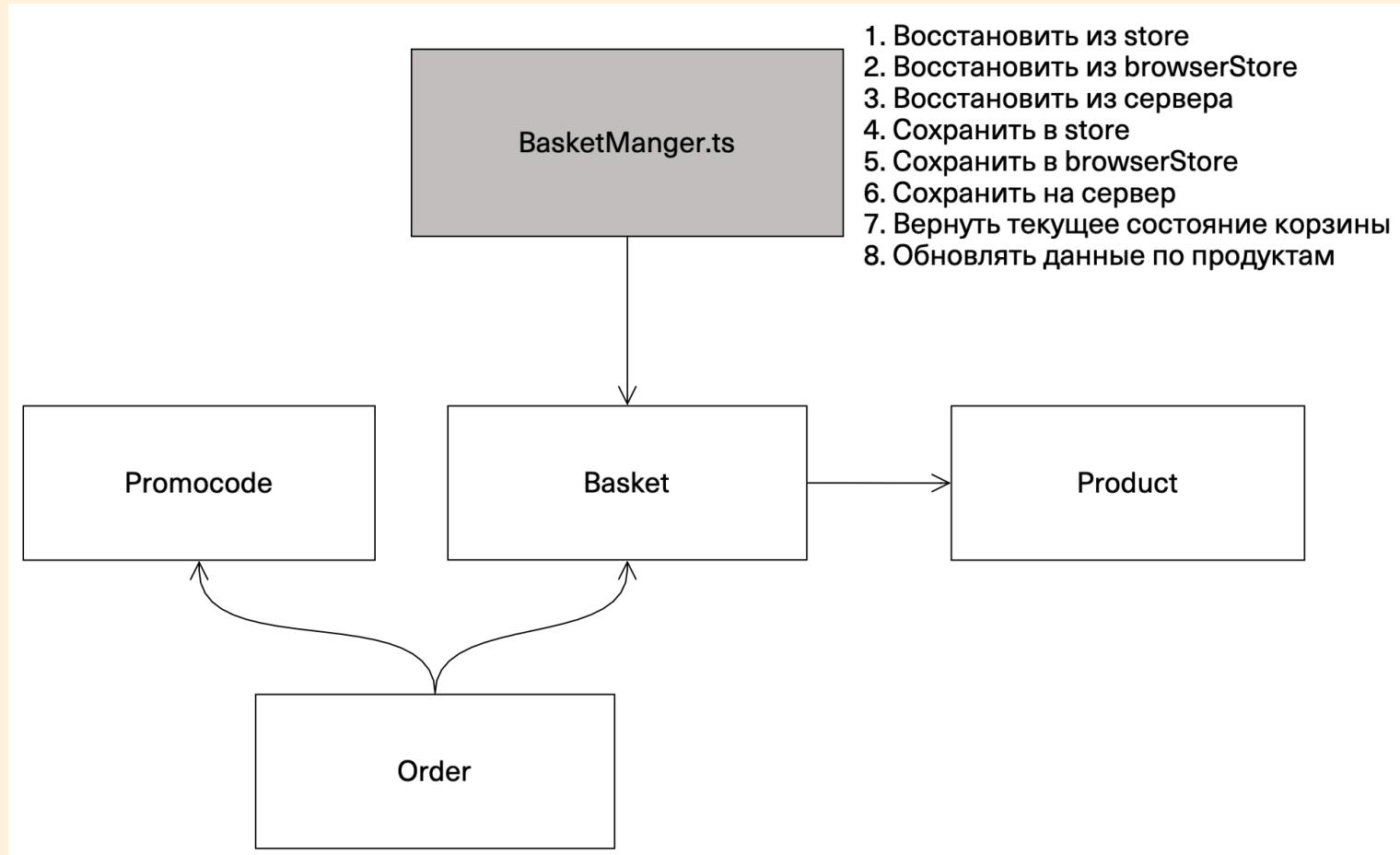
# СЛОЖНО РАСШИРЯТЬ

- Приходиться залезать в логику экшенов
- Есть риск что-нибудь поломать
- Переписывать тесты

