# Код-ревью (из 74 ревью) от 01.12.2024 07:49

## Файл: antd-table-develop/src/app.tsx

Строка 7

- **Комментарий:**

  Используйте функциональные компоненты с помощью стрелочных функций без присваивания переменной. Например: const App = (): JSX.Element => (...);

- **Текущий код:**

```
import { PageRoutes } from './routes';

const App = (): JSX.Element => (
```

---

Строка 9

- **Комментарий:**

  Лишние пустые строки могут затруднить чтение кода. Уберите лишние пустые строки.

- **Текущий код:**

```
const App = (): JSX.Element => (
  <BrowserRouter>
    <PageRoutes />
```

---

Строка 12

- **Комментарий:**

  Лишние пустые строки могут затруднить чтение кода. Уберите лишние пустые строки.

- **Текущий код:**

```
<BrowserRouter>
    <PageRoutes />
  </BrowserRouter>
);
```

---

Строка 15

- **Комментарий:**

Лишние пустые строки могут затруднить чтение кода. Уберите лишние пустые строки.

- **Текущий код:**

```
</BrowserRouter>
);
```

---

Строка 17

- **Комментарий:**

Лишние пустые строки могут затруднить чтение кода. Уберите лишние пустые строки.

- **Текущий код:**

```
);

export default App;
```

---

## Файл: antd-table-develop/src/routes.tsx

Строка 9

- **Комментарий:**

Используйте более описательное имя для функции, например 'renderPageRoutes'. Это улучшит читаемость кода.

- **Текущий код:**

```
import { MainPage } from './pages/main-page';
import { UserPage } from './pages/user-page';

export const PageRoutes = (): JSX.Element => {
  return (
```

---

Строка 13

- **Комментарий:**

Используйте строгое равенство (===) для сравнения значений, если это возможно. Это предотвратит ошибки, связанные с приведением типов.

- **Текущий код:**

```
export const PageRoutes = (): JSX.Element => {
  return (
    <Routes>
      <Route path={pageRoutes.MAIN} element={<MainPage />} />
```

---

Строка 15

- **Комментарий:**

Используйте строгое равенство (===) для сравнения значений, если это возможно. Это предотвратит ошибки, связанные с приведением типов.

- **Текущий код:**

```
<Routes>
      <Route path={pageRoutes.MAIN} element={<MainPage />} />
      <Route path={pageRoutes.USER_FORM_ID} element={<UserPage />} />
    </Routes>
```

---

## Файл: antd-table-develop/src/types/user.ts

Строка 2

- **Комментарий:**

  Необходимо добавить запятую после имени типа для лучшей читаемости.

- **Текущий код:**

```
export type UserType = {
  id: number;
```

---

Строка 5

- **Комментарий:**

  Необходимо добавить запятую после поля 'id' для лучшей читаемости.

- **Текущий код:**

```
export type UserType = {
  id: number;
  name: string;
  allow_edit: boolean;
```

---

Строка 8

- **Комментарий:**

  Необходимо добавить запятую после поля 'name' для лучшей читаемости.

- **Текущий код:**

```
name: string;
  allow_edit: boolean;
};
```

---

Строка 11

- **Комментарий:**

  Использование snake_case для имени поля 'allow_edit' не соответствует общепринятым практикам в TypeScript. Рекомендуется использовать camelCase: 'allowEdit'.Проблема: Отсутствует импорт типа UserType. Решение: Добавьте импорт UserType из соответствующего модуля.

- **Текущий код:**

```
};

export type User = Omit<UserType, 'allow_edit'> & {
  login: string;
```

---

Строка 14

- **Комментарий:**

  Проблема: Отсутствует точка с запятой в конце строки. Решение: Добавьте точку с запятой после строки.

- **Текущий код:**

```
export type User = Omit<UserType, 'allow_edit'> & {
  login: string;
  password: string;
  type_id: number;
```

---

Строка 17

- **Комментарий:**

  Проблема: Отсутствует точка с запятой в конце строки. Решение: Добавьте точку с запятой после строки.

