


Требования к структуре проекта

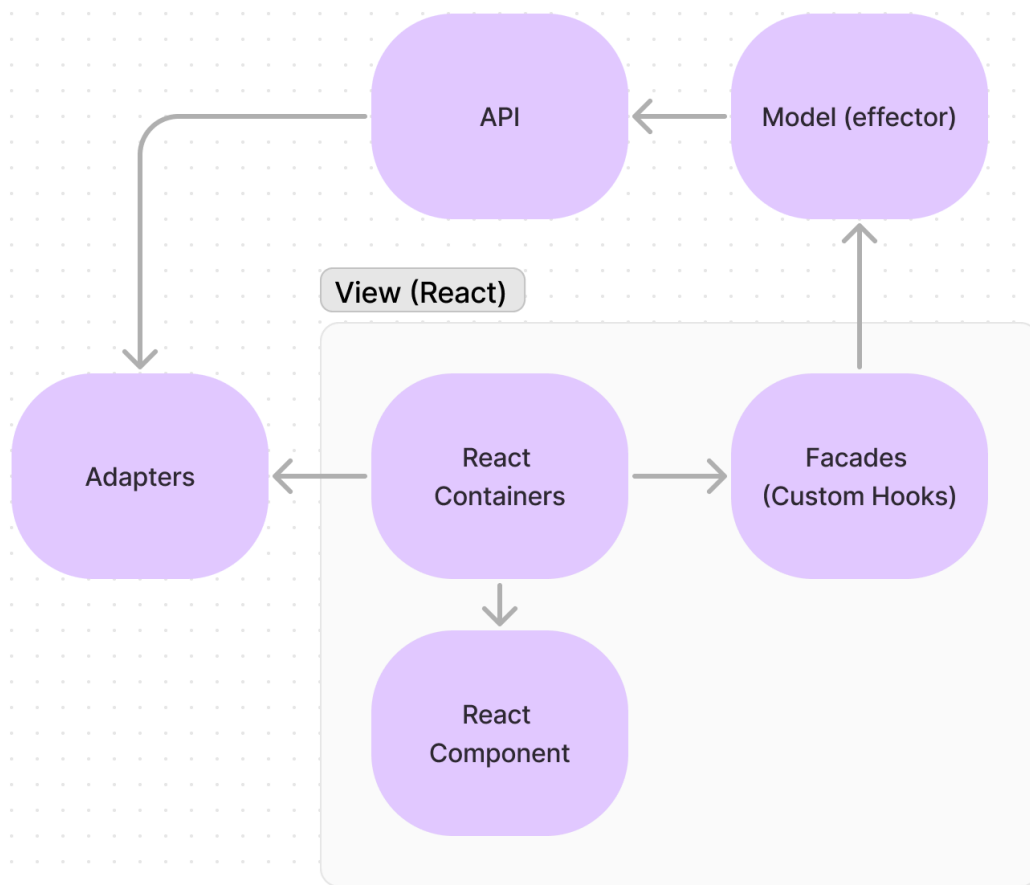
Автор последнего обновления: Pavel.Tokarenko@evraz.com | 3 апр. 2024 г.в07:57 GMT+3

В качестве сборщика проекта используется [Vite](#) . Все работы ДОЛЖНЫ начинаться с разворачивания шаблона проекта. Актуальный шаблон можно получить у Pavel.Tokarenko@evraz.com.

структура в папке `src` проекта

| | |
|-------------------|------|
| Adapters | (1) |
| API | (2) |
| App | (3) |
| Assets | (4) |
| Components | (5) |
| Containers | (6) |
| Facades | (7) |
| Mock | (8) |
| Models | (9) |
| Pages | (10) |
| Shared | (11) |
| index.css | (12) |
| main.tsx | (13) |
| routing.tsx | (14) |

Схема взаимодействия компонентов системы



1. Adapters

Адаптеры используются для преобразования данных из одного формата в другой. Для каждого ответа сервера ДОЛЖЕН использоваться свой адаптер. Для компонента, который использует массивы объектов, объект в качестве атрибута props, ДОЛЖЕН создаваться адаптер. Структура файлов адаптеров:

```

ComponentName
  fromAdapter.ts
  fromAdapter.ts
  ...
fetchAdapter.ts
fetchAdapter.ts
  ...
  
```

Адаптеры для компонентов помещаются в каталог с названием компонента.

Имя адаптера ДОЛЖНО соответствовать следующей схеме:

```
fetch[FetchName]Adapter.ts
```

```
from[DataType]To[ComponentType]Adapter.ts
```

Название функции адаптера ДОЛЖНО начинаться с `convert...` и содержать исходный тип (который преобразуется) конечный тип (в который преобразуется). Например:

```
function convertMeltInfoOriginToMeltInfo(origin: ResponseMeltInfo): MeltInfo {  
  // ...  
}
```

2. API

Содержит функции получение данных. ДОЛЖЕН содержать запросы API или подписки RabbitMQ

3. App

Основной компонент приложения, точка входа.