**ВСЕ НАЧИНАЕТСЯ С ДОМЕНА**

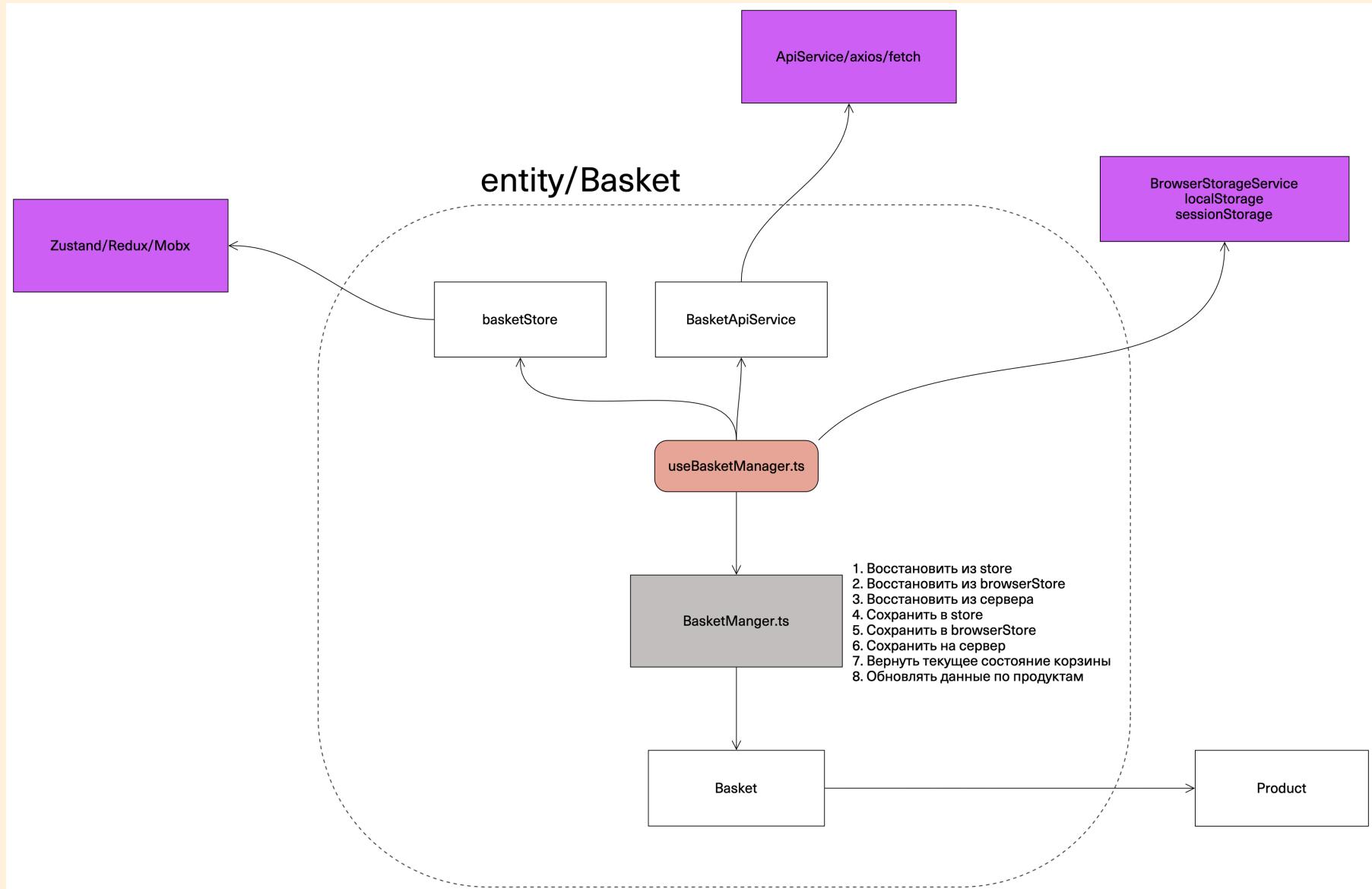


```
1 import { ProductId } from '@/shared/types/product';
2 import { IBasket, IBasketProduct } from '../types';
3
4
5 export class Basket {
6     products: IBasketProduct[] = [];
7     productsCount: Record<ProductId, number> = {};
8
9     constructor(defaultBasketState: IBasket) {}
10
11    addItem(product: IBasketProduct) {}
12
13    removeItem(product: IBasketProduct) {}
14
15    clearProducts() {}
16
17    get totalPrice() {}
18
19    get obj(): IBasket {
20        return {
21            products: this.products,
22            productsCount: this.productsCount,
23        };
24    }
25}
```