- **Текущий код:**

```
password: string;
  type_id: number;
  type?: string;
  last_visit_date: string;
```

Строка 20

- **Комментарий:**

  Проблема: Отсутствует точка с запятой в конце строки. Решение: Добавьте точку с запятой после строки.

- **Текущий код:**

```
type?: string;
  last_visit_date: string;
};
```

---

Строка 23

- **Комментарий:**

  Проблема: Отсутствует точка с запятой в конце строки. Решение: Добавьте точку с запятой после строки.Использование Omit для исключения полей 'last_visit_date' и 'type' может быть избыточным. Рассмотрите возможность создания нового типа с нужными полями напрямую.

- **Текущий код:**

```
};

export type UpdateUser = Omit<User, 'last_visit_date' | 'type'>;
```

---

Строка 26

- **Комментарий:**

  Проблема: Отсутствует точка с запятой в конце строки. Решение: Добавьте точку с запятой после строки.Использование Omit для создания CreateUser из UpdateUser может быть не самой лучшей практикой. Предпочтительнее явно определить тип CreateUser, чтобы избежать потенциальных ошибок и неясностей. Например, можно создать CreateUser с явными полями, которые необходимы для создания пользователя.

- **Текущий код:**

```
export type UpdateUser = Omit<User, 'last_visit_date' | 'type'>;

export type CreateUser = Omit<UpdateUser, 'id'>;
```

---

Строка 29

- **Комментарий:**

Проблема: Отсутствует точка с запятой в конце строки. Решение: Добавьте точку с запятой после строки.

- **Текущий код:**

```
export type CreateUser = Omit<UpdateUser, 'id'>;

export interface UsersFilters {
```

---

Строка 31

- **Комментарий:**

Проблема: Отсутствует точка с запятой в конце строки. Решение: Добавьте точку с запятой после строки.

- **Текущий код:**

```
export interface UsersFilters {
  name?: string;
  type?: string;
```

---

Строка 34

- **Комментарий:**

Необходимо уточнить тип для dateRange. Предпочтительно использовать более конкретный тип, например, [string, string] или DateRange.

- **Текущий код:**

```
name?: string;
  type?: string;
  dateRange: string[];
}
```

---

## Файл: antd-table-develop/src/components/user-form/user-form.tsx

Строка 15

- **Комментарий:**

Использование useParams без проверки на наличие параметра может привести к ошибке. Рекомендуется добавить проверку на undefined.

- **Текущий код:**

```
export function UserForm(): JSX.Element {
  const { id } = useParams();
```

---

Строка 17

- **Комментарий:**

Преобразование строки в число без проверки на NaN может привести к ошибкам. Рекомендуется использовать Number.isNaN для проверки.

- **Текущий код:**

```
export function UserForm(): JSX.Element {
  const { id } = useParams();
  const userId = Number(id);
```

---

Строка 21

- **Комментарий:**

  Использование useUnit для получения нескольких значений из стора может быть неэффективным. Рассмотрите возможность использования отдельных хуков для каждого значения.

- **Текущий код:**

```
const userId = Number(id);


  const [user, loadingUser, userTypes, clearUserForm] = useUnit([
    $user,
```

---

Строка 30

- **Комментарий:**

  useEffect вызывается при каждом рендере, что может быть избыточно. Рекомендуется добавить пустой массив зависимостей, если это возможно.

- **Текущий код:**

```
]);
  const [form] = Form.useForm();

  useEffect(() => {
    getUserFx(userId);
```

---

Строка 38

- **Комментарий:**

  Комментарий должен быть написан на английском языке для лучшей читаемости кода. Рекомендуется переписать комментарий на английский.

- **Текущий код:**

```
};
  }, [userId, clearUserForm]);

  const userTypesOptions = useStoreMap({
```

Строка 51

- **Комментарий:**

  Использование async/await внутри onFinish может привести к блокировке UI. Рекомендуется использовать Promise.then для обработки результата.

- **Текущий код:**

```
    })),
      });

      const onFinish: FormProps<CreateUser>['onFinish'] = async (values) => {
```

Строка 53

- **Комментарий:**

  Поиск типа пользователя в массиве userTypes может быть неэффективным. Рекомендуется использовать объект для быстрого доступа по id.

