**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Фронт-энд разработка

Отчет

# Лабораторная работа №3

Выполнил:

Рыбалко Олег

K33392

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

**Задание**

Мигрировать ранее написанный сайт на фреймворк Vue.JS.

**Решение**

Так как изначально лабораторные работы были написаны с использованием React, в данной лабораторной работе по согласованию с преподавателем было необходимо вынести логику обращения к серверу в отдельные функции и хранить данные в redux.

Для начала создадим reducer для публикаций пользователей.

```
export default function postsReducer(state = initialState, action) {
  switch (action.type) {
    case 'posts/following':
      return { ...state, following: action.payload }
    case 'posts/forYou':
      return { ...state, forYou: action.payload }
    case 'posts/byAuthorId':
      return {
        ...state,
        byAuthorId: {
          ...state.byAuthorId,
          [action.payload.authorId]: action.payload.data,
        },
      }
    case 'posts/publish': {
      const posts = state.byAuthorId[action.payload.author]
      return {
        ...state,
        byAuthorId: {
          ...state.byAuthorId,
          [action.payload.author]:
            posts === undefined ? [action.payload] : [action.payload, ...posts],
        },
      }
    }
    case 'posts/delete': {
      const posts = state.byAuthorId[action.payload.authorId].filter(
        (el) => el.id !== action.payload.postId
      )
      return {
        ...state,
        byAuthorId: { ...state.byAuthorId, [action.payload.authorId]: posts },
      }
    }
    default:
      return state
  }
}
```

В данном reducer будут определены следующие события:

- получение постов от пользователей, на которых подписан человек
- получение предложений
- получение постов по id автора
- создание поста
- удаление поста

Функции для получения данных выглядят следующим образом:

```
57  export function fetchForYou(authorId: string): AnyAction {
58    return async function fetchForYouThunk(dispatch: () => void) {
59      const posts = await pb.collection('posts').getList(1, 30, {
60        sort: '-created',
61        filter: `author!="${authorId}"`,
62        expand: 'author',
63      })
64      dispatch({
65        type: 'posts/forYou',
66        payload: posts.items.map((el) => {
67          return {
68            authorUsername: el.expand!.author.username,
69            body: el.body,
70            title: el.title,
71            id: el.id,
72            likesCount: 0,
73          }
74        }),
75      })
76    }
77  }
```

Флоу данных следующий:

1. Клиентский код вызывает dispatch данной функции
2. Функция получает нужные данные и вызывает dispatch нашего события
3. reducer получает данные вместе с событием и обновляет state

Далее создадим reducer для работы с пользователями:

```
export default function usersReducer(state = initialState, action) {
  switch (action.type) {
    case 'users/getByUsername':
      return { ...state, [action.payload.username]: action.payload.data }
    case 'users/updateBio':
    case 'users/create':
    case 'users/auth':
      return { ...state, [action.payload.username]: action.payload }
    default:
      return state
  }
}
```

Пример определения функции для получения пользователя по никнейму:

```
export function getByUsername(username: string): AnyAction {
  return async function getByUsernameThunk(dispatch: () => void) {
    const record = await pb
      .collection('users')
      .getFirstListItem(`username="${username}"`)
    dispatch({
      type: 'users/getByUsername',
      payload: { username: username, data: record },
    })
  }
}
```

Также у нас на сайте есть страница с поиском. Определим также reducer и функции, необходимые для поиска данных

```
23  export default function searchReducer(
24    state = initialState,
25    action
26  ): SearchState {
27    switch (action.type) {
28      case 'search/username':
29        return { ...state, byUsername: action.payload }
30      case 'search/title':
31        return { ...state, byTitle: action.payload }
32      default:
33        return state
34    }
35  }
36
37  export function searchByUsername(username: string): AnyAction {
38    return async function searchByUsernameThunk(dispatch: () => void) {
39      const records = await pb.collection('users').getFullList({
40        filter: `username~"${username}"`,
41        sort: '-created',
42      })
43      dispatch({ type: 'search/username', payload: records })
44    }
45  }
46
47  export function searchByTitle(title: string): AnyAction {
48    return async function searchByTitleThunk(dispatch: () => void) {
49      const records = await pb.collection('posts').getFullList({
50        filter: `title~"${title}"`,
51        sort: '-created',
52        expand: 'author',
53      })
54      dispatch({ type: 'search/title', payload: records })
55    }
56  }
57
```

Более того, на нашем веб сайте есть страницы пользователей, данные на которых нужно также получить при помощи reducer

```
15  export default function usersReducer(state = initialState, action) {
16    switch (action.type) {
17      case 'users/getByUsername':
18        return { ...state, [action.payload.username]: action.payload.data }
19      case 'users/updateBio':
20      case 'users/create':
21      case 'users/auth':
22        return { ...state, [action.payload.username]: action.payload }
23      default:
24        return state
25    }
26  }
27
28  export function getByUsername(username: string): AnyAction {
29    return async function getByUsernameThunk(dispatch: () => void) {
30      const record = await pb
31        .collection('users')
32        .getFirstListItem(`username="${username}"`)
33      dispatch({
34        type: 'users/getByUsername',
35        payload: { username: username, data: record },
36      })
37    }
38  }
```

**Вывод**

В данной лабораторной работе мы научились работать с reducer в redux и использовать данный инструмент для асинхронного получения и отображения данных