# ИЗБАВЛЯЕМСЯ ОТ ЗАЦЕПЛЕНИЯ С ПОМОЩЬЮ DI

```
1
2 import { Basket } from './Basket';
3 import { IBasket, IBasketProduct } from '../types';
4 import { BrowserStorageType } from '@/shared/BrowserStorage';
5 import { ProductId } from '@/shared/types/product';
6
7 export class BasketManager {
8   constructor(
9     updateStore: (basket: IBasket) => void,
10    updateApi: (basket: IBasket) => Promise<void>,
11    browserStorageService: BrowserStorageType,
12    defaultBasketState: IBasket
13  ) {
14    this.updateStore = updateStore;
15    this.updateApi = updateApi;
16    this.browserStorageService = browserStorageService;
17    this.basket = new Basket(defaultBasketState);
18  }
19
20  handleAddItemToBasket(product: IBasketProduct) {
21    this.basket.addItem(product);
22    this.updateStore(this.basket.obj)
23    this.updateApi(this.basket.obj)
24    this.browserStorageService.set('basket', this.basket.obj)
25  }
}
```



```
1 import { BasketManager } from '../models/BasketManager';
2 import { basketStore } from '../store/basketStore';
3 import { basket ApiService } from '@shared/api/basket ApiService';
4 import { browserStorageService } from '@shared/services/browserStorageService';
5
6
7 export const useGetBasketManager = () => {
8     const basketState = basketStore().getBasketState();
9     const updateBasketState = basketStore().updateBasketState;
10
11     const basketManager = new BasketManager(