- **Текущий код:**

```
    const onFinish: FormProps<CreateUser>['onFinish'] = async (values) => {
        const selectedUserType = userTypes.find((userType) => userType.id === values.type_id);
        const newValues = { ...values, type: selectedUserType?.name };
```

Строка 64

- **Комментарий:**

  Fragment тут не имеет смысла. Следует вместо него использовать return промежуточного состояния через условие usersLoading

- **Текущий код:**

```
    };

      return (
```

```
    <>
```

---

Строка 65

- **Комментарий:**

  Комментарий должен быть написан на английском языке для лучшей читаемости кода. Рекомендуется переписать комментарий на английский.

- **Текущий код:**

```
return (
    <>
```

---

Строка 67

- **Комментарий:**

  Использование фрагмента <> без необходимости может затруднить читаемость кода. Рекомендуется использовать React.Fragment с ключом.

- **Текущий код:**

```
<>
      {!loadingUser && (
        <Form
```

---

Строка 79

- **Комментарий:**

  Использование || undefined для initialValues не является необходимым. Рекомендуется передавать user напрямую.

- **Текущий код:**

```
border: '1px solid #52618d',
          }}
          onFinish={onFinish}
```

```
            initialValues={user || undefined}
            layout="vertical"
```

---

Строка 107

- **Комментарий:**

  Повторяющийся код для Form.Item можно вынести в отдельный компонент для улучшения читаемости и поддержки кода.

- **Текущий код:**

```
    >
              <Input allowClear placeholder="Введите пароль пользователя" />
            </Form.Item>
            <Form.Item
              label="Тип пользователя:"
```

---

## Файл: antd-table-develop/src/components/filters-form/filters-form.tsx

Строка 15

- **Комментарий:**

  Использование useUnit для получения состояния загрузки может быть избыточным. Рассмотрите использование прямого доступа к состоянию через useStore.

- **Текущий код:**

```
    export function FiltersForm(): JSX.Element {
      const [filtersLoading] = useUnit([getFilteredUsersFx.pending]);
      const [form] = Form.useForm();
```

---

Строка 20

- **Комментарий:**

useStoreMap используется без ключей, что может привести к ненужным перерисовкам. Убедитесь, что это необходимое поведение.

- **Текущий код:**

```
const [form] = Form.useForm();


  const userTypes = useStoreMap({
    store: $userTypes,
```

---

Строка 27

- **Комментарий:**

Использование userType.name в качестве значения и метки может привести к ошибкам, если имена не уникальны. Рассмотрите использование userType.id или другого уникального идентификатора.

- **Текущий код:**

```
fn: (userTypes) =>
      userTypes.map((userType) => ({
        label: userType.name,
        value: userType.name,
      })),
```

---

Строка 37

- **Комментарий:**

Использование dayjs() для текущей даты может привести к непредсказуемым результатам из-за изменения времени. Рассмотрите использование фиксированной даты или функции, которая возвращает текущую дату один раз.

- **Текущий код:**

```
form={form}
      style={{ padding: 20 }}
      onFinish={getFilteredUsersFx}
      initialValues={{ dateRange: [dayjs(FILTER_START_DATE), dayjs()] }}
      layout="vertical"
```

---

Строка 50

- **Комментарий:**

  Фильтрация опций на стороне клиента может быть неэффективной для больших списков. Рассмотрите возможность фильтрации на сервере.

- **Текущий код:**

```
    showSearch
              placeholder="Выберите тип пользователя"
              optionFilterProp="children"
              filterOption={(input, option) =>
                (option?.label ?? '').toLowerCase().includes(input.toLowerCase())
```

---

Строка 58

- **Комментарий:**

  Использование Flex для создания метки формы может быть избыточным. Рассмотрите использование стандартных элементов формы или более простого способа разметки.