4. Assets

Необязательно. МОЖЕТ использоваться для хранения вспомогательных элементов приложения (изображения, шрифты). НЕЛЬЗЯ использовать для хранения контента не относящегося к интерфейсу. НЕЛЬЗЯ хранить SVG иконки, так как они реализуются, как компоненты.

5. Components

Содержит компоненты приложения в предметной области UI. Компоненты ДОЛЖНЫ соответствовать определённым [требованиям](#).

6. Containers

Содержит компоненты приложения в предметной области приложения. Контейнеры ДОЛЖНЫ соответствовать определённым [требованиям](#). Контейнер МОЖЕТ содержать дополнительную разметку. Контейнеры связывают модель приложения с компонентом.

Контейнер НЕЛЬЗЯ использовать в качестве элемента компонента. Контейнеры МОГУТ использоваться внутри других контейнеров.

7. Facades

Содержит универсальные хуки: предоставляющие взаимодействие с одним разделом модели.

```
// useUsers.ts

import { useUnit } from 'effector';
import { $users, $currentUser } from '../Models/Users/state';
import { setCurrentUser } from '../Models/Users/events';

export function useUsers() {
  const users = useUnit($users);
  const currentUser = useUnit($currentUser);

  function selectUserId(userId) {
    const user = users.find((user) => user.id === userId);
    if (user !== null) setCurrentUser(user);
  }

  // ...



  return {
    users,
    currentUser,
    selectUserId,
    deleteUserById,
    editUser,
  };
}
```

Например, Модель содержит раздел `users` для работы с пользователями: списки пользователей, события взаимодействия со списком, отправка данных и т.п. Фасад предоставляет упрощённый доступ к этим операциям.

```
const {
  users,
  currentUser,
  selectUserId,
  deleteUserById,
  editUser
} = useUsers();
```

Сам фасад включает все необходимые манипуляции и взаимодействия с моделью

8. Mock

Содержит файлы конфигурации Mock-сервера. В качестве Mock-сервера МОГУТ выступать: [json-server](#) , для него используются файлы формата `*.json`; [Swagger](#)  с форматом `*.yaml` | `*.yml`.

9. Models

Общее состояние приложения. В качестве менеджера используется [effector](#) .

Структура (Вариант 1)

```
SideEffects
  <sideEffectGroupName>.ts
<modelName>
  index.ts
  init.ts
...
init.ts
```

Структура (Вариант 2)

```
<moduleName>  
  events.ts  
  effects.ts  
  store.ts  
  index.ts
```

Пример

```
// effects.ts  
import { createEffect } from 'effector';  
import { fetchUsers, postUser, deleteUser, putUser } from '../API';  
  
export const getUsersFx = createEffect(fetchUsers);  
export const editUserFx = createEffect(putUser);  
export const createUserFx = createEffect(postUser);  
export const deleteUserFx = createEffect(deleteUser);  
  
  
// store.ts  
import { createStore } from 'effector';  
  
export const $users = createStore([]);  
export const $backupUser = createStore(null);  
export const $currentUser = createStore({});  
  
  
// events.ts  
import { createEvent } from 'effector';  
  
export const setCurrentUser = createEvent();  
export const editUser = createEvent();
```

```
// index.ts
import { sample } from 'effector';
import { getUserFx } from './effects';
import { $users, $currentUser, $backupUser } from './store';
import { setCurrentUser, editUser } from './events';
import {
  isEmpty,
  saveDataIsNotEqual,
  prepareOptimisticSave,
  prepareRestoreUserData,
  clearBackup
} from './utils';

$users.on(getUserFx.doneData, saveDataIsNotEqual);

// Установить текущего пользователя
sample({
  source: setCurrentUser,
  filter: isEmpty,
  target: $currentUser,
});

// Изменить данные о пользователе, отправить на сервер,
// сделать бэкап изменяемых данных
sample({
  source: $users,
  clock: editUser,
  fn: prepareOptimisticSave,
  target: [$users, $backupUser, editUserFx],
});

// Если сервер при сохранении ответит ошибкой,
// восстановить данные о пользователе, очистить бэкап
sample({
  source: [$users, $backupUser],
  clock: editUserFx.error,
  fn: prepareRestoreUserData,
  target: [$users, $backupUser],
});

// Если сохранение прошло успешно, очистить бэкап
sample({
  source: $backupUser,
  clock: editUserFx.done,
  fn: clearBackup,
  target: $backupUser,
})
```

10. Pages

Содержит компоненты представляющие один экран. Подчиняется тем же правилам [разработки КОМПОНЕНТОВ](#), что и обычные компоненты или контейнеры с некоторыми отступлениями.

11. Shared

Содержит общие типы для всего приложения. Общие утилиты.

12. index.css

Глобальные стили приложения

13. main.tsx

Точка входа в приложение

14. routing.tsx

Конфигурация маршрутизации