12         updateBasketState,
13         basket ApiService,
14         browserStorageService,
15         basketState
16     );
17
18     return {
19         basketManager,
20     };
21 };
22
```

**ЛЕГЧЕ ТЕСТИРОВАТЬ  
ФУНКЦИОНАЛ**

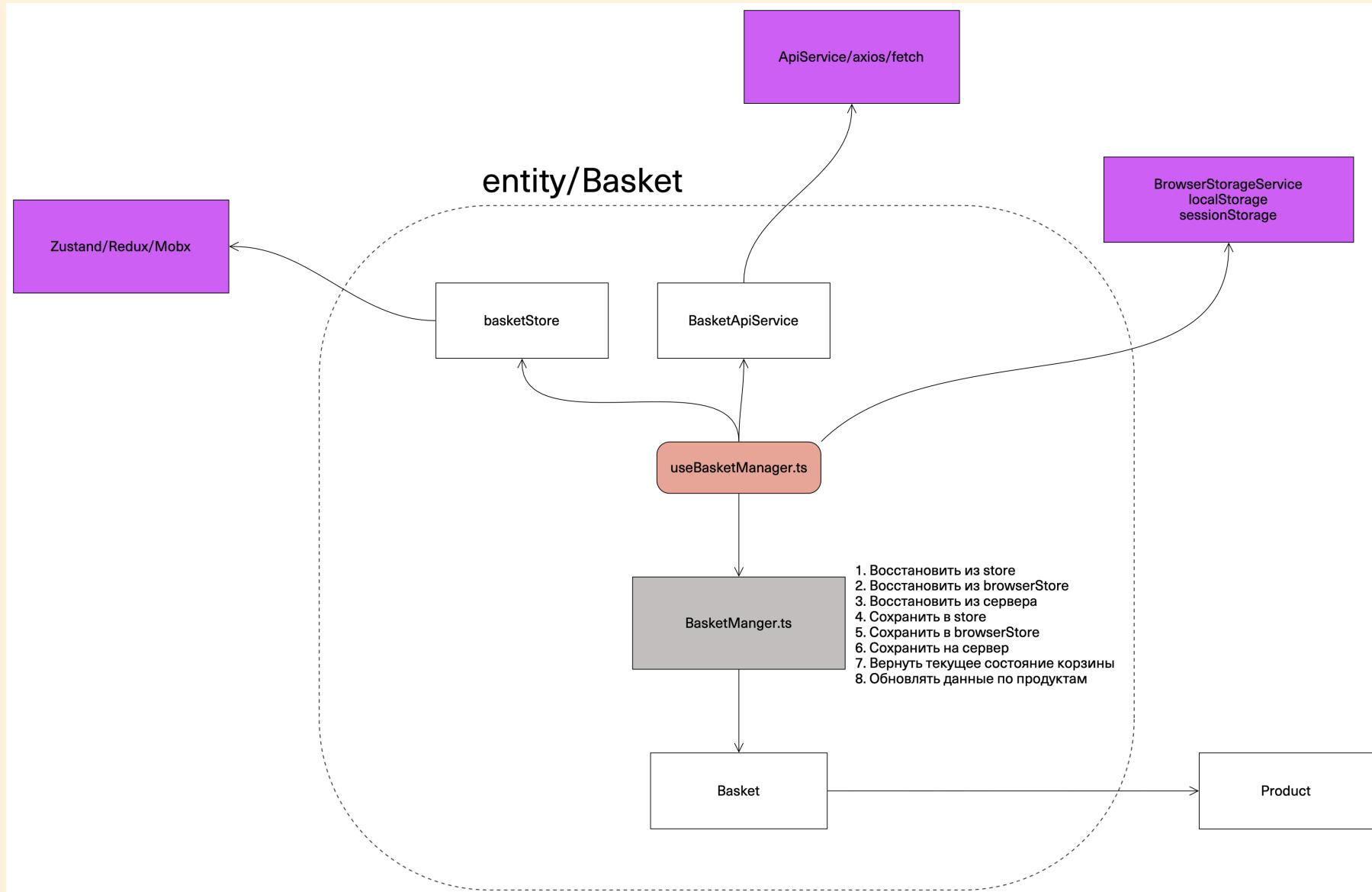
```
1
2 it('Товар успешно добавляется в корзину', () => {
3     basket = new Basket({ products: [], productsCount: {} });
4     basket.addItem(mockedProduct);
5
6     expect(basket.products).toHaveLength(1);
7     expect(basket.productsCount).toStrictEqual({ '1': 1 });
8     expect(basket.totalPrice).toBe(438.95);
9     expect(basket.obj).toStrictEqual({
10         products: [product],
11         productsCount: {
12             '1': 1,
13         },
14     });
15 });
16
```

```
1
2 test('Данные синхронизируются при добавлении в корзину', () => {
3     const basketManager = new BasketManager(
4         updateStoreMock, updateApiMock, browserStorageMock, defaultBasket);
5
6     basketManager.handleAddItemToBasket(mockedProduct);
7
8     const dataForProduct = {
9         products: [product],
10        productsCount: {
11            '1': 1,
12        },
13    };
14
15    expect(basketManager.basket.obj).toEqual(dataForProduct);
16    expect(browserStorageMock.getItem('basket')).toEqual(dataForProduct);
17    expect(updateStoreMock).toHaveBeenCalled();
18    expect(updateApiMock).toHaveBeenCalled();
19 });
20
```