- **Текущий код:**

```
    />
          </Form.Item>
          <Form.Item
            label={
              <Flex gap={96}>
```

---

Строка 68

- **Комментарий:**

  Установка allowClear={false} для RangePicker может быть неинтуитивной для пользователя. Рассмотрите возможность предоставления опции очистки.

- **Текущий код:**

```
name={'dateRange'}
      >
        <RangePicker
          allowClear={false}
```

---

Строка 70

- **Комментарий:**

  Обработка события onKeyDown для RangePicker может быть избыточной. Рассмотрите возможность использования стандартных механизмов формы для отправки данных.

- **Текущий код:**

```
<RangePicker
          allowClear={false}
          onKeyDown={(e) => {
            if (e.key === 'Enter') {
```

---

Строка 81

- **Комментарий:**

  Использование onClick для сброса полей формы может быть неэффективным. Рассмотрите возможность использования стандартных методов формы для сброса значений.

- **Текущий код:**

```
<Button type="primary" block loading={filtersLoading} htmlType="submit">
        Поиск
      </Button>
      <Button type="link" onClick={() => form.resetFields()}>
        Сбросить фильтры
```

---

## Файл: antd-table-develop/src/components/users-table/users-table.tsx

Строка 15

- **Комментарий:**

  Функция UsersTable должна быть объявлена как React компонент с использованием React.FC или React.FunctionComponent для лучшей типизации.

- **Текущий код:**

```
import { User } from '@/types/user';


export function UsersTable(): JSX.Element {
  const navigate = useNavigate();
```

---

Строка 19

- **Комментарий:**

  Использование useUnit для получения нескольких значений может быть заменено на отдельные хуки для каждого значения для улучшения читаемости и отладки.

- **Текущий код:**

```
export function UsersTable(): JSX.Element {
  const navigate = useNavigate();

  const [users, selectedUserIds, usersPending, setSelectedUsers] = useUnit([
    $users,
```

---

Строка 27

- **Комментарий:**

  useEffect без зависимостей вызывает getUsersFx при каждом рендере. Добавь пустой массив зависимостей, чтобы вызывать эффект только один раз.

- **Текущий код:**

```
selectUserIds,
  ]);

  useEffect(() => {
```

```
        getUsersFx();
```

---

Строка 33

- **Комментарий:**

  Переменная columns может быть вынесена за пределы компонента для улучшения читаемости и переиспользования.

- **Текущий код:**

```
    }, []);


      const columns = [
        ...userTableHeaders,
```

---

Строка 37

- **Комментарий:**

  Пустое значение dataIndex не рекомендуется. Укажи dataIndex, соответствующий полю в объекте User.

- **Текущий код:**

```
    const columns = [
        ...userTableHeaders,
        {
          title: '',
          dataIndex: '',
```

---

Строка 44

- **Комментарий:**

  Использование img внутри Button может привести к проблемам с доступностью. Рассмотри использование SVG-иконок или других методов для вставки иконок.

- **Текущий код:**

```
render: (record: User) => (
        <Button
          icon={
            <img width={24} height={24} src={editIcon} title="Редактировать" alt="Редактирование"
          }
```

---

Строка 54

- **Комментарий:**

rowSelection может быть вынесена за пределы компонента для улучшения читаемости и переиспользования.

- **Текущий код:**

```
},
  ];

  const rowSelection = {
    onChange: (_: Key[], selectedRows: User[]) => {
```

---

Строка 63

- **Комментарий:**

Добавь key prop для Table, чтобы избежать предупреждений React о повторяющихся ключах.

- **Текущий код:**

```
};

  return (
    <Table
      bordered
```

---

## Файл: antd-table-develop/src/components/buttons/form-submit-button/form-submit-button.tsx

Строка 7

- **Комментарий:**

  Используй более описательные имена для интерфейсов, например IFormSubmitButtonProps.

- **Текущий код:**

```
import { type FC, type PropsWithChildren, useEffect, useState } from 'react';


interface FormSubmitButtonProps {
```

---

Строка 9

- **Комментарий:**

  Используй более описательные имена для пропсов, например formInstance вместо form.

- **Текущий код:**

```
interface FormSubmitButtonProps {
  form: FormInstance;
}
```

---

Строка 13

- **Комментарий:**

  Используйте более описательные имена для компонентов. Например, FormSubmitButton можно переименовать в SubmitButton.

- **Текущий код:**

```
form: FormInstance;
}
```

```
export const FormSubmitButton: FC<PropsWithChildren<FormSubmitButtonProps>> = ({
  form,
```

---

Строка 16

- **Комментарий:**

  Лишние пустые строки затрудняют чтение кода. Уберите их.

- **Текущий код:**

```
export const FormSubmitButton: FC<PropsWithChildren<FormSubmitButtonProps>> = ({
  form,
  children,
}) => {
```

---

Строка 22

- **Комментарий:**

  Необходимо инициализировать состояние submittable с учетом начального состояния формы. Например, можно использовать form.getFieldsValue() для определения начального значения.

- **Текущий код:**

```
const [submittable, setSubmittable] = useState<boolean>(false);

  const values = Form.useWatch([], form);

  useEffect(() => {
```

---

Строка 25

- **Комментарий:**

  Form.useWatch([], form) может быть неэффективным, так как следит за всеми полями формы. Лучше следить только за теми полями, которые влияют на валидацию.

- **Текущий код:**

```
useEffect(() => {
    form
      .validateFields({ validateOnly: true })
```

---

Строка 28

- **Комментарий:**

useEffect должен иметь пустой массив зависимостей, чтобы избежать лишних рендеров. Используйте useMemo для memoization значений, которые не меняются часто.

- **Текущий код:**

```
form
      .validateFields({ validateOnly: true })
      .then(() => setSubmittable(true))
```

---

Строка 30

- **Комментарий:**

Метод form.validateFields() может быть вызван с конкретными полями, чтобы уменьшить количество операций валидации.

- **Текущий код:**

```
.validateFields({ validateOnly: true })
      .then(() => setSubmittable(true))
      .catch(() => setSubmittable(false));
  }, [form, values]);
```

---

Строка 38

- **Комментарий:**

Используйте более описательные пропсы для Button, например, buttonText вместо children, чтобы явно указать, что это текст кнопки.

- **Текущий код:**

```
<Button type="primary" htmlType="submit" disabled={!submittable}>
      {children}
    </Button>
  );
};
```

---

## Файл: antd-table-develop/src/components/buttons/delete-users-button/delete-users-button.tsx

Строка 11

- **Комментарий:**

Неправильное использование useUnit. Ожидается массив эффектов и стор, а не массив сторов. Используйте useStore и useEvent для разделения.

- **Текущий код:**

```
export function DeleteUsersButton(): JSX.Element {

  const [selectedUsers, deleteUsers] = useUnit([$selectedUserIds, deleteUsersByIds]);
```

---

Строка 18

- **Комментарий:**

Необходимо передать selectedUsers в deleteUsers, чтобы функция знала, каких пользователей удалять. Используйте deleteUsers(selectedUsers).

- **Текущий код:**

```
<Popconfirm
      title="Удаление"
      description="Вы уверены, что хотите удалить выбранных пользователей?"
      onConfirm={() => deleteUsers()}
      okText="Да"
```

Строка 25

- **Комментарий:**

  Неправильное использование disabled. selectedUsers – это массив, а не булево значение. Используйте disabled={selectedUsers.length === 0}.

- **Текущий код:**

```
okButtonProps={{ style: { width: 70 } }}
      cancelButtonProps={{ style: { width: 70 } }}
    >
      <Button danger disabled={selectedUsers.length === 0}>
        Удалить выбранных
```

---

## Файл: antd-table-develop/src/pages/user-page/user-page.tsx

Строка 11

- **Комментарий:**

  Функция UserPage должна быть объявлена как React компонент, используя стрелочную функцию или функциональный компонент с хуками.

- **Текущий код:**

```
const { Header, Content } = Layout;

export function UserPage(): JSX.Element {
```

---

Строка 13

- **Комментарий:**

  Хук useNavigate должен быть использован внутри функционального компонента или другого хука.