**ЛЕГКО РАСШИРЯТЬ**

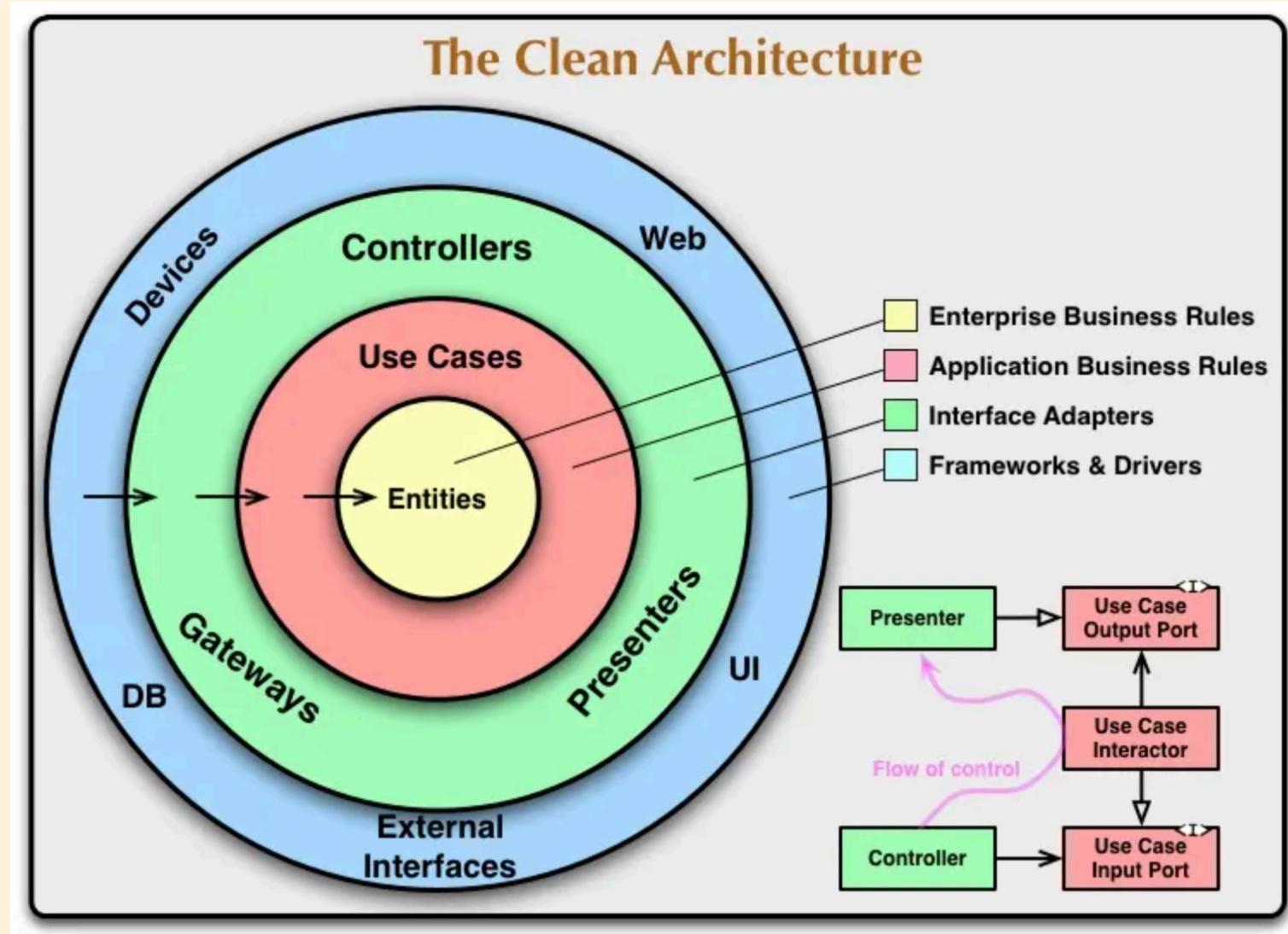
```
1
2 export class BasketManager {
3   constructor(
4     ...
5       notify: (message: string) => void
6   ) {
7     this.notify = notify;
8   }
9
10  handleAddItemToBasket(product: IBasketProduct) {
11    ...
12    this.notify('Товар добавлен в корзину')
13  }
14}
15
```

**ЛОГИКА НЕ ЗАВИСИТ ОТ  
ИНСТРУМЕНТА**



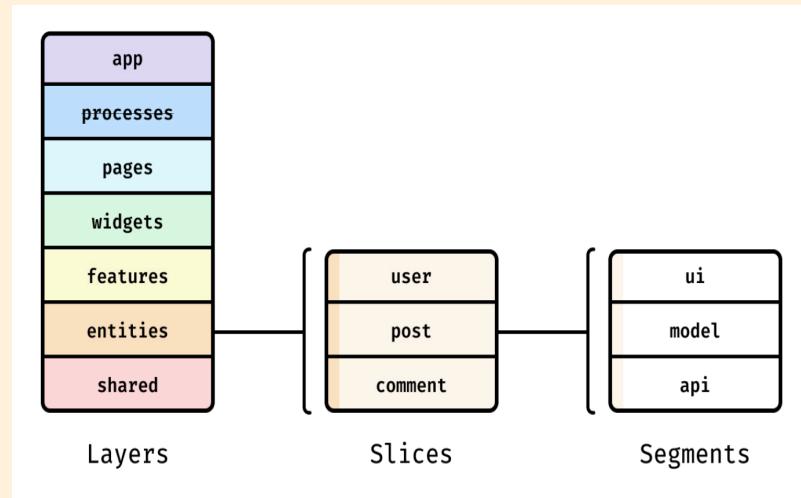
**НИЖЕ ПОРОГ ВХОДА И ПРОЩЕ  
ОНБОРДИНГ**

# ЧИСТАЯ АРХИТЕКТУРА





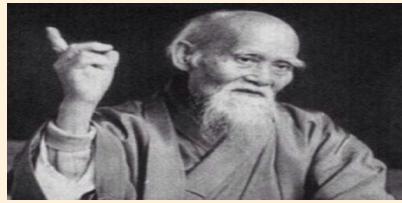
# FEATURE SLICED DESIGN



- Разделение архитектурных слоев
- Понятные направления зависимостей
- Не является панацеей

# ЧТО НУЖНО УЧИТЬ ВАТЬ?

- Усложняется инфраструктура
- Учитывать цели и задачи бизнеса



> Сначала думай потом кодируй

# ПОЛЕЗНЫЕ ССЫЛКИ