- **Текущий код:**

```
export function UserPage(): JSX.Element {
  const navigate = useNavigate();
  return (
```

---

Строка 17

- **Комментарий:**

Использование инлайн-стилей в JSX не рекомендуется. Рассмотрите использование CSS-модулей или styled-components для стилизации.

- **Текущий код:**

```
const navigate = useNavigate();
  return (
    <Layout>
      <Header style={{ backgroundColor: '#52618d' }}>
```

---

Строка 19

- **Комментарий:**

Использование инлайн-функций в onClick не рекомендуется. Рассмотрите вынесение обработчика в отдельную функцию.

- **Текущий код:**

```
<Layout>
      <Header style={{ backgroundColor: '#52618d' }}>
        <Button onClick={() => navigate(pageRoutes.MAIN)}>На главную</Button>
      </Header>
```

---

Строка 22

- **Комментарий:**

Использование инлайн-стилей в JSX не рекомендуется. Рассмотрите использование CSS-модулей или styled-components для стилизации.

- **Текущий код:**

```
<Button onClick={() => navigate(pageRoutes.MAIN)}>На главную</Button>
      </Header>
      <Content style={{ marginInline: 50 }}>
        <UserForm />
```

---

## Файл: antd-table-develop/src/pages/main-page/main-page.tsx

Строка 13

- **Комментарий:**

Используйте React.FC для типизации компонентов, чтобы явно указать, что это функциональный компонент.

- **Текущий код:**

```
const { Header, Sider, Content } = Layout;

export function MainPage(): JSX.Element {
  const navigate = useNavigate();
```

---

Строка 19

- **Комментарий:**

Избегайте жестко закодированных значений для стилей. Используйте переменные или темы для более гибкого управления стилями.

- **Текущий код:**

```
return (
    <Layout>
      <Header style={{ backgroundColor: '#52618d', position: 'sticky', top: 0, zIndex: 1 }}>
        <Flex justify="end" gap={14} align="center" style={{ height: '100%' }}>
```

Строка 22

- **Комментарий:**

  Используйте useCallback для мемоизации обработчиков событий, чтобы избежать ненужных перерисовок.

- **Текущий код:**

```
<Header style={{ backgroundColor: '#52618d', position: 'sticky', top: 0, zIndex: 1 }}>
        <Flex justify="end" gap={14} align="center" style={{ height: '100%' }}>
          <Button onClick={() => navigate(pageRoutes.NEW_USER_FORM)}>Добавить пользователя</Button
          <DeleteUsersButton />
```

---

Строка 27

- **Комментарий:**

  Избегайте вложенности Layout внутри Layout. Используйте более подходящие компоненты или разделите логику на более мелкие компоненты.

- **Текущий код:**

```
<DeleteUsersButton />
        </Flex>
      </Header>
      <Layout>
        <Content style={{ marginInline: 50 }}>
```

---

Строка 32

- **Комментарий:**

  Используйте переменные для значений ширины и других стилей, чтобы упростить поддержку и изменение кода.

- **Текущий код:**

```
<Content style={{ marginInline: 50 }}>
          <UsersTable />
        </Content>
```

```
            <Sider
              width={320}
```

---

Строка 37

- **Комментарий:**

  Используйте camelCase для свойств стилей, таких как insetInlineStart, чтобы соответствовать общепринятым соглашениям.

- **Текущий код:**

```
    width={320}
              collapsible
              collapsedWidth={0}
              zeroWidthTriggerStyle={{
                insetInlineStart: '-40px',
```

---

Строка 45

- **Комментарий:**

  Используйте переменные для высоты и других размеров, чтобы упростить поддержку и изменение кода.

- **Текущий код:**

```
    color: '#1890ff',
              }}
              style={{
                height: 'calc(100vh - 64px)',
                position: 'sticky',
```

---

Строка 50

- **Комментарий:**

  Свойство display: 'block' является избыточным, так как элемент Sider по умолчанию отображается как блочный.

- **Текущий код:**

```
position: 'sticky',
            top: 64,
            right: 0,
            display: 'block',
            backgroundColor: '#f7f9ff',